Adeept

Ultimate Starter Kit for Raspberry Pi

Sharing Perfects Innovation

GitHub

python™

Component List

1x Triaxial Accelerometer Sensor Module (ADXL345)
1x DHT-11(Digital Temperature & Humidity Sensor)
1x Ultrasonic Distance Sensor Module
1x PIR Movement Sensor
1x PS2 Joystick Module
1x LCD1602
1x Servo
1x Stepper Motor
1x Stepper Motor Driver Module (Based on ULN2003A)
1x ADC0832
1x L9110 motor driver
1x DC Motor
1x 4*4 Matrix Keyboard
1x Breadboard Power Supply Module
1x 40 pin GPIO Extension Board
1x 40 pin GPIO Cable
2x Light Sensor (Photoresistor)
2x Analog Temperature Sensor (Thermistor)
1x Relay
1x Active Buzzer
1x Passive Buzzer
1x 7-Segment Display
1x 4-bit 7-segment Display
1x LED Bar Graph Display
1x Dot-matrix Display
2x 74HC595
2x Switch
1x RGB LED
8x Red LED
4x Green LED
4x Yellow LED
4x Blue LED
4x Button (large)
5x Button (small)
1x Button cap (red)
1x Button cap (white)
2x Button cap (blue)
16x Resistor (220Ω)
10x Resistor (1kΩ)
10x Resistor (10kΩ)
2x Potentiometer (10KΩ)
5x Capacitor (104)
4x Capacitor (10uF)
2x 1N4148 Diode
2x 1N4001 Diode
4x NPN Transistor (8050)
4x PNP Transistor (8550)
1x Breadboard
40x Male to Male Jumper Wires
20x Male to Female Jumper Wires
20x Female to Female Jumper Wires
1x Header (40pin)
1x Band Resistor Card
1x Project Box

Preface

About this kit

This is a learning kit for Raspberry Pi starter. Some common electronic components and sensors are included. Through the learning, you will get a better understanding of electronics and Raspberry Pi, and be able to make many fascinating works based on Raspberry Pi.

About Adeept

Adeept is a technical service team of open source software and hardware. Dedicated to applying the Internet and the latest industrial technology in open source area, we strive to provide best hardware support and software service for general makers and electronic enthusiasts around the world. We aim to create infinite possibilities with sharing. No matter what field you are in, we can lead you into the electronic world and bring your ideas into reality.

If you have any problems for learning, please contact us at support@adeept.com. We will do our best to help you solve the problem.

Content

About the Raspberry Pi ..... - 1 -
Raspberry Pi Pin Numbering Introduction ..... - 2 -
Raspberry Pi GPIO Library Introduction ..... - 4 -
How to use the wiringPi and the RPi.GPIO ..... - 6 -
Lesson 1 Blinking LED ..... - 10 -
Lesson 2 Active Buzzer ..... - 16 -
Lesson 3 Passive Buzzer ..... - 20 -
Lesson 4 Controlling an LED with a button ..... - 23 -
Lesson 5 Relay ..... - 28 -
Lesson 6 LED Flowing Lights ..... - 31 -
Lesson 7 Breathing LED ..... - 34 -
Lesson 8 Controlling a RGB LED with PWM ..... - 38 -
Lesson 9 7-segment display ..... - 41 -
Lesson 10 4-digit 7-segment display ..... - 44 -
Lesson 11 LCD1602 ..... - 47 -
Lesson 12 A Simple Voltmeter ..... - 51 -
Lesson 13 Matrix Keyboard ..... - 54 -
Lesson 14 Measure the distance ..... - 57 -
Lesson 15 Temperature & Humidity Sensor—DHT-11 ..... - 60 -
Lesson 16 Dot-matrix display ..... - 63 -
Lesson 17 Photoresistor ..... - 67 -
Lesson 18 Thermistor ..... - 70 -
Lesson 19 LED Bar Graph ..... - 73 -
Lesson 20 Controlling an LED through LAN ..... - 76 -
Lesson 21 Movement Detection Based on PIR ..... - 80 -
Lesson 22 DC Motor ..... - 83 -
Lesson 23 How to control a servo ..... - 87 -
Lesson 24 How to control a stepper motor ..... - 90 -
Lesson 25 How to use the acceleration sensor ADXL345 ..... - 93 -
Lesson 26 PS2 Joystick ..... - 96 -

About the Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

What's more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. We want to see the Raspberry Pi being used by kids all over the world to learn to program and understand how computers work.

Link:

https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

Raspberry Pi Pin Numbering Introduction

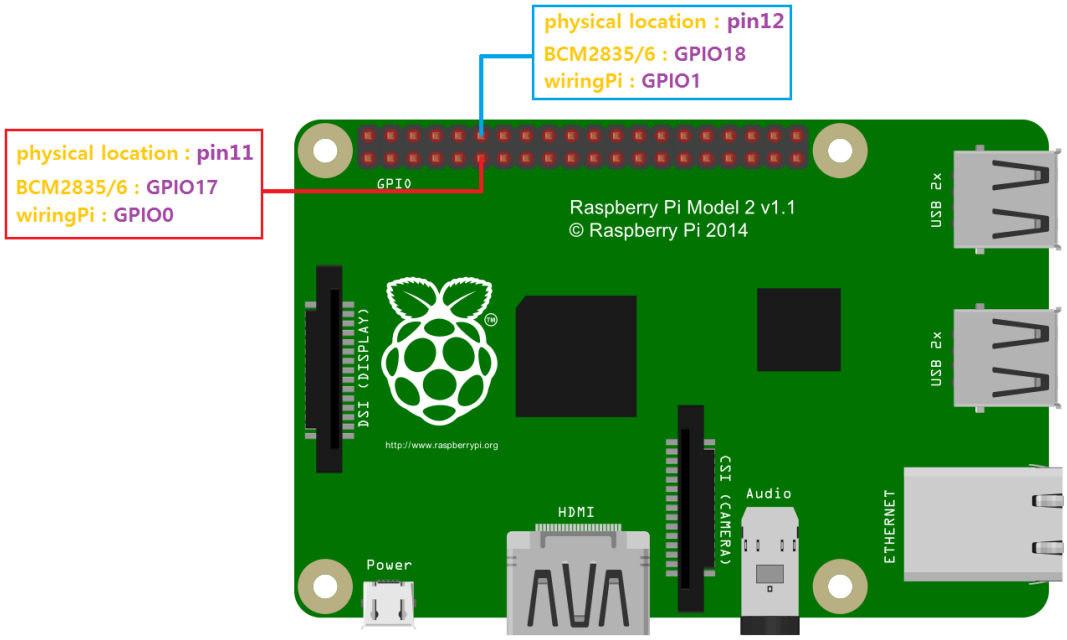
WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin
–	–	3.3v	1 | 2	5v	–	–
8	R1:0/R2:2	SDA1	3 | 4	5v	–	–
9	R1:1/R2:3	SCL1	5 | 6	0V	–	–
7	4	GPIO7	7 | 8	TXD	14	15
–	–	0V	9 | 10	RXD	15	16
0	17	GPIO0	11 | 12	GPIO1	18	1
2	R1:21/R2:27	GPIO2	13 | 14	0V	–	–
3	22	GPIO3	15 | 16	GPIO4	23	4
–	–	3.3v	17 | 18	GPIO5	24	5
12	10	MOSI	19 | 20	0V	–	–
13	9	MISO	21 | 22	GPIO6	25	6
14	11	SCLK	23 | 24	CE0	8	10
–	–	0V	25 | 26	CE1	7	11
30	0	SDA0	27 | 28	SCL0	1	31
21	5	GPIO21	29 | 30	0V	–	–
22	6	GPIO22	31 | 32	GPIO26	12	26
23	13	GPIO23	33 | 34	0V	–	–
24	19	GPIO24	35 | 36	GPIO27	16	27
25	26	GPIO25	37 | 38	GPIO28	20	28
		0V	39 | 40	GPIO29	21	29
WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin

There are three methods for numbering Raspberry Pi's GPIO:

1. Numbering according to the physical location of the pins, from left to right, top to bottom, the left is odd, the right is even.

2. Numbering according the GPIO registers of BCM2835/2836 SOC.

3. Numbering according the GPIO library wiringPi.


physical location : pin12
BCM2835/6 : GPIO18
wiringPi : GPIO1
physical location : pin11
BCM2835/6 : GPIO17
wiringPi : GPIO0
GPIO
Raspberry Pi Model 2 v1.1
© Raspberry Pi 2014
USB 2x
USB 2x
DSI (DISPLAY)
http://www.raspberrypi.org
CSI (CAMERA)
Audio
ETHERNET
HDMI
Power

Raspberry Pi GPIO Library Introduction

Currently, there are two major GPIO libraries for Raspberry Pi, RPi.GPIO and wiringPi.

RPi.GPIO:

RPi.GPIO is a python module to control Raspberry Pi GPIO channels. For more information about RPi.GPIO, please visit:

https://pypi.python.org/pypi/RPi.GPIO/

For examples and documentation, please visit:
http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/

The RPi.GPIO module is pre-installed in the official Raspbian operating system, you can use it directly.

wiringPi:

The wiringPi is a GPIO access library written in C for the BCM2835/6 SOC used in the Raspberry Pi. It's released under the GNU LGPLv3 license and is usable from C and C++ and many other languages with suitable wrappers. It's designed to be familiar to people who have used the Arduino "wiring" system.

For more information about wiringPi, please visit : http://wiringpi.com/

Install wiringPi:

Step 1 : Get the source code

```
$ git clone git://git.drogon.net/wiringPi
```

Step 2 : Compile and install

```
$ cd wiringPi
$ git pull origin
$ sudo ./build
```

Press Enter, the script "build" will automatically compile wiringPi source code and then install it to the Raspberry Pi.

Next, verify whether the wiringPi is installed successfully or not:

wiringPi includes a command-line utility gpio which can be used to program and setup the GPIO pins. You can use this to read and write the pins and even use it to control them from shell scripts.

You can verify whether the wiringPi is installed successfully or not by the following commands:

$ sudo gpio -v

```
pi@raspberrypi ~ $ sudo gpio -v
gpio version: 2.26
Copyright (c) 2012-2015 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Model 2, Revision: 1.1, Memory: 1024MB, Maker: Sony
pi@raspberrypi ~ $
```

$ sudo gpio readall

```
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---+---+------+---------+-----+-----+
pi@raspberrypi ~ $
```

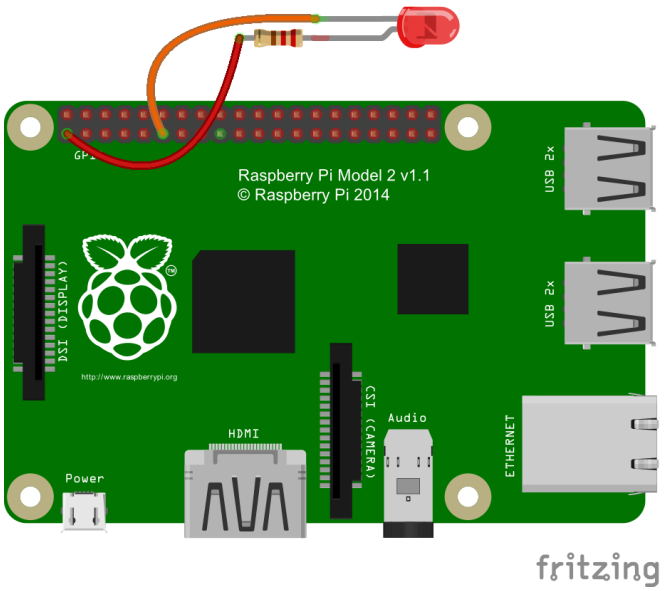
If you can see the information shown above, it indicates that the wiringPi has been installed successfully.

How to use the wiringPi and the RPi.GPIO

Here we take a blinking LED for example to illustrate how to use the wiringPi C library and the RPi.GPIO Python module.

Step 1 : Build the circuit according to the following schematic diagram

Note : Resistance=220Ω



For Python user:

Step 2 : Create a file named led.py

```
$ sudo touch led.py
```

```
pi@raspberrypi /home $ ls
pi
pi@raspberrypi /home $ sudo touch led.py
pi@raspberrypi /home $ ls
led.py  pi
pi@raspberrypi /home $
```

Step 3 : Open the file led.py with vim or nano

```
$ sudo vim led.py
```

Write down the following source code, then save and exit.

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time

Led = 11    # pin11

def setup():
    GPIO.setmode(GPIO.BOARD)    # Numbering according to the physical location
    GPIO.setup(Led, GPIO.OUT)   # Set pin mode as output
    GPIO.output(Led, GPIO.HIGH) # Output high level(+3.3V) to off the led

def loop():
    while True:
        print '...led on'
        GPIO.output(Led, GPIO.LOW)  # led on
        time.sleep(0.5)
        print 'led off...'
        GPIO.output(Led, GPIO.HIGH) # led off
        time.sleep(0.5)

def destroy():
    GPIO.output(Led, GPIO.HIGH)     # led off
    GPIO.cleanup()                  # Release resource

if __name__ == '__main__':     # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press 'Ctrl+C' to end the program
        destroy()
```

Step 4 : Run the program

$ sudo python led.py

```
pi@raspberrypi /home $ ls
led.py  pi
pi@raspberrypi /home $ sudo python led.py
...led on
led off...
...led on
led off...
...led on
led off...
```

Press Enter, you should see that the LED is blinking. Press 'Ctrl+C', the program execution will be terminated.

For C language user:

Step 2 : Create a file named led.c

$ sudo touch led.c

```
pi@raspberrypi /home $ ls
led.py  pi
pi@raspberrypi /home $ sudo touch led.c
pi@raspberrypi /home $
```

Step 3 : Open the file led.c with vim or nano

$ sudo vim led.c

Write down the following source code, then save and exit.

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

#define  Led    0  //wiringPi GPIO0, pin11

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

    pinMode(Led, OUTPUT);

    while(1){
        digitalWrite(Led, LOW);   //led on
        printf("led on...\n");
        delay(500);
        digitalWrite(Led, HIGH);  //led off
        printf("...led off\n");
        delay(500);
    }

    return 0;
}
```

Step 4 : Compile the code

$ sudo gcc led.c -lwiringPi

```
pi@raspberrypi /home $ ls
led.c  led.py  pi
pi@raspberrypi /home $ sudo gcc led.c -lwiringPi
pi@raspberrypi /home $
```

After executing this command, you'll find a file named a.out appear in the current directory. It is an executable program.

```
pi@raspberrypi /home $ ls
a.out  led.c  led.py  pi
pi@raspberrypi /home $
```

Step 5 : Run the program

$ sudo ./a.out

```
pi@raspberrypi /home $ sudo ./a.out
led on...
...led off
led on...
...led off
```

Press Enter, you should see that the LED is blinking. Press 'Ctrl+C', the program execution will be terminated.

Resources :

http://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/

http://wiringpi.com/reference/

NOTE:

Before learning the next courses, please copy the source code we provided to your Raspberry Pi's /home/ directory, or you can get the source code directly from our github repository:

C Language Source Code :

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

Python Source Code :

$ git clone https://github.com/adeept/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi.git

Lesson 1 Blinking LED

Overview

In this tutorial, we will start the journey of learning Raspberry Pi. In the first lesson, we will learn how to control an LED.

Requirement

- 1* Raspberry Pi
- 1* 220Ω Resistor
- 1* LED
- 1* Breadboard
- 2* Jumper Wires

Principle

In this lesson, we will program the Raspberry Pi to output high(+3.3V) and low level(0V), and then make an LED which is connected to the Raspberry Pi GPIO flicker with a certain frequency.

1. What is LED ?

The LED is the abbreviation of light emitting diode. It is usually made of gallium arsenide, gallium phosphide semiconductor materials. The LED has two electrodes, a positive electrode and a negative electrode, it will light only when a forward current passes, and it can be red, blue, green or yellow light, etc. The color of light depends on the materials it was made.

In general, the drive current for LED is 5-20mA. Therefore, in reality it usually needs an extra resistor for current limitation so as to protect the LED.

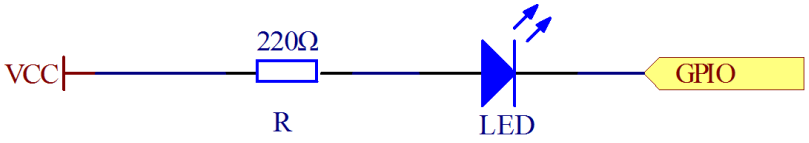
2. What is the resistor ?

The main function of the resistor is to limit current. In the circuit, the character 'R' represents resistor, and the unit of resistor is ohm(Ω).

The band resistor is used in this experiment. A band resistor is one whose surface is coated with some particular color through which the resistance can be identified directly.

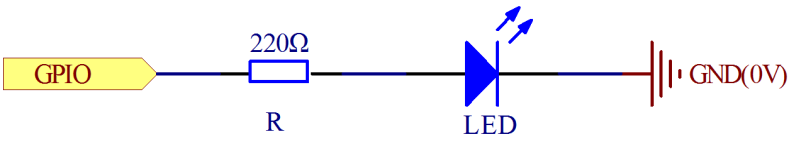
There are two methods for connecting an LED with Raspberry Pi GPIO:

①



As shown in the schematic diagram above, the anode of LED is connected to VCC(+3.3V), and the cathode of LED is connected to the Raspberry Pi GPIO. When the GPIO output low level, the LED is on; when the GPIO output high level, the LED is off.

②



As shown in the schematic diagram above, the anode of LED is connected to Raspberry Pi GPIO via a resistor, and the cathode of LED is connected to the ground (GND). When the GPIO output high level, the LED is on; when the GPIO output low level, the LED is off.

The size of the current-limiting resistor is calculated as follows : 5~20mA current is required to make an LED on, and the output voltage of the Raspberry Pi GPIO is 3.3V, so we can get the resistance：

R = U / I = 3.3V / (5~20mA) = 165Ω~660Ω

In this experiment, we choose a 220ohm resistor.

The experiment is based on method ①, we select pin 11 of Raspberry Pi to control an LED. When the pin 11 of Raspberry Pi is programmed to output low level, then the LED will be lit, next delay for the amount of time, and then programmed the pin 11 to high level to make the LED off. Continue to perform the above process, you can get a blinking LED.

3. Key functions

C language user:

- **int wiringPiSetup (void)**

The function must be called at the start of your program or your program will fail to work correctly. You may experience symptoms from it simply not working to segfaults and timing issues.

Note: This function needs to be called with root privileges.

- **void pinMode (int pin, int mode)**

This sets the mode of a pin to either **INPUT**, **OUTPUT**, **PWM_OUTPUT** or **GPIO_CLOCK**. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes.

This function has no effect when in Sys mode. If you need to change the pin mode, then you can do it with the gpio program in a script before you start your program.

- **void digitalWrite (int pin, int value)**

Writes the value **HIGH** or **LOW** (1 or 0) to the given pin which must have been previously set as an output. ***WiringPi*** treats any non-zero number as HIGH, however 0 is the only representation of LOW.

- **void delay (unsigned int howLong)**

This causes program execution to pause for at least howLong milliseconds. Due to the multi-tasking nature of Linux it could be longer. Note that the maximum delay is an unsigned 32-bit integer or approximately 49 days.

Python user:

- **GPIO.setmode(GPIO.BOARD)**

There are two ways of numbering the IO pins on a Raspberry Pi within RPi.GPIO. The first is using the **BOARD** numbering system. This refers to the pin numbers on the P1 header of the Raspberry Pi board. The advantage of using this numbering system is that your hardware will always work, regardless of the board revision of the RPi. You will not need to rewire your connector or change your code.

The second numbering system is the **BCM**(GPIO.BCM) numbers. This is a lower level way of working - it refers to the channel numbers on the Broadcom SOC. You have to always work with a diagram of which channel number goes to which pin on the RPi board. Your script could break between revisions of Raspberry Pi boards.

- **GPIO.setup(channel, mode)**

This sets every channel you are using as an input(GPIO.IN) or an output(GPIO.OUT).

- **GPIO.output(channel, state)**

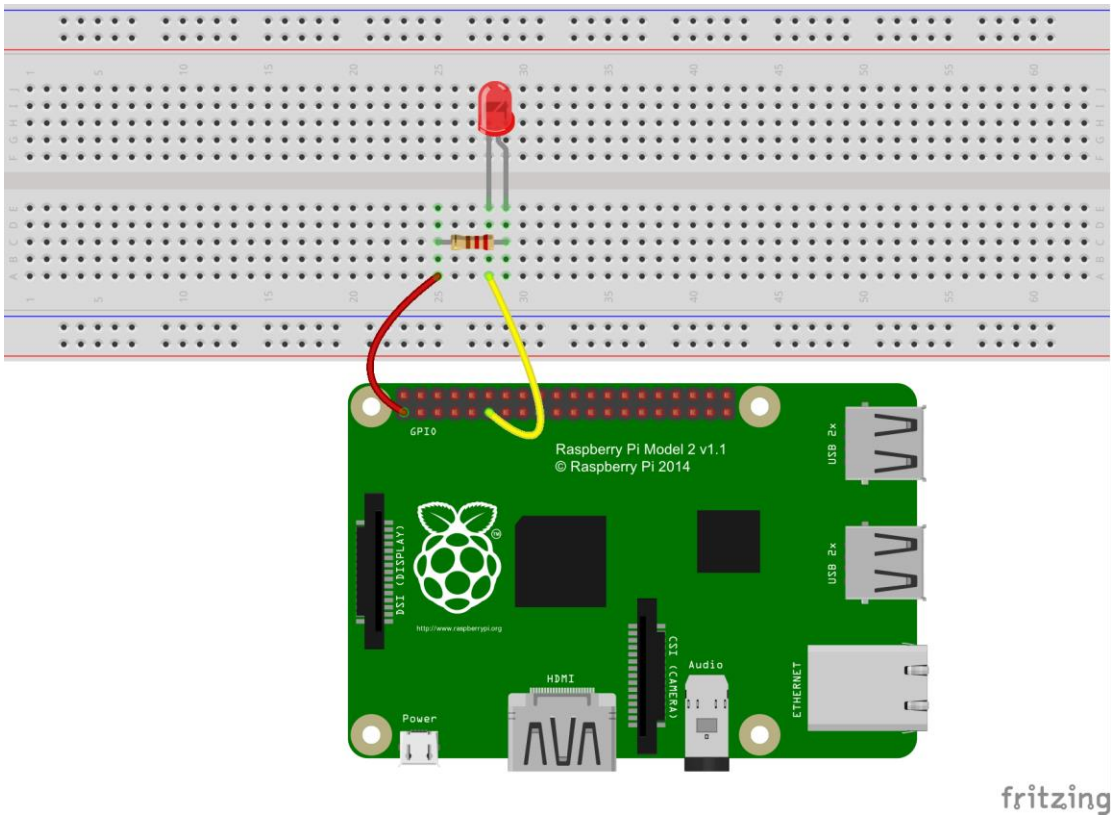
This sets the output state of a GPIO pin. Where channel is the channel number based on the numbering system you have specified (BOARD or BCM). **State** can be 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.

