Embedded System Lab 4 & 5

Watchdog Timer

ECE 150 Technical Report

Team 17

Jing Liu

Haolong Yang

He Zhang

November 30th, 2018

Table of Contents

List of figures

1.Project Overview

For the past few weeks, our group has created and implemented a laser system that will be able to detect and analyze the number of objects at any given time in a room and whether objects are moving in or out of the system. The project has highlighted the key operating functions and features used in security laser systems and motion detecting sensors in by current real-time applications around the world. Our group was able to model a scaled version of this technology using simple laser diodes, photodiode semiconductors and a Raspberry Pi computer module. Our project consisted of a software and hardware component that allowed it to function effectively as an independently controlled system. This system was able to read configuration files, create and write to log entry and result files to capture specific activities from the laser system. It achieved the purpose of creating a self-sustaining laser system that would automatically log events to these files every 12 seconds and effectively keep a record of the events that are happening. While our project adequately achieved the main objective, it is currently unable to detect the type of objects passing through and the size of these objects. In the end, the project concluded with the success of several tests trials that accurately logged and displayed a summary of the activities stimulating the laser system.

2.High-Level System Design

The requirements of the circuit for lab3 [1] and lab4 [2] were identical so we followed our circuit design for lab3, which include two breadboards, two photodiodes, two capacitors, and a few other components. Figure. 1. provides the design of our circuit

