ECE 150 – Fundamentals of Programming
Self-Directed Embedded Systems Project
Final Report

Group Name: Three Little Pigs Went Out One Day
Group Number: 35
Project Name: Digital Piggy Bank

Project Overview

The main aim of our project is to construct a "digital piggy bank". The project should collect coins through a narrow slit. A laser will shine at a photodiode and the coin will break the laser beam. The program then computes the duration the laser beam was block, which would be directly related to the diameter of the coin. The program should then use the diameter of the coin to work out which coin it is. The program should keep track of the total number of each coin and the total balance of the piggy bank.

High Level System Design

Hardware

Project Components:

- Raspberry Pi Zero
- Laser
- Photodiode
- TL082CP Operational Amplifier
- LM339AN Comparator
- 3D Printed Coin Slide

When the laser beam is broken and the laser beam shines on the photodiode, the photodiode allows current to flow through it and so there is a very small potential difference across it (about +0.7V). The Pi is unable to detect voltages this low, so an operational amplifier is used. The TL082CP operational amplifier amplifies the voltage to a higher value. This voltage is then read by the comparator. The LM339AN comparator compares that voltage to a reference voltage and converts it to a digital signal. To elaborate, if the amplified voltage is above the reference voltage, the comparator outputs 3.3V to the Pi, if not the comparator outputs 0V. This is detected as an input by pin 4 of the Pi. The Pi uses the input value of pin 4 to judge if the laser beam is broken or not.

There are also two buttons. The reset and cancel buttons are connected to pins 2 and 3 respectively. There are pulled up using 1000kΩ resistors to avoid button bouncing and noise interruption. The functions of these buttons are described in more detail later in this document.

Software

The program uses while loops to continuously monitor the state of the laser beam and uses that to identify the coin. When it detects a coin and identifies it, it adds to the balance of the piggy bank and keeps count of the number of the same coin values there are. For example, when it detects a dime, it increments the total balance by 0.10 and increments the dime count

by 1. The program also prints all of these in the logfile. The program also makes use of config files and watchdog timers. The contents of the program are described in more detail later in this document.

Physics

The physical logic behind this is that the velocity of a coin dropped from a certain height is only dependent on the initial height and the gravitational acceleration, and independent of the mass or size of the coin.

$$Loss\ in\ Gravitational\ Potential\ Energy = Gain\ in\ Kinetic\ Energy$$

$$mgh = \frac{1}{2}mv^2$$

$$v = \sqrt[2]{2gh}$$

Therefore, we know coins of all sizes and masses will slide down with approximately the same velocity. Because all the coins have the same velocity, the time taken for the coin to pass the laser will depend only on the diameter of the coin. (Theoretically this may not precisely be the case as there are other factors not considered. Friction will act on all coins and slow down the lighter coins to a greater extent than the heavier coins. The moment of inertia of the coins may also affect the velocity of the coins as it's rolling. However, these effects are likely to have very minimal impact on the velocity and should not make a noticeable difference for such a miniature, simple project).

Software Design

Overall Structure

As mentioned before, the Pi executes a continuous while loop, through which the Pi checks the input of pin 4, i.e. the state of the photodiode. As long as the laser beam is not broken, the Pi only keeps running the while loop. The moment the laser beam is broken, the Pi exits that while loop and enters another one which loops as long as the laser beam remains broken. A very short usleep() function is called in every cycle of the while loop and a counter variable is incremented every time the while loop is re-entered. When the laser beam is unbroken again, the program exits the while loop. The counter variable gives the number of times the while loop was re-entered and is directly related to the duration the beam was broken. This is directly related to the diameter of the coin. The counter is therefore used to determine the coin's value (since every coin has a different diameter). The detected coins and the total balance are both printed to the logfile in real time.

Logging Infrastructure

In order to write relevant information to the log file, we are utilizing three macros in our program: PRINT_MSG (which was obtained from the Lab 4 Sample code), PRINT_MSG_COIN (which is just a modification of PRINT_MSG so that we can print the balance of the coin that passed), and PRINT_TOTAL_BALANCE (which is also a

modification of PRINT_MSG so that we can print the total balance of in the piggy bank). We utilize the PRINT_MSG macro whenever we want to do any general-purpose logging (for example, saying that the watchdog was enabled) and utilize the other two macros whenever we need to log which coin has passed through the laser and to state what is the current balance of the piggy bank. These macros utilize the fprintf() function, which allows us to write to the log file.

