EEE174 –CPE185 INTRODUCTION TO MICROPROCESSORS

LAB 3 – RASPBERRY PI

Lab Session: Wednesday 6:30PM - 9:10PM

Section 32385

Lab Instructor: Sean Kennedy

Student Name: Andrew Robertson

TABLE OF CONTENTS

PART 1

OVERVIEW

Part 1 of this lab is simply a setup process. The goal is to have an operating system installed, be able to log in and navigate, and connect to the internet. The only demo portion for this section is to show the instructor we connected to the internet.

Installation resource: https://www.raspberrypi.org/learning/hardware-guide/

Lab Discussion

Work Performed / Solution:

Since I bought an SD card from Amazon without an OS preinstalled I needed to download the NOOBS software and install it myself. This process was easy and straight forward. After installing the SD formatter and formatting my SD card from https://www.sdcard.org/downloads/formatter_4/index.html I then downloaded NOOBS from https://www.raspberrypi.org/downloads/ and followed the directions to install it.

Once that was done I plugged up my monitor (via HDMI), a wireless mouse and keyboard, and a 5.0v 2.0A micro USB power supply. The power supply isn't quite the 2.5A recommended so it was a bit of an experiment. Side note, I have now heard from a few sources that even the 2.5A power supplies can underdeliver at times. With the described setup the Raspberry Pi came to life, however I do need to note that the low power lightning bolt was continuously visible while operating.

Since I was at home when performing the step to connect the Raspberry Pi to the internet, this final step posed no issue at all and was just as easy as my windows desktop.

Listing File(s):

https://www.raspberrypi.org/learning/hardware-guide/equipment/	check equipment
https://www.raspberrypi.org/downloads/	NOOBS
https://www.raspberrypi.org/learning/hardware-guide/quickstart/	hardware setup
https://www.raspberrypi.org/learning/hardware-guide/networking/	internet connectivity
https://www.raspberrypi.org/learning/hardware-guide/audio/	audio

PART 2

OVERVIEW

We now verify Python is installed and apply an existing piece of Python code to confirm all is working well. This is my first time ever seeing Python so also sparked a bit of interest in trying to learn more about its rules and syntax.

Lab Discussion

Work Performed / Solution:

First thing to do is to make sure Python is installed, as can be seen in the first two lines of the image below. The version preinstalled for me is newer than that in the documentation, so it seemed I was ready to go.

The next step is to open a text editor to write the provided python code to a local file. As can be seen in lines 3 and 4 of the image below I first opened vi, then Nano. Nano is what I chose to use for the remainder of this lab since I have used it before. After writing the file I then changed it's permissions to be executable via "chmod".

I then goofed on the line afterward by mistaking the text in the documentation as an I instead of an L for the list command to make sure the file was saved where expected. Once we confirm it's there we can run it by using the python command and providing the name of the file since we're in the same directory. Running this file created a text file which we can also confirm exists with "LS". Dumping the contents of this file to the display is done with "cat" and should display the text Hello World as mine does below.

```
pi@raspberrypi:~ $ python --version
Python 2.7.13
pi@raspberrypi:~ $ vi
pi@raspberrypi:~ $ nano
pi@raspberrypi:~ $ chmod +x Write_to_file.py
pi@raspberrypi:~ $ Is
bash: Is: command not found
pi@raspberrypi:~ $ ls
Desktop    Downloads  Pictures  Templates  Write_to_file.py
Documents  Music      Public    Videos     python_games
pi@raspberrypi:~ $ python Write_to_file.py
pi@raspberrypi:~ $ ls
Desktop    Downloads  Pictures  Templates  Write_to_file.py  testfile.txt
Documents  Music      Public    Videos     python_games
pi@raspberrypi:~ $ cat testfile.txt
Hello Worldpi@raspberrypi:~ $ scrot
```

Listing File(s):

```
#!/usr/bin/python

file = open("testfile.txt","w")

file.write("Hello World")

file.close()
```

PART 3

OVERVIEW

This is the same idea as part 2, however now we are going to execute a Hello World program written in C instead of Python.

