Smart Weather Station

Instrumentation

Prof. Nasiri

January 2, 2023

Shiva Shakeri - 810197691
Samira Hajizadeh - 810198378

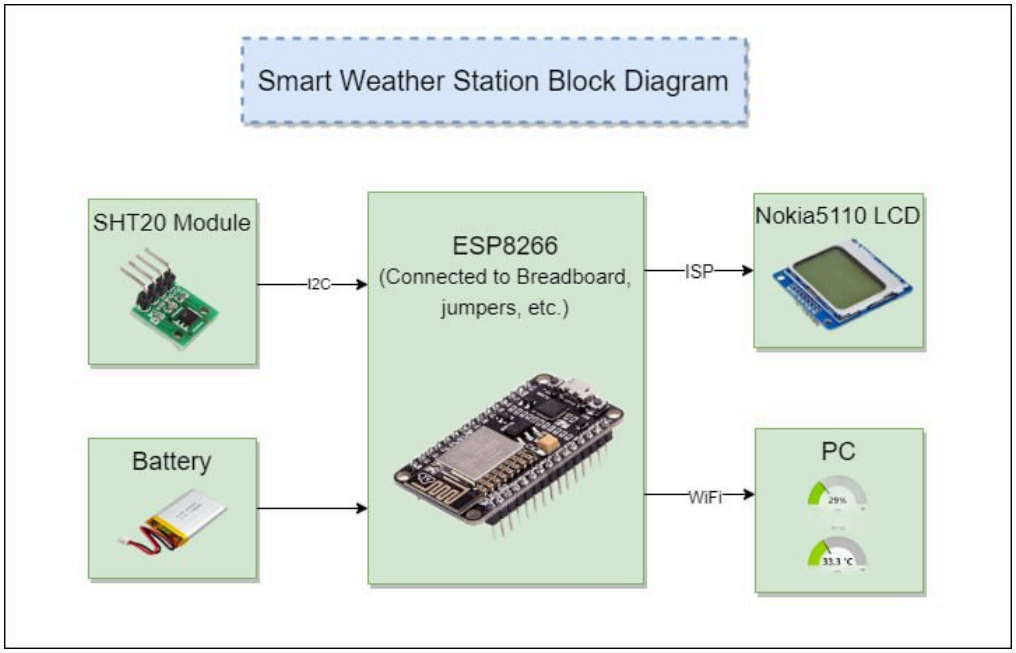
Contents

Introduction

Due to the emergence of cheap chips and the presence of wireless networks everywhere, everything can now be connected to the Internet and become part of the Internet of Things (IoT). As a result, intelligent systems can be designed without human intervention by adding digital intelligence to them.

The main purpose in the project will be to become familiar with these systems by designing and constructing an IoT system called a Smart Weather Station. This project will be done using ESP32 hardware. We aim to become familiar with and set up the I2C and SPI protocols, interrupt, WiFi, MQTT, and solutions for managing energy consumption in the system and increasing efficiency in this project.

In this project, we'll implement an IoT system that is normally in Deep Sleep mode. It'll wake up every one minute to get the sensor data and publish it on out server using MQTT and Node-Red. It will be awakened also by a pushbutton which restarts the ESP8266 development board.



1. Circuit

1.1. Components

- **ESP8266 Development Board**

To construct and assemble the circuit and avoid the design of the printed circuit, it is necessary to prepare an ESP8266 development board.

The ESP8266 is a low-cost Wi-Fi microchip, with built-in TCP/IP networking software, and microcontroller capability, produced by Espressif. Figure.1 shows the pinouts of ESP8266.

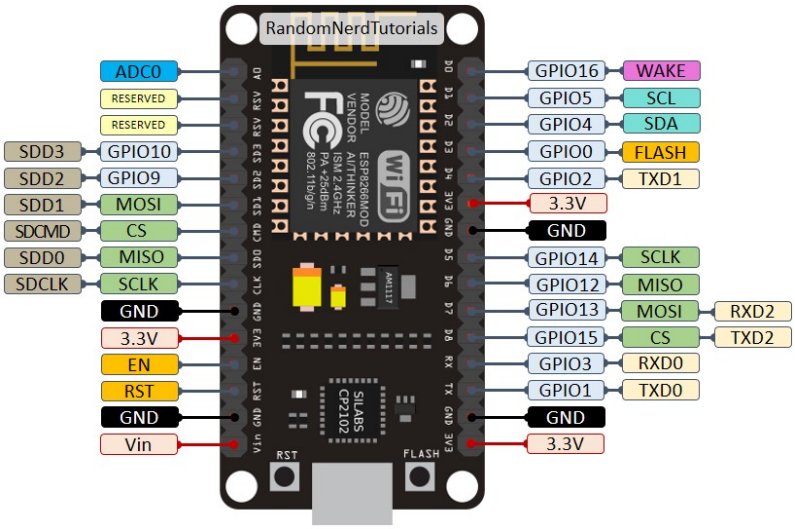


Fig. 1 ESP8266 Pinout Reference

- **Temperature and Humidity Sensor**

This project requires an I2C-connected temperature and humidity sensor. We used the SHT20 sensor.(Fig. 2)

