BÁO CÁO

HỆ THỐNG QUAN TRẮC KHÍ TƯỢNG

***

Năm học	HK01, NH2024-2025
Môn học	Cơ sở ứng dụng và IoT
GVHD	Trương Quang Phúc
Nhóm	MEOW
Thành viên	1. Nguyễn Thanh Phú - 221191211 2. Trần Đức Tài - 22119226 3. Nguyễn Hữu Trí - 22119244 4. Vũ Mai Liên - 22119194 5. Trần Thuỷ Tiên - 22119238

MỤC TIÊU

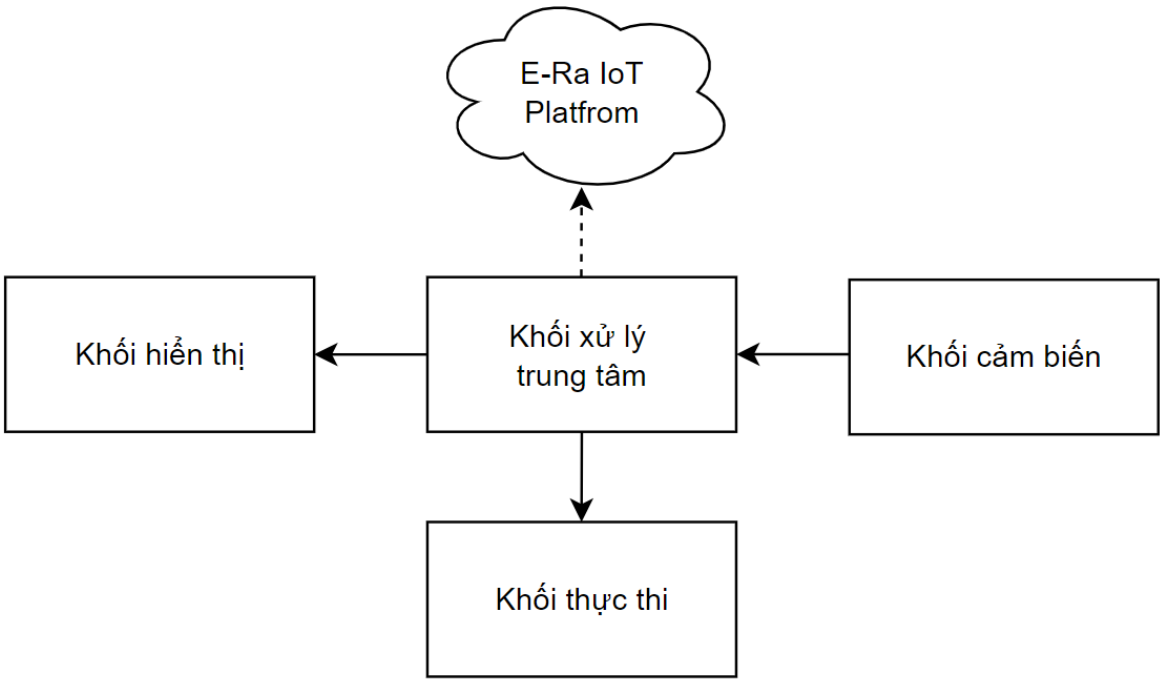
Ứng dụng kiến thức về lập trình, vi điều khiển và các nền tảng IoT để xây dựng hệ thống quan trắc khí tượng. Ngoài vai trò phục vụ phòng chống thiên tai, lũ lụt, thông tin có được từ trạm quan trắc khí tượng cũng được sử dụng để phục vụ ngành nông nghiệp như trồng trọt, chăn nuôi; hoặc cung cấp thông tin cho ngành điện lực, trạm quan trắc khí tượng có dự báo trước về dòng chảy để đảm bảo dung tích hồ chứa sản xuất điện tốt nhất; đối với ngành hàng không, hàng hải cũng rất cần thông tin của trạm quan trắc khí tượng thủy văn để phòng ngừa, ứng phó các sự cố trong quá trình hoạt động lưu thông.

Các nội dung được quan trắc bao gồm:

1. Nhiệt độ
2. Độ ẩm
3. Tốc độ gió
4. Hướng gió

Từ đó góp phần cung cấp dữ liệu khí tượng để tính toán và dự báo thời tiết.

