Isabella Phung
5/17/2024
CSE121

Lab 5 Report

5.1:

I used C and the wiringPi library to make the led of the raspberry pi blink. There was some initial confusion as the version of wiringPi I installed from apt-get was an older, depreciated version, but I managed to get the most up to date version by downloading the .deb file from github and using apt-get install directly on this. The remainder of this project then consisted of string parsing. I created an array of strings consisting of 1s and 0s, where the 1s represented dashes and 0s represented dots for each letter, these entries were then ordered in accordance with ASCII number order such that I could directly reference each character as an index to this array in order to locate the corresponding morse code. While I did use chatGPT for this part, I used it more as an example on string parsing rather than copying over the entire code. I don't really know why, but I wanted to use ChatGPT as little as possible for this project. The biggest difficulty I encountered was properly iterating through a character array, in which I referenced this stack overflow comment on how to do so properly.

5.2 & 5.3:

I combined these two into one since I didn't really have a 5.2 phase and a 5.3 phase when developing this project. I first looked at the one shot and continuous read esp-idf example and ultimately went with the continuous read example. From this, I tweaked the code to remove any ADC2 code and selected only 1 channel to read from. I started with printing out values read by the photodiode to get a sense of how fast it was sampling and how high the raw data values were. I then needed to figure out how to begin reading the dots and dashes. While I could have used a timer to read the values, I decided to count the number of samples to determine what was a dot and dash. I had a counter that would iterate each time a sample passed a threshold. Depending on the counter's value, it was a dot or a dash. In order to understand the length of a dot and dash, I used this morse code timing diagram. As the program reads a dot or dash, it traverses a morse code binary tree implemented in array form to determine which character it corresponded to. I had some difficulty piecing together the final string of the message in which the final message size wasn't big enough to house the additional characters, but found some advice when using strcat in this stack overflow post by initializing the buffer of some larger arbitrary size. The fastest I could get my program was based off of the length of a dot in time, which was 30ms, which approximated to about 2.7 characters per second. The speed of my program hinged on the number of samples found in one dot, which became a problem when I wanted to speed up my program even faster. I found that the ESP_LOG command to print out each photodiode would slow down the program, preventing it from reading more samples, but I need to count how many high samples were in the duration of a dot. Not only that, but if my program was too fast, then it wouldn't iterate the counter properly. I think that I could have improved the speed by switching over from counting samples to using a timer to sample the time at the start of a transmission, and ending once the photodiode values dropped low. But for the time being, I'm quite satisfied with the speed that my transmitter runs at.