**Self-Directed Embedded Lab Project Final Report**

**Group 43**

**Joanne Baba, Devinn Doering, and Yrina Guarisma**

**Project Overview**:

The project attempts to create a system that acts as a speedometer. It is meant to assesses whether the speed of an object or person walking in a narrow corridor is acceptable based on a user inputted speed limit. If the system is running and measuring the speed a light will turn on to indicate this and if the person is going too fast for the speed limit a message is printed out to indicate the infraction and a light is meant to flash as well. This system is able to track the flow of traffic and record the system specific statistics that an analyst could use to identify whether a specific walkway is used with the appropriate number of people going at the intended speed.

The system measures the velocity of the object by calculating the time the object takes to pass between 2 lasers and their corresponding photodiodes through the system and determine whether the object is speeding. The object breaks the 2 lasers in one direction and, from measuring the time between the breaks and the distance between the lasers, the speed is calculated, compared to the speed limit and a response is printed out accordingly.

**High-Level System Design:**

Hardware Design

The system’s hardware design integrates two lasers with their corresponding photodiodes and senses whether the laser has been broken in order to record the time. In terms of hardware, two photodiode amplifier circuits are used to ensure the voltage reading across the photodiode is of a large enough quantity, so the raspberry pi’s GPIO pins detect it. This circuit implements an Op-Amp, two photodiodes, a 339 differential comparator and various resistors and capacitors. The system also implements two simple LED circuits that are meant to turn on the LED if certain conditions are met in the system. This circuit implements an LED and a resistor. Finally, this system integrates a simple laser circuit in which two lasers are connected directly to the power supply for the light to shine. The system is set up in the corridor as follows. The photodiodes are placed in their holding bases and placed on one side of the corridor about two meters apart with the voltage amplifying circuit in the middle connected to the power supply and the raspberry pi. On the opposite side of the hallway the lasers are set up in a similar fashion such that the laser light shines directly onto the photodiode.

Hardware Components:

* 2x Laser Diodes: When given 2.4V -5.2V the laser diode will turn on and create a concentrated beam of red light that will shine across the corridor onto the photodiode. When a person passes through it will break this connection and can therefore record a time of break.
* 2x Photodiodes: A photodiode creates an electric current and voltage when it detects light. This helps us create a light activated circuit where we can sense if a voltage is created which indicates the laser is shining onto it and if it does not create this voltage it indicates the laser has been broken.
* Op-Amp(Operational Amplifier): The Op-Amp amplifies electric signals. It is used to amplify the electric signals created by the photodiode so that the Raspberry Pi can detect them as inputs.
* 339 Differential Comparator: The 339 differential comparator compares an input voltage to a reference voltage. In this circuit it is used to compare the voltage generated by the photodiode to the reference voltage.
* Power Supply or 9V battery (DC power supply): A DC power supply, like the 9V battery, supplies current flowing in only one direction. The Operational Amplifier and the 339 Differential Comparator require direct current to function properly, so one DC power supply must be used.
* Resistors (10k, 1M, 16k, 12k, 1k, 2.4k, 220): A resistor is used to reduce the current flowing to an electric component. Resistors are used throughout both the photodiode amplifier circuit and the LED circuit in different strengths to ensure the proper amount of current is used for each component.
* Capacitors (10u, 100p, 0.1u): A capacitor stores electric charge in an electric field. This device is used in the photodiode amplifying circuit to store electric charge and release it in order for the other components to function properly.
* 2x LEDs: A light emitting diode emits light when current passes through it. It is used to emit light and indicate certain conditions have been met in the system.
* Wire(Male to Male, and Male to Female): Wire is metal conductor surrounded by an insulator. It is used to carry electric current throughout the circuit and ensure all connections are made.
* Two photodiode holding bases and two Laser holding bases: These two bases are made of fibreboard to be able to hold the photodiodes and lasers horizontally. In this way they can be set up in any location easily.
* 2x Breadboard: This is a board that connects electronic components together temporarily as needed. It is used for prototyping electric circuits and in this case it is used to ensure connections for all the electronic components.

Software Design

The software system implements a state machine and a watchdog. The state machine in this system has been implemented to identify the person or object’s location in the hallway and record the times the lasers have been broken. It is then used to make calculations to be able to acquire a value for the velocity of the object and compare it to the user inputted speed limit value in the log file. These calculations are then recorded in a stats file. The watchdog is a system in the raspberry pi that checks that the program is running properly. If the program is not running properly it will reboot the system and attempt to run it again. A configuration file is implemented to set up the environment for the program. It initializes all required parameters based on the user.

