IoT Smart Alarm Clock

Garrett Strange, Nick Boruff

[strangeg@iu.edu](mailto:strangeg@iu.edu), NICK EMAIL

0003607005, NICK ID

December 9, 2019

Garrett Strange, Nick Boruff

IoT Final Project

December 9, 2019

Alarm Clock Project Report

**Project Description**

For this project we set out to make a IoT alarm clock. We wanted a time to be displayed on a 16x2 LCD screen and be able to set at least one alarm time by using a PUT command to the CoAP server using TxThings. We used a simple buzzer for the alarm noise and a button to stop the alarm when it sounds. We also had a potentiometer to adjust the contrast on the LCD screen.

**Design and Implementation**

First, we started by looking up how to wire our LCD screen. To do this we found a simple diagram online that included the potentiometer so that we could adjust the contrast easily. The potentiometer allowed us to adjust the contrast since depending on the light levels in the room the display was hard to see. This tutorial also came with example code to show how to display text to the screen and change the text as well. We ended up using this code since the LCD display required memory slots since each display bit had some sort of memory attached to it. The datetime library in python became very useful since it allowed us to format a time string with the current date, as well as change the time into the usual AM/PM setting without having to do extra parsing. After this was setup all we had to do was send the time in the 2 available lines on the LCD and we could move on the next step.

Setting up the server was simple since it just required reusing most of the same TxThings server code from project 2, but this time with a PUT command. The PUT command was the harder part since it assumed the user puts in the correct input since we didn’t parse the input. To work around this, we did a GET command with an example of how the input should look so the user could check before they sent the command. The next part of receiving the put and doing something with it, however, took the longest time to figure out by a long shot.

To receive the PUT command input we had to use threads since the server couldn’t receive the input and do something with it. We had to make the clock work on 1 thread while the server worked on the main thread which allowed us to make sure the server was always checking for input and the clock could parse the alarm time set by the user separately. This took a big chunk of brainstorming and we ultimately decided with a global variable ALARM\_TIME which would communicate with both threads so when the server thread changed the variable the clock thread knew to change the alarm.

After we got the inputs all figured out the next step was making the buzzer work when the ALARM\_TIME set equaled the current time. Using the gpiozero library for the buzzer and later the button made this code very easy to read and manipulate. This was done with a simple comparison so that when the alarm was set the buzzer would go off for 1 second then turn off and loop until either the next minute passed or the button was pressed to end the buzzer. The button we created used a global ON variable which would be set to true and would allow the buzzer to cycle between on and off if the ON button was true. Turning the on button off meant the buzzer would end after it beeped twice effectively simulating someone turning the alarm off.

The next step in our design was we wanted to make this program work with a mobile device. Since we didn’t have time to make our own app we browsed the google store and found an app called IoT CoAp which allowed us to use a CoAp client on our phone which we could use to send PUT requests as long as we were on the same network.

**Test and Evaluation**

Testing was straightforward with this program since we had set up print statements on the LCD to show when the user inputted a time. If the user inputted a wrong time, they could change it easily by just sending another PUT command. We found that there was a small delay for when the alarm went off since we did put sleep statements in the display code which triggered before the alarm code ran. This wasn’t a big deal since the alarm still went off until the user pushed the button, or the current time incremented over the alarm time.

Testing with the phone wasn’t too much of a hassle since the server code gave us detailed errors about what was wrong. We ended up realizing that the phone app automatically put /s before the resource which for some reason the server couldn’t handle so we took the / out and we could GET and PUT requests in every time.

**Lessons Learned**

The biggest lesson learned was about how servers work in CoAp, we tried many times to upgrade from TxThings since other CoAp servers like aiocoap are in python 3 but we could just not get them to work at all. In the end we just ended up doing what worked, and TxThings worked fine for us so we just went with it.

The next lesson learned was about threading and global variables. We couldn’t figure out why the threads weren’t communicating for days and trying to pass variables between them, but all we had to do was make a global variable and everything worked out perfectly. Of course, we know global variables are volatile and threads are super volatile too, but it worked, and we were careful, so it all turned out fine in the end.

**Work Distribution**

Overall the work distribution was pretty even with Garrett doing mainly the code and Nick working on the sensors and buying most of the extra stuff needed. Both of us worked together to brainstorm so while Garrett wrote the code, both were involved in the project evenly since Nick helped a ton with the PowerPoint and paper.

