

Lab 1
Lab Section: Monday

September 27, 2020

Jonathan Nusantara, Eric Kahn,
Eric Hall

jan265, edk52, ewh73

Introduction

The main objective of this lab is to familiarize ourselves with the Raspberry Pi, the PiTFT that is connected to it, and some input and output capabilities utilizing the GPIO. This involves configuring the Raspbian Linux Kernel for the Raspberry Pi and figuring out different ways to run Raspberry Pi (via monitor and keyboard, via PiTFT, or via ssh and VNC viewer). This also involves installing drivers for the PiTFT, video, and audio playbacks and testing what they are capable of. Finally, all of these components are set up to be integrated with each other and tested out to learn how everything is able to work together.

We also familiarize ourselves with python and bash scripting, which can be used to automate interactions with the Pi, including from physical button presses (i.e. ways that aren't over SSH or via a keyboard).

Design and Testing

Week 1:

Setting up Raspberry Pi:

The raspberry pi setup started by placing the Pi in its case to protect it from static discharge, and then placing the 40 pin connector/cables onto the PiTFT. After making sure it was oriented correctly (white stripe near the TFT's buttons in picture) the PiTFT was connected to the raspberry pi. Once this was completed, we could connect to the pi. I initially did this using ssh over my apartment wifi. There was some difficulty getting this to work as I don't have access to the router in my apartment but eventually we were able to connect. Later after getting the necessary cables I was able to connect my pi more easily by using a monitor and keyboard.



Figure 1 - Raspberry Pi setup with PiTFT attached.

Installing Raspbian Kernel:

We had previously set up our SD card with the raspbian kernel by installing latest Buster release, 4.19.97. Now we used raspi-config to setup most other aspects. Using raspi-config we set the pi to boot directly to the console window, set the default locale to english-US-UTF8, the time zone, the hostname and password, and the default keyboard layout. Finally, we performed the following to check for the latest update for the Raspbian Kernel:

```
Sudo apt-get update  
Sudo apt-get full-upgrade
```

After this we re-booted so the changes could take effect. We then installed vim and used startx (if we had a monitor) to ensure it worked. Once this was all finished, we backed up our SD cards.

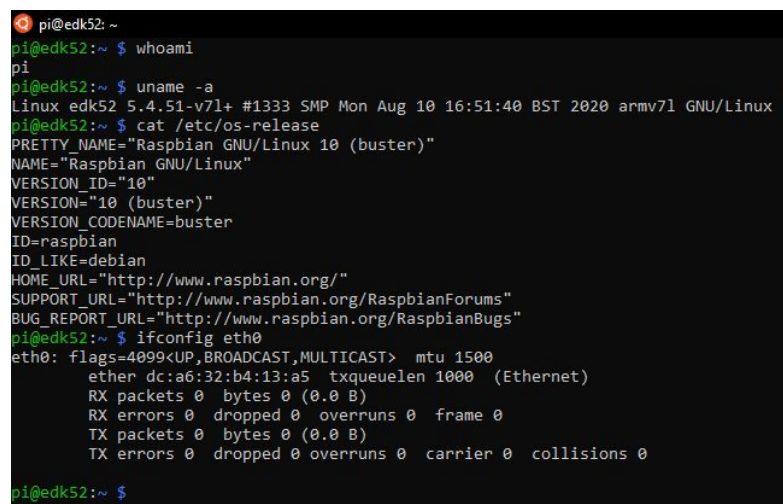
A terminal window on a Raspberry Pi showing the output of several system information commands. The prompt is 'pi@edk52: ~'. The commands and their outputs are: 'whoami' returns 'pi'; 'uname -a' returns 'Linux edk52 5.4.51-v7l+ #1333 SMP Mon Aug 10 16:51:40 BST 2020 armv7l GNU/Linux'; 'cat /etc/os-release' returns 'PRETTY_NAME="Raspbian GNU/Linux 10 (buster)"', 'NAME="Raspbian GNU/Linux"', 'VERSION_ID="10"', 'VERSION="10 (buster)"', 'VERSION_CODENAME=buster', 'ID=raspbian', 'ID_LIKE=debian', 'HOME_URL="http://www.raspbian.org/"', 'SUPPORT_URL="http://www.raspbian.org/RaspbianForums"', 'BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"'. Then 'ifconfig eth0' returns 'eth0: flags=4099<UP,BROADCAST,MULTICAST> mtu 1500', 'ether dc:a6:32:b4:13:a5 txqueuelen 1000 (Ethernet)', 'RX packets 0 bytes 0 (0.0 B)', 'RX errors 0 dropped 0 overruns 0 frame 0', 'TX packets 0 bytes 0 (0.0 B)', 'TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0'. The prompt returns to 'pi@edk52: ~ \$'.

Figure 2 - Display information about the OS.

TFT screen install and setup:

For the TFT, we had to install many packages, including git, python-pip, evtest, libts-bin and more. We then edited our config.txt file to be compatible with the PiTFT, re-booted and ran dmesg to check that the modules needed for the PiTFT were installed properly.

