**Final Report: Lasers Lab**  
by Daniel Arakcheev, Umar Hussain, Daniel Dobre

ECE 150 Fall 2018

**Project Overview:**

This project was made to mimic a laser doorway such that the number of objects entering/leaving a ‘room’ is recorded. This is done by reading the number of breaks in a laser’s path, as sensed by a photodiode. The project also incorporates debouncing with hysteresis to increase the effectiveness of the photodiodes and to not overcome false positives/negatives. In addition, the raspberry pi watchdog was implemented to recover in case the system gets stuck in a loop/crashes/etc. This system was designed to run without constant monitoring, and as a result, it was configured to run on startup, and print log information to a log file and laser breaks information to a statistics file.

**High-Level System Design:**

Hardware**:**

As the program begins two lasers are shining on two distinctive photodiodes. The photodiodes generate a voltage change that corresponds to a light being shone on it. When the light is blocked, the voltage drops; however, this change is too small to be picked up by the raspberry pi and therefore an op amp is used in the circuit to amplify the voltage obtained from the photodiodes. The other IC used is a differential comparator which serves the purpose of comparing the voltage value from the photodiode to the supply voltage in order to determine if the laser had been blocked. The entire circuit (apart from the lasers) are connected to a DC power supply since the output voltage of the pi is not high enough to support the operation of the IC’s. A capacitor also connects the inverting input of the op amp to the input of the comparator. The output from the comparator connects to a GPIO pin on the pi, which is read in the code as either a high or a low depending on whether the light is broken or not.

Software:

The overall system code consists of two main parts. The first is that which gets the status of the diode (broken/unbroken), with hysteresis implemented, and uses a state machine to determine which lasers have been broken or if an object is entering or exiting the room. The second main part of the program is responsible for reading a configuration file and outputting information to the correct log file and stats file. This consists of a state machine which reads the config file line by line and determines if a parameter is listed. It will then get the value and set the correct log file/stats file destination. The system also has a watchdog timer implemented which will restart if the system stops working.

**Software Design:**

Functions:

The Main function calls the following functions:

* readConfig()
* initializeGPIO()
* getTime()
* countLaserBreaks()
* errorMessage()
* outputMessage().

In addition the header files contain other functions that are called from the main such as;

* prerror()
* printf()
* fprintf()
* PRINT\_MSG()
* open()
* close().
* The countLaserBreaks() function calls the diodeStatusDebounce() function.
* The diodeStatusDebounce() function calls the laserDiodeStatus() function.

**Function Description:**

The code contained multiple functions including:

1. **GPIO\_Handle initializeGPIO()**
   1. Purpose: This function initializes the GPIO pins.
   2. Input: none
   3. Output: GPIO\_Handle type variable
2. **int laserDiodeStatus(GPIO\_Handle gpio, int diodeNumber)**
   1. Purpose: This function determines if the laser beam is shining at the photodiode or not. This method is called in the diodeStatusDebounce() function.
   2. Input:
      1. GPIO\_Handle
      2. Integer (1 or 2) which represents the diode number, for the sake of differentiating between the two.
   3. Output:
      1. 1 if the laser beam is shining at the diode
      2. 0 if the laser beam’s path is blocked from reaching the diode
      3. -1 if an error occurs.
3. **int diodeStatusDebounce(GPIO\_Handle gpio, int diodeNumber, int kMax)**
   1. Purpose: calls the diode status function multiple times and returns the correct diode status after debouncing with hysteresis. The hysteresis section (using state machine) receives the diode status multiple times until the status is a constant pattern of repeating 0’s or 1’s, meaning that the diode has completed the change of state (off to on, or on to off).
   2. Input:
      1. GPIO\_Handle
      2. Integer diodeNumber which is used to determine which diode is being examined
      3. Integer kMax which is used in hysteresis as a timeout so that it will stop when reaching a certain number of consistant repeated values
   3. Output:
      1. 1 if the laser beam is shining at the diode
      2. 0 if the laser beam’s path is blocked from reaching the diode
      3. -1 if an error occurs.

