Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

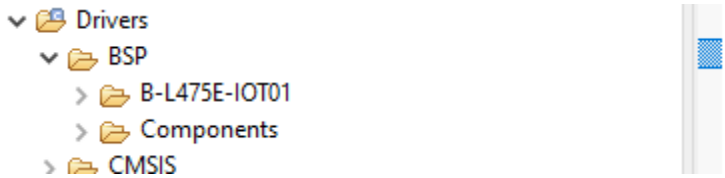
Date: 5/19/2021

Assignment 7: WiFi Hands On

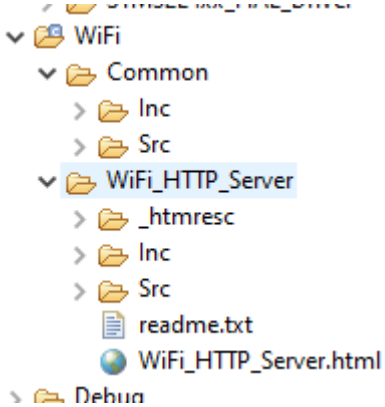
The following will document completion of the seventh assignment for ECE-40293, using the onboard hardware to complete the following task:

You are to implement the Wi-Fi HTTP Server based off the sample HTTP server included in the STM32Cube repository. You should show output from the console and also from the web page.

To begin, we will generate a new project as in other assignments using the default configuration methods. Once the project has generated, we will begin the process of manually adding the support files needed to for the server project. To begin with, import the BSP files of the appropriate FW revision (1.17 in my case) using the same techniques applied in the SPI assignment. We should have both the board directory and the Components directory, as so:



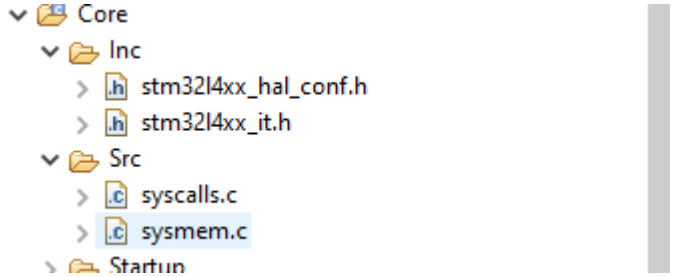
Next, we will follow a similar process to bring in the WiFi project folder, including the Common Folder and the WiFi_HTTP_Server, deleting the unneeded Client folder from the project:



Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

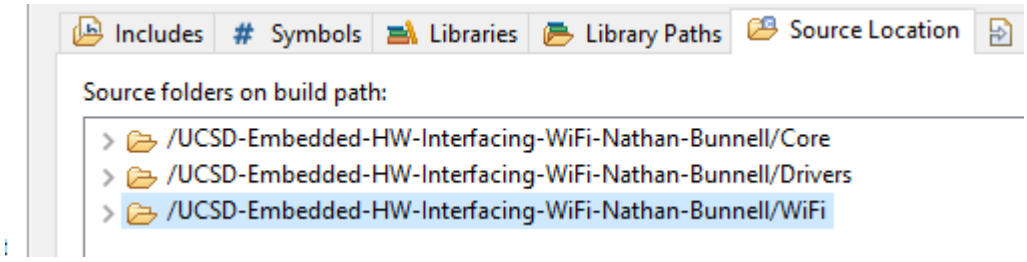
Date: 5/19/2021

At this point, we will have duplicates to several files between the Core and WiFi folders and can delete the unneeded ones that were automatically generated by the IDE when we built the project. Within the Core directory, we can delete everything from the Inc and Src dirs. Except for the following:



Our next configuration step will be to update the Include's path, so the linker knows to look in the new BSP and WiFi folders we created, again using the same methods applied to import BSP functionality within the SPI project.