We specifically looked for the “stmpe-spi” and “graphics fb1” (or “graphics fb0” if we were running headless like Eric Hall) modules. We next added a udev rule which allows us to use the TFT as a touch screen and verified the touch screen was working by running evtest and checking for output while touching the screen. After calibrating the screen and starting the console window, we downloaded a test video to try running on the PiTFT.

```
#Added for piTFT
[pi0]
device_tree=bcm2708-rpi-0-w.dtb
[pi1]
device_tree=bcm2708-rpi-b-plus.dtb
[pi2]
device_tree=bcm2709-rpi-2-b.dtb
[pi3]
device_tree=bcm2710-rpi-3-b.dtb
[all]
dtparam=spi=on
dtparam=i2c1=on
dtparam=i2c_arm=on
dtoverlay=piTFT28-resistive,rotate=90,speed=6400000,fps=30
pi@edk52:~$ time date
Sun Sep 27 18:02:23 EDT 2020

real    0m0.009s
user    0m0.008s
sys      0m0.002s
pi@edk52:~$
```

Figure 3 - PiTFT setup.

```
pi@edk52:~$ cat /etc/udev/rules.d/95-stmpe.rules
SUBSYSTEM=="input", ATTRS(name)=="*stmpe*", ENV(DEVNAME)=="*event*", SYMLINK+="input/touchscreen"
pi@edk52:~$ cat /etc/udev/rules.d/95-touchmouse.rules
SUBSYSTEM=="input", ATTRS(name)=="*touchmouse*", ENV(DEVNAME)=="*event*", SYMLINK+="input/touchscreen"
pi@edk52:~$ cat /etc/udev/rules.d/95-ftcaptouch.rules
SUBSYSTEM=="input", ATTRS(name)=="*EP0110H09*", ENV(DEVNAME)=="*event*", SYMLINK+="input/touchscreen"
pi@edk52:~$ cat /etc/pointercal
33 -5782 21364572 4221 35 -1006432 65536
pi@edk52:~$ cat /boot/cmdline.txt
console=serial0,115200 console=tty1 root=PARTUUID=ea7d04d6-02 rootfstype=ext4 elevator=deadline fsck.repair=yes rootwait quiet splash plymouth.ignore-serial-consoles fbcon=map:10 fbcon=font:VGA8x8
pi@edk52:~$ cat /etc/default/console-setup
# CONFIGURATION FILE FOR SETUPCON

# Consult the console-setup(5) manual page.

ACTIVE_CONSOLES="/dev/tty[1-6]"

CHARMAP="UTF-8"

CODESET="guess"
FONTFACE="Terminus"
FONTSIZE="6x12"

VIDEOMODE=

# The following is an example how to use a braille font
# FONT="lat9w-00.psf.gz brl-8x8.psf"
pi@edk52:~$
```

Figure 4 - Content of edited file for PiTFT implementation.

```
pi@edk52:~$ dmesg | grep stmpe-spi
[ 4.459850] stmpe-spi spi0.1: stmpe610 detected, chip id: 0x811
pi@edk52:~$ dmesg | grep graphics
[ 5.731686] graphics fb1: fb_ili9340 frame buffer, 320x240, 150 KiB video memory, 4 KiB buffer memory, fps=33, spi0.0 at 64 MHz
pi@edk52:~$
```

Figure 5 - Content of dmesg that we searched to confirm modules for PiTFT are installed.

We verified that PiTFT is able to run the video by running this in terminal:

```
sudo SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -vo sdl  
-framedrop
```

The audio is also tested via the headphone jack. The mplayer is also tested by calling by running the command when in startx, and verified that the video is able to play via mPlayer. Finally, we verified that our ssh is working properly and we were able to run the command to play the video on RPi via ssh.



Figure 6 - Video playing via mPlayer in PiTFT.

We found a few challenges using mplayer to start the video. During Eric's first time doing this, he was not using a monitor and the piTFT was set to fb0. This created confusion because the directions assumed he had a monitor set to fb0, and suggested running mplayer with fb1. This, of course, did not work, but we resolved the error by running `dmesg` again and noticing that the TFT was using fb0 and specifying that instead of fb1.

Week 2:

After week 1, we have our Raspberry Pis set up, and the video and audio playback work for all of us on the PiTFT and monitor. In week 2, we explored the different ways to control the playback using external methods. The first step in this process is to redirect commands into MPlayer via a FIFO special file object, then we automate the sending with Python, and finally use button presses as the input to send commands.

Control MPlayer with FIFO:

A FIFO is a special file object that works like a pipe. It has the distinct advantage over a pipe of allowing us to feed the output of one command into the input of another command between processes, because it is represented as a file in the file system rather than an output redirection.