END OF HARDWARE DEPENDENT CODE

1. **void getTime(char\* buffer)**
   1. Purpose: to
   2. Input:
      1. Character array used to store the current time
   3. Output: none (void)
2. **void readConfig(FILE\* configFile, int\* timeout, char\* logFileName, char\* statsFileName)**
   1. Purpose: This function goes through the config file line by line, character by character, reading the valuable characters (with the help of a state machine) and using the pointers passed in to point to these grouped characters
   2. Input:
      1. configFile: a pointer to the file being read
      2. int\* timeout: a pointer to an integer, will be set in this function to a corresponding read value
      3. char\* logFileName: a pointer to the first element of a char array, will be set in this function to a corresponding read value
      4. char\* statsFileName: a pointer to the first element of a char array, will be set in this function to a corresponding read value
   3. Output: none (void)
3. **void outputMessage(int laser1Count, int laser2Count, int numberIn, int numberOut)**
   1. Purpose: outputs a message on the number of times each laser was broken and states how many objects have moved into and out of the room. (Not used in final version since the program outputs to a stats file.)
   2. Input:
      1. int laser1Count: how many times laser 1 is broken (the left laser)
      2. intlaser2Count: how many times laser 2 is broken (the right laser)
      3. int numberIn: the number of objects that moved into the room
      4. int numberOut will be the number of objects that moved out of the room
   3. Output: none (void)
4. **void errorMessage(int errorCode)**
   1. Purpose: Outputs an error message stating that an error occurred and includes the code corresponding to a specific error.
   2. Input:
      1. int errorCode: an integer which holds the error number
   3. Output: none (void)
5. **int countLaserBreaks(GPIO\_Handle gpio, const int kMax, int \*laser1Count, int \*laser2Count, int \*numIn, int \*numOut, FILE\* logFile, FILE\* statsFile, char\* programName, int watchdog)**
   1. Purpose: Determines (using state machine) how many times each laser was broken, and also how many objects had entered and exited over the given time interval.
   2. Input:
      1. GPIO\_Handle gpio
      2. const int kMax: passed in when calling the diodeStatusDebounce()
      3. int \*laser1Count: pointer to number of times laser1 is broken
      4. int \*laser2Count: pointer to number of times laser2 is broken
      5. int \*numIn: pointer to number of times object entered the room
      6. int \*numOut: pointer to number of times object entered the room
      7. FILE\* logFile: pointer to log file used
      8. FILE\* statsFile: pointer to stats file used
      9. char\* programName: pointer to first element of char array holding the name of the program
      10. int watchdog
   3. Output:
      1. 0 if no errors occurred
      2. -1 if error occurred

**System independent/dependent code:**

The following functions are hardware specific:

* int laserDiodeStatus(GPIO\_Handle gpio, int diodeNumber)
* **(State Machine)** int diodeStatusDebounce(GPIO\_Handle gpio, int diodeNumber, int kMax)