Lab Discussion

Work Performed / Solution:

Despite the nature of this step being a rinse a repeat sort of step I did learn something unexpected from it. This time, instead of writing the code I attempted to copy and paste it from the instructions. At a glance it looked just fine but when I tried to compile it I run into some issues. Some hidden formatting symbols from the instructions were interpreted as other unknown symbols. Once I opened the file with Nano and cleaned it up, I was able to compile it and run the object file without issue.

```
pi@raspberrypi:~ $ nano hello.c
pi@raspberrypi:~ $ gcc -o hello hello.c
hello.c: In function 'main':
hello.c:3:8: error: stray '\342' in program
 printf(ÔÔHello World\n");
        ^
hello.c:3:9: error: stray '\200' in program
 printf(ÔÔHello World\n");
         ^
hello.c:3:10: error: stray '\234' in program
 printf(ÔHello World\n");
          ^
hello.c:3:11: error: 'Hello' undeclared (first use in this function)
 printf("Hello World\n");
           ^~~~~
hello.c:3:11: note: each undeclared identifier is reported only once for each function it appears in
hello.c:3:17: error: expected ')' before 'World'
 printf("Hello World\n");
                 ^~~~~
hello.c:3:22: error: stray '\' in program
 printf("Hello World\n");
                      ^
hello.c:3:24: error: stray '\342' in program
 printf("Hello World\nÔÔ");
                        ^
hello.c:3:25: error: stray '\200' in program
 printf("Hello World\nÔÔ");
                         ^
hello.c:3:26: error: stray '\235' in program
 printf("Hello World\nÔ");
                          ^
pi@raspberrypi:~ $ nano hello.c
pi@raspberrypi:~ $ gcc -o hello hello.c
pi@raspberrypi:~ $ ./hello
Hello World
pi@raspberrypi:~ $ scrot
```

Listing File(s):

```
#include <stdio.h>

int main(){
    printf("Hello World\n");
    return 0;
}
```

PART 4

OVERVIEW

We now explore one way of transferring signals in and out of the Raspberry Pi using both C and Python. This will be the physical input and output GPIO pins built into the board.

Lab Discussion

Work Performed / Solution:

First we break out a breadboard and wire both a push button and a LED. The LED should be appropriately current limited and connected to GPIO 17. The pushbutton should be wired for normally low, active high and connected to GPIO 27.

After this is completed we then open a text editor and write the Python file provided for this part of the lab first. Attempting to run this file is where I ran into a very new issue to me. Being new to Python, I did not know white space was so important and received and error for it. Fixing the indentation cleared this up and allowed the program to work as intended. Pressing the pushbutton makes the LED light.

We then follow this same path again for the C program. It is again provided, uses the same pins, and should result in the same functionality. After fixing one small typo in the provided code, it did just this. The typo in the provided code is a misspelling of the word “button”.

```
pi@raspberrypi:~ $ nano gpiotest.py
pi@raspberrypi:~ $ nano gpiotest.py
pi@raspberrypi:~ $ python gpiotest.py
  File "gpiotest.py", line 15
    while 1:
        ^
IndentationError: expected an indented block
pi@raspberrypi:~ $ nano gpiotest.py
pi@raspberrypi:~ $ python gpiotest.py
Simple Python GPIO Start! Press CTRL+C to exit
^Cpi@raspberrypi:~ $ nano gpiotest.c
pi@raspberrypi:~ $ gcc -o gpiotest gpiotest.c -l wiringPi
gpiotest.c: In function 'main':
gpiotest.c:11:10: error: 'butontPin' undeclared (first use in this function)
  pinMode(butontPin, INPUT); // Set button as INPUT
          ^~~~~~~~~
gpiotest.c:11:10: note: each undeclared identifier is reported only once for each function it appears in
pi@raspberrypi:~ $ nano gpiotest.c
pi@raspberrypi:~ $ gcc -o gpiotest gpiotest.c -l wiringPi
pi@raspberrypi:~ $ ./gpiotest
C GPIO program running! Press CTRL+C to quit.
^C
pi@raspberrypi:~ $
```

LISTING FILES(S):

(Provided Python code)

```
# External module imports
import RPi.GPIO as GPIO
import time

# Pin Definitons:
ledPin = 17 # Broadcom pin 17 (PI pin 11)
buttonPin = 27 # Broadcom pin 17 (PI pin 13)

# Pin Setup:
GPIO.setmode(GPIO.BCM)
GPIO.setup(ledPin, GPIO.OUT)
GPIO.setup(buttonPin, GPIO.IN)

# Initial state for LED:
GPIO.output(ledPin, GPIO.LOW)

print("Simple Python GPIO Start! Press CTRL+C to exit")
try:
    while 1:
        if GPIO.input(buttonPin):
            GPIO.output(ledPin, GPIO.HIGH)
        else:
            GPIO.output(ledPin, GPIO.LOW)

except KeyboardInterrupt: # If CTRL+C is pressed, exit and clean
    GPIO.cleanup() # cleanup all GPIO
```