SƠ ĐỒ KHỐI HỆ THỐNG



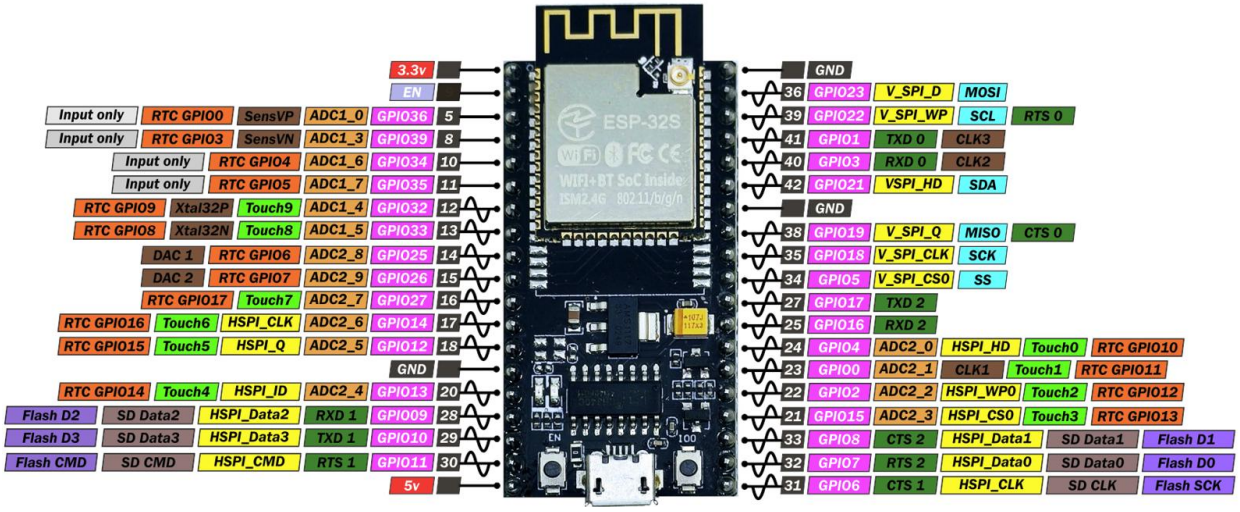
Hình 1 – Sơ đồ khối hệ thống.

- Khối hiển thị: Bao gồm một LED đơn để báo hiệu quá trình hoạt động của hệ thống và các cảnh báo lỗi thông qua số lần nhấp nháy.
- Khối thực thi: Bao gồm hai LED đơn đại diện cho hai thiết bị ở trạm quan trắc.
- Khối cảm biến: bao gồm hai cảm biến, DHT11 để thu thập dữ liệu về nhiệt độ và độ ẩm tại môi trường và Weather Station WS-3000 Probe để thu thập dữ liệu về tốc độ gió và hướng gió.

- Khối xử lý trung tâm: bao gồm một ESP32 NodeMCU-32S của Ai-Thinker để xử lý và đồng bộ dữ liệu với nền tảng E-Ra.

1 - Khối xử lý trung tâm

Sơ đồ chân:



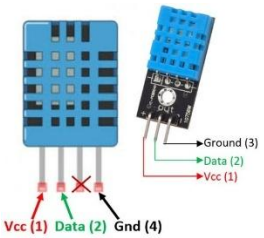
Hình 2 – Sơ đồ các chân của ESP32 NodeMCU-32S Ai-Thinker.

Thông số kỹ thuật:

Mô-đun trung tâm	Ai-Thinker ESP32-S
Tốc độ SPI Flash	32Mbits
Tần số làm việc	2400~2483.5Mhz
Bluetooth	BLE 4.2 BR/EDR
Wi-Fi	802.11 b/g/n/e/i
Giao diện kết nối	UART/SPI/SDIO/I2C/PWM/I2S/IR/ADC/DAC
Giao tiếp nối tiếp (Serial)	UART CH340

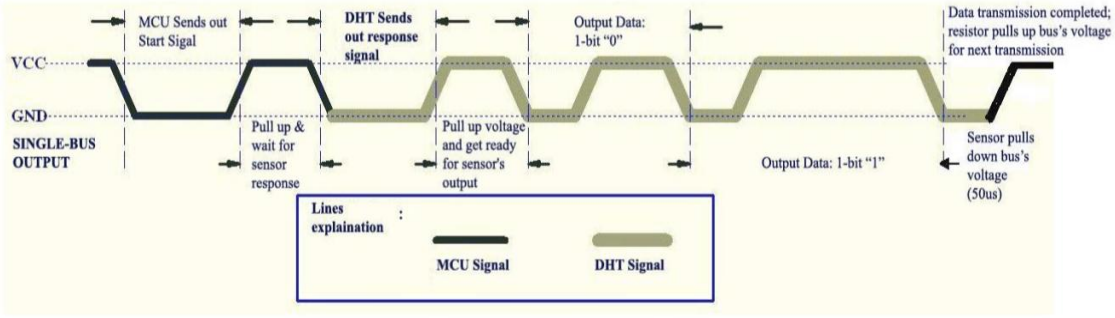
2 - Khối cảm biến

Cảm biến DHT11:



Hình 3 – Sơ đồ chân của DHT11

Giao thức truyền nhận dữ liệu giữa MCU và DHT11:



