Lab Manual for IECE 553/453, ICSI 553 Cyber-Physical Systems Fall 2022

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 and install Raspberry Pi OS from the official Raspberry Pi website (`https://www.raspberrypi.com/software/`).
2. 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.
3. Insert the Micro SD card in the Raspberry Pi and power it. This will boot up in Raspberry Pi OS 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 6, Advanced Options, then choose A1 Expand Filesystem. Save the changes
4. Change the hostname. Use `sudo raspi-config`. Choose Option 1, System Options. Then choose S4 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: option 1 System Options → S5 Boot → 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 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. Download WiringPi and install it.
   - Download the repository - `git clone https://github.com/WiringPi/WiringPi.git`

   - Go to the directory of the downloaded repository - `cd WiringPi`
   - Build it - `./build`
   - Check if it is installed - `gpio -v`

6. Download codes from your sensor kit.
   (a) Create a directory - `mkdir Adeept`
   (b) Go to that directory - `cd Adeept`
   (c) Clone the code for C language -
   `git clone`
   `https://github.com/adeept/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi.git`
   (d) Clone the Python code -
   `git clone`
   `https://github.com/adeept/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi.git`

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

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.

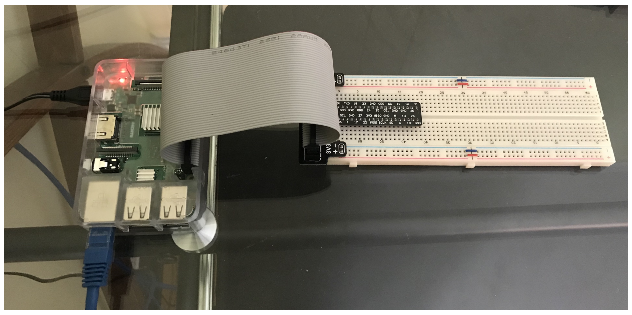
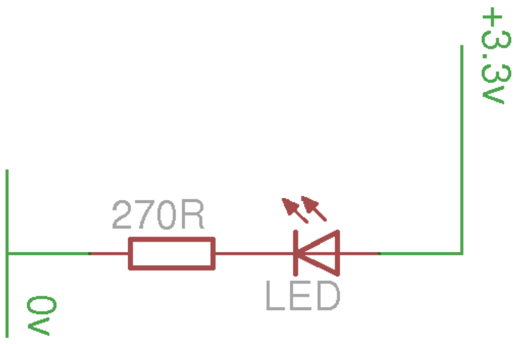
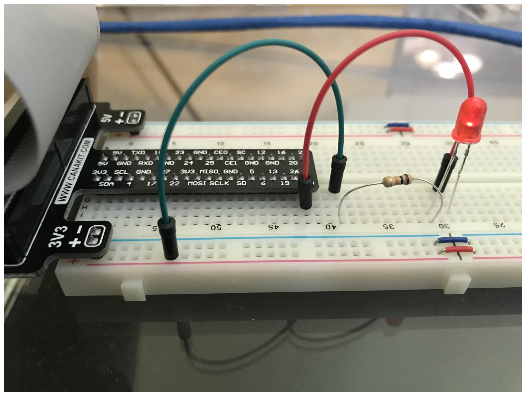


Figure 2.1: GPIO to Breadboard Interface Board and Ribbon Cable.



(a) Schematic Diagram.



(b) Circuit on breadboard.

Figure 2.2: LED Resistor Circuit.

The first circuit that we will work on is shown in Figure 2.2. 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!

Chapter 3

Basic Output Using Pseudo Filesystem

3.1 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 3.1.

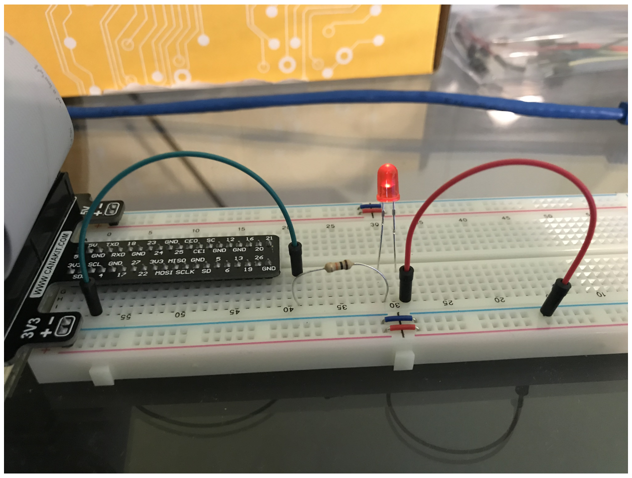


Figure 3.1: LED Resistor Circuit connected to GPIO Pin 4.

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 gpiochip504 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 gpiochip504 unexport
   ```

3.2 The First Circuit Using Bash, Python and C

Listing 3.1 [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 §3.1. Listings 3.2 and 3.3 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
  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"
```

[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.

```
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 3.1: 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 3.2: 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");
   }
```

```c
   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 3.3: LED code in C

Chapter 4

Digital Input

In the next experiment, we will use GPIO Pin as an input. The circuit is shown as in figure 4.1. A button switch, when pressed, will be detected by the program and a message will be printed accordingly.

4.1 Polling

Polling is a process where the program continuously polls the status of the input pin. Listings 4.1, 4.2 and 4.3 shows the code using `C`, and `RPi` and `gpiozero` libraries in Python respectively.

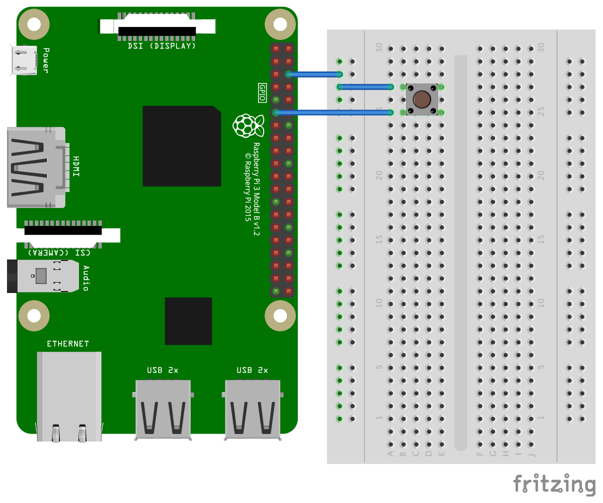


Figure 4.1: Button Switch connected to GPIO Pin.

```
#include <stdio.h>
#include <unistd.h>
#include <wiringPi.h>

#define PIN_BUTTON 18

int main (int argc, char **argv)
{
    wiringPiSetupGpio();

    pinMode(PIN_BUTTON, INPUT);
    pullUpDnControl(PIN_BUTTON, PUD_UP);
    printf("Button pin has been setup.\n");

    while (1)
    {
        if (digitalRead(PIN_BUTTON) == 0) {
            printf("Button pressed\n");
        }
        delay(100);
    }
}
```

Listing 4.1: Button press detection polling code in C

```
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 4.2: Button Press detection polling 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 4.3: Button Press detection polling code using GPIOZero Library in Python

4.2 Interrupt

An interrupt is a hardware signal from a device (in this case GPIO pin) to a CPU. When the interrupt occurs, CPU calls the interrupt service routine for that device. `wiringPiISR()` is a function that registers a an interrupt service routine (`myISR()`) to receive interrupts on the specified pin. When interrupt occurs, the processor invokes the `myISR()` function. Listings 4.4 shows the code using `C` language.

```
#include <wiringPi.h>
#include <stdio.h>

#define PIN_BUTTON 18

void myISR(void)
{
    printf("Button Pressed\n");
    delay(50);
}
```

```
int main(void)
{
    if(wiringPiSetupGpio() == -1){ //when initialize wiring failed,print message to
        screen
        printf("setup wiringPi failed !\n");
        return -1;
    }

    pinMode(PIN_BUTTON, INPUT);
    pullUpDnControl(PIN_BUTTON, PUD_UP);

    if(wiringPiISR(PIN_BUTTON, INT_EDGE_FALLING, myISR) < 0){
        printf("ISR setup error!\n");
        return -1;
    }

    while(1){
        // Infinite loop
    }

    return 0;
}
```

Listing 4.4: Button press detection interrupt code in C

4.3 Handling Jitter

The button press creates jitter in the hardware signal, which signals Raspberry Pi multiple times. To avoid this, we can handle in software or in hardware.