Software Components:

* State Machine: A state machine implements a set number of conditions in which the program can be and depending on the conditions and inputs is in one of the states. In this project the state machine is used to identify the different states of the hallway. If there was a person entering, exiting or within the hallway in order to record the time.
* Watchdog: A watchdog is used to identify and reboot the system if the program is not functioning properly. The watchdog from the raspberry pi was implemented to ensure the service ran consistently.
* Configuration File: A config file is used to initialize the values or parameters for a program. It is used in the system to set up the speed limit, the distance between the lasers, the log file name, the watchdog timeout time, stats file name, and the frequency at which stats get outputted.
* Stats File: The stats file records different statistics measured by the system as a certain amount of time passes to be used in order to analyze the functionality of the corridor and the speed limit used.
* Log file: The log file records the events that occurred in our system. Any error message or message printed as the object or person’s speed was measured is recorded in the log file to keep track of the system.

**Software Design:**

For the software portion of our self-directed project we used a familiar structure consisting of multiple functions that are both system dependent and system independent and a main function. We isolated our state machine and therefore most of our code into a function that is called measureSpeed. The majority of this function is contained within a while(1) loop with the intention to run continuously. We understand that, by convention, while(1) loops are generally not accepted however in this case, the program is to continue to run whenever the pi is plugged in and we think putting our state machine in a while(1) loop illustrates this intention nicely. In our main function, we take care of initializing the gpio, reading the config file and opening the stats and log file, essentially taking care of everything that needs to occur before we call the measureSpeed function.

In addition, an important thing to note is that we use #ifndef RUN\_AS\_SERVICE to enclose all printf statements. We chose to do this so that, when we want to run our program as a service, we can compile the function with -D RUN\_AS\_SERVICE so the the printf statements will not print when the program is running as a service. This minimizes strain on the Pi’s processor and prevents any accidental crashes from happening due to overfilling the print buffer file.

In our code, we have 9 functions with the functionalities as follows:

**GPIO\_Handle initializeGPIO(): Defined on line 240**

This function initializes the GPIO pins. It returns a GPIO\_Handle which will be null if there are any issues with the initialization. This function takes no parameters.

**int laserDiodeStatus(GPIO\_Handle gpio, int diodeNumber): Defined on line 256**

This function returns the status of a input pin on the pi. If that pin is receiving a voltage, it returns a zero and if the pin is not receiving a voltage, the function returns a one. Since we use this function in conjunction with lasers, we can translate this functions output to mean that when the photodiode is receiving a laser beam, the function returns a one and when the photodiode is not receiving a laser beam, the function returns a zero. It takes two parameters, the GPIO\_Handle that it will be checking and the int value of the diodeNumber to check (either diode 1 or 2). The diode numbers given are then processed to check the correct pin numbers based on a definition of the diode pin numbers given at the top of the code. (Lines 43 & 44).

**void setToOutput(GPIO\_handle gpio, int pinNumber): Defined on line 288**

This function will set the given gpio pin to be an output pin on the pi. It essentially bitshifts the correct bit to be a 1 to change the 3 bits corresponding to that pin to be 001 instead of the default 000. It calculates the gpio register to access based on the 10's place of the pinNumber given. It takes two parameters, the GPIO\_Handle of the pin it will be changing and the pinNumber that is to be set to an output.

**void outputOn(GPIO\_Handle gpio, int pinNumber): Defined on line 331**

This function will tell the pi to output a voltage on the given pin. It essentially bitshifts the correct bit to be 1. It takes two parameters, the GPIO\_Handle of the pin it will be changing and the pinNumber that is to be set to on.

**void outputOff(GPIO\_Handle gpio, int pinNumber): Defined on line 337**

This function will tell the pi to stop outputting a voltage on the given pin. It essentially bit shifts the correct bit to be 0. It takes two parameters, the GPIO\_Handle of the pin it will be changing and the pinNumber that is to be set to off.

**void getTime(char\* buffer): Defined on line 342**

This function simply retrieves the current time using the gettimeofday() function and then converts this time into month/day/year/time format and sets the string given to be this format of the time. It takes one parameter which is a pointer to a char array and this parameter is where the resultant time is to be stored.