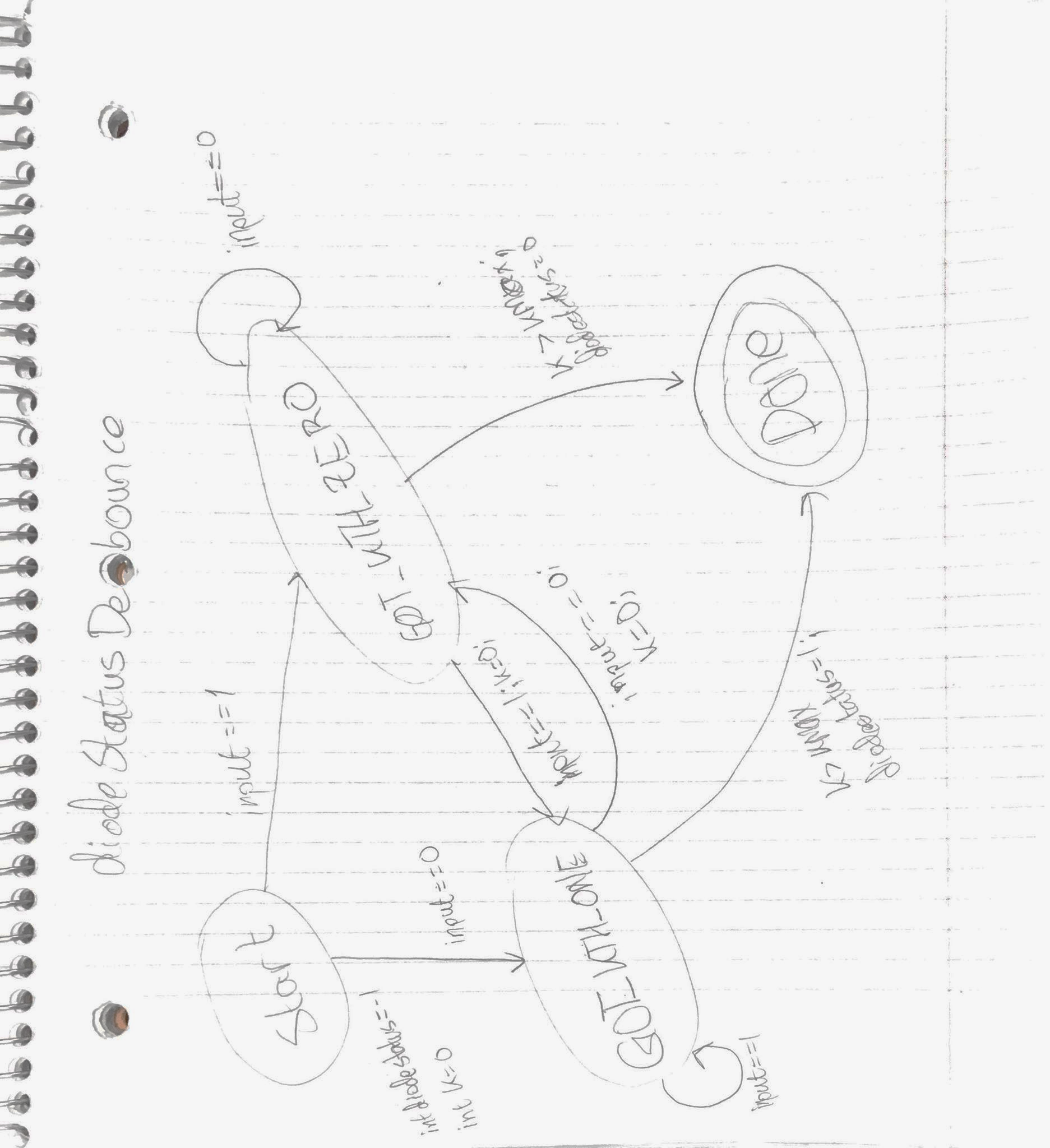
The following functions are independent of the Pi:

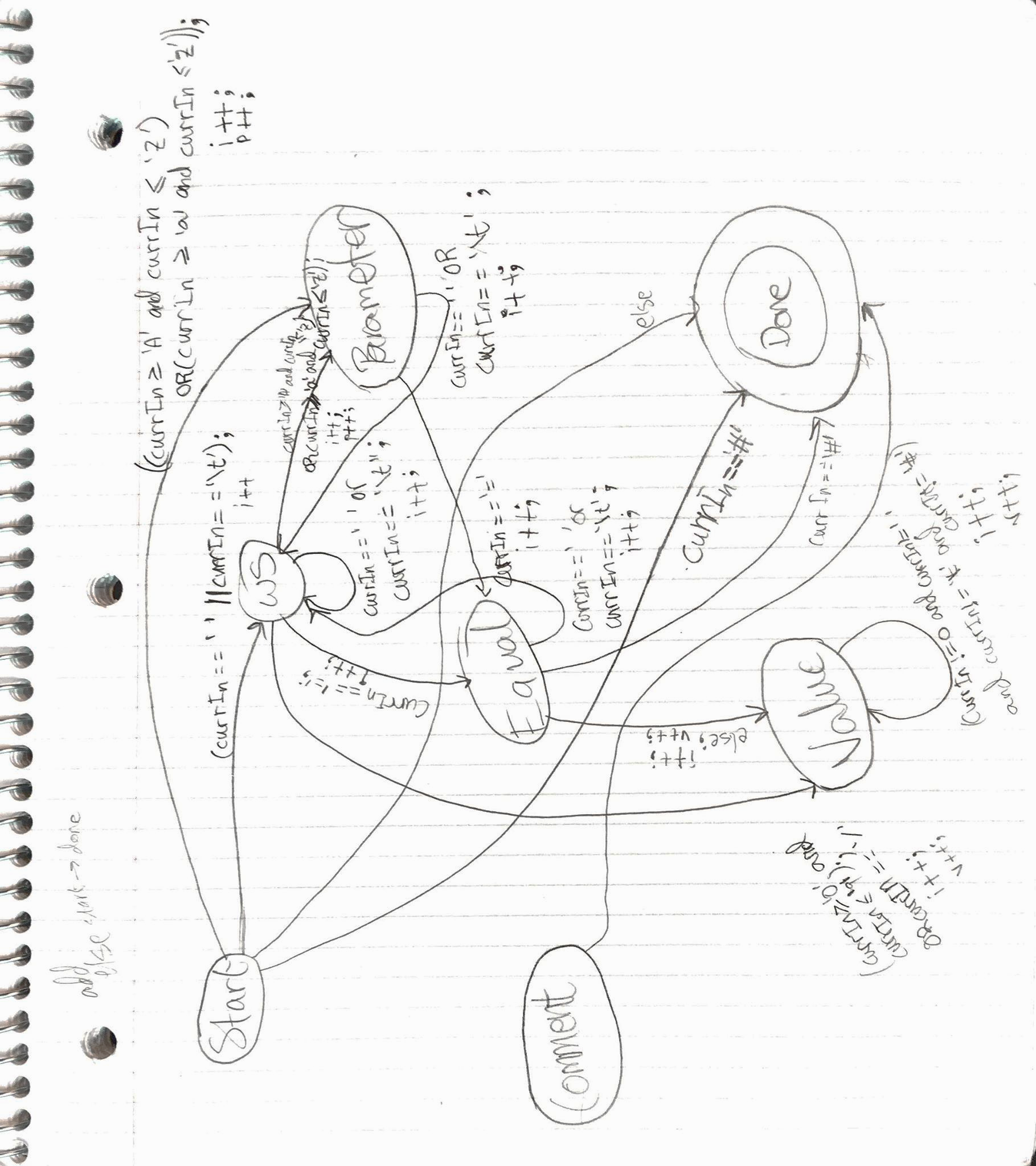
* void getTime(char\* buffer)
* **(State Machine)** void readConfig(FILE\* configFile, int\* timeout, char\* logFileName, char\* statsFileName)
* void outputMessage(int laser1Count, int laser2Count, int numberIn, int numberOut)
* void errorMessage(int errorCode)
* **(State Machine)** int countLaserBreaks(GPIO\_Handle gpio, const int kMax, int \*laser1Count, int \*laser2Count, int \*numIn, int \*numOut, FILE\* logFile, FILE\* statsFile, char\* programName, int watchdog)