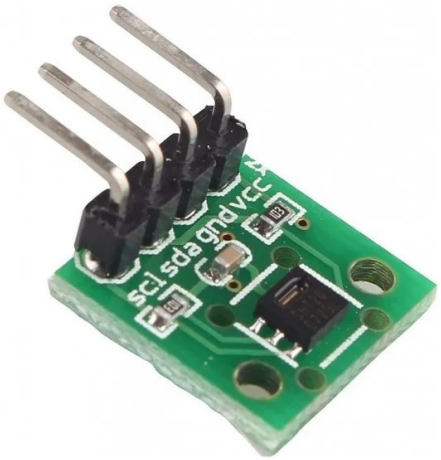


Fig. 2 SHT20 Sensor

- **Graphic Display Module**

The graphic display is launched using the SPI protocol. In this project we used the NOKIA-5110 Graphic LCD Display.(Fig. 4)

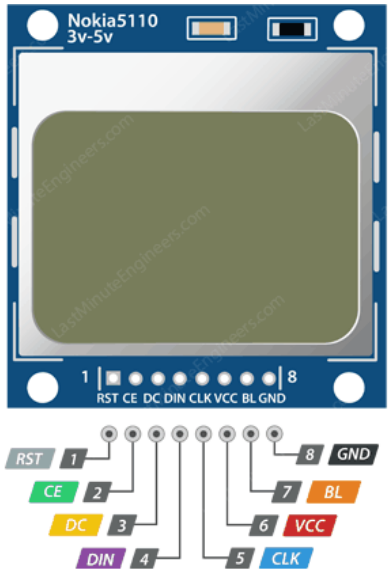


Fig. 3 Nokia 5110 LCD Display Pinout

- **Breadboard and Wires**

Preparing this part is necessary for connecting different parts cleanly and neatly; It is also possible to build the system without using the board.

- **Battery**

The device is mobile and does not require an electrical connection to operate. A power bank is the easiest way to accomplish this. However, it is also possible to implement using a battery with a charger board if it is standard. In this project, we used a lithium battery.

Fig. 4 Battery ION

- **Pushbutton**

In order to call the MCU to display information, a push buttons are needed.

1.2. Wiring

1.2.1. Temperature and Humidity Sensor

In this project, we'll use the SHT20 sensor. The SHT20 temperature and humidity module is high-precious, full calibrated, with very low power consumption and high response speed. This sensor is a combination of a capacitive humidity sensor and a silicon band gap temperature sensor. The key features are shown is Table. 1.

Table 1. SHT20 Key Features

Temperature Range	-40 to +125 °C
Temperature Accuracy	0.3 °C
Humidity Range	0 to 100%
Humidity Accuracy	3% RH
Communication Protocol	I2C

First, we connect the pins. Fig. 7 shows the connections.

Black = GND, Red = 3V3, Green = SDA(to D5), and SCL(to D6).

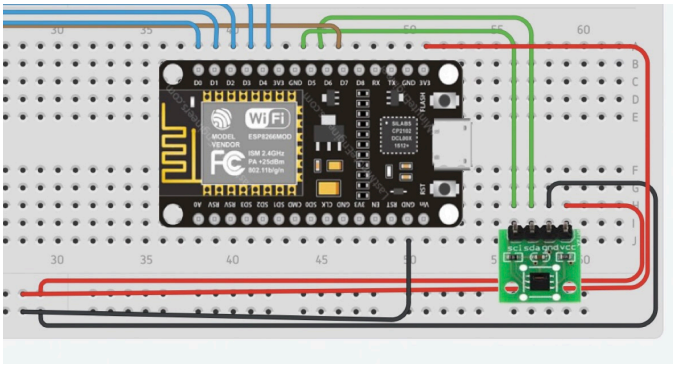


Fig. 5 SHT20 Sensor Wiring

1.2.2. Graphic Display

We used the LCD display version NOKIA 5101. The picture (Fig. 4) shows the pinout of the LCD screen.

Table. 2 shows how we connected the LCD display with the ESP8266 with a description of the LCD pin. Fig. 6 shows the circuit implementation.

Table 2. LCD and EPS8266 Pin Connection

NOKIA 5110 LCD Pin Label	ESP8266 NodeMCU Connection
RST	D0
CE	D1
DC	D2
DIN	D3
CLK	D4
VCC	3V3
BL	3.3V
GND	GND

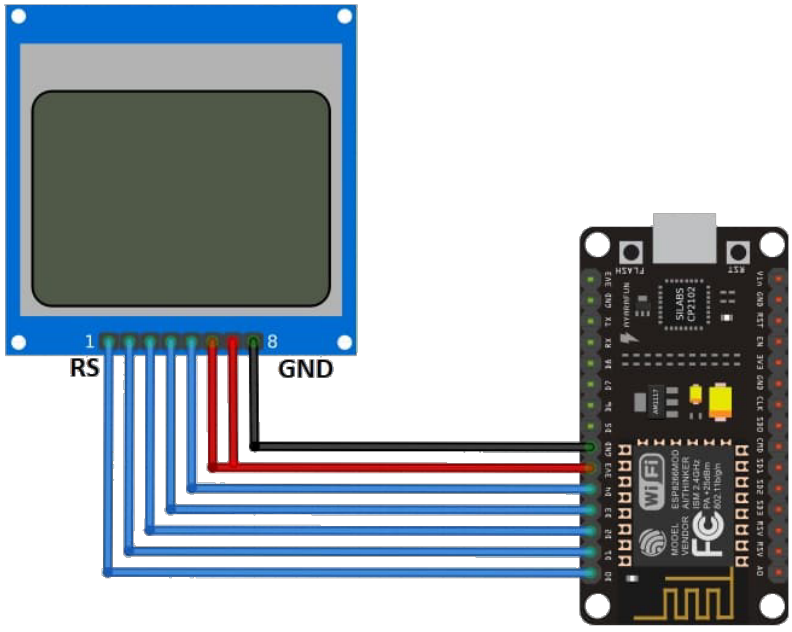


Fig.6 Connections for ESP8266 Nokia 5110 LCD interface

2.2. Assembling

- **Circuit Diagram**

In Fig. 7, the circuit diagram which we implemented is shown. The real implemented circuit is shown is Fig. 8.