The following procedures are run after calling `startx`, so the desktop will be displayed on the monitor. The first step is to create a fifo named `video_fifo`. This is done by first creating the fifo by running `mkfifo video_fifo` in the terminal in the directory of lab 1. Then, we will open another terminal console, so a total of two terminal console windows are open. One will be used to run `mplayer`, and the other will be used to pass commands to `mplayer` via the fifo.

In one terminal,

```
$ mplayer -input file=video_fifo bigbuckbunny320p.mp4
```

was run. The “\$” indicates the command was run in a terminal; we will use “%” to indicate a command run as part of a bash script and “>>” to indicate a Python command. This tells `mplayer` to run the `.mp4` while listening to `video_fifo` for input. On the second window, we would pass the `mplayer` commands via `video_fifo`. The complete list of commands may be found [here](http://www.mplayerhq.hu/DOCS/tech/slave.txt): <http://www.mplayerhq.hu/DOCS/tech/slave.txt>. The pause command was tested by running:

```
$ echo "pause" > video_fifo
```

where `echo` prints the command “`pause`” to `video_fifo`, which would then be read by `mplayer` resulting in the video being paused. When that same command is run again, it will unpauses the video. The quit command, which stops video playback, was also tested by running:

```
$ echo "quit" > video_fifo
```

By running the commands above, we confirmed that the `mplayer` is reading from the fifo, and that we are able to pass commands to the fifo by `echo`-ing to it. From this section, we learned how to set up `mplayer` to listen to fifo by adding the flag `-input file=<fifo name>` when calling `mplayer`, and how to write to the fifo with `echo` to pass commands to `mplayer`. This section of the lab went pretty well, without any significant challenges.

```

File Edit Tabs Help
subprocess.Popen(send_command)
AttributeError: 'module' object has no attribute 'popen'
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python fifo_test.py
pause
pause
exit
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python fifo_test.py
pause
pause
exit
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python
Python 2.7.16 (default, Oct 10 2019, 22:02:15)
Type "help", "copyright", "credits" or "license()" for more information.
>>> quit()
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ scrot
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "pause" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "pause" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "speed_mult 2" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "speed_incr 3" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "stop" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ scrot

File Edit Tabs Help
Clip info:
major_brand: mp42
minor_version: 0
compatible_brands: mp42isomavc1
creation_time: 2013-11-26T16:05:17.000000Z
encoder: HandBrake 0.9.9 2013052900
Load subtitles in ./
=====
Opening audio decoder: [ffmpeg] FFmpeg/libavcodec audio decoders
AUDIO: 48000 Hz, 2 ch, floatle, 160.0 kbit/5.21% (ratio: 19998->384000)
Selected audio codec: [ffaac] afm: ffmpeg (FFmpeg AAC (MPEG-2/MPEG-4 Audio))
=====
AO: [pulse] Init failed: Connection refused
Failed to initialize audio driver 'pulse'
AO: [alsa] 48000Hz 2ch floatle (4 bytes per sample)
Starting playback...
Movie-Aspect is 1.78:1 - prescaling to correct movie aspect.
VO: [xv] 320x180 => 320x180 Planar YV12
A: 34.2 V: 34.2 A-V: 0.000 ct: 0.063 0/ 0 6% 0% 1.6% 0 0
A: 135.4 V: 135.4 A-V: 0.001 ct: 0.031 0/ 0 21% 1% 8.4% 0 0 4.00x
Exiting... (End of file)
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $

```

Figure 7 - One terminal running mPlayer, and the other used to pass commands to FIFO.

Using Python to control mPlayer with FIFO:

Next, we created a python program called `fifo_test.py`. This python program runs in the foreground, listening for the input from the user. The user will be able to input mplayer commands, such as “pause” or “quit” to the program, and the program will echo it to `video_fifo`, which mplayer will in turn read from and execute the given command. This idea of the program is built upon the previous task, where the user can pass commands to mplayer via another terminal window. But instead of running the full `echo` command in the terminal, the user can just type out the specific mplayer command to the python program.

In the python program, we first imported `sys` and `subprocess`. Then, we created an infinite while loop to listen for text passed by the user. We capture the user’s input as a string with `raw_input()`. Then, we check if the input is equivalent to “end”, which quits the python script. Otherwise, we treat the input as a command to pass to the fifo. We then create a string variable called `sendcommand` to create the command to pass to terminal:

```
>>> sendcommand = "echo " + command + " > video_fifo"
```

where variable “command” is the variable containing the user input. Finally, we use the `subprocess.check_output()` function to pass the command to Bash:

```
>>> subprocess.check_output(sendcommand, shell = True)
```