**Separating Hardware independent/dependent code:**

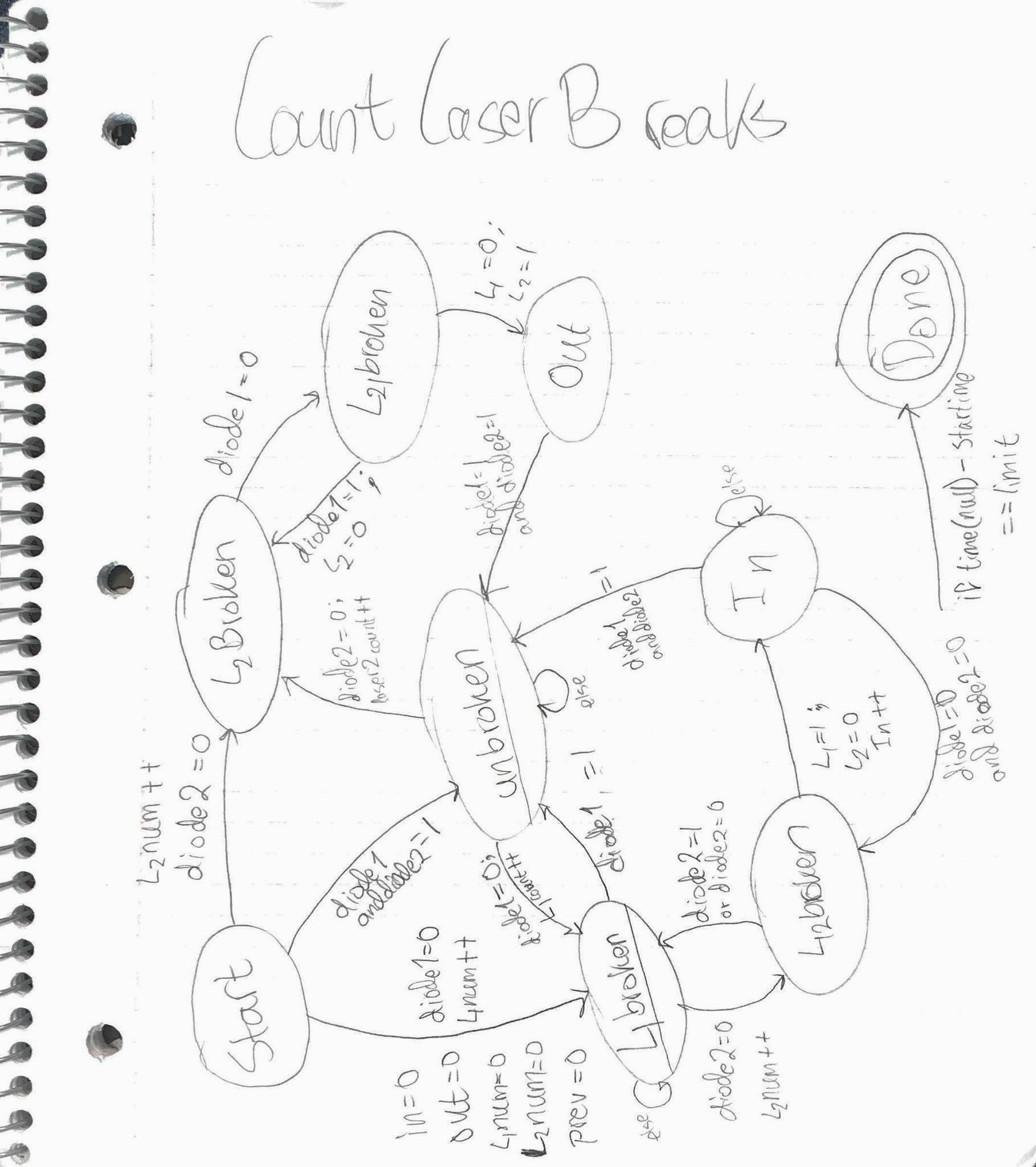
The hardware dependent and independent functions/state machines are separated chronologically such that the hardware dependant functions are implemented at the top of the code before the code independent of the Pi.

**State-machine diagrams**

****

****

readConfig()

****

**Logging infrastructure**

The logging section of this program records a message into a specified log file every time a checkpoint is passed. The logging is done by two Macros called PRINT\_MSG(), and PRINT\_STATS(). These simply take the information passed in and insert it into the defined appropriate file (logFile or statsFile). When called in countLaserBreaks(), PRINT\_MSG() logs when the watchdog was kicked and then logs the number of times an object entered/exited the room every 10 seconds into the file. In the main method, these macros log when the gpio pins were initialized, freed, when the watchdog was opened, limit was set, disabled and closed. It also logs when the program starts counting the laser interruptions. The logging points are justifiable because they log every significant occurrence. This makes it easy to debug the program afterward as you know exactly where and why the program stopped.

