Chukwuemeka Osaretin Ike

Mr. Brent Garner

ELEN 462-A

16 November 2018

**HUD Alarm Clock**

**Background**

This project came from the desire to develop an alarm clock with a heads-up display that could be viewed through a transparent film. It was intended to be a simple project utilizing parts that had fallen out of use in my home. The project stayed mostly true to that initial idea

The aim of this project was to develop my skills in implementing network protocols including HTTP for use in future IOT projects and applications. The project is a simple alarm clock that sources real-time weather and clock data from an online API provided by ThingSpeak, and it displays the information on an LCD screen which is reflected on a transparent secondary display.

At this time, the alarm clock is an unrefined prototype. The criteria stated in the Honors Option form have been met, but more work needs to be done to deliver a fully functional product. This document will also be updated and re-submitted accordingly.

**Parts**

1. **mBed NXP LPC1768 MCU**. This microcontroller was chosen because it was available from a previous class and the fact that it outputs 3.3V on its pins - the ESP’s preferred voltage level.
2. **ESP8266 WiFi Module**. This was the most complicated component in the whole setup and the main attraction. It is a $5 module that can add Wi-Fi capability to any project
3. **JHD659 16x2 LCD Screen**. This is our display. It is capable of holding 2 lines of 16 characters each. It runs on an SPI interface
4. **Generic Piezoelectric Buzzer**. This is the alarm for the intended alarm feature to be added at a later date
5. **Generic button**
6. **10 kΩ resistors**. These were used in pulling the GPIO pins on the ESP up to 3.3V
7. **Male-to-male headers**

**Hardware Connections**

1. Upon shifting to the mBed, there was an initial inability to find a discernible tutorial on wiring the ESP-01 to the mBed. One was eventually found and the suggested wiring is listed below:
   1. Reset to p26
   2. TX to p27
   3. RX to p28
   4. GND and 3V3 to the corresponding pins
   5. CH\_PD is the enable pin. Tie this HIGH.
   6. I believe GPIO0 and GPIO2 both need to be tied high through 10㏀ resistors, although this has not been confirmed. The board seems to react erratically at times to the position of the GPIO pins, but has been most accommodating of this setup. As of this writing, the ESP has been powered and used regularly for over two months without issue.

|  |  |
| --- | --- |
| **ESP Pin** | **Mbed Pin** |
| RST | p26 |
| TX | p27 |
| RX | p28 |
| GND | GND |
| 3V3 | 3.3V |
| CH\_PD or EN | 3.3V |
| GPIO0 | 3.3V through 10kΩ |
| GPIO2 | 3.3V through 10kΩ |

Table 1: Mbed-ESP Wiring Table

1. Once reliable wiring was established, I wrote setup and configuration code based heavily on earlier work published by Jim Hamblen accessible at link 1 in the list of links further in this document.
2. The ESP-01 board works on AT (modem) commands. In the configuration code, these are copied into the module’s buffer in two stages. The first uses the “strcpy” and “strcat” C functions to change the contents of a pre-declared character array, and the second uses an esp.printf() function to copy these instructions into the buffer.
3. This method was employed because of an inability to find pre-written libraries for the ESP8266 module. The only library found presented more challenges than solutions, so I elected to abandon it.
4. The initial configuration code written by Jim Hamblen simply connects the ESP to the access point and acquires IP and MAC addresses for further use.