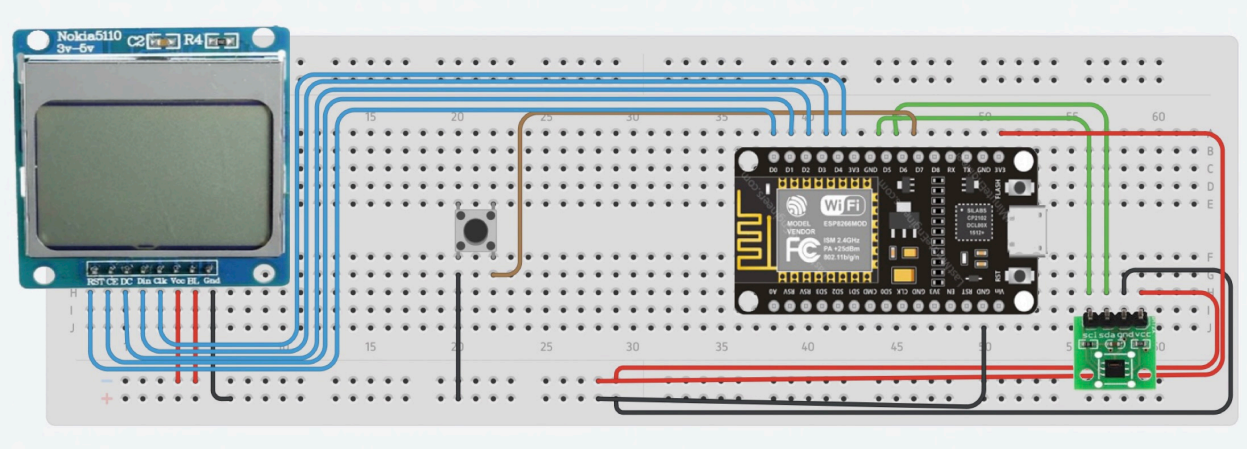


Fig. 7 Circuit Diagram

- **Implemented Circuit**

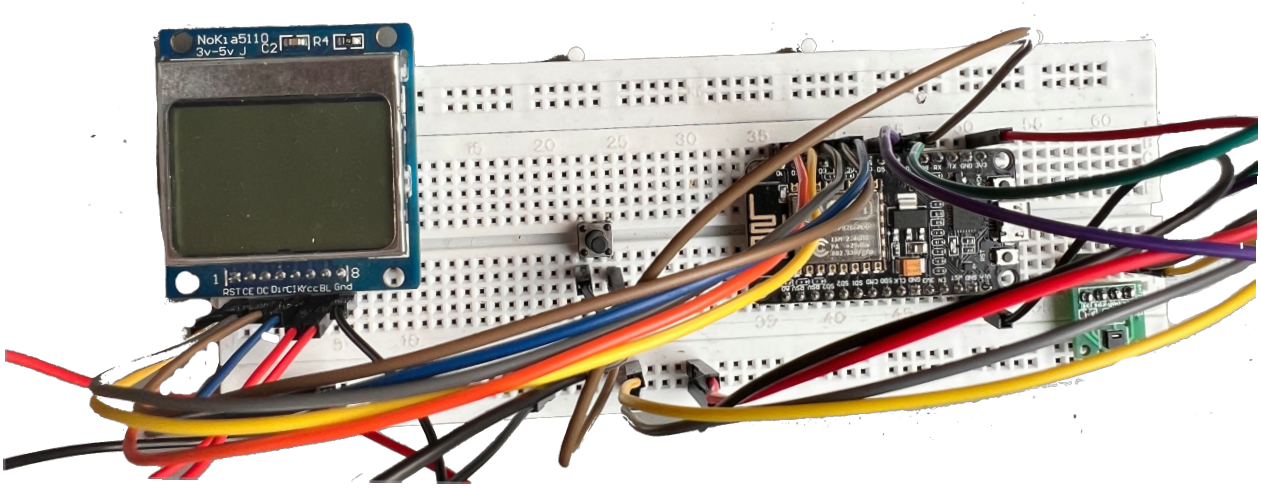


Fig. 8 Implemented Circuit

2. Setup

2.1. SHT20 Sensor Setup

One of the key goals of this project is to launch the I2C communication protocol. To accomplish this, we need a library and the correct transfer of information between the processor and the sensor. Moreover, a function must be defined to read data from the sensor. In order to work with the sensor using Arduino, we use a library named "DFRobot_SHT20.h." There exists a class named "DFRobot_SHT20" in this library and its methods are shown as Table. 3:

Table 3. DFRobot_SHT20 Methods and Descriptions

`void initSHT20(void);`	`/**`
	`* @fn initSHT20`
	`* @brief Init function`
	`* @return None`
	`*/`

`float readHumidity(void);`	`/**`
	`* @fn readHumidity`
	`* @brief Read the measured data of air humidity`
	`* @return Return the measured air humidity data of float type, unit: %`
	`*/`