**Configuration**

Our system is configurable. The program reads the timeout value, log file name, and the stats file name from the configuration file identified. The timeout value is very easy to read as it consists of either one or two single digit integers. The log file name and stats file name are sequences of characters beginning with a slash which are pointed to by the corresponding pointers passed into the function.

**Watchdog**

The watchdog is opened after reading the config file and the timeout value read from the config file (if any) is set. Once the program enters the function which counts the number of laser breaks, the watchdog is kicked every 2 seconds to signal that the system continues to run correctly. The systems is ensured to run continuously since the program is set to run on startup. If the watchdog timer is invoked, the raspberry pi will restart, and the program will be run on startup. The same will occur if the system is shutdown normally. On startup the program will once again run and begin counting laser breaks.

**Testing**

In order to ensure each part of the overall system worked correctly, they were tested independently. Initially, following lab 3 we only had a program that would count the number of breaks and display them. We tested this code and made sure it and the hardware dependent code such as the GPIO and Diode Status functions worked correctly. This all worked correctly, however, the number of laser breaks was higher than it should have been since there was no debouncing. Once the basic laser program was determined to be working correctly, we worked towards creating a function which would debounce the diode input. We then tested the code with this new function implemented and determined a suitable parameter for debouncing, without introducing unnecessary delay. We were then able to successfully demo this for our lab 3 grade and knew that the laser setup worked correctly. In order to test the file reading and writing, we first made a program which consisted of only a function that would read the config file. We then ran this and had it print the results. Once this was working correctly, we added file output and tested it by outputting simple messages. Since this worked, we added the file read/write functions to the main laser program and configured it to print the necessary data (i.e. log file info & stats info). We also added a watchdog timer and tested it by adding a delay in the program where it would not ping it in time. We then tested the program as a whole and made sure that it was outputting the necessary material to the corresponding files, and that the watchdog timer was working correctly.

**ELR**

**Extras**

* Use of Macros within the code
* Debouncing photodiodes with hysteresis. This occurs in a function separate from the laser counting, which calls the diode status function directly. This allows for accurate debouncing with little delay.

**Limitations**

The only problem we had run into was after setting the watchdog timeout to 1 second (as instructed by the marking TA). This had bootlooped the pi and rendering it unusable, and new code could not have been updated to it as the timeout was so short. Nonetheless, this was not an issue of the code but an unrealistic timeout. Additionally, the issue was resolved by reflashing the pi and loading our code onto it again.

**Reflection:**

**- Reflect on your project: what would you do differently if you were starting it again?**

Better communication throughout development of the project as all two of the group members lived off campus and time together after class hours/weekends is limited. Furthermore the code was written in such a way that realtime mutual editing was not an efficient or viable option. In future services such as *Codeshare* could be used for development.

**Appendix**

**Source Code**

Our main source code file is “LaserCounts.c”. Also used were "gpiolib\_addr.h" and "gpiolib\_reg.h", provided in class to allow for access to the GPIO pins.

This is compiled on Ubuntu using the command *arm-linux-gnueabihf-gcc -o LaserCounts LaserCounts.c*

It is then transferred to the Pi using SCP *scp <username>@ecelinux4.uwaterloo.ca:LaserCounts.c*

On the Pi a nano file was created to allow the program to run as a service on startup.

The source file can be found under this link:

<https://github.com/dandobs/150_Embedded>

**Peer contribution:**

Daniel Dobre: LED / laser circuit, program design, watchdog, laser functions with hysteresis, testing section of report

Umar Hussain: LED circuit, IC circuit, Majority of Final Report, state machine diagrams

Daniel Arakcheev: Laser circuit, File Reading function, function purpose section of report