Lab #6
ESP8266 WiFi Module in AP Mode

Ver 0.50 w/software serial

ESP8266 Background:

In the last lab, we used the ESP8266 as a client sending information to a server. In this lab, the ESP8266 will act as a wireless access point (AP), a very simple web server and as an IoT device to allow you to see the ADC values on a browser as well as turn on/off an LED. In addition to last week's circuit, you will control the built in LED (PB5 pin 13) via the internet.

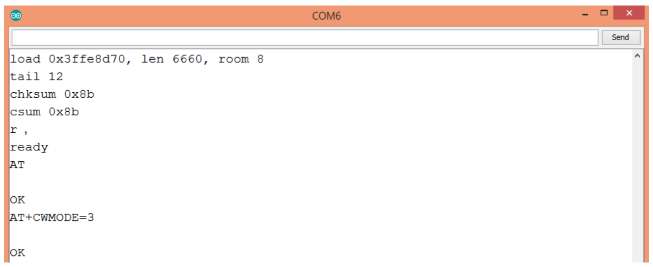
You will set up your ESP8266 as an AP, wait for an incoming connection, read the ADC's values and test the incoming request to see if LED should be turned on/off (and do so) then resend the HTML page to the client and close the connection (and go back and wait for a new incoming message).

To set up the ESP8266 as a WiFi AP the instructions are as follows[1]:

Set the WiFi mode, so the device operates both as station and access point. To do so, we just send the following command:

AT+CWMODE=3

We should get an OK message, as indicated in figure 1. Please note that more WiFi modes are supported. The various options are documented in the AT command firmware manual.



```
load 0x3ffe8d70, len 6660, room 8
tail 12
chksum 0x8b
csum 0x8b
r ,
ready
AT

OK
AT+CWMODE=3

OK
```

Figure 1 – Output of the AT command to set the WiFi mode.

Then, we will configure the access point we are setting with the AT+CWSAP command. This command receives as parameters the name of the network we are setting, the password, the channel and the encryption mode. Note that this last parameter is passed as a number, with the options available listed bellow:

0 – Open
2 – WPA_PSK
3 – WPA2_PSK
4 – WPA_WPA2_PSK

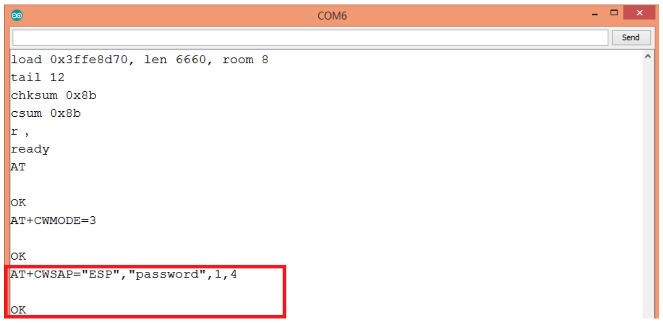
[1] https://techtutorialsx.com/2017/05/20/esp8266-wifi-bee-setting-an-access-point-with-at-commands/

In the example below, the network name is set to “ESP”, and the password is set to “password”, define channel 1 and encryption type equal to WPA_WPA2_PSK. Check the command we need to send to the device bellow:

AT+CWSAP=”ESP”,”password”,1,4

[please use TECH4234_<your initials> as the network name, and use password “4234”]

The result of the command is shown in figure 2. As can be seen, an OK message should be returned if everything is correctly configured.



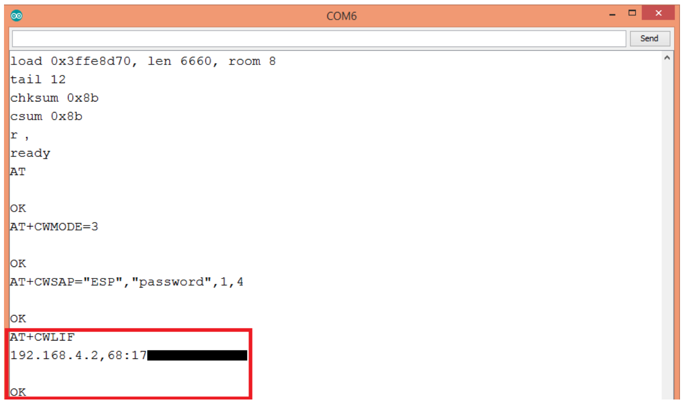
```
load 0x3ffe8d70, len 6660, room 8
tail 12
chksum 0x8b
csum 0x8b
r ,
ready
AT

OK
AT+CWMODE=3

OK
AT+CWSAP="ESP","password",1,4

OK
```

Figure 2- Setting the AP configurations.

