Number Display

Frank Li

July 2021

1 Introduction

In Figure 4, it is a small electronic device that will display a number of your choosing. In order to display a number, the user has to first log into the ESP8266 local server, and from there, user can input a number to the electronic device, increments or decrements that number, or display the last digit of bitcoin at that moment. This device has both hardware and software component, and the details about each respective design will be given below.

2 Motivation

This project is a way for me to implement what I have learned in electrical engineering and computer science, and put them together to achieve an embedded system. Since the schools went remote all an entire year, there has been little or no in-person lab components of my engineering classes. I believe it is essential for me, as a future engineer, to have as many hands-on experiences as I can. And this project is a perfect way for me to practice my circuit design, soldering, and many more.

3 First Design

In Figure 1, it is the first iteration of the project. In this iteration, it can only display the last digit of bitcoin. The functions and design process will be explained below.

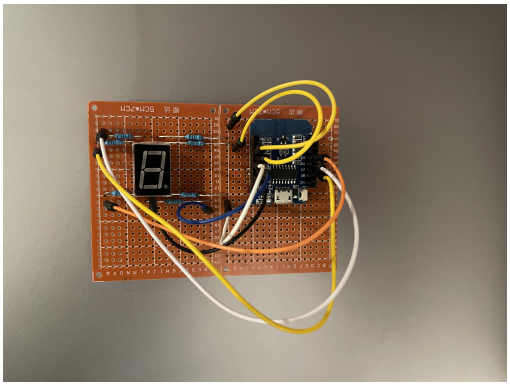

Figure 1: First Design

3.1 Hardware

The hardware part of the Number Display is fairly straightforward, it concerned about two simple concepts in electrical engineering: voltage divider and Ohms Law. Let's start with Ohms Law:

$$V = I * R$$

Ohms Law describes the relationship between current, resistance and voltage. If we have two elements known, then we can find the third one by using this equation. As for voltage divider, it simply states that when there are two resistors in series, the voltage of each component is proportional to each resistance.

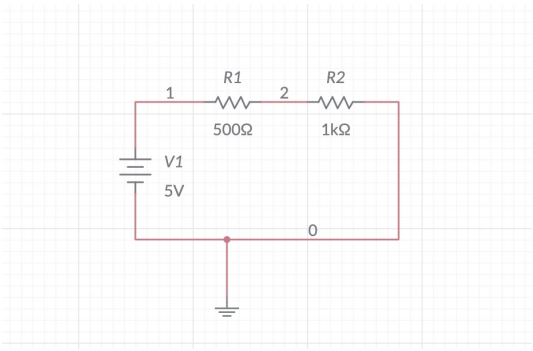

Figure 2: Voltage Division

As shown in Figure 2, the input voltage is 5v, and since the resistors are in series, we can use Ohms Law and Voltage Division to calculate the voltage across each resistor. If we want to designate 1k resistor as Output Voltage, R2, and 500 resistor as R1, then it can be written as:

$$V_{out} = \frac{V_{in}}{R1 + R2} * R2$$

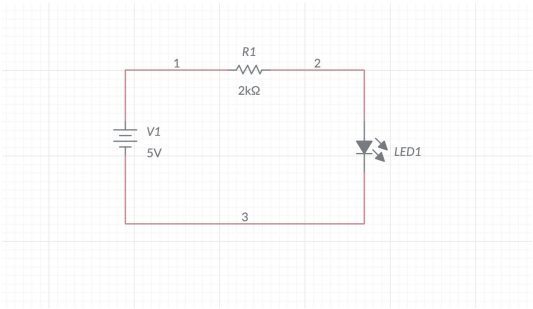

Figure 3: Simple LED Circuit

These two concepts are important in designing the circuits because if there is no resistor between the input voltage and the LED, then the LED will be burned. However, in order to figure out how much voltage should be in LED, we need to look into the Data sheet for the typical voltage in the LED and then choose a good resistance for the voltage division. And in my case, either 1k or 2k resistor will do the work. In Figure 3, a 2k resistor will do the work.

3.1.1 Electronics

The electronics involved in this projects are resistors, a seven segment display, and a ESP8266. As shown in Figure 3, resistor is used for voltage division. Since seven segment display is consists of many LED, Figure 3 is an illustration of the connection to one of the seven LED. The seven segment display I am using is Common Cathode, therefore COM pin should be connected to the ground. As for ESP8266, it is the WiFi module that will be able to connect to a local WiFi network. In my case, it also serves as a micro controller to manage the outputs to the seven segment display.