In our program, we have implemented multiple logging points:

- When the GPIO Pins are initialized
- When the watchdog file is opened
- When the watchdog timer is changed
- When the watchdog is kicked
- When the watchdog is disabled and closed
- Whenever a coin passes through the laser (which logs what coin passes through and states the new total balance)
- Whenever the reset button or the clear button is pressed

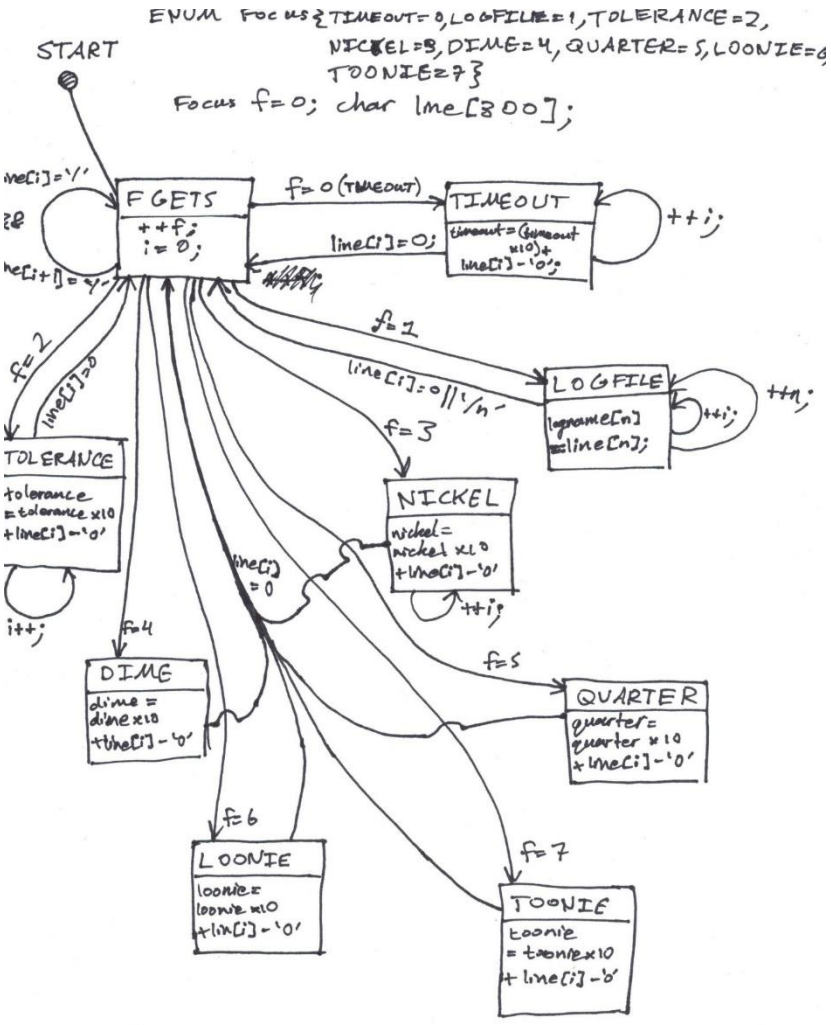
We chose these particular logging points because we believe the points stated above are critical points in our program. Consequently, whenever these critical points occur, we want to log the information into the log file.