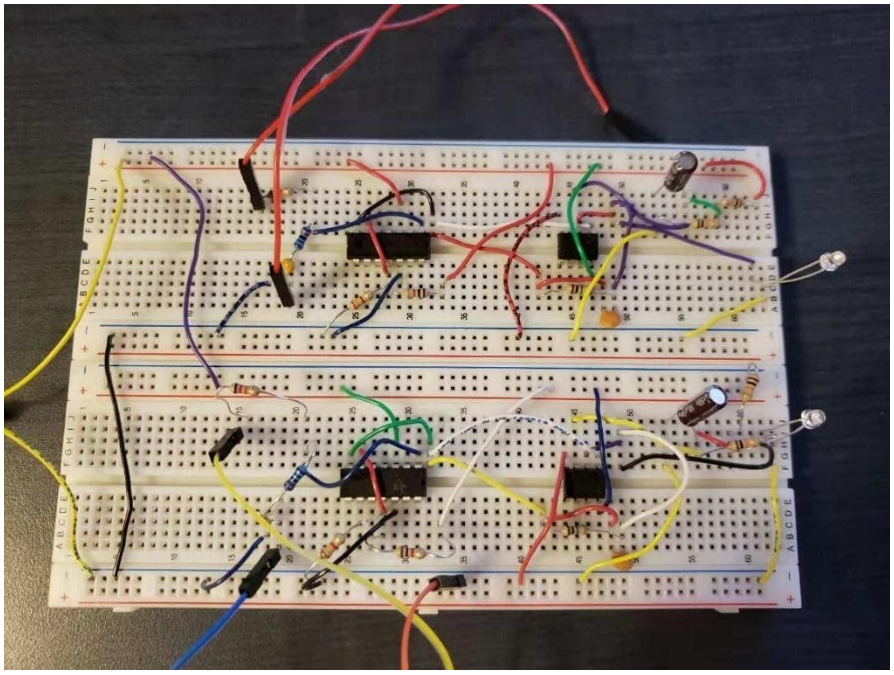


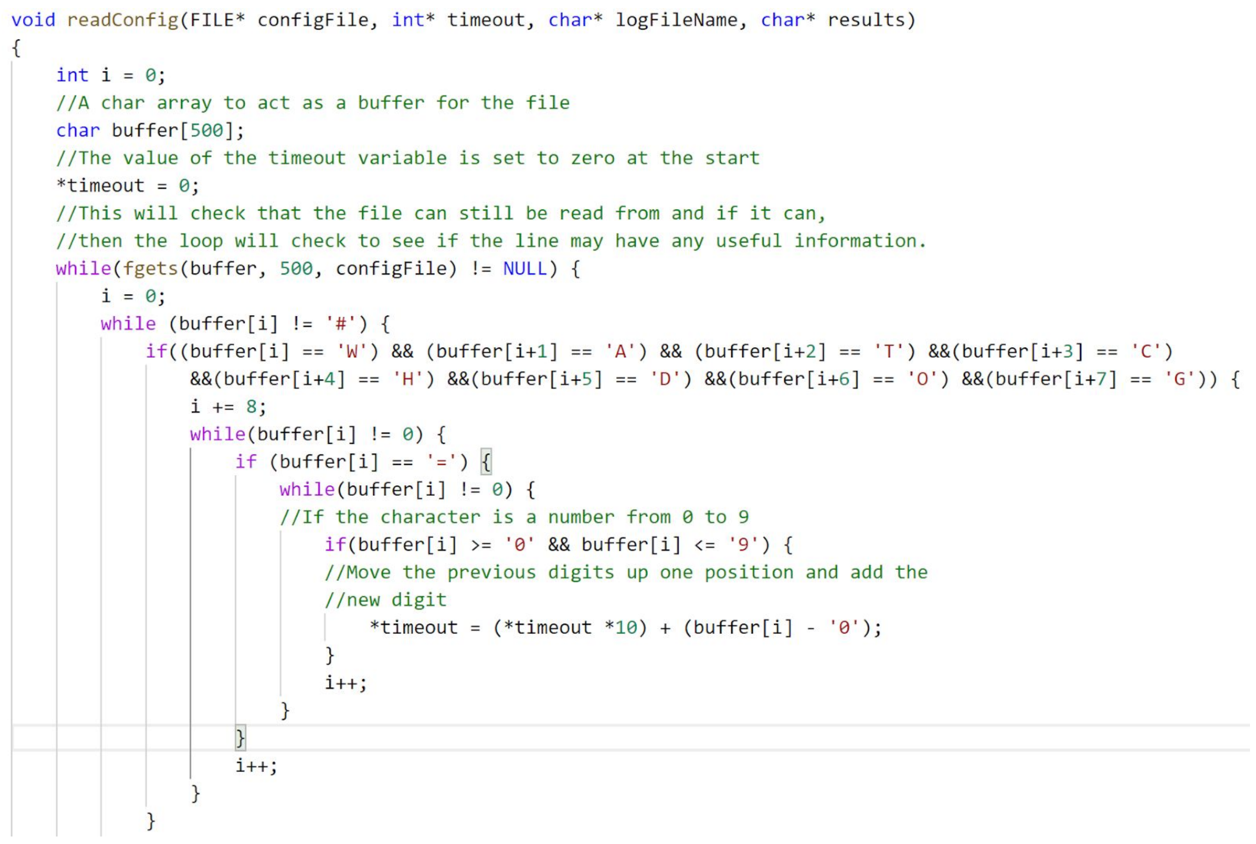
Figure. 1. Lab3 circuit

Figure. 1. shows that our circuit includes some critical components. Two photodiodes were used to detect the light source from the laser as an input. The Operational Amplifier was used to amplify electrical output signals from the photodiode. The comparator was used to check if the amplified voltage from the photodiode is greater or less than the reference voltage [1]. The resistors and capacitors were used to control the limit of the current/voltage in the circuit and most of these components were driven by a DC power supply. The jumper wires were used to connect the circuit to the pi to exchange inputs and outputs. The Raspberry Pi was used as a GUI for running the watchdog system and displaying information in the terminal [2]. The Linux Ubuntu operating system on the virtual machine provided the resources to compile and transfer the code onto the Pi.

Likewise, we keep using our program for Lab3 which also include an initialGPIO function, a pin-set function, and a state machine, except we implemented a watchdog timer in the program to meet the newly stated requirements. We added a readConfig function in order to read the value of the watchdog timer and the addresses of other files in the configuration file. This function will read the content from the specific configuration file and get the essential value and addresses while ignoring the comments and other invalid text. We will call this function in our main function to get the inputs and it will serve the purpose of setting a watchdog timer and allow the program to write to files at a specific location.