4.3.1 Software

We detect whether the level of button interface is low or high. Low value indicates that the button is pressed. When the detected level is low, $\approx$ 10 ms delay is needed to settle the signal. So, we add that extra delay in software to detect the level again. If the signal is low after the delay, we can confirm that the button is pressed once. Listings 4.5 shows the code using `C` language to handle the jitter.

```
#include <stdio.h>
#include <unistd.h>
#include <wiringPi.h>

#define PIN_BUTTON 18

int main (int argc, char **argv)
{
    wiringPiSetupGpio();

    pinMode(PIN_BUTTON, INPUT);
    pullUpDnControl(PIN_BUTTON, PUD_UP);

    printf("Button pin has been setup.\n");

    while (1)
    {
        if (digitalRead(PIN_BUTTON) == 0) {
            delay(10);
            if (digitalRead(PIN_BUTTON) == 0)
                printf("Button pressed\n");
        }
        delay(100);
        }
}
```
Listing 4.5: Button press detection code in C to handle jitter

4.3.2 Hardware

We can also create a RC circuit to by using a capacitance to clean up the jitter of buttons, as shown in Figure 4.2.

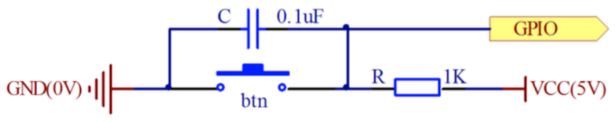


Figure 4.2: Button Switch connected to GPIO Pin with a capacitor to hold the charge.

Chapter 5

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).

$$PWM\ frequency = \frac{19.2MHz}{(divisor \times range)} \tag{5.1}$$

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.

5.1 Circuit

Create a circuit similar to the one shown in Figure 3.1.

5.2 Hard PWM

PWM can be implemented using WiringPi Library.

There are following functions that are available.

1. `pwmSetMode (int mode) ;`
   The PWM generator can run in 2 modes – “balanced” and “mark:space”. The mark:space mode is traditional, however the default mode in the Pi is “balanced”. You can switch modes by supplying the parameter: `PWM_MODE_BAL` or `PWM_MODE_MS`.

2. `pwmSetRange (unsigned int range) ;`
   This sets the range register in the PWM generator. The default is 1024.

3. `pwmSetClock (int divisor) ;`
   This sets the divisor for the PWM clock.

Listing 5.1 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` Implement the circuit corresponding to the 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`.

```
#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 5.1: PWM code in C++

5.3 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. WiringPi includes a software-driven PWM handler capable of outputting a PWM signal on any of the Raspberry Pi's GPIO pins. To maintain a low CPU usage, the minimum pulse width is $100\mu s$. That combined with the default suggested range of 100 gives a PWM frequency of $100Hz$. You can lower the range to get a higher frequency, at the expense of resolution, or increase to get more resolution, but that will lower the frequency.

The following two functions are available:

1. `int softPwmCreate (int pin, int initialValue, int pwmRange);`
   This creates a software controlled PWM pin. You can use any GPIO pin and the pin numbering will be that of the wiringPiSetup() function you used. Use 100 for the pwmRange, then the value can be anything from 0 (off) to 100 (fully on) for the given pin. The return value is 0 for success. Anything else and you should check the global errno variable to see what went wrong.

2. `void softPwmWrite (int pin, int value) ;`
   This updates the PWM value on the given pin. The value is checked to be in-range and pins that haven't previously been initialised via softPwmCreate will be silently ignored.

An example of Software PWM is shown in Listing 5.2. Implement the circuit corresponding to the above script to turn on two LEDs using a Sofware PWM output.

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

#define GPIO1 4
#define GPIO2 17

int main(int argc, char *argv[])
{
   if (wiringPiSetupGpio() < 0) return 1;

    pinMode(GPIO1, OUTPUT);
    digitalWrite(GPIO1, LOW);
    //int softPwmCreate (int pin, int initialValue, int pwmRange) ;
    softPwmCreate(GPIO1, 0, 200);
    // void softPwmWrite (int pin, int value) ;
    softPwmWrite(GPIO1, 15);

    pinMode(GPIO2, OUTPUT);
    digitalWrite(GPIO2, LOW);
    softPwmCreate(GPIO2, 0, 500);
    softPwmWrite(GPIO2, 15);
    sleep(10);
}
```

Listing 5.2: PWM code in C

5.4 RGB LED

In this section, we will use PWM to change the color of an RGB LED. There are three LEDs combined in the RGB LED: red, green and blue. Since the LEDs are very close to each other, our eyes see the result of the combination of colors, rather than the three colors individually. By configuring the intensity of each LED, we can produce different colors. We can use a PWM signal to modify the intensity of each LED.

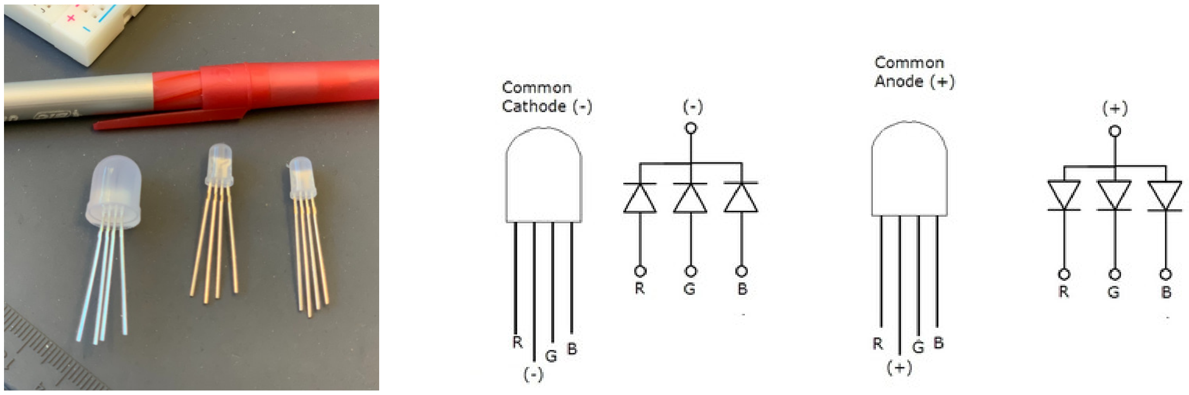


Figure 5.1: Common Cathode and Common Anode RGB LED.

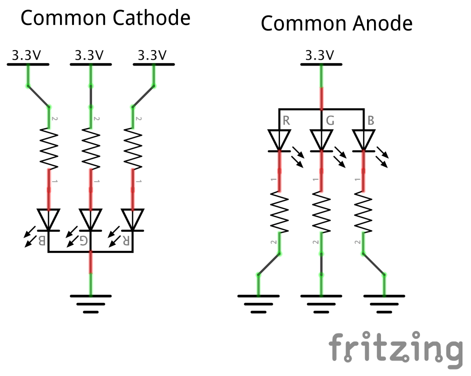


Figure 5.2: Check circuit for Common Cathode or Common Anode.

There are two kinds of RGB LEDs: common cathode LED and common anode LED, both look identical as in Figure 5.1. In a common cathode RGB LED, all three LEDs share a negative connection. In a common anode RGB LED, the three LEDs share a positive connection. The four leads can be identified as shown in the figure.

To check whether you have a common anode or a common cathode LED, you may create only one of the paths (either red, green or blue) of the circuit, using 3.3V, as shown in Figure 5.2. If you have a Common Anode LED, and you created the circuit as in the first figure, the LED color path that you created, will turn on. If you have a Common Cathode LED and you created the first circuit, it will not turn on. Based on your findings, you will have to create your final circuit. Instead of using 3.3V, you should use PWM signal to change the intensity of the LED.

Listing 5.3 provides a code example for slowly fading an LED on and off using PWM. Implement the circuit corresponding to the script to fade in and out and LED.

