Lab Manual for IECE 553/453
Cyber-Physical Systems
Fall 2019

Prof. Dola Saha
Assistant Professor
Maqsood Abdul Careem
PhD Student and Teaching Assistant of CPS

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 (Desktop version) 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`.
   If you are NOT able to delete the user `pi`, perform the following steps:
   - Use the command, `sudo raspi-config`.
   - Choose: Boot Options → Desktop/CLI → Console AutoLogin: Text console, automatically logged in as 'new user'. Save changes.
   - You will be prompted to reboot. If not reboot manually using `sudo reboot`.
   - Remotely connect to the raspberry pi again as in Step 1.2.2.
   - Remove the default user `pi` using 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`.

1.5 Initial setup

1. Make sure that the Raspberry Pi is up to date with the latest versions of Raspbian: (this is a good idea to do regularly, anyway).

   ```
   sudo apt-get update
   sudo apt-get upgrade
   ```
2. Setting up RPi libraries,
   The RPi.GPIO module is installed by default in Raspbian Desktop image. To make sure that it is at the latest version:
   `sudo apt-get install python-rpi.gpio python3-rpi.gpio`
3. Setting up GPIO Zero libraries,
   GPIO Zero is installed by default in the Raspbian image, and the Raspberry Pi Desktop image. However it can be installed or updated using,
   `sudo apt install python3-gpiozero` (for Python 3)
   `sudo apt install python-gpiozero` (for Python 2)

4. Setting up Exploring Raspberry pi libraries,
   `sudo apt-get install git`
   `git clone https://github.com/derekmolloy/exploringrpi.git`
5. WiringPi is pre-installed with standard Raspbian systems. However it can be installed or updated using,
   `sudo apt-get install wiringpi`.

1.6 Shutdown and Restart

After every use, make sure to properly shutdown the Pi using the command:
`sudo shutdown -h now`.
To reboot the Pi, use the following command:
`sudo reboot`

Chapter 2

Basic Input and Output Using Pseudo Filesystem

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.

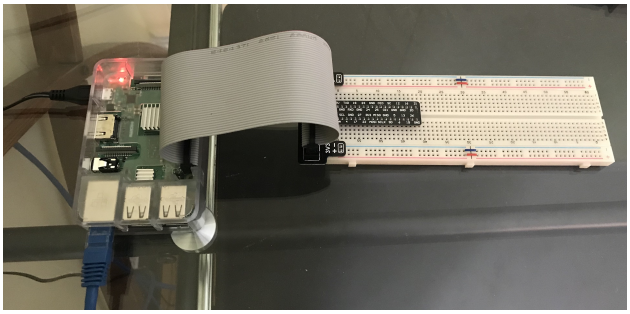
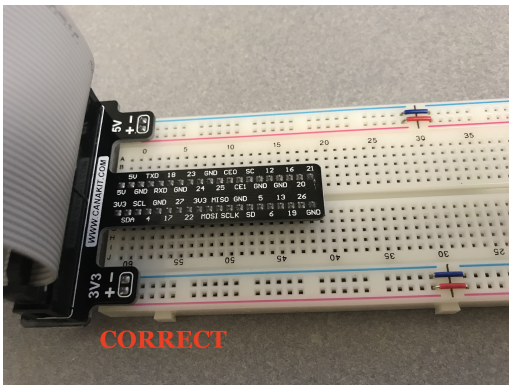
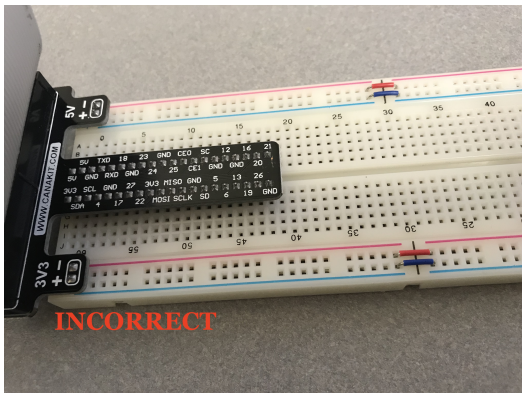


Figure 2.1: GPIO to Breadboard Interface Board and Ribbon Cable.



(a) Red line matching positive.

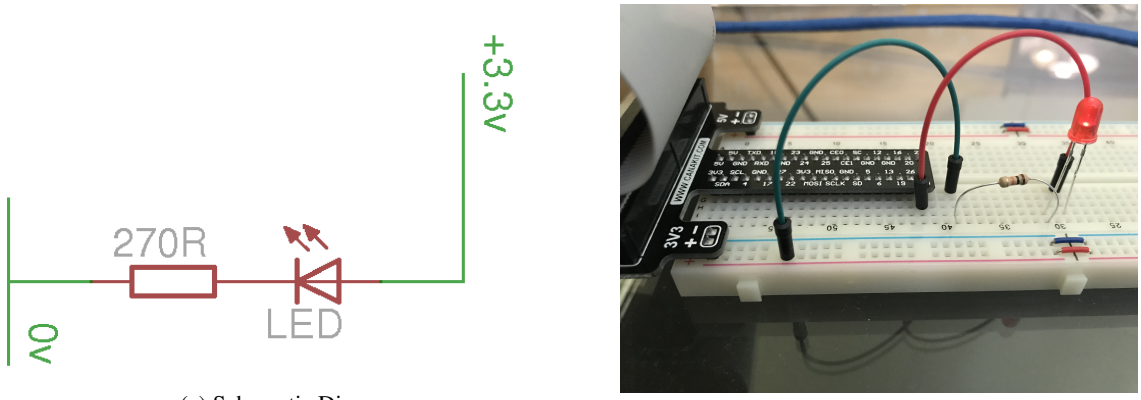


(b) Red line not matching positive.

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.

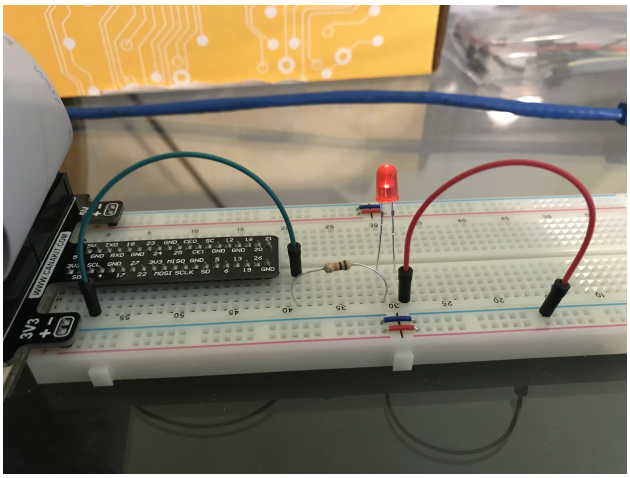


Figure 2.4: LED Resistor Circuit connected to GPIO Pin 4.