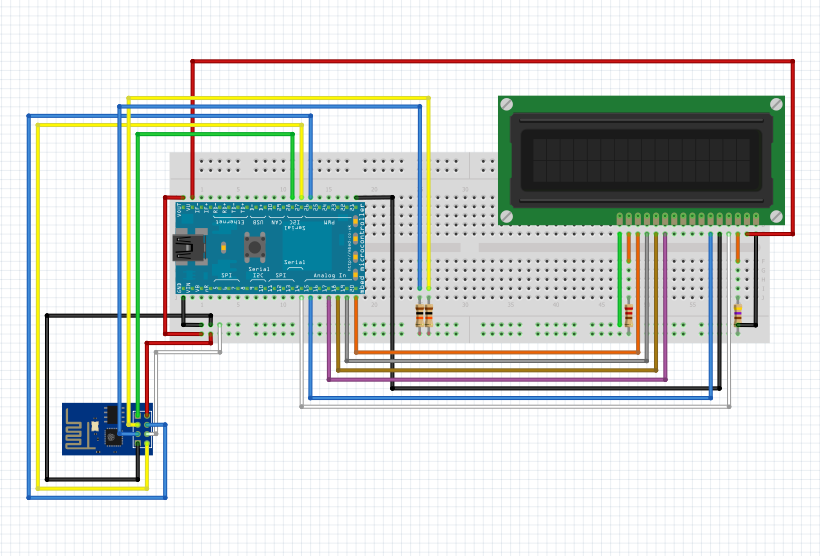
****

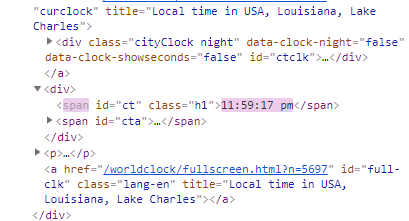
Figure 1: Wiring Schematic for the ESP module and LCD screen.

**Weather and Time via ThingHTTP**

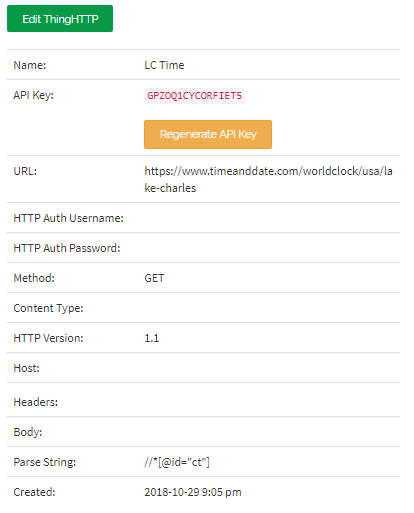
1. In order to get the weather and time to the mBed, I used the ThingHTTP app from ThingSpeak to strip the desired information from a single web page.
2. To set up the simple web page that contains the time, a ThingHTTP app will need to be set up.
3. In ThingSpeak, go to the Apps toolbar and select ‘ThingHTTP.’
4. Select ‘New ThingHTTP,’ give it an appropriate name - ‘LC Time’ in this example
5. You will need to provide the URL of the target page: <https://www.timeanddate.com/worldclock/usa/lake-charles>.
6. Set Method to ‘GET,’ and HTTP Version to 1.1. The last piece needed is the Parse String. To obtain this, navigate to the URL as above
7. The target component of the webpage is the time as displayed below:



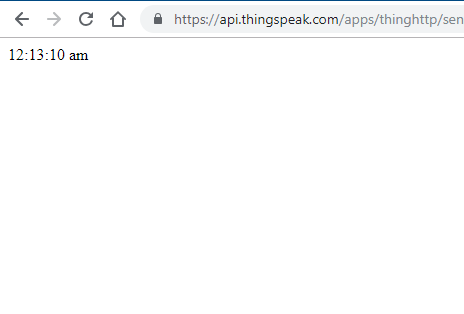
1. Right-click anywhere on the page and select ‘Inspect.’ This will provide an additional pane with the html code for the current web page. It takes some inspection to find the portion that displays the time on the page as highlighted below:



1. Right-click this section of code and select ‘Copy XPath.’
2. Navigate back to the open ThingHTTP tab and paste this XPath into the Parse String field. This is all that is necessary to set up an API to access the time on the web page. Your ThingHTTP setup should bear some resemblance to the figure below



1. Once this is saved, there should be a toolbar to the right of the page showing ‘GET [https://api.thingspeak.com/apps/…](https://api.thingspeak.com/apps/%E2%80%A6)’ You can confirm that the procedure up to this point has worked by copying this path and going to it in your browser. The response should be similar to below



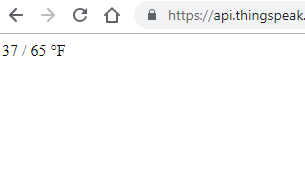
1. For the weather, the ThingHTTP process is the same. The only difference is that the data is sourced from slightly lower in the web page.



