ECE40862: Software for Embedded Systems

Fall 2021

Lab 4 – HTTP Web Server and Client using ESP32

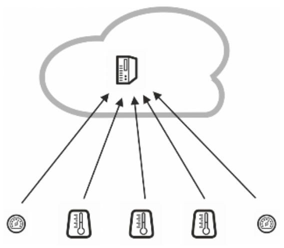
To be done individually; Due by 11:59pm, Friday, October 22, 2021.

1. Overview

The goal of this assignment is to familiarize you with **Internet of Things (IoT)**. The IoT is a network of *'smart'* embedded devices that connect and communicate via the Internet. The key to IoT is the *interconnectivity* of devices, which collect and exchange information through embedded software, cameras, and sensors which sense things like light, sound, distance, movement, temperature, and other environmental parameters. Smart embedded devices either operate autonomously or are controlled and monitored remotely. In this assignment, you will be using your ESP32 as a *'smart'* embedded device operating automatically as well as communicating with the Internet.

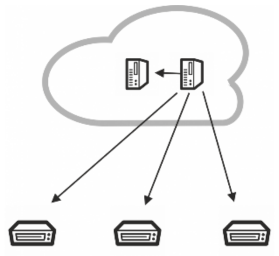
The assignment is subdivided into two sections. In both sections, you will be implementing **client-server communication** using **socket APIs** and **HTTP** protocol. To review, network socket APIs are used to transfer *messages* between two software processes over a network. HTTP is an application layer protocol that defines different parameters of these *messages*, viz., type, format, length, authentication, error handling, etc.

1.1. Device as Client



A typical IoT system has a cloud server and multiple client devices. Data flows from the IoT-devices to the cloud-server, as shown in this image. In the first section, you will configure the ESP32 as an HTTP client, measure the onboard *temperature sensor data* and *hall sensor data*, and SEND them **periodically** to an IoT Application Platform hosted on the internet. Specifically, you will be using the ThingSpeak IoT Platform as the server, collecting sensor data transmitted by the ESP.

1.2. Device as Server



In some scenarios/configuration, e.g., when you have a single IoT device and a local PC or a phone, the IoT device can also be setup as the server. In the second section of the assignment, you will configure the ESP32 as a HTTP web server, measure the onboard *temperature* *sensor data* and *hall sensor data*, monitor the *status of different GPIO pins*, and **SEND** these data to the client ONLY WHEN THE CLIENT REQUESTS. You will be using your local PC or phone's web browser (e.g., Chrome, Firefox, etc.) as the client. Additionally, you will also **RECEIVE** inputs from the client (i.e., the web browser) and control GPIO pins on the board based on these inputs.

NOTE: ESP as client (1.1) only SENDS data to cloud server. ESP as server (1.2) both SENDS data to client and RECEIVES data from client.

2. Programming Exercises

2.1. Hardware Interfacing

Interface the following components to the board (you've done this step many times now, so this should be super simple by now!)

- Two external **LED**s (red and green) as GPIO outputs.

2.2. Software Implementation - ESP32 HTTP Client

2.2.1. Setup ThingSpeak

ThingSpeak is a free IoT Platform for gathering, analyzing, and visualizing data from IoT devices. ThingSpeak allows you to publish your sensor readings to their website and display them in a plot with time stamps. It provides a RESTful API for IoT devices (don't worry if you don't know what that means, it's not important for this lab). You need to perform the following steps to configure it as your cloud server before you can start sending data from the ESP32.

- Create a free account on the ThingSpeak website.
- Create a new channel and enable two fields to receive data from the ESP32.
  - Field1: Temperature Sensor
  - Field2: Hall Sensor

You'll get an API key for posting data to your channel. This API key should be used for sending data to ThingSpeak from your ESP32.