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

#define RANGE       100
#define RED_PIN     17
#define GREEN_PIN   27
#define BLUE_PIN    22


int main()
{
    int LedPin[3] = {RED_PIN, GREEN_PIN, BLUE_PIN};

    int i=0;

    // Check how to Setup - http://wiringpi.com/reference/setup/
    wiringPiSetupGpio();

    for(i=0;i<3;i++) // Setup three PINS for Soft PWM
    {
        //int softPwmCreate (int pin, int initialValue, int pwmRange) ;
        softPwmCreate(LedPin[i], 0, RANGE);
    }
    // void softPwmWrite (int pin, int value) ;
    // Note: If you have a Common Anode RGB LED,
    // 100 indicates maximum intensity
    // If you have a Common Cathode RGB LED,
    // 0 indicates maximum intensity.
    softPwmWrite(LedPin[0], 0);   // Value for Red Pin
    softPwmWrite(LedPin[1], 100); // Value for Green Pin
    softPwmWrite(LedPin[2], 0);   // Value for Blue Pin
    delay(10);

    return 0;
}
```

Listing 5.3: PWM code for RGB LED in C

Chapter 6

More Input and Output

In this lab, we will work on another input and output device.

6.1 7 Segment Display

The seven-segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays. It is an 8-shaped LED display device composed of eight LEDs (including a decimal point). These segments are termed a, b, c, d, e, f, g, dp respectively as shown in 6.1. It can be either common anode or common cathode segment display through internal connections, as you have experienced with RGB LED. When using a common anode LED, the common anode should to be connected to the power supply (VCC); when using a common cathode LED, the common cathode should be connected to the ground (GND). Each segment of a segment display is composed of LED, so a resistor is needed for protecting the LED. A 7-segment display has seven segments for displaying a figure and a segment for displaying a decimal point. If you want to display a number '1', you should only light the segment b and c.

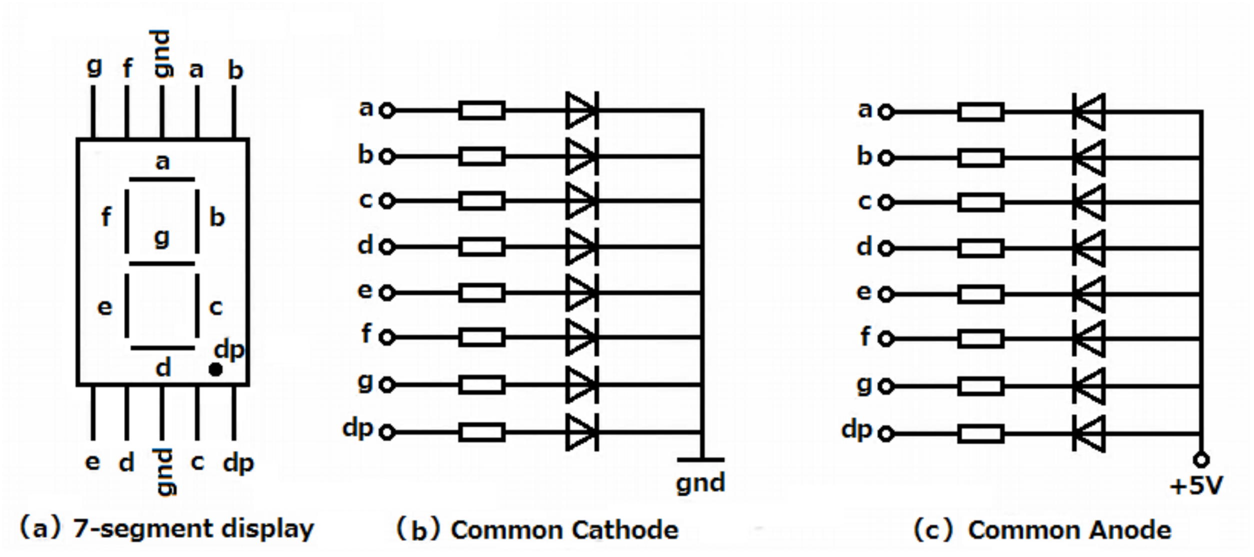


Figure 6.1: 7 segment display.

Figure 6.2 shows the circuit diagram for a common cathode 7 segment display. We will use the code at `Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/09_segment/segment.c` to determine the wiring. The code is shown in Listing 6.1.

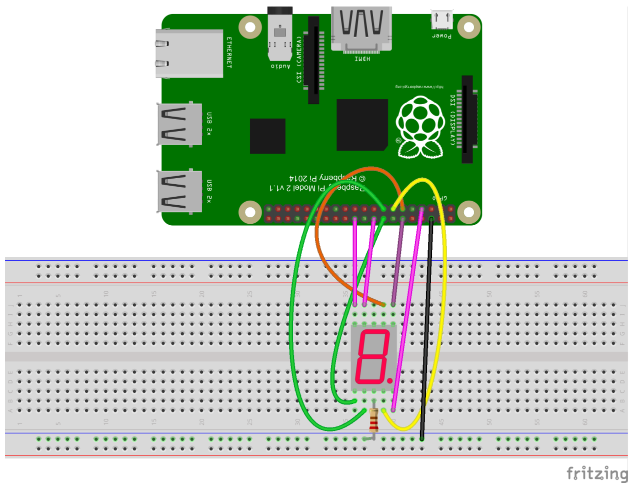


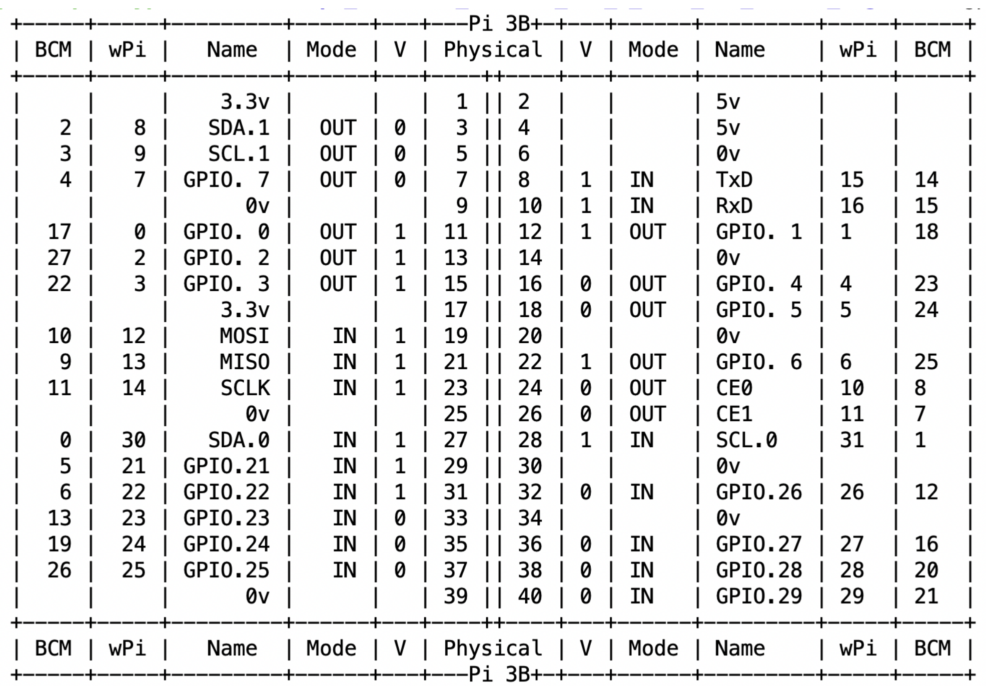
Figure 6.2: Circuit for 7 segment display with common cathode.

```
/*
* File name   : segment.c
* Description : display 0~9, A~F on 7-segment display
* Website     : www.adeept.com
* E-mail      : support@adeept.com
* Author      : Jason
* Date        : 2015/05/26
*/
#include <stdio.h>
#include <wiringPi.h>

typedef unsigned char uchar;

const uchar SegCode[17] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c
    ,0x39,0x5e,0x79,0x71,0x80};

int main(void)
{
        int i;

        if(wiringPiSetup() < 0){ // setup wiringPi
                printf("wiringPi setup failed !\n");
                return -1;
        }

        for(i = 0; i < 8; i++){  // set pin mode as output(GPIO0~GPIO7)
                pinMode(i, OUTPUT);
        }

        while(1){
                for(i = 0; i < sizeof(SegCode)/sizeof(uchar); i++){ // display 0~9,A~F
                        digitalWriteByte(SegCode[i]);
                        delay(500);
                }
                digitalWriteByte(0x00);   //segment off
                delay(1000);
        }
        return 0;
}
```

Listing 6.1: C code for 7 Segment Display

You may notice that the leads are addressed using pins 0-7. Also, notice that `wiringPiSetup()` is used for setting up, which uses `wiringPi` pin numbers. Make sure that you are connecting to the correct pin number by checking the mapping using the command in th terminal: `$gpio readall`.

The output of `$gpio readall` is shown in Figure 6.3.