`float readTemperature(void);`	`/**`
	`* @fn readTemperature`
	`* @brief Read the measured temp data`
	`* @return Return the measured temp data of float type, unit: C`
	`*/`

`void checkSHT20(void);`	`/**`
	`* @fn checkSHT20`
	`* @brief Check the current status information of SHT20`
	`* @n Status information: End of battery, Heater enabled, Disable OTP reload`
	`* @n Check result: yes, no`
	`* @return None`
	`*/`

In order to make the sensor read the temperature and humidity data, first, we define an instance named "sht20" of class "DFRobot_SHT20." Then, we use the methods `initSHT20();` and `checkSHT20();` in the setup function:

```
void setup(void)
{
  pinMode(button.PIN, INPUT_PULLUP);
  attachInterrupt(button.PIN, isr, FALLING);

  // initialize the display
  display.begin();
  Wire.begin(D6, D5);  // set I2C pins [SDA = D6, SCL = D5], default clock is 100kHz
  display.setContrast(contrastValue);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(BLACK, WHITE);
  display.setCursor(3, 0);
  display.print("SHT20 SENSOR");
  display.display();
  display.setCursor(7, 10);
  display.print("TEMPERATURE:");
  display.setCursor(16, 31);
  display.print("HUMIDITY:");
  display.display();

  sht20.initSHT20();
  delay(100);
  sht20.checkSHT20();
  delay(5000);
}
```

Moreover, in the main loop, we use the `readHumidity();` and `readTemprature();` methods in the main loop:

```
void loop()
{
  if (button.pressed) {
    // get temperature and humidity from library
    float humd = sht20.readHumidity(); //get humidity
    float temp = sht20.readTemperature(); // get temperature
    display.setCursor(15, 20);
    if(temp < 0)
      display.printf("-%02u.%02u C", (int)abs(temp) % 100, (int)(abs(temp) * 100) %
100 );
    else
      display.printf(" %02u.%02u C", (int)temp % 100, (int)(temp * 100) % 100 );
    display.drawRect(53, 20, 3, 3, BLACK); // print degree symbol ( ° )
     display.setCursor(9, 41);
    display.printf("%04u.%02u", (int)(humd/100), (int)((uint32_t)humd % 100) );
    display.display();
    delay(2000);
  }
  static uint32_t lastMillis = 0;
  if (millis() - lastMillis > 10000) {
    lastMillis = millis();
    detachInterrupt(button.PIN);
  }
}
```

2.2. Nokia 5110 Graphic Display Setup

The Nokia 5110 LCD uses an Serial Peripheral Interface(SPI) like serial interface to communicate with a microcontroller. So, we have to use the SPI pins of the ESP8266 Microcontroller:

- SPI Interface uses four wires for communication. Hence it is also known as a four-wire serial communication protocol.
- SPI is a full-duplex master-slave communication protocol. This means that only a single master and a single slave can communicate on the interface bus at the same time.
- SPI enabled devices to work in two basic modes of SPI operation i.e. SPI Master Mode and SPI Slave Mode.
- The master Device is responsible for the initiation of communication. The master Device generates a Serial Clock for synchronous data transfer. Master Device can handle multiple slave devices on the bus by selecting them one by one.

NodeMCU based ESP8266 has Hardware SPI with four pins available for SPI communication. With this SPI interface, we can connect any SPI enabled device with NodeMCU and make communication possible with it.

ESP8266 has SPI pins (SD1, CMD, SD0, CLK) which are exclusively used for Quad-SPI communication with flash memory on ESP-12E, hence, they can't be used for SPI applications. We can use the Hardware SPI interface for user-end applications.

The Fig. 8 shows Quad SPI interface pins that are internally used for flash. It consists of quad i/o (4-bit data bus) i.e. four (SDIO_DATA0 – SDIO_DATA3) bidirectional (i/p and o/p) data signals with synchronizing clock (SDIO_CLK) and chip select pin (SDIO_CMD). It is mostly used to get more bandwidth/throughput than dual i/o (2-bit data bus) interface. Table 4. contains the MOSI, CS, MISO, and SCLK's description.

Table 4. NodeMCU SPI Pins

MISO (Master In Slave Out)	The Master receives data and the slave transmits data through this pin.
MOSI (Master Out Slave In)	The Master transmits data and the slave receives data through this pin.
SCLK (Serial Clock)	Only the master can initiate a serial clock.
CS (Chip Select)	Master can select the slave device through this pin to start communication with it

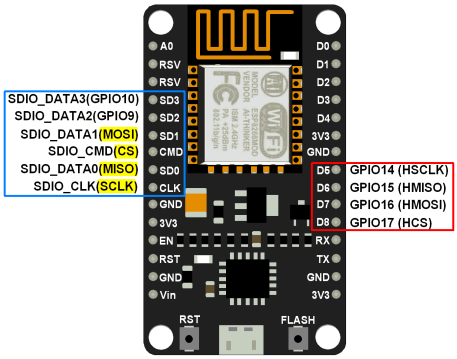


Fig. 9 NodeMCU SPI Pins

The following Arduino code requires 2 libraries from Adafruit Industries: “Adafruit_PCD8544.h” and “Adafruit_GFX.h”. The first library is a driver for the Nokia 5110 LCD (PCD8544 controller), and the second library is Adafruit graphics library.