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.

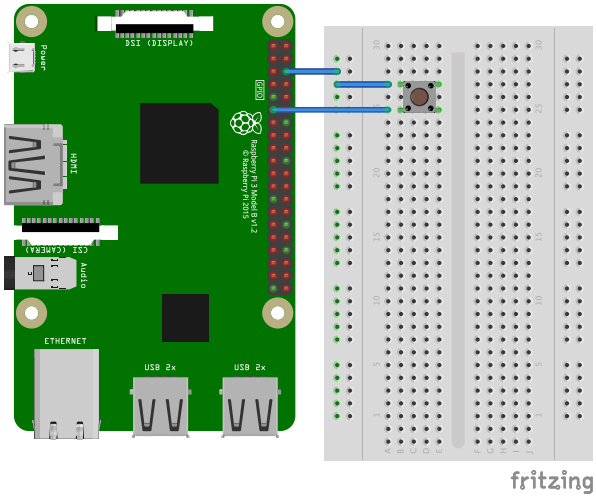


Figure 2.5: Button Switch connected to GPIO Pin.

```
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

Chapter 3

Basic Input and Output Using Address Map

In this lab, we will use memory mapped I/O to a) change the GPIO pin output and b) control the pull-up/down resistor configuration.

3.1 Memory Mapped Input and Output Access

All the instructions below must be executed with root privilege.

1. Get the file describing the *Address Mapping* of the system's memory for each physical device, using the command:
   `cat /proc/iomem`
   The output would be an *Address Mapping* of the form
   `3f200000-3f2000b3 : gpio@7e200000`
   The first column displays the memory registers used by each of the different types of memory. The second column lists the kind of memory located within those registers and displays which memory registers are used by the kernel within the system RAM or, if the network interface card has multiple Ethernet ports, the memory registers assigned for each port.
2. /dev/mem provides access to the system's physical memory. It is primarily used to access IO memory addresses related to peripheral hardware.
   Get the file for direct memory access using the command.
   `wget http://www.lartmaker.nl/lartware/port/devmem2.c`
   If you do not have access to internet on the Raspberry Pi, download the file from URL above to your machine, and copy the contents to the Raspberry Pi.

   Compile the code using
   `gcc devmem2.c -o devmem2`
   Usage: ./devmem2 address [type [data]] address : memory address to act upon type: access operation type : [b]yte, [h]alfword, [w]ord data: data to be written

3.2 GPIO Pull-up/down Register Control

Pull-down and Pull-up Resistors are used to ensure that the switches do not create floating inputs. We will configure the Internal pull-up/pull-down Resistors using memory based GPIO control. The GPIO Pull-up/down Register controls the actuation of the internal pull-up/down control line to ALL the GPIO pins. This register must be used in conjunction with the 2 GPPUDCLKn registers.

We will change the Pull-up/ pull-down configurations of GPIO pin 4. For this we set bit 4 on the GPPUDCLK0 register, clear the GPPUD register, and then remove the clock control signal from GPPUDCLK0. GPIO4 is bit 4, which is '10000' in binary ($0x10_{16}$).

Get the value of GPIO pin 4 using the `sysfs` file system.

3.2.1 Pull-down Configuration

1. GPPUD Enable Pull-down
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x01
   ```

2. GPPUDCLK0 enable GPIO 4
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x10
   ```

3. GPPUD Disable Pull-down
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x00
   ```

4. GPPUDCLK0 disable Clk signal
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x00
   ```

5. cat value
   ```
   value:  0
   ```

3.2.2 Pull-up Configuration

1. GPPUD Enable Pull-up
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x02
   ```

2. GPPUDCLK0 enable GPIO 4
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x10
   ```

3. GPPUD Disable Pull-up
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x00
   ```

4. GPPUDCLK0 disable Clk signal
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x00
   ```

5. cat value
   ```
   value:  1
   ```

3.3 GPIO Contol

3.3.1 GPIO Output Example

In this code we will turn on an LED using the Address map. All of the GPIOs can be set to read or write mode, or they can be set to an ALT mode. The mode is set using a 3-bit value, for example, by setting the 3-bit value to be 001 then the pin will act as an output. The location to which you should write the 3-bit mode is described in the BCM 2837 Manual. For example, to set GPIO17 to be an output, you can write 001 to the FSEL17 value, which is bits 21, 22, and 23 in the GPFSEL1 register. Importantly, you need to ensure that you only manipulate those specific 3 bits when you change FSEL17, because to change any other bits will impact on GPIO10-GPIO19, likely changing their pin modes.

Set up the circuit similar to figure 2.3 so that an LED can be turned on via GPIO17.

1. Read the status of the Function Select Registry
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200004
   ```

2. Set GPIO17 to be an output (Bits 23,22,21 are set to 001 in GPFSEL1 registry)
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200004 w 0x200000
   ```

3. Check the value/status of GPIO17 pi (using the GPVL0 registry)
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200034
   ```

4. Clear the output of GPIO17 (Bit 17 is set to 1 in GPCLR0 registry)
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200028 w 0x20000
   ```

5. Set the output of GPIO17 to 1 (Bit 17 is set to 1 in GPSET0 registry)
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F20001C w 0x20000
   ```

6. Set GPIO17 to be an input (Bits 23,22,21 are set to 000 in GPFSEL1 registry)
   ```
   sudo /home/dsaha/myCode/devmem2 0x3F200004 w 0x00
   ```

Chapter 4

Analog Output: PWM (Pulse Width Modulation)

The RPi has pulse-width modulation (PWM) capabilities that can provide digital-to-analog conversion (DAC), or generate control signals for motors and certain types of servos. The number of PWM outputs is very limited on the RPi boards. The RPi B+ model has two PWMs (PWM0 & PWM1) output at Pins 12 and 33 (GPIO18, GPIO13).

$$PWM\ frequency = \frac{19.2MHz}{(divisor \times range)} \tag{4.1}$$

The PWM device on the RPi is clocked at a fixed base-clock frequency of 19.2 MHz, and therefore integer divisor and range values are used to tailor the PWM frequency for your application according to the following expression: where the range is subsequently used to adjust the duty cycle of the PWM signal. RPi PWMs share the same frequency but have independent duty cycles. The default PWM mode of operation on the RPi is to use balanced PWM. Balanced PWM means that the frequency will change as the duty cycle is adjusted, therefore to control the frequency you need to use the call `pwmSetMode(PWM_MODE_MS)` to change the mode to mark-space.

