CSC 615-01 SP 21 Assignment 1 – Traffic Light

Name: Jungsun Eoh
Student#: 918590990
Github: jungsun-eoh

This assignment was an excellent opportunity to revisit the general convention of programming in C (Makefile and file comment, etc.) and to learn how to connect and control basic hardware components from code. I built a program simulating Traffic-Light. The program executed Green LED for 6 seconds, switched to Yellow LED for 1.5 seconds, and 5 seconds to Red LED. It repeated 3 times and automatically turned off all LEDs.
I wired Red LED in GPIO 17(physical pin 11), Yellow LED in GPIO 27(physical pin 13), and Green LED in GPIO 22(physical pin 15) with 3 of 220Ω resistors. Ground it with a physical pin 39.

Resistor = (battery voltage - LED voltage) / desired LED current

(3.3 - 2)/10 = 130Ω (3.3 - 3)/5 = 6Ω

To connect the PI from C, I used 2 different methods; using wiringPi library and sysfs. The wiringPi library is a popular method to control the GPIO hardware. It is simple yet efficient and fast. There are key functions in the wiringPi library:

1. Int wiringPiSetup(void)

The setup function must be called at the beginning of the program to initiate the wiringPi pin. WiringPi pin numbering system is different from GPIO pins or physical pins. It is using Broadcom GPIO pin numbers.

2. Void pinMode(int pin, int mode)

This function sets the pin either INPUT, OUTPUT, which is necessary to use the pin.

3. Void digitalWrite(int pin, int value)

It writes the value 1 or 0 on the pin if the pin was set as an output.

Compare to the wiringPI library method, Sysfs is less efficient since file-based access to the hardware can be slower than other methods. To connect the PI from code, I opened 3 different files for 3 different LEDs and closed them after finishing the tasks. When Pi is connected using

the sysfs interface, the user should be used as root. To use the GPIO pin, the GPIO pin should be exported and set the direction to “out.” if its value gets 1, the signal is on, 0 to off.

The most critical yet stupid mistake I made was starting code without fully understanding how the GPIO works. The first code using the wiringPi library was okay since the library did all the hard works; set the GPIO out and write the value 1 or 0. It was a whole different experience with the sysfs interface. I spent too much time on the second code without any luck. I kept getting errors on direction or export; resource busy. I went over and over my code and tweaked it too many. I thought my problem was my code, but it turns out it was GPIO pins’ status. Because my process was stopped with errors, some of the pins are already exported, and the direction was already set to OUT. After I manually unexported the GPIO pin, the code worked perfectly. I missed the GPIO flow: export the pin -> set GPIO direction to out -> write GPIO value 1 or 0 -> unexport the pin. This flow should be in a line at a time in a certain order.