CSCE 432/4301: Embedded Systems

Spring 2020

Semester Project Report

Simple IoT Application

Dr. Mohamed Shalan

Maram Abbas 900153570

Table of Contents

Project Description

Aim

This project's aim is to utilize the ESP8266 module to create a small IoT application that can enable the user to perform I/O operations with the STM32 module through a web interface.

The I/O operations include:

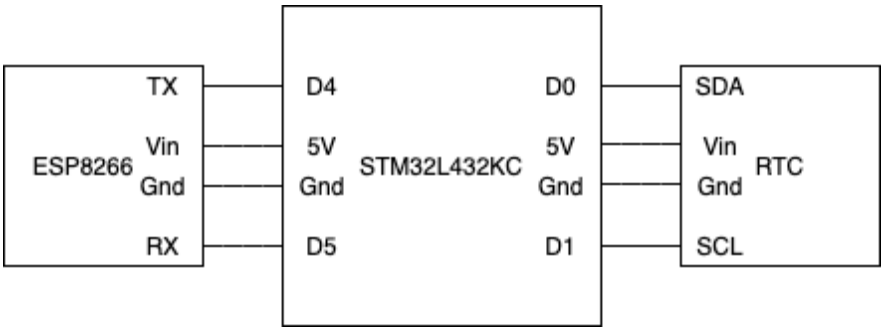
- Retrieving date and time from RTC module connected to the STM32 module
- Control the STM32 LEDs status

System Architecture

This project follows the client-server architecture. The ESP8266 module is known to run the HTTP server and the web page that the user will be using will act as the client. Client requests may be as follows:

- Turn on LED on STM32 module
- Turn off LED on STM32 module
- Send real date from RTC module to STM32 module and then to ESP8266 module
- Send real time from RTC module to STM32 module and then to ESP8266 module

Block Diagram



Communication Explained

Here is a table explaining how the STM32 module is communicating with the ESP8266 module:

ESP8266 Module	STM32 Module
Sends '1' and waits to receive date/time	Receives '1' and sends back date/time
Sends '2'	Receives '2' and turns on LED
Sends '3'	Receives '1' and turns off LED

The codes of the STM32 module and ESP8266 module are totally based on the table above. The mentioned characters of '1', '2', and '3' are the major characters being sent and received between both modules and based on them, the desired output is shown on the webpage (date and time) or on the STM32 module (LED turning on or off).

STM32 Code

Setting RTC Module Registers

ADDRESS	BIT 7 MSB	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0 LSB	FUNCTION	RANGE
00h	0	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12/$\overline{24}$	$\overline{AM}$/PM 20 Hour	10 Hour	Hour				Hours	1–12 + $\overline{AM}$/PM 00–23
03h	0	0	0	0	0	Day			Day	1–7
04h	0	0	10 Date		Date				Date	01–31
05h	Century	0	0	10 Month	Month				Month/ Century	01–12 + Century
06h	10 Year				Year				Year	00–99

First of all, in order to retrieve the date and time from the RTC module, the clock needs to be set initially. Each of these registers, with addresses 00h to 06h, need to be set with an initial value. The figure above is taken from the RTC module (DS3231) datasheet which explains how each of the registers is formatted. In order to do so, some assumptions were made:

- Days of the week were numbered as follows:

Week Day	Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
It's assumed representation in the DAY REGISTER	1	2	3	4	5	6	7

- Hour register uses the 24 hour timing and not the AM/PM timing. Hence, in the register address 02h, bit 6 is always 0.

```
// seconds
secbuffer[0] = 0x00; //register address
secbuffer[1] = 0x00; //data to put in register --> 0 sec

HAL_I2C_Master_Transmit(&hi2c1, 0xD0, secbuffer, 2, 10);

// minutes
minbuffer[0] = 0x01; //register address
minbuffer[1] = 0x00; //data to put in register --> 00 min

HAL_I2C_Master_Transmit(&hi2c1, 0xD0, minbuffer, 2, 10);
```

The above sample of code shows how the RTC registers are set. The function ***HAL_I2C_Master_Transmit()*** is used to transmit the register data and set it.