3.Software Design

The software design of this project consisted of three major components. The first being the initialization of the GPIO pins and the setting of two specific pins for output on the Raspberry Pi using the LaserDiodeStatus and initializeGPIO methods [2]. This part of the code was crucial in allowing the Raspberry Pi to receive signals from the specific pins connecting to the lasers on the breadboard. It is hardware dependent specific to external hardware such as the Raspberry Pi and breadboard. The main purpose of the method was to allow the Pi to receive input from the breadboard through the pins. This code can be run on any Pi Zero and is flexible to different sets of the pins due to the use of the macro in the initialization [3]. The second major component of the watchdog program was the readConfig function. This function served a purpose of allowing the computer system to read the configuration file which contains important information as to how long the watchdog timer was set to and where the entry or result files are located in order for the program to log specific activities from the laser system. This is a system independent method as it can be used by any general-purpose system to read configuration files and to capture data for writing. The code is robust enough to withstand content changes and extra parameters when reading only the important data used in the program later on. Figure 2 shows part of our readConfig function.



```
void readConfig(FILE* configFile, int* timeout, char* logFileName, char* results)
{
    int i = 0;
    //A char array to act as a buffer for the file
    char buffer[500];
    //The value of the timeout variable is set to zero at the start
    *timeout = 0;
    //This will check that the file can still be read from and if it can,
    //then the loop will check to see if the line may have any useful information.
    while(fgets(buffer, 500, configFile) != NULL) {
        i = 0;
        while (buffer[i] != '#') {
            if((buffer[i] == 'W') && (buffer[i+1] == 'A') && (buffer[i+2] == 'T') &&(buffer[i+3] == 'C')
                &&(buffer[i+4] == 'H') &&(buffer[i+5] == 'D') &&(buffer[i+6] == 'O') &&(buffer[i+7] == 'G')) {
                i += 8;
                while(buffer[i] != 0) {
                    if (buffer[i] == '=') {
                        while(buffer[i] != 0) {
                        //If the character is a number from 0 to 9
                            if(buffer[i] >= '0' && buffer[i] <= '9') {
                            //Move the previous digits up one position and add the
                            //new digit
                                *timeout = (*timeout *10) + (buffer[i] - '0');
                            }
                            i++;
                        }
                    }
                    i++;
                }
            }
```

Figure. 2. Part of the readConfig function.

Figure. 2. indicates that we rely on recognizing the special symbol and letters to distinguish the valid and invalid lines, and read the addresses and watchdog value behind these letters. The function gets input from the configuration file as well as pointers to memory spaces for the location of the log and result files. These parameters can be changed and this function can effectively set a default watchdog time and file locations if there are any errors or corrupted data from the configuration file. This method is quite straightforward and effective but it also has limitations, if we do not put “#” in front of the invalid lines and the end of our config file, the program will keep reading that line and never go to the next line or stop reading, which could cause Segmentation Fault in the end. Moreover, if we change the name defined that before the file location and watchdog time value, the corresponding letters need to be recognized by our program also have to be changed. The final and most important content of the code would be the state machine that would determine the current real-time status of the laser system. It would capture and record the number of times each laser was broken, the number of objects in the room at any given time, and the number of objects that passed through the system. This part of the program is hardware dependent as it relies on direct input from the breadboard via the Raspberry Pi. It can be used with other systems and the state machine can be modified to run efficiently with other setups. The function is independent of the readConfig method but relies on the correct initialization of the corresponding pins reading inputs [3] from the laser system hardware. The purpose of this state machine is to provide a quick and efficient method of analyzing the status of the photodiode and laser diode. With proper suitable substitutes for system-dependent code, this method will run on any general system given the correct hardware setup.

All of the functions mentioned above are well separated and only rely on each other to a certain limited extent. The call tree of this laser system program is shallow and only relies heavily on the functions that initialize the pins and output pins specific to the hardware configuration. Other macros and message functions are embedded and called within the other three main functions but do not affect the robustness and core functionality of the individual important main methods.