- **GPIO.cleanup( )**

At the end any program, it is good practice to clean up any resources you might have used. This is no different with RPi.GPIO. By returning all channels you have used back to inputs with no pull up/down, you can avoid accidental damage to your RPi by shorting out the pins. Note that this will only clean up GPIO channels that your script has used. Note that GPIO.cleanup() also clears the pin numbering system in use.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/01_blinkingLed/blinkingLed.c)

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

#define  LedPin    0

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

    pinMode(LedPin, OUTPUT);

    while(1){
        digitalWrite(LedPin, LOW);   //led on
        printf("led on...\n");
        delay(500);
        digitalWrite(LedPin, HIGH);  //led off
        printf("...led off\n");
        delay(500);
    }

    return 0;
}
```

2.2 Compile the program

```
$ gcc blinkingLed.c -o led -lwiringPi
```

Note : The parameter '-o' is to specify a file name for the compiled executable program. If you do not use this parameter, the default file name is *a.out*.

2.3 Run the program

```
$ sudo ./led
```

Python user:

2.1 Edit and save the code with vim or nano.

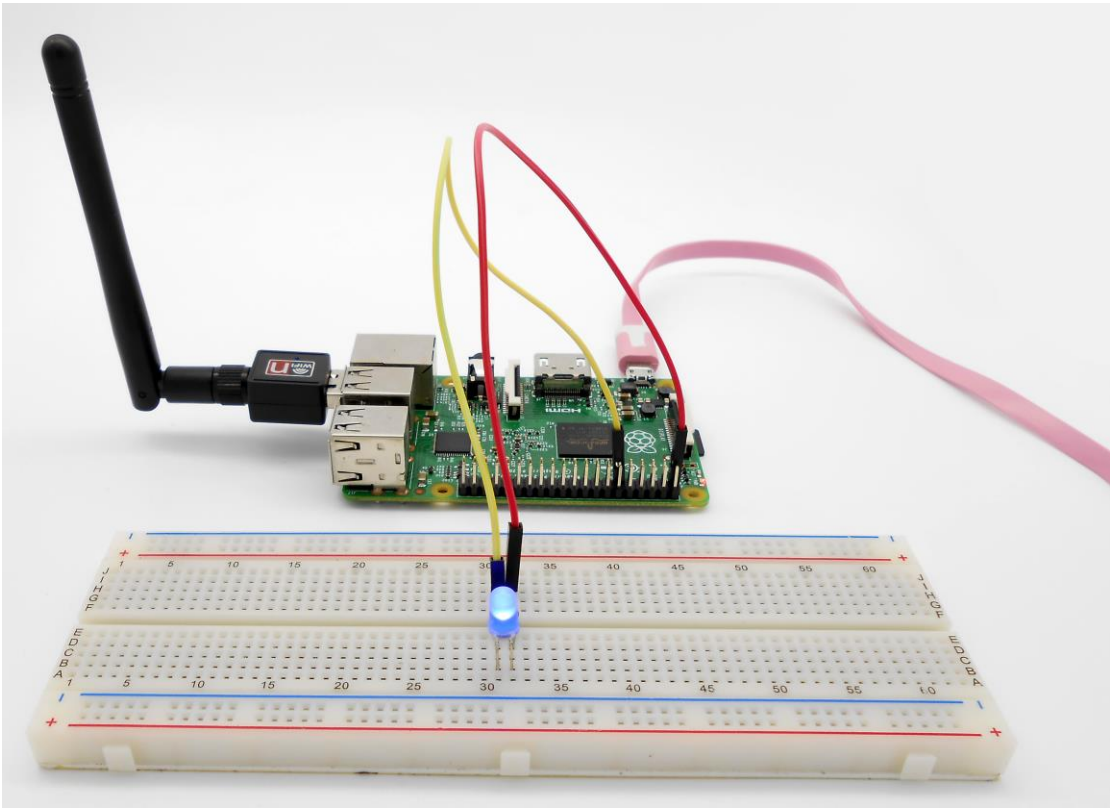
(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/01_blinkingLed_1.py)

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time

LedPin = 11    # pin11

def setup():
    GPIO.setmode(GPIO.BOARD)       # Numbers GPIOs by physical location
    GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode is output
    GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

def loop():
    while True:
        print '...led on'
        GPIO.output(LedPin, GPIO.LOW)  # led on
        time.sleep(0.5)
        print 'led off...'
        GPIO.output(LedPin, GPIO.HIGH) # led off
        time.sleep(0.5)

def destroy():
    GPIO.output(LedPin, GPIO.HIGH)     # led off
    GPIO.cleanup()                     # Release resource

if __name__ == '__main__':     # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will
be  executed.
        destroy()
```

2.2 Run the program

```
$ sudo python 01_blinkingLed_1.py
```

Press Enter, and then you can see the LED is blinking.



Lesson 2 Active Buzzer

Overview

In this lesson, we will learn how to program the Raspberry Pi to make an active buzzer sound.

Requirement

- 1* Raspberry Pi
- 1* Active buzzer
- 1* 1 kΩ Resistor
- 1* NPN Transistor (S8050)
- 1* Breadboard
- Several Jumper wires

Principle

A buzzer or beeper is an audio signaling device. As a type of electronic buzzer with integrated structure, which use DC power supply, are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic equipments, telephones, timers and other electronic products for voice devices. Buzzers can be categorized as active and passive buzzers (See the following pictures).

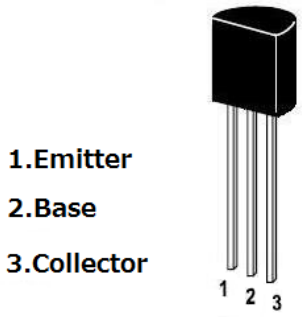
When you place the pins of buzzers upward, you can see that two buzzers are different, the buzzer that green circuit board exposed is the passive buzzer.

In this study, the buzzer we used is active buzzer. Active buzzer will sound as long as the power supply. We can program to make the Raspberry Pi output alternating high and low level, so that the buzzer sounds.

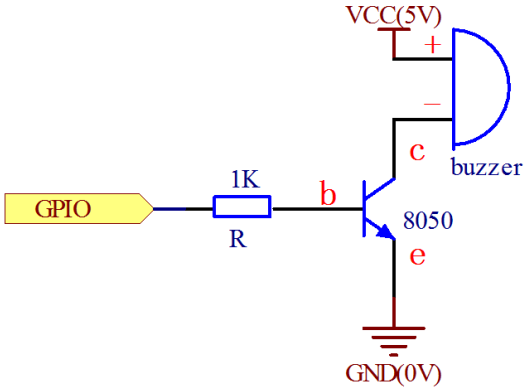
A slightly larger current is needed to make a buzzer sound. However, the output current of Raspberry Pi GPIO is weak, so we need a transistor to drive the buzzer.

The main function of transistor is blowing up the voltage or current. The transistor can also be used to control the circuit conduction or deadline. And the transistor is divided into two kinds, one kind is NPN, for instance, the S8050 we provided; another kind is PNP transistor such as the S8550 we provided. The transistor we used is as shown in

below:



There are two driving circuit for the buzzer:



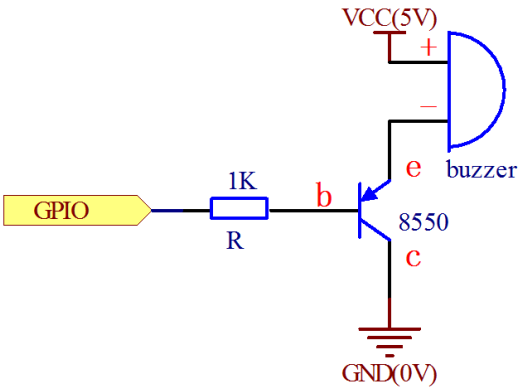


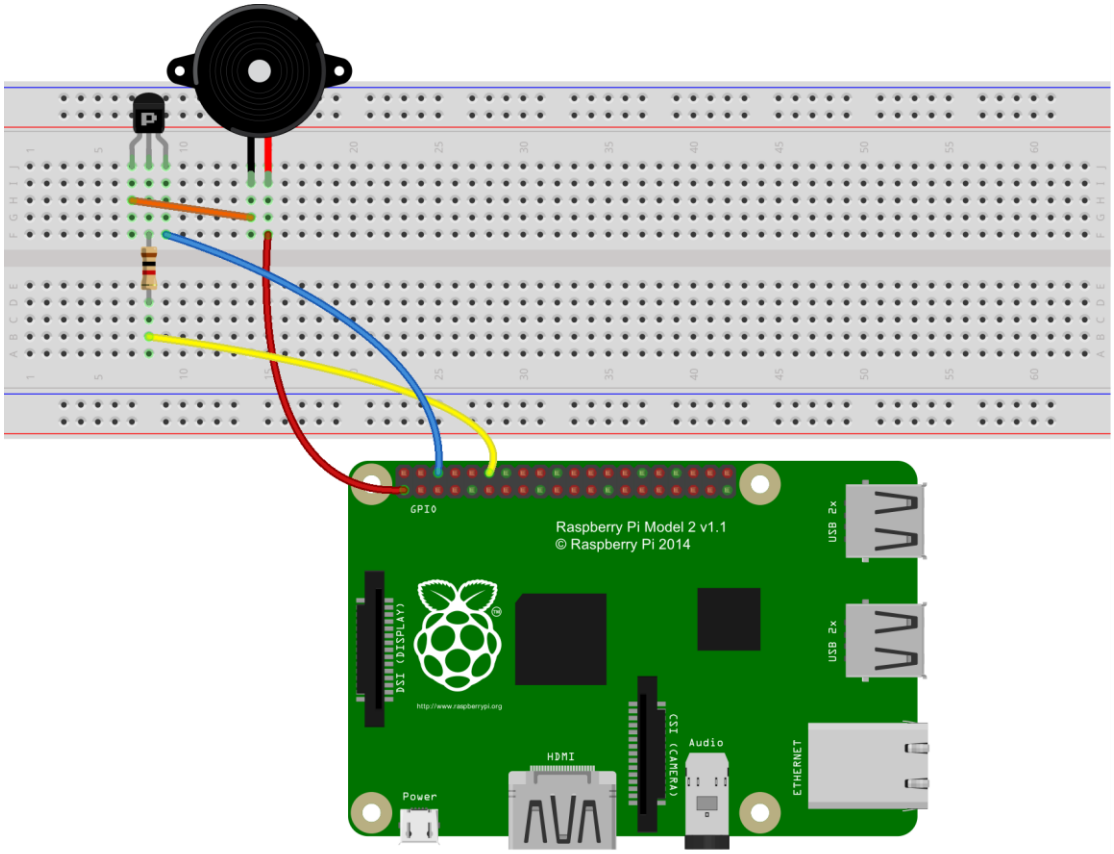
Figure1 Figure2

Figure 1: Set the Raspberry Pi GPIO as a high level, the transistor S8050 will conduct, and then the buzzer will sound; set the Raspberry Pi GPIO as low level, the transistor S8050 will cut off, then the buzzer will stop.

Figure 2: Set the Raspberry Pi GPIO as low level, the transistor S8550 will conduct, and the buzzer will sound; set the Raspberry Pi GPIO as a high level, the transistor S8550 will cut off, then the buzzer will stop.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/02_ativeBuzzer/buzzer.c)

2.2 Compile the program

```
$ gcc buzzer.c -o buzzer -lwiringPi
```

2.3 Run the program

```
$ sudo ./buzzer
```

Python user:

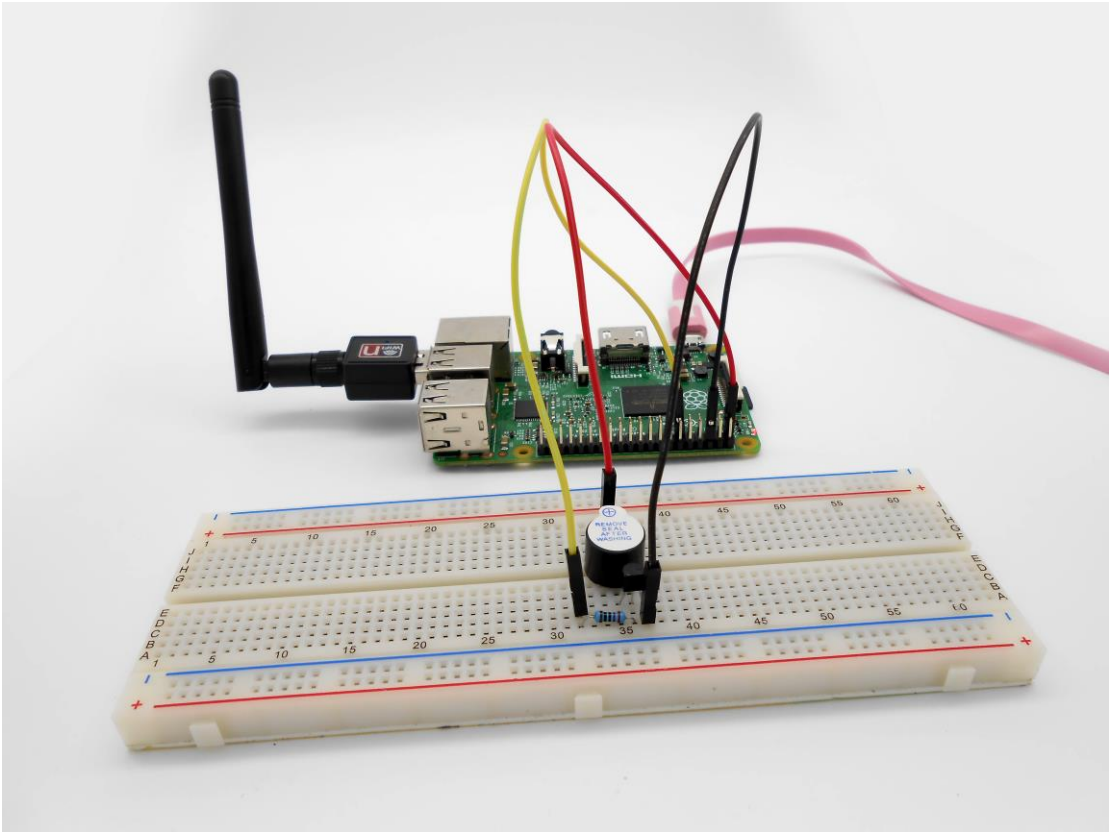
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/02_activeBuzzer.py)

2.2 Run the program

```
$ sudo python 02_activeBuzzer.py
```

Now, you should be able to hear the sound of the buzzer.



Summary

By learning this lesson, we have mastered the basic principle of the buzzer and the transistor. We also learned how to program the Raspberry Pi and then control the buzzer. I hope you can use what you have learned in this lesson to do some interesting things.

Lesson 3 Passive Buzzer

Overview

In this lesson, we will learn how to program the Raspberry Pi to make a passive buzzer sound with different frequency.

Requirement

- 1* Raspberry Pi
- 1* Passive buzzer
- 1* 1 kΩ Resistor
- 1* NPN Transistor (S8050)
- 1* Breadboard
- Several Jumper wires

Principle

As long as you send the square wave signals to a passive buzzer with different frequency, then the passive buzzer will make different sound.

In this experiment, we continuously send different square wave signal to a passive buzzer to play a piece of music.

Key functions

C language user:

- **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.

- **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.

Python user:

- GPIO.cleanup( )

At the end any program, it is good practice to clean up any resources you might have used. This is no different with RPi.GPIO. By returning all channels you have used back to inputs with no pull up/down, you can avoid accidental damage to your RPi by shorting out the pins. Note that this will only clean up GPIO channels that your script has used. Note that GPIO.cleanup() also clears the pin numbering system in use.

- p = GPIO.PWM(channel, frequency)

To create a PWM instance

- p.start(dc)

To start PWM.

- p.ChangeFrequency(freq)

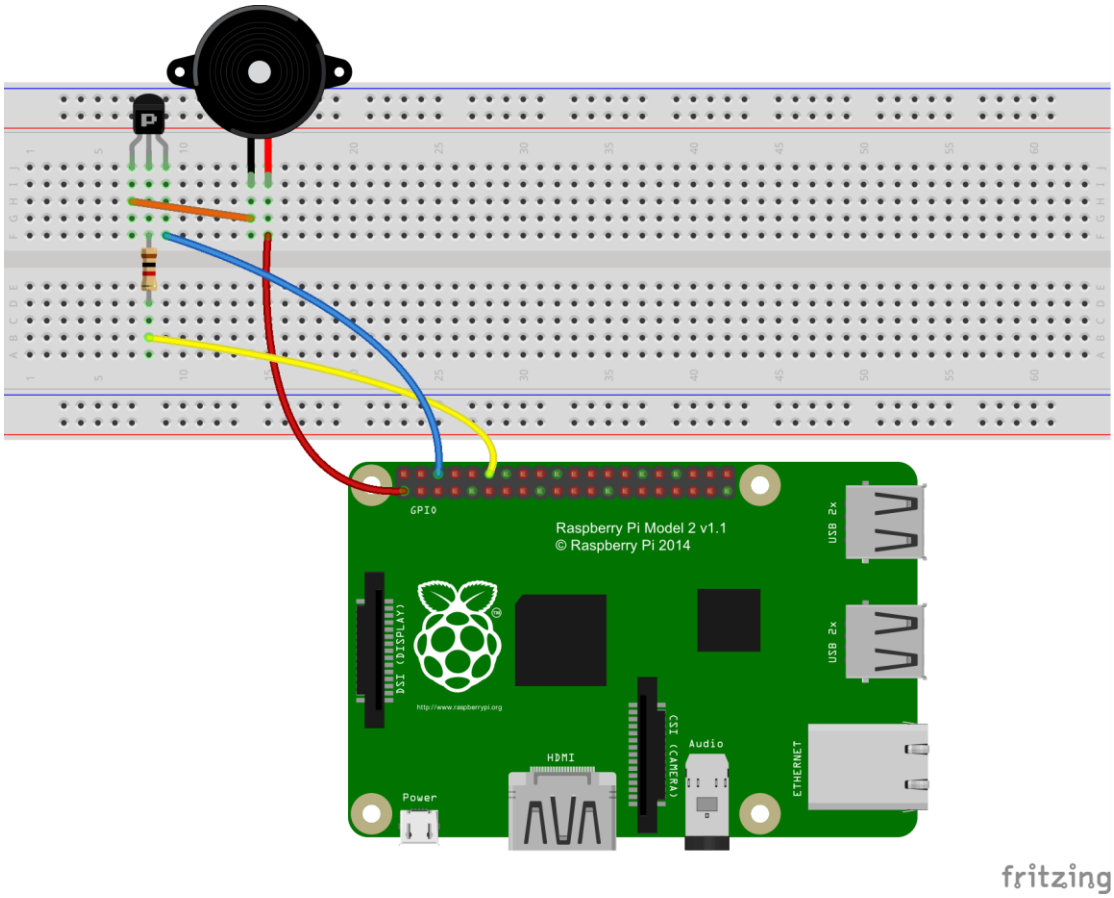
To change the frequency.

- p.stop()

To stop PWM.

Procedure

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/03_passiveBuzzer/passiveBuzzer.c)

2.2 Compile the program

```
$ gcc passiveBuzzer.c -o passiveBuzzer -lwiringPi -lpthread
```

2.3 Run the program

```
$ sudo ./passiveBuzzer
```

Python user:

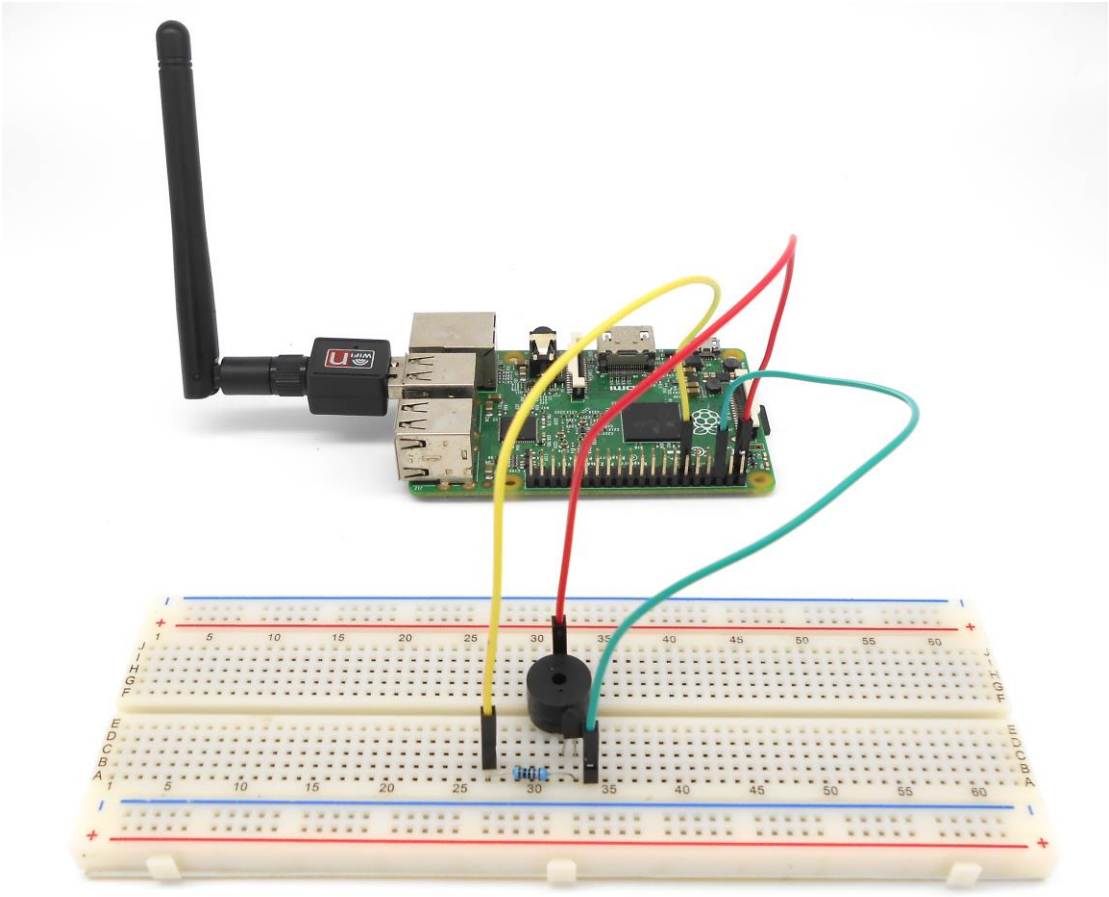
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/03_passiveBuzzer.py)

2.2 Run the program

```
$ sudo python 03_passiveBuzzer.py
```

Now, you should be able to hear the sound of the buzzer.



Lesson 4 Controlling an LED with a button

Overview

In this lesson, we will learn how to detect the status of a button, and then toggle the status of LED based on the status of the button.

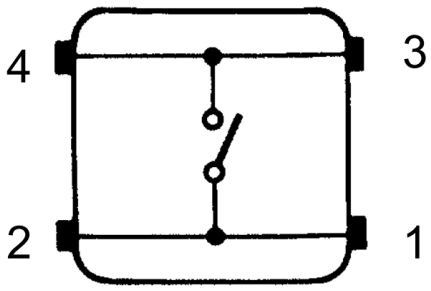
Requirement

- 1* Raspberry Pi
- 1* Button
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

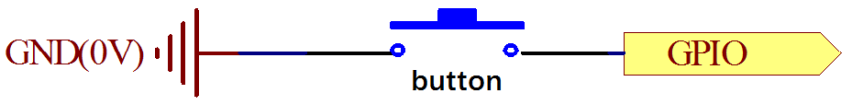
1. Button

Buttons are a common component used to control electronic devices. They are usually used as switches to connect or disconnect circuits. Although buttons come in a variety of sizes and shapes, the one used in this experiment will be a 12mm button as shown in the following pictures. Pins pointed out by the arrows of same color are meant to be connected.