Getting Data from RTC Module Registers

```
HAL_I2C_Master_Transmit(&hi2c1, 0xD0, yearbuffer, 1, 10);
HAL_I2C_Master_Receive(&hi2c1, 0xD1, yearbuffer+1, 1, 10);

out[9] = hexToAscii(yearbuffer[1] >> 4 );
out[10] = hexToAscii(yearbuffer[1] & 0x0F);

HAL_I2C_Master_Transmit(&hi2c1, 0xD0, monthbuffer, 1, 10);
HAL_I2C_Master_Receive(&hi2c1, 0xD1, monthbuffer+1, 1, 10);

out[6] = hexToAscii(monthbuffer[1] >> 4 );
out[7] = hexToAscii(monthbuffer[1] & 0x0F);
```

In the above code sample, it is shown how the RTC module registers are read. ***HAL_I2C_Master_Transmit()*** is used to specify which register exactly we will be reading from. ***HAL_I2C_Master_Receive()*** receives the exact value in the specified register. The array named ***out[]*** is used to store the read data. Since the registers store the data in hex values, they are needed to be converted before adding them to the ***out[]*** array. The ***out[]*** array then takes this format:

```
out[] = {0,0,',',0,0,' ',0,0,',',0,0,' ',0,0,':',0,0,':',0,0,'\r','\n'};
```

Or to be clearer, the table below explains it better:

0	Day, tens digit
0	Day, ones digit
‘,’	Comma separating day and date
0	Date, tens digit
0	Date, ones digit
‘ ‘	Space separating date and month
0	Month, tens digit
0	Month, ones digit
‘,’	Comma separating month and year
0	Year, tens digit
0	Year, ones digit
‘ ‘	Space between year and time
0	Hour, tens digit
0	Hour, ones digit
‘:’	Separator between hour and minute
0	Minute, tens digit
0	Minute, ones digit
‘:’	Separator between minute and second
0	Second, tens digit
0	Second, ones digit
‘\r’	Carriage return
‘\n’	New line

STM32 Module Handling Requests

Requests coming to the STM32 module may include the following:

- Turn on LED
- Turn off LED
- Send date/time back to ESP8266 module

The code below simply shows how these requests are handled.

```
// transmit/receive to/from UART

HAL_UART_Receive(&huart1,&choice, sizeof(choice),50);

if (choice == '1')  //transmit date/time
{
    HAL_UART_Transmit(&huart1,out, sizeof(out),500);
    choice = '\0';
}

else if (choice == '2')     //led on
{
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3,1);
    HAL_UART_Transmit(&huart2,&choice, sizeof(choice),500);
    choice = '\0';
}

else if (choice == '3')     //led off
{
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3,0);
    HAL_UART_Transmit(&huart2,&choice, sizeof(choice),500);
    choice = '\0';
}
```

The STM32 module receives the request in a variable named ***choice*** and if it contains the character '1', the data and time get transmitted back. If it contains the character '2', the LED on the STM32 module is set to 1 in order to turn on and finally, if it contains the character '3', the LED on the STM32 module is set to 0 in order to turn off.

ESP8266 Code

Some Required Settings

When setting the ESP8266 module in Access Point mode, it will create a WiFi network.

Hence, we need to set its SSID, Password, IP address, IP subnet mask and IP gateway.

```
/* Put your SSID & Password */
const char* ssid = "Embedded Project";  // Enter SSID here
const char* password = "12345678";  //Enter Password here

/* Put IP Address details */
IPAddress local_ip(192,168,1,1);
IPAddress gateway(192,168,1,1);
IPAddress subnet(255,255,255,0);
```

Then, we declare an object of ESP8266WebServer library, so we can access its functions.

The constructor of this object takes port (where the server will be listening to) as a parameter. Since 80 is the default port for HTTP, we will use this value. Now you can access the server without needing to specify the port in the URL.

```
ESP8266WebServer server(80);
```

setup () Function

Serial.begin() specifies the baud rate. Then, we set up a soft access point to establish a Wi-Fi network by proving SSID, Password, IP address, IP subnet mask and IP gateway.

```
void setup() {
  Serial.begin(9600);

  WiFi.softAP(ssid, password);
  WiFi.softAPConfig(local_ip, gateway, subnet);
```

In order to handle incoming HTTP requests, we need to specify which code to execute when a particular URL is hit. To do so, we use ***on*** method. This method takes two parameters. First one is a URL path and second one is the name of function which we want to execute when that URL is hit.