Yet, these small functions and macros are equally important to the overall functionality of the watchdog program. The logging infrastructure utilizes these macros to generate error messages and output specific activities to the log and result files by accepting parameters of variables of laser count, time and file name. Our group has collectively decided these action macros to log the outputs received from the laser system to the log and result files every 12 seconds. This justifies ensuring that the system is being updated and checks whether the system is still functioning properly or if there is an error. Our group has also implemented the watchdog kick to be at the end of each cycle of the state machine. This is justified as the system will exit as soon as possible after executing a specific given countdown watchdog time if the user decides to terminate the program. The watchdog is ensured functioning by exiting the program completely after the timer automatically and by rebooting the Raspberry Pi, the program has implemented the watchdog to start the system immediately after reboot in the background. The log and result files also confirmed the correct functionality of this whole laser system comparing the data within those files. The inclusion of the time and messages ensured the system ran continuously until the user terminated the program. This laser system also included additional design features of creating individual macros and default configuration, result and log files and outputting extra messages to these files for better analysis of the status and data of the laser system. The use of classes and structures were disregarded in this program as it did not increase the efficiency and operating functionality of this specific program given the software and hardware setup.

4.Testing

The ensuring of the operational function of this laser system program included using cat and fewer commands in the putty terminal to analyze the current real-time output data from the breadboard via the pin on the Raspberry Pi. The use of the time function aided the analysis as our group was able to correctly identify the time and event of any specific activity readings from the laser diode and photodiode setup. The break of each laser is logged in the result file and can be compared to the actual number of times it was broken. This was also done for the number of objects entering and exiting the room at any given time. The overall careful inspection of the data on the log result files corresponding to the real-time event provided a reliable method of testing to ensure the proper functioning of our watchdog timer system.

5.ELR

Our group has implemented some extra code within the program to allow it to run more efficiently and be less prone to errors. One part of the watchdog program that went beyond the expected components was the implementation of multiple error message and print message onto the log entry and result files. This allowed the user to keep track of the events happening more efficiently and accurately. The idea of using a new macro that our group created for the print message function allowed the program to be more efficient and run effectively. The initialization of several default sets configuration, log and result files reduced the probability of error due to corrupted data during file reading. This was different and superior to the idea of the originally expected method of handling such errors. Along with the robustness of the readConfig function, the program was designed to withstand many changes within the configuration file. With the implementation of these ideas that went beyond the expected components, the program itself was also restricted to some limitations due to the setup. One of these limitations was in the function for reading the configuration file. This method is only able to recognize the 3 specific variables of "WATCHDOG_TIMEOUT", "LOGFILE", and "results". Any other variations of these variables were disregarded as valid parameters and the program would automatically utilize the default files and setting. Our group decided that it was reasonable to assume that any other names for these 3 parameters should be determined invalid as it compromises the integrity and functionality of this program. As for the project reflection, our group decided that if this project was to be redone, we would make sure to write a message to the log file whenever the default file was used instead of the correct original result and log entry files. Additionally, another area for improvement would be spending more time on the readConfig function and implement the use of a state machine to allow for a more efficient execution. Though this implementation does not affect the current functionality of the method, it would increase the overall speed and aesthetic of the program.

6. Appendix

Watchdog.c and configFile.cfg were the main obvious source files for the project's demonstration. The .c file was compiled on the virtual machine, then upload them to ecelinux server, and retrieved to the pi by accessing the server on the pi using the specified commands. Other files that were an important part of our program were the .h header files. The gpiolib_addr.h and gpiolib_reg.h consisted of important information for the initialization of the GPIO pins. Files like linux/watchdog.h and sys/ioctl.h were needed for watchdog specific constants and the ioctl function as well as time.h for the time function. As for the distribution of tasks, He (Jack) wrote the functions and watchdog timer as well as construct the circuit for the watchdog laser system. Haolong debugged the program and included additional features for it to work correctly and efficiently. Jing's tasks consisted of completing the main function and executing the demonstration with the watchdog timer using the commands.. In conclusion, the tasks were equally distributed and our group was satisfied with the results.

Source code: https://git.uwaterloo.ca/h294yang/lab4-lab5

7.References

[1] "ECE150 embedded system lab 3 manual", University of Waterloo, Oct. 2018

[2] "ECE150 embedded system lab 4 manual", University of Waterloo, Nov. 2018

[3] BCM2835 ARM Peripherals, [Online]. Available: https://www.raspberrypi.org/documentation/hardware/raspberrypi/bcm2835/BCM2835-ARM-Peripherals.pdf