Lab Manual for ICEN 553/453
Cyber-Physical Systems
Fall 2018

Prof. Dola Saha
Assistant Professor
Department of Electrical & Computer Engineering
University at Albany, SUNY

Chapter 1

Setup Headless Raspberry Pi

This tutorial is to setup the Raspberry Pi without requirement of any keyboard, mouse or monitor. You should be able to `ssh` into the Pi from your laptop using an Ethernet cable.

1.1 Boot up Raspberry Pi for the first time

1. Download Raspbian from the official Raspberry Pi website (`https://www.raspberrypi.org/downloads/raspbian/`).
2. Use the tool Etcher (`https://etcher.io`) to burn the Raspbian in the Micro SD card.
3. Create an empty file in `boot` partition of the Micro SD card and name it `ssh` without any extension. Use commands `touch` in Linux/OSX or `fsutil` in Windows. This enables `ssh` in Raspberry Pi.
4. Insert the Micro SD card in the Raspberry Pi and power it. This will boot up in Raspbian with SSH being enabled.

1.2 Login to your Raspberry Pi and setup hostname

1. Connect Raspberry Pi to your Home Router using Ethernet cable.
2. From your laptop (also connected to your Home Router by wired or wireless connection), use ssh to connect to the Raspberry Pi. The default values are:
   `hostname: raspberrypi`
   `username: pi`
   `password: raspberry`
   If you are using Linux or OSX, you can use the following command to ssh in with X-forwarding enabled: `ssh -X pi@raspberrypi.local`
   If you are using Windows, choose PuTTY to ssh in with the same credentials.
3. Expand the Filesystem. Use the following command: `sudo raspi-config`. Choose Option 7, Advanced Options, then choose A1 Expand Filesystem. Save the changes.
4. Change the hostname. Use `sudo raspi-config`. Choose Option 2, Network Options. Then choose N1 Hostname. Enter the hostname of your choice, for example mine is `sahaPi`. Reboot for the changes to be affected.

1.3 Create new User

1. Add new user. In terminal, use the following command. `sudo adduser newusername`, for example I used `sudo adduser dsaha` You will be asked to provide password.
2. Add user to sudo group using the following command: `sudo usermod -aG sudo dsaha`. Check if the command worked without any error. The output of the following command will show both `pi` and

`newusername` as the sudo users. `cat /etc/group | grep sudo`

3. Reboot as new user, '`dsaha`' in my case.
4. Use the following two commands to disconnect user `pi` from all default services.
   a) Open the file `/etc/lightdm/lightdm.conf`. `sudo nano /etc/lightdm/lightdm.conf`
   Change the line: `autologin-user=pi` to `autologin-user=dsaha`
   b) In command line, use the command: `sudo systemctl stop autologin@tty1`
5. Delete the user `pi`. Use the command: `sudo userdel pi`
6. Shutdown the Pi using the command `sudo shutdown -h now`

1.4 Connect directly to your laptop

1. Now connect the Raspberry Pi directly to your laptop using Ethernet cable. We will use Laptop's Wi-Fi connection to access the Internet and Ethernet to communicate to the Raspberry Pi.
2. Share the Internet Connection in your Laptop. Use the appropriate link for your Operating System and version to enable this.

   Windows: `https://answers.microsoft.com/en-us/windows/forum/windows_10-networking/internet-connection-sharing-in-windows-10/f6dcac4b-5203-4c98-8cf2-dcac86d98fb9`
   Ubuntu : `https://help.ubuntu.com/community/Internet/ConnectionSharing`
   MAC : `https://support.apple.com/kb/ph25327?locale=en_US`
3. SSH directly to the Pi. Use `ssh -X username@hostname.local`. In my case, it is `ssh -X dsaha@sahaPi.local`.

Chapter 2

Basic Input and Output

2.1 The First Circuit

We will use GPIO to Breadboard Interface Board with GPIO Ribbon Cable to connect the Raspberry PI for ease of use. Figure 2.1 shows the connection. Make sure the red line of the breadboard is aligned with positive voltage in the interface board, as shown in Figure 2.2.