The command

```
$ mplayer -input file=video_fifo bigbuckbunny320p.mp4
```

was run to test mplayer listening to fifo. In the second terminal window, we used

```
$ python fifo_test.py
```


to run the python program, then passed input commands like “pause”. The program will pass this command to the fifo, and the mplayer will pause. Other commands such as “seek 10 0” to skip forwards and “quit” to quit mplayer were also tested, and we are able to observe the expected behavior from mplayer. This verifies that our python program is running successfully.

```

File Edit Tabs Help
exit
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python fifo_test.py
pause
pause
exit
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python
Python 2.7.16 (default, Oct 10 2019, 22:02:15)
[GCC 8.3.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> quit()
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ scrot
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "pause" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "pause" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "speed_mult 2" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "speed_incr 3" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ echo "stop" > video_fifo
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ scrot
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python fifo_test.py
pause
pause
speed_mult 2
stop
exit
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ scrot

File Edit Tabs Help
major_brand: mp42
minor_version: 0
compatible_brands: mp42isomavc1
creation_time: 2013-11-26T16:05:17.000000Z
encoder: HandBrake 0.9.9 2013052900
Load subtitles in ./
=====
Opening audio decoder: [ffmpeg] FFmpeg/libavcodec audio decoders
AUDIO: 48000 Hz, 2 ch, floatle, 160.0 kbit/5.21% (ratio: 19998->384000)
Selected audio codec: [ffaac] afm: ffmpeg (FFmpeg AAC (MPEG-2/MPEG-4 Audio))
=====
AO: [pulse] Init failed: Connection refused
Failed to initialize audio driver 'pulse'
AO: [alsa] 48000Hz 2ch floatle (4 bytes per sample)
Starting playback...
Movie-Aspect is 1.78:1 - prescaling to correct movie aspect.
VO: [xv] 320x180 => 320x180 Planar YV12
A: 13.5 V: 13.5 A-V: 0.000 ct: 0.063 0/ 0 6% 0% 1.6% 0 0
A: 17.7 V: 17.7 A-V: 0.000 ct: 0.063 0/ 0 6% 0% 1.6% 0 0
A: 38.0 V: 38.0 A-V: 0.000 ct: 0.063 0/ 0 6% 0% 1.6% 0 0
Exiting... (End of file)
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $

```

Figure 8 - One terminal is running mPlayer and the other running fifo_test.py to pass commands from user input

One challenge that we faced in this program is in getting the input of the user. We initially tried to get the input from the user by running `input()`. However, we are receiving errors because it turns out that in Python 2, `input()` is used to take in input as Python expression, thus causing an error. We tested `raw_input()`, which coerces the user input to a string by default, resolving the errors.

Getting input from button connected to GPIO:

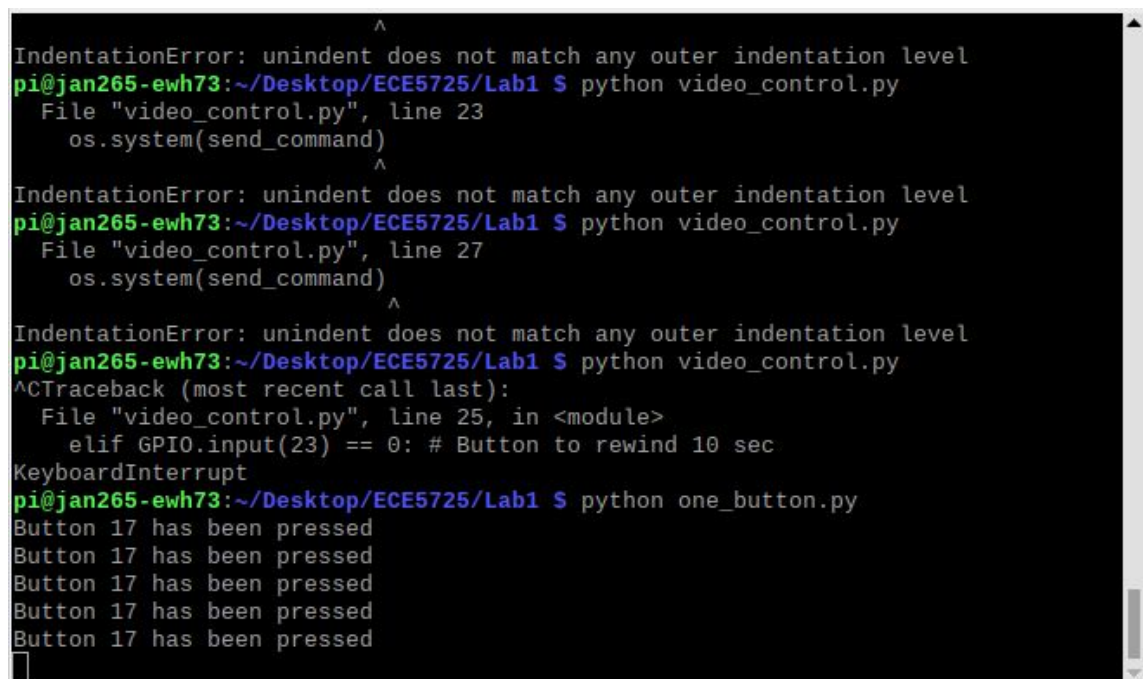
Next, we set up the buttons on the PiTFT as GPIO inputs. We first researched the PiTFT schematic to see which GPIO pins the buttons on the PiTFT are connected to. It turns out that the Broadcom GPIO numbers are labeled under each button. With the buttons' pin IDs in mind, the python program “one_button.py” was created. This program waits for the button of our choosing, and then displays a console message on the terminal when the button is depressed.