**void readConfig(FILE\* configFile, int\* timeout, char\* logFileName, char\* statsFileName, int\* statsFrequency, int\* speedLimit, int\* distanceBetweenLasers): Defined on line 363**

This function reads the configFile for the program and receives the user-changable values that are in the configFile. It closely follows the readConfig function that was given for the lab 4 sample code. It scans through the config file by looking specifically for lines not preceded with a "#". When it finds a line that does not have a "#", it goes down that line until it finds an equals sign and then copies the value of the number after the equals sign. There are 7 parameters passed to this function. The first parameter, configFile is the name of the config file that this function needs to look through. The remaining 6 parameters are the values which we are fetching from the config file and are passed to the function as pointers to variables created in main().

**void computeStats(float\* maxSpeed, float\* minSpeed, float\* averageSpeed, float objectSpeeds[], int peoplePassedThrough): Defined on line 490**

This function computes statistical information about the array objectSpeeds. It goes through the array and finds the maximum speed and the minimum speed as well as finding the average by summing the speeds and dividing by the number of elements in the array. This function takes five parameters. The first three parameters are pointers to floats and are used to hold the calculated maximum, minimum and average speeds. These parameters are declared in the measureSpeed function. The fourth parameter is the float array that contains the speeds of the people passing through the hallway. We use the fifth parameter, peoplePassedThrough to calculate both the average and to know how far to loop through the array. This function is completely system independent.

**void measureSpeed(GPIO\_Handle gpio, int watchdog, const int statsFrequency, const int speedLimit, const int distance, FILE\* logFile, FILE\* statsFile): Defined on line 511**

This function is the main function in our program, and it contains the state machine and logic which measures the speed of objects passing through the hallway and outputs the results to the stats file. Additionally, this function continues pinging the watchdog to ensure that the watchdog is never allowed to reboot the system and it also outputs messages to the log file accordingly. This function also controls the LEDs that are used to indicate to the user the state of the program. This function takes 7 parameters. The first parameter, gpio is the GPIO\_Handle of all of pins being used. The second parameter, watchdog is required to ping the watchdog. The remaining parameters are all the values that were previously read from the config file and will be used to modify how the function works. The two FILE\* parameters logFile and statsFile are the pointers to the corresponding files and they are where the log messages and statistic messages are printed to. The statsFrequency determines how often statistical messages are printed to the stats file. Distance is the distance between the two lasers, set by the user in the config file. Finally, speedLimit is the maximum speed, in m/s that a person travelling through the hall can travel before a warning is issued. More about this function will be included but this is a quick synopsis of how it works and what the parameters do.

In addition to the functions listed above, our code has two macros, PRINT\_MSG and PRINT\_VALUE. They both serve similar purposes, to print a message into a file but the PRINT\_MSG macro prints a string and PRINT\_VALUE prints a value. In addition to the main message, PRINT\_MSG will print the name of the file, the time, and the severity of the message to the file.

The functions are called in the order as follows:

main() directly calls readConfig, initializeGPIO, setToOutPut and measureSpeed. It calls setToOutput twice to set both LED pins to outputs. In addition to these functions, it also calls the macro PRINT\_MSG multiple times to print messages to the log file.

intitializeGPIO, laserDiodeStatus, setToOutput, outputOn, outputOff, and readConfig do not call any other user created functions.

measureSpeed calls outputOn and outputOff as necessary to turn on/off the LEDs. It also calls the PRINT\_MSG and PRINT\_VALUE macros as necessary to print messages to either the log file or the config file. It also calls the laserDiodeStatus function twice for each iteration of the while loop to update the state of both photodiodes. This is critical to call laserDiodeStatus this often to ensure maximum accuracy in photodiode readings. It also calls computeStatistics to figure out the maximum, minimum and average speeds in the last interval of time. It calls this function at an interval as determined by statsFrequency.

**System Dependent Code:**

The functions that would be considered system dependent would be initializeGPIO, laserDiodeStatus, setToOutput, outputOn and outputOff. The reason why these would be considered system dependent is because they interact directly with the Raspberry Pi GPIO pins and control them or read from them to interact appropriately with the rest of the program. These functions are all very nicely isolated from the rest of the code.

**System Independent Code:**

The functions that would be considered system independent would be readConfig, computeStatistics and measureSpeed. Specifically, the readConfig and computeStatistics functions could operate in an environment entirely away from the Raspberry Pi and would be able to successfully read from a file in most situations. For instance, the computeStatistics function only performs basic statistical analysis on an array and could be directly used in any other situation where the maximum, minimum and average values need to be calculated over an array. Additionally, the measureSpeed function could operate outside of the enviroment hooked up to the Raspberry Pi with only minor modifications. Currently, the measureSpeed function does call some of the system dependent functions and in order to make measureSpeed truly system independent, these function calls would need to be replaced with appropriate function calls or other substitutes.