Our new network should be accessible from other devices. But we still need to allow the ESP8266 to assign an IP address to the device connecting to it. For this we need to turn on DHCP with the command:

AT+CWDHCP=0,1

To finish this tutorial, we will check the IP of devices that have joined the network. To do so, we just sent the AT+CWLIF command. As can be seen in the figure below, both the IP assigned to the device and its MAC are shown.



```
load 0x3ffe8d70, len 6660, room 8
tail 12
chksum 0x8b
csum 0x8b
r ,
ready
AT

OK
AT+CWMODE=3

OK
AT+CWSAP="ESP","password",1,4

OK
AT+CWLIF
192.168.4.2,68:17[redacted]

OK
```

Figure 3- IP and MAC addresses of the device previously connected to the ESP8266 access point.

Now we need to tell the ESP8266 to allow multiple TCP connections using the command:

AT+CIPMUX=1

This mode MUST be set for the ESP8266 to act as a server.

Lastly we have to tell the ESP8266 to listen for incoming connections by using the command:

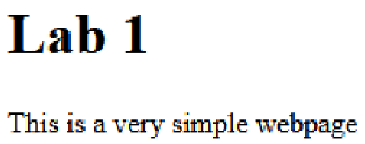
AT+CIPSERVER=1,80

(were 80 is the port to listen to…in this case 80 is the standard port for a web server).

Once this command is sent you can watch the input counter (c) to determine if an incoming request has been received (I found when c is > than 40 characters works well)

Making your Arduino look like a web server:

If you recall from Lab #8 in TECH 3812 (Digital Communications), if you type into a browser http://tech-uofm.info/TECH3812a.html you should get:



Lab 1

This is a very simple webpage

But if you look at the response via wireshark you will get:

```
Frame 294: 390 bytes on wire (3120 bits), 390 bytes captured (3120 bits) on interface
\Device\NPF_{6A3C51E9-B582-4FBA-BEC3-1A85FCB3FC0E}, id 0
Ethernet II, Src: HonHaiPr_8c:ef:f7 (90:4c:e5:8c:ef:f7), Dst: Dell_d8:9c:76 (f8:b1:56:d8:9c:76)
Internet Protocol Version 4, Src: 50.246.145.13, Dst: 192.168.0.122
Transmission Control Protocol, Src Port: 80, Dst Port: 55877, Seq: 1, Ack: 363, Len: 336
Hypertext Transfer Protocol
    HTTP/1.1 200 OK\r\n
    Date: Tue, 28 Nov 2023 18:41:30 GMT\r\n
    Server: Apache/2.4.56 (Debian)\r\n
    Last-Modified: Sat, 14 May 2022 16:10:26 GMT\r\n
    ETag: "35-5defb076db52d"\r\n
    Accept-Ranges: bytes\r\n
    Content-Length: 53\r\n
    Keep-Alive: timeout=5, max=100\r\n
    Connection: Keep-Alive\r\n
    Content-Type: text/html\r\n
    \r\n
    [HTTP response 1/2]
    [Time since request: 0.001131000 seconds]
    [Request in frame: 292]
    [Next request in frame: 295]
    [Next response in frame: 296]
    [Request URI: http://tech-uofm.info/favicon.ico]
    File Data: 53 bytes
Line-based text data: text/html (3 lines)
    <h1>Lab 1</h1>\n
    <p>This is a very simple webpage</p>\n
    \n

0000  f8 b1 56 d8 9c 76 90 4c e5 8c ef f7 08 00 45 00   ..V..v.L......E.
0010  01 78 37 ce 40 00 40 06 7c 8c 32 f6 91 0d c0 a8   .x7.@.@.|.2.....
0020  00 7a 00 50 da 45 6d 1a 09 ab 64 85 7f e0 50 18   .z.P.Em...d...P.
0030  01 f5 a0 1c 00 00 48 54 54 50 2f 31 2e 31 20 32   ......HTTP/1.1 2
0040  30 30 20 4f 4b 0d 0a 44 61 74 65 3a 20 54 75 65   00 OK..Date: Tue
0050  2c 20 32 38 20 4e 6f 76 20 32 30 32 33 20 31 38   , 28 Nov 2023 18
0060  3a 34 31 3a 33 30 20 47 4d 54 0d 0a 53 65 72 76   :41:30 GMT..Serv
0070  65 72 3a 20 41 70 61 63 68 65 2f 32 2e 34 2e 35   er: Apache/2.4.5
0080  36 20 28 44 65 62 69 61 6e 29 0d 0a 4c 61 73 74   6 (Debian)..Last
0090  2d 4d 6f 64 69 66 69 65 64 3a 20 53 61 74 2c 20   -Modified: Sat, 
00a0  31 34 20 4d 61 79 20 32 30 32 32 20 31 36 3a 31   14 May 2022 16:1
```

```
00b0  30 3a 32 36 20 47 4d 54 0d 0a 45 54 61 67 3a 20   0:26 GMT..ETag:
00c0  22 33 35 2d 35 64 65 66 62 30 37 36 64 62 35 32   "35-5defb076db52
00d0  64 22 0d 0a 41 63 63 65 70 74 2d 52 61 6e 67 65   d"..Accept-Range
00e0  73 3a 20 62 79 74 65 73 0d 0a 43 6f 6e 74 65 6e   s: bytes..Conten
00f0  74 2d 4c 65 6e 67 74 68 3a 20 35 33 0d 0a 4b 65   t-Length: 53..Ke
0100  65 70 2d 41 6c 69 76 65 3a 20 74 69 6d 65 6f 75   ep-Alive: timeou
0110  74 3d 35 2c 20 6d 61 78 3d 31 30 30 0d 0a 43 6f   t=5, max=100..Co
0120  6e 6e 65 63 74 69 6f 6e 3a 20 4b 65 65 70 2d 41   nnection: Keep-A
0130  6c 69 76 65 0d 0a 43 6f 6e 74 65 6e 74 2d 54 79   live..Content-Ty
0140  70 65 3a 20 74 65 78 74 2f 68 74 6d 6c 0d 0a 0d   pe: text/html...
0150  0a 3c 68 31 3e 4c 61 62 20 31 3c 2f 68 31 3e 0a   .<h1>Lab 1</h1>.
0160  3c 70 3e 54 68 69 73 20 69 73 20 61 20 76 65 72   <p>This is a ver
0170  79 20 73 69 6d 70 6c 65 20 77 65 62 70 61 67 65   y simple webpage
0180  3c 2f 70 3e 0a 0a                                 </p>..
```