**References**

<https://www.win.tue.nl/~lrahman/iot_2016/tutorial/txThings_2016.pdf>

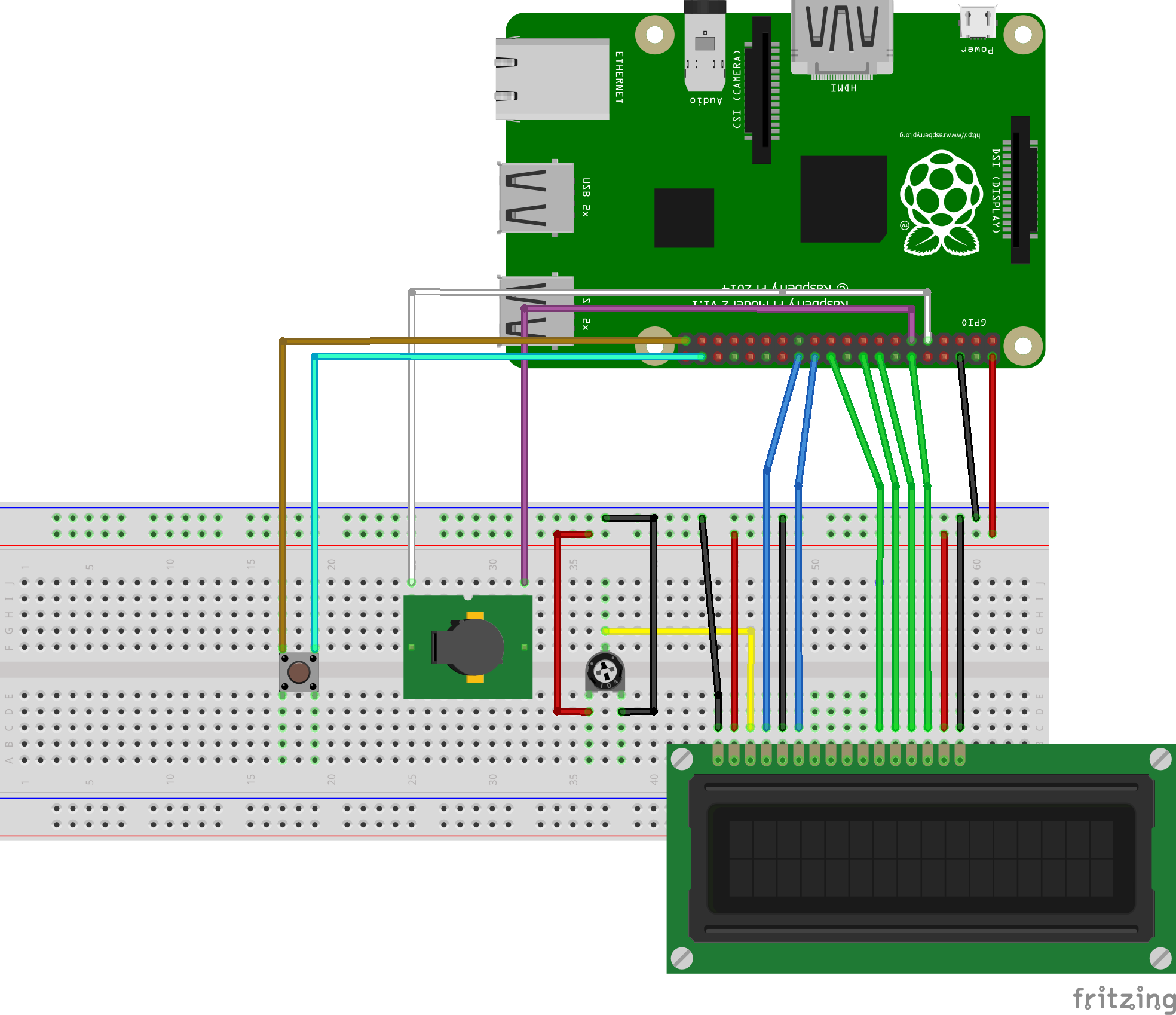
<https://github.com/mwasilak/txThings>

<https://www.mbtechworks.com/projects/drive-an-lcd-16x2-display-with-raspberry-pi.html>

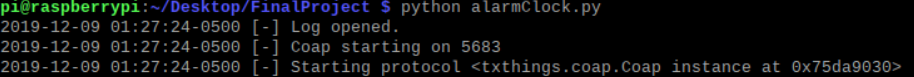
<https://github.com/mkovatsc/Copper4Cr>

<https://gpiozero.readthedocs.io/en/stable/api_output.html>

**Wiring Diagram**



Note: Buzzer shown is not the same exact one we used. Fritzing didn’t seem to have a buzzer that looked like ours, so we used this one as a placeholder.

**Example Server Startup**

**Example Alarm PUT**