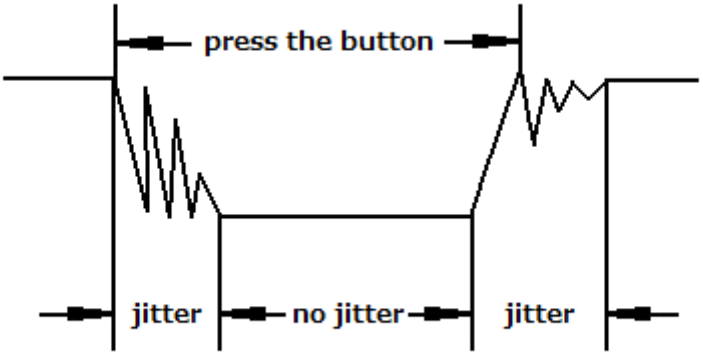


The button we used is a normally open type button. The two contacts of a button is in the off state under the normal conditions, only when the button is pressed they are closed.

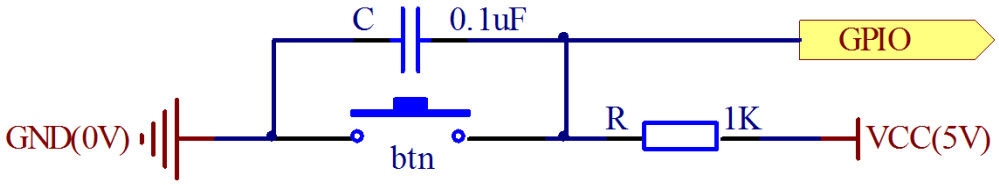
The schematic diagram we used is as follows:



The button jitter must be happen in the process of using. The jitter waveform is as the flowing:



Each time you press the button, the Raspberry Pi will think you have pressed the button many times due to the jitter of the button. We must to deal with the jitter of buttons before we use the button. We can remove the jitter of buttons through the software programming, besides, we can use a capacitance to remove the jitter of buttons. Here we introduce the software method. First, we detect whether the level of button interface is low level or high level. When the level we detected is low level, 5~10 MS delay is needed, and then detect whether the level of button interface is low or high. If the signal is low, we can confirm that the button is pressed once. You can also use a 0.1uF capacitance to clean up the jitter of buttons. The schematic diagram is shown in below:



2. interrupt

Hardware interrupts were introduced as a way to reduce wasting the processor's valuable time in polling loops, waiting for external events. They may be implemented in hardware as a distinct system with control lines, or they may be integrated into the memory subsystem.

3. Key functions:

C user:

- **void pullUpDnControl (int pin, int pud)**

This sets the pull-up or pull-down resistor mode on the given pin, which should be set as an input. Unlike the Arduino, the BCM2835 has both pull-up a down internal resistors. The parameter pud should be; PUD_OFF, (no pull up/down), PUD_DOWN (pull to ground) or PUD_UP (pull to 3.3v). The internal pull up/down resistors have a value of approximately 50KΩ on the Raspberry Pi.

This function has no effect on the Raspberry Pi's GPIO pins when in Sys mode. If you

need to activate a pull-up/pull-down, then you can do it with the gpio program in a script before you start your program.

- int digitalRead (int pin)

This function returns the value read at the given pin. It will be HIGH or LOW (1 or 0) depending on the logic level at the pin.

- int wiringPiISR (int pin, int edgeType, void (*function)(void))

This function registers a function to received interrupts on the specified pin. The edgeType parameter is either **INT_EDGE_FALLING**, **INT_EDGE_RISING**, **INT_EDGE_BOTH** or **INT_EDGE_SETUP**. If it is **INT_EDGE_SETUP** then no initialisation of the pin will happen – it's assumed that you have already setup the pin elsewhere (e.g. with the gpio program), but if you specify one of the other types, then the pin will be exported and initialised as specified. This is accomplished via a suitable call to the gpio utility program, so it need to be available.

The pin number is supplied in the current mode – native wiringPi, BCM_GPIO, physical or Sys modes.

This function will work in any mode, and does not need root privileges to work.

The function will be called when the interrupt triggers. When it is triggered, it's cleared in the dispatcher before calling your function, so if a subsequent interrupt fires before you finish your handler, then it won't be missed. (However it can only track one more interrupt, if more than one interrupt fires while one is being handled then they will be ignored)

This function is run at a high priority (if the program is run using sudo, or as root) and executes concurrently with the main program. It has full access to all the global variables, open file handles and so on.

Python user:

- GPIO.input(channel)

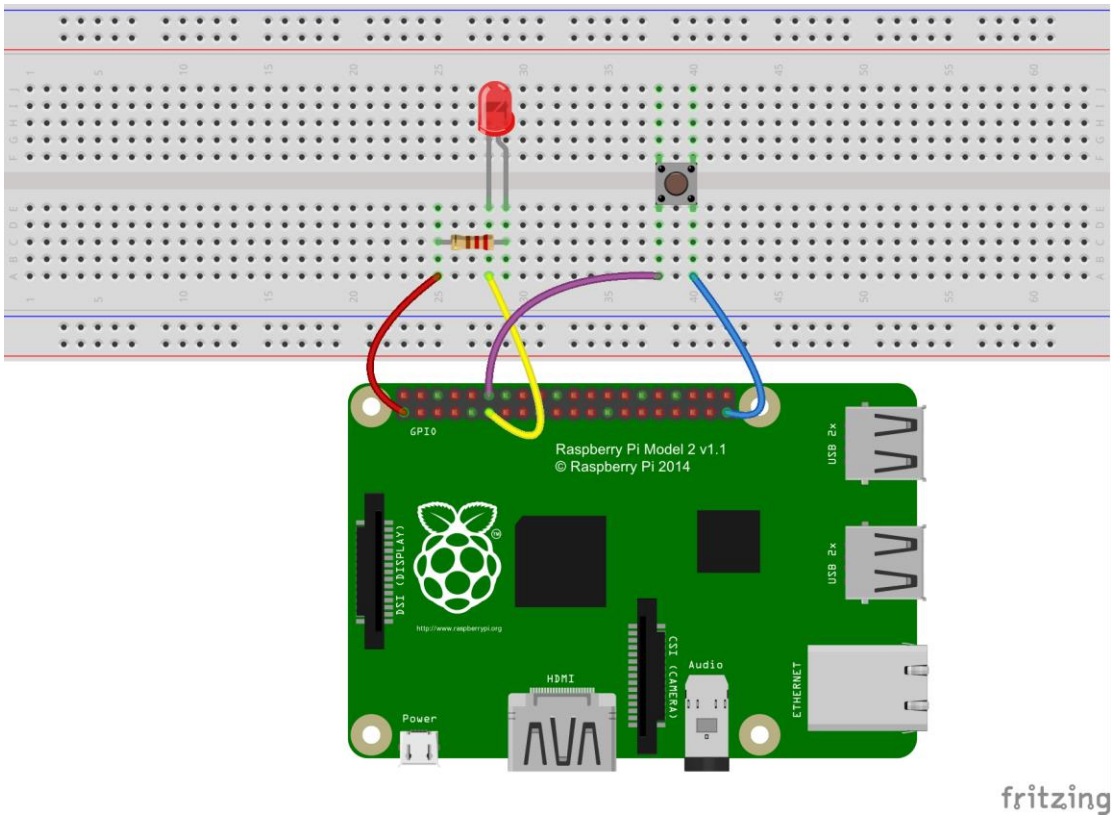
This is used for reading the value of a GPIO pin. Where channel is the channel number based on the numbering system you have specified (BOARD or BCM)). This will return either 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.

- GPIO.add_event_detect(channel, mode)

The event_detected( ) function is designed to be used in a loop with other things, but unlike polling it is not going to miss the change in state of an input while the CPU is busy working on other things. This could be useful when using something like Pygame or PyQt where there is a main loop listening and responding to GUI events in a timely basis.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/04_btnAndLed/btnAndLed_2.c)

2.2 Compile the program

```
$ gcc btnAndLed_2.c -o btnAndLed -lwiringPi
```

2.3 Run the program

```
$ sudo ./btnAndLed
```

Python user:

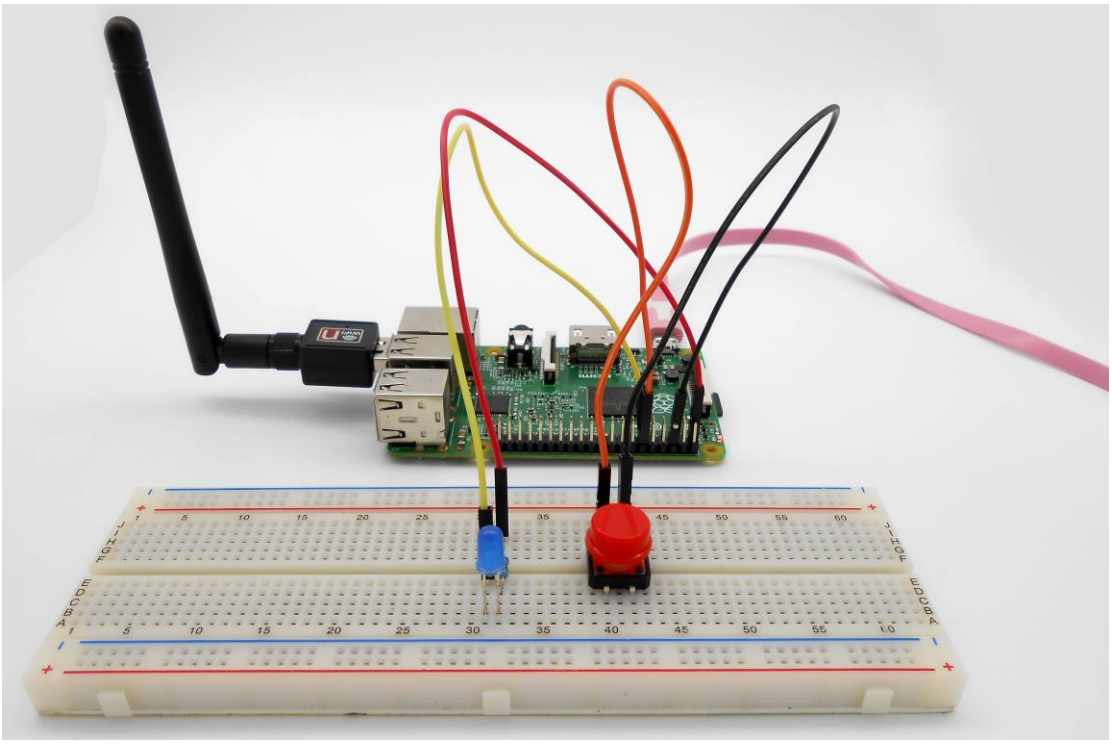
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/04_btnAndLed_1.py)

2.2 Run the program

```
$ sudo python 04_btnAndLed_1.py
```

Now, when you press the button, you can see the state of the LED will be toggled. (ON->OFF, OFF->ON).



Summary

Through this lesson, you should have learned how to use the Raspberry Pi detect an external button status, and then toggle the status of LED relying on the status of the button detected before.

Lesson 5 Relay

Overview

In this lesson, we will learn how to control a relay to cut off or connect a circuit.

Requirement

- 1* Raspberry Pi
- 1* Relay
- 1* NPN Transistor (S8050)
- 1* Diode (1N4001)
- 1* 1KΩ Resistor
- 1* Breadboard
- Several Jumper wires

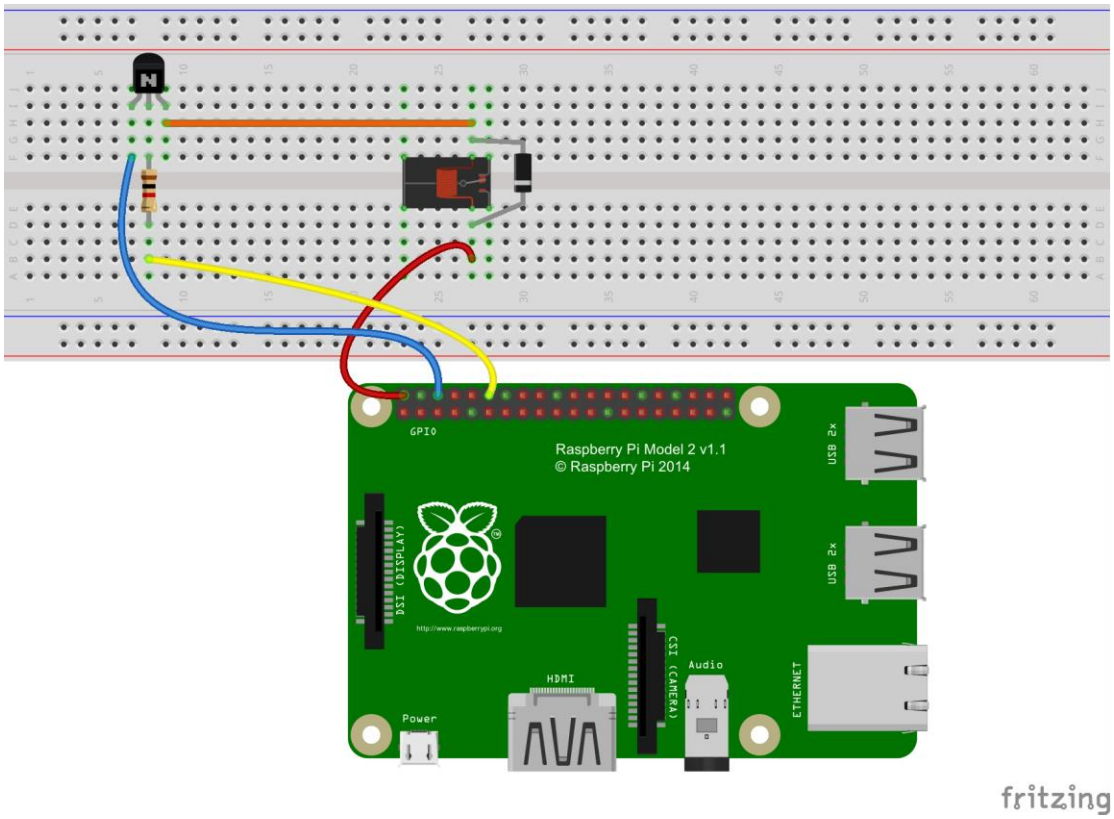
Principle

A relay is an electrically operated switch. It is generally used in automatic control circuit. Actually, it is an "automatic switch" which uses low current to control high current. It plays a role of automatic regulation, security protection and circuit switch. When an electric current is passed through the coil it generates a magnetic field that activates the armature, and the consequent movement of the movable contact(s) either makes or breaks (depending upon construction) a connection with a fixed contact. If the set of contacts was closed when the relay was de-energized, then the movement opens the contacts and breaks the connection, and vice versa if the contacts were open. When the current to the coil is switched off, the armature is returned by a force, approximately half as strong as the magnetic force, to its relaxed position. Usually this force is provided by a spring, but gravity is also used commonly in industrial motor starters. Most relays are manufactured to operate quickly. In a low-voltage application this reduces noise; in a high voltage or current application it reduces arcing.

When the coil is energized with direct current, a diode is often placed across the coil to dissipate the energy from the collapsing magnetic field at deactivation, which would otherwise generate a voltage spike dangerous to semiconductor circuit components.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/05_relay/relay.c)

2.2 Compile the program

```
$ gcc relay.c -o relay -lwiringPi
```

2.3 Run the program

```
$ sudo ./relay
```

Python user:

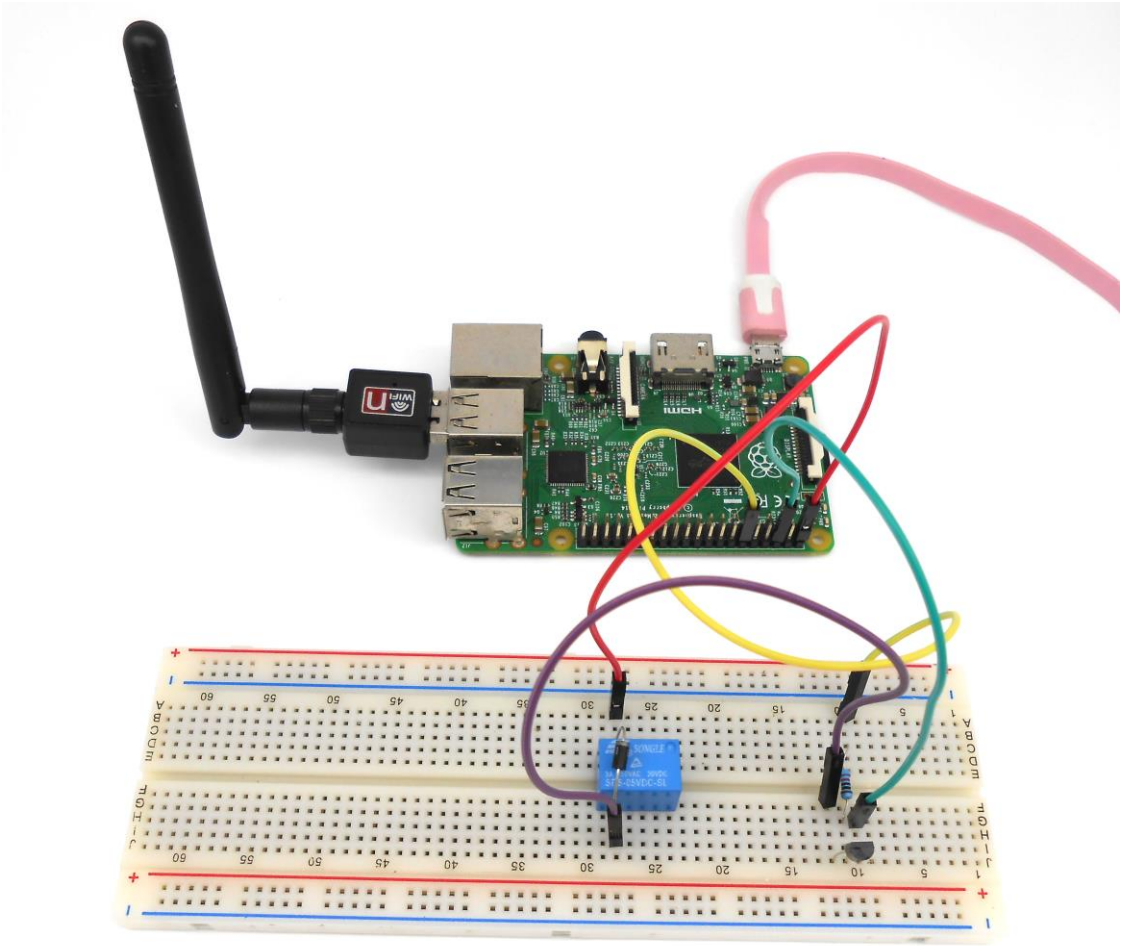
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/05_relay.py)

2.2 Run the program

```
$ sudo python 05_relay.py
```

Press Enter, you should be able to hear the sound of relay toggling.



Lesson 6 LED Flowing Lights

Overview

In the first class, we have learned how to make an LED blink by programming the Raspberry Pi. Today, we will use the Raspberry Pi to control 8 LEDs, so that 8 LEDs showing the result of flowing.

Requirement

- 1* Raspberry Pi
- 8* LED
- 8* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

The principle of this experiment is very simple. It is very similar with the first class.

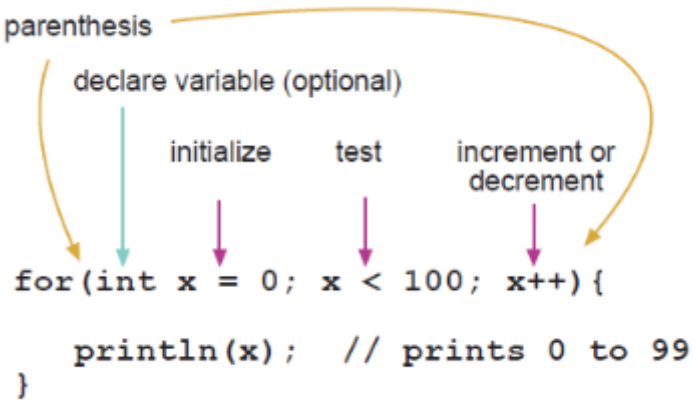
Key function:

- for statements

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

There are three parts to the for loop header:

```
for (initialization; condition; increment) {
    //statement(s);
}
```

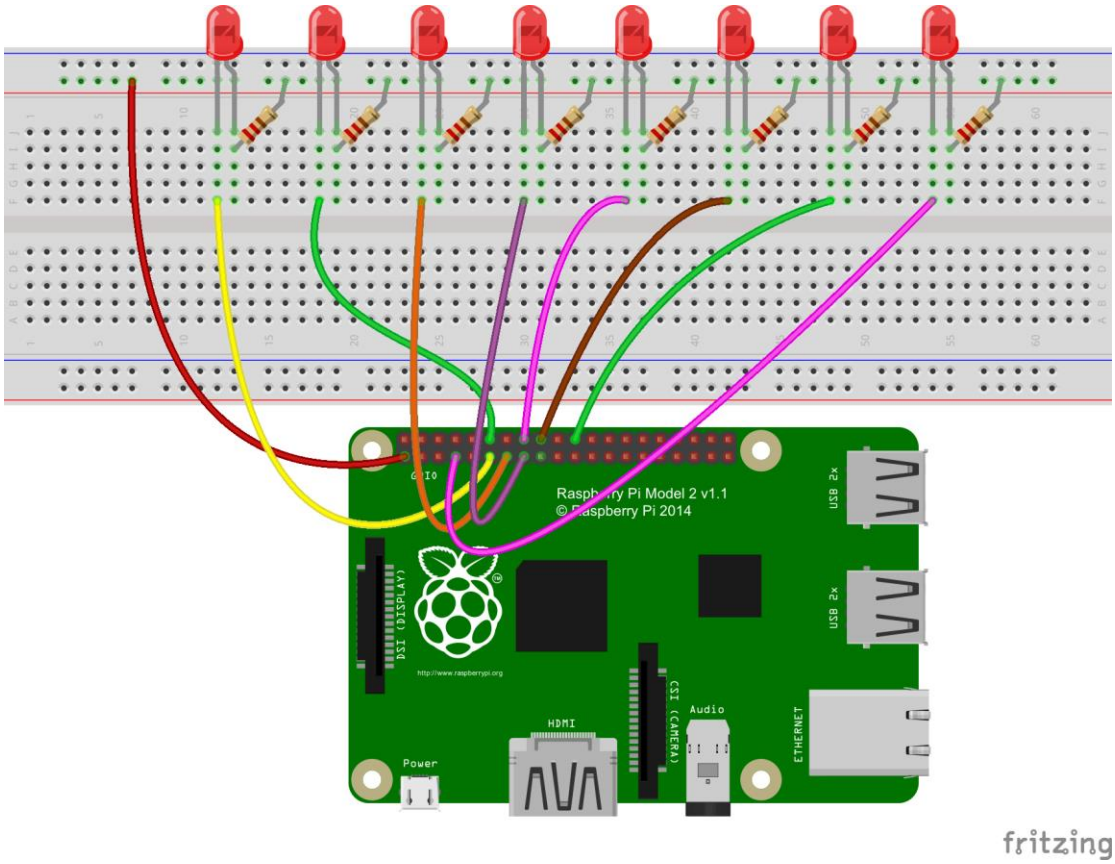


The initialization happens first and exactly once. Each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed,

then the condition is tested again. When the condition becomes false, the loop ends.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/06_flowingLed/flowingLed.c)

2.2 Compile the program

```
$ gcc flowingLed.c -o flowingLed -lwiringPi
```

Run the program

```
$ sudo ./flowingLed
```

Python user:

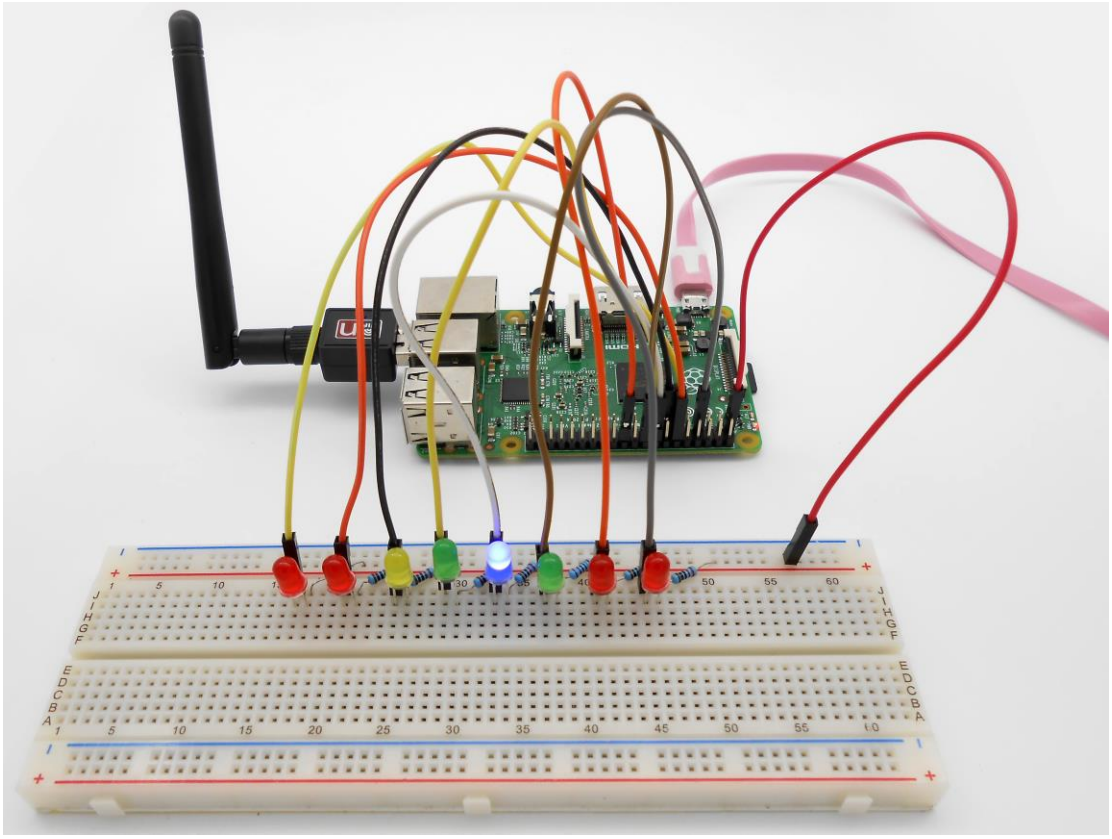
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/06_flowingLed.py)

2.2 Run the program

```
$ sudo python 06_flowingLed.py
```

Now, you should see 8 LEDs are lit in sequence from the right red one to the left, next from the left to the right one. And then repeat the above phenomenon.