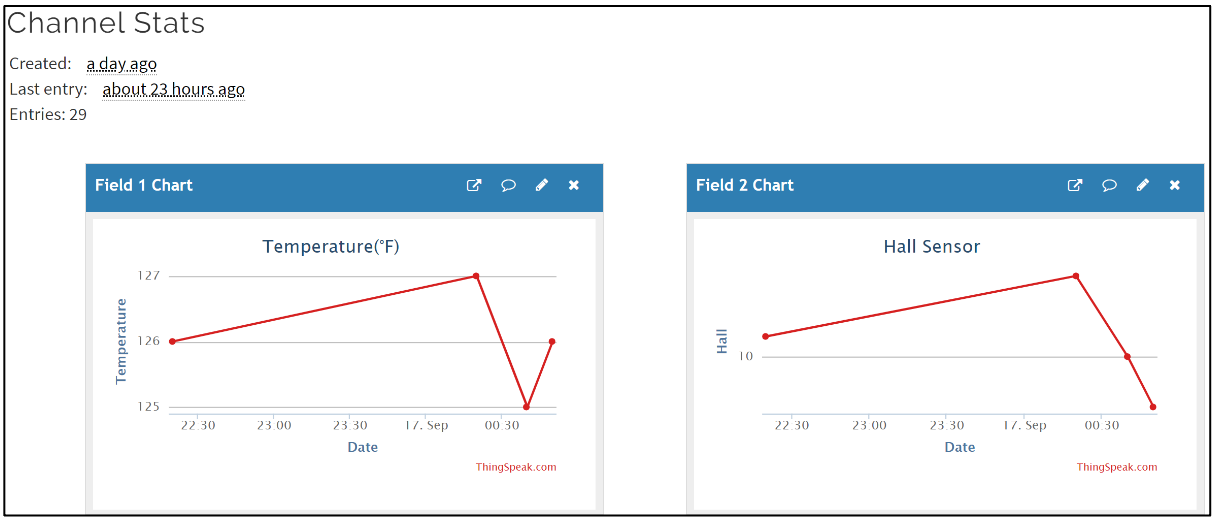
2.2.2. ESP32 Program (espclient.py)

Implement the following functionalities in your program.

- Connect to Internet and print out the local IP address (same as Lab 3).
- Initialize a **hardware timer** with a period of **16 seconds**. Perform the following operations:
  - Measure the onboard *temperature sensor data* and *hall sensor data.*
  - Display both the measured data on the terminal.
  - **SEND** these data to ThingSpeak cloud server using **socket API** and **HTTP GET request**. You need to use your specific Write API Key to upload data to your channel in your ThingSpeak account.
- Run your program for 5 minutes.

2.2.3. Sample Result

Your ThingSpeak account should show the visualizations for both sensor data, as shown in Fig. 1. Since, the timer fires every 16 seconds, a maximum of **18 data points (for each sensor)** will be uploaded by your client over a duration of 5 minutes. Fig. 1 only shows a sample screenshot from the ThingSpeak website showing only 4 data points. **Run your program for 5 minutes**. After the program has finished, take a screenshot (similar to Fig. 1) from the ThingSpeak website showing ~18 data points.


Channel Stats

Created: a day ago

Last entry: about 23 hours ago

Entries: 29

Figure 1. Screenshot from ThingSpeak showing Temperature and Hall Senor data

2.3. Software Implementation - ESP32 HTTP Server

2.3.1. ESP32 Program (espserver.py)

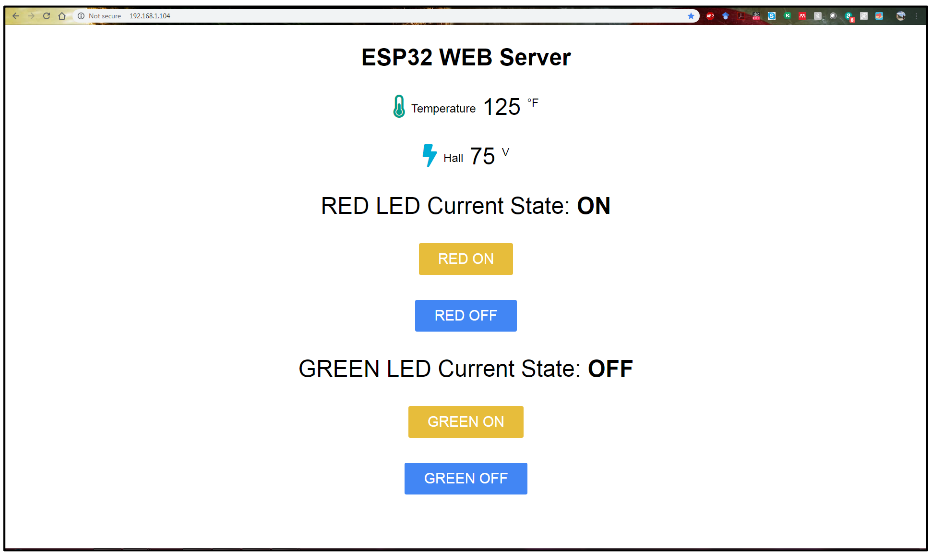
Implement the following functionalities in your program. You can update the **espserver.py** file which has been provided to you.