In the “Adafruit_PCD8544.h” library, a class named “Adafruit_PCD8544” is defined. We define an instance named “display” of this class.

```
Adafruit_PCD8544 display = Adafruit_PCD8544(2, 0, 4, 5, 16);
```

In setup function, we initialize the display. We use the methods display(), begin(), setContrast(), and clearDisplay() in “Adafruit_PCD8544.h” class and drawRect(), setCursor(), setTextSize(), setTextColor() in “Adafruit_GFX.h” class.

```
void setup(void)
{
  pinMode(button.PIN, INPUT_PULLUP);
  attachInterrupt(button.PIN, isr, FALLING);

  // initialize the display
  display.begin();
  Wire.begin(D6, D5);  // set I2C pins [SDA = D6, SCL = D5], default clock is 100kHz
  display.setContrast(contrastValue);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(BLACK, WHITE);
  display.setCursor(3, 0);
  display.print("SHT20 SENSOR");
  display.display();
  display.setCursor(7, 10);
  display.print("TEMPERATURE:");
  display.setCursor(16, 31);
  display.print("HUMIDITY:");
  display.display();

  sht20.initSHT20();
  delay(100);
  sht20.checkSHT20();
  delay(5000);
}
```

```
void loop()
{
  if (button.pressed) {
    // get temperature and humidity from library
    float humd = sht20.readHumidity(); //get humidity
    float temp = sht20.readTemperature(); // get temperature

    display.setCursor(15, 20);
    if(temp < 0)
      display.printf("-%02u.%02u C", (int)abs(temp) % 100, (int)(abs(temp) * 100) %
100 );
    else
      display.printf(" %02u.%02u C", (int)temp % 100, (int)(temp * 100) % 100 );
    display.drawRect(53, 20, 3, 3, BLACK); // print degree symbol ( ° )
     display.setCursor(9, 41);
    display.printf("%04u.%02u", (int)(humd/100), (int)((uint32_t)humd % 100) );
    display.display();
    delay(2000);
  }

  static uint32_t lastMillis = 0;
  if (millis() - lastMillis > 10000) {
    lastMillis = millis();
    detachInterrupt(button.PIN);
  }
}
```

2.3. Interrupt Setup

In order to setup a button that by pressing the device will show the temperature and humidity on the screen, we should implement an Arduino code which contains a condition (pressing button.) We connected the pushbutton to ESP8266 (pin D7.)

First we defined a Button struct:

```
//Push Button
struct Button {
    const uint8_t PIN;
    uint32_t numberKeyPresses;
    bool pressed;
};
```

Then, we define the button we're going to use:

```
Button button = {13, 0, false}; //D7
```

Furthermore, we define two functions to interpret the pressing button action:

```
void IRAM_ATTR isr(void* arg) {
    Button* s = static_cast<Button*>(arg);
    s->numberKeyPresses += 1;
    s->pressed = true;
}
void IRAM_ATTR isr() {
    button.numberKeyPresses += 1;
    button.pressed = true;
}
```

In setup function, we define the pinmode of the button and the Falling action of the button as interrupt:

```
  pinMode(button.PIN, INPUT_PULLUP);
  attachInterrupt(button.PIN, isr, FALLING);
```

In the void loop we set a condition based on pressing the pushbutton:

```
void loop()
{
  if (button.pressed) {
    // get temperature and humidity from library
    float humd = sht20.readHumidity(); //get humidity
    float temp = sht20.readTemperature(); // get temperature

    display.setCursor(15, 20);
    if(temp < 0)
      display.printf("-%02u.%02u C", (int)abs(temp) % 100, (int)(abs(temp) * 100) %
100 );
    else
      display.printf(" %02u.%02u C", (int)temp % 100, (int)(temp * 100) % 100 );
    display.drawRect(53, 20, 3, 3, BLACK); // print degree symbol ( ° )
     display.setCursor(9, 41);
    display.printf("%04u.%02u", (int)(humd/100), (int)((uint32_t)humd % 100) );
    display.display();
    delay(2000);
  }

  static uint32_t lastMillis = 0;
  if (millis() - lastMillis > 10000) {
    lastMillis = millis();
    detachInterrupt(button.PIN);
  }
}
```

2.4. WiFi and MQTT Setup

There are three modes of WiFi Operation in the ESP8266 WiFi Module. They are:

- Station Mode (STA)
- Soft Access Point (AP)
- Soft AP + Station

We'll set up WiFi in Station Mode (STA), the ESP8266 WiFi Module will be connected to a WiFi Network that is already setup by an Access Point, like a WiFi Router. We'll use the

“ESP8266WiFi” library to program the ESP8266 WiFi Module. First we need to define the SSID and Password of the WiFi we’re connecting, as constants.

```
#define STASSID "UT-WiFi"
#define STAPSK  "12345678"
```

MQTT stands for Message Queuing Telemetry Transport, a simple messaging protocol suitable for communication between IoT devices. MQTT communication works as a publish and subscribe system. Devices publish messages on a specific topic. All devices that are subscribed to that topic receive the message.

The MQTT broker is responsible for receiving all messages, filtering the messages, deciding who is interested in them, and then publishing the message to all subscribed clients.

To use MQTT with the ESP8266 we’ll use the “Async MQTT Client” and “ESPAsyncTCP” libraries.