4.1 Hard PWM

Listing 4.1 shows a PWM example, which uses both PWMs on the RPi to generate two signals with different duty cycles. The script can be accessed by navigating to
`exploringPi/chp06/wiringPi/pwm.cpp`

```
#include <iostream>
#include <wiringPi.h>
using namespace std;

#define PWM0        12                  // this is physical pin 12
#define PWM1        33                  // only on the RPi B+/A+/2

int main() {                            // must be run as root
   wiringPiSetupPhys();                 // use the physical pin numbers
   pinMode(PWM0, PWM_OUTPUT);           // use the RPi PWM output
   pinMode(PWM1, PWM_OUTPUT);           // only on recent RPis

   // Setting PWM frequency to be 10kHz with a full range of 128 steps
   // PWM frequency = 19.2 MHz / (divisor * range)
   // 10000 = 19200000 / (divisor * 128) => divisor = 15.0 = 15
   pwmSetMode(PWM_MODE_MS);             // use a fixed frequency
   pwmSetRange(128);                    // range is 0-128
   pwmSetClock(15);                     // gives a precise 10kHz signal
   cout << "The PWM Output is enabled" << endl;
   pwmWrite(PWM0, 32);                  // duty cycle of 25% (32/128)
   pwmWrite(PWM1, 64);                  // duty cycle of 50% (64/128)
```

```
    return 0;                                   // PWM output stays on after exit
}
```

Listing 4.1: PWM code in C++

1. Implement the circuit corresponding to the above script to turn on two LEDs using an Analog output. LEDs are current-controlled devices, so PWM is typically employed to provide brightness-level control. This is achieved by flashing the LED faster than can be perceived by a human, where the amount of time that the LED remains on, versus off (i.e., the duty cycle) affects the human-perceived brightness level.
   Note: To compile this code use: `g++ pwm.cpp -o pwm -lwiringPi`.
2. **Exercise:** Try changing the PWM output of the two pins to gradually increase and reduce the brightness of the LEDs.

4.1.1 PWM Application- Fading an LED

Listing 4.2 provides a code example for slowly fading an LED on and off using PWM until a pushbutton is pressed. This example employs an ISR on the button press to ensure that the program ends gracefully, having completed a full fade cycle.

```
#include <iostream>
#include <wiringPi.h>
#include <unistd.h>
using namespace std;
#define PWM_LED         18              // this is PWM0, Pin 12
bool running = true;                    // fade in/out until button pressed
int main() {                                    // must be run as root
    wiringPiSetupGpio();                        // use the GPIO numbering
    pinMode(PWM_LED, PWM_OUTPUT);               // the PWM LED - PWM0
    cout << "Fading the LED in/out until the button is pressed" << endl;
    while(running) {
        for(int i=1; i<=1023; i++) {            // Fade fully on
            pwmWrite(PWM_LED, i);
            usleep(1000);
        }
        for(int i=1022; i>=0; i--) {            // Fade fully off
            pwmWrite(PWM_LED, i);
            usleep(1000);
        }
    }
    cout << "LED Off: Program has finished gracefully!" << endl;
    return 0;
}
```

Listing 4.2: PWM code in C++

1. Implement the circuit corresponding to the above script to turn on two LEDs using an Analog output.
2. **Exercise:** Use the Push Button to turn off the LED and terminate the script gracefully.

4.2 Soft PWM

It is possible to use software PWM on the other GPIO pins by toggling the GPIO, but this approach has a high CPU cost and is only suitable for low-frequency PWM signals. Alternatively, additional circuity can be used to add hardware PWMs to each I^2C bus. WiringPi includes a software-driven PWM handler capable of outputting a PWM signal on any of the Raspberry Pis GPIO pins. An example of Software PWM is shown in Listing 4.3. Implement the

circuit corresponding to the above script to turn on two LEDs using a Sofware PWM output. Look at the header file `WiringPi/wiringPi/softPwm.c` to understand the implementation of Software PWM in WiringPi. `exploringPi/chp06/wiringPi/softPWM.cpp`

```c
#include <wiringPi.h>
#include <softPwm.h>

#define GPIO1 4
#define GPIO2 17

int main(int argc, char *argv[])
{
    if (wiringPiSetupGpio() < 0) return 1;
        pinMode(GPIO1, OUTPUT);
        digitalWrite(GPIO1, LOW);
        softPwmCreate(GPIO1, 0, 200);
        softPwmWrite(GPIO1, 15);

        pinMode(GPIO2, OUTPUT);
        digitalWrite(GPIO2, LOW);
        softPwmCreate(GPIO2, 0, 500);
        softPwmWrite(GPIO2, 15);
        sleep(30);
}
```

Listing 4.3: PWM code in C

This can also be done using the gpiozero using the following scripts.

```python
from gpiozero import PWMLED
from time import sleep

led = PWMLED(17)

while True:
    led.value = 0  # off
    sleep(1)
    led.value = 0.5  # half brightness
    sleep(1)
    led.value = 1  # full brightness
    sleep(1)
```

Listing 4.4: LED code in Python

```python
from gpiozero import PWMLED
from signal import pause

led = PWMLED(17)

led.pulse()

pause()
```

Listing 4.5: LED code in Python

Chapter 5

Analog Input

In this lab we will work learn how to use the Raspberry pi for Analog input through various examples. We will rely on the *Adeept sensor kit* and the *wiringPi* library for the following labs.

The default configurations of the RPi GPIO pins can be viewed by querying the following:
`sudo gpio readall`
This will display the default configurations as shown in the figure below. If the output below is visible, it also means that wiringPi is successfully installed.

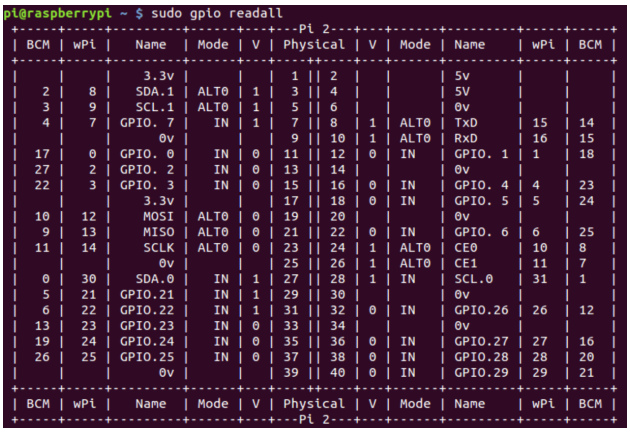


Figure 5.1: Default GPIO configurations.