3.1.2 Assembly

The assembly process is basically soldering, however, it would be extremely hard to describe the soldering process in this paper. Therefore, I can only describe a few key things during the assembly. The first being

that soldering has safety concerns. In all times, the user should wear a safety goggles, and be absolutely careful to not have any contact with the iron for it is burning hot. Secondly, For the first design, I have made the mistake by soldering the jumper wires onto the perfboard. Being this is the first time, I have doing anything like this, I think it is understandable why I would make such mistake. And finally, be sure to not creating short circuit during the soldering process(e.g. do not put solder into places that will mess up the original circuit).

3.2 Code

Here is the software part of the project, I am thankful for Brian Lough[1] for putting up a sample code of using ESP8266 to make API request. This code, written in arduino C, is first going to connect to the local WIFI network, then it is going to the targeted website, cryptonator.com, to fetch the bitcoin data. Then with the get request, it will use the arduino Json library to parse the get request and display the bitcoin last digit onto the seven segment display.

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>

// ----------------------------
// Additional Libraries - each one of these will need to be installed.
// ----------------------------

#include <ArduinoJson.h>
// Library used for parsing Json from the API responses

// Search for "Arduino Json" in the Arduino Library manager
// https://github.com/bblanchon/ArduinoJson

//------- Replace the following! ------
char ssid[] = "********";       // your network SSID (name)
char password[] = "**********";  // your network key

// For Non-HTTPS requests
  //WiFiClient client;

// For HTTPS requests
WiFiClientSecure client;


// Just the base of the URL you want to connect to
#define TEST_HOST "api.cryptonator.com"

// OPTIONAL - The finferprint of the site you want to connect to.
#define TEST_HOST_FINGERPRINT "10 76 19 6B E9 E5 87 5A 26 12 15 DE 9F 7D 3B 92 9A 7F 30 13"
// The finger print will change every few months.


void setup() {

  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(0, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(14, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  Serial.begin(115200);

  // Connect to the WiFI
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  // Attempt to connect to Wifi network:
  Serial.print("Connecting Wifi: ");
```

```
  Serial.println(ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  IPAddress ip = WiFi.localIP();
  Serial.println(ip);

  //--------

  // If you don't need to check the fingerprint
  // client.setInsecure();

  // If you want to check the fingerprint
  client.setFingerprint(TEST_HOST_FINGERPRINT);


}

void makeHTTPRequest() {

  // Opening connection to server (Use 80 as port if HTTP)
  if (!client.connect(TEST_HOST, 443))
  {
    Serial.println(F("Connection failed"));
    return;
  }

  // give the esp a breather
  yield();

  // Send HTTP request
  client.print(F("GET "));
  // This is the second half of a request (everything that comes after the base URL)
  client.print("/api/ticker/btc-usd"); // %2C == ,
  client.println(F(" HTTP/1.1"));

  //Headers
  client.print(F("Host: "));
  client.println(TEST_HOST);

  client.println(F("Cache-Control: no-cache"));

  if (client.println() == 0)
  {
    Serial.println(F("Failed to send request"));
    return;
  }
  //delay(100);
  // Check HTTP status
  char status[32] = {0};
  client.readBytesUntil('\r', status, sizeof(status));
  if (strcmp(status, "HTTP/1.1 200 OK") != 0)
  {
    Serial.print(F("Unexpected response: "));
    Serial.println(status);
    return;
  }

  // Skip HTTP headers
  char endOfHeaders[] = "\r\n\r\n";
  if (!client.find(endOfHeaders))
  {
    Serial.println(F("Invalid response"));
```

```
    return;
  }

  // This is probably not needed for most, but I had issues
  // with the Tindie api where sometimes there were random
  // characters coming back before the body of the response.
  // This will cause no hard to leave it in
  // peek() will look at the character, but not take it off the queue
  while (client.available() && client.peek() != '{')
  {
    char c = 0;
    client.readBytes(&c, 1);
    Serial.print(c);
    Serial.println("BAD");
  }