First, we define the MQTT host(IP Address) and MQTT port so that the ESP8266 connects to our broker.

```
#define MQTT_HOST IPAddress(192, 168, 1, XXX) //Dynamic
#define MQTT_PORT 1883
```

Then, The temperature and humidity will be published on the following topics:we define the topics of MQTT we’re going to use:

```
// Temperature MQTT Topics
#define MQTT_PUB_TEMP "esp/dht/temperature"
#define MQTT_PUB_HUM  “esp/dht/humidity"
```

We create an “AsyncMqttClient” object called “mqttClient” to handle the MQTT client and timers to reconnect to your MQTT broker and router when it disconnects.

```
AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;

WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisconnectHandler;
Ticker wifiReconnectTimer;
```

the connectToWifi() connects your ESP8266 to your router:

```
void connectToWifi() {
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
}
```

The connectToMqtt() connects your ESP8266 to your MQTT broker:

```
void connectToMqtt() {
  Serial.println("Connecting to MQTT…");
  mqttClient.connect();
}
```

The onWifiConnect() and onWifiDisconnect() functions are responsible for handling the Wi-Fi events. For example, after a successful connection with the router and MQTT broker, it prints the ESP8266 IP address. On the other hand, if the connection is lost, it starts a timer and tries to reconnect.

```
void onWifiConnect(const WiFiEventStationModeGotIP& event) {
  Serial.println("Connected to Wi-Fi.");
  connectToMqtt();
}

void onWifiDisconnect(const WiFiEventStationModeDisconnected& event) {
  Serial.println("Disconnected from Wi-Fi.");
  mqttReconnectTimer.detach();
  wifiReconnectTimer.once(2, connectToWifi);
}
```

The onMqttConnect() function runs after starting a session with the broker.

```
void onMqttConnect(bool sessionPresent) {
  Serial.println("Connected to MQTT.");
  Serial.print("Session present: ");
  Serial.println(sessionPresent);
}
```

If the ESP8266 loses connection with the MQTT broker, it calls the “onMqttDisconnect” function that prints that message in the serial monitor.

```
void onMqttDisconnect(AsyncMqttClientDisconnectReason reason) {
  Serial.println("Disconnected from MQTT.");

  if (WiFi.isConnected()) {
    mqttReconnectTimer.once(2, connectToMqtt);
  }
}
```

When you publish a message to an MQTT topic, the onMqttPublish() function is called. It prints the packet id in the Serial Monitor.

```
void onMqttPublish(uint16_t packetId) {
  Serial.println("Publish acknowledged.");
  Serial.print("  packetId: ");
  Serial.println(packetId);
}
```

WiFi and MQTT setup is in the setup function:

```
wifiConnectHandler = WiFi.onStationModeGotIP(onWiFiConnect);
wifiDisonnectHandler = WiFi.onStationModeDisconnected(onWiFiDisconnect);
mqttClient.onConnect(onMqttConnect);
mqttClient.onDisconnect(onMqttDisconnect);
mqttClient.onPublish(onMqttPublish);
mqttClient.setServer(MQTT_HOST, MQTT_PORT) ;

connectToWiFi();
delay(10000);
```

In the next step we used “moquitto” as a MQTT Broker and Node-Red to publish the reading data of sensor on our server.

- **Node-Red**

After installing, we’ll run Node-Red in our cmd using this command:

```
>node-red
```

We also need to connect the mqtt server using this command:

```
>mosquitto -v -c ./mosquitto.conf
```

Fig. 10 shows the dashboard of Node-Red.

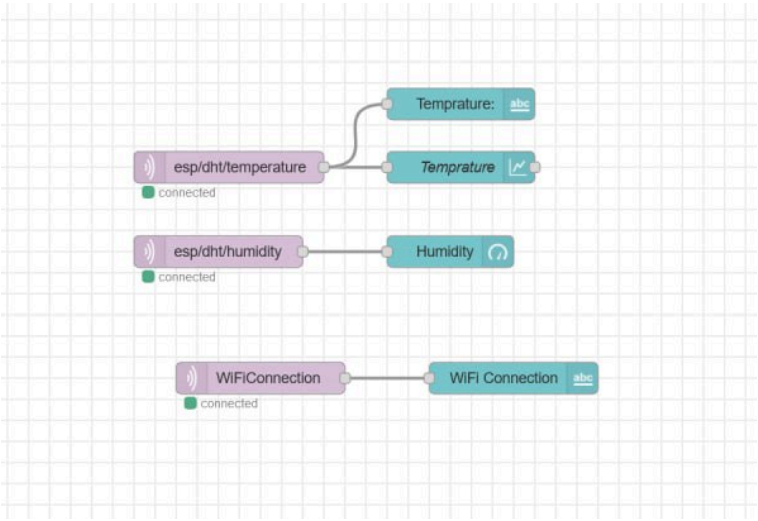


Fig. 10 Node-Red Diagram

Two “mqtt-in” are added to publish the temperate and humidity data. The humidity data is published in gauge format and temperature data will be shown in a chart which shows the last 30 minutes data of temperature. There also exits the same diagram to get the WiFi status and shows that either it's connected or not.

The codes that publish data on the server are:

```
#define MQTT_PUB_TEMP "esp/dht/temperature"
#define MQTT_PUB_HUM "esp/dht/humidity"
#define MQTT_PUB_WiFi “WiFiConnection"

uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());

Serial.println("Temp Sent");
uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(humd).c_str());

  if(WiFi.isConnected()){

  uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_WiFi, 1, true,
String(STASSID).c_str());

  }

String noneCon = “Disconnected";
uint16_t packetIdPub4 = mqttClient.publish(MQTT_PUB_WiFi, 1, true, noneCon.c_str());
```

2.5. Deep Sleep Setup

The described system is a mobile system and takes its power from the battery, so an important point in its implementation is to optimize energy consumption so that it can be active for a longer period of time without the need for service and battery replacement. Using the Deep Sleep and Wakeup modes that exist in this processor, we implemented the following two scenarios:

1. In normal mode, the device should be activated once in a minute from deep sleep mode, start its display and sensor and connect to Wi-Fi, after reading the temperature and humidity, show it on the screen and send it via MQTT, after doing If the previous steps are successful, it will go back to deep sleep mode and repeat the previous action after one minute.