5.1 Application 1 - Temperature & Humidity Sensor

Sensor: In this section we will learn how to use DHT-11 to collect temperature and humidity data and display it on the terminal. DHT-11 features a temperature and humidity sensor module with a calibrated digital signal output. This sensor includes a resistive humidity measurement component and an NTC (Negative Temperature Coefficient) temperature measurement component, and connects to an 8-bit microcontroller to communicate with the RPi. Complete the circuit as shown in the figure below.

Communicating to One-Wire Sensors: DHT11 can digitally communicate with the RPi using a single GPIO. The GPIO can be set high and low with respect to time to send data bits to the sensor to initiate communication. The same GPIO can then be sampled over time to read the sensors response. The consistency of the sample time is vital for this application, because the data response is 40 bits long and takes less than 4.3 ms to transfer. Therefore, memory-mapped wiringPi code is used.

The script is available at

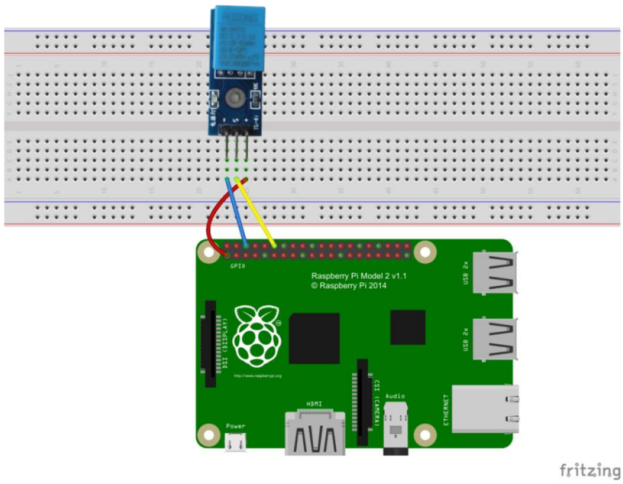


Figure 5.2: Temperature & Humidity Sensor circuit.

`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/15_DHT11/dht11.c`.

This script might produce incorrect results. Try to troubleshoot the issue by referring to the DHT-11 Manual:
`https://www.mouser.com/ds/2/758/DHT11-Technical-Data-Sheet-Translated-Version-1143054.pdf`.

5.2 Application 2: Photoresistor (Light Dependent Resistor)

In this section, we will learn how to measure the light intensity using a photoresistor and display the measurement result. A photoresistor is a light-controlled variable resistor. The resistance of a photoresistor decreases with increasing incident light intensity (in other words, it exhibits photoconductivity). A photoresistor can be applied in light-sensitive detector circuits. A photoresistor is made of a high resistance semiconductor. In the dark, a photoresistor can have a resistance as high as a few megohms ($M\Omega$), while in the light, a photoresistor can have a resistance as low as a few hundred ohms. If incident light on a photoresistor exceeds a certain frequency, photons absorbed by the semiconductor give bound electrons enough energy to jump into the conduction band. The resulting free electrons (and their hole partners) conduct electricity, thereby lowering resistance. The resistance range and sensitivity of a photoresistor can substantially differ among dissimilar devices. Moreover, unique photoresistors may react substantially differently to photons within certain wavelength bands.

The schematic diagram of this experiment is shown below:

The script is available at
`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/17_photoresistor/photoresistor.c`.

Now, when you try to block the light incident on the photoresistor, the value displayed on the screen reduces. When you use a powerful light to irradiate the photoresistor, the value displayed increases.

5.3 Application 3: Thermistor

In this section, we will learn how to use a thermistor to collect temperature by programming the Raspberry Pi and an ADC (ADC0832). A thermistor is a type of resistor whose resistance varies significantly with temperature, more so than in standard resistors. When the temperature increases, the thermistor resistance decreases; when the temperature decreases, the thermistor resistance increases. It can detect surrounding temperature changes in real time. In the experiment, we need an analog-digital converter (ADC) ADC0832 to convert analog signal into digital signal.

The script is available at
`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/18_thermistor/thermistor.c`.

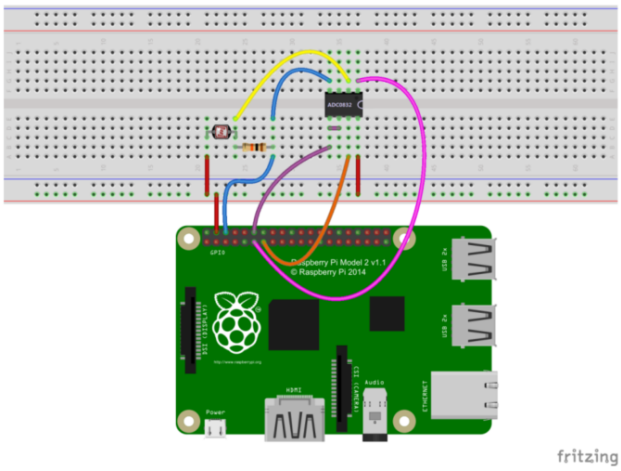


Figure 5.3: Photoresistor circuit.

Now, when you touch the thermistor, you can see current temperature value displayed on the screen change accordingly.

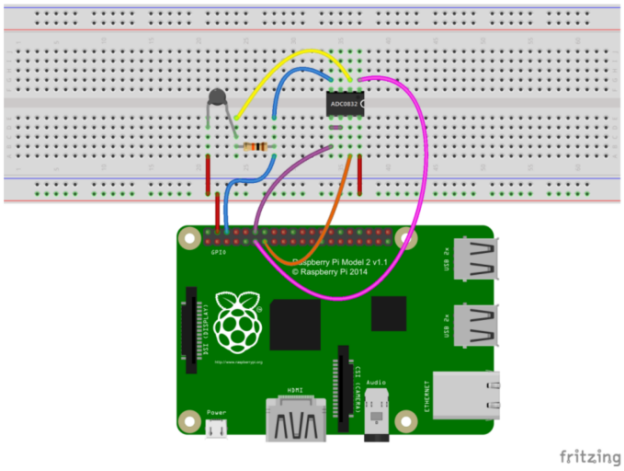


Figure 5.4: Thermistor circuit.

Chapter 6

Inter-Integrated Circuit (I2C) Communications

Enable the I^2C Interface in the RPi by doing the following:
`sudo raspi-config`
Choose Interfacing Options $\rightarrow$ I^2C $\rightarrow$ Yes.

