CENG3410 Smart Hardware Design
The Chinese University of Hong Kong
Lab 4: Arduino Weather Notifier(2): Implementation

Student ID:

2019 Spring

1 Objectives

1) To use Arduino and WiFi module to act as a web client;

2) To build a weather notifier, get weather information from the Internet and make notification when necessary.

2 Materials

1) Arduino Uno

2) ESP8266-Based Serial Wifi Shield

3) Breadboard

4) LEDs

5) Solid core wire

6) USB A to B wire

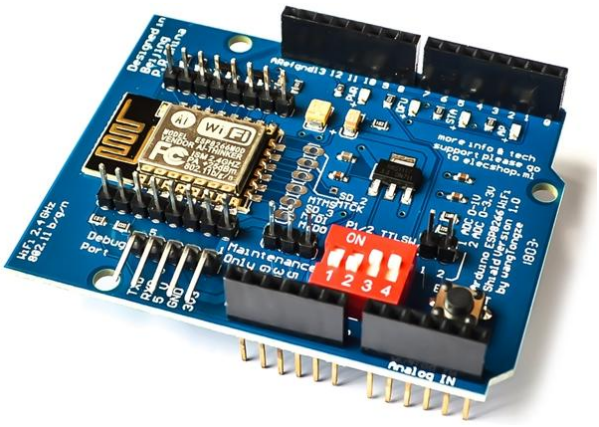

Figure 1: ESP8266-Based Serial WiFi Shield

3 Procedures

3.1 Configuration of WiFi module

Plug ESP8266 WIFI Module to Arduino Uno and power on the board.

Attention: SW 1, 2 on WiFi module control connection of Arduino with ESP8266. SW 1 allows Arduino to write to ESP8266 and SW2 allows Arduino to read from ESP8266. As Arduino Uno only has one serial port for downloading program, when uploading Arduino program with IDE, Arduino serial port should NOT be used and the SW 1, 2 should switch to OFF position to disconnect the serial port of ESP8266. When Arduino and WIFI shield start to work, SW 1, 2 should be on.

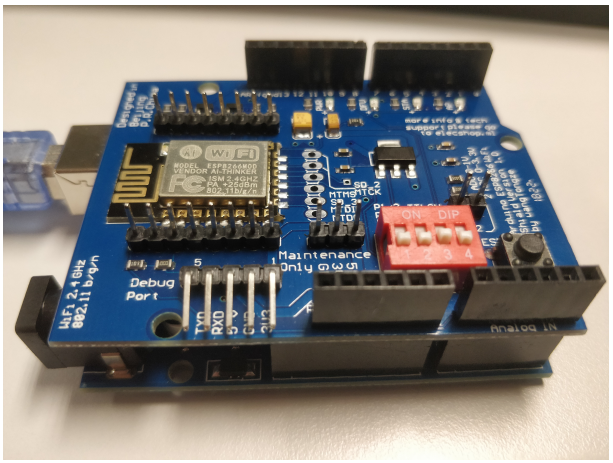

Figure 2: WiFi shield and Arduino

Refer to Lab3 and finish the following steps. You can connect to the WiFi module with your laptop or mobilephone.

- **Step 1:**

  Set the AP name by yourself to distinguish your board from the boards of other groups. (We have already done this step in Lab3. The default SSID "DoitWIFI_xx", where xx is the number on your WIFI shield, but you can change the AP name the your name or CUHK ID).

Figure 3: Find the WiFi signal

- **Step 2:**

  2) Connect the WIFI module to our provided WIFI server. SSID:CENG3410WIFI, Password:88888888.

For serial setting, keep the default parameters. Remote IP is the IP of the website we want to visit, here it is 192.168.1.101. Remote port is set as 8000.

IMPORTANT: Please be carefully to choose the right mode (AP/STA) in every step.

3.2 Access Weather Data

After the configuration, the WiFi module can be used to send and receive network traffic. In this lab, we use 192.169.1.101 to get weather information. This server gets weather information from Openweathermap.org, which provides flexible HTTP APIs supporting various types of queries (query with city name, city ID, geographic coordinates etc.). The weather information will be returned in the the JSON (JavaScript Object Notation) format.

The communication between your weather notifier and the weather information server is built on HTTP protocol. In this lab, you are required to construct a HTTP request according to the discription http://192.168.1.101:8000, and then parse the weather description from JSON document.

The HTTP (Hyper Text Transmission Protocol) functions as a request-response protocol in a client-sever model. In this lab, your Arduino board and WiFi module work as a client and send HTTP request message to the server. The server would return a HTTP response message containing the information you requested.

The request message consists of the following:

- **A request line** (e.g., GET /images/logo.png HTTP/1.1, which requests a resource called /images/logo.png from the server).
- **Request header fields** (e.g., Accept-Language: en).
- **An empty line.**
- **An optional message body.**

The **request line** and **other request header fields** must each end with $< CR >< LF >$(that is, a carriage return character followed by a line feed character). **The empty line must consist of only** $< CR >< LF >$**and no other whitespace**. In the HTTP/1.1 protocol, **all header fields except Host are optional**.

(In C++, a few characters can indicate a new line. The usual ones are these two:
'\n' or '0x0A' (10 in decimal) -> this character is called "Line Feed" (LF);
'\r' or '0x0D' (13 in decimal) -> this one is called "Carriage return" (CR).)