First, the Raspberry Pi GPIO module is imported to the program to ease the usage of GPIO pins. The time library is also imported to ham-handedly counter “bouncing” in the button, where the buttons may connect and break connection several times in a single physical press. Next, we told the GPIO module to use the broadcom numbering system, as it translates directly to the pin numbering on our provided hardware. This is done by running `GPIO.setmode(GPIO.BCM)`. We also set up pin 17, the pin connected to our chosen button, to use the internal pull up resistor by running

`GPIO.setup(17,GPIO.IN, pull_up_down=GPIO.PUD_UP)`. Using the internal pull-up resistor forces the GPIO input to be high when the button is not pressed, instead of letting the pin float and possibly read an input when the button was not actually pressed.

Finally, we created an infinite while loop to listen. It has a branch to check if the GPIO 17 value is pulled low, which indicates the button was pressed. When the branch is taken, the message "Button 17 has been pressed" is printed in the terminal window. The `time.sleep(.3)` command is also added after the button is pressed, to have the wait after each button press to debounce the input. The only way to exit the program is to kill the process with `ctrl+c`.

We then ran this python program, as you can see in the screenshot below. When button 17 is pressed, it prints a message to the window. When other buttons are pressed, nothing is printed, which verifies that the program is working as expected. This program introduced how a button can be used via GPIO to control the Raspberry Pi. This section is straightforward and no challenges were encountered.



```
IndentationError: unindent does not match any outer indentation level
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python video_control.py
File "video_control.py", line 23
    os.system(send_command)
    ^
IndentationError: unindent does not match any outer indentation level
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python video_control.py
File "video_control.py", line 27
    os.system(send_command)
    ^
IndentationError: unindent does not match any outer indentation level
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python video_control.py
^CTraceback (most recent call last):
  File "video_control.py", line 25, in <module>
    elif GPIO.input(23) == 0: # Button to rewind 10 sec
KeyboardInterrupt
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python one_button.py
Button 17 has been pressed
Button 17 has been pressed
Button 17 has been pressed
Button 17 has been pressed
Button 17 has been pressed
```

Figure 9 - Python program is run to print button message when button is pressed.

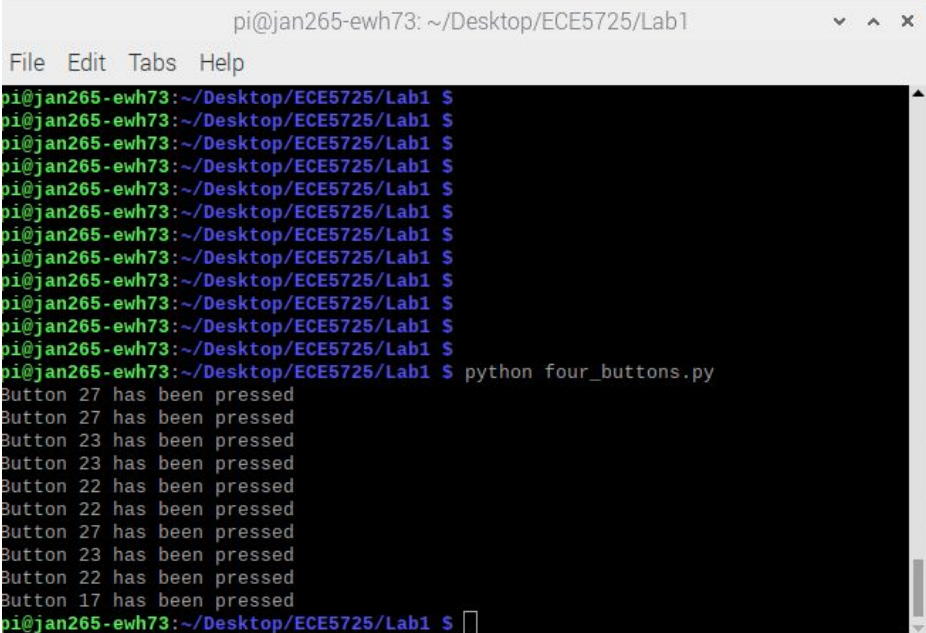
Getting input from four buttons connected to GPIO:

In this section, we will be designing the python program "four_buttons.py". The goal of this program is to build upon the previous python program

“one_button.py” by reading from all four buttons on the PiTFT. Furthermore, one of the buttons should be set to quit the program when pressed.