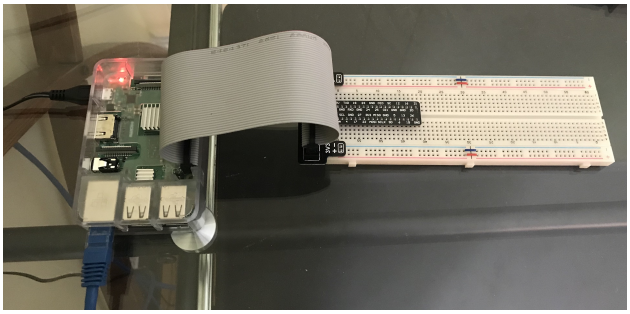
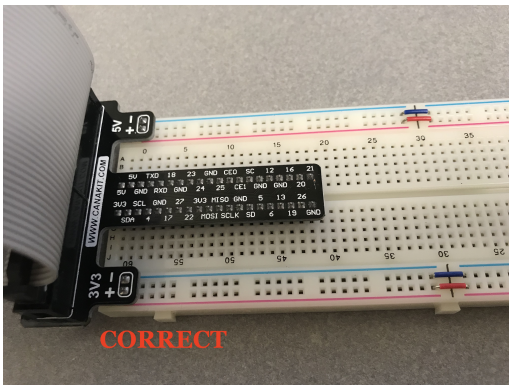
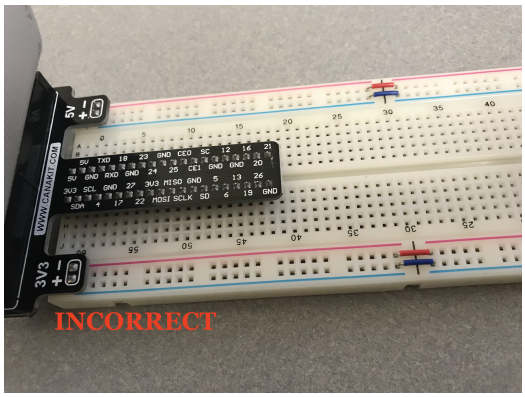


Figure 2.1: GPIO to Breadboard Interface Board and Ribbon Cable.



(a) Red line matching positive.

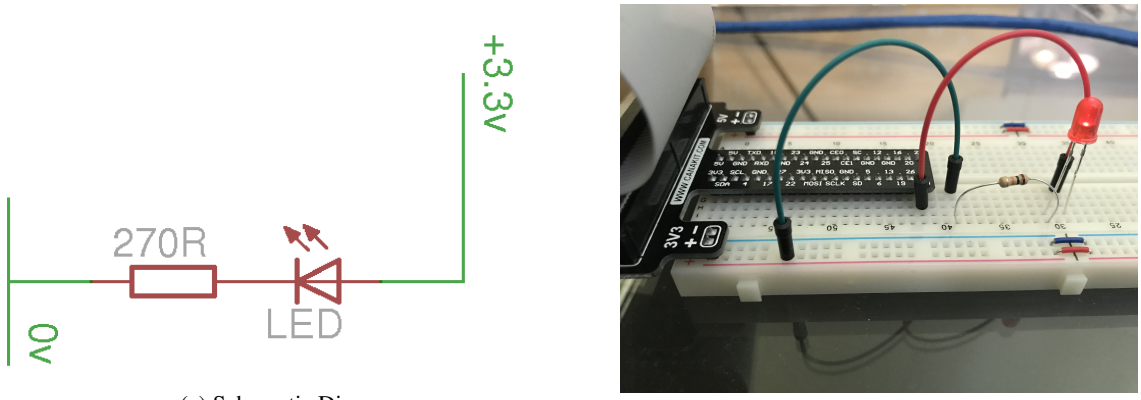


(b) Red line not matching positive.

Figure 2.2: Breadboard.

The first circuit that we will work on is shown in Figure 2.3. It is the simplest circuit, where the circuit is always connected to +3.3V line. This will keep the LED turned on until the circuit is disconnected.

Once you complete the circuit and LED is turned on, you have completed the first circuit. Congratulations!



(a) Schematic Diagram.

(b) Circuit on breadboard.

Figure 2.3: LED Resistor Circuit.

2.2 Programming The First Circuit Using `sysfs`

We will use the GPIO pins to program the controlling of the LED. Our Raspberry Pi uses Raspbian based on Debian and optimized for the Raspberry Pi hardware. We will use `sysfs` to access the kernel space for controlling the GPIO pins. Change the circuit to get power from GPIO pin 4 instead of 3.3V as shown in figure 2.4.