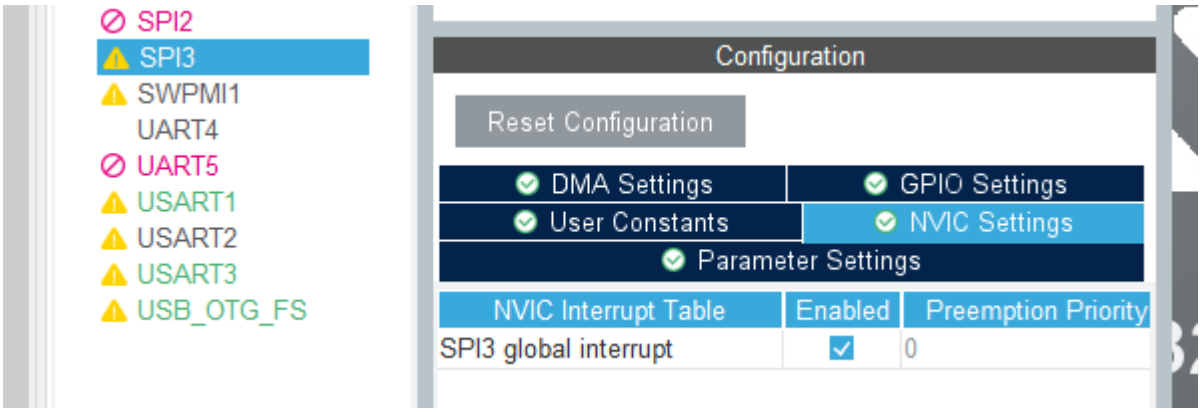
An additional step at this point is update the location of our source folders, again using the same methods as above:



Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/19/2021

And the final step of our project configuration is to enable the SPI3 NVIC so that we can correctly interface with the WiFi module, as explored in the debugging video:



With all this complete, we can compile and flash our Disco board. As recommended in the video, I attempted builds between most of these steps to verify that we were continuing to function incrementally. However, unlike the video, I added the WiFi portion first and started with several dozen errors until I remembered that we need the BSP code for this project. After sorting this, I could build without errors or warning but didn't see anything under my terminal connection. Right before entering the debugger, I realized that the WiFi/Common/Inc dir was not in the project path. After a rebuild, I was finally able to see the server start up from the console.

At this point, I realized I had also breezed right past the step of adding my SSID and password to within main.c. After another rebuild with my home wifi credentials added, I ran into the same issue as in the second debugging video: the wifi module didn't have enough signal strength to make it the 20 feet or so across the basement to my router.

So, with a *final* rebuild and flash, I successfully connected to my cell phone as a mobile hotspot.

```
23  /* Update SSID and PASSWORD with own Access point settings */
24  #define SSID     "Bunnell"
25  #define PASSWORD "Password"     // Not my normal password :)
```

Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/19/2021

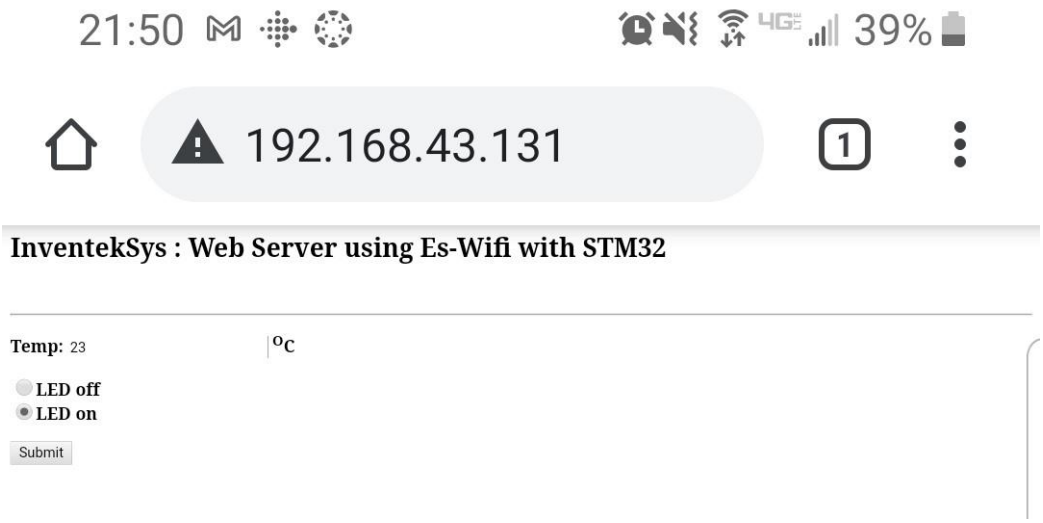
Below is a trimmed down expert of the PuTTY log, with "…" replacing multiple lines of "Waiting connection to 192.168.43.131".