These functions are all isolated from each other quite well which is very good in order to have easily debuggable code. One thing that could possibly separate more is the measureSpeed function which is quite large. We could not easily find a way to effectively shrink the measureSpeed function as most of this function is one state machine which for obvious reasons, cannot be broken up further.

**State Machine:**

The system implements a state machine in order to identify whether a person has entered and exited the hallway by means of reading the photodiode status in a state machine. The attached state machine demonstrates the states and the transitions between them. These transitions are dependent only on the photodiode status, thus the attached diagram includes only how the status impacts the functionality of the state of the machine. However, some functionality does not affect the states and was excluded from the diagram. This functionality impacts measurements recorded for time, error messages that are printed, and various counters for the number of people in the corridor are incremented in various states. The exact functionality can be seen in the source file SelfDirected.c.

**Logging:**

In our program we log into two different files – a log file and a stats file. The log file receives more generic messages and information such as when the config file is read, the watchdog is enabled and other information that is relevant to the program but not specifically to what our program tests for. The stats file on the other hand receives statistical information related to the speed of people passing through the hallway. Statistics are printed to the log file at the frequency that is determined in the config file (by the statsFrequency variable).

Messages are printed to the log file using both the PRINT\_MSG and PRINT\_VALUE macros and also just by using fprintf() if the message is more specific, requiring variables to be printed into the file in an order more specific than the PRINT\_MSG macro does. All messages are printed to the files in the following format:

TIME : function name : severity of message : message

Where “function name” is the name of the function (argv[0]) if PRINT\_MSG is called in main() and, if PRINT\_MSG is called in the function measureSpeed, then “function name” is “measureSpeed”. Severity of message is one of either debug, info, warning, error, or critical depending on the severity of the message. All these severities are defined as strings at the top of the program. Ex #define SEVERITY\_INFO “info”

We log into the log file whenever a notable event occurs and only into the stats file at intervals that are determined by statsFrequency. A list of notable events that we print to the log file are as follows:

* GPIO pins being initialized (or failing to initialize)
* The watchdog being initialized (or failing to initialize)
* The watchdog timer being set
* GPIO pins being set to outputs for purpose of powering LEDs
* If, when initially running the program, the lasers are not connected to the photodiodes
* When both lasers are connected to their respected photodiodes
* If the speed limit read from the config file is invalid
* If someone/something is blocking a laser
* If someone/something is blocking the hallway (if they enter but do not leave)
* If someone enters the hallway and leaves in the same direction
* If someone speeds through the hallway
* If someone passes through the hallway under the speed limit

**Configuration:**

The program is designed to allow the user to change multiple things about how the program operates by changing values in the configuration file. The things that we allow to be configurable are the location and name of the stats file, the location and name of the log file, the duration that the watchdog requires to be pinged, the frequency that statistics are printed into the stats file, the speed limit of the hallway and finally, the distance between the two laser sensors. The reason we decided to make this level of customization was because we felt that it allowed the user to change exactly how the program runs without breaking away from the fundamental purpose of the program. The configuration file is just a standard text document that can be edited by the user.

The values of some of the afore mentioned variables have defaults that are defined at the top of the program. We chose to set the default stats frequency to 60 seconds, the default speed limit to 1 metre per second and the default distance between the lasers to be 3 metres. We thought these defaults best captured the “default” environment which the program would be run in.

**Watchdog:**

The watchdog is initialized in main and continues to be written to throughout the program. In our primary function, measureSpeed, we write to the watchdog file every iteration of the while loop which should be approximately every 1/1000th of a second (we kick the ol’ dog often). This ensures that we are never even remotely close to allowing the watchdog to reboot the system under normal conditions so the watchdog should work as intended. We never close the watchdog by writing “V” to the file since we want the program to continue to run whenever the Pi is booted.

Whenever the system is booted up, the program is automatically run as a service to ensure that people are always being monitored as they move through the hallway. This is done by creating a service file to run the executable for our program upon start-up.

We set the watchdog timer to a default of 15 seconds but then change it to the user configurable value in the configuration file

**Testing:**

To test that our system works properly, we checked whether the first LED turned on at start-up. This would indicate that our program was also running on start-up. We also checked various scenarios of people entering the hallway. We checked that the program would detect when a person was blocking either laser and would not measure a speed if a person were to enter the hallway then exit the hallway through the same laser so that they are only allowed to move in one direction. We checked that the second LED turned on when a person was in the hallway and turned off when a person exited the hallway.