To detect both available I^2C devices on the RPi use the following command:
`sudo i2cdetect -l`

The I^2C bus 1 can be probed to detect connected devices by using:
`sudo i2cdetect -y -r 1`

6.1 Application 1: Acceleration Sensor

In this lesson, we will learn how to use an acceleration sensor ADXL345 to get acceleration data. The datasheet can be found at `www.analog.com/ADXL345`.

The ADXL345 is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to 16g. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3-wire or 4-wire) or I2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 mg/LSB) enables measurement of inclination changes less than 1.0.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

NOTE: The following program use I2C interface. Before you run the program, please make sure the I2C driver module of Raspberry Pi is successfully loaded. The class `I2CDevice` captures the general functionality you would associate with an I2C bus device. You can extend this code to control any type of I2C device. It can be found in the `I2CDevice.cpp` and `I2CDevice.h` files in the `chp08/i2c/cpp/` directory.

You should see that the acceleration data will be displayed on the terminal.

The code to probe the output of the ADXL345 is at:
`exploringrpi/chp08/i2c/test/ADXL345.cpp`

Additional code is required to convert these values into pitch and roll form. The script corresponding to this is available at:
`exploringrpi/chp08/i2c/cpp/application.cpp`

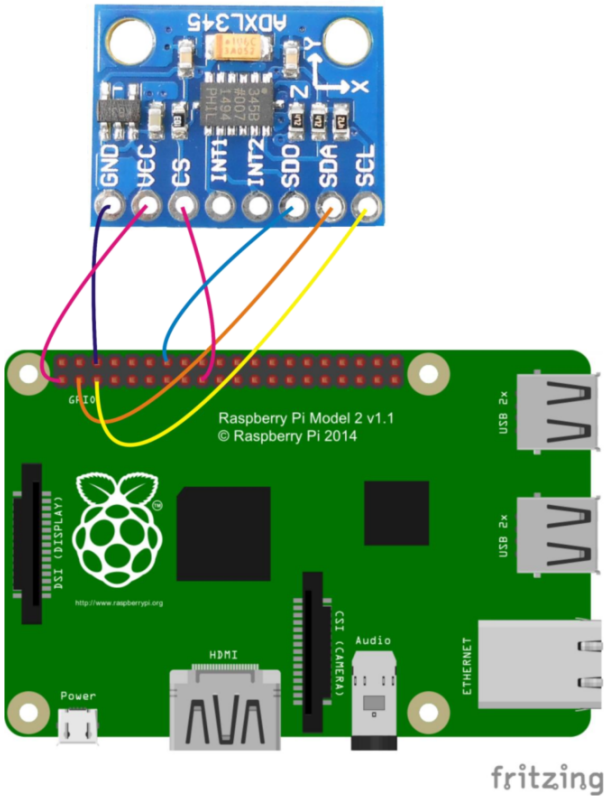


Figure 6.1: Acceleration Sensor circuit.

Chapter 7

SPI (Serial Peripheral Interface) Communication

We begin by enabling the SPI interface on the RPi by doing the following:
`sudo raspi-config`
Choose Interfacing Options → SPI → Yes.

First we will query the device ID by reading the register value at (0x00) of the ADXL345, which should return the DEVID (This value should be $E5_{16}$, which is 229_{10}). This can be done by using the script:
`/exploringrpi/chp08/spi/spiADXL345/spiADXL345.c`.
The output should be:

```
SPI mode:  3
Bits per word:  8
Speed:  5000000 Hz
Return value:  229
```

The maximum recommended SPI clock speed for the ADXL345 is 5 MHz, so this value is used in the program code.

A C++ class is available in `/chp08/spi/spiADXL345_cpp/SPIDevice.h` that wraps the software interface to the SPI bus. This SPI class can be used in a standalone form for any SPI device type. For example, `/chp08/spi/spiADXL345_cpp/SPITest.cpp` demonstrates how to probe the ADXL345 device.

The same SPIDevice class can be used as the basis for modifying the ADXL345 class discussed in the previous lab to support the SPI bus rather than the I2C bus:
`/chp08/spi/spiADXL345_cpp/ADXL345.h`.

These classes can be used to build a combined example as in:
`/chp08/spi/spiADXL345_cpp/testADXL345.cpp`
When this program is executed, it displays the current accelerometer pitch and roll values on a single line of the terminal window.

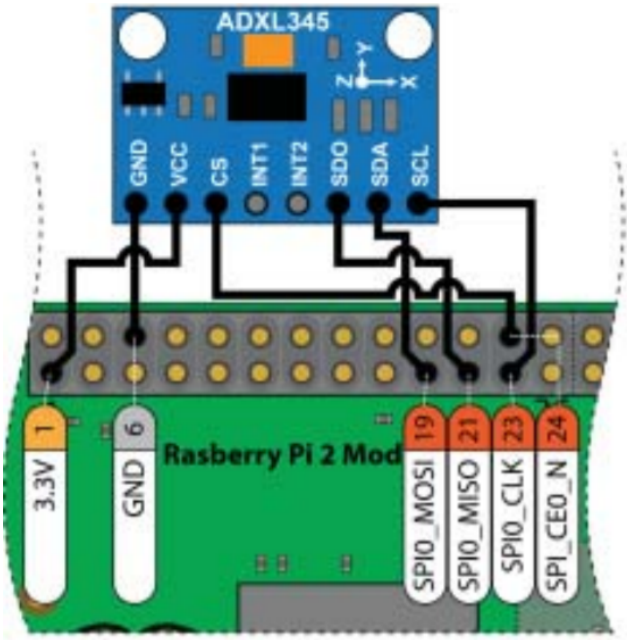


Figure 7.1: Acceleration Sensor circuit.

Chapter 8

ADC

The RPi does not have onboard analog-to-digital conversion (ADC) or digital-to-analog conversion (DAC) capabilities. We will focus on using discrete components and modules to provide the required expanded input/output functionality.