2. If the defined button is pressed, the device will be activated from sleep mode, after starting the communication, it will show the temperature and humidity on the screen and send it via MQTT, after completing the mentioned steps, the timer will be reset.

- **Change Wiring**

In this section, we changed the wiring to the Fig. 11, in order to setup the deep sleep.

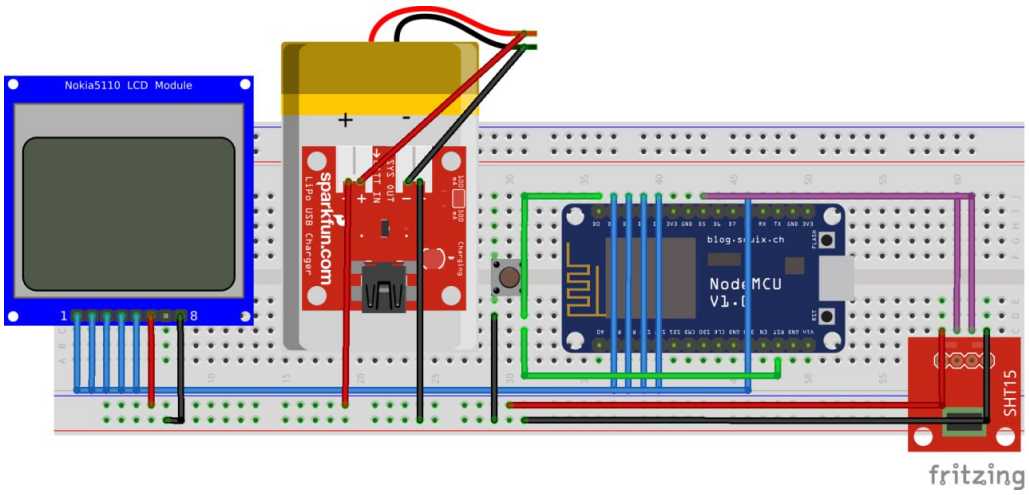


Fig. 11 New Circuit and Wiring

In order to make the system sleep for a minute, we add the following code at the end of the setup function, where every code in loop function is also added there:

```
ESP.deepSleep(60e6); //20 second
```

We connected the pushbutton to the “rst” GPIO in ESP8266. Therefore, when the pushbutton is pressed, the system restarts and will wake up as we desired.

The final results can be seen in node-red ui in Fig. 12.

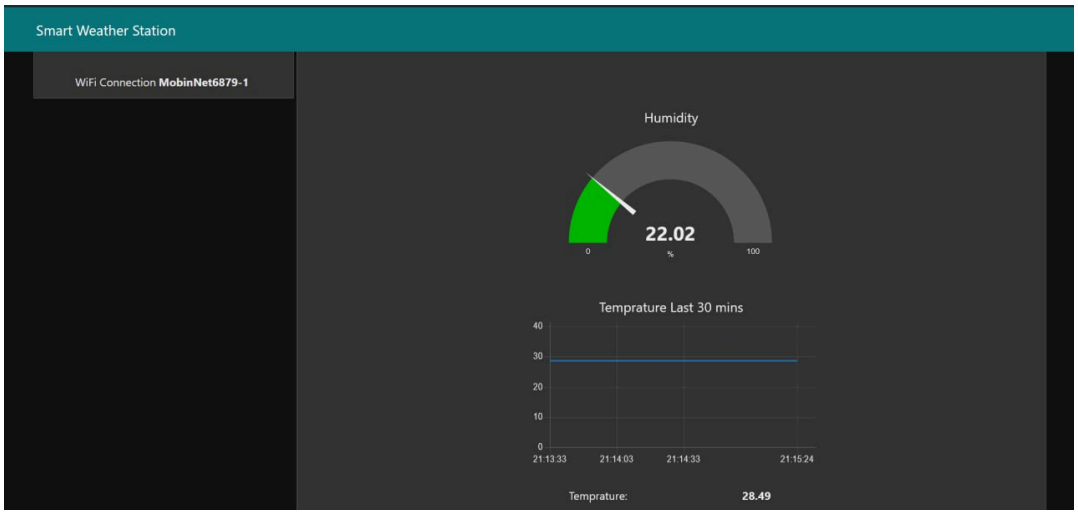


Fig. 12 Final Result

It’s necessary to mention that the sensor data will also shown in NOKIA 5110 LCD, in Fig. 13.

