Embedded Lab Report

Group 1: Kishen, Matthew, Saif

November 30, 2018

Contents

Chapter 1

Introduction

1.1 Project Overview

This project is an embedded system that measures the number of objects entering and exiting a room. It uses a Raspberry Pi Zero along with 2 lasers to achieve this.

1.2 High Level System Design

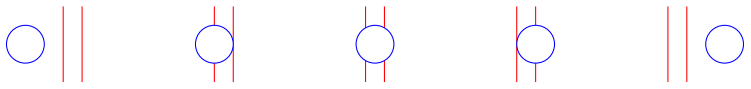
1.2.1 Software

The code is written in C, compiled with the arm-linux-gnueabihf toolchain, then copied to the Raspberry Pi using a provided file server. The main part of the code follows a state-machine setup to take inputs from the laser diodes and perform the measurements (accounting for partial entrances, etc.).

1.2.2 Hardware

There are two lasers each aimed at a laser diode. While laser light falls on these diodes, a signal of '1' is sent to the Raspberry Pi's GPIO pin (the signal that the diode receives is magnified through a system of op-amps and comparators).

When the laser beam is broken, the GPIO reads a '0'. By having two lasers, we can follow these signals through a predictable procedure for entrances and exits. The following figure shows the breakage steps for entrance/exit (depending on the point of view).



Chapter 2

Software Design

2.1 Overview

The code is based on the provided gpiolib, which handles the memory mapping and basic register reading/writing functions. There are also functions within our main program, as well as an extension to the gpiolib that we created called `gpiolib_pinfuncs` . A diagram of function calls follows:

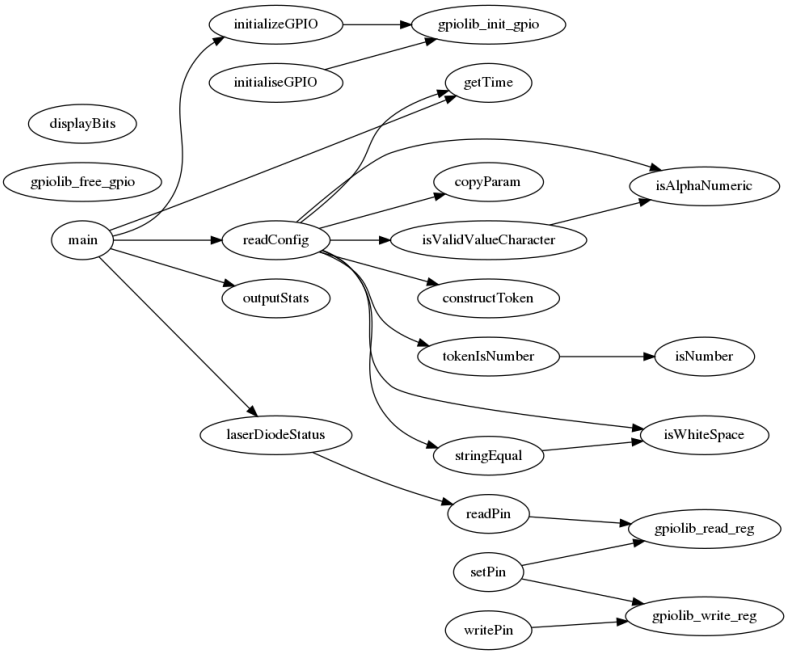


Figure 2.1: Function call graph generated with egypt.

2.2 Function Descriptions

2.2.1 Main Program Functions

laserDiodeStatus

Purpose: Read the signal from the laser diodes

Inputs: A gpio handle to access the appropriate registers, and an integer (1 or 2) to decide which diode to read from

Outputs: An integer (0 or 1) representing the current status of the given diode.

readConfig

Purpose: Read the configuration file and set the watchdog timeout, as well as the locations of the log and statistics files.

Inputs: Pointers to the values for the parameters to be set (timeout, logFileName, statsFile)

Outputs: None, but the parameters are set within the function based on the config file

getTime

Purpose: Get the current datestamp and set it as a string for stats/logfile purposes

Inputs: A pointer to the string we want to set the datestamp to

Outputs: None, but the given string is set to the datestamp

isAlphaNumeric

Purpose: Check if a character is an alphanumeric value

Inputs: A character

Output: Boolean value representing whether the character is alphanumeric or not

isWhiteSpace

Purpose: Check if a character is whitespace

Inputs: A character

Output: Boolean value representing whether the character is whitespace or not

isNumber

Purpose: Check if a character is a number

Inputs: A character

Output: Boolean value representing whether the character is a number or not

isValidValueCharacter

Purpose: Check if a character is something that is valid for a parameter value (within the context of the config file)