(Provided C code)

```
#include <stdio.h>    // Used for printf() statements
#include <wiringPi.h> // Include WiringPi library!

// Pin number declarations. We're using the Broadcom chip pin numbers.
const int ledPin = 17; // Regular LED - Broadcom pin 23, P1 pin 16
const int buttonPin = 27; // Active-low button - Broadcom pin 17, P1 pin 11

int main(void)
{
    // Setup stuff:
    wiringPiSetupGpio(); // Initialize wiringPi -- using Broadcom pin numbers

    pinMode(ledPin, OUTPUT);     // Set regular LED as output
    pinMode(butontPin, INPUT);      // Set button as INPUT

    printf("C GPIO program running! Press CTRL+C to quit.\n");

    while(1)
    {
        if (digitalRead(buttonPin))
            digitalWrite(ledPin, HIGH);
        else
            digitalWrite(ledPin, LOW);
    }

    return 0;
}
```

PART 5

OVERVIEW

We now explore another way of transferring signals in and out of the Raspberry Pi using Python. This will be the internet capabilities of the device, in my case wireless but can be done with the ethernet port as well.

Lab Discussion

Work Performed / Solution:

The lab description said a lot of Pythons greatest uses have to do with its internet abilities. We now get to try it out using some provided, known – good code. This part requires two computers capable of running python programs, both operating on the same internet connection. For me this was my Raspberry Pi and my windows computer initially. Using my favorite text editor on the Raspberry Pi, I wrote the provided Server (transmitter) program and saved it locally. Next, I wrote the client (receiver) program on my windows computer and saved it locally. Ultimately, both programs need to be running at the same time, but the order does matter. I launched the server program first, followed by the client program and was immediately met with an issue.

The documentation said we are to expect Hello World on both displays but all I saw was an IP address. After looking at the client computer I saw it said there was an issue retrieving it's own network information despite the network address coming through. I thought this must be an antivirus or firewall issue so I then I made a few more attempts seen below while fiddling with different settings on my windows machine. Unfortunately all my attempts were unsuccessful so I decided to launch a live disc of Ubuntu on my windows computer and try it that way. On the first try, it worked.

LISTING FILES(S):

(Server program)

```
#!/usr/bin/env python

import socket

TCP_IP = '0.0.0.0'
TCP_PORT = 8080
BUFFER_SIZE = 20  # Normally 1024, but we want fast response

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((TCP_IP, TCP_PORT))
s.listen(1)

conn, addr = s.accept()
print 'Connection address:', addr
while 1:
    data = conn.recv(BUFFER_SIZE)
    if not data: break
    print "received data:", data
    conn.send(data)  # echo
conn.close()
```

(Client program)

```
#!/usr/bin/env python

import socket

TCP_IP = '0.0.0.0" #Put your RPI Ip Address here
TCP_PORT = 8080
BUFFER_SIZE = 1024
MESSAGE = "Hello, World!"

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((TCP_IP, TCP_PORT))
s.send(MESSAGE)
data = s.recv(BUFFER_SIZE)
s.close()

print "received data:", data
```

PART 6

OVERVIEW

We are now let loose with newfound knowledge and a creative mind. The task here is to create some sort of project on our own that combines code with some form of board interaction.

Lab Discussion

Work Performed / Solution:

The first idea that came to mind for me was to make use of the joystick yet again, but I wanted something different. I decided I would find out if hardware triggered interrupt services are able to be implemented for this platform and they are. A counting pattern is very easy to recognize, and as such easy to tell when something changes unexpectedly so I decided to base my project around this. The idea became a seven-segment repeating 0-9 counter (for visibility) that would have a pushbutton interrupt setting the counter to 0.

I placed my seven-segment display on my breadboard and wired a button for active high. Next, I Looked up the seven-segment wiring diagram and assigned pins with names to make high and low assignment more intuitive. After this I realized the counter variable would need to be global because the interrupt function needs to be a void function with no parameter list. I then began with the provided gpiotest.c code as a framework.

With the variable framework in place I then wrote the quicker of the two functions first, the interrupt function. All it does it sets the counter to 0. Next was the display function, this simply interprets a value from 0-9 as a series of high and low signals to send the seven-segment display to make the number visible. Last, was the main function to make it all work. The main function displays the current value of the counter to the seven-segment using the display function, increments the counter, sets the counter to 0 if it has been incremented to 10, waits half a second, and then repeats this process. At the same time the program keeps an eye on the pushbutton always (not actually but polls very quickly) and if it is pressed, pauses the execution of main, sets the counter to 0, and resumes the execution of main.