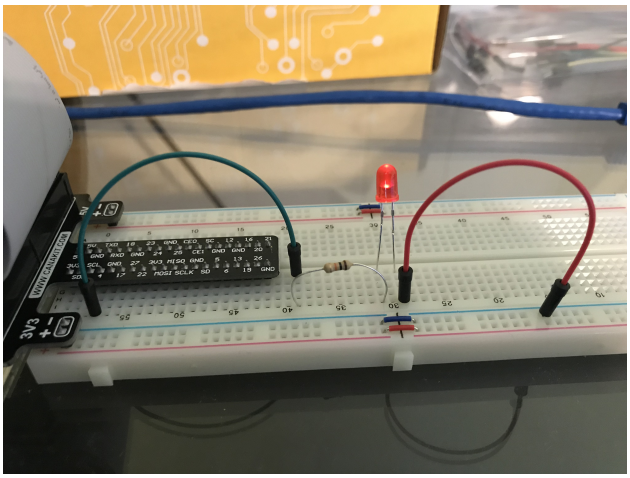


Figure 2.4: LED Resistor Circuit connected to GPIO Pin 4.

The steps for controlling the GPIO pin using `sysfs` is given below.

1. Become the Sudo user to access `sysfs`.

```
dsaha@sahaPi:~ $ sudo su
```

2. Go to the GPIO folder and list the contents

```
root@sahaPi:/home/dsaha# cd /sys/class/gpio/
root@sahaPi:/sys/class/gpio# ls
export gpiochip0 gpiochip128 unexport
```

3. Export gpio 4

```
root@sahaPi:/sys/class/gpio# echo 4 > export
root@sahaPi:/sys/class/gpio# ls
export gpio4 gpiochip0 gpiochip128 unexport
```

4. Go to the gpio4 folder and list contents

```
root@sahaPi:/sys/class/gpio# cd gpio4/
root@sahaPi:/sys/class/gpio/gpio4# ls
active_low device direction edge power subsystem uevent value
```

5. Set direction (in or out) of pin

```
root@sahaPi:/sys/class/gpio/gpio4# echo out > direction
```

6. Set value to be 1 to turn on the LED

```
root@sahaPi:/sys/class/gpio/gpio4# echo 1 > value
```

7. Set value to be 0 to turn off the LED

```
root@sahaPi:/sys/class/gpio/gpio4# echo 0 > value
```

8. Check the status (direction and value) of the pin

```
root@sahaPi:/sys/class/gpio/gpio4# cat direction
out
root@sahaPi:/sys/class/gpio/gpio4# cat value
0
```

9. Ready to give up the control? Get out of gpio4 folder and list contents, which shows gpio4 folder

```
root@sahaPi:/sys/class/gpio/gpio4# cd ../
root@sahaPi:/sys/class/gpio# ls
export gpio4 gpiochip0 gpiochip128 unexport
```

10. Unexport gpio 4 and list contents showing removal of gpio4 folder

```
root@sahaPi:/sys/class/gpio# echo 4 > unexport
root@sahaPi:/sys/class/gpio# ls
export gpiochip0 gpiochip128 unexport
```

2.3 The First Circuit Using Bash, Python and C

Listing 2.1 [1] shows the bash script to turn on or off the LED using GPIO pins. Note that each step in the script is same as shown in §2.2. Listings 2.2 and 2.3 show similar procedure in Python and C languages respectively.

```
#!/bin/bash
#  A small Bash script to turn on and off an LED that is attached to GPIO 4
#  using Linux sysfs. Written by Derek Molloy (www.derekmolloy.ie) for the
#  book Exploring Raspberry PI
LED_GPIO=4    # Use a variable -- easy to change GPIO number

# An example Bash functions
function setLED
{ # $1 is the first argument that is passed to this function
  echo $1 >> "/sys/class/gpio/gpio$LED_GPIO/value"
}

# Start of the program -- start reading from here
if [ $# -ne 1 ]; then    # if there is not exactly one argument
```

[1] This part of the lab follows the book "Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux", by Derek Molloy, Wiley, ISBN 978-1-119-18868-1, 2016.

```
  echo "No command was passed. Usage is: bashLED command,"
  echo "where command is one of: setup, on, off, status and close"
  echo -e "e.g., bashLED setup, followed by bashLED on"
  exit 2      # error that indicates an invalid number of arguments
fi
echo "The LED command that was passed is: $1"
if [ "$1" == "setup" ]; then
  echo "Exporting GPIO number $1"
  echo $LED_GPIO >> "/sys/class/gpio/export"
  sleep 1     # to ensure gpio has been exported before next step
  echo "out" >> "/sys/class/gpio/gpio$LED_GPIO/direction"
elif [ "$1" == "on" ]; then
  echo "Turning the LED on"
  setLED 1    # 1 is received as $1 in the setLED function
elif [ "$1" == "off" ]; then
  echo "Turning the LED off"
  setLED 0    # 0 is received as $1 in the setLED function
elif [ "$1" == "status" ]; then
  state=$(cat "/sys/class/gpio/gpio$LED_GPIO/value")
  echo "The LED state is: $state"
elif [ "$1" == "close" ]; then
  echo "Unexporting GPIO number $LED_GPIO"
  echo $LED_GPIO >> "/sys/class/gpio/unexport"
fi
```