This program is built based on “one_button.py”, so the first step is to copy the contents from that file. Next, the GPIO for pins 22, 23, and 27 are set up in a similar way to pin 17, by calling the `GPIO.setup()` function and specifying the pin number, setting it as input, and requiring the pull-up resistor. Then, in the infinite while loop, we implement additional branches to check if pins 22, 23, and 27 are low. Each is implemented in a similar way to how pin 17 was polled in the previous section. The `time.sleep(0.3)` command is still used to debounce each button. Each button is given its own, unique print statement. One important part of the program specification is to have either button 17 or 27 (the outermost buttons) quit the program. We chose to have button 17 quit the program by `break`-ing out of the infinite loop if its button press is detected.

This program was then tested in the terminal. When each button is pressed, it prints a short statement including its pin number. As you can see in the screenshot below, the buttons perform consistently. In the special case when button 17 is pressed, prints the same statement but then quits the program as expected. This section is straightforward and we encountered no challenges.



```
pi@jan265-ewh73: ~/Desktop/ECE5725/Lab1
File Edit Tabs Help
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $ python four_buttons.py
Button 27 has been pressed
Button 27 has been pressed
Button 23 has been pressed
Button 23 has been pressed
Button 22 has been pressed
Button 22 has been pressed
Button 27 has been pressed
Button 23 has been pressed
Button 22 has been pressed
Button 17 has been pressed
pi@jan265-ewh73:~/Desktop/ECE5725/Lab1 $
```

Figure 10 - Python program is run to print button message when one of four buttons is pressed.

Controlling mPlayer through FIFO using buttons and python:

The next python program “`video_control.py`” is intended to control mPlayer by using the buttons on the PiTFT. The four buttons are to be set up where the buttons will pause the video, fast forward 10 seconds, rewind 10 seconds, or quit mplayer, respectively. These commands are passed from the python script to mPlayer through a fifo.

This program is mostly the same as `four_buttons.py`. So, the first step is to copy the content from `four_buttons.py` to `video_control.py`. Next, since commands need to be passed to the terminal, the `subprocess.check_output()` function will be used. So, the subprocess module will need to be imported in the beginning of the file. We then remove the print statement of the copied program from “`four_buttons.py`”, since nothing should be printed. Instead, each branch should make the appropriate call to `subprocess.check_output()` to pass mplayer commands to the video_fifo. The commands to be passed are “pause” for pin 17, “seek 10 0” for pin 22 to fast forward 10 secs, “seek -10 0” for pin 23 to rewind 10 secs, and “quit” for pin 27 to quit mplayer. As an example, to pause the video

```
>> subprocess.check_output("echo pause > video_fifo", shell =  
                             True)
```

is called. The check for pin 17 still contains the `break` instruction, as it quits the mPlayer and is required to end the python script.

This program is then tested to verify its implementation. Two terminals are opened, the first running

```
$ mplayer -input file=video_fifo bigbuckbunny320p.mp4
```

to call mplayer to play the video while listening to fifo. Then, on the second terminal, `video_control.py` is called. Then, the buttons on the PiTFT were pressed, and we observed the expected effects on the video playback. When the button connected to pin 27 was pressed, it quit both mplayer and the python program. This test verified that all of the buttons are working correctly to pass commands to fifo and that the mplayer is able to read and execute those commands.

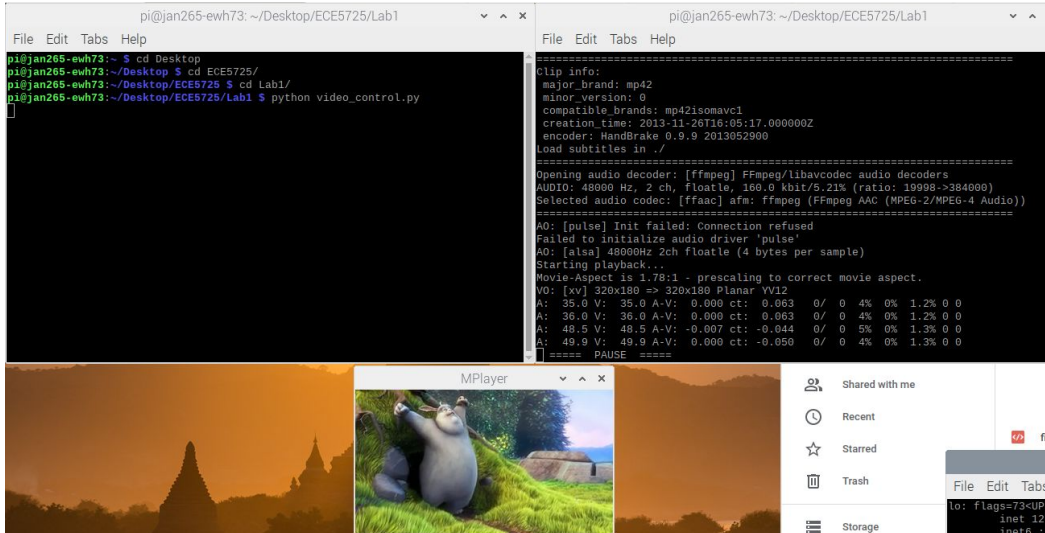


Figure 11 - One terminal is running mPlayer and the other is running python program to allow video playback control using buttons.