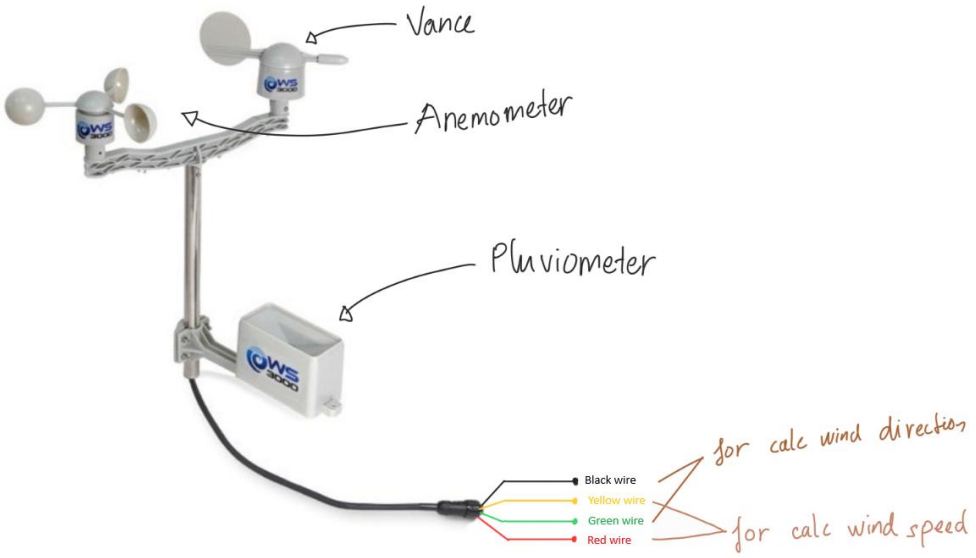
Hình 4 – Giao thức đọc dữ liệu trên DHT11.

Khung truyền dữ liệu của DHT11, truyền bit có trọng số cao trước:

MSB LSB

8 bit	8 bit	8 bit	8 bit	8 bit
Độ ẩm		Nhiệt độ		Check-Sum
Phần nguyên	Phần thập phân	Phần nguyên	Phần thập phân	

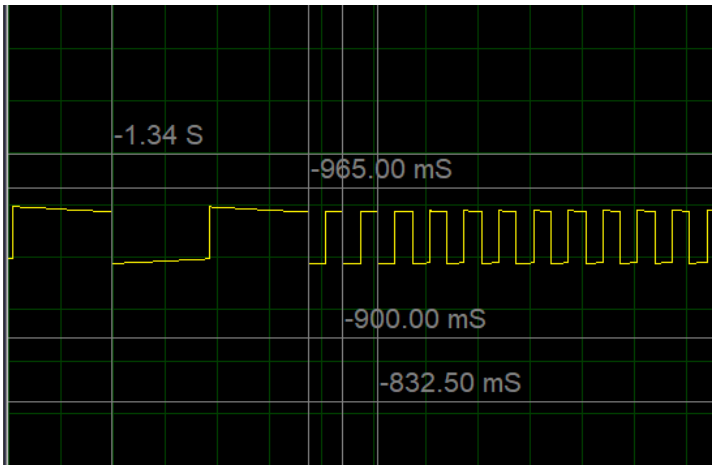
Weather Station WS-3000 Probe:



Hình 5 – Sơ đồ chân của Weather Station WS-3000 Probe.

Tính toán tốc độ gió:

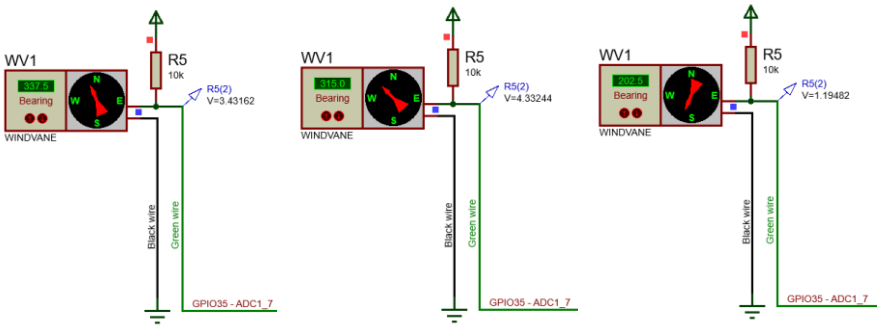
Dữ liệu về tốc độ gió là một tín hiệu kỹ thuật số có tần số tỉ lệ thuận với tốc độ gió như Hình 6.



Hình 6 – Tín hiệu về tốc độ gió (Proteus).

Tốc độ gió = 2.4*(Số vòng/Thời gian khảo sát) (Km/h)

Tính toán hướng gió:



Hình 7 – Giá trị ngõ ra của cảm biến hướng gió tuỳ thuộc vào hướng.

Ta có công thức điện áp ngõ ra tại *Green wire*:

$$Y_{green\ wire} = \frac{VR}{10+VR} \cdot 3{,}3\ (V) = \frac{VR}{10+VR} \cdot 1023(ADC's\ unit)$$

Từ đó quy đổi ra góc hiện tại dựa theo bảng sau:

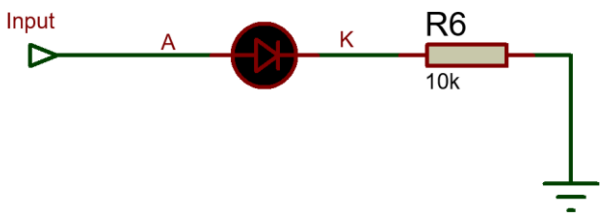
Angle°	R(KΩ)	Y_1 (mV)	ADC $(= \frac{Y_1}{3{,}3} \cdot 1023)$
0	33	2532	786
22,5	6,57	1308	406
45	8,2	1487	461
67,5	0,891	270	89
90	1	300	93
112,5	0,688	212	65
135	2,2	595	185
157,5	1,41	407	126
180	3,9	926	287
202,5	3,19	789	244
225	16	2030	630
247,5	19,12	1931	599
270	120	3046	945
292,5	42,12	2666	827
315	69,9	2859	887
337,5	21,8	2262	702