Summary

Through this simple and fun experiment, we have learned more skilled programming about the Raspberry Pi. In addition, you can also modify the circuit and code we provided to achieve even more dazzling effect.

Lesson 7 Breathing LED

Overview

In this lesson, we will learn how to program the Raspberry Pi to generate PWM signal. And use the PWM square-wave signal control an LED gradually becomes brighter and then gradually becomes dark like the animal's breath.

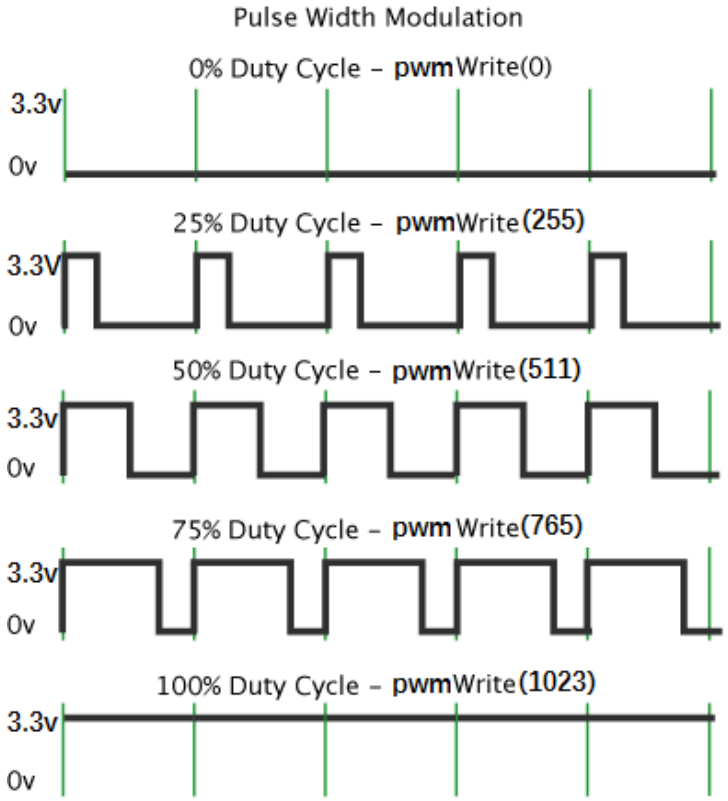
Requirement

- 1* Raspberry Pi
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 3.3v controlling the brightness of the LED.

In the graphic below, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Raspberry Pi's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to pwmWrite() is on a scale of 0-1023, such that pwmWrite(1023) requests a 100% duty cycle (always on), and pwmWrite(511) is a 50% duty cycle (on half the time) for example.



Key function:

C user:

- pwmWrite(int pin, int value)

Writes the value to the PWM register for the given pin. The Raspberry Pi has one on-board PWM pin, pin 1 (BMC_GPIO 18, Phys 12) and the range is 0-1024. Other PWM devices may have other PWM ranges.

This function is not able to control the Pi's on-board PWM when in Sys mode.

Python user:

- p = GPIO.PWM(channel, frequency)

This is used for creating a PWM.

- p.start(dc)

Start the pwm you have created.

- p.ChangeFrequency(freq)

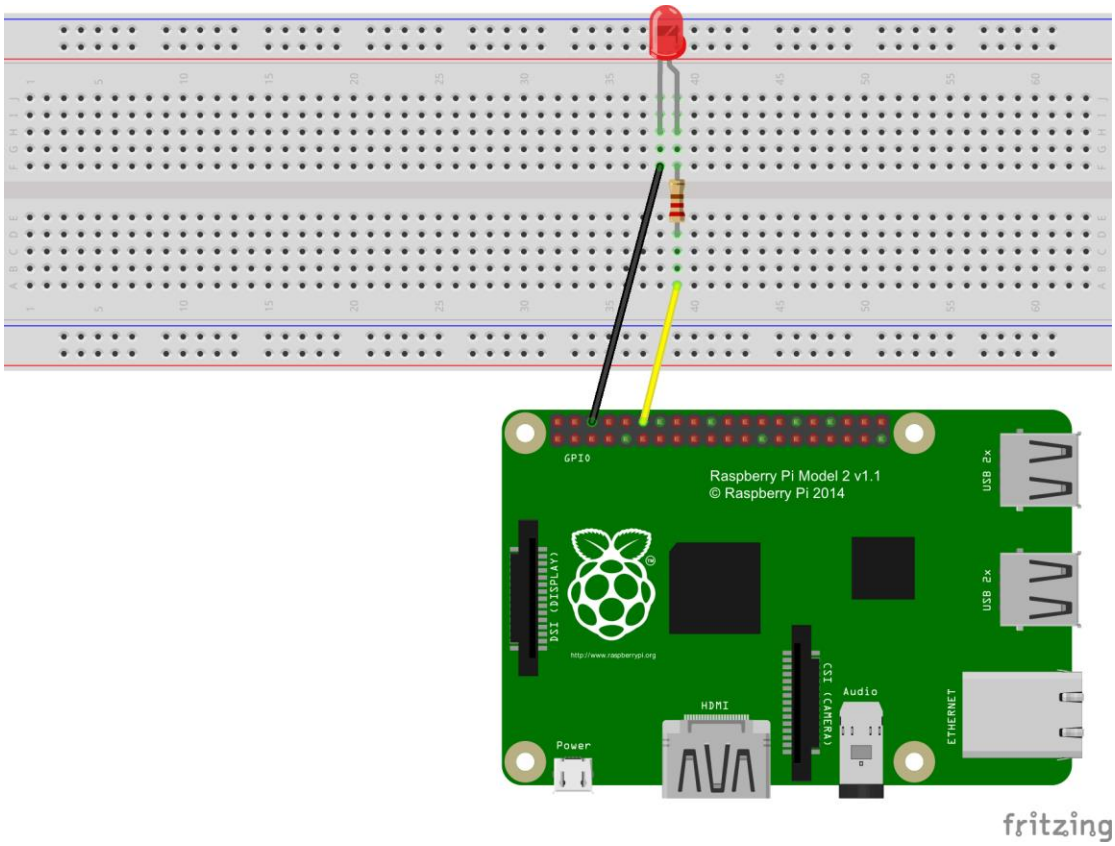
Change the frequency of pwm.

- p.stop( )

Stop the pwm.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/07_breathingLed/breathingLed.c)

2.2 Compile the program

```
$ gcc breathingLed.c -o breathingLed -lwiringPi
```

2.3 Run the program

```
$ sudo ./breathingLed
```

Python user:

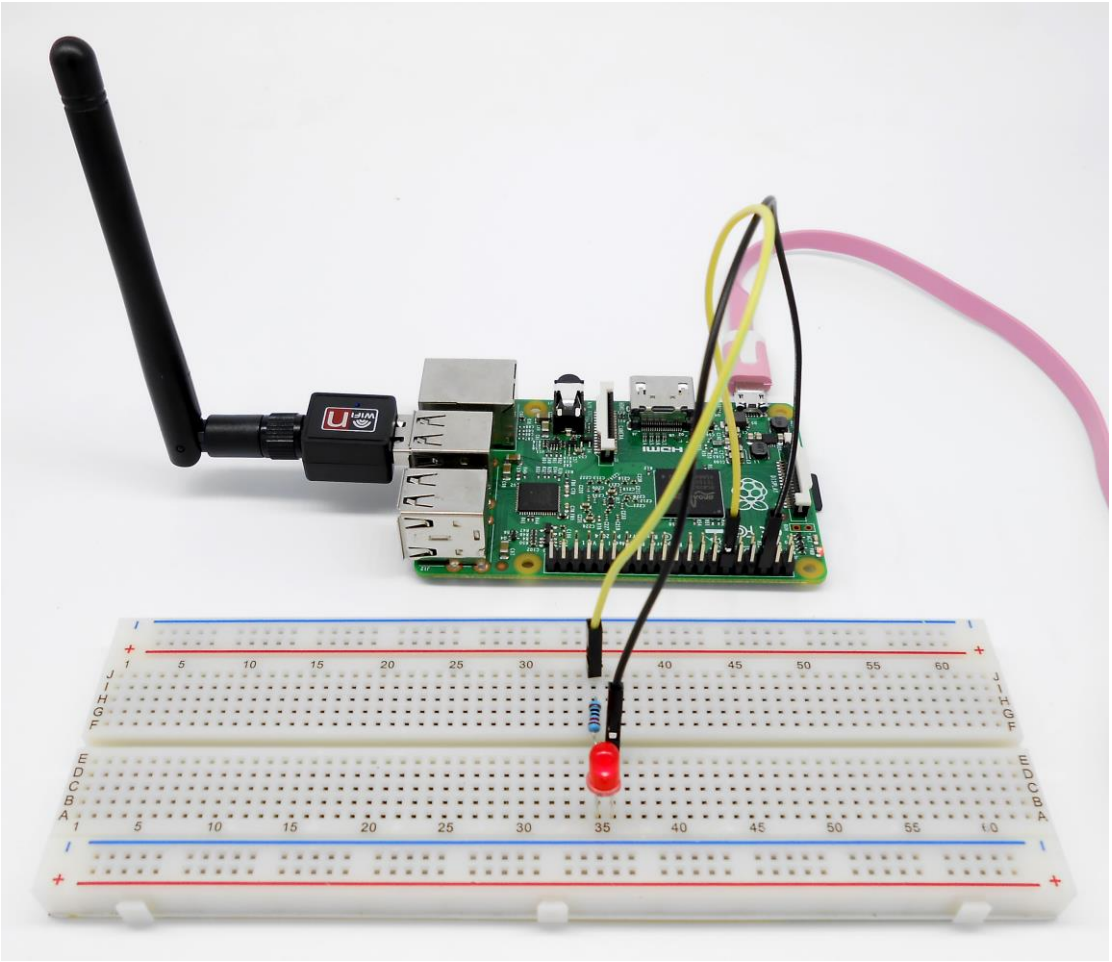
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/07_breathingLed.py)

2.2 Run the program

```
$ sudo python 07_breathingLed.py
```

Now, you should see the LED gradually from dark to brighter, and then from brighter to dark, continuing to repeat the process, its rhythm like the animal's breathing.



Summary

By learning this lesson, I believe that you have understood the basic principles of the PWM, and mastered the PWM programming on the Raspberry Pi platform.

Lesson 8 Controlling a RGB LED with PWM

Overview

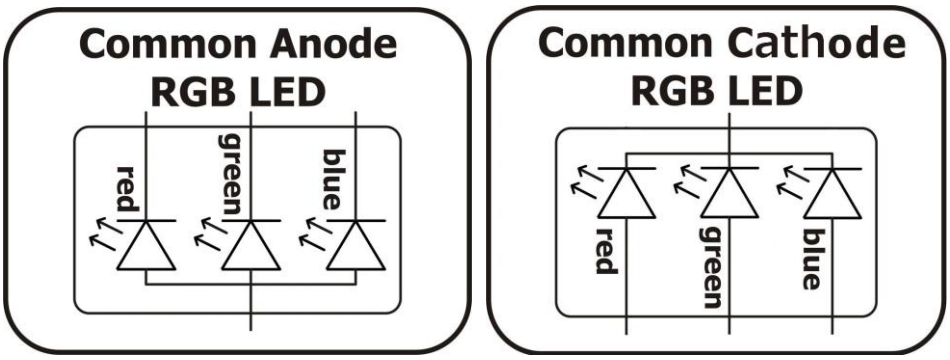
In this lesson, we will program the Raspberry Pi for RGB LED control, and make RGB LED emits a variety of colors of light.

Requirement

- 1* Raspberry Pi
- 1* RGB LED
- 3* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

RGB LEDs consist of three LEDs, one red, one green and one blue. These three colored LEDs are capable of producing any color. Tri-color LEDs with red, green, and blue emitters, in general using a four-wire connection with one common lead (anode or cathode). These LEDs can have either common anode or common cathode leads.



What we used in this experiment is the common anode RGB LED. The longest pin is the common anode of three LEDs. The pin is connected to the +3.3V pin of the Raspberry Pi, and the three remaining pins are connected to the Raspberry Pi's pin11, pin12, pin13 through a current limiting resistor.

In this way, we can control the color of RGB LED by 3-channel PWM signal.

Key function:

- **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.

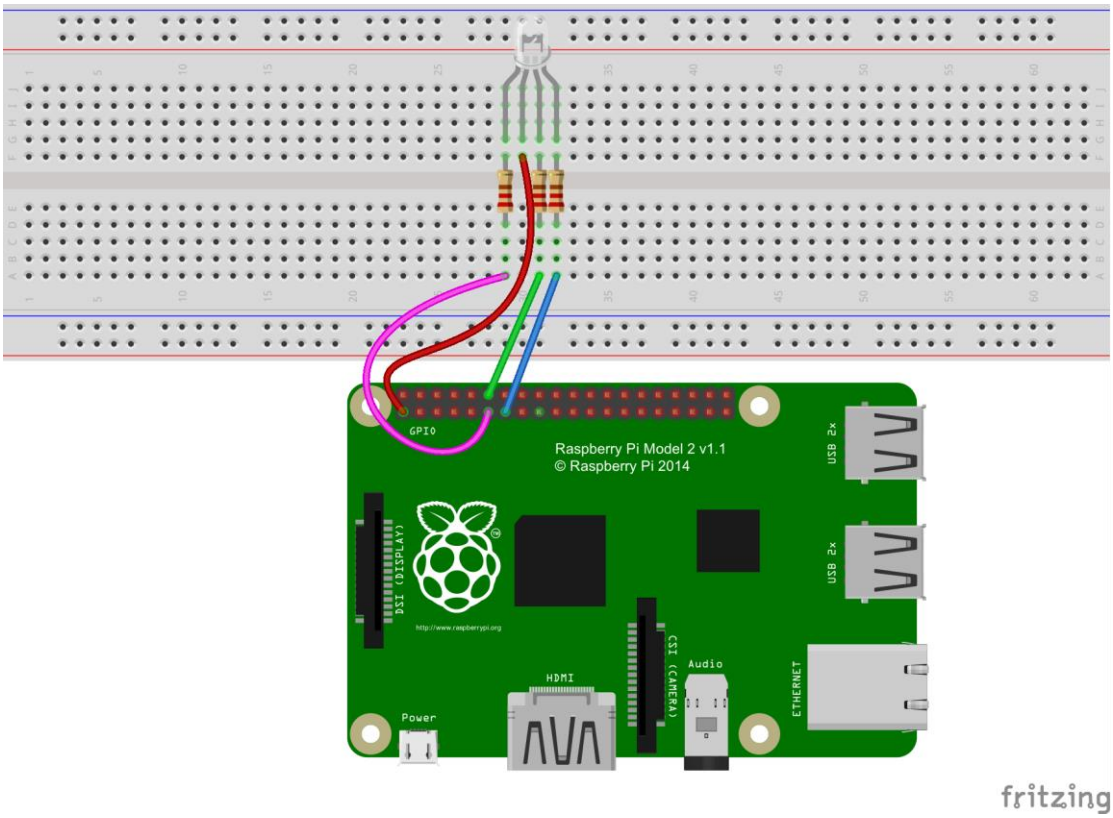
The return value is 0 for success. Anything else and you should check the global errno variable to see what went wrong.

- 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.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/08_rgbLed/rgbLed.c)

2.2 Compile the program

```
$ gcc rgbLed.c -o rgbLed -lwiringPi -lpthread
```

NOTE: The compiler option '-lpthread' is essential, because the implementation of softPwm is based on linux multithreading.

2.3 Run the program

```
$ sudo ./rgbLed
```

Python user:

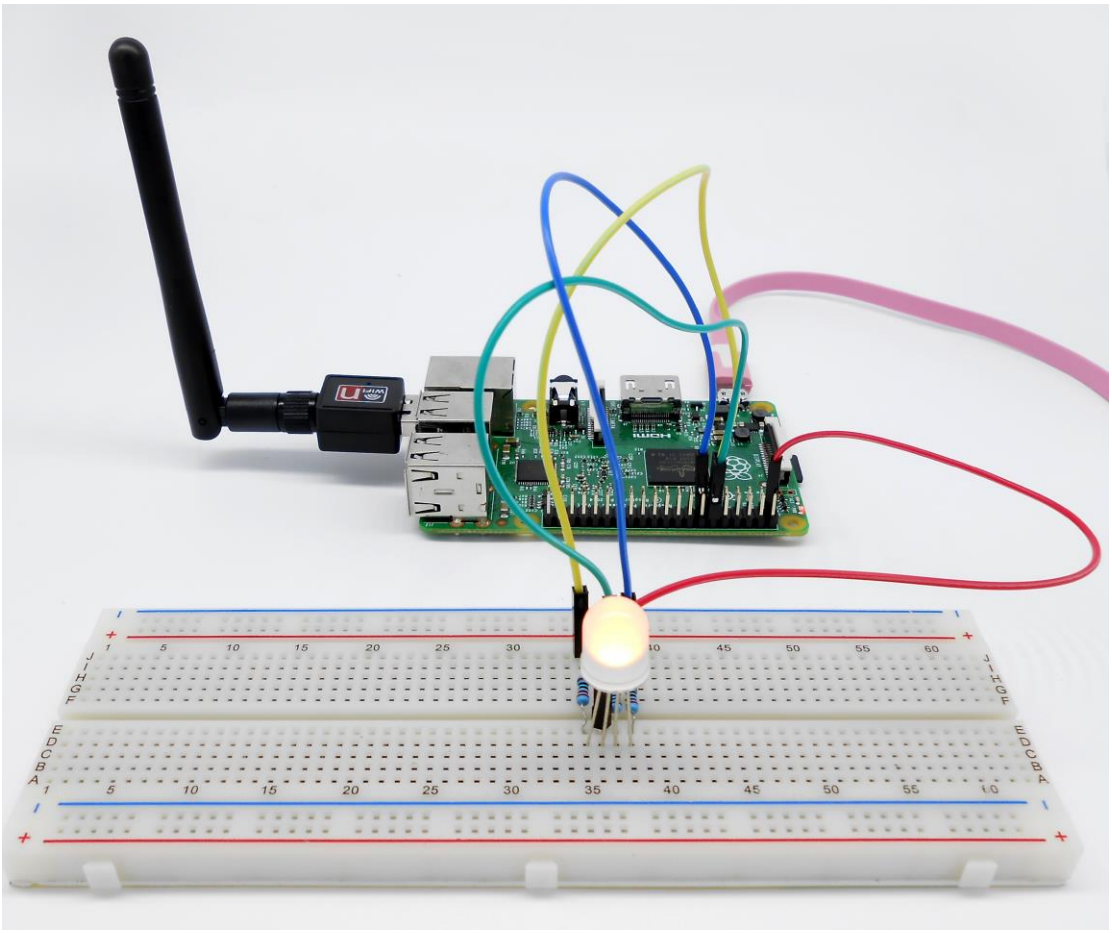
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/08_rgbLed.py)

2.2 Run the program

```
$ sudo python 08_rgbLedLed.py
```

Now, you can see that the RGB LED emitting red, green, blue, yellow, white and purple light, then the RGB LED will be off, each state continues 1s, after repeating the above procedure.



Summary

By learning this lesson, I believe that you have already known the principle and the programming of RGB LED. I hope you can use your imagination to achieve even more cool ideas based on this lesson.

Lesson 9 7-segment display

Overview

In this lesson, we will program the Raspberry Pi to achieve the controlling of segment display.

Requirement

- 1* Raspberry Pi
- 1* 220Ω Resistor
- 1* 7-Segment display
- 1* Breadboard
- Several Jumper wires

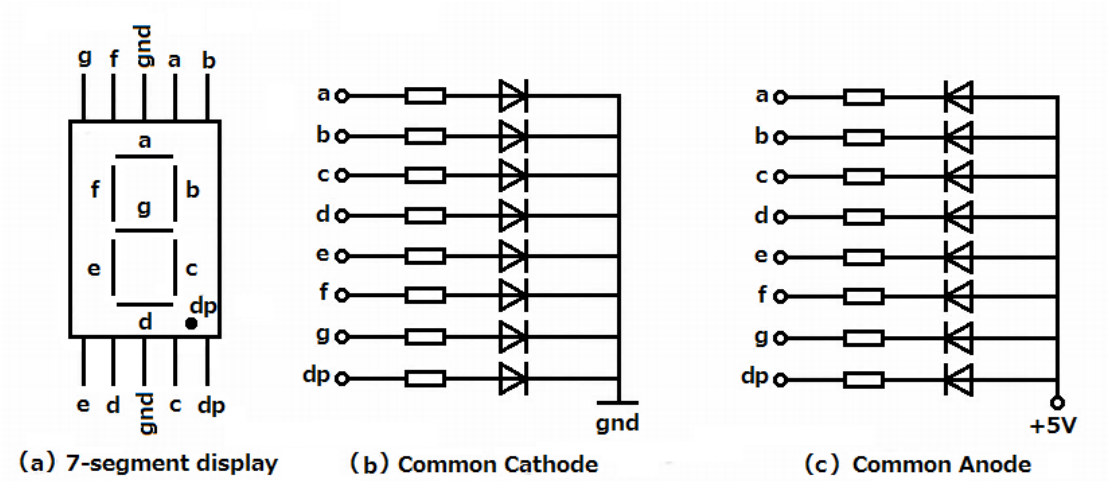
Principle

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.

Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

The seven-segment display is an 8-shaped LED display device composed of eight LEDs (including a decimal point), these segments respectively named a, b, c, d, e, f, g, dp.

The segment display can be divided into common anode and common cathode segment display by internal connections.

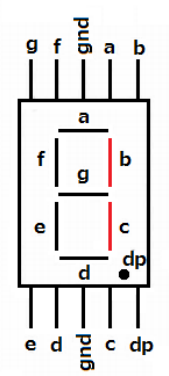


(a) 7-segment display (b) Common Cathode (c) Common Anode

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.



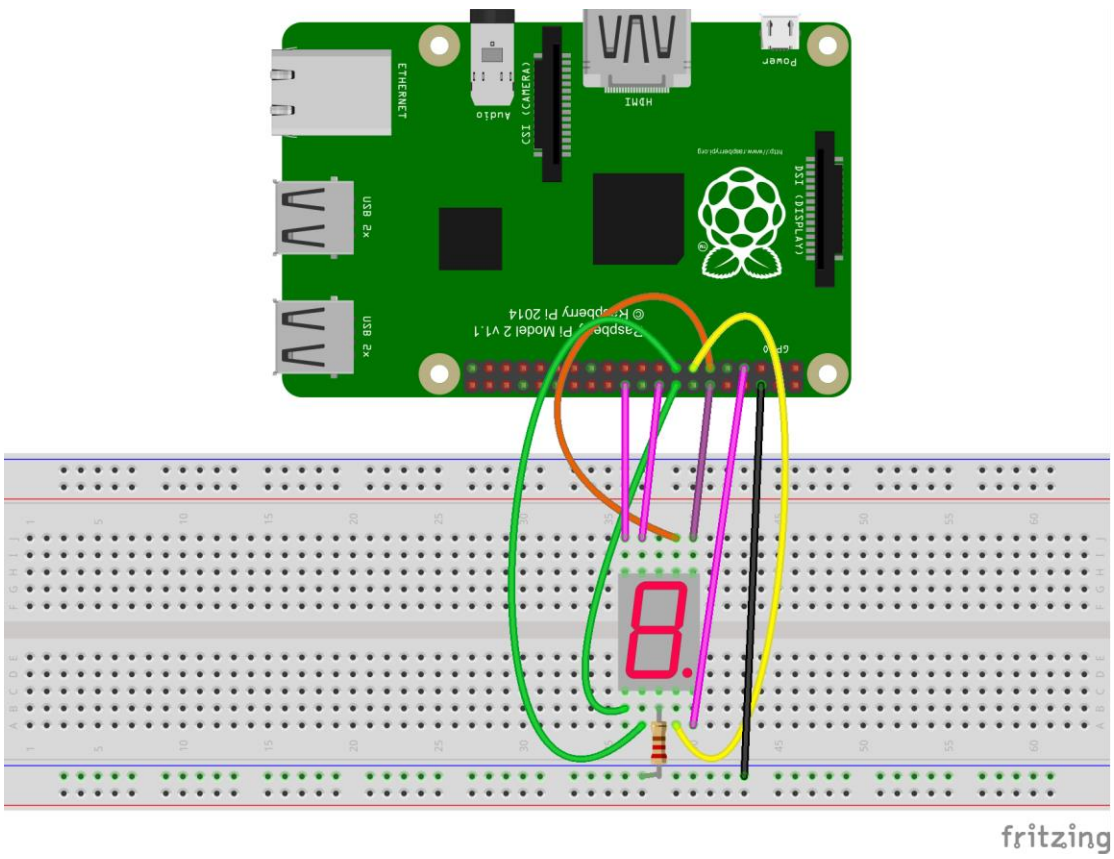
Key function:

- **void digitalWriteByte (int value)**

This writes the 8-bit byte supplied to the first 8 GPIO pins. It's the fastest way to set all 8 bits at once to a particular value, although it still takes two write operations to the Pi's GPIO hardware.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/09_segment/segment.c)

2.2 Compile the program

```
$ gcc segment.c -o segment -lwiringPi
```

2.3 Run the program

```
$ sudo ./segment
```

Python user:

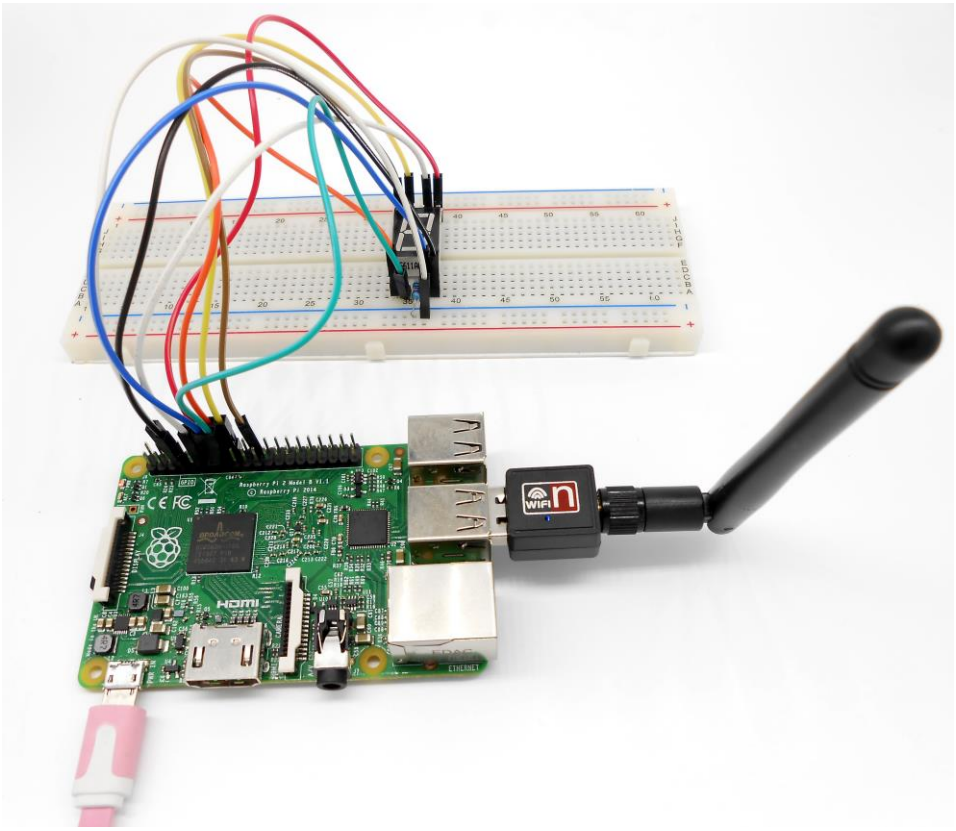
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/09_segment.py)

2.2 Run the program

```
$ sudo python 09_segment.py
```

Now, you should see the number 0~9, a~F are displayed on the segment display.



Summary

Through this lesson, we have learned the principle and programming of segment display. I hope you can combine the former course to modify the code we provided in this lesson to achieve cooler originality.

Lesson 10 4-digit 7-segment display

Overview

In this lesson, we will program the Raspberry Pi to achieve the controlling of 4-digit 7-segment display.

Requirement

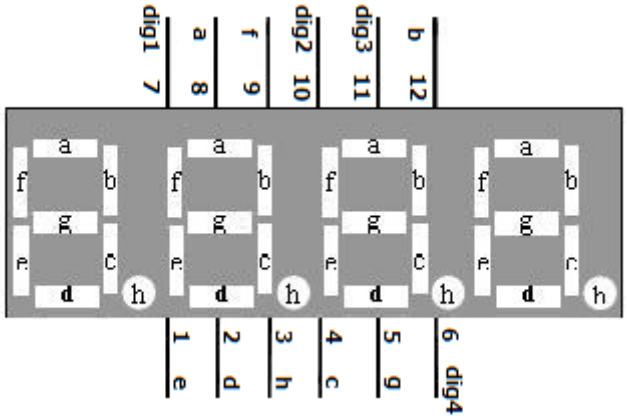
- 1* Raspberry Pi
- 4* 220Ω Resistor
- 1* 4-digit 7-segment display
- 1* Breadboard
- Several Jumper wires

Principle

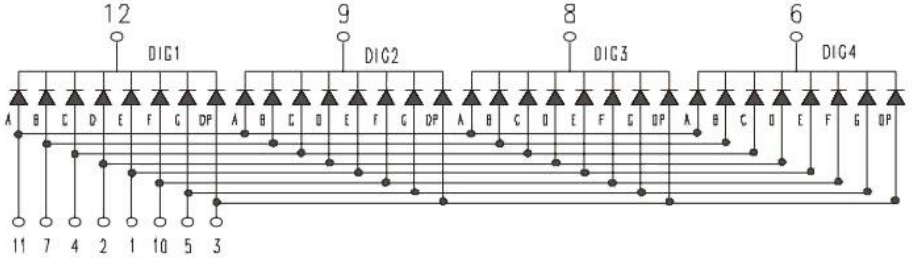
The four-digit segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

Four-digit segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

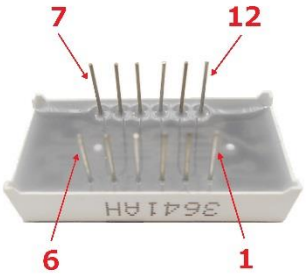
The four-digit segment display is an 4*8-shaped LED display device composed of 32 LEDs (including four decimal points), these segments respectively named a, b, c, d, e, f, g, h, dig1, dig2, dig3, dig4.



What we used in this experiment is a common cathode 4-digit 7-segment display. Its internal structure is shown below:

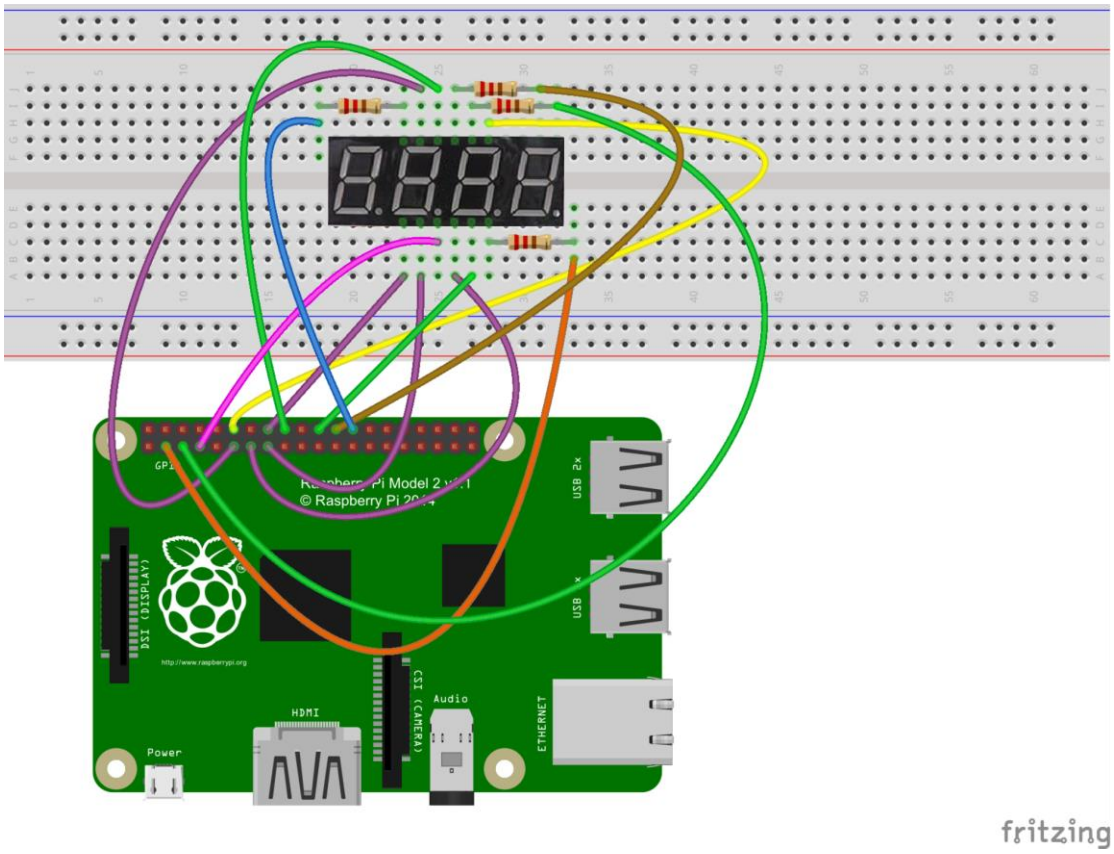


The pin number is shown below:



Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/10_4bitSegment/fourBitSegment.c)

2.2 Compile the program

```
$ gcc fourBitSegment.c -o fourBitSegment -lwiringPi
```

2.3 Run the program

```
$ sudo ./fourBitSegment
```

Python user:

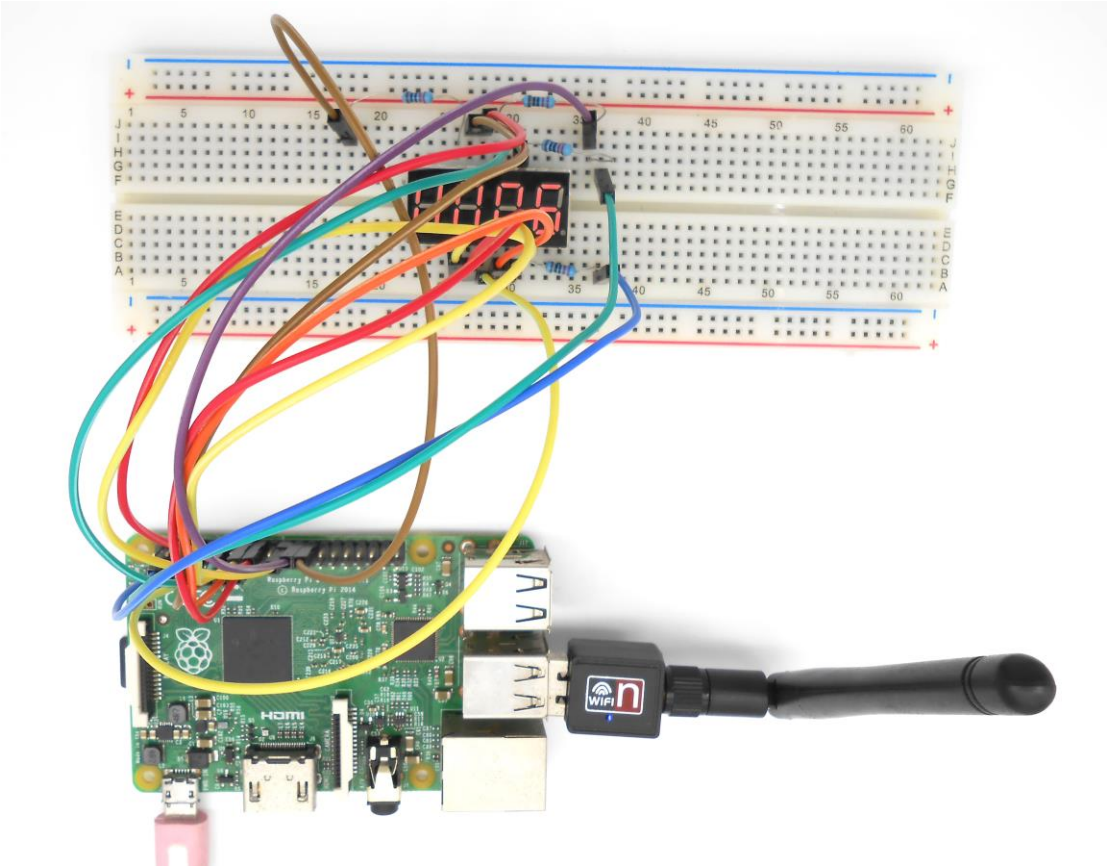
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/10_fourBitSegment.py)

2.2 Run the program

```
$ sudo python 10_fourBitSegment.py
```

Now, you should see the number is displayed on the segment display.



Lesson 11 LCD1602

Overview

In this lesson, we will learn how to use a character display device—LCD1602 on the Raspberry Pi platform. First, we make the LCD1602 display a string "Hello Geeks!" scrolling, then display “Adeept” and “www.adeept.com” static.

Requirement

- 1* Raspberry Pi
- 1* LCD1602
- 1* 10KΩ Potentiometer
- 1* Breadboard
- Several Jumper wires

Principle

LCD1602 is a kind of character LCD display. The LCD has a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

- A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.
- A Read/Write (R/W) pin that selects reading mode or writing mode
- An Enable pin that enables writing to the registers
- 8 data pins (D0-D7). The status of these pins (high or low) are the bits that you're writing to a register when you write, or the values when you read.
- There's also a display contrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on or off the LED backlight respectively.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The wiringPiDev Library simplifies this for you, so you don't need to know the low-level instructions.

The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires six I/O pins from the Raspberry Pi, while the 8-bit mode requires 10 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

A potentiometer, informally a pot, is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.

Key function:

- int lcdInit (int rows, int cols, int bits, int rs, int strb, int d0, int d1, int d2, int d3, int d4, int d5, int d6, int d7)

This is the main initialisation function and must be called before you use any other LCD functions.

Rows and cols are the rows and columns on the display (e.g. 2, 16 or 4,20). Bits is the number of bits wide on the interface (4 or 8). The rs and strb represent the pin numbers of the displays RS pin and Strobe (E) pin. The parameters d0 through d7 are the pin numbers of the 8 data pins connected from the Pi to the display. Only the first 4 are used if you are running the display in 4-bit mode.

The return value is the 'handle' to be used for all subsequent calls to the lcd library when dealing with that LCD, or -1 to indicate a fault. (Usually incorrect parameters)

- lcdPosition (int handle, int x, int y)

Set the position of the cursor for subsequent text entry. x is the column and 0 is the left-most edge. y is the line and 0 is the top line.

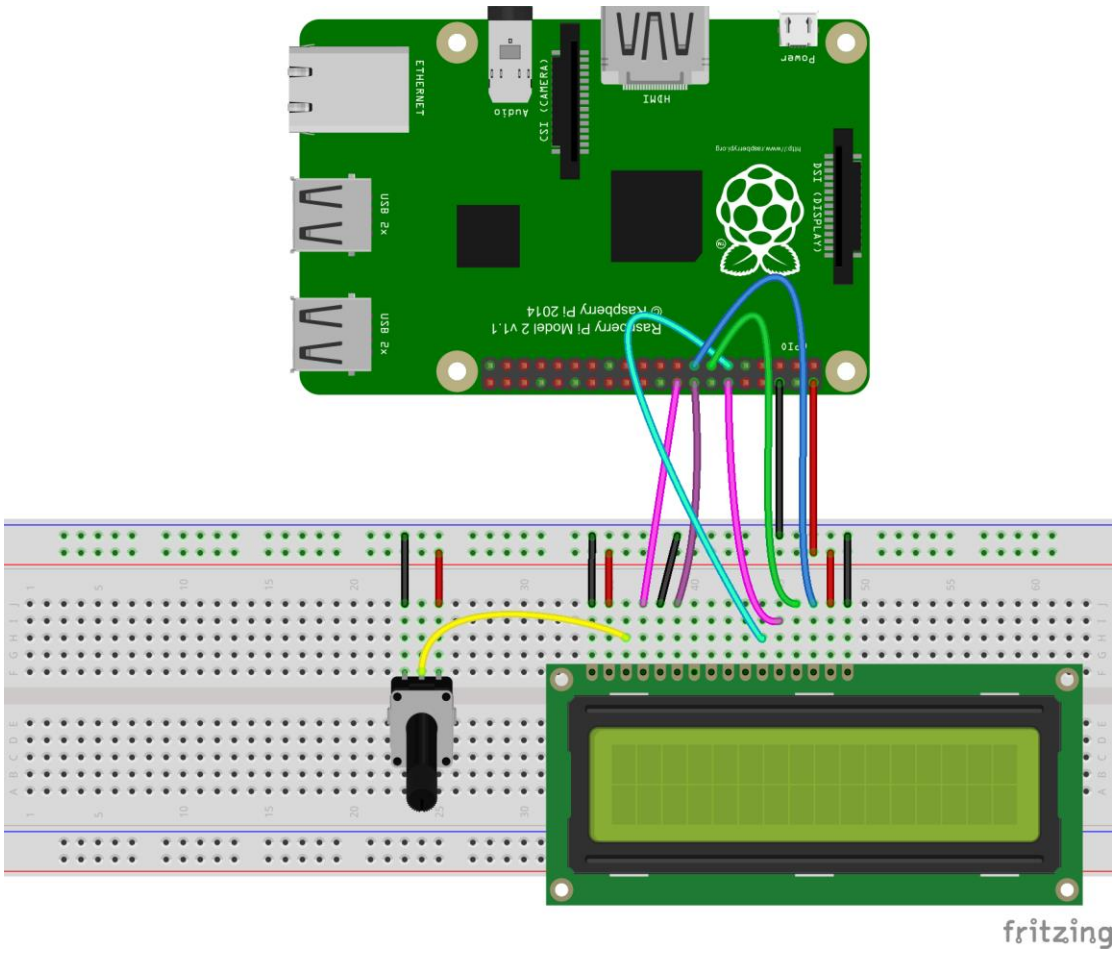
- lcdPuts (int handle, const char *string)
- lcdPrintf (int handle, const char *message, ...)
- lcdPutchar (int handle, unsigned char data)

These output a single ASCII character, a string or a formatted string using the usual printf formatting commands.