1. The target is the high and low for the day. Inspect the page to find the target area and copy the XPath.



1. Use a second ThingHTTP app with all the same parameters, but be sure to use the right Parse String for the weather. Check that this has been successful by navigating to the [https://api.thingspeak.com/apps/…](https://api.thingspeak.com/apps/%E2%80%A6) link that is generated. The result should be similar to that below:



1. You have now set up the API’s for easy access to the necessary information.

**Code**

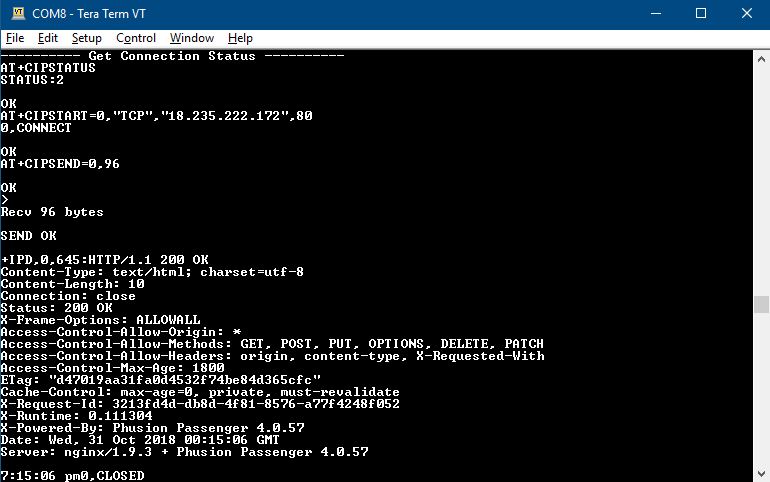
The C++ code is accessible in the main.cpp file at:

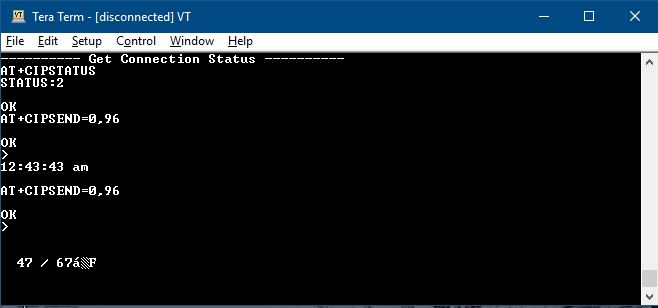
<https://github.com/Chukwuemeka23/espMbedThingSpeak/blob/master/main.cpp>

This code will be updated to be more effective and scalable once this semester is over.

**Testing**

The terminal output showing the server’s response to the time request:



Terminal output after parsing the server’s response for just the time and weather in each case:

**Challenges**

1. The ESP-01 is not breadboard friendly, so jumper wires needed to be soldered on to provide proper connectivity.
2. The project was initially planned to be deployed on the Arduino Uno MCU with the ESP-01 providing WiFi capabilities. This presented a fairly simple issue - the Uno outputs 5V, but the ESP-01 runs on 3.3V. The proposed solution was to buy a bi-directional logic level converter (bi-di converter) to carry out the necessary logic level conversions between both boards.
3. Upon purchase and implementation of the converter, it was found that the converter had continuity issues. It is unclear whether this problem arose from my soldering, or if it was a defective board from the start.
4. The inability to use the bi-di converter made it unreliable to use the Arduino Uno in the project, and because I already had an mBed microcontroller which naturally outputs at 3.3V, I decided to switch the main board to the mBed NXP LPC1768.
5. Once a connection with the ThingSpeak server was made and the desired information could be reliably printed to the PC terminal, I attempted to connect the LCD and print out the data there. A new problem then reared its head - the connection with the server became unviable.
6. A ‘link error’ has been displayed on every connection attempt in the last 48 hours. I do not believe it is a result of the code or hardware as the same setup was used with success in the past. This is still being worked on at this time.

**Note:** It has since been discovered that the ESP can run on 5V from the Arduino as Mr. Mitch

Morgan is using the ESP in such a setup as of this time. It seems that connecting the Vcc pin to 5V instead of 3.3V might cause the ESP to take and output 5V logic levels. I have not tested this approach and neither encourage nor discourage it.

**Links**

1. Simple tutorial by Jim Hamblen on wiring and code connecting the ESP8266 to the WLAN: <https://os.mbed.com/users/4180_1/notebook/using-the-esp8266-with-the-mbed-lpc1768/>
2. Information for connecting the GPIO pins: <http://www.forward.com.au/pfod/ESP8266/GPIOpins/index.html>
3. ESP8266 header files: <https://os.mbed.com/teams/ESP8266/code/esp8266-driver/>
4. Website used for time and date: <https://www.timeanddate.com/worldclock/usa/lake-charles>
5. Common AT commands for the ESP: <https://github.com/espressif/ESP8266_AT/wiki>
6. LCD Data Sheet: <https://www.arduino.cc/documents/datasheets/LCDscreen.PDF>