Inputs: A character

Output: Boolean value representing whether the character is a number or not

tokenIsNumber

Purpose: After tokenizing, check if the string is a pure number

Inputs: A character array

Output: Boolean value representing whether the array has only numbers in it

constructToken

Purpose: Given certain values, construct a char array for the token (within the context of the config file)

Inputs: A buffer representing a line in the config file, the start index of the token, and its length

Output: A character array containing the token

copyParam

Purpose: Copy over the value of a parameter into the given pointer (in the context of the config file)

Inputs: A value token, its length, and the pointer we want to set to it

Output: None, but within the function the 'to' pointer is set to the given value

stringEqual

Purpose: Compare two strings, used to check if a token is equal to a parameter we want to set in the logfile (see line 544, 582, and 611 in LaserLab3.c)

Inputs: Two char arrays

Output: Boolean value representing whether the two strings are the same

outputStats

Purpose: Write the statistics to the stats file

Inputs: Stats file location, and the statistics values

Output: None, but overwrites the stats file to reflect the updates

2.2.2 gpiolib_pinfuncs Functions

setPin

Purpose: Set a pin to either output or input

Inputs: A gpio handle, a pin number, and the mode we are setting it to

Output: None, but writes to the appropriate gpio register

writePin

Purpose: Write a 1 or a 0 to the given pin

Inputs: A gpio handle, pin number, and the value we want to write (1 or 0)

Output: None, but writes to the appropriate gpio register

readPin

Purpose: Read a 1 or a 0 from the given pin

Inputs: A gpio handle, and a pin number

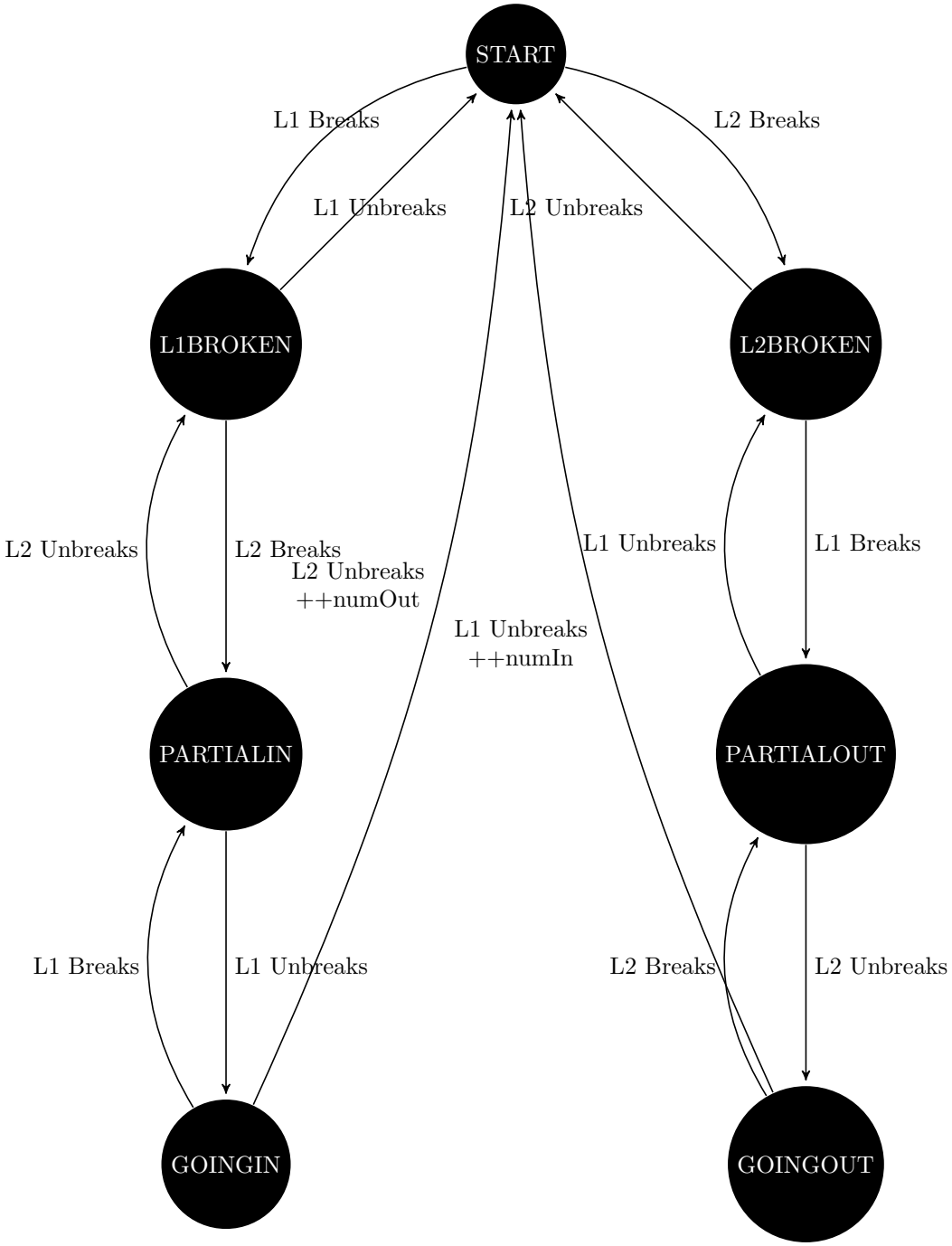
Output: The value received from the given pin