Fig. 11 Final Result in LCD

The complete code is in the following box:

```
#ifndef STASSID
#define STASSID "MobinNet6879-1"
#define STAPSK  “XXXXXXXXX"
#define MQTT_HOST IPAddress(192, 168, 1, 153)

#define MQTT_PORT 1883
#define MQTT_PUB_TEMP "esp/dht/temperature"
#define MQTT_PUB_HUM "esp/dht/humidity"
#define MQTT_PUB_WiFi "WiFiConnection"


#endif

#include <Wire.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_PCD8544.h>
#include "DFRobot_SHT20.h"
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <Ticker.h>
#include <AsyncMqttClient.h>
#include <String.h>

AsyncMqttClient mqttClient;
Ticker mqttReconnectTimer;


WiFiEventHandler wifiConnectHandler;
WiFiEventHandler wifiDisonnectHandler;
Ticker wifiReconnectTimer;
```

```
unsigned long previousMillis = 0;
const long interval = 50;


void connectToWiFi(){
  Serial.println();
  Serial.println("Connecting to WiFi...");
  WiFi.begin(STASSID, STAPSK);
}

void onWiFiConnect(const WiFiEventStationModeGotIP& event){
  Serial.println("Connected To WiFi.");
  connectToMqtt();
}

void onWiFiDisconnect(const WiFiEventStationModeDisconnected& event){
  Serial.println("Disconnected from WiFi.");
  mqttReconnectTimer.detach();
  wifiReconnectTimer.once(2, connectToWiFi);
  }

void connectToMqtt(){
  Serial.println("Connecting to MQTT...");
  mqttClient.connect();

}

void onMqttConnect(bool sessionPresent){
  Serial.println("Connected to MQTT.");
  Serial.print("Session present: ");
  Serial.println(sessionPresent);

}

void onMqttDisconnect(AsyncMqttClientDisconnectReason reason){
  Serial.println("Disconnected from MQTT.");
  if(WiFi.isConnected()){
    mqttReconnectTimer.once(2, connectToMqtt);
  }
}

void onMqttPublish(uint16_t packetID){
  Serial.print("Publish acknowledged.");
  Serial.print("packetID:");
  Serial.println(packetID);
}

//Push Button
struct Button {
  const uint8_t PIN;
  uint32_t numberKeyPresses;
  bool pressed;
};

Button button = {13, 0, false}; //D7

void IRAM_ATTR isr(void* arg) {
  Button* s = static_cast<Button*>(arg);
  s->numberKeyPresses += 1;
  s->pressed = true;
}
```

```
void IRAM_ATTR isr() {
  button.numberKeyPresses += 1;
  button.pressed = true;
}

//SHT20 Sensor
Adafruit_PCD8544 display = Adafruit_PCD8544(15, 2, 0, 4, 5);
int contrastValue = 60;
DFRobot_SHT20 sht20;

///Web server
const char* ssid = STASSID;
const char* password = STAPSK;

ESP8266WebServer server(80);



void setup(void)
{

  Serial.begin(115200);
  Serial.println("I'm awake");
  delay(500);

  wifiConnectHandler = WiFi.onStationModeGotIP(onWiFiConnect);
  wifiDisonnectHandler = WiFi.onStationModeDisconnected(onWiFiDisconnect);
  mqttClient.onConnect(onMqttConnect);
  mqttClient.onDisconnect(onMqttDisconnect);
  mqttClient.onPublish(onMqttPublish);
  mqttClient.setServer(MQTT_HOST, MQTT_PORT) ;

  connectToWiFi();
  delay(10000);

  if(WiFi.isConnected()){

    uint16_t packetIdPub3 = mqttClient.publish(MQTT_PUB_WiFi, 1, true,
String(STASSID).c_str());

  }

  //pinMode(button.PIN, INPUT_PULLUP);

  attachInterrupt(button.PIN, isr, FALLING);


  // initialize the display
  display.begin();
  Wire.begin(D6, D5);  // set I2C pins [SDA = D6, SCL = D5], default clock is 100kHz
  display.setContrast(contrastValue);
  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(BLACK, WHITE);
  display.setCursor(3, 0);
  display.print("SHT20 SENSOR");
  display.display();
  display.setCursor(7, 10);
  display.print("TEMPERATURE:");
  display.setCursor(16, 31);
  display.print("HUMIDITY:");
  display.display();
```

```
  sht20.initSHT20();
  delay(100);
  sht20.checkSHT20();
  delay(100);

  float humd = sht20.readHumidity();
  float temp = sht20.readTemperature();

  Serial.println();
  Serial.print(temp);
  Serial.print(" ");
  Serial.print(humd);
  Serial.println();
  Serial.println("here3");


  uint16_t packetIdPub1 = mqttClient.publish(MQTT_PUB_TEMP, 1, true,
String(temp).c_str());
  Serial.println("Temp Sent");
  uint16_t packetIdPub2 = mqttClient.publish(MQTT_PUB_HUM, 1, true,
String(humd).c_str());

  Serial.println("Humd Sent");

  delay(1000);


  display.setCursor(15, 20);
  if(temp < 0)
    display.printf("-%02u.%02u C", (int)abs(temp) % 100, (int)(abs(temp) * 100) %
100 );
  else
    display.printf(" %02u.%02u C", (int)temp % 100, (int)(temp * 100) % 100 );
    display.drawRect(53, 20, 3, 3, BLACK); // print degree symbol ( ° )

  display.setCursor(9, 41);
  display.printf("%04u.%02u", (int)(humd/100), (int)((uint32_t)humd % 100) );

  display.display();

  delay(10000);
  String noneCon = "Disconnected";
  uint16_t packetIdPub4 = mqttClient.publish(MQTT_PUB_WiFi, 1, true,
noneCon.c_str());
  delay(500);


  display.clearDisplay();
  display.display();
  Serial.println("I'm awake, but I'm going into deep sleep mode for 60 seconds");
  ESP.deepSleep(60e6); //20 second

}

// main loop
void loop(){

}
```