For example, the first line of below code snippet indicates that when a server receives an HTTP request on the root (/) path, it will trigger the ***handle_OnConnect()*** function. Note that the URL specified is a relative path.

```
server.on("/", handle_OnConnect);
server.on("/ledon", handle_ledon);
server.on("/ledoff", handle_ledoff);
server.on("/date_time", handle_datetime);
server.onNotFound(handle_NotFound);
```

Now, to start our server, we call the begin method on the server object.

```
server.begin();
```

loop () Function

To handle the actual incoming HTTP requests, we need to call the ***handle_Client()*** method on the server object.

```
void loop() {
  server.handleClient();
}
```

In order to handle the requests, the following functions were made. The ***Serial.print()*** function is used to transmit the desired request to the STM32 module.

```
void handle_OnConnect() {
  LEDstatus = LOW;
  Serial.print('3');
  server.send(200, "text/html", SendHTML());
  delay(1000);
}

void handle_ledon() {
  LEDstatus = HIGH;
  Serial.print('2');
  server.send(200, "text/html", SendHTML());
  delay(1000);
}

void handle_ledoff() {
  LEDstatus = LOW;
  Serial.print('3');
  server.send(200, "text/html", SendHTML());
  delay(1000);
}

void handle_datetime() {
  dt1 = HIGH;
  dt2 = HIGH;
  server.send(200, "text/html", SendHTML());
  delay(1000);
}

void handle_NotFound(){
  server.send(404, "text/plain", "Not found");
}
```

get_date_time () Function

In this function, a request is sent to the STM32 module in order to get the current date/time. Until the STM32 module replies back, a delay is made. The receiving buffer is then looped on and the date/time is retrieved.

```
String get_date_time()
{
  String date_time = "";

  Serial.print('1');
  delay (1000);
  while(Serial.available() > 0)    //Checks is there any data in buffer
  {
    date_time += char(Serial.read());  //Read serial data byte
  }
  return date_time;
}
```

SendHTML () Function

This function is responsible for generating a web page whenever the ESP8266 web server gets a request from a web client. It merely concatenates HTML code into a big string and returns to the ***server.send()*** function. It also changes some of the webpage information based on the program flags.

```
String SendHTML(){
  String date_time = "";
  if (dt1 == HIGH)
  {
    date_time = get_date_time();
  }

  String ptr = "<!DOCTYPE html> <html>\n";
  ptr +="<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
  ptr +="<title>LED Control</title>\n";
  ptr +="<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center;}\n";
  ptr +="body{margin-top: 50px;} h1 {color: #444444;margin: 50px auto 30px;} h3 {color: #444444;margin-bottom: 50px;}\n";
  ptr +=".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;}\n";
  ptr +=".button-on {background-color: #1abc9c;}\n";
  ptr +=".button-on:active {background-color: #16a085;}\n";
  ptr +=".button-off {background-color: #34495e;}\n";
  ptr +=".button-off:active {background-color: #2c3e50;}\n";
  ptr +="p {font-size: 14px;color: #888;margin-bottom: 10px;}\n";
  ptr +="</style>\n";
  ptr +="</head>\n";
  ptr +="<body>\n";
  ptr +="<h1>Embedded Project</h1>\n";
  ptr +="<h3>Demo</h3>\n";

   if(LEDstatus == HIGH)
  {ptr +="<p>LED Status: ON</p><a class=\"button button-off\" href=\"/ledoff\">OFF</a>\n";}
  else
  {ptr +="<p>LED Status: OFF</p><a class=\"button button-on\" href=\"/ledon\">ON</a>\n";}

  ptr +="<a class=\"button\" href=\"/date_time\">Get Date & Time</a>\n";
```

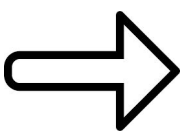
This part of the function shows how the date and time was formatted and displayed.

```
ptr +="<a class=\"button\" href=\"/date_time\">Get Date & Time</a>\n";

if (dt1 == HIGH)
{
  day = date_time.substring(0,2);
  day_month = date_time.substring(3,5);
  month = date_time.substring(6,8);
  year = date_time.substring(9,11);
  hour = date_time.substring(12,14);
  minute = date_time.substring(15,17);
  second = date_time.substring(18,20);

  switch (day[1])
  {
    case '1':
      day = "Sunday";
      break;
    case '2':
      day = "Monday";
      break;
    case '3':
      day = "Tuesday";
      break;
    case '4':
      day = "Wednesday";
      break;
    case '5':
      day = "Thursday";
      break;
    case '6':
      day = "Friday";
      break;
    case '7':
      day = "Saturday";
      break;
  }

  if (month == "01")
  {
    month = "January";
```



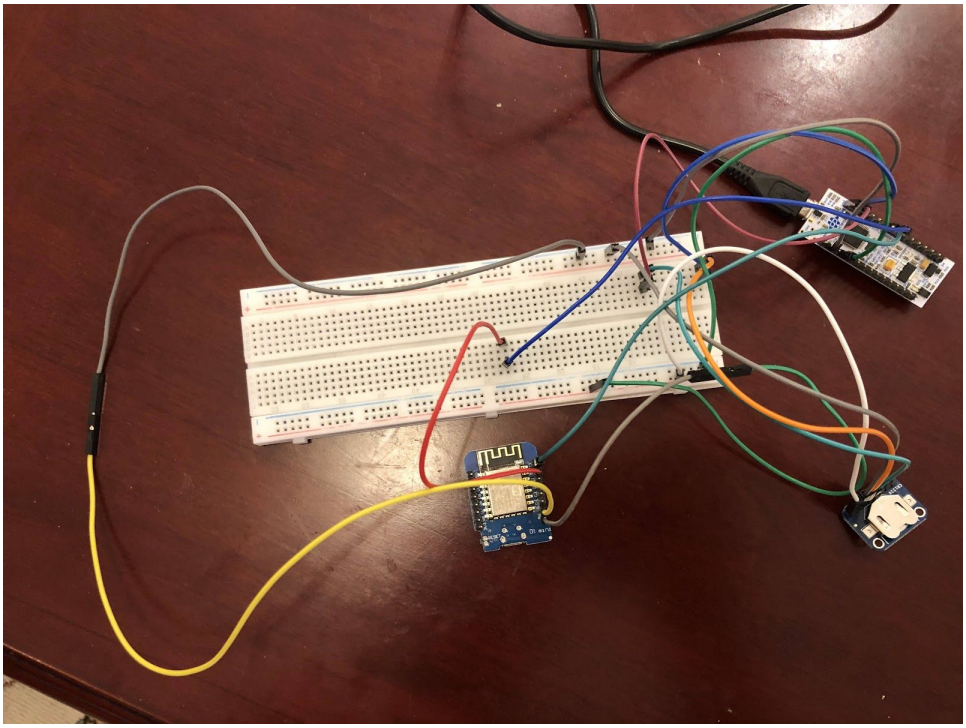
```
if(dt2 == HIGH)
{
  String date = "<p>Date: " + day + ", " + month + " " + day_month + ", 20" + year + "</p
  String timee = "<p>Time: " + hour + ":" + minute + ":" + second + "</p>\n";

  ptr+=date;
  ptr+=timee;
}

ptr +="</body>\n";
ptr +="</html>\n";
return ptr;
```

Demo and Screenshots

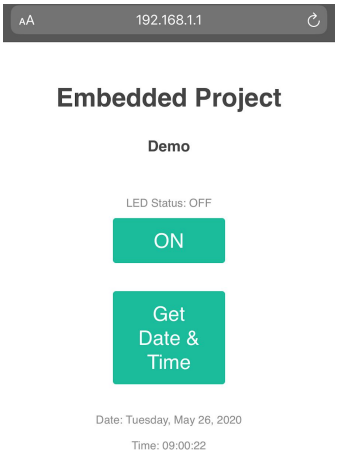
Connections

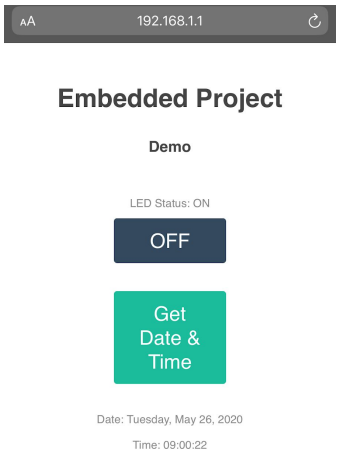


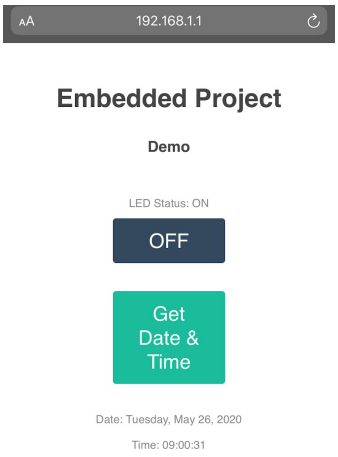
Video Demo Link

https://drive.google.com/file/d/1buJrjO16JRljvoi-9w4ytanVHjY_ZoTu/view?usp=sharing

Some Screenshots







References

RTC module Datasheet:
https://datasheetspdf.com/pdf-file/1081920/MaximIntegrated/DS3231/1

STM32L432KC Datasheet:
https://datasheetspdf.com/pdf-file/1349053/STMicroelectronics/STM32L432KC/1

ESP8266 module Datasheet:
https://www.openimpulse.com/blog/wp-content/uploads/wpsc/downloadables/0A-ESP8266__Datasheet__EN_v4.3.pdf

Arduino code resource:
https://lastminuteengineers.com/creating-esp8266-web-server-arduino-ide/