Listing 2.1: LED code in Bash script

```
#!/usr/bin/python2
# A small Python program to set up GPIO4 as an LED that can be
# turned on or off from the Linux console.
# Written by Derek Molloy for the book "Exploring Raspberry Pi"

import sys
from time import sleep
LED4_PATH = "/sys/class/gpio/gpio4/"
SYSFS_DIR = "/sys/class/gpio/"
LED_NUMBER = "4"

def writeLED ( filename, value, path=LED4_PATH ):
   "This function writes the value passed to the file in the path"
   fo = open( path + filename,"w")
   fo.write(value)
   fo.close()
   return

print "Starting the GPIO LED4 Python script"
if len(sys.argv)!=2:
   print "There is an incorrect number of arguments"
   print "  usage is:  pythonLED.py command"
   print "  where command is one of setup, on, off, status, or close"
   sys.exit(2)
if sys.argv[1]=="on":
   print "Turning the LED on"
   writeLED (filename="value", value="1")
elif sys.argv[1]=="off":
```

```
    print "Turning the LED off"
    writeLED (filename="value", value="0")
elif sys.argv[1]=="setup":
    print "Setting up the LED GPIO"
    writeLED (filename="export", value=LED_NUMBER, path=SYSFS_DIR)
    sleep(0.1);
    writeLED (filename="direction", value="out")
elif sys.argv[1]=="close":
    print "Closing down the LED GPIO"
    writeLED (filename="unexport", value=LED_NUMBER, path=SYSFS_DIR)
elif sys.argv[1]=="status":
    print "Getting the LED state value"
    fo = open( LED4_PATH + "value", "r")
    print fo.read()
    fo.close()
else:
    print "Invalid Command!"
print "End of Python script"
```

Listing 2.2: LED code in Python

```
/** Simple On-board LED flashing program - written in C by Derek Molloy
*    simple functional struture for the Exploring Raspberry Pi book
*
*    This program uses GPIO4 with a connected LED and can be executed:
*         makeLED setup
*         makeLED on
*         makeLED off
*         makeLED status
*         makeLED close
*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define GPIO_NUMBER "4"
#define GPIO4_PATH "/sys/class/gpio/gpio4/"
#define GPIO_SYSFS "/sys/class/gpio/"

void writeGPIO(char filename[], char value[]){
   FILE* fp;                          // create a file pointer fp
   fp = fopen(filename, "w+");        // open file for writing
   fprintf(fp, "%s", value);          // send the value to the file
   fclose(fp);                        // close the file using fp
}

int main(int argc, char* argv[]){
   if(argc!=2){                       // program name is argument 1
      printf("Usage is makeLEDC and one of:\n");
      printf("    setup, on, off, status, or close\n");
      printf("  e.g. makeLEDC on\n");
      return 2;                       // invalid number of arguments
   }
   printf("Starting the makeLED program\n");
```

```
    if(strcmp(argv[1],"setup")==0){
        printf("Setting up the LED on the GPIO\n");
        writeGPIO(GPIO_SYSFS "export", GPIO_NUMBER);
        usleep(100000);                    // sleep for 100ms
        writeGPIO(GPIO4_PATH "direction", "out");
    }
    else if(strcmp(argv[1],"close")==0){
        printf("Closing the LED on the GPIO\n");
        writeGPIO(GPIO_SYSFS "unexport", GPIO_NUMBER);
    }
    else if(strcmp(argv[1],"on")==0){
        printf("Turning the LED on\n");
        writeGPIO(GPIO4_PATH "value", "1");
    }
    else if (strcmp(argv[1],"off")==0){
        printf("Turning the LED off\n");
        writeGPIO(GPIO4_PATH "value", "0");
    }
    else if (strcmp(argv[1],"status")==0){
        FILE* fp;          // see writeGPIO function above for description
        char line[80], fullFilename[100];
        sprintf(fullFilename, GPIO4_PATH "/value");
        fp = fopen(fullFilename, "rt");          // reading text this time
        while (fgets(line, 80, fp) != NULL){
            printf("The state of the LED is %s", line);
        }
        fclose(fp);
    }
    else{
        printf("Invalid command!\n");
    }
    printf("Finished the makeLED Program\n");
    return 0;
}
```