  //  // While the client is still availble read each
  //  // byte and print to the serial monitor
  //  while (client.available()) {
  //    char c = 0;
  //    client.readBytes(&c, 1);
  //    Serial.print(c);
  //  }

  int valid = bitCoinPrice();
  if (valid < 0){
    return;
  }else{
    Serial.print("Bitcoin most significant digit: ");
    Serial.println(valid);
    //5 to A, 4 to B and 0 to C 2 to D 14 to E 12 to F 13 to G
    switch(valid){
    case 0:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(14, HIGH);
      digitalWrite(12, HIGH);
      break;
    case 1:
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      break;
    case 2:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(14, HIGH);
      digitalWrite(13, HIGH);
      break;
    case 3:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(13, HIGH);
      Serial.println("Good");
      break;
    case 4:
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(12, HIGH);
      digitalWrite(13, HIGH);
      break;
    case 5:
      digitalWrite(5, HIGH);
      digitalWrite(0, HIGH);
```

```
      digitalWrite(2, HIGH);
      digitalWrite(12, HIGH);
      digitalWrite(13, HIGH);
      break;
    case 6:
      digitalWrite(5, HIGH);
      digitalWrite(13, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(14, HIGH);
      digitalWrite(12, HIGH);
      break;
    case 7:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      break;
    case 8:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(14, HIGH);
      digitalWrite(12, HIGH);
      digitalWrite(13, HIGH);
      break;
    case 9:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(12, HIGH);
      digitalWrite(13, HIGH);
      break;
    }
  }

}

int bitCoinPrice() {
  // Stream& input;

  DynamicJsonDocument doc(384);

  DeserializationError error = deserializeJson(doc, client);

  if (error) {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.f_str());
    return -1;
  }

  JsonObject ticker = doc["ticker"];
  const char* price = ticker["price"]; // "33908.95216874"
  char temp = price[strlen(price) - 10];
  int bitPrice = temp - '0';
  return bitPrice;
}
void loop() {
  // put your main code here, to run repeatedly:
  makeHTTPRequest();
  delay(2000);
}
```

4 Second Design

As shown in Figure 1, the second iteration is cleaner and I did not solder any jumper wire this time. The hardware aspects are the same, but there are more functions in the software part.

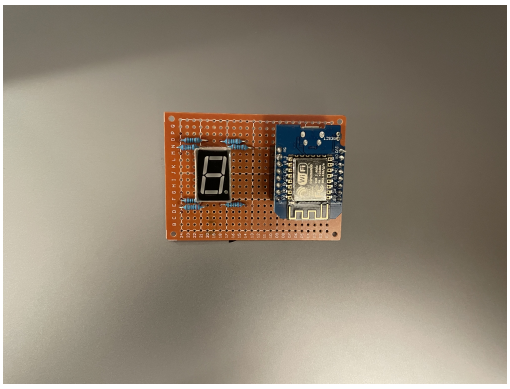

Figure 4: Second Design

4.1 Hardware

The hardware part is exactly the same as the first design. The two key concepts in the second iteration are still Ohms Law and Voltage Divider.

4.1.1 Electronics

The electronic part is exactly the same as the first design. The components used are still resistors, seven segment display and ESP8266.

4.1.2 Assembly

The assembly process in the second iteration is very similar to the first design except I did not use jumper wires this time. All the connections are on the backside. As shown in Figure 5, I used 22 gauge wire to make all the connections. One thing to be careful is that, do not let wire touch each other because it will create a short circuit and mess up the whole system.

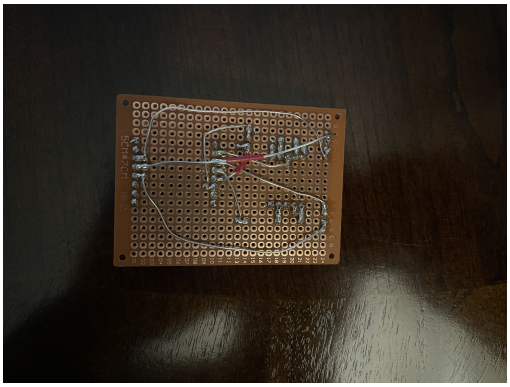

Figure 5: Second Design Soldering

4.2 Code

The second iteration adds a few features compare to the first design. First, the ESP8266 is a server, and user can log into this server and interact with the system. Second, the ESP8266 will not only show the last digit of bitcoin, it can also allow user to input a number in its local website, and user can increments or decrements that number. I would like to thank circuit4you.com [2] for providing me a sample of handling HTML forms and input information on ESP8266.