This program shows how nicely buttons can be used in a system as an input and to work together with the PiTFT display and video playback. We didn't discover any challenges in this section, as it is just a slight upgrade from the previous programs created.

Creating bash script to play video and control via buttons:

The final section of this lab is to create a program to automate the process of calling mplayer and `video_control.py`. We use a bash script `video_fifo` for this.

At the top of the bash script, there needs to be the "shebang" `#!/bin/bash`. This tells the OS to invoke the specified shell to execute the commands that follow in the script. Next, the command

```
% python video_control.py &
```

is stated in the script. This line runs the python program `video_control.py` (that we created in the last section) in the background. Finally, the command

```
% SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -input
file=video_fifo -vo sdl bigbuckbunny320p.mp4
```

is called to run mplayer on the PiTFT and tell it to listen to `video_fifo` for commands.

The program is then tested on PiTFT. We quit `startx` to move to a terminal on the PiTFT display. Then, the bash script is executed by running:

```
$ ./start_video
```

The video will start automatically, and button inputs from the PiTFT are translated to the expected operations on mplayer video playback. This shows that the entire program and system is running well. Both programs basically behave the same as in previous sections, but because the programs are called by the bash script running them together is easier and faster.

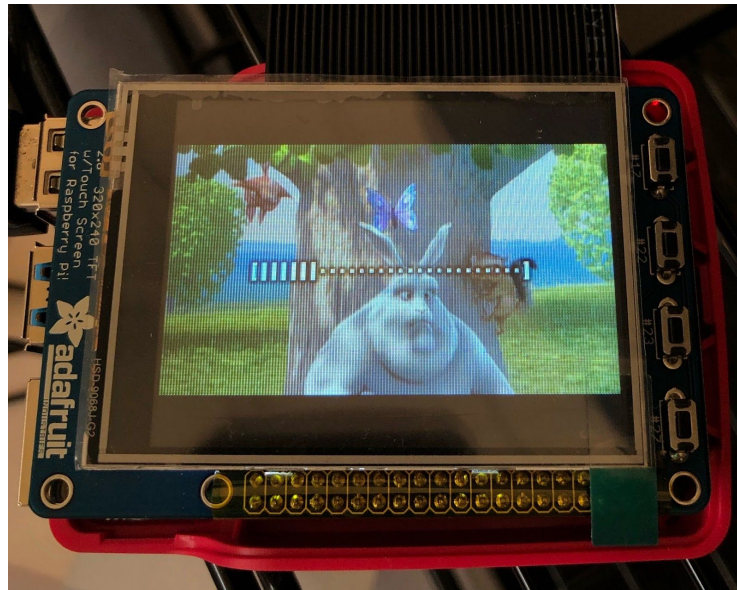


Figure 12 - Bash script would run mPlayer on PiTFT and set buttons to pass commands.

We met no major difficulties implementing this section. We learned how simple bash scripts can help automate the execution of complicated sequences of programs, making things easier. We also learned to execute some programs in the background, such `video_control.py`, so the bash script can trigger several parallel programs. One Jonathan found was on how the video was not able to be played on PiTFT, despite having the buttons working (able to quit mplayer and python program). After some time of debugging, he discovered there was an extra slash "/" in his `SDL_FBDEV=/dev/fb1` command in the script, causing the mplayer setup on PiTFT to fail.

Results:

The majority of the lab everything went smoothly and we were able to meet our goals and complete the checkoff on time. The two main struggles were connecting to the Pi over ssh initially and getting the video to play with mplayer. Eric Kahn could not always ssh over wifi, for an unknown reason. Unfortunately his laptop does not have an ethernet port and he does not have access to the router, but after rebooting a second time the wifi connection worked. Also, Eric Hall lives in campus housing and connecting over eduroam is not easy. He was finally able to create a hotspot from his laptop and

ssh-ed to the Pi using that to discover the MAC address and connect the Pi to eduroam. Jonathan faced a similar issue on how his RPi was not able to connect to his apartment's wifi, and thus not able to ssh to the RPi. Using a monitor is much easier!

As discussed in the sections above, our team was able to meet all of the goals outlined in the lab manual. We demonstrated each task required in the lab, including the various ways of playing videos on the PiTFT or monitor, reading inputs from the buttons on PiTFT, and running bash scripts.

In the first week, our team consisted of only Jonathan Nusantara and Eric Hall. We were able to meet all of the milestones in Week 1 during the lab hours, however we did not have the chance to demo to the TA. Eric Kahn did check off in the first week before he joined our team. In the second week, we were able to roll through the tasks smoothly and complete all checkoffs on time.