2.3 System Dependence

The functions that are specific to the Raspberry Pi are the functions within gpiolib, for example those listed int 2.2.2. Everything else is separate from the hardware.

Given substitutes for the files/inputs/outputs, the rest of the program could function as normal. As per the diagram 2.1, all the connections between the main file and gpiolib are the ones that would need to be replaced to run this code on another system.

2.4 The Main State Machine



2.5 Configuration File

The system allows for a configuration file that can set three parameters:

`WATCHDOG_TIMEOUT`: The timeout value for the watchdog

`LOGFILE`: The file path for the log file

`STATSFILE`: The file path for the stats file

These are set by editing the configuration file located at `/home/pi/LaserLab.cfg`. To set a parameter, simply write the name (exactly as it appears in the list) followed by an equal sign, then the parameter value. Comments can appear in the log file and are preceded by a `#`. Invalid/absent values for these parameters are replaced with their defaults as follows:

`WATCHDOG_TIMEOUT`: 10 seconds

`LOGFILE`: `/home/pi/LaserLab.log`

`STATSFILE`: `/home/pi/LaserLab.stats`

2.6 Log File

The log file line format is as follows:
`==<PID>==` — `<Program Name>` — `: <Severity>: <Message> (<Datestamp>)`
The log points are as follows:

1) When the watchdog is first opened
2) Every `STATUSUPDATE = 15` seconds with the number of times the watchdog has been pinged, and the number of times the stats file was updated
3) Whenever a laser breaks, output what happened and what state we are going into to the log file
4) When the program ends, to signify that shutdown went okay

The log file is written to using the `PRINT_MSG` macro to keep the code cleaner and allow us to change the line format in the future.

2.7 Watchdog

After reading the configuration file, the `timout` is set to a value between 1 and 15 seconds. The watchdog is pinged whenever `difftime(time(0), watchdogPingTime) > timeout/2` (i.e. every half-timeout) to prevent the watchdog from killing the process. On a reboot, the process starts again and continues to update the statistics file (since it is symlinked as a service running in the background).

Chapter 3

Further Information

3.1 Testing

Throughout the process of developing the project, we tested the functionality by simply moving an object through the laser setup, recording the number of laser breaks, entrances, and exits, and then compared this to the output in the statistics file. We also added various `fprintf` statements to make sure things were running as expected, though these are commented out in the final release.

3.2 Miscellaneous

3.2.1 Extras

As stated earlier, we added `gpiolib\_pinfuncs` that includes some functions (2.2.2) that make writing/reading to pins easier, as the register bitfield calculations don't need to be done by hand. This allows for flexibility when choosing the pins for I/O.

3.2.2 Limitations

In its current state, there is no immediate feedback for when the program is running. Adding an LED or physical display to count the items would make the user experience better, rather than having them check the log/stats file to make sure everything is right.

3.2.3 Reflection

As stated previously, We would add physical displays to reflect the status of the system to make debugging easier, and also make the system more of an embedded system. We would also create a physical device to keep the lasers/diodes in place as we found that during the demo they would constantly be a little bit off their marks.

Chapter 4

Appendix

4.1 Source Code

The source code, along with the call graph and this document, are available on GitLab. This includes the gpiolib files, the main file `LaserLab3.c`, and the gpiolib extension `gpiolib_reg.c`.

4.2 Peer Contributions

Member	**Contributions**
Kishen	State machine development Circuit debugging
Matthew	Circuit building State machine debugging
Saif	Config/Log/Stats functionality Software debugging