At the moment, there is no clever scrolling of the screen, but long lines will wrap to the next line, if necessary.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/11_lcd1602/lcd1602_2.c)

2.2 Compile the program

```
$ gcc lcd1602_2.c -o lcd1602_2 -lwiringPi -lwiringPiDev
```

2.3 Run the program

```
$ sudo ./lcd1602_2
```

Python user:

2.1 Edit and save the code with vim or nano.

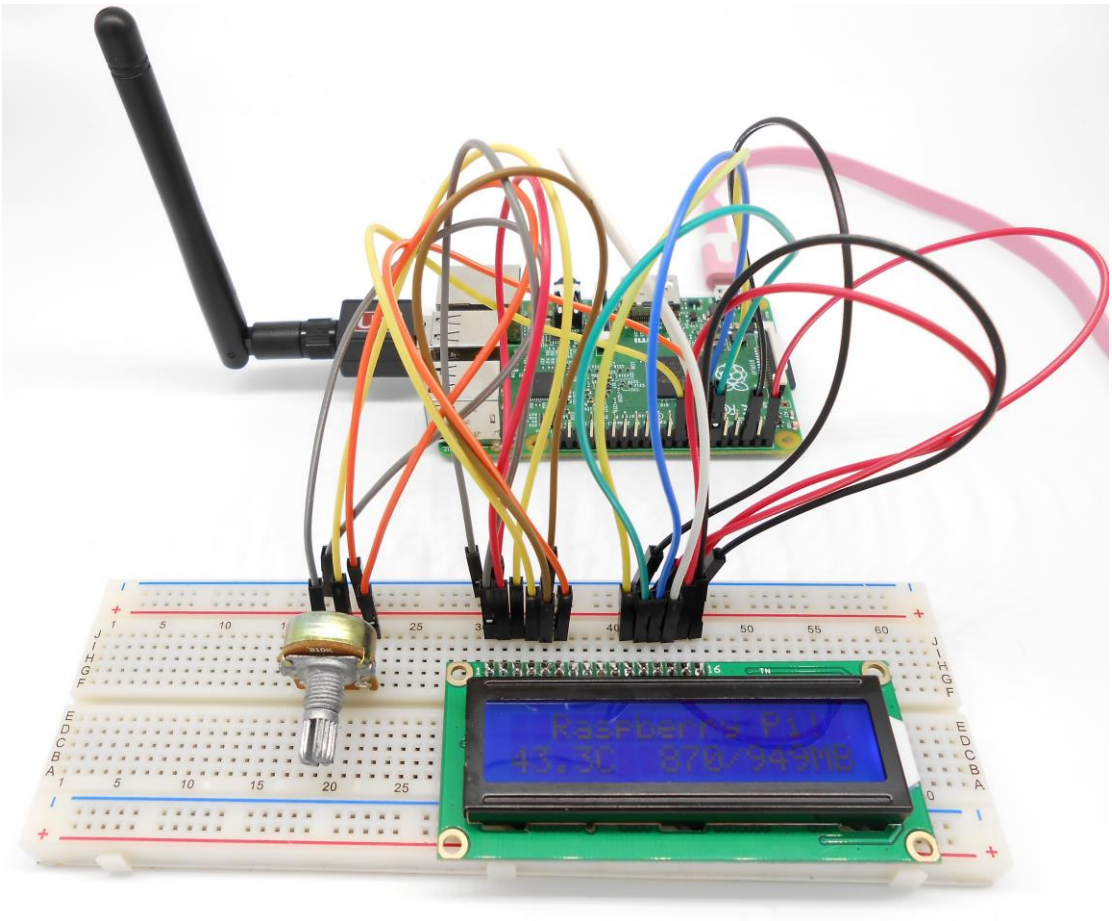
(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/11_lcd1602.py)

2.2 Run the program

```
$ sudo python 11_lcd1602.py
```

Now, you can see the string "Hello Geeks!" is shown on the LCD1602 scrolling, and

then the string "Adeept" and "www.adeept.com" is displayed on the LCD1602 static.



Summary

I believe that you have already mastered the driver of LCD1602 through this lesson. I hope you can make something more interesting base on this lesson and the previous lesson learned.

Lesson 12 A Simple Voltmeter

Overview

In this lesson, we will make a simple voltmeter with Raspberry Pi and LCD1602, the range of this voltmeter is 0~5V. Then, we will measure the voltage of the potentiometer's adjustment end with the simple voltmeter and display it on the LCD1602.

Requirement

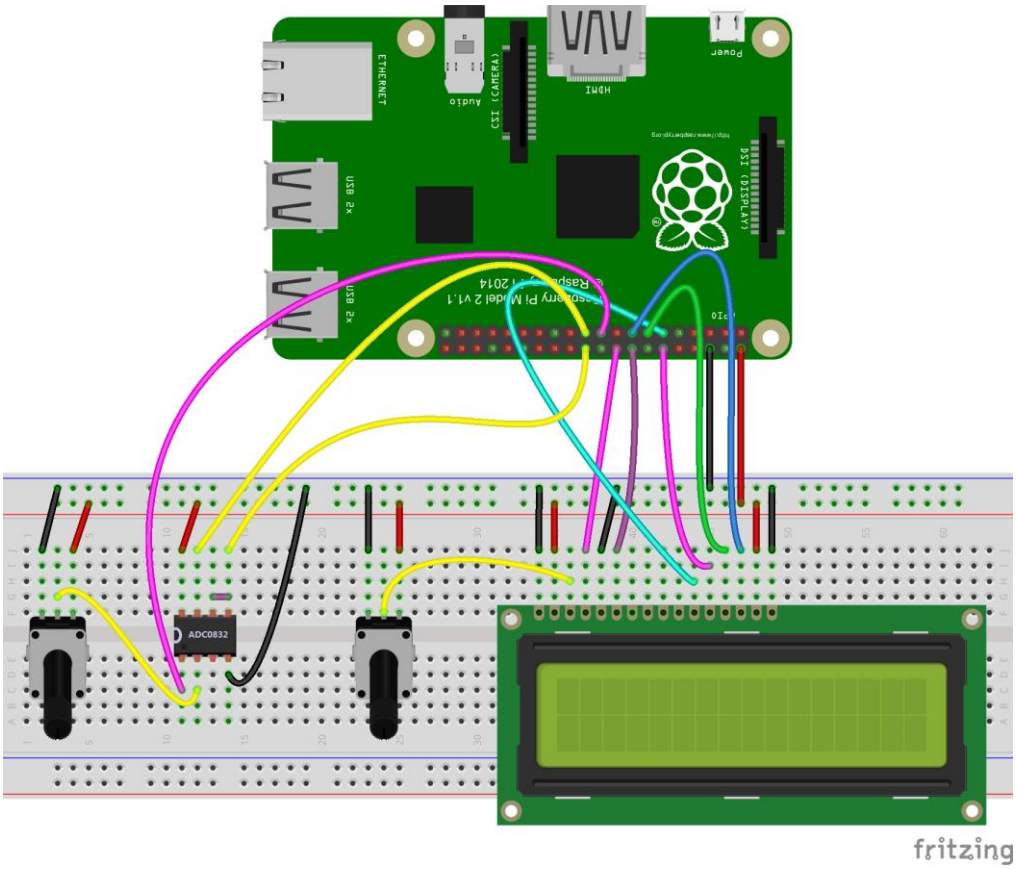
- 1* Raspberry Pi
- 1* LCD1602
- 2* Potentiometer
- 1* Breadboard
- Several Jumper wires

Principle

The basic principle of this experiment: Converting the analog voltage that the ADC0832 collected to digital quantity through programming, then display the voltage on the LCD1602.

Procedures

1. Build the circuit



2. Program

2.1 Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/12_voltage/voltmeter.c)

2.2 Compile the program

```
$ gcc voltmeter.c -o voltmeter -lwiringPi
```

2.3 Run the program

```
$ sudo ./voltmeter
```

Python user:

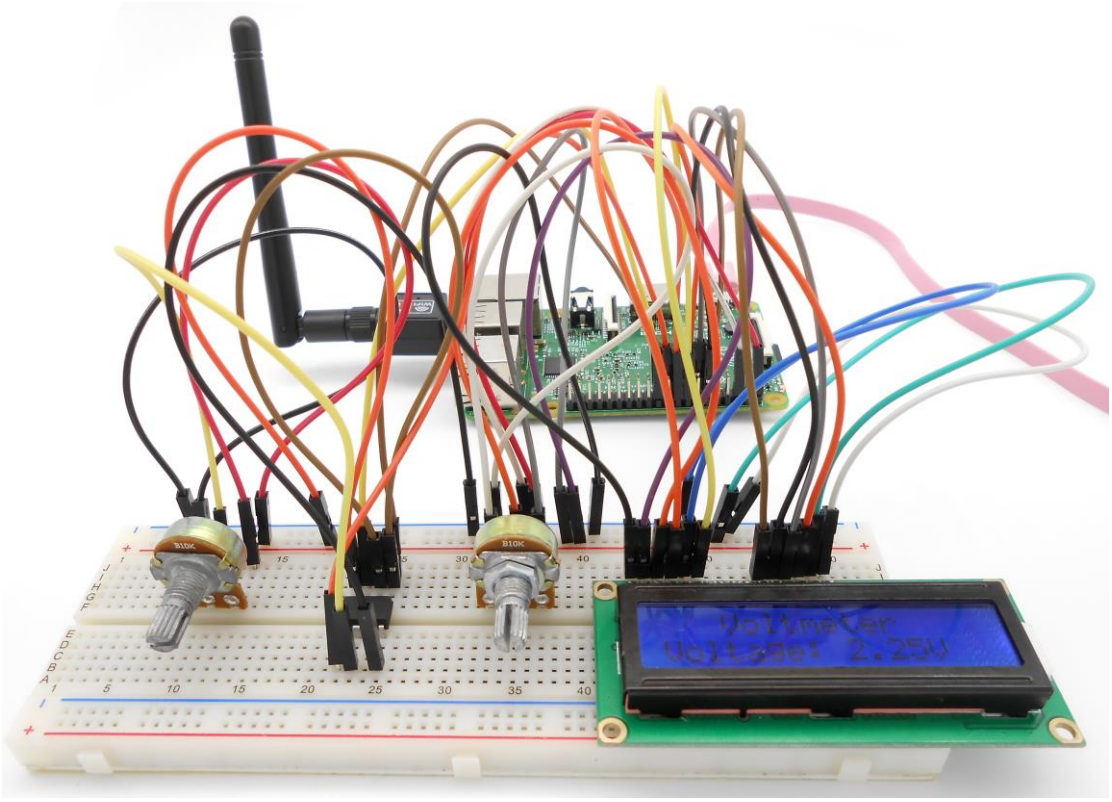
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/12_voltmeter.py)

2.2 Run the program

```
$ sudo python 12_voltmeter.py
```

Now, when you turning the shaft of the potentiometer, you will see the voltage displayed on the LCD1602 will be changed.



Summary

The substance of voltmeter is reading analog voltage which input to ADC. Through this course, I believe that you have mastered how to read analog value and how to make a simple voltmeter with Raspberry Pi.

Lesson 13 Matrix Keyboard

Overview

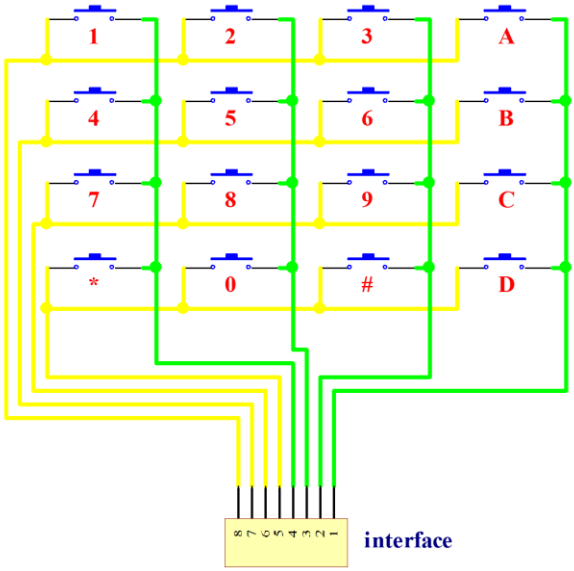
In this lesson, we will learn how to use a matrix keyboard.

Requirement

- 1* Raspberry Pi
- 1* 4x4 Matrix Keyboard
- Several Jumper wires

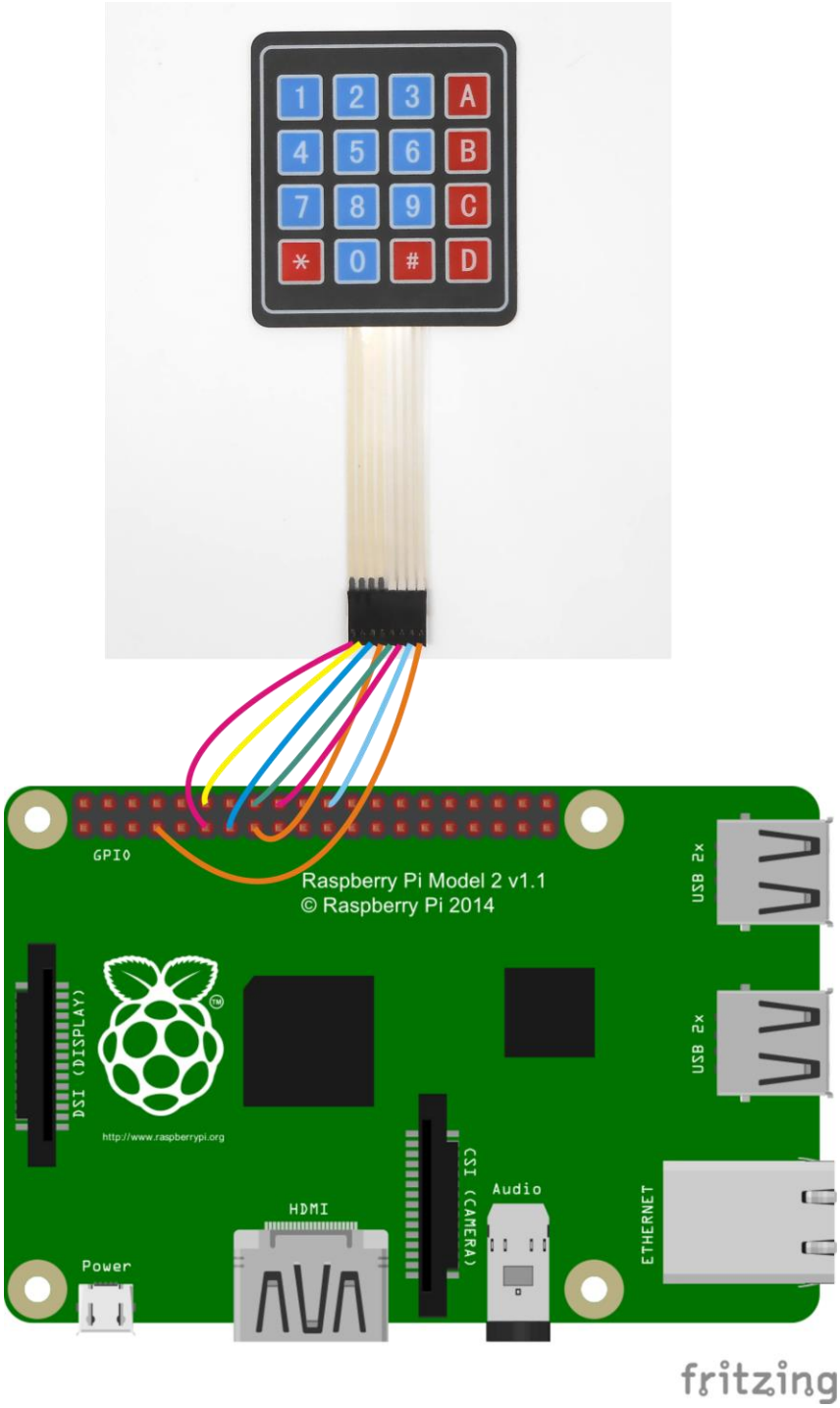
Principle

In order to save the resources of the microcontroller port, we usually connect the buttons in a matrix in an actual project.
The following is the schematics of 4x4 matrix keyboard:



Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/13_matrixKeyboard/matrixKeyboard.c)

2.2 Compile the program

```
$ gcc matrixKeyboard.c -o matrixKeyboard -lwiringPi
```

2.3 Run the program

```
$ sudo ./matrixKeyboard
```

Python user:

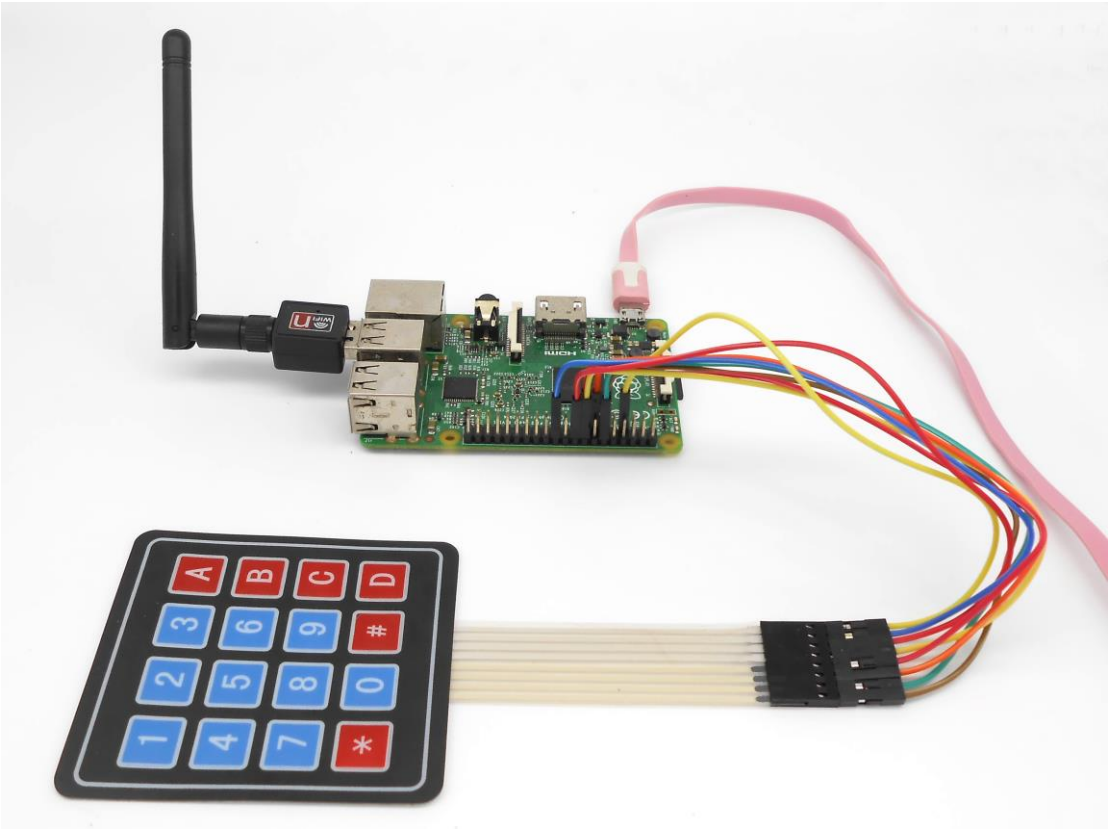
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/13_matrixKeyboard.py)

2.2 Run the program

```
$ sudo python 13_matrixKeyboard.py
```

Now, when you press one of the button on the 4x4 matrix keyboard, you will see the corresponding key value will be displayed on the terminal.



Lesson 14 Measure the distance

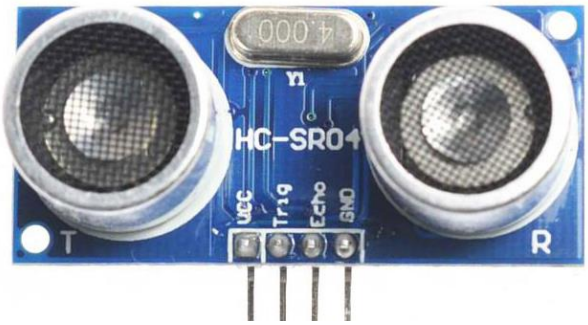
Overview

In this lesson, we will learn how to measure the distance by the ultrasonic distance sensor.

Requirement

- 1* Raspberry Pi
- 1* Ultrasonic distance sensor
- 1* Breadboard
- Several Jumper wires

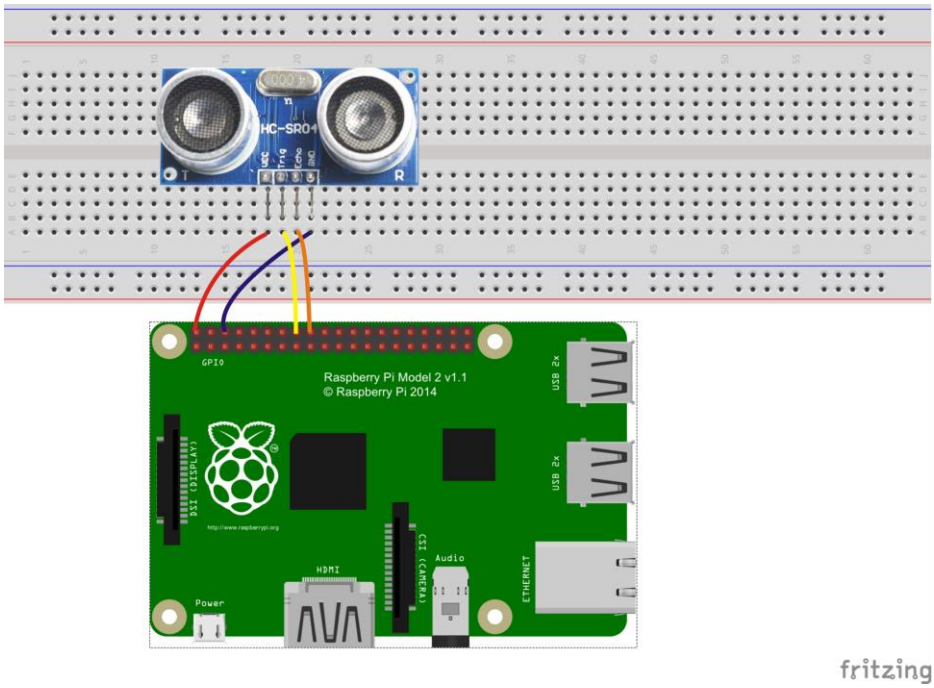
Principle



This recipe uses the popular Parallax PING ultrasonic distance sensor to measure the distance of an object ranging from 2 cm to around 3 m. This sensor works by sending a sound wave out and calculating the time it takes for the sound wave to get back to the ultrasonic sensor. By doing this, it can tell us how far away an obstacle is relative to the ultrasonic.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/14_ultrasonicSensor/distance.c)

2.2 Compile the program

```
$ gcc distance.c -o distance -lwiringPi
```

2.3 Run the program

```
$ sudo ./distance
```

Python user:

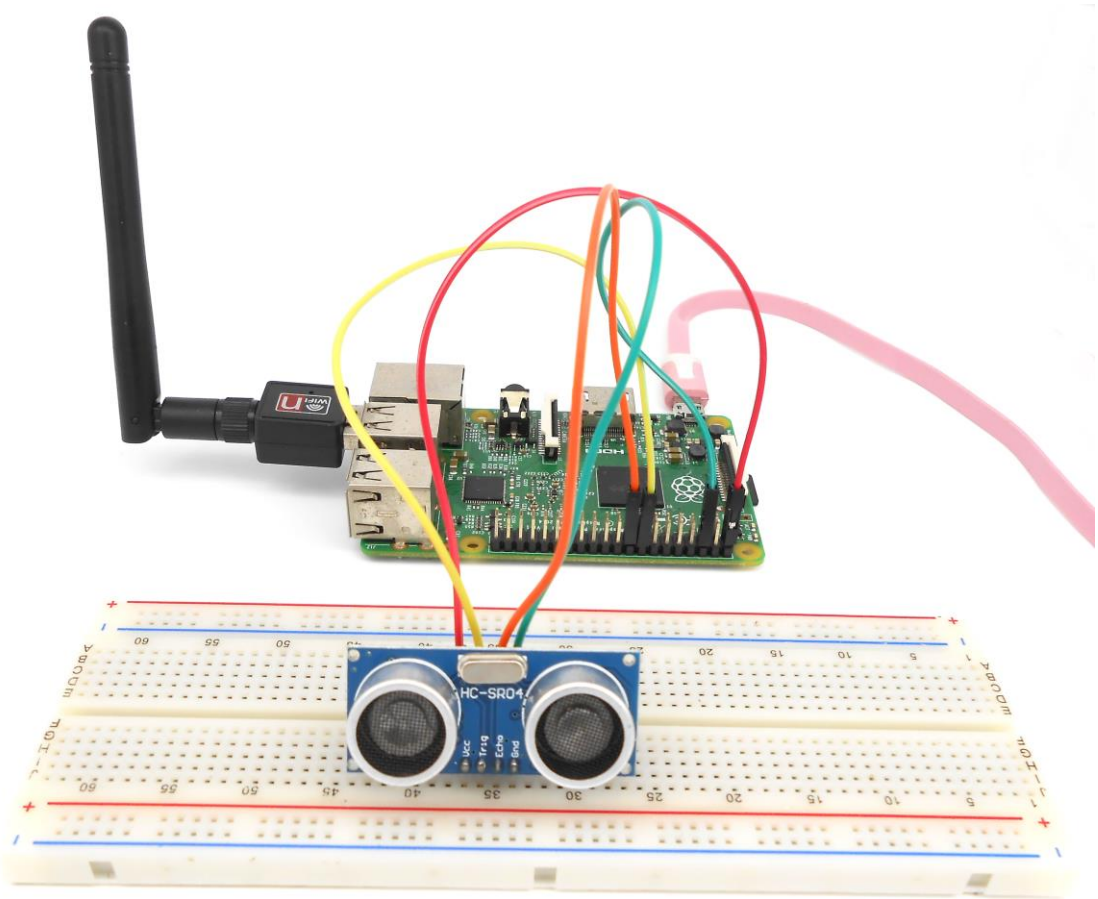
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/14_distance.py)

2.2 Run the program

```
$ sudo python 14_distance.py
```

Now, you will see the distance between the obstacle and the ultrasonic sensor display on the screen.