8.1 Application 1: A Simple Voltmeter

In this section we will create a simple voltmeter (with a range of 0 to 5V) using the Raspberry Pi and the LCD display (LCD1602). The core principle behind this voltmeter example is the ability to convert an analog voltage to a digital voltage via an ADC. We will vary the input voltage using a potentiometer. The analog voltage that is received via an ADC0832 IC is converted to a digital quantity via programming on the RPi, then displayed as the voltage on the LCD display. Complete the circuit as shown in the figure below:

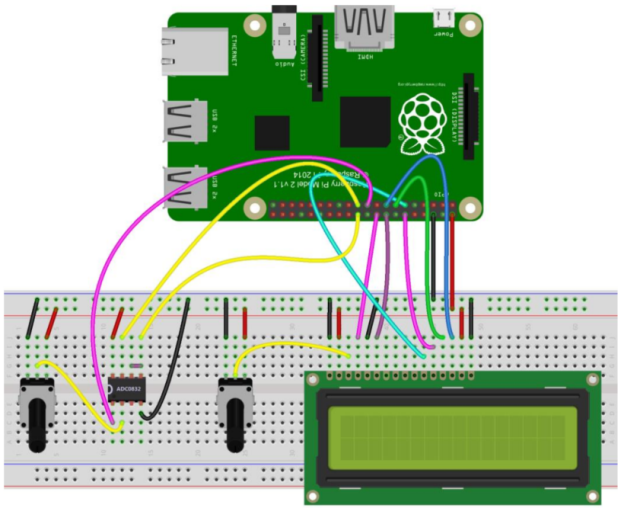


Figure 8.1: Voltmeter circuit.

8.2 Application 2: LED Bar Graph

In this section, we will learn how to control an LED bar graph by programming the Raspberry Pi.

The bar graph (a series of in-line LEDs), such as you see on an audio display - is a common hardware display for analog sensors. It's made up of a series of LEDs in a row, an analog input like a potentiometer, and a little code in between. You can buy multi-LED bar graph displays fairly cheaply. This tutorial demonstrates how to control a series of LEDs in a row, but can be applied to any series of digital outputs.

The sketch works like this: first you read the input. You map the input value to the output range, in this case ten LEDs. Then you set up a for loop to iterate over the outputs. If the output's number in the series is lower than the mapped input range, you turn it on. If not, you turn it off.

The schematic diagram of this experiment is shown below:

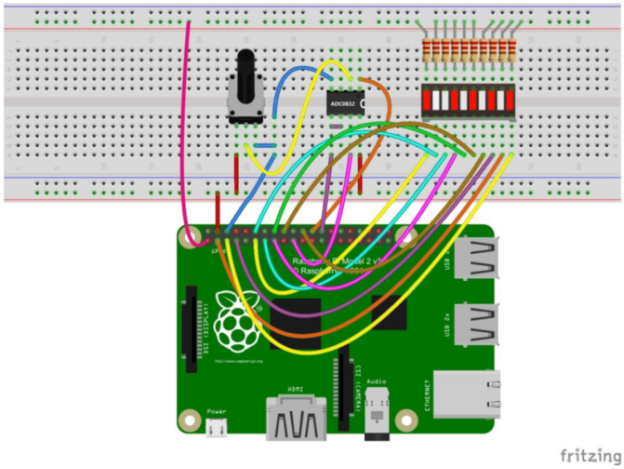


Figure 8.2: LED bar Graph circuit.

Now, when you turn the knob of the potentiometer, you will see that the number of LEDs in the LED bar graph will be changed.

8.3 Application 3: PS2 Joystick

In this section, we will learn the usage of joystick. We program the Raspberry Pi to detect the state of joystick. A joystick is an input device consisting of a stick that pivots on a base and reports its angle or direction to the device it is controlling. A joystick, also known as the control column, is the principal control device in the cockpit of many civilian and military aircraft, either as a center stick or side-stick. It often has supplementary switches to control various aspects of the aircraft's flight.

Joysticks are often used to control video games, and usually have one or more push-buttons whose state can also be read by the computer. A popular variation of the joystick used on modern video game consoles is the analog stick. Joysticks are also used for controlling machines such as cranes, trucks, underwater unmanned vehicles, wheelchairs, surveillance cameras, and zero turning radius lawn mowers. Miniature finger-operated joysticks have been adopted as input devices for smaller electronic equipment such as mobile phones.

Press Enter, you should see that the joystick state information displayed on the terminal.

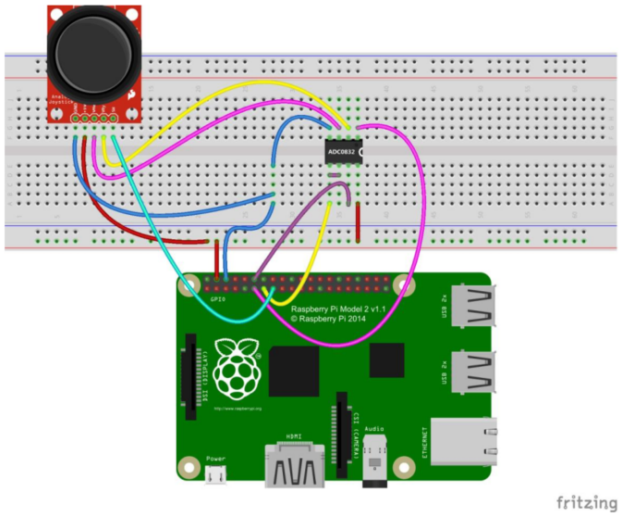


Figure 8.3: LED bar Graph circuit.

Chapter 9

DAC

Digital-to-analog conversion (DAC) enables a digital device to output an analog voltage level, which is specifi ed using a digital value; this is the opposite of ADC. In this section, DAC capabilities are added to the RPi using either the I2C or the SPI buses.

9.1 Application 1: RGB LED

RGB LEDs consist of three LEDs (Red, Green and Blue LEDs), which when combined are capable of producing any color. A fourth connection typically serves as a common anode or cathode. In this experiment we use a common anode RGB LED (where the longest pin is the common anode and is connected to the +3.3V pin of the Raspberry Pi), and the three remaining pins are connected to the GPIO pins of the Raspberry Pi through current limiting resistors. This allows us to control the color of RGB LED by employing a 3-channel PWM signal. Connect the circuit as shown in figure 9.1.

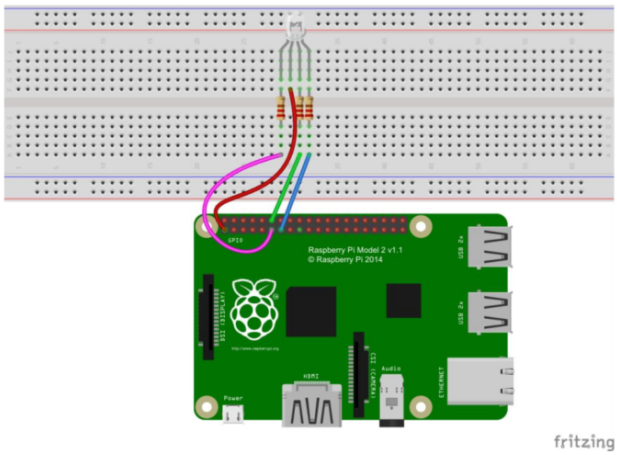


Figure 9.1: Connections to an RGB LED.