```
 +-----+-----+---------+------+---+---Pi 3B+-+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 |  OUT | 0 |  3 || 4  |   |      | 5v      |     |     |
 |   3 |   9 |   SCL.1 |  OUT | 0 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |  OUT | 0 |  7 || 8  | 1 | IN   | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | IN   | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |  OUT | 1 | 11 || 12 | 1 | OUT  | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |  OUT | 1 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |  OUT | 1 | 15 || 16 | 0 | OUT  | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | OUT  | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI |   IN | 1 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO |   IN | 1 | 21 || 22 | 1 | OUT  | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK |   IN | 1 | 23 || 24 | 0 | OUT  | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 0 | OUT  | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 3B+-+---+------+---------+-----+-----+
```

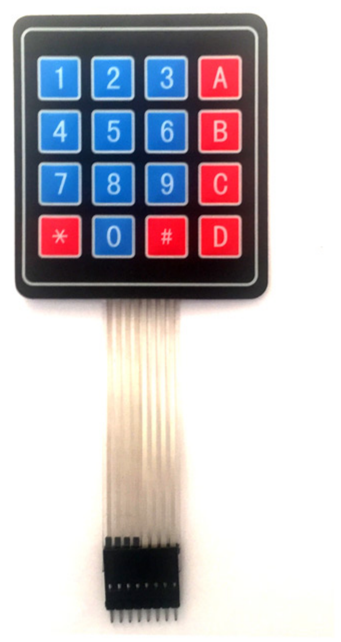
Figure 6.3: GPIO Pin Map.

6.2 4x4 Matrix Keyboard/Keypad

A 4x4 keypad module and its pin connections are shown in figure 6.4. Internally, it has 16 buttons arranged in matrix formation, as shown in Figure 6.5.

Listing 6.2 shows the code available at `Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/13_matrixKeyboard/matrixKeyboard.c`. Pay extra attention to the row and column pin numbers to make the connection.

```
#include <wiringPi.h>
#include <stdio.h>

const int ROW[]    = {0, 1, 2, 3};
const int COLUMN[] = {4, 5, 6, 7};

int getKey(void)
{
        int i;
        int tmpRead;
        int rowVal = -1;
        int colVal = -1;
        char keyVal;
```



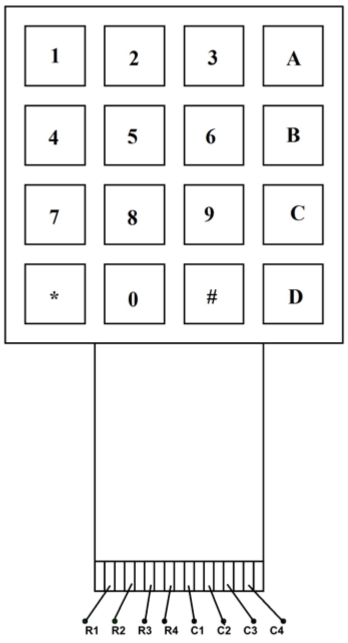


Figure 6.4: 4x4 Matrix Keyboard.

```
        for(i = 0; i < 4; i++){
                pinMode(COLUMN[i], OUTPUT);
                digitalWrite(COLUMN[i], LOW);
        }

        for(i = 0; i < 4; i++){
                pinMode(ROW[i], INPUT);
                pullUpDnControl(ROW[i], PUD_UP);
        }

        for(i = 0; i < 4; i++){
                tmpRead = digitalRead(ROW[i]);
                if(tmpRead == 0){
                        rowVal = i;
                }
        }

        if(rowVal < 0 || rowVal > 3){
                return -1;
        }

        for(i = 0; i < 4; i++){
                pinMode(COLUMN[i], INPUT);
                pullUpDnControl(COLUMN[i], PUD_UP);
        }

        pinMode(ROW[rowVal], OUTPUT);
        digitalWrite(ROW[rowVal], LOW);

        for(i = 0; i < 4; i++){
                tmpRead = digitalRead(COLUMN[i]);
                if(tmpRead == 0){
```

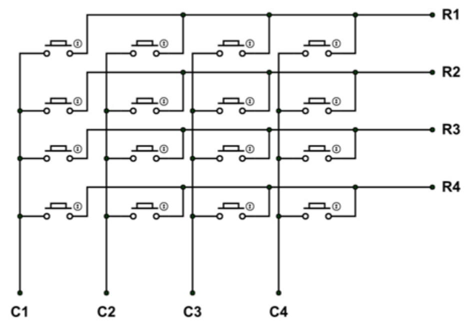


Figure 6.5: 4x4 Matrix Keyboard internal connections.

```
                                colVal = i;
                        }
                }

                if(colVal < 0 || colVal > 3){
                        return -1;
                }

                //printf("%d, %d\n", rowVal, colVal);
                switch(rowVal){
                        case 0:
                                switch(colVal){
                                        case 0: keyVal = 0; break;
                                        case 1: keyVal = 1; break;
                                        case 2: keyVal = 2; break;
                                        case 3: keyVal = 3; break;
                                        default:
                                                break;
                                }
                                break;
                        case 1:
                                switch(colVal){
                                        case 0: keyVal = 4; break;
                                        case 1: keyVal = 5; break;
                                        case 2: keyVal = 6; break;
                                        case 3: keyVal = 7; break;
                                        default:
                                                break;
                                }
                                break;
                        case 2:
                                switch(colVal){
                                        case 0: keyVal = 8; break;
                                        case 1: keyVal = 9; break;
                                        case 2: keyVal = 10; break;
                                        case 3: keyVal = 11; break;
                                        default:
                                                break;
                                }
                                break;
                        case 3:
                                switch(colVal){
                                        case 0: keyVal = 12; break;
                                        case 1: keyVal = 13; break;
                                        case 2: keyVal = 14; break;
                                        case 3: keyVal = 15; break;
                                        default:
                                                break;

                                }
                                break;
                        default:
                                break;
        }

        return keyVal;
}

int main(void)
{
        int i;
        int key = -1;

        if(wiringPiSetup() == -1){
                printf("setup wiringPi failed !\n");
                return -1;
        }

        while(1){
                key = getKey();
                if(key != -1){
                        switch(key){
                                case 0: printf("1\n"); break;
                                case 1: printf("2\n"); break;
                                case 2: printf("3\n"); break;
                                case 3: printf("A\n"); break;
                                case 4: printf("4\n"); break;
                                case 5: printf("5\n"); break;
                                case 6: printf("6\n"); break;
                                case 7: printf("B\n"); break;
                                case 8: printf("7\n"); break;
                                case 9: printf("8\n"); break;
                                case 10: printf("9\n"); break;
                                case 11: printf("C\n"); break;
                                case 12: printf("*\n"); break;
                                case 13: printf("0\n"); break;
                                case 14: printf("#\n"); break;
                                case 15: printf("D\n"); break;
                                default:
                                        break;
                        }
                }
                delay(200);
        }

        return 0;
}
```

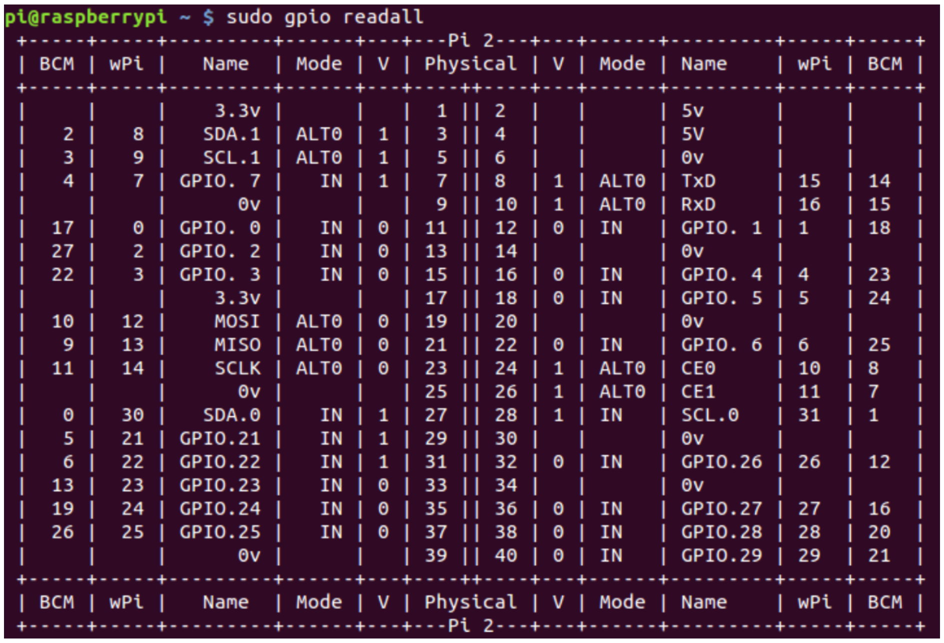
Listing 6.2: C code for Matrix Keyboard

Chapter 7

Analog Input

In this lab we will learn how to use the Raspberry Pi for Analog input.

This is to recall that the default configurations of the RPi GPIO pins can be viewed by querying the following in the terminal:
`sudo gpio readall`
This will display the default configurations as shown in Figure 7.1. Make sure to use the correct pin as initialized in the initial setup of wiringPi (http://wiringpi.com/reference/setup/).