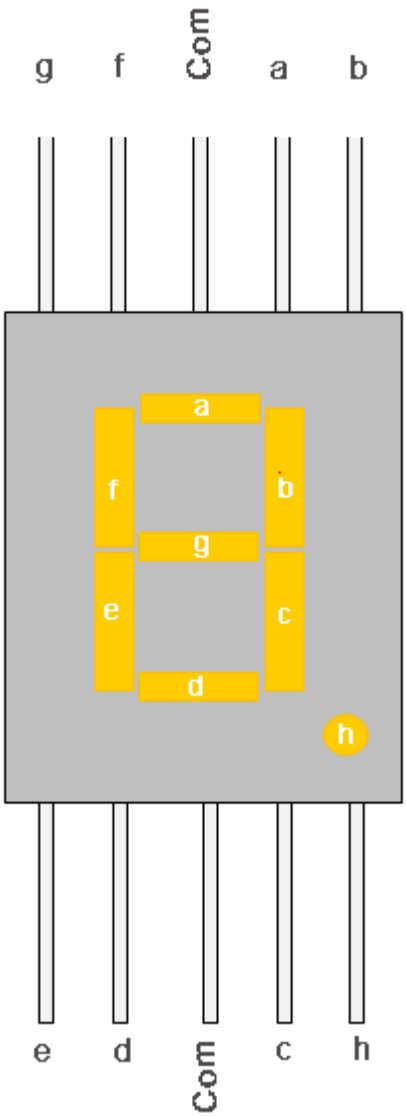
Below is the output of the program, there isn't much to see so I've included a photo of the circuit too.

```
pi@raspberrypi:~ $ nano part6.c
pi@raspberrypi:~ $ gcc -o part6 part6.c -l wiringPi
pi@raspberrypi:~ $ ./part6
C GPIO program running! Press CTRL+C to quit.
^C
```

LISTING FILES(S):

WiringPi functions, including interrupts: https://projects.drogon.net/raspberry-pi/wiringpi/functions/

Seven segment pin association courtesy of: https://circuitdigest.com/article/7-segment-display



My C code, using the provided gpiotest.c code as a base:
(Note, I removed cases 3- 9 to avoid redundancy)

```
#include <stdio.h> // Used for printf() statements
#include <wiringPi.h> // Include WiringPi library!
// Pin number declarations. We're using the Broadcom chip pin numbers.
const int segA = 13, segB = 6, segC = 16, segD = 20, segE = 21,
          segF = 19, segG = 26, segDP = 12;
int counter = 0;

void interrupt()
{
counter = 0;
}

void display(int number)
{
        switch(number)
        {
        case 0:
                digitalWrite(segA, 1);
                digitalWrite(segB, 1);
                digitalWrite(segC, 1);
                digitalWrite(segD, 1);
                digitalWrite(segE, 1);
                digitalWrite(segF, 1);
                digitalWrite(segG, 0);
                digitalWrite(segDP, 0);
                break;
        case 1:
                digitalWrite(segA, 0);
                digitalWrite(segB, 1);
                digitalWrite(segC, 1);
                digitalWrite(segD, 0);
                digitalWrite(segE, 0);
                digitalWrite(segF, 0);
                digitalWrite(segG, 0);
                digitalWrite(segDP, 0);
                break;
        case 2:
                digitalWrite(segA, 1);
                digitalWrite(segB, 1);
                digitalWrite(segC, 0);
                digitalWrite(segD, 1);
                digitalWrite(segE, 1);
                digitalWrite(segF, 0);
                digitalWrite(segG, 1);
                digitalWrite(segDP, 0);
                break;
```

```
int main(void)
{

        // Setup stuff:
        wiringPiSetupGpio(); // Initialize wiringPi -- using Broadcom pin numbers
        pinMode(segA, OUTPUT); // Set regular LED as output
        pinMode(segB, OUTPUT); // Set regular LED as output
        pinMode(segC, OUTPUT); // Set regular LED as output
        pinMode(segD, OUTPUT); // Set regular LED as output
        pinMode(segE, OUTPUT); // Set regular LED as output
        pinMode(segF, OUTPUT); // Set regular LED as output
        pinMode(segG, OUTPUT); // Set regular LED as output
        pinMode(segDP, OUTPUT); // Set regular LED as output

        wiringPiISR(18, INT_EDGE_RISING, &interrupt);

        printf("C GPIO program running! Press CTRL+C to quit.\n");
        while(1)
        {
                display(counter);
                counter = counter + 1;
                if (counter == 10)
                {
                        counter = 0;
                }
                delay(500);
        }
        return 0;
}
```

CONCLUSION

I learned that a very basic text editor with a limited character set is not forgiving in any way. If there is an odd interpretation it is not thrown away or carried through some exception but kept as potential junk that can cause some issues, luckily easy to identify in my case earlier. I also learned that Python cares about white space and thus white space is not cosmetic or even just for legibility but as part of the syntax. This lab has sparked me to save a new bookmark for Python SoloLearn modules so I can get somewhat up to speed with the language.