HC-SR04
Vcc
Trig
Echo
Gnd

Lesson 15 Temperature & Humidity Sensor—DHT-11

Overview

In this lesson, we will learn how to use DHT-11 to collect temperature and humidity data.

Requirement

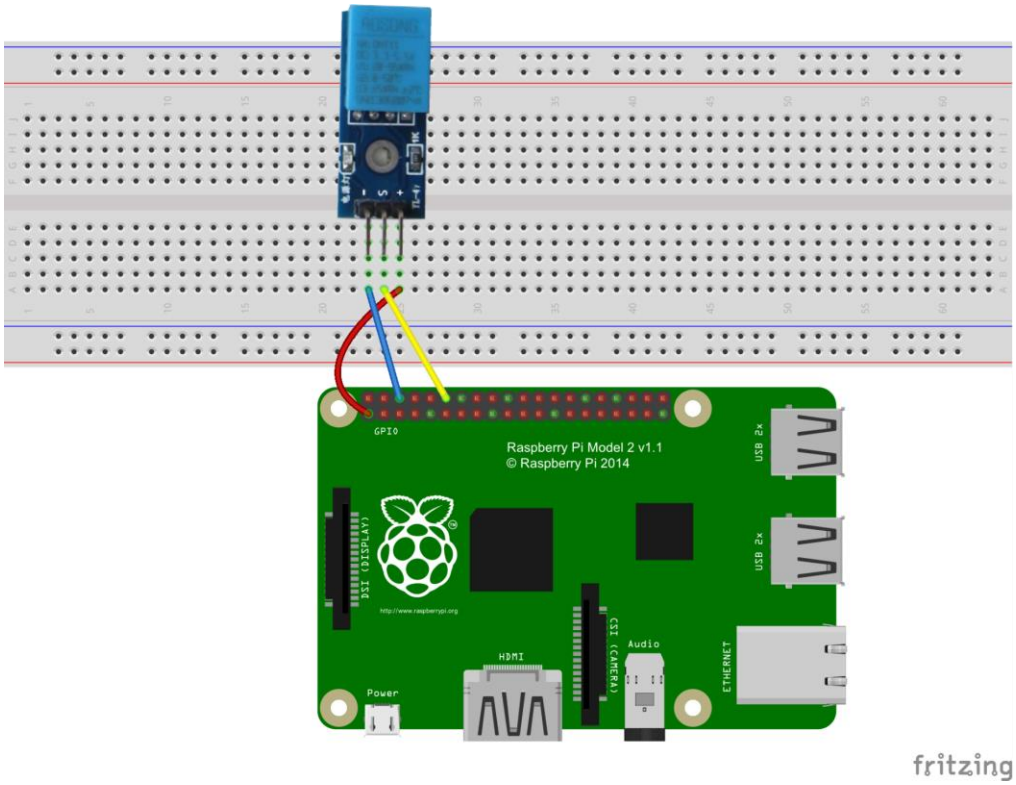
- 1* Raspberry Pi
- 1* DHT-11
- 1* Breadboard
- Several Jumper wires

Principle

This DHT-11 temperature & humidity sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital-signal-acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/15_DHT11/dht11.c)

2.2 Compile the program

```
$ gcc dht11.c -o dht11 -lwiringPi
```

2.3 Run the program

```
$ sudo ./dht11
```

Python user:

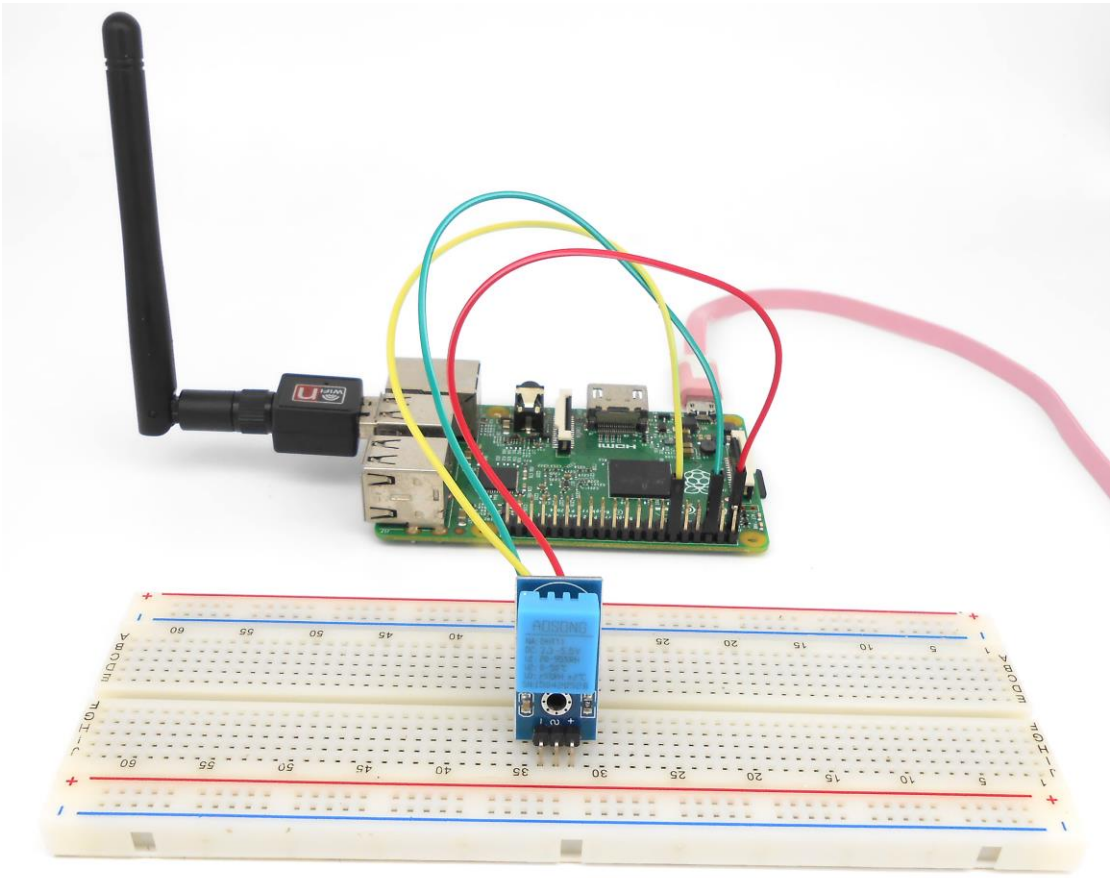
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/15_dht11.py)

2.2 Run the program

```
$ sudo python 15_dht11.py
```

Now, you can see the temperature and humidity data displayed on the terminal.



Lesson 16 Dot-matrix display

Overview

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

Requirement

- 1* Raspberry Pi
- 1* 8*8 Dot-matrix display
- 2* 74HC595
- 1* Breadboard
- Several Jumper wires

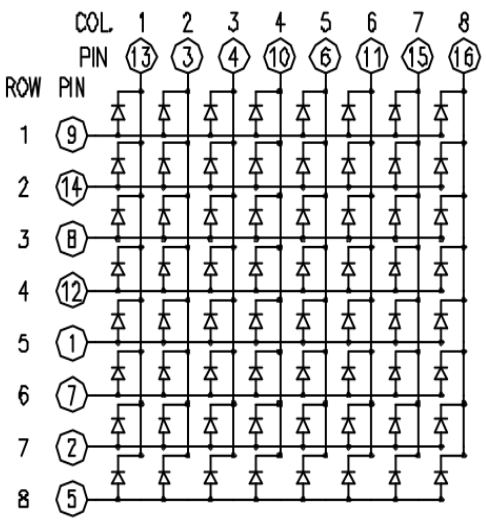
Principle

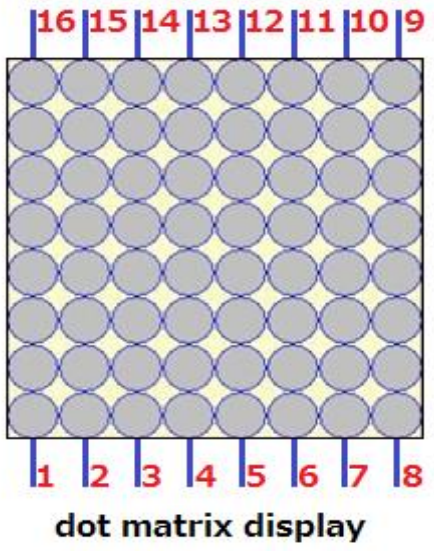
1. Dot-matrix display

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:





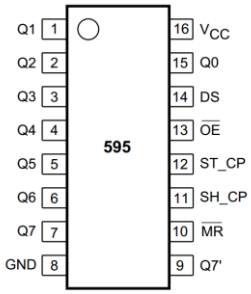
A 8*8 dot-matrix display consists of 64 LEDs, and each LED is placed at the intersection of the lines and columns. When the corresponding row is set as high level and the column is set as low level, then the LED will be lit.

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 Pi's 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 flowing is the function of each pin:

DS: Serial data input

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

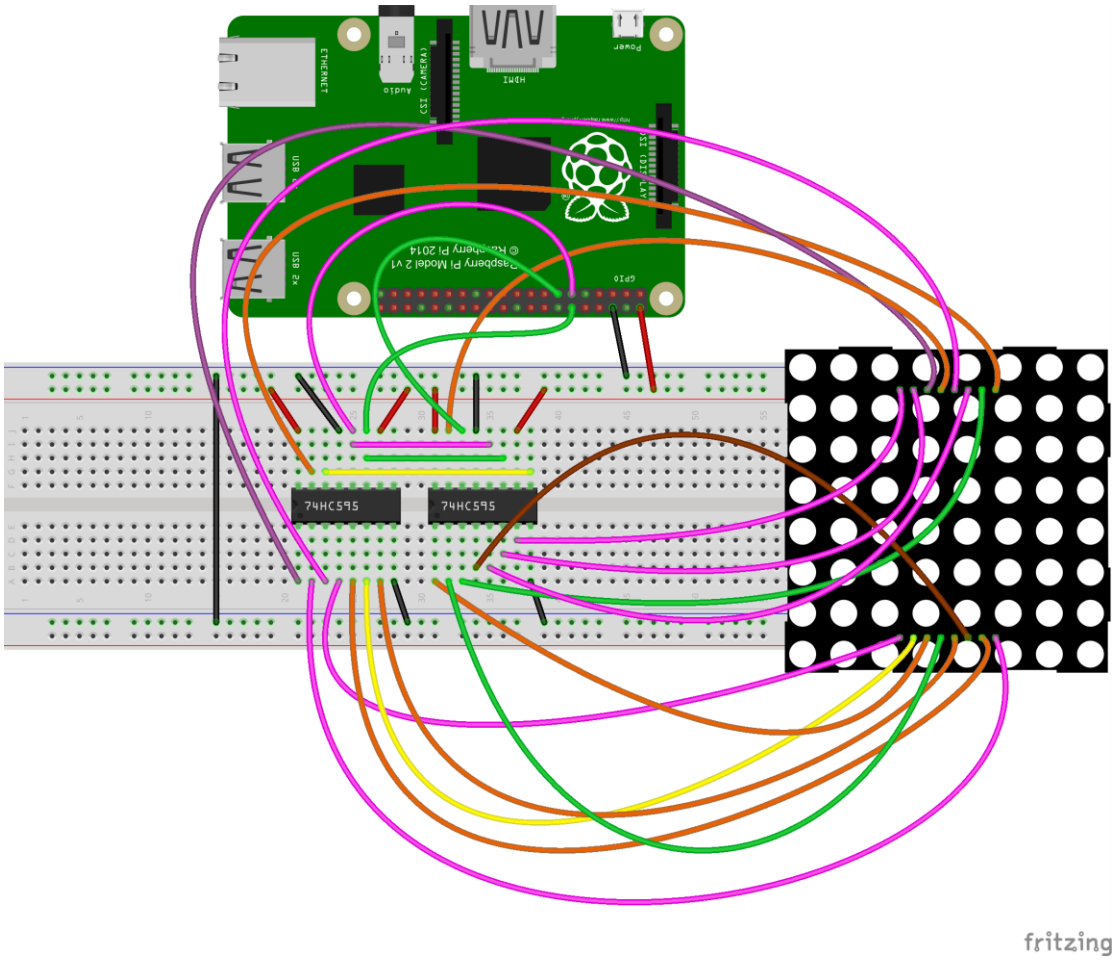
MR: Reset pin, effective at low level, directly connected to 5V high level in practical applications

SH_CP: Shift register clock input

ST_CP: storage register clock input

Procedures

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



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/16_ledMatrix/ledMatrix.c)

2.2 Compile the program

```
$ gcc ledMatrix.c -o ledMatrix -lwiringPi
```

2.3 Run the program

```
$ sudo ./ledMatrix
```

Python user:

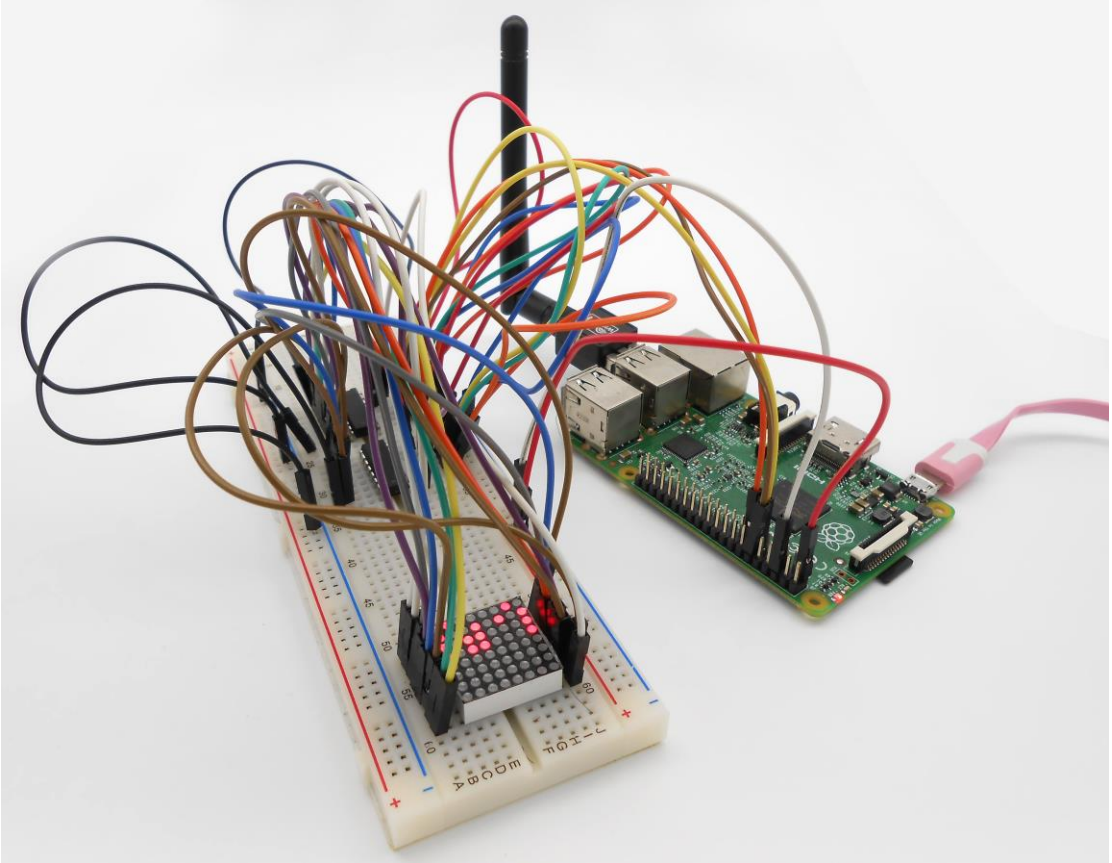
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/16_ledMatrix.py)

2.2 Run the program

```
$ sudo python 16_ledMatrix.py
```

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



Summary

In this experiment, we have not only learned how to operate a dot-matrix display to display numbers and letters, but also learned the basic usage of 74HC595, then you can try operating the dot-matrix display to show other images.

Lesson 17 Photoresistor

Overview

In this lesson, we will learn how to measure the light intensity by photoresistor and make the measurement result displayed in the screen.

Requirement

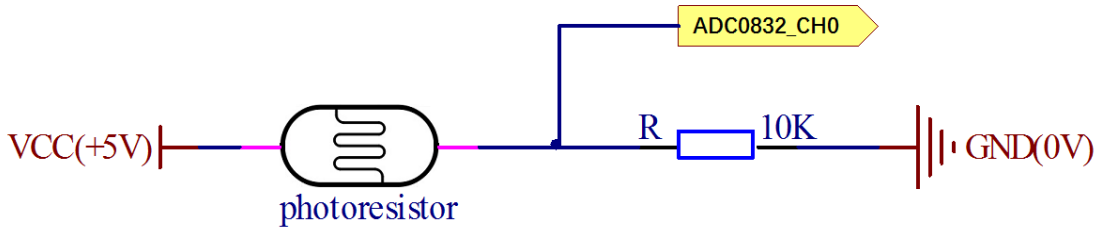
- 1* Raspberry Pi
- 1* ADC0832
- 1* Photoresistor
- 1* 10KΩ Resistor
- 1* Breadboard
- Several Jumper wires

Principle

A photoresistor is a light-controlled variable resistor. The resistance of a photoresistor decreases with the 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 (MΩ), 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.

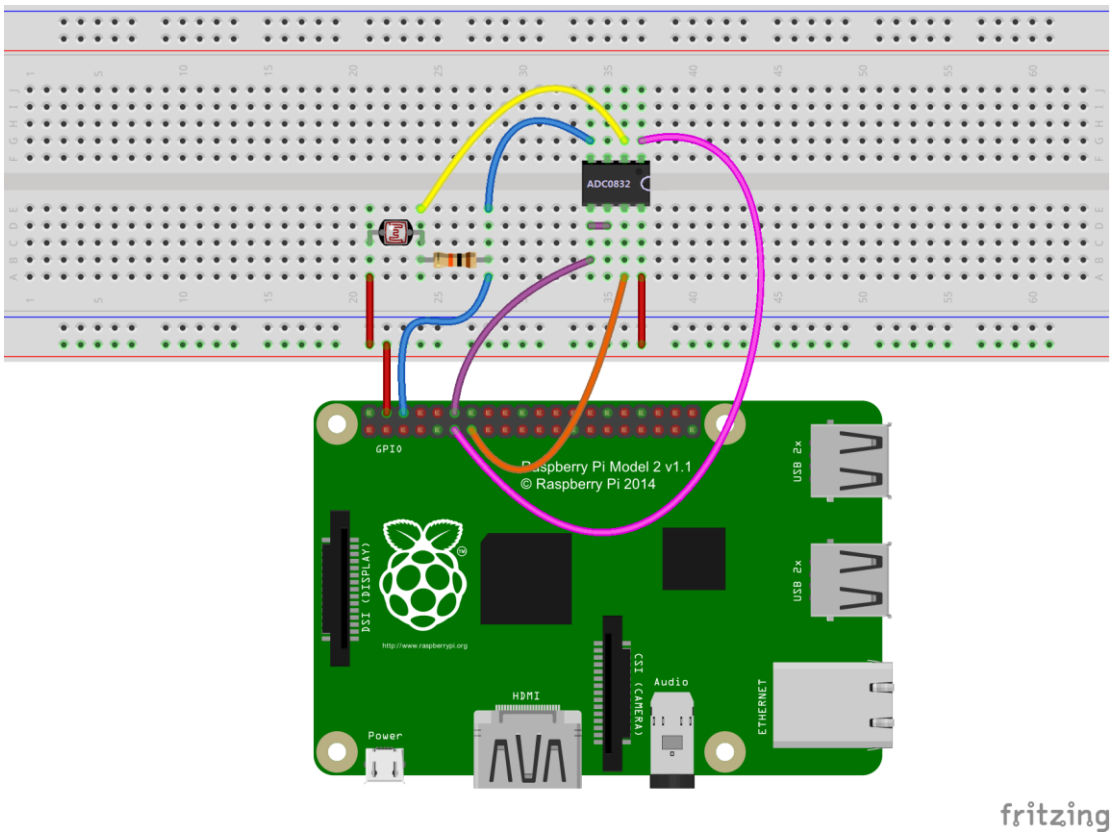
The schematic diagram of this experiment is shown below:



With the increase of the light intensity, the resistance of photoresistor will be decreased. The voltage of GPIO port in the above figure will become high.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/17_photoresistor/photoresistor.c)

2.2 Compile the program

```
$ gcc photoresistor.c -o photoresistor -lwiringPi
```

2.3 Run the program

```
$ sudo ./photoresistor
```

Python user:

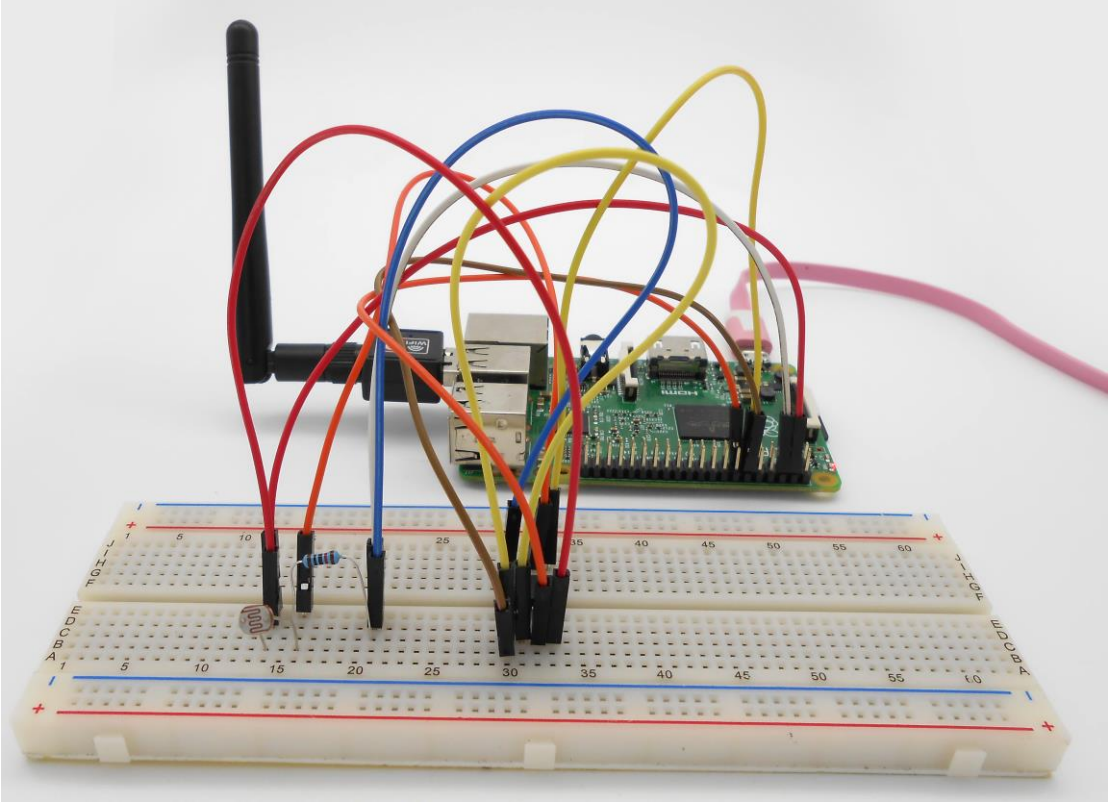
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/17_photoresistor.py)

2.2 Run the program

```
$ sudo python 17_photoresistor.py
```

Now, when you try to block the light towards the photoresistor, you will find that the value displayed in the screen will be reduced. Otherwise, when you use a powerful light to irradiate the photoresistor, the value displayed will be increased.