```
pi@raspberrypi ~ $ sudo gpio readall
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | ALT0 | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | ALT0 | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
```

Figure 7.1: Default GPIO configurations.

7.1 Temperature & Humidity Sensor

Sensor: In this section we will learn how to use DHT-11 to read 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 Figure 7.2.

Communicating to One-Wire Sensors: DHT11 can digitally communicate with the RPi using a single GPIO. The

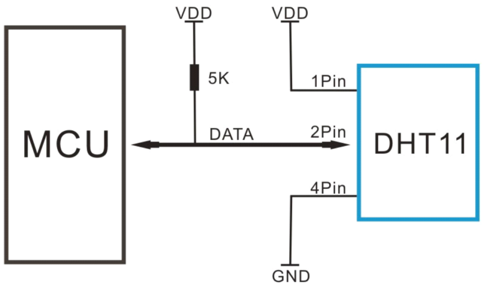


Figure 7.2: Temperature & Humidity Sensor circuit.

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 sensor's response. Communication takes place when the RPi pulls the GPIO low for $18ms$ and then releases the line high for a further 20–$40\mu s$. The GPIO switches to read mode and ignores the $80\mu s$ low level and the $80\mu s$ high pulse that follows. The sensor then returns 5 bytes of data (i.e., 40-bits) in most-significant bit (MSB) first form, where the first 2 bytes represent the humidity value, the following 2 bytes represent the temperature, and the final byte is a parity-sum, which can be used to verify that the received data is valid (it is the 8-bit bounded sum of the preceding 4 bytes). The bits are sent by varying the duration of high pulses. When DHT is sending data to MCU, every bit of data begins with the $50\mu s$ low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1". A high for $26\mu s$–$28\mu s$ signifies a binary 0, and a high for $70\mu s$ signifies a binary 1. Figure 7.3 illustrates an actual oscilloscope data capture and worked calculations to explain the process for the DHT11.

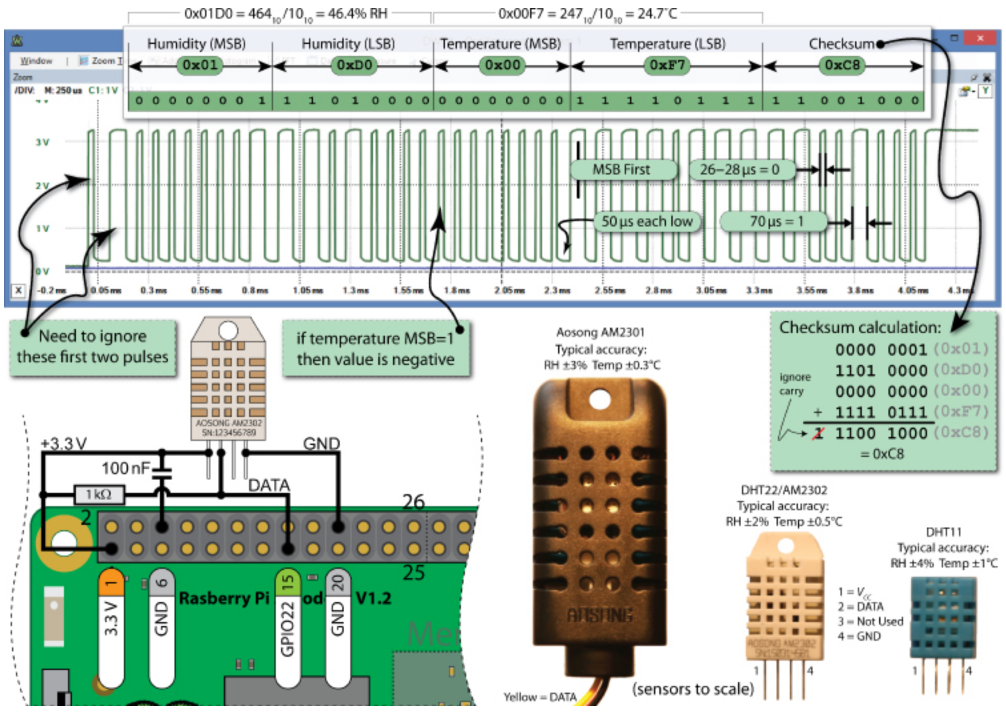


Figure 7.3: Data captured from DHT11.

The script to read the temperature and humidity is available at `/exploringrpi/chp06/dht/dht.cpp`. The code is modified for correct implementation.

```
#include<iostream>
#include<unistd.h>
#include<wiringPi.h>
#include<iomanip>

using namespace std;
#define USING_DHT11 true // The DHT11 uses only 8 bits
#define DHT_GPIO 22 // Using GPIO 22 for this example
#define LH_THRESHOLD 26 // Low= 1 4 , High= 3 8  - pick avg.

int main(){
    double humid = 0, temp = 0;
    cout << "Starting the one-wire sensor program" << endl;
    wiringPiSetupGpio();
    piHiPri(99);
TRYAGAIN: // If checksum fails (come back here)
    unsigned char data[5] = {0,0,0,0,0};
    pinMode(DHT_GPIO, OUTPUT); // gpio starts as output
    digitalWrite(DHT_GPIO, LOW); // pull the line low
    usleep(18000); // wait for 18ms
    digitalWrite(DHT_GPIO, HIGH); // set the line high
    pinMode(DHT_GPIO, INPUT); // now gpio is an input

    // need to ignore the first and second high after going low
    do { delayMicroseconds(1); } while(digitalRead(DHT_GPIO)==HIGH);
    do { delayMicroseconds(1); } while(digitalRead(DHT_GPIO)==LOW);
    do { delayMicroseconds(1); } while(digitalRead(DHT_GPIO)==HIGH);
    // Remember the highs, ignore the lows -- a good philosophy!
    for(int d=0; d<5; d++) { // for each data byte
        // read 8 bits
        for(int i=0; i<8; i++) { // for each bit of data
            do { delayMicroseconds(1); } while(digitalRead(DHT_GPIO)==LOW);
            int width = 0; // measure width of each high
            do {
                width++;
                delayMicroseconds(1);
                if(width>1000) break; // missed a pulse -- data invalid!
            } while(digitalRead(DHT_GPIO)==HIGH); // time it!
            // shift in the data, msb first if width > the threshold
            data[d] = data[d] | ((width > LH_THRESHOLD) << (7-i));
        }
    }
    if (USING_DHT11){
        humid = data[0]; // one byte - no fraction_tempal part
        temp = data[2] ;

        if(data[3] < 10.0){
            double fraction_temp = double(data[3])/10.0;
            temp = temp+ fraction_temp;
        }
        else if(data[3] < 100.0){
            double fraction_temp = double(data[3])/100.0;
            temp = temp+ fraction_temp;
        }
        else {
            double fraction_temp = double(data[3])/1000.0;
            temp = temp+ fraction_temp;
        }
        if(data[1] < 10.0){
            double fraction_humid = double(data[1])/10.0;
            humid = humid+ fraction_humid;
        }
        else if(data[1] < 100.0){
            double fraction_humid = double(data[1])/100.0;
            humid = humid+ fraction_humid;
        }
        else {
            double fraction_humid = double(data[1])/1000.0;
            humid = humid+ fraction_humid;
        }
    }
    else { // for DHT22 (AM2302/AM2301)
        humid = (data[0]<<8 | data[1]); // shift MSBs 8 bits left and OR LSBs
        temp = (data[2]<<8 | data[3]); // same again for temperature
    }