Δ = 9 ADC's unit

Δ = 28 ADC's unit

1 (ADC's unit) = 0,0032258 (V) = 3,23 (mV)

3,4 – Khối hiển thị, khối chấp hành



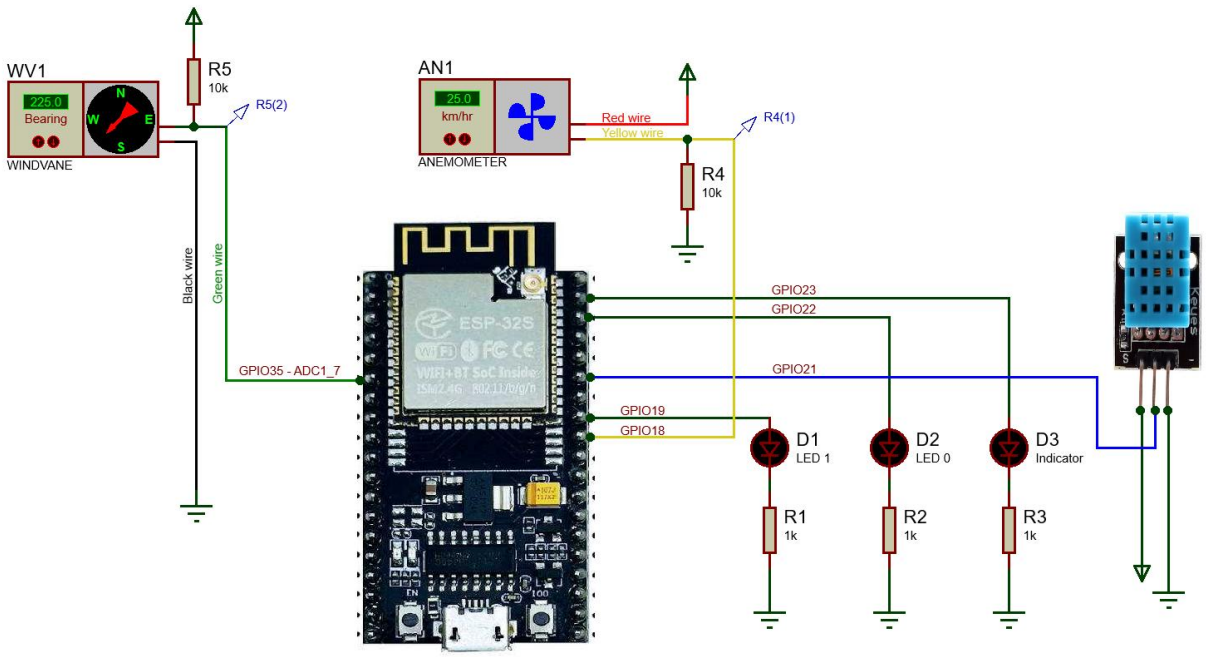
Hình 8 – Sơ đồ kết nối một LED đơn.

5 – Nền tảng E-Ra IoT

E-Ra IoT Platform là nền tảng IoT mở của người Việt, được phát triển và vận hành bởi đội ngũ EoH, giúp hỗ trợ, đồng hành và cho phép các nhà phát triển phần mềm và thiết bị IoT có thể