The response message consists of the following:

- A **status line** which includes the status code and reason message (e.g., HTTP/1.1 200 OK, which indicates that the client's request succeeded).
- **Response header fields** (e.g., Content-Type: text/html).
- **An empty line.**
- An **optional message body**.

The status line and other response header fields must all end with $< CR >< LF >$. The empty line must consist of only $< CR >< LF >$and no other whitespace. This strict requirement for $< CR >< LF >$ is relaxed somewhat within message bodies for consistent use of other system linebreaks such as $< CR >$ or $< LF >$ alone.

The message body is in JSON format. After receiving response message, you need to extract message body firstly, then parse this JSON body to find weather description (clear, light rain etc.) and light up LEDs accordingly.

And example of the API calls querying the weather in Hong Kong: (You should change the host to the one we provided)

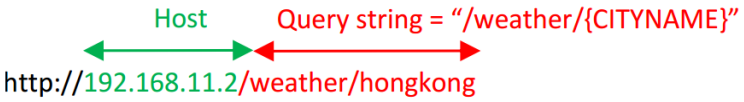

Figure 4: API example

If the weather server successfully processes your request, you will get an HTTP response with status code 200. Then the following string (in a JSON format) will be returned in the **message body** of **the HTTP response**.

You need to find the weather description (e.g., clouds or scattered clouds) by parsing the string and light up LEDs accordingly.

IMPORTANT:

The structure of the HTTP response is provieded at https://openweathermap.org.current, also some examples can be found there. Please go through the website carefully.

3.3 Arduino Programming

Below shows the program skeleton to send HTTP request and parse the response. Download the code from Piazza. You're free to modify the skeleton code as long as it functions correctly.

You are free to design notification (LED's state for different kinds of weather).

Attention: The SRAM of our Arduino board is only 2K. Please write your program wisely or it may not work.

In the code, LED = 5 is for debug, which will be explained in the next part. Please read the debug part before you start to write the program.

```
1   include <ArduinoJson.h>
2
3   // Use an LED connected to PWM 5 as an indicator
4   int LED = 5;
5
6   // Set the LED
7   // TODO: Add your code
8
9
10  // Construct Query String cmd
11  // TODO: Add your code
12  String cmd = ...;
13
14  void setup(){
15  // Configure the Serial Port to ESP8266
16  // Set the baud rate to 9600bps
17  Serial.begin(9600);
18  pinMode(LED, OUTPUT);
```

```
19
20          // Initial LED pins
21          // TODO: Add your code
22
23
24          }
25
26          void loop(){
27
28          digitalWrite(LED, HIGH);
29          delay(5000);
30
31          // Clear up the receive buffer before sending
32          while(Serial.available()) {
33          Serial.read();
34          }
35
36          // Send HTTP request
37          Serial.println(cmd);
38
39          // Receive start
40          digitalWrite(LED, LOW);
41
42          // Initialize LEDs state
43          // TODO: Add your code
44
45
46          // Receive the HTTP response q_str
47          int timeout = 3000;
48          long start=millis();
49          char tempAChar;
50          String q_str;
51          while((millis()-start)<timeout){
52          if(Serial.available() > 0){
53          tempAChar = Serial.read();
54          q_str += tempAChar;
55          }
56          }
57          // Receive End
58
59          delay(2000);
60          //Serial.println("Received: ");
61          //Serial.println(q_str.length(), DEC);
62          //Serial.println(q_str);
63
64          // Extract message body (Json format) from response message
65          // Message body should be stored in a char array for later parse
66          // Please name this char array char_array here
```

```
67          // TODO: Add your code
68
69
70          // Parse Json
71          StaticJsonBuffer<400> jsonBuffer;
72          //Serial.println(char_array);
73          JsonObject& root = jsonBuffer.parseObject(char_array);
74          if (!root.success()) {
75          //Serial.println("parseObject() failed");
76          }else{
77          //Serial.println("suc");
78          const char* name = root["city"];
79          //Serial.println(name);
80
81          // Get weather information
82          // TODO: Add your code
83
84
85          // Light up LEDs accordingly
86          // TODO: Add your code
87
88
89          }
90          // Send next request after some time
91          delay(10000);
92          }
```

3.4 Uploading and Debug of Arduino Program

To watch the status of Arduino serial port, choose "Tools" and "Serial Monitor". You can use serial.println() to print some intermediate results about your received message to check whether the weather notifier works properly.

Attention: If switch 1 is open, anything print by **Serial.print()** or **Serial.println()**will be written to ESP8266 WIFI shield, then be sent to the server and your weather station may not work properly. Therefore, before you print any intermediate results, you need to shut down switch 1 to disable writing. LED 5 is used for debug. It lights up 5 seconds before Arduino sends HTTP request to ESP8266 and goes out after. Therefore, during debug, turn on switch 1 when LED 5 lights up and turn it off when LED 5 goes out. After your program can function properly, comment all these serial print commands except **Serial.println(cmd)**.

3.5 Wiring

Wire LEDs with Arduino and WIFI module to make notifications.

4 Accessment

- **Basic requirement:** Design at least one notification pattern (LEDs' state for different kinds of weather) in class and demo to TA.

- **Combining with OLED display:** You have 2 weeks to design a more friendly notification pattern using OLED we used in Lab2. Also, you should write a report to simply explain the work you do (within 2 A4 pages).You will get higher marks with a good design. The details will released on Piazza soon.