    unsigned char chk = 0; // the checksum will overflow automatically
    for(int i=0; i<4; i++){ chk+= data[i]; }
    if(chk==data[4]){
        cout << "The checksum is good" << endl;
        cout << "The temperature is " << (float)temp << "C" << endl;
        cout << "The humidity is " << (float)humid << "%" << endl;
    }
    else {
        cout << "Checksum bad - data error - trying again!" << endl;
        usleep(2000000); // have to delay for 1-2 seconds between readings
        goto TRYAGAIN; // a GOTO!!! call yourself a C/C++ programmer!
    }
    return 0;
}
```

Listing 7.1: DHT code in C++

Chapter 8

Music Creation with Passive Buzzer

In this chapter, we will use a passive buzzer to create any tone (frequency) from software. The circuit of the buzzer is shown in figure 8.1. The base pin (b pin) of the NPN (8050) transistor is connected to the GPIO pin of Raspberry Pi, the collector pin (c pin) to the cathode pin of the buzzer, and the emitter pin (e pin) to GND. The anode of the buzzer is connected to 5V power supply. When GPIO pin inputs high voltage, the transistor will be switched on, and the collector will output low level. When there is a level difference between the two pins of the buzzer, the buzzer rings. When the GPIO inputs low power level, the transistor is cut off, and the collector is at high level, and both ends of the buzzer are at high level making the buzzer silent.

We will use a PWM to change the tone of the buzzer. When a square wave (PWM) modulates the movement of something, like a buzzer, it generates a pressure wave that the human ear interprets as sound. By changing the frequency, or how often the square wave changes from high to low, we can change the frequency of the tone that we hear.

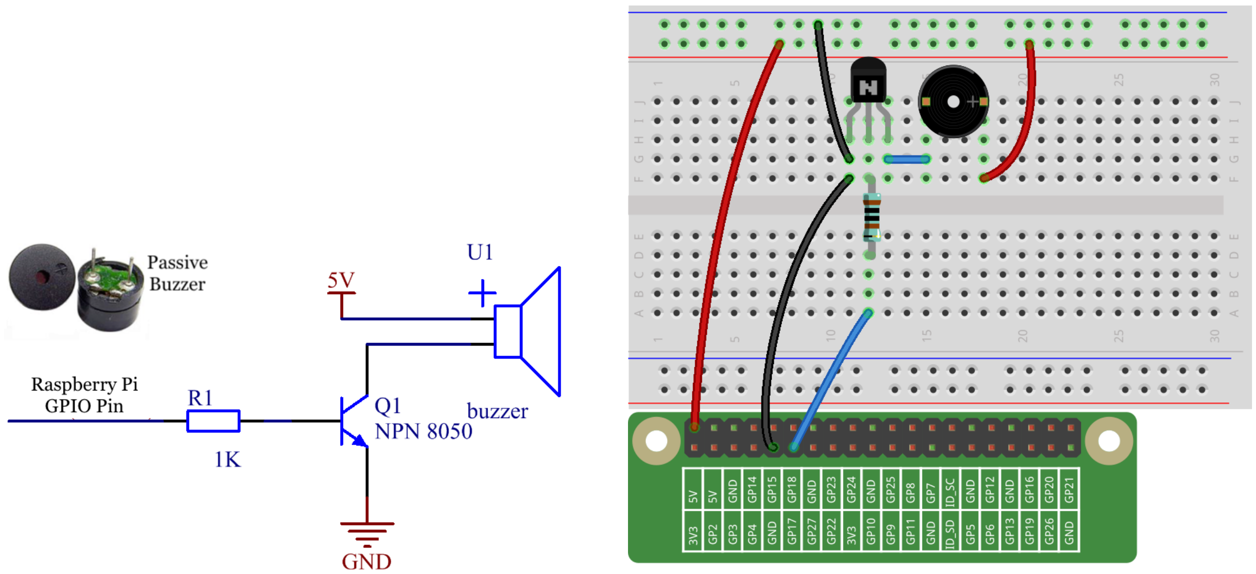

Passive Buzzer
U1
5V
Raspberry Pi GPIO Pin
R1
1K
Q1
NPN 8050
buzzer
GND
5V 5V GND GP14 GP15 GP18 GND GP23 GP24 GND GP25 GP8 GP7 ID_SC GND GP12 GND GP16 GP20 GP21
3V3 GP2 GP3 GP4 GND GP17 GP27 GP22 3V3 GP10 GP9 GP11 GND ID_SD GP5 GP6 GP13 GP19 GP26 GND

Figure 8.1: Buzzer Connection.

WiringPi includes a software-driven sound handler capable of outputting a simple tone/square wave signal on any of the Raspberry Pis GPIO pins. The details are available at http://wiringpi.com/reference/software-tone-library/.

The following two functions are available in wiringPi library:

1. `int softToneCreate (int pin);`
   This creates a software controlled tone pin. You can use any GPIO pin and the pin numbering will be that of the `wiringPiSetup()` function you used. The return value is `0` for success. Anything else and you should check the global errno variable to see what went wrong.

2. `void softToneWrite (int pin, int freq);`
   This updates the tone frequency value on the given pin. The tone will be played until you set the frequency to `0`.

The C program to generate the Tone is available at
`WiringPi/examples/softTone.c`.

```c
/*
 * softTone.c:
 *      Test of the softTone module in wiringPi
 *      Plays a scale out on pin 3 - connect pizeo disc to pin 3 & 0v
 *
 * Copyright (c) 2012-2013 Gordon Henderson.
 ***********************************************************************
 * This file is part of wiringPi:
 *      https://github.com/WiringPi/WiringPi
 *
 *    wiringPi is free software: you can redistribute it and/or modify
 *    it under the terms of the GNU Lesser General Public License as published by
 *    the Free Software Foundation, either version 3 of the License, or
 *    (at your option) any later version.
 *
 *    wiringPi is distributed in the hope that it will be useful,
 *    but WITHOUT ANY WARRANTY; without even the implied warranty of
 *    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *    GNU Lesser General Public License for more details.
 *
 *    You should have received a copy of the GNU Lesser General Public License
 *    along with wiringPi.  If not, see <http://www.gnu.org/licenses/>.
 ***********************************************************************
 */

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

#include <wiringPi.h>
#include <softTone.h>

#define PIN     3

int scale [8] = { 262, 294, 330, 349, 392, 440, 494, 525 } ;

int main ()
{
  int i ;

  wiringPiSetup () ;

  softToneCreate (PIN) ;

  for (;;)
  {
    for (i = 0 ; i < 8 ; ++i)
    {
      printf ("%3d\n", i) ;
      softToneWrite (PIN, scale [i]) ;
      delay (500) ;
    }
  }
}
```

Listing 8.1: Use of PWM to generate a Tone in a Passive Buzzer

Chapter 9

I2C Communication

In this chapter, we will study I2C. I2C on the RPi is implemented using the Broadcom Serial Controller (BSC), which supports 7-bit/10-bit addressing and bus frequencies of up to 400 kHz.

We will use ADXL345 Accelerometer in this example. The level of acceleration supported by a sensors output signal specifications is specified in $\pm g$. This is the greatest amount of acceleration the part can measure and accurately represent as an output. For example, the output of a $\pm 2g$ accelerometer is linear with acceleration up to $\pm 2g$. If it is accelerated at $4g$, the output may saturate.

1. We begin by enabling the I2C interface on the RPi by typing the following in terminal:
   ```
   $sudo raspi-config
   ```
   Choose Interface Options $\rightarrow$ I2C $\rightarrow$ Yes.

2. Check whether I2C is enabled. `/dev` is the location of device files. If I2C was enabled correctly, it will show up in `/dev/`. Type the following in terminal.
   ```
   $sudo ls /dev/i2c*
   ```
   Your output will be similar to:
   ```
   /dev/i2c-1 /dev/i2c-2
   ```
   Then check if the kernel module is loaded by issuing `lsmod` command, which outputs information for each loaded kernel module on a new line:
   ```
   $lsmod | grep i2c
   ```
   Your output will be similar to:
   ```
   i2c_bcm2835 16384 0
   i2c_dev 16384 0
   ```
   If those two modules are not loaded, we can use `modprobe` to load the modules in Linux Kernel.
   ```
   $modprobe i2c_dev
   $modprobe i2c_bcm2835
   ```

3. In the next step, we need to create the circuit as below:
   (a) Connect SCL on the RPi to SCL on the ADXL345
   (b) Connect SDA on the RPi to SDA in the ADXL345
   (c) Connect GND on the RPi to GND on the ADXL345
   (d) Connect 3.3V on the RPi to VIN on the ADXL345

   Refer Figure 9.1 for visual guidance.

4. Note that Raspberry Pi has in-built pull-up registers on board for SDA and SCL lines as shown in Figure 9.2.

5. The I^{2}C bus can be probed to detect connected devices by using the following command:
   ```
   sudo i2cdetect -y -r 1
   ```
   The output shows the I2C addresses of the attached devices. For example, when ADXL345 (`0x53`), MPU-6050 (`0x68`) and PCF8591T (`0x48`) are attached to the I2C bus, we get the following output as in Figure 9.3.

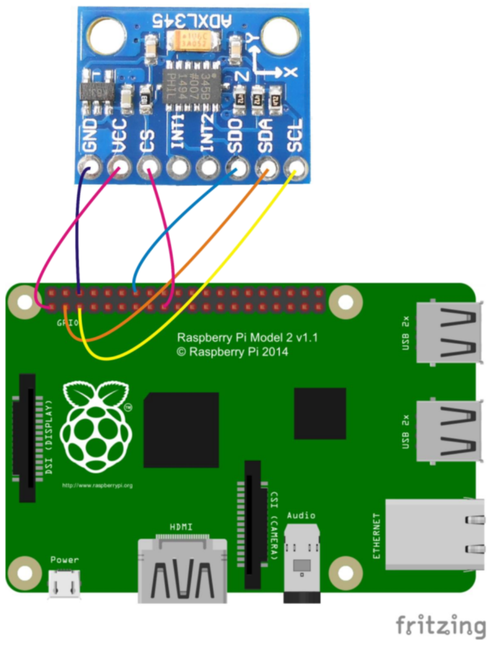


Figure 9.1: I2C Circuit Diagram.

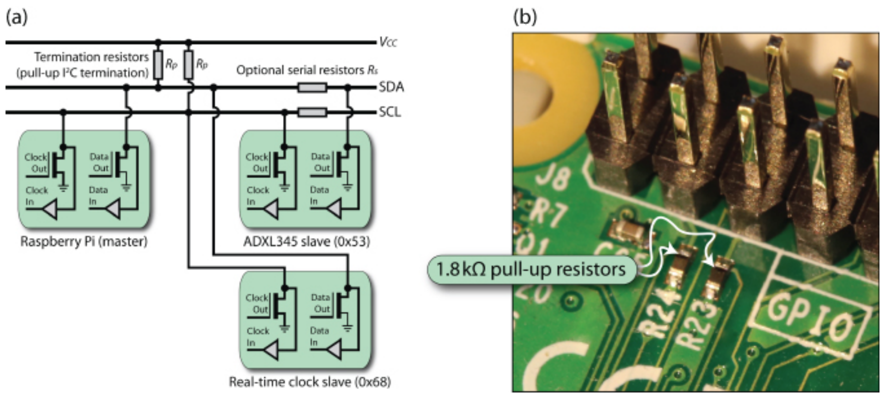


Figure 9.2: The I2C bus configuration, and (b) the built-in pull-up resistors on the I2C1 bus.

6. Refer to the manual (ADXL345) for understanding the functionality and registers.
7. We will use the code that came with your Sensor kit. For ADXL345, the code resides in folder `Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/25_ADXL345/`, with extra comments. Listing 9.1 shows the code.
8. The code uses `wiringPi`, for which you need to compile with `-lwiringPi`.
9. The code uses I2C wiringPi functions, the description of which are available at http://wiringpi.com/reference/i2c-library/
10. WiringPi implementation for I2C is available at https://github.com/WiringPi/WiringPi/blob/master/wiringPi/wiringPiI2C.c
11. Note that calibration should be performed for correct values, which can be set in the offset registers. https://learn.adafruit.com/adxl345-digital-accelerometer?view=all#programming