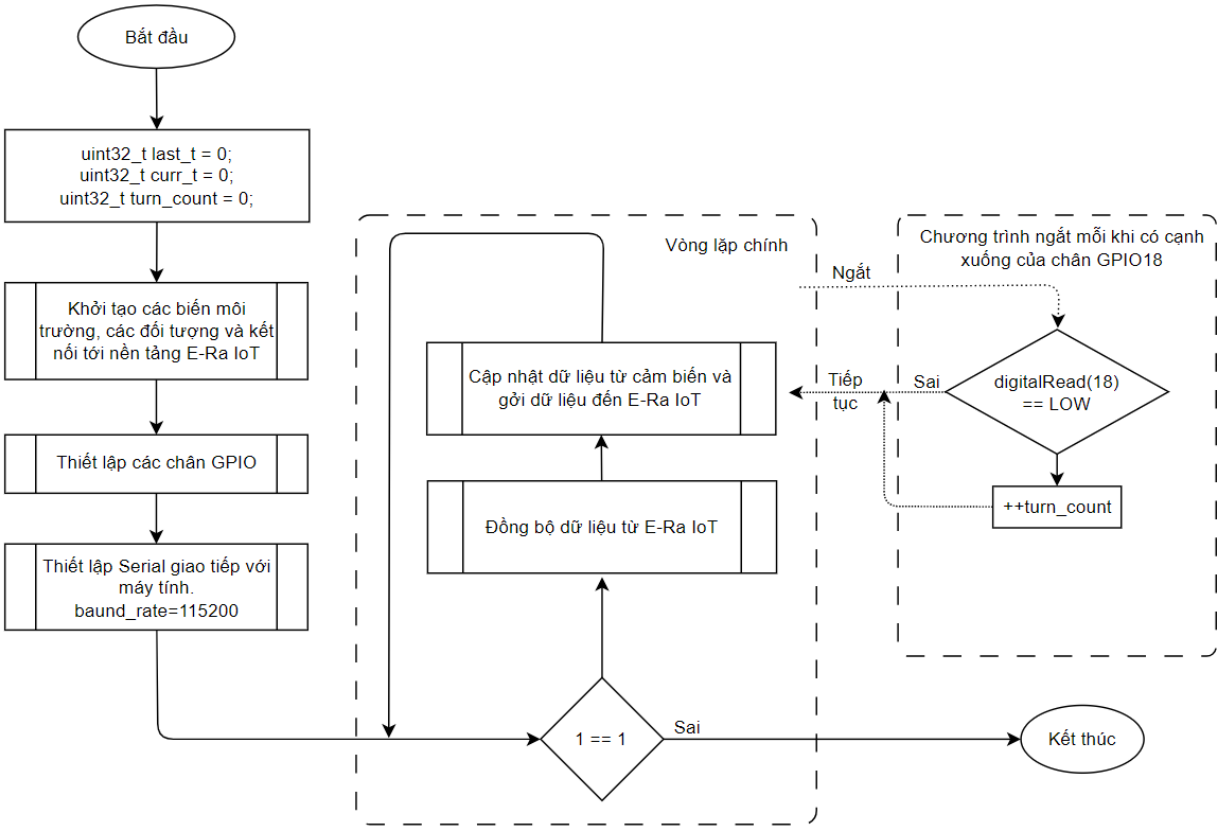
theo dõi kết quả dự án trên giao diện đẹp mắt, đồng nhất và chuyên nghiệp (Web Dashboard/iOS và Android App).