Listing 2.3: LED code in C

2.4 Simple Circuit Using Python Libraries

2.4.1 RPi Library

In this section, we will use `RPi` Python library to demonstrate similar LED switching functions. Listing 2.4 shows a simple code in Python for GPIO pin manipulation. Check the website [`https://sourceforge.net/projects/raspberry-gpio-python/`] to learn other functions available through this library. Note that the current release does not support SPI, I2C, 1-wire or serial functionality on the RPi yet.

```
import RPi.GPIO as GPIO
from time import sleep

ledPin = 4                          # GPIO Pin Number, where LED is connected

GPIO.setmode(GPIO.BCM)              # Broadcom pin-numbering scheme
GPIO.setup(ledPin, GPIO.OUT)        # LED pin set as output

GPIO.output(ledPin, GPIO.HIGH)      # Turn the LED on
```

```
sleep(1)                          # Sleep for 1 sec
GPIO.output(ledPin, GPIO.LOW)     # Turn the LED off
```

Listing 2.4: LED code using RPi Library in Python

2.4.2 GPIOZero Library

Another efficient Python library is `gpiozero` [`https://gpiozero.readthedocs.io/en/stable/`]. It provides a simple interface to GPIO devices with Raspberry Pi. It was created by Ben Nuttall of the Raspberry Pi Foundation, Dave Jones, and other contributors. Listing 2.5 shows a simple code for switching on and off the LED.

```
from gpiozero import LED
from time import sleep

led = LED(4)      # GPIO Pin Number
led.on()          # Turn on LED
sleep(1)          # Sleep for 1 sec
led.off()         # Turn off LED
```

Listing 2.5: LED code using GPIOZero Library in Python

2.5 Use GPIO Pins for Input

In the next experiment, we will use GPIO Pin as an input. The circuit is shown as in figure 2.5. A button switch, when pressed, will be detected by the program and a message will be printed accordingly. In this case, the process continuously polls the status of the input pin. Listings 2.6 and 2.7 shows the code using `RPi` and `gpiozero` libraries.

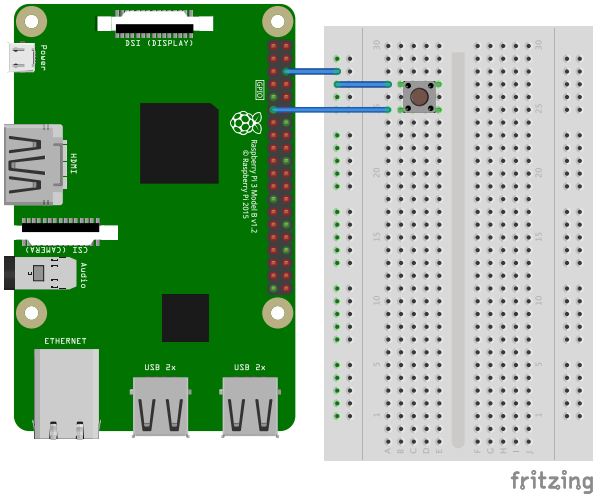


Figure 2.5: Button Switch connected to GPIO Pin.

```
import RPi.GPIO as GPIO
import time

buttonPin=17      # GPIO Pin Number where Button Switch is connected

GPIO.setmode(GPIO.BCM)             # Broadcom pin-numbering scheme
GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
# Button pin set as input
```

```
while True:                                 # Monitor continuously
    input_state = GPIO.input(buttonPin) # Get the input state
    if input_state == False:                # Check status
        print('Button Pressed')             # Print
        time.sleep(0.2)                     # Sleep before checking again
```

Listing 2.6: Button Press detection code using RPi Library in Python

```
from gpiozero import Button
import time

button = Button(17) # GPIO Pin Number where Button Switch is connected

while True:                                 # Monitor continuously
    if button.is_pressed:                   # Check Status
        print("Button Pressed")             # Print
        time.sleep(0.2)                     # Sleep before checking again
```

Listing 2.7: Button Press detection code using GPIOZero Library in Python