```
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- 53 -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- 68 -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Figure 9.3: I2C Detect Output.

```
#include <stdio.h>
#include <unistd.h>
#include <wiringPiI2C.h>

#define X_REG 0x32
#define Y_REG 0x34
#define Z_REG 0x36
#define DEV_ADDRESS 0x53

short int axis_sample_average(int axis, int fd);
short int axis_sample(int axis,int fd);

int main(int argc, char *argv[])
{
        int fd = 0;
        short int data = 0;
        short int data2 = 0;
        int datasimple = 0;

        fd = wiringPiI2CSetup(DEV_ADDRESS);

        // Data Format Register
        datasimple = wiringPiI2CReadReg8(fd,0x31);
        wiringPiI2CWriteReg8(fd,0x31,datasimple|0xb);

        wiringPiI2CWriteReg8(fd,0x2d,0x08); //POWER_CTL
        usleep(11000);

        // erase offset bits
        wiringPiI2CWriteReg8(fd,0x1e,0);
        wiringPiI2CWriteReg8(fd,0x1f,0);
        wiringPiI2CWriteReg8(fd,0x20,0);
        usleep(11000);
        // calibrate X axis
        data = axis_sample_average(X_REG,fd);
        wiringPiI2CWriteReg8(fd,0x1e,-(data / 4));
        // calibrate Y axis
        data = axis_sample_average(Y_REG,fd);
        wiringPiI2CWriteReg8(fd,0x1f,-(data / 4));
        // calibrate Z axis
        data = axis_sample_average(Z_REG,fd);
        wiringPiI2CWriteReg8(fd,0x20,-((data - 256 ) / 4));

        usleep(100000);

        float gravity_range = 32.0; // plus/minus 16g
        float resolution = 8192; // 13 bit resolution
```

```c
        float factor = gravity_range/resolution;
        while(1){
                fprintf(stderr,"x:%f\n",axis_sample(X_REG,fd) * factor); // X
                fprintf(stderr,"y:%f\n",axis_sample(Y_REG, fd) * factor); // Y
                fprintf(stderr,"z:%f\n\n",axis_sample(Z_REG,fd) * factor); // Z
                usleep(100000);
        }

        return 0;
}

short int axis_sample(int axis,int fd)
{
        short int data = 0;
        short int data2 = 0;

        usleep(10000);
        data  =  wiringPiI2CReadReg8(fd,axis);
        data2 =  wiringPiI2CReadReg8(fd,axis+1);

        return ( (data2<<8)|data );
}

short int axis_sample_average(int axis, int fd)
{
        int c = 10;
        int value = 0;

        while(c--){
                value += axis_sample(axis, fd);
        }

        return ( value/10 );
}
```

Listing 9.1: C code for ADXL345

Chapter 10

Servo Motor

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

Servo motors consist of a DC motor that is attached to a potentiometer and a control circuit. The position of the motor shaft can be controlled by sending a PWM signal to the controller. The Servo motor consists of a shell, circuit board, non-core motor, gear and location detection modules. Its working principle is as follow:

1. The Raspberry Pi sends a PWM signal to the servo motor.
2. 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.
3. 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 is shown in Figure 10.1. The servo expects to see a pulse every 20 ms. The length of the pulse will determine how far the motor turns. For example, a 1.5ms pulse will make the motor turn to the 90 degree position (neutral position). The minimum pulse duration is 0.5 ms wide to position it at $0°$ and the maximum pulse width is 2.5 ms to position it to $180°$.

Hence, the pulse width w_α for an angle α can be calculated as:

$$w_\alpha = \frac{(w_{max} - w_{min}) \times (\alpha - \alpha_{min})}{(\alpha_{max} - \alpha_{min})} + w_{min} = \frac{(2.5 - 0.5) \times (\alpha - 0)}{(180 - 0)} + 0.5,$$

where w_{max} and w_{min} are the expected maximum and minimum pulse widths and α_{max} and α_{min} are the maximum and minimum angles of the servo motor.

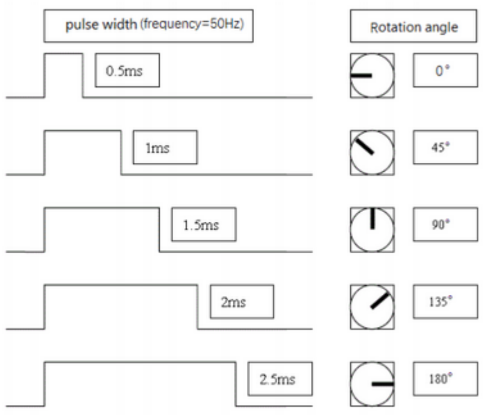
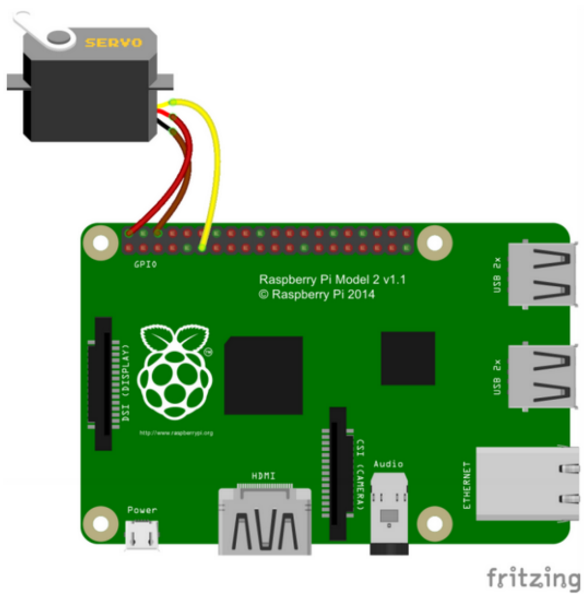


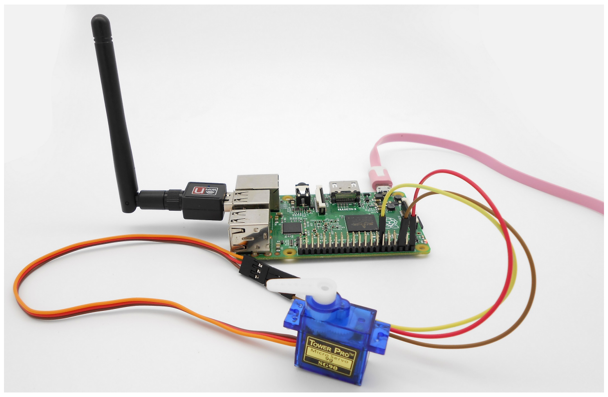
Figure 10.1: Servo Motor Principle.