We tested that our program output a speed when a person walked through the hallway from both directions and tested this with various speeds. We moved slowly between the lasers, at a regular speed, and broke them extremely fast using a person at both ends so that they could break the lasers almost instantaneously. All these scenarios outputted the expected speeds. We also checked the speeds were accurate by checking that the second LED flashes when a person moves faster than the speed limit.

We also checked that the stats file was printed at the expected frequency, and that the information was accurate. We also checked that the log file was printing accurate information and that the stats file and log file were created under the correct name.

We also tested the watchdog in multiple different scenarios. First, we ran the program under command line and stopped the program (control + C) to test the watchdog would reboot the system. Then, we ran the program as a service and used systemctl to stop the service running to test the watchdog. In both cases, the watchdog worked as expected and rebooted the Pi which is excellent.

**Extras, Limitations, Reflection:**

Extras:

In addition to what we promised in our proposal, our project can detect when a person is blocking the hallway, as well as measure the speed of a person moving in the hallway in both directions. It also includes two LEDs, one of which turns on when the project is running. The other LED is only on when a person is in the hallway, and flashes when the person is speeding. These extra implementations helped us to debug and test our code and provides visual indicators to the users.

Limitations:

One of the limitations in our project is gauging when a person has entered the hallway. For the most part we can accurately assess this, however if a person steps in and breaks the laser and then unbreaks the laser by stepping out, we assume that person has entered the hallway. This is due to our project consisting of single lasers at each end of the hallway as opposed to two lasers at each end. This is an issue because two lasers would be able to detect a proper entering by a double break of both lasers, whereas single lasers cannot. Currently, we assume that a person has entered the hallway anytime a laser is broken, which is problematic. Another limitation is that time is only measured in integer values, meaning that for shorter periods of time such as when a person moves very fast, the outputted speed is inaccurate.

Reflection:

If we started the project again, we would use double lasers at each end of the hallway in order to accurately detect when a person has entered the hallway. This would eliminate the limitation we currently have and make our project more robust. We would also make the laser stands more stable so that the lasers are easier to set up, as the connection is not reliable and difficult to align. In terms of project management, in the future we would attempt to use Git in order to be able to work on one .c file for the project instead of constantly having to ensure we had the most recent version of the code to add to. Also, in the future we should attempt to implement a more iterative process for dividing the software tasks. Instead of dividing the project into large portions that need to be completed, the software should be subdivided into its functions that each member can work on individually and append to one git file. Overall, in terms of future improvements to the idea, we would like to include more features like a speed classification system and with more lasers be able to add more measurements like accelerations or being able to test multiple objects in the hallway.

**Appendix:**

Source Code:

The Source Code files included are selfDirected.cpp, selfDirected.cfg, and all Laser GPIO pin configuration files provided in the lab: gpiolib\_addr.h, gpiolib\_reg.h, and gpiolib\_reg.c. (The three GPIO files have not been included). Other files required for functionality are stdint.h, stdio.h,(for the printf()), fcntl.h, linux/watchdog.h(needed for watchdog), unistd.h(for sleep), sys/ioctl.h(for ioctl function), stdlib.h, time.h(for time\_t and time()), sys/time.h(gettimeofday()). These files have not been included as many are libraries included in the language. The .cpp file is compiled on a Virtual Machine Linux environment and the resulting executable is sent to the ecelinux4 server. When the pi is connected, Putty is then used to transfer the executable from the server to the raspberry pi to be able to be run. The .cfg file is created directly on a text editor and then transferred to the ecelinux4 server. Then similarly, putty is used to transfer the file from the server to the raspberry pi.

Peer Contribution:

The idea for the project was a collaboration from all three group members. During brainstorm sessions all three members came up with potential ideas. The idea to measure velocity was initially suggested by Devinn and adapted into a speedometer by Yrina and Joanne. The idea of using lasers and photodiodes as sensors for the hallway was decided by Yrina and Joanne who also built the circuits and tested them until fully functional. Yrina and Joanne also provided any external materials necessary like electrical tape and masking tape. The laser state machine logic and watchdog implementation were designed, implemented and coded by Devinn. The fibreboard bases for the lasers and photodiodes were built using CAD and designed by all three group members. Testing and implementation of the code was completed by all three group members. The report was a collaboration between all three members.