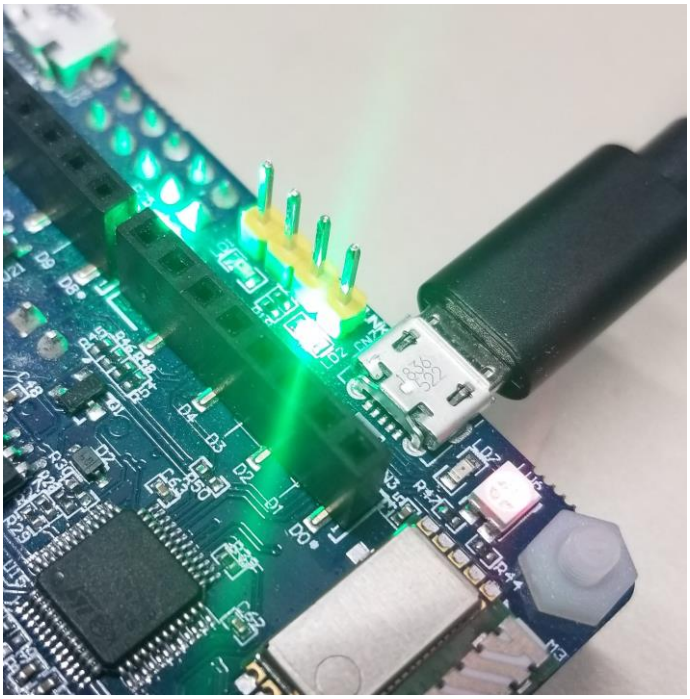
```
****** WIFI Web Server demonstration******


Running HTML Server test
ES-WIFI Initialized.
> eS-WiFi module MAC Address : C4:7F:51:07:D0:6F

Connecting to Bunnell , Password
> es-wifi module connected: got IP Address : 192.168.43.131
Server is running and waiting for an HTTP  Client connection to 192.168.43.131
Waiting connection to  192.168.43.131
Client connected 192.168.43.1:33644
get 439 byte from server
Send page after  GET command
Waiting connection to  192.168.43.131
Client connected 192.168.43.1:33646
get 382 byte from server
Send page after  GET command
Waiting connection to  192.168.43.131
...
Client connected 192.168.43.1:33648
get 605 byte from server
Post request
Send Page after POST command
Client connected 192.168.43.1:33650
get 382 byte from server
Send page after  GET command
Waiting connection to  192.168.43.131
...
Client connected 192.168.43.1:33654
get 605 byte from server
Post request
Send Page after POST command
Waiting connection to  192.168.43.131
Client connected 192.168.43.1:33656
get 382 byte from server
Send page after  GET command
Waiting connection to  192.168.43.131
```

Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/19/2021

The above captures part of the following processes of connecting to the server via a webrowser on my phone and being able to get a reasonable temperature read as well as toggling the LED on and off.





And while I didn't capture it, I did test the temp reading by exhaling over the sensor for a few seconds, refreshing the page, and seeing the value had gone up 1 degree.

Nathan Bunnell
Embedded Systems Hardware Interfacing
ECE-40293
Student ID: U08895857

Date: 5/19/2021

Closing Thoughts

Just like with the USB assignment, my two big takeaways are that this was 1) extremely easy to set up and get running, and 2) I would really like to dive deeper into the code that provides this functionality. I've played around a bit with WiFi chips in the past, generally something from the Espressif ESP lines, but have never really dug too deeply beyond what is broken out in the SDKs or abstracted under the Lua or MicroPython interfaces. Sometime in the (far) future, I have a project planned to develop and build a serial-WiFi adapter to allow me to use my TRS-80 100 as a Telnet interface to my various development machines. I'd always assumed that this would be accomplished with an ESP8266 or something similar, but after this assignment, I'll have to take a closer look at the ST offerings and the example code provided under the board packages.