Pin Number	Signal Name	(Futaba)	(JR)	(Hitec)
1	Ground	Black	Brown	Black
2	Power Supply	Red	Red	Red or Brown
3	Control Signal	White	Orange	Yellow or White

Table 10.1: Color coding for the wiring of Servo



(a) Circuit Connection.



(b) Servo Connected with Raspberry Pi.

Figure 10.2: Servo Connection with Raspberry Pi.

There are three pins in a Servo motor, where we should **use 5V for power supply**. The color coding of wiring of the servo can be of three different types as shown in Table 10.1.

In the next step, you should connect the Servo motor with the Raspberry Pi as shown in Figure 10.2, using the color code in Table 10.1.

The script, as shown in Listing 10.1, uses Software PWM signal generated by wiringPi Library to turn the motor to the correct position.

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

// WiringPi PIN 0 ==> BCM 17
#define ServoPin    0

// Map the angle to PWM width
int Map(int angle, int fromLow, int fromHigh, int toLow, int toHigh)
{
    int mapped = (toHigh-toLow)*(angle-fromLow) / (fromHigh-fromLow) + toLow;
    return mapped;
}

// Create a funtion to control the angle of the servo
void setAngle(int pin, int angle){
    if(angle < 0)
```

```
        angle = 0;
    if(angle > 180)
        angle = 180;
    int pwmVal = Map(angle, 0, 180, 5, 25);
    softPwmWrite(pin,pwmVal);
}

int main(void)
{
    int i;

    // Setup WiringPi
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }

    //initialize PMW pin of servo using soft PWM
    softPwmCreate(ServoPin, 0, 100);

    while(1){
        for(i=0;i<181;i+=10){
            // Rotate from 0 to 180;
            // the angle is increased by 10 every 500ms.
            setAngle(ServoPin,i);
            delay(500);
        }
        // Pause for a second
        delay(1000);

        for(i=181;i>-1;i-=10){
            // Rotate from 180 to 0;
            // the angle is decreased by 10 every 500ms.
            setAngle(ServoPin,i);
            delay(500);
        }
        // Pause for a second before looping again
        delay(1000);
    }
    return 0;
}
```

Listing 10.1: Servo Motor Code

Chapter 11

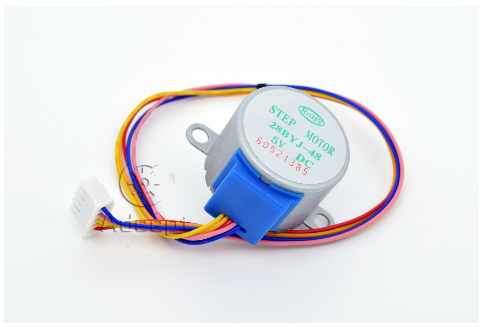
Stepper Motor

In this lab, we will learn the principles of Stepper Motor and how to control it with Raspberry Pi. We will use the code that came with your Sensor kit. For that, you can use the following command in your terminal to get all the code.

```
$git clone
https://github.com/adeept/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi.git
```

Stepper Motor: A stepper motor is a motor that converts electrical pulse signals into corresponding angular or linear displacements. Unlike DC motors, which rotate continuously when a DC voltage is applied, stepper motors normally rotate in discrete fixed-angle steps. Each time a pulse signal is input, the rotor rotates by an angle or a step forward. The output angular displacement or linear displacement is proportional to the number of pulses input, and the rotation speed is proportional to the pulse frequency. Therefore, stepper motors are also called pulse motors. 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. The motor from your Sensor kit is shown in Figure 11.1.

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 to indicate stepping state. The white socket in the middle is for connecting a stepper motor. IN1, IN2, IN3, IN4 are used to connect with the microcontroller. The power supply [5V to 12V DC] and an On/Off jumper also resides on the board as shown in Figure 11.1.



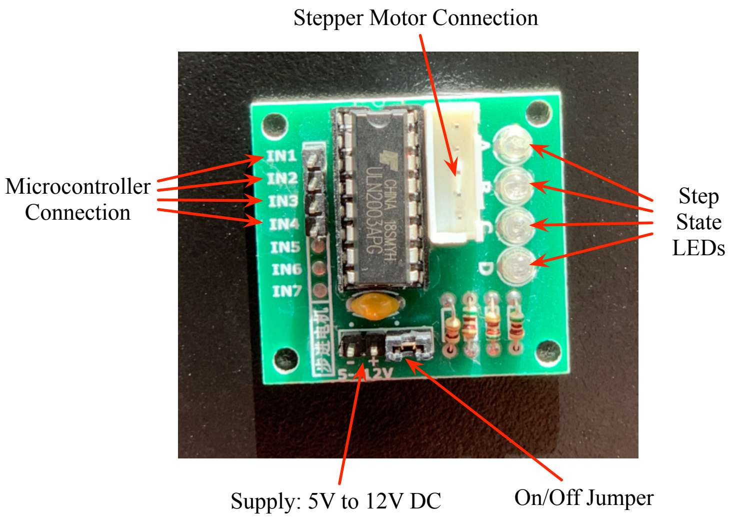


Figure 11.1: Stepper Motor and ULN 2003A.

Circuit: Figure 11.2 shows the circuit connection for the stepper motor with the driver and the Microcontroller. The script to control the Stepper Motor can be accessed at: `Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/24_stepperMotor/stepperMotor.c`. It is also shown in Listing 11.1. Based on the code that you will be using, connect 4 GPIO pins from your Raspberry Pi to the four inputs of the ULN2003A. When connecting the circuit, we should pay attention to the difference between positive and negative poles.

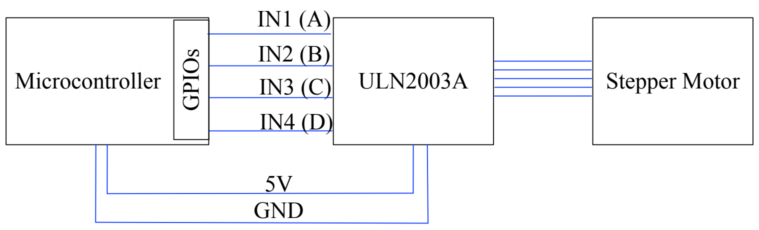


Figure 11.2: Stepper Motor Circuit.

Understanding the Code:

1. The GPIO pins should be output.
2. The four pins should be set at high level for the four pins in sequence to control the clockwise rotation of the stepping motor.
3. The four pins should be set at high level for the four pins in sequence *in reverse order* to control the *anti-clockwise* rotation of the stepping motor.
4. When all output are 0, the driver stops.
5. 512 steps are needed to turn 360°.

```
/*
 * File name   : stepperMotor.c
 * Description : control a stepper motor.
 * Website     : www.adeept.com
 * E-mail      : support@adeept.com
 * Author      : Jason
 * Date        : 2015/06/21
 */
#include <wiringPi.h>
#include <stdio.h>

#define IN1 0    // wiringPi GPIO0(pin11)
#define IN2 1    // pin12
#define IN3 2    // pin13
#define IN4 3    // pin15

void setStep(int a, int b, int c, int d)
{
        digitalWrite(IN1, a);
        digitalWrite(IN2, b);
        digitalWrite(IN3, c);
        digitalWrite(IN4, d);
}

void stop(void)
{
        setStep(0, 0, 0, 0);
}
```

```
void forward(int t, int steps)
{
        int i;

        for(i = 0; i < steps; i++){
                setStep(1, 0, 0, 0);
                delay(t);
                setStep(0, 1, 0, 0);
                delay(t);
                setStep(0, 0, 1, 0);
                delay(t);
                setStep(0, 0, 0, 1);
                delay(t);
        }
}

void backward(int t, int steps)
{
        int i;

        for(i = 0; i < steps; i++){
                setStep(0, 0, 0, 1);
                delay(t);
                setStep(0, 0, 1, 0);
                delay(t);
                setStep(0, 1, 0, 0);
                delay(t);
                setStep(1, 0, 0, 0);
                delay(t);
        }
}

int main(void)
{
        if (wiringPiSetup() < 0) {
                printf("Setup wiringPi failed!\n");
                return -1;
        }

        /* set pins mode as output */
        pinMode(IN1, OUTPUT);
        pinMode(IN2, OUTPUT);
        pinMode(IN3, OUTPUT);
        pinMode(IN4, OUTPUT);

        while (1){
                printf("forward...\n");
                forward(3, 512);

                printf("stop...\n");
                stop();
                delay(2000);       // 2s

                printf("backward...\n");
                backward(3, 256);  // 512 steps ---- 360 angle

                printf("stop...\n");
                stop();
                delay(2000);       // 2s
        }

        return 0;
}
```
Listing 11.1: Stepper Motor Code