SƠ ĐỒ NGUYÊN LÝ

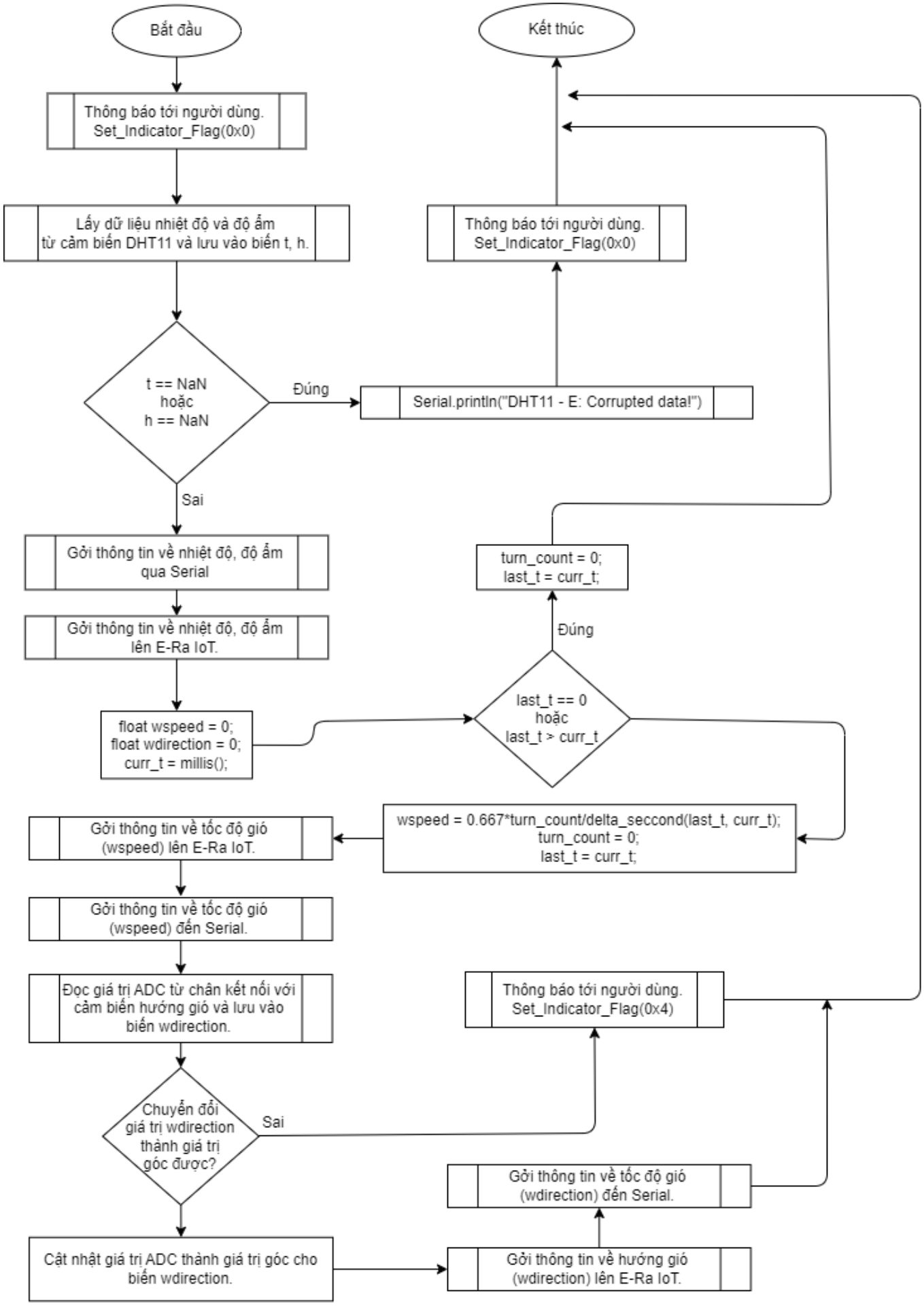


Hình 9 – Sơ đồ nguyên lý của hệ thống.

LƯU ĐỒ CHƯƠNG TRÌNH

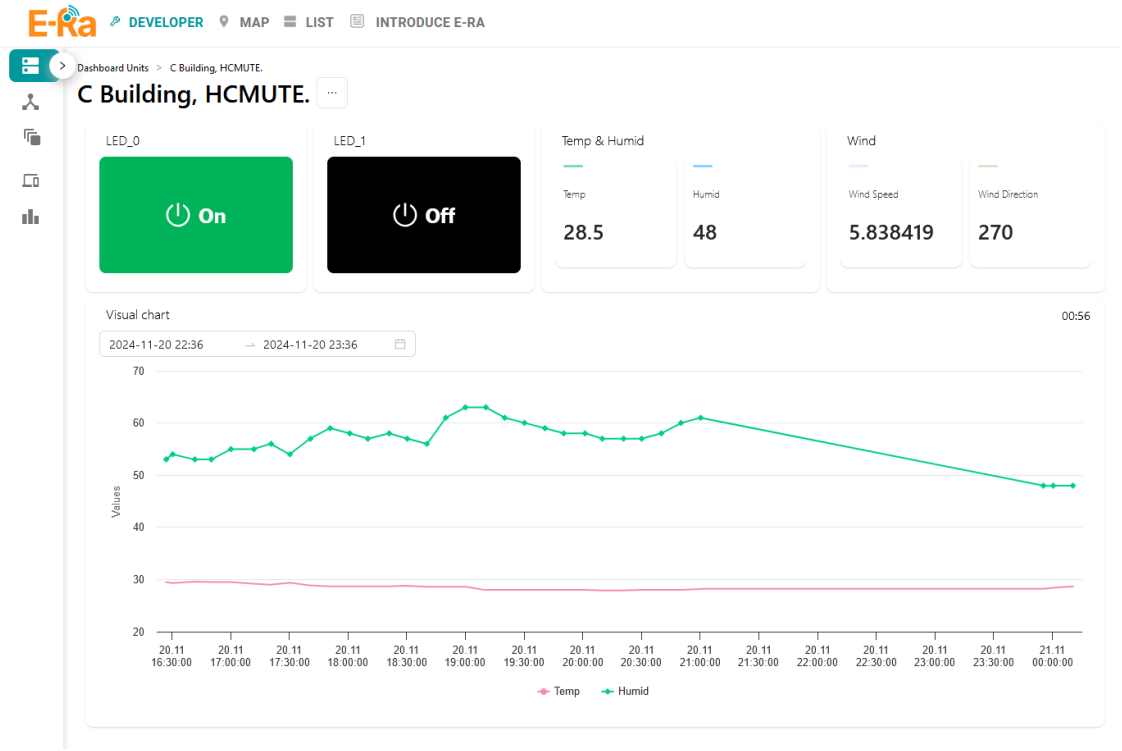


Hình 10 – Lưu đồ chương trình chính.

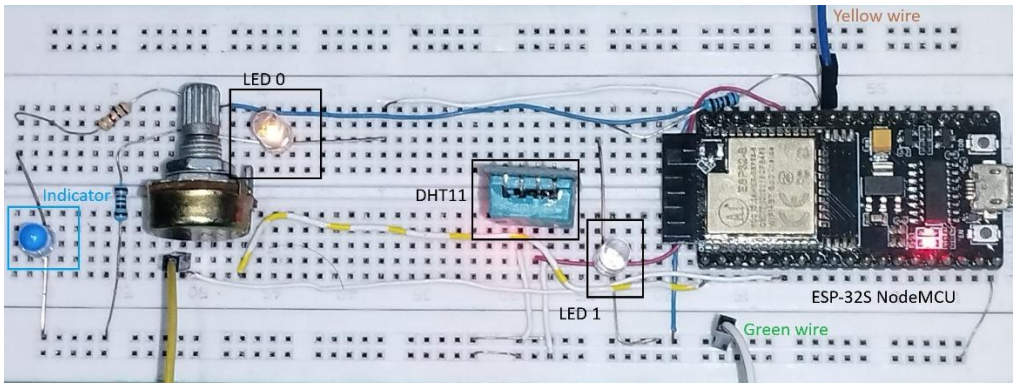


Hình 11 – Lưu đồ chương trình cập nhật dữ liệu từ cảm biến và gởi dữ liệu lên E-Ra IoT.