The script can be accessed by navigating to:
`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/08_rgbLed/rgbLed.c`.
Compile the program using: `gcc rgbLed.c -o rgbLed -lwiringPi -lpthread`
NOTE: The compiler option '-lpthread' is essential, because the implementation of softPwm is based on linux multi-threading.

9.2 Application 2: Passive Buzzer

Here, will learn how to program the Raspberry Pi to make a passive buzzer emanate sound with different frequency. Using square wave signals with different frequencies as input to the passive buzzer, it will generate sound with different modality. In this experiment, we continuously send different square wave signal to a passive buzzer to play a piece of music. Complete the circuit as shown in the figure below.

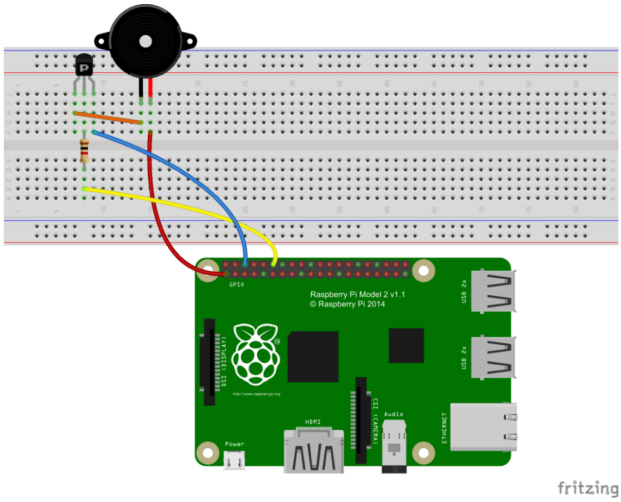


Figure 9.2: Passive Buzzer Circuit.

The code can be found at:
`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/03_passiveBuzzer/passiveBuzzer.c)`
Compile the script using: `gcc passiveBuzzer.c -o passiveBuzzer -lwiringPi -lpthread`

Resources:
https://www.sunfounder.com/learn/sensor-kit-v2-0-for-raspberry-pi-b-plus/lesson-10-buzzer-module-sensor-kit-v2-0-for-b-plus.html
http://www.farnell.com/datasheets/1653447.pdf?_ga=2.230578900.999400705.1571780342-2029348066.1571780342
http://media.nkcelectronics.com/datasheet/s8050.pdf

Optional:
git clone https://github.com/leon-anavi/rpi-examples.git
cd rpi-examples
cd buzzer/c/
gcc starwars.c -o starwars -lwiringPi -std=c99

Chapter 10

Stepper Motor

Unlike DC motors, which rotate continuously when a DC voltage is applied, stepper motors normally rotate in discrete fixed-angle steps. The motor steps each time a pulse is applied to its input, so the speed of rotation is proportional to the rate at which pulses are applied.

Stepper motors can be positioned very accurately, because they typically have a positioning error of less than 5% of a step (i.e., typically 0.1^o). The error does not accumulate over multiple steps, so stepper motors can be controlled in an open-loop form, without the need for feedback. Unlike servo motors, but like DC motors, the absolute position of the shaft is not known without the addition of devices like rotary encoders, which often include an absolute position reference that can be located by performing a single shaft rotation.

Stepper motors have toothed permanent magnets that are fixed to a rotating shaft, called the rotor. The rotor is surrounded by coils (grouped into phases) that are fixed to the stationary body of the motor (the stator). The coils are electromagnets that, when energized, attract the toothed shaft teeth in a clockwise or counterclockwise direction, depending on the order in which the coils are activated: **Full step:** Two phases always on (max torque). **Half step:** Double the step resolution. Alternates between two phases on and a single phase on (torque at about 3/4 max). **Microstep:** Uses sine and cosine waveforms for the phase currents to step the motor rather than the on/off currents and thus allows for higher step resolutions (though the torque is significantly reduced).

In this lesson, we will introduce a new electronic device stepper motor and learn how to control it with Raspberry Pi.

Stepper Motor: Stepper motors, due to their unique design, can be controlled to a high degree of accuracy without any feedback mechanisms. The shaft of a stepper, mounted with a series of magnets, is controlled by a series of electromagnetic coils that are charged positively and negatively in a specific sequence, precisely moving it forward or backward in small ”steps”. There are two types of steppers, Unipolars and Bipolars, and it is very important to know which type you are working with. In this experiment, we will use an Unipolar stepper.

ULN2003 Driver Module: The Raspberry Pis GPIO cannot directly drive a stepper motor due to the weak current. Therefore, a driver circuit is necessary to control the stepper motor. This experiment uses the ULN2003 driver module. There are four LEDs on the module. The white socket in the middle is for connecting a stepper motor. IN1, IN2, IN3, IN4 are used to connect with the Raspberry Pi.

The script to control the Stepper Motor in *full step mode* can be accessed at:
`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/24_stepperMotor.py`.

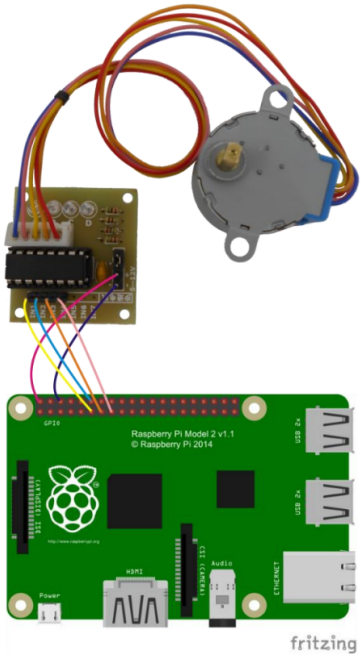


Figure 10.1: Stepper Motor Circuit.

Chapter 11

Servo Motor

In this section, we will discuss Servo Motors and learn how to control it with the Raspberry Pi.

The Servo motor is a type of geared motor that can only rotate 180 degrees. It is controlled by sending pulses signal from your microcontroller. These pulses tell the servo what position it should move to. The Servo motor consists of a shell, circuit board, non-core motor, gear and location detection modules. Its working principle is as follow: The Raspberry Pi sends a PWM signal to the servo motor, and then this signal is processed by an IC on circuit board to calculate the rotation direction to drive the motor, and then this driving power is transferred to the swing arm by a reduction gear. At the same time, the position detector returns the location signal to gauge whether the set location is reached or not.