```
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <WiFiClientSecure.h>

// ----------------------------
// Additional Libraries - each one of these will need to be installed.
// ----------------------------

#include <ArduinoJson.h>
// Library used for parsing Json from the API responses

// Search for "Arduino Json" in the Arduino Library manager
// https://github.com/bblanchon/ArduinoJson

//------- Replace the following! ------
char ssid[] = "*********";       // your network SSID (name)
char password[] = "***********";  // your network key
// For Non-HTTPS requests
  //WiFiClient client;

// For HTTPS requests
WiFiClientSecure client;


// Just the base of the URL you want to connect to
#define TEST_HOST "api.cryptonator.com"

// OPTIONAL - The finferprint of the site you want to connect to.
#define TEST_HOST_FINGERPRINT "10 76 19 6B E9 E5 87 5A 26 12 15 DE 9F 7D 3B 92 9A 7F 30 13"
// The finger print will change every few months.

const char MAIN_page[] PROGMEM = R"=====(
<!DOCTYPE html>
<html>
<title>Number Display</title>
<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align:
    center;}
  body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;
    margin-bottom: 50px;}
  .button {display: block;width: 80px;background-color: #1abc9c;border: none;color: white;
    padding: 13px 30px;text-decoration: none;font-size: 25px;margin: 0px auto 35px;cursor:
    pointer;border-radius: 4px;}
  .button:active {background-color: #16a085;}
  p {font-size: 14px;color: #888;margin-bottom: 10px;}
</style>
<body>

<h2>A Number Display<h2>
<h3> By Frank Li</h3>

<form action="/input_page">
  <label for="number">Enter a Number between 0-9:</label>
  <input type="number" id="number" name="number" min="0" max="9">
  <input type="submit" value="Submit">
</form>

<p>Increment Button</p><a class="button" href="increment">Increment</a>
<p>Decrement Button</p><a class="button" href="decrement">Decrement</a>

```

```
<p>Display Bitcoin Last Digit</p><a class="button" href="bitcoin">Bitcoin</a>

</body>
</html>
)=====";

ESP8266WebServer server(80);
int globalNum;

void handleRoot() {
  String s = MAIN_page;
  server.send(200, "text/html", s);
}

void handleForm() {
  int number = getNumber();
  globalNum = number;
  String s = MAIN_page;
  server.send(200, "text/html", s);
  chooseNumber(number);
}

void handleIncrement() {
  String s = MAIN_page;
  int input = globalNum;
  Serial.println(input);
  int number = (input + 1)%10;
  globalNum = number;
  server.send(200, "text/html", s);
  chooseNumber(number);
}

void handleDecrement() {
  String s = MAIN_page;
  int input = globalNum;
  Serial.println(input);
  int number = ((input - 1)%10);
  if(number < 0) number=number+10;
  globalNum = number;
  server.send(200, "text/html", s);
  chooseNumber(number);
}

void handleBitcoin() {
  String s = MAIN_page;
  makeHTTPRequest();
  server.send(200, "text/html", s);
}
void setup() {

  pinMode(5, OUTPUT);
  pinMode(4, OUTPUT);
  pinMode(0, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(14, OUTPUT);
  pinMode(12, OUTPUT);
  pinMode(13, OUTPUT);
  Serial.begin(115200);

  // Connect to the WiFI
  WiFi.mode(WIFI_STA);
  WiFi.disconnect();
  delay(100);

  // Attempt to connect to Wifi network:
  Serial.print("Connecting Wifi: ");
  Serial.println(ssid);
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  IPAddress ip = WiFi.localIP();
  Serial.println(ip);

  //--------

  // If you don't need to check the fingerprint
  // client.setInsecure();

  // If you want to check the fingerprint
  client.setFingerprint(TEST_HOST_FINGERPRINT);
  server.on("/", handleRoot);
  server.on("/input_page", handleForm);
  server.on("/increment", handleIncrement);
  server.on("/decrement", handleDecrement);
  server.on("/bitcoin", handleBitcoin);
  server.begin();
  Serial.println("HTTP started");
}

void makeHTTPRequest() {

  // Opening connection to server (Use 80 as port if HTTP)
  if (!client.connect(TEST_HOST, 443))
  {
    Serial.println(F("Connection failed"));
    return;
  }

  // give the esp a breather
  yield();

  // Send HTTP request
  client.print(F("GET "));
  // This is the second half of a request (everything that comes after the base URL)
  client.print("/api/ticker/btc-usd"); // %2C == ,
  client.println(F(" HTTP/1.1"));

  //Headers
  client.print(F("Host: "));
  client.println(TEST_HOST);

  client.println(F("Cache-Control: no-cache"));

  if (client.println() == 0)
  {
    Serial.println(F("Failed to send request"));
    return;
  }
  //delay(100);
  // Check HTTP status
  char status[32] = {0};
  client.readBytesUntil('\r', status, sizeof(status));
  if (strcmp(status, "HTTP/1.1 200 OK") != 0)
  {
    Serial.print(F("Unexpected response: "));
    Serial.println(status);
    return;
  }