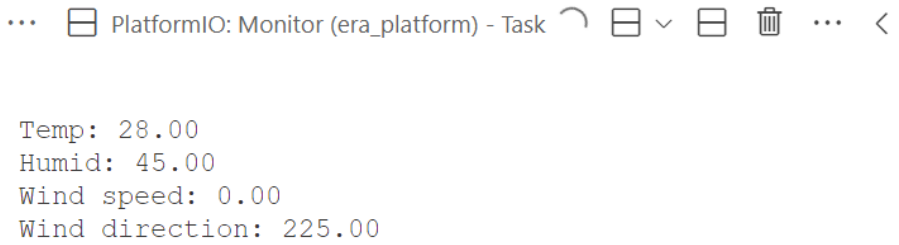
KẾT QUẢ



Hình 12 - Dashboard trên nền tảng E-Ra.



Hình 13 – Kết quả khi bật LED 0 ở Hình 12.



Hình 14 – Dữ liệu nhận được thông qua Serial.

NHẬN XÉT

Hệ thống hoạt động đúng với thiết kế ban đầu. Chu kỳ cập nhật dữ liệu là ~4 giây. Phản hồi phần cứng (phần cứng có sự thay đổi về trạng thái ngõ ra tương tứng) và phần mềm (nhận được trị giá trị ngõ ra thực tế từ ESP-32S được gởi trở lại E-Ra IoT) sau khi nhấn nút giao động từ <1 giây đến <4 giây.

- Hạn chế:

  + Chưa có đầy đủ các giá trị mà một trạm quan trắc khí tượng cần có (giá trị tia UV, áp suất khí quyển, lưu lượng mưa, …)

  + Thời gian phản hồi từ ~1 → ~4 giây vẫn còn là tương đối chậm.

  + Hệ thống còn sơ sài, chưa có ứng dụng trên điện thoại thông minh.

  + Chưa có lưu trữ giá trị cảm biến, chỉ hiển thị theo thời than thực.

- Hướng phát triển trong tương lai:

  + Gia tăng số lượng cảm biến, để cung cấp đủ các trị số mà một trạm quan trắc khí tượng cần có (giá trị tia UV, áp suất khí quyển, lưu lượng mưa, …)

  + Cải thiện thời gian phản hồi.

  + Thêm các tính năng thông minh (phát hiện dữ liệu bất thường và gởi cảnh báo, …)