The relationship between the rotation angle of the servo and pulse width as shown below:

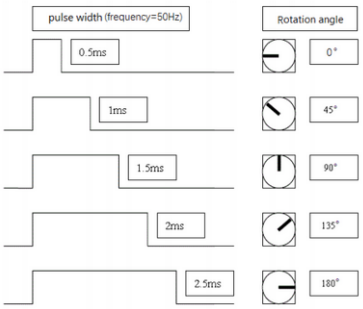


Figure 11.1: Servo Motor Principle.

The script can be accessed at:
`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/23_servo/servo.c`

You should see the servo motor rotate 180 degrees. And then rotate in opposite direction.

The following is the function of each pin:

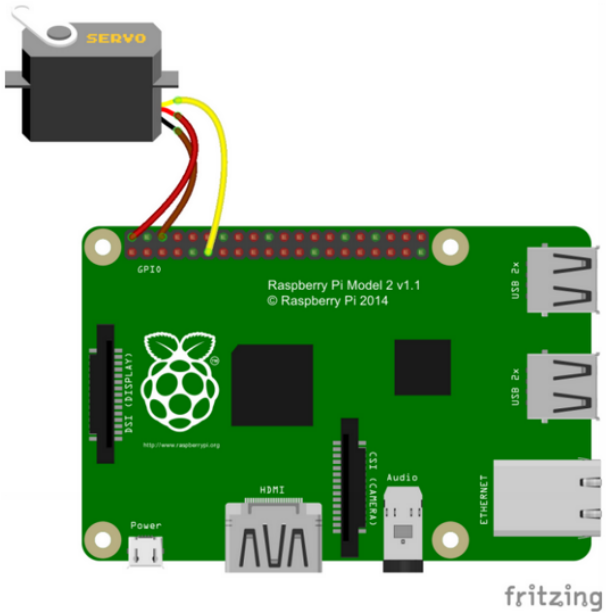


Figure 11.2: Servo Motor Circuit.

Chapter 12

Shift Register

12.1 Application 1: Dot-matrix display

In this lesson, we will program to control a 8*8 dot-matrix display to realize graphical and digital displays.

12.1.1 Principle

1. Dot-matrix display: A dot-matrix display is a display device used to display information on machines, clocks, railway departure indicators and many other devices requiring a simple display device of limited resolution. The display consists of a dot-matrix of lights or mechanical indicators arranged in a rectangular configuration (other shapes are also possible, although not common) such that by switching on or off selected lights, text or graphics can be displayed. A dot-matrix controller converts instructions from a processor into signals which turns on or off lights in the matrix so that the required display is produced. The internal structure and appearance of the dot-matrix display is as shown in below:

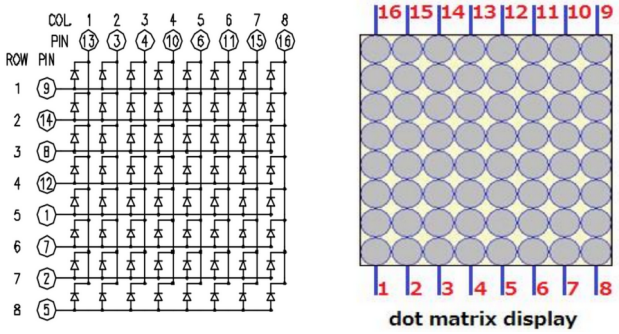


Figure 12.1: An 8*8 dot-matrix display consists of 64 LEDs, and each LED is placed at the intersection of the lines and columns. When the corresponding row is set as high level and the column is set as low level, then the LED will be lit.

A certain drive current is required for the dot-matrix display. In addition, more pins are needed for connecting dot-matrix display with controller. Thus, to save the Raspberry Pis GPIO, driver IC 74HC595 is used in this experiment.

2. 74HC595: The 74HC595 is an 8-stage serial shift register with a storage register and 3-state outputs. The shift register and storage register have separate clocks. Data is shifted on the positive-going transitions of the SH_CP input. The data in each register is transferred to the storage register on a positive-going transition of the ST_CP input. The shift register has a serial input (DS) and a serial standard output (Q7) for cascading. It is also provided with asynchronous reset (active LOW) for all 8 shift register stages. The storage register has 8 parallel 3-state bus driver outputs. Data in the storage register appears at the output whenever the output enable input (OE) is LOW. In this experiment, only 3 pins of Raspberry Pi are used for controlling a dot-matrix display due to the existence of 74HC595.

The following is the function of each pin:
DS: Serial data input

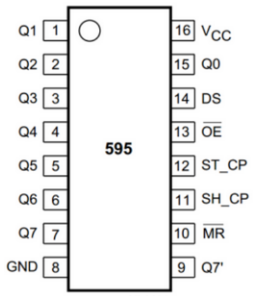


Figure 12.2: Shif Register IC.

Q0-Q7: 8-bit parallel data output
Q7 : Series data output pin, always connected to DS pin of the next 74HC595
OE: Output enable pin, effective at low level, connected to the ground directly
MR: Reset pin, effective at low level, directly connected to 5V high level in practical applications
SH_CP: Shift register clock input
ST_CP: storage register clock input

12.1.2 Procedure

1. Build the circuit (**Make sure that the circuit connection is correct and then power, otherwise it may cause the chips to burn.**)

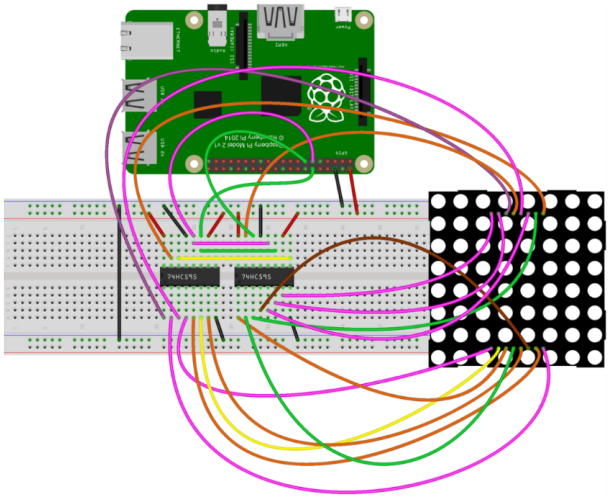


Figure 12.3: LED Dot Matrix Circuit.

2. Code path:
`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/16_ledMatrix/ledMatrix.c`

3. Compile the program using
`gcc ledMatrix.c -o ledMatrix -lwiringPi`

Now, you can see a rolling Adeept should be displayed on the dot-matrix display.