- Connect to Internet and print out the local IP address (same as Lab 3 Section 2.2.1).
- Measure the onboard *temperature sensor data* and *hall sensor data.*
- Measure the values of different GPIO pins, viz., *two LEDs.*
- Use the function web_page (present in the provided file) to create simple HTML text to build the webpage with 2 sensor data and 2 LED pin values. *The HTML text has been provided in the* ***espserver.py***. You can use the 4 variables defined in the file (*temp, hall, red_led_state, green_led_state*) to measure the sensor and pin values.
- **Create an HTTP server** using **socket API** to listen for incoming requests from any client (browser on your local PC or phone connected to same Wi-Fi network).
- **Use an infinite loop**: Whenever your ESP server receives any client request (e.g., pressing any button on the webpage), it should use the function web_page to update the HTML text with the current sensor and LED pin values, SEND necessary HTTP headers and finally SEND the updated HTML text as the response to the client.

2.3.2. Sample Result

Open a web browser on your PC or phone and type in the IP Address of your ESP32 server. You should be able to access the web page with the latest sensor readings and GPIO states.

NOTE: Pressing any buttons on the webpage counts as one CLIENT REQUEST. So, every time you press any button, the *temperature*, *hall*, and *led states* should update. e.g., if you press the **RED ON** button, red led connected to your board should light up, and the **RED LED Current State** should display ON as shown in Fig. 2. Now if you press the GREEN ON button, green led connected to your board should light up, and the GREEN LED Current State should display ON, as shown in Fig. 3.