  // Skip HTTP headers
  char endOfHeaders[] = "\r\n\r\n";
```

```
  if (!client.find(endOfHeaders))
  {
    Serial.println(F("Invalid response"));
    return;
  }

  // This is probably not needed for most, but I had issues
  // with the Tindie api where sometimes there were random
  // characters coming back before the body of the response.
  // This will cause no hard to leave it in
  // peek() will look at the character, but not take it off the queue
  while (client.available() && client.peek() != '{')
  {
    char c = 0;
    client.readBytes(&c, 1);
    Serial.print(c);
    Serial.println("BAD");
  }

  //  // While the client is still availble read each
  //  // byte and print to the serial monitor
  //  while (client.available()) {
  //    char c = 0;
  //    client.readBytes(&c, 1);
  //    Serial.print(c);
  //  }

  int valid = bitcoinPrice();
  if (valid < 0){
    return;
  }else{
    Serial.print("Bitcoin most significant digit: ");
    Serial.println(valid);
    //5 to A, 4 to B and 0 to C 2 to D 14 to E 12 to F 13 to G
    chooseNumber(valid);
  }
}
void clear() {
  digitalWrite(5, LOW);
  digitalWrite(4, LOW);
  digitalWrite(0, LOW);
  digitalWrite(2, LOW);
  digitalWrite(12, LOW);
  digitalWrite(13, LOW);
  digitalWrite(14, LOW);
}
void chooseNumber(int input) {
  Serial.println(input);
  clear();
  switch(input){
    case 0:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(14, HIGH);
      digitalWrite(12, HIGH);
      //Serial.print(0);
      break;
    case 1:
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      //Serial.print(1);
      break;
    case 2:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(2, HIGH);
```

```
      digitalWrite(14, HIGH);
      digitalWrite(13, HIGH);
      //Serial.print(2);
      break;
    case 3:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(13, HIGH);
      //Serial.print(3);
      break;
    case 4:
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(12, HIGH);
      digitalWrite(13, HIGH);
      //Serial.print(4);
      break;
    case 5:
      digitalWrite(5, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(12, HIGH);
      digitalWrite(13, HIGH);
      //Serial.print(5);
      break;
    case 6:
      digitalWrite(5, HIGH);
      digitalWrite(13, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(14, HIGH);
      digitalWrite(12, HIGH);
      //Serial.print(6);
      break;
    case 7:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      //Serial.print(7);
      break;
    case 8:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(14, HIGH);
      digitalWrite(12, HIGH);
      digitalWrite(13, HIGH);
      //Serial.print(8);
      break;
    case 9:
      digitalWrite(5, HIGH);
      digitalWrite(4, HIGH);
      digitalWrite(0, HIGH);
      digitalWrite(2, HIGH);
      digitalWrite(12, HIGH);
      digitalWrite(13, HIGH);
      //Serial.print(9);
      break;
    }
  }

int getNumber() {
  String inputNumber = server.arg("number");
  char temp = inputNumber[0];
  int number = temp - '0';
```

```
  return number;
}
int bitcoinPrice() {
  // Stream& input;

  DynamicJsonDocument doc(384);

  DeserializationError error = deserializeJson(doc, client);

  if (error) {
    Serial.print(F("deserializeJson() failed: "));
    Serial.println(error.f_str());
    return -1;
  }

  JsonObject ticker = doc["ticker"];
  const char* price = ticker["price"]; // "33908.95216874"
  char temp = price[strlen(price) - 10];
  int bitPrice = temp - '0';
  return bitPrice;
}
void loop() {
  // put your main code here, to run repeatedly:
  //makeHTTPRequest();
  server.handleClient();
}
```

5 Conclusion

This project is a great way to put together what I have learned in computer science and electrical engineering and create an embedded system. Throughout the project, I became better at soldering, writing code and writing out circuit diagrams. I also have the chance to learn about ESP8266 and its many great libraries. In the end, it might be the first project that I have implemented both software and hardware, but it is stepping a stone for something larger later on.

6 References

[1] Brian Lough, May 24, 2021, arduino-sample-api-request,
`https://github.com/witnessmenow/arduino-sample-api-request`

[2] Circuit4you.com, March 20, 2019,
`https://circuits4you.com/2019/03/20/esp8266-receive-post-get-request-data-from-website/`