MÃ NGUỒN

main.cpp

```
#define DEFAULT_MQTT_HOST "mqtt1.eoh.io"
#define ERA_AUTH_TOKEN "a48df8f0-3073-430a-9148-b89a532c778d"

#define delta_seccond(t_i, t_f) (((t_f)-(t_i))/1000.0f)
#define in_accepted_range(mean, x) (((mean)-4<=(x)) && ((x)<=(mean)+4))

#define DHTTYPE DHT11
#define DHT_DAT_PIN 21
#include <DHT.h>
DHT dht(DHT_DAT_PIN, DHTTYPE);

#define LED_CTL_PIN_1 22
#define INDICATOR_LED_PIN 23
#define LED_CTL_PIN_2 19
#define WIND_SPEED_PIN 18
#define WIND_DIRECTION_PIN 35

#include <Arduino.h>
#include <ERa.hpp>
#include <ERa/ERaTimer.hpp>

const char ssid[] = "rm.note.11";
const char pass[] = "nGXXFUS@3204";

WiFiClient mbTcpClient;

ERA_CONNECTED() {
    ERA_LOG(ERA_PSTR("ERa"), ERA_PSTR("ERa connected!"));
}

ERA_DISCONNECTED() {
    ERA_LOG(ERA_PSTR("ERa"), ERA_PSTR("ERa disconnected!"));
}

// Codes:
// 0x0: Read data from sensor
// 0x1: DHT Received data corrupted
// 0x2: WiFi error
// 0x3: Database connection error
// 0x4: Speed direction error
void Set_Indicator_Flag(int err_code = 0){
  // Blink led to noti error
  switch (err_code){
    case 0x0: err_code = 1; break;
    case 0x1: err_code = 2; break;
    case 0x2: err_code = 3; break;
    case 0x3: err_code = 4; break;
    case 0x4: err_code = 5; break;
    default:  err_code = 0;
  }
  while(err_code--){
    digitalWrite(INDICATOR_LED_PIN, LOW); delay(50);
    digitalWrite(INDICATOR_LED_PIN, HIGH); delay(100);
    digitalWrite(INDICATOR_LED_PIN, LOW); delay(50);
  }
}

uint32_t last_t = 0;
uint32_t curr_t = 0;
uint32_t turn_count = 0;
void Wind_Speed_Count_Turn(){
    if ( digitalRead(WIND_SPEED_PIN) == LOW )
    turn_count++;
}

void Initial_GPIO_Pin(){
    pinMode(LED_CTL_PIN_1, OUTPUT);
    pinMode(INDICATOR_LED_PIN, OUTPUT);
    pinMode(LED_CTL_PIN_2, OUTPUT);
    pinMode(WIND_SPEED_PIN, INPUT_PULLDOWN);
    attachInterrupt( digitalPinToInterrupt(WIND_SPEED_PIN), Wind_Speed_Count_Turn, FALLING);
    pinMode(WIND_DIRECTION_PIN, INPUT_PULLUP);
    analogReadResolution(10);
}

ERA_WRITE(V3){
    int value = param.getInt();
    (value)?(digitalWrite(LED_CTL_PIN_1,HIGH)):
            digitalWrite(LED_CTL_PIN_1,LOW);
};
```

```
ERA_WRITE(V4){
    int value = param.getInt();
    (value)?(digitalWrite(LED_CTL_PIN_2,HIGH)):
             digitalWrite(LED_CTL_PIN_2,LOW);
};

void Update_Sensor_Data(){
    Serial.println();
    Set_Indicator_Flag(0x0);
    // Update LED pins
    ERa.virtualWrite(V4, digitalRead(LED_CTL_PIN_2));
    ERa.virtualWrite(V3, digitalRead(LED_CTL_PIN_1));
    // Read humid and temp from dht sensor
    delay(4000);
    float h = dht.readHumidity();
    float t = dht.readTemperature();
    // Check read data
    if( isnan(t) || isnan(h) ){
            // Send Error msg to serial
            Serial.println("DHT11 - E: Corrupted data!");
            Set_Indicator_Flag(0x1);
        }else{
        // Send Temp + Humid data to serial
        Serial.print("Temp: ");
        Serial.println(t);
        Serial.print("Humid: ");
        Serial.println(h);
        // Update data sensor
        ERa.virtualWrite(V0, t);
        ERa.virtualWrite(V1, h);
    }
    //WIND SPEED
    float wspeed = 0;
    float wdirection = 0;
    curr_t = millis();
    if( !last_t || last_t > curr_t){
        turn_count = 0;
        last_t = curr_t;
        return;
    }{
        wspeed = 0.667*turn_count/delta_seccond(last_t, curr_t);
        turn_count = 0;
        last_t = curr_t;
        ERa.virtualWrite(V5, wspeed);

        Serial.print("Wind speed: ");
        Serial.println(wspeed);
    }
    //WIND DIRECTION
    wdirection = analogRead(WIND_DIRECTION_PIN);
    if(in_accepted_range(786, wdirection))
        wdirection = 0;
    else if(in_accepted_range(406, wdirection))
        wdirection = 22.5;
    else if(in_accepted_range( 461, wdirection))
        wdirection = 45;
    else if(in_accepted_range( 84, wdirection))
        wdirection = 67.5;
    else if(in_accepted_range( 93, wdirection))
        wdirection = 90;
    else if(in_accepted_range( 65, wdirection))
        wdirection = 112.5;
    else if(in_accepted_range( 185, wdirection))
        wdirection = 135;
    else if(in_accepted_range( 126, wdirection))
        wdirection = 157.5;
    else if(in_accepted_range( 287, wdirection))
        wdirection = 180;
    else if(in_accepted_range( 244, wdirection))
        wdirection = 202.5;
    else if(in_accepted_range( 630, wdirection))
        wdirection = 225;
    else if(in_accepted_range( 599, wdirection))
        wdirection = 247.5;
    else if(in_accepted_range( 945, wdirection))
        wdirection = 270;
    else if(in_accepted_range( 827, wdirection))
        wdirection = 292.5;
    else if(in_accepted_range( 887, wdirection))
        wdirection = 315;
    else if(in_accepted_range( 702, wdirection))
        wdirection = 337.5;
    else {
```

```
        Set_Indicator_Flag(0x4);
        return;
    }
    Serial.print("Wind direction: ");
    Serial.println(wdirection);
    ERa.virtualWrite(V6, wdirection);

}

void timerEvent() {
    ERA_LOG(ERA_PSTR("Timer"), ERA_PSTR("Uptime: %d"), ERaMillis() / 1000L);
    Update_Sensor_Data();
}

void Initial_Serial(){
  Serial.begin(115200);
  Serial.println(("\n\nHello!\nFrom MEOW GROUP!"));
}

void setup() {
    Initial_Serial();
    Initial_GPIO_Pin();
    ERa.setModbusClient(mbTcpClient);
    ERa.setScanWiFi(true);
    ERa.begin(ssid, pass);
    ERa.addInterval(1000L, timerEvent);
}

void loop() {
    ERa.run();
}
```

ENV

```
[env:esp32dev]
platform = espressif32
board = esp32dev
framework = arduino
lib_deps =
    ;mbed-zrem/WiFi
    adafruit/DHT sensor library@^1.4.6
    adafruit/Adafruit Unified Sensor@^1.1.14
    ;mobizt/Firebase ESP32 Client@^4.4.14
    eoh-ltd/ERa@^1.4.3
monitor_speed = 115200
upload_speed = 921600
; upload_port = *
board_build.partitions = .pio/libdeps/$PIOENV/ERa/era_partition.csv
```