Conclusion

Overall, the lab went smoothly after we all gained access to the raspberry pi. Getting ssh to work on a network that isn't yours is much more difficult than if you have access to your router. Connecting to a monitor with a keyboard is by far the easiest way to start using a Pi. There were a few hiccups with getting mplayer to work properly and using a VNC program for those who didn't have a monitor but overall everyone set up their raspberry Pi's successfully and were able to use the PiTFT pins in conjunction with a python script to successfully control a video. All of our code has been uploaded to the ece5725 server, but since there isn't a /home/lab1 directory yet, it is currently located at:

```
/home/Lab1/M_jan265_ewh73_edk52_1
```

Overall, we have learned on how to setup the Raspberry Pi with the Raspbian Kernel, implementing PiTFT with the required setups, and running the buttons as input to Raspberry Pi. We built python programs that enable us in testing all of these capabilities using FIFO, buttons, and PiTFT in controlling video playback. These learnings would prepare us for future labs.

Code Appendix

Fifo_test.py:

```
# Jonathan Nusantara (jan265), Eric Hall (ewh73), Eric Kahn (edk52)
# Lab 1, 09-27-2020
```

```
import subprocess
```

```
while True: # Infinite while loop
    command = raw_input("Enter command: ") # Inout as string

    if command == 'exit': # Command to exit
        break

    # Create command to pass to terminal
    send_command = 'echo ' + command + " > video_fifo"
    subprocess.check_output(send_command, shell = True)
```

One_button.py:

```
# Jonathan Nusantara (jan265), Eric Hall (ewh73), Eric Kahn (edk52)
# Lab 1, 09-27-2020
```

```
import RPi.GPIO as GPIO
import time
```

```
# Set numbering convention
GPIO.setmode(GPIO.BCM)
```

```
# Set GPIO channels for input
# Set pin 17 to be input and set initial value to be pulled up
resistor
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

```
# Monitor the button
while True:
    if GPIO.input(17) == 0: # If low
        print("Button 17 has been pressed")
        time.sleep(.3) # Prevent button bouncing
```

```

Four_buttons.py:
# Jonathan Nusantara (jan265), Eric Hall (ewh73), Eric Kahn (edk52)
# Lab 1, 09-27-2020

import RPi.GPIO as GPIO
import time

# Set numbering convention
GPIO.setmode(GPIO.BCM)

# Set GPIO channels for input
# Set the pins to be input and set initial value to be pull up
resistor
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Monitor the button
while True: # Infinite while loop
    if GPIO.input(17) == 0: # Check if GPIO is low
        print("Button 17 has been pressed")
        time.sleep(.3) # Prevent button bouncing
        break # Quit from the python program
    elif GPIO.input(22) == 0: # Check if GPIO is low
        print("Button 22 has been pressed")
        time.sleep(.3) # Prevent button bouncing
    elif GPIO.input(23) == 0: # Check if GPIO is low
        print("Button 23 has been pressed")
        time.sleep(.3) # Prevent button bouncing
    elif GPIO.input(27) == 0: # Check if GPIO is low
        print("Button 27 has been pressed")
        time.sleep(.3) # Prevent button bouncing

```

```

Video_control.py:
# Jonathan Nusantara (jan265), Eric Hall (ewh73), Eric Kahn (edk52)
# Lab 1, 09-27-2020

import RPi.GPIO as GPIO
import time
import subprocess

# Set numbering convention
GPIO.setmode(GPIO.BCM)

# Set GPIO channels for input
# Set the pins to be input and set initial value to be pull up
resistor
GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(23, GPIO.IN, pull_up_down=GPIO.PUD_UP)
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_UP)

# Monitor the button
while True:
    if GPIO.input(17) == 0: # Button to pause
        send_command = 'echo pause > video_fifo' # Set command
string for mplayer command
        subprocess.check_output(send_command, shell = True) # Send
command to terminal
        time.sleep(.3) # Prevent button bouncing
    elif GPIO.input(22) == 0: # Button to fast-forward 10 sec
        send_command = 'echo seek 10 0 > video_fifo'
        subprocess.check_output(send_command, shell = True)
        time.sleep(.3) # Prevent button bouncing
    elif GPIO.input(23) == 0: # Button to rewind 10 sec
        send_command = 'echo seek -10 0 > video_fifo'
        subprocess.check_output(send_command, shell = True)
        time.sleep(.3) # Prevent button bouncing
    elif GPIO.input(27) == 0: # Button to quit mplayer
        send_command = 'echo quit > video_fifo'
        subprocess.check_output(send_command, shell = True)
        break # Quit the program

```

```
Start_video:
#!/bin/bash

# Jonathan Nusantara (jan265), Eric Hall (ewh73), Eric Kahn (edk52)
# Lab 1, 09-27-2020

# Will run in background
python video_control.py &
# Run in foreground
sudo SDL_VIDEODRIVER=fbcon SDL_FBDEV=/dev/fb1 mplayer -input
file=video_fifo -vo sdl -framedrop bigbuckbunny320p.mp4
```