Summary

By learning this lesson, we have learned how to detect surrounding light intensity with the photoresistor. You can play your own wisdom, and make more originality based on this experiment and the former experiment.

Lesson 18 Thermistor

Overview

In this lesson, we will learn how to use a thermistor to collect temperature by programming the Raspberry Pi and ADC0832.

Requirement

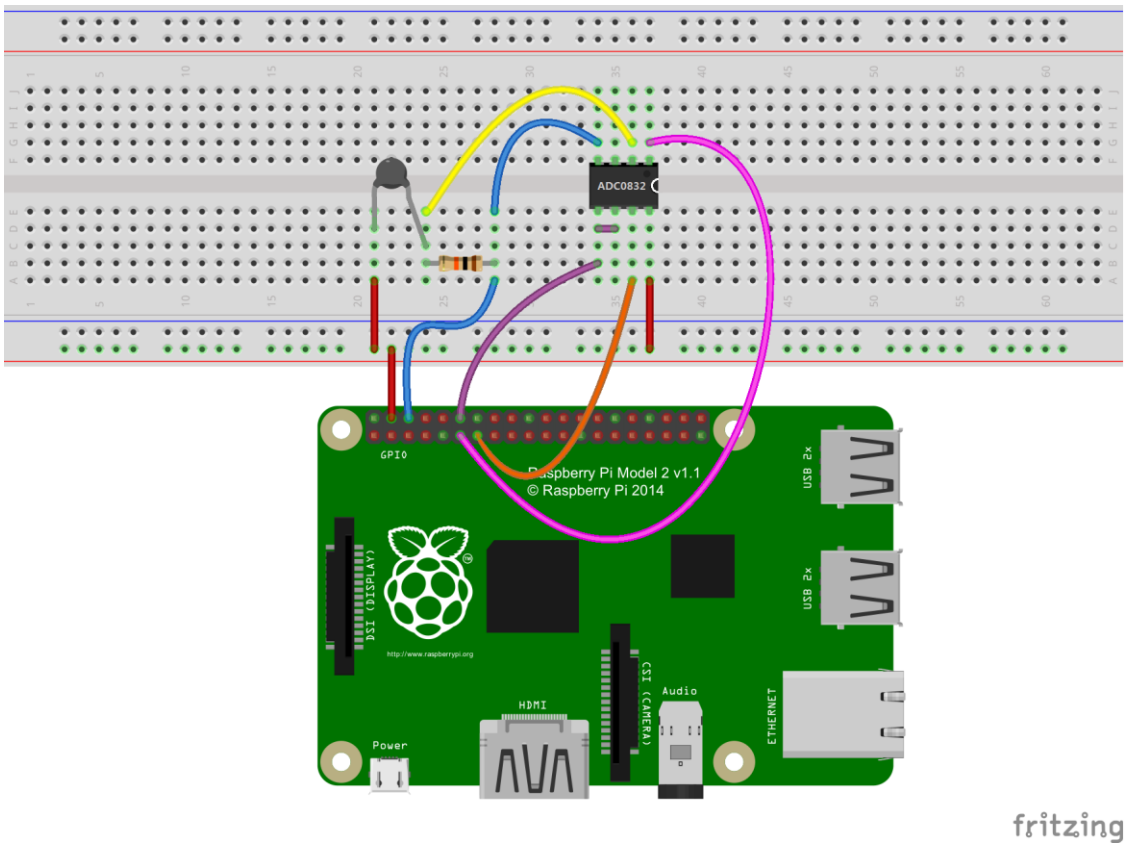
- 1* Raspberry Pi
- 1* ADC0832
- 1* Thermistor
- 1* 10KΩ Resistor
- 1* Breadboard
- Several Jumper wires

Principle

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 ADC0832 to convert analog signal into digital signal.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/18_thermistor/thermistor.c)

2.2 Compile the program

```
$ gcc thermistor.c -o thermistor -lwiringPi
```

2.3 Run the program

```
$ sudo ./thermistor
```

Python user:

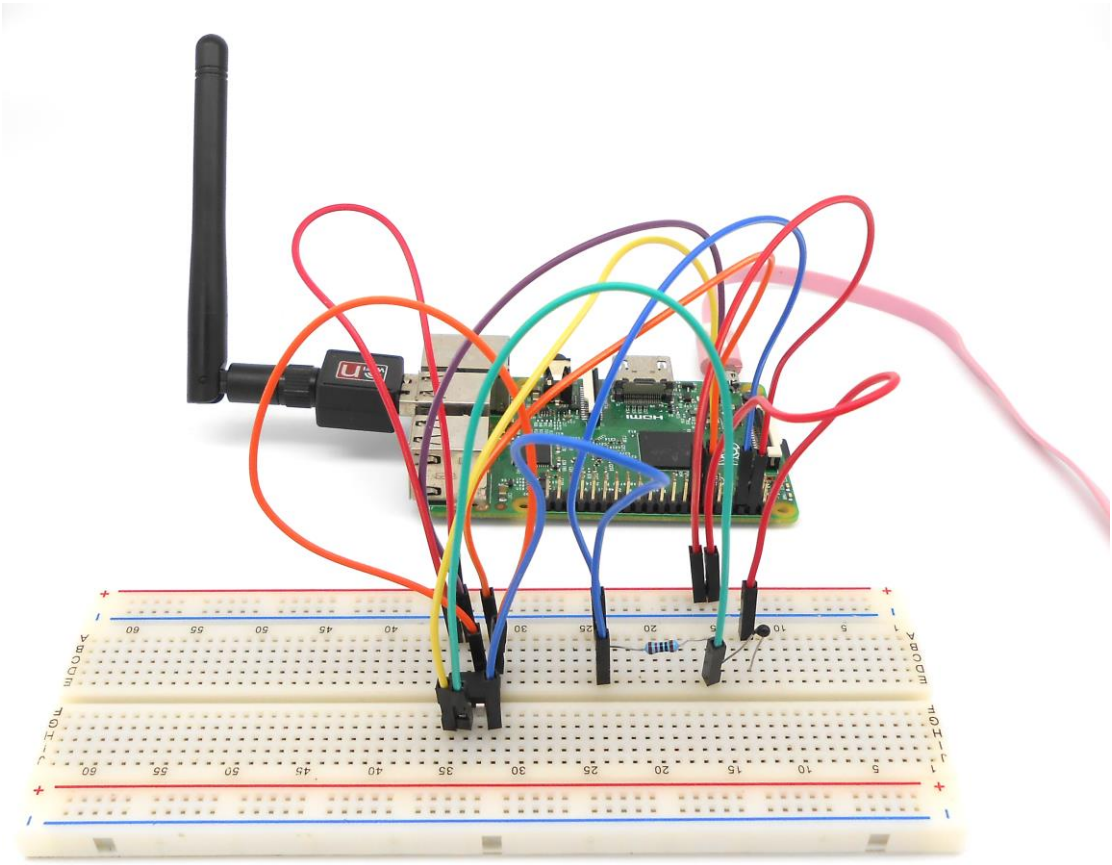
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/18_thermistor.py)

2.2 Run the program

```
$ sudo python 18_thermistor.py
```

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



Lesson 19 LED Bar Graph

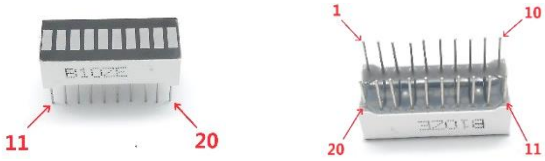
Overview

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

Requirement

- 1* Raspberry Pi
- 1* ADC0832
- 1* LED bar graph
- 10* 220Ω Resistor
- 1* 10KΩ Potentiometer
- 1* Breadboard
- Several Jumper wires

Principle



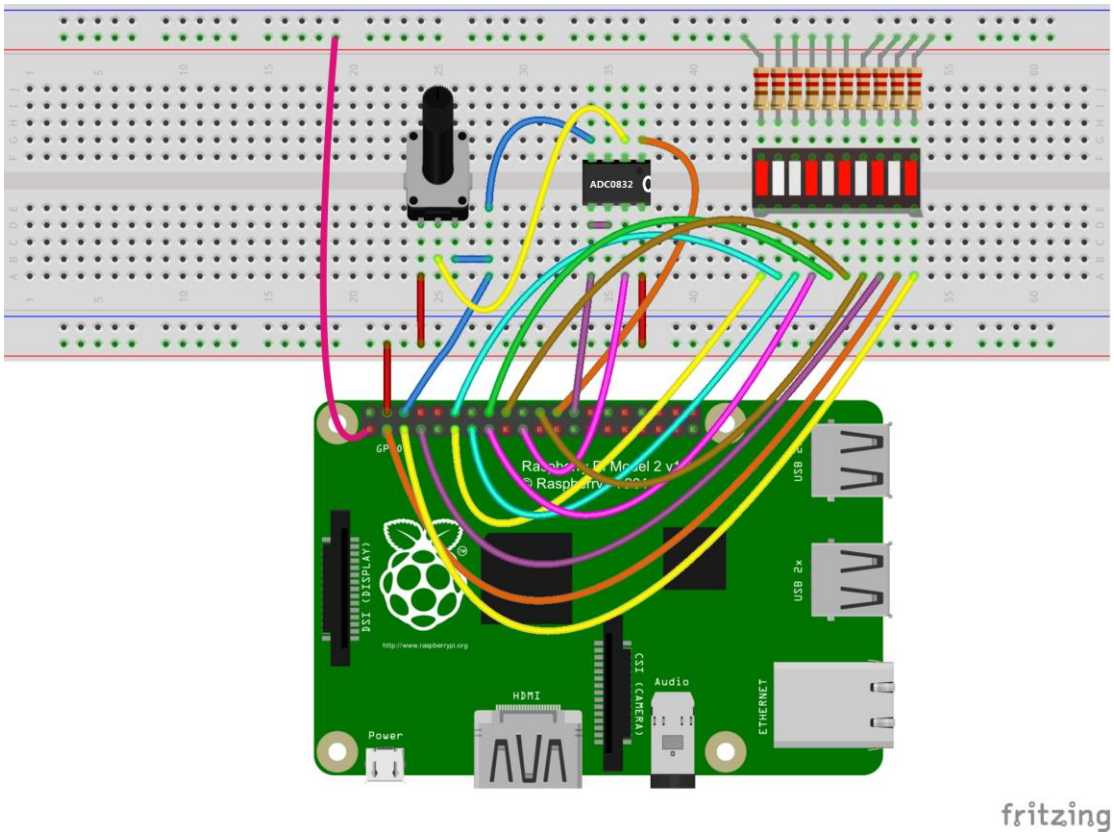
The bar graph - a series of LEDs in a line, such as you see on an audio display - is a common hardware display for analog sensors. It's made up of a series of LEDs in a row, an analog input like a potentiometer, and a little code in between. You can buy multi-LED bar graph displays fairly cheaply. This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

This tutorial borrows from the For Loop and Arrays tutorial as well as the Analog Input tutorial.

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.

Procedures

1. Build the circuit



C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/19_ledBar/ledBar.c)

2.2 Compile the program

```
$ gcc ledBar.c -o ledBar -lwiringPi
```

2.3 Run the program

```
$ sudo ./ledBar
```

Python user:

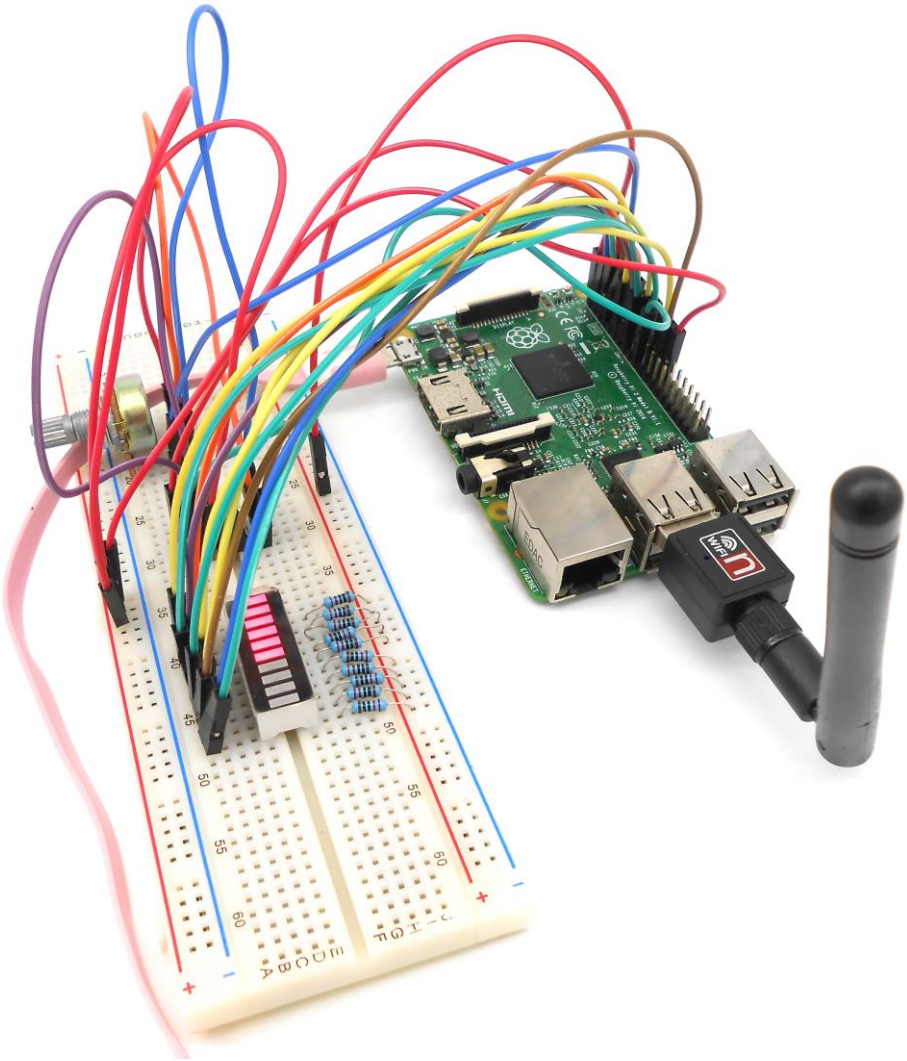
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/19_ledBar/ledBar.py)

2.2 Run the program

```
$ sudo python ledBar.py
```

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



Lesson 20 Controlling an LED through LAN

Overview

In this lesson, we will introduce TCP and socket, and then programming to control an LED through the local area network(LAN).

Requirement

- 1* Raspberry Pi
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

1. TCP

The Transmission Control Protocol (TCP) is a core protocol of the Internet Protocol Suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network. TCP is the protocol that major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.

2. Socket

A network socket is an endpoint of an inter-process communication across a computer network. Today, most communication between computers is based on the Internet Protocol; therefore most network sockets are Internet sockets.

A socket API is an application programming interface (API), usually provided by the operating system, that allows application programs to control and use network sockets. Internet socket APIs are usually based on the Berkeley sockets standard.

A socket address is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension. Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread.

Several Internet socket types are available:

1. Datagram sockets, also known as connectionless sockets, which use User Datagram Protocol (UDP).

2. Stream sockets, also known as connection-oriented sockets, which use Transmission

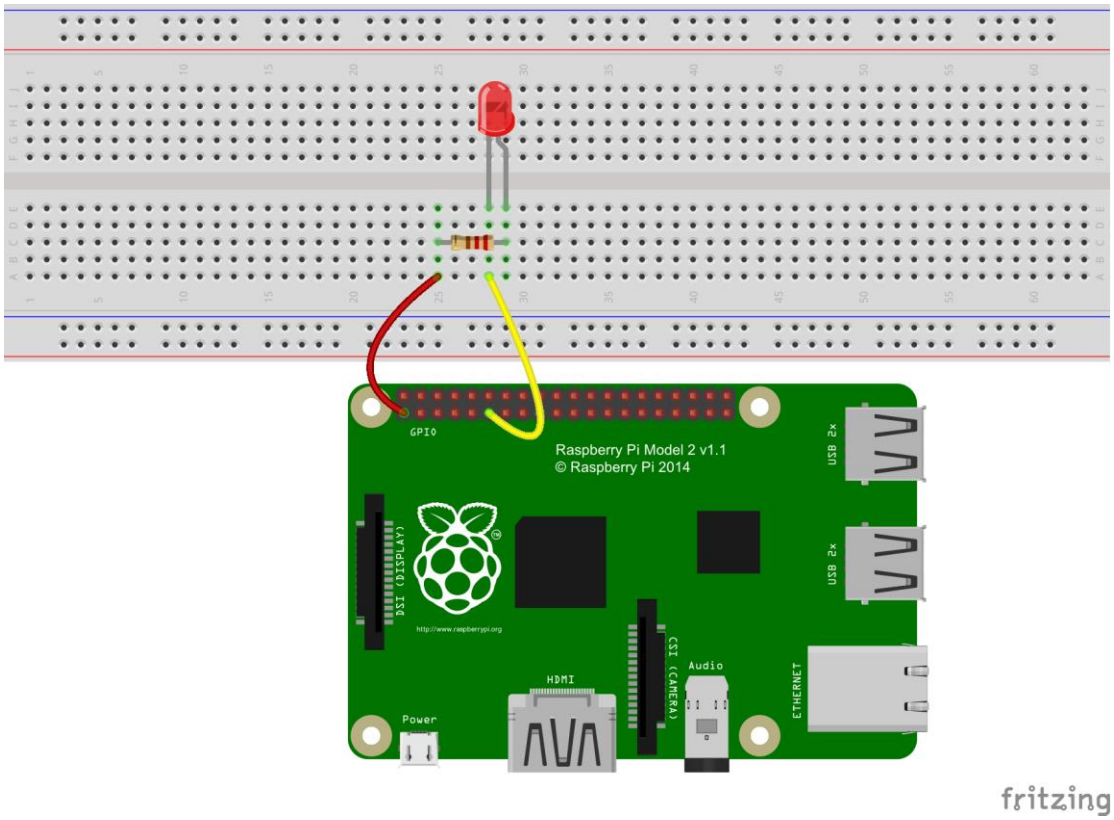
Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP).

3. Raw sockets (or Raw IP sockets), typically available in routers and other network equipment. Here the transport layer is bypassed, and the packet headers are made accessible to the application.

In this experiment, our program is based on stream socket, and the program is divided into two parts, the client and the server. The server routine is run on the Raspberry Pi, and the client routine is run on the PC. So you can send command to the server through the client, and then control the LED connected to the Raspberry Pi.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the server code with vim or nano on the Raspberry Pi.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/20_TCPCtrlLed/ledServer.c)

2.2 Compile the program(On Raspberry Pi)

```
$ gcc ledServer.c -o ledServer –lwiringPi
```

2.3 Edit and save the client code with vim or nano on the PC.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/20_TCPCtrlLed/ledClient.c)

2.4 Compile the program(On Linux PC)

$ gcc ledClient.c -o ledClient

2.5 Run the program

$ sudo ./ledServer (On Raspberry Pi)

$ ./ledClient 192.168.1.188 (On PC, Modify the IP Address to your Raspberry Pi's IP Address)

Now, input "ON" in the terminal and then press Enter, you will find the LED connected to the Raspberry Pi is on, input "OFF", the LED is off.

Python user:

2.1 Edit and save the server code with vim or nano on the Raspberry Pi.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/20_TCPCtrlLed/ledServer.py)

2.2 Edit and save the client code with vim or nano on the PC.

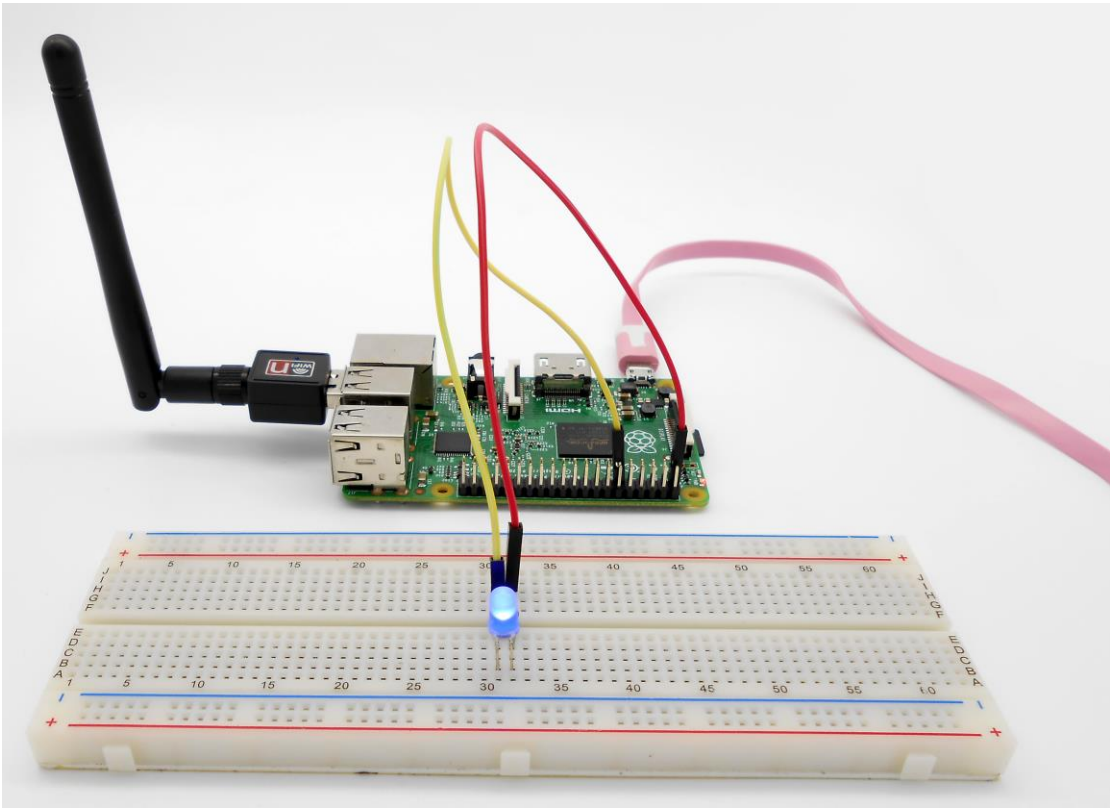
(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/20_TCPCtrlLed/ledClient.py)

2.3 Run the