Brandon Stultz

February 17, 2022

CS-350: Emerging Systems

Southern New Hampshire University

Thermostat Process Document

Developing the logic for this project different from others I have previously done. This difference can be attributed to the use of a task scheduler to determine when a state function should be run rather than having each state flow into each other sequentially as needed. I first began the project be setting up a structure that I would use to track each states important variables. This in essence made each state an object. I then established the time periods each state would need to be run at. To accomplish this, I needed to determine how often my timer would tick and how this time mapped to my designed design parameters.

The timing constraints for checking interrupts/changing temperature, checking the current temperature, and displaying this information was 200ms, 500ms, and 1000ms respectively. I used the GCD of these values and found my clock needed to tick every 100ms. Now that I understood how often my clock would tick, I simply assigned each state a wait period of their time constraint divided by the clock interval. For example, since checking for interrupts occurred every 200ms I set its wait period to 2. The logic of this reason will become apparent when I discuss my use of the task scheduler function.

The task schedular is the engine that drives the entire state machine. It was implemented using an array that stores each of the struct objects of the state I had mentioned earlier. It utilizes a for loop to check each state and their corresponding wait period. Each time the check is made and the wait time for that state has yet to be met a state object variable known as ‘elapsed’ time is incremented by 1; 1 being a representation of a single clock tick of 100ms. When the wait period is met however the task scheduler function calls the states run function and executes its code, resetting its elapsed time variable to zero before moving to the next state object in the array. This occurs indefinitely since the program is nested inside a while(1) loop. Iterating through each element of the array, checking that states wait time against its elapsed time, and either running the code and resetting its elapsed time, or incrementing the elapsed time and moving to the next element.

Next it is important to discuss how each state was designed. The check interrupt/change set point temp is designed using binary logic. Its main logic checks a variable known as ‘intFlag’ which is set either to a 1 or 2 depending on which button on the board was pressed. A value of 1 will increase the variable set\_point, a 2 will decrease set\_point, and a zero will change nothing. At the end of the states function it will reset ‘intFlag’ to zero to ensure it only catches interrupts from the buttons and doesn’t read a bug.

The change temperate state begins with setting a variable to store the current temperature captured by out ‘readTemp()’ function. This temp value is then checked against out set\_point value. If the temp is less then set\_point a light is turned on indicating power to be relocated to a heater, else the light remains/turns off. The display state has the simplest logic implementation. It simply takes all the data that has been checked and reevaluated in the previous states and prints them to the console every second.

The main purpose of this project was to develop a prototype thermostat at generates data that could be sent to a cloud server using Wi-Fi. Currently our data format is being printed to the console. Our designed product would send this formatted data over Wi-Fi to be stored in a cloud database. Performing such a task requires the use of integrated boards that enable such transfers and provide the proper hardware. One such useful hardware is the ESP32. created by Espressif Systems. This board makes a strong candite for the job, supporting Wi-Fi and enough flash memory to effectively store and transfer data all for a relatively low price point. A second choice could always be an Arduino board that utilizes Wi-Fi. Such a board can effectively accomplish the designed goal but may fall short if more complex functionality is required. Lastly, I also like the idea of the Rock Pi 4. Supporting Wi-Fi and over 40 GPIO pins makes it a perfect fit for any potential needs for the project.