In the program you are going to write, you will need to emulate the response of a web server. If you look in the above wireshark output, you will see the standard server response: "HTTP/1.1 200 OK". This tells the client that the page was found and what version of HTTP is being used. Further on you will see: "Content-Type: text/html" telling the client that it is being sent as text/html code. Lastly, in the above you will see "Connection: Keep-Alive" but for OUR program we will use: Connection: close" telling the browser to close the connection when done (instead of the more typical Connection: Keep-Alive).

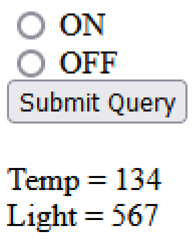
After an extra \r\n the server puts the HTML code for the page to be displayed.

The above is the simplest response needed to emulate a web server.

So to sum up, you will need to send:

```
        HTTP/1.1 200 OK\r\n
        Content-Type: text/html\r\n
        Connection: close\r\n
        \r\n
        <HTML Code HERE>
```

For the HTML code, we want to generate a page that looks like this:



Temp = 134
Light = 567

The code for the above is:

```
        <!DOCTYPE HTML>
        <html>
        <body>
        <form>
        <input type = "radio" name = "LED" value = "ON"> ON<br>
        <input type = "radio" name = "LED" value = "OFF"> OFF<br>
        <input type = "submit"
        </form>
        <br><br>
        Temp = 134<br>
        Light = 567
        </body>
        </html>
```

You will need to replace the static values “134” and “567” with data read from the ADC as you did last week.

To send the html page you will need to do the command:

AT+CIPSEND=0,length

Where length is the length of the entire header (HTTP….</html>) calculated by the strLen function.

Once the page is sent (and re get any response back) you will need to do one final command:

AT+CIPCLOSE=0

To close the connection and go back into listen mode for the next connection.

When you select one of the two radio buttons, and hit the “submit query” button the program will receive one of two strings:

LED=ON
LED=OFF

The two will be PART of a larger string (in the array x). To test if a string is present in a larger string you can use:

```
If(strstr(x,”LED=ON”)!=NULL)
```

And then do the appropriate steps to turn on (or off in the case of LED=OFF”)