Configuration

Our system is configurable. This is because our program reads the expected time that each coin breaks the laser from a config file. Consequently, we can change these values in the config file if we wanted to adjust these values. Additionally, we also read the file location of the log file and the time that the watchdog causes a time out from the config file. The config reading function doesn't pay attention to any alphanumeric characters before the equals signs simply because the config file was extremely simple and a full string-comparing state machine is frankly unnecessary. The config reader simply uses the fgets() function to store a line of the file in a temporary buffer array called line[]. The comment system is double slashes: //. The format and the default values of our config file are shown below:

```
//  This config file defines default calibrations for how long each type of coin will
//  break the laser. These assume a perfectly flat surface; any significantly slanted surfaces
//  will require calibration on startup. NOTE: We no longer use the value tolerance in our program
//  but we still read it from the config file.

WATCHDOG_TIMEOUT = 15

LOG_FILE = /home/pi/coin.log

TOLERANCE = 100

NICKEL = 390

DIME = 330

QUARTER = 420

LOONIE = 470

TOONIE = 600
```

State Machine



Testing

Testing the system was done with all five coins repeatedly and the readings were analyzed. The system was not 100% accurate, and we did not expect it to be. The program had a hard time distinguishing between the Loonie (26.5 mm) and the Toonie (28 mm), and sometimes between the Quarter (23.8 mm) and the Loonie. So, we tested the values and laid out the results on an excel spreadsheet:

Coins:	Nickel	Dime	Quarter	Loonie	Toonie
Number of while loop cycles:	340	308	458	443	453
	367	322	372	429	432
	346	328	375	432	439
	396	324	458	431	465
	366	286	417	406	451
	361	322	383	433	430
	327	300	430	456	440
	338	296	434	437	439
	364	325	389	432	491
Maximum:	396	328	458	456	491
Minimum:	327	286	372	406	430
Tolerance:	69	42	86	50	61
Midpoint:	361.5	307	415	431	460.5

Using the maximum values of each coin, we approximated the upper limit values of every coin and updated the config file with that data. The data was not very accurate but was somewhat relatively distinguishable. This led us to design the cancel button. That is discussed later in this document.

Extras, Limitations & Reflection

Coin Chute

The coin chute was designed to have every coin roll uniformly past the laser. The dimensions of each coin were carefully measured not only so every coin could fit, but so the coins would roll with minimal wobble as well. This introduced a problem: the slide would have to be modeled after the toonie as it is the largest coin with a thickness of 1.8 millimetres. The dime, on the other hand, is only 1.22 millimetres thick. This means that dimes would inevitably wobble going down our chute, skewing measurements. This was alleviated somewhat due to the very small, easily distinguishable diameter of the dime.

The structure at the top of the chute was introduced to normalize entry speed of coins. It works best with the larger coins, but again the smaller coins would experience deviance in motion. The structure also had to be taped down because it is very light, and the weight of some coins would wobble it.

The solution to all of these problems would be a larger, more complicated structure. This would be out of our scope simply because the much higher cost of materials. As this is more of a proof of concept, the structure was satisfactory.

Reset & Cancel Buttons

We built a reset button in the circuit which is self-explanatory. Pressing the reset button clears the total balance and the number of coins of the bank. It is also written in the logfile.

We also built a cancel button when we realised that the system was not as accurate as we needed it to be. The cancel button clears the last coin added to the bank. This is useful if the coin added is detected incorrectly.

Both the buttons were pulled up using a 1000kΩ resistors to prevent noise interruption. Button bouncing was also accounted for using appropriate delays and while loops.

Limitations

There are many limitations to this project. The system was very inaccurate as there's a lot of random error. The software timing method is also very inaccurate as the usleep() function is

not very precise. The diameters of the coins being so similar made it even more difficult for the system to detect coins correctly.

Reflection

If we could start over, we would probably take better measures to make sure the velocities of all the coins are always the same. Reducing friction would be helpful. A better approach would be to actually have a “hand” drag every coin down the slide at a certain velocity instead of allowing the coins to roll freely.

It would also be helpful to use a better timing system. Usleep() is very inaccurate and doesn’t keep time very well. If we could start over, we think it would be a good idea to implement the microcontrollers internal timer based on the clock cycle. This would introduce a huge increase in accuracy as the timer runs independently from the program.

Appendix

Source Code

All codes can be found in the link below:
https://drive.google.com/open?id=1TYQcBqAmZmssYcUj1N2UtJ7EyiQvwxqB

Peer Contribution

Everyone in the group contributed to everything in the project. In particular, Raphael was more focused on the programming and software details, Vojdan was more focused on the config file, state machine and the 3D modelling of the slide, and Zahin was more focused on the setup of the hardware.