Not secure | 192.168.1.104

ESP32 WEB Server

Temperature 125 °F

Hall 75 V

RED LED Current State: **ON**

RED ON

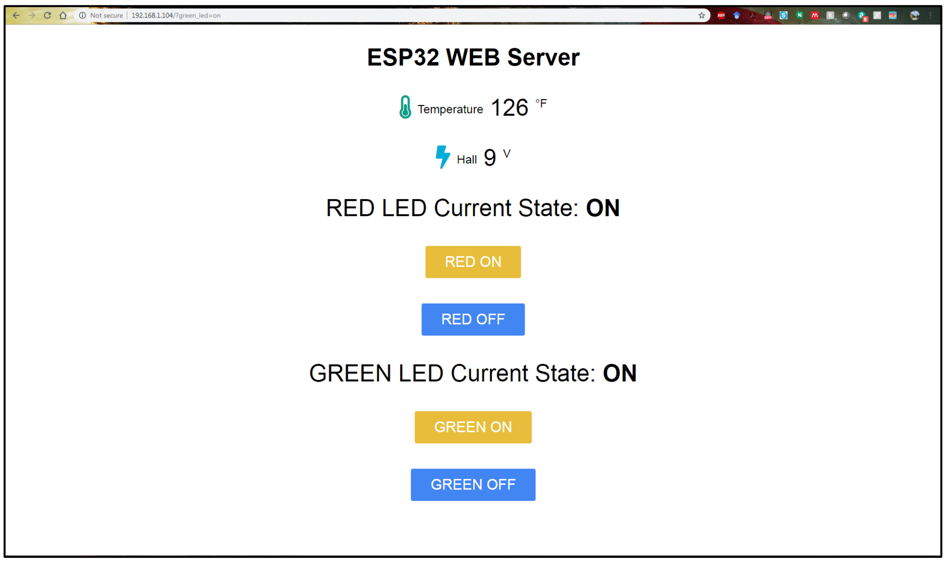
RED OFF

GREEN LED Current State: **OFF**

GREEN ON

GREEN OFF

Figure 2. Webpage on pressing RED ON button



Not secure | 192.168.1.104/?green_led=on

ESP32 WEB Server

Temperature 126 °F

Hall 9 V

RED LED Current State: **ON**

RED ON

RED OFF

GREEN LED Current State: **ON**

GREEN ON

GREEN OFF

Figure 3. Webpage on pressing GREEN ON button

3. Submission

Make sure you follow these instructions precisely. Points will be deducted for any deviations. You need to turn in your code on Brightspace. Please create a directory named ***username*_lab4**, where username is your CAREER account login ID. *This directory should contain only the following files, i.e., no executables, no temporary files, no sub-directories, etc.*

1. ***espclient.py***: your program for ESP32 HTTP Client (**2.2.2**)
2. ***screenshot.jpg (or .png or .gif)***: Screenshot obtained from ThingSpeak website showing Temperature and Hall Senor data, each with ~18 entries, as shown in Fig. 1.
3. ***espserver.py***: your program for ESP32 HTTP Web Server (**2.3.1**)
4. ***README.txt***: A text file with a description of your hardware connections (which pins you used, exactly what they were connected to, etc.).

Zip the files and name it as *username*_lab4.zip and **upload the .zip file to Brightspace**.

Note: For reference, the directory and file structure of your submission should look something like this. Note the spellings, spaces, etc. in the file/directory names.

```
username_lab4.zip
    |---username_lab4 (directory)
        |---espclient.py
        |---screenshot.jpg
        |---espserver.py
        |---README.txt
```

4. Video Submission

Create a short video that shows you demonstrating your working solution (explain your hardware connections briefly before demonstrating the working solution). Please do not upload video files directly on Brightspace. Instead, upload the video to YouTube (or other such video hosting platform) and include the link to the video in your README file above.

5. Bonus Exercise (15% of the total)

- Interface one external **push button switch** with your ESP32 board.
- Update the HTML text given in the espserver.py file to display the switch value, i.e., pressed or not. You don't need to create any buttons for the switch. Your webpage should display the following in addition to the sensors and LEDs information:
  - Either SWITCH Current State: ON or SWITCH Current State: OFF depending on switch state.

NOTE: You don't need to update the webpage upon every switch press. Instead, the client (webpage) is still responsible for reading new values, e.g., if you keep the switch pressed, and then refresh the webpage, webpage should show SWITCH Current State: ON.

6. Grading

We will use a combination of automatic grading scripts and manual code inspection for evaluation. If your code is unusually complex or different from expected solutions, you may be asked to attend an office hour and explain your code.

NOTE: Follow the lab document strictly when using different peripherals/modules/packages. DO NOT USE 'urequests' module for this lab. Points will be deducted if you fail to follow the lab instructions. If anything is NOT mentioned explicitly, you can use package/module to write your program.

REFERENCES

[1] Getting started with MicroPython on the ESP32 https://docs.micropython.org/en/latest/esp32/tutorial/intro.html

[2] ESP32 WROOM-32 Datasheet https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

[3] ESP32 Technical Reference Manual https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf

[4] Adafruit HUZZAH32 – ESP32 Feather Online Manual https://learn.adafruit.com/adafruit-huzzah32-esp32-feather

[5] Adafruit ESP32 Feather Schematics https://cdn-learn.adafruit.com/assets/assets/000/041/630/original/feather_schem.png?1494449413

[6] MicroPython GitHub https://github.com/micropython/micropython

[7] ESP32 specific functionalities in MicroPython http://docs.micropython.org/en/latest/library/esp32.html

[8] ThingSpeak: https://thingspeak.com/

[9] MicroPython socket module: https://docs.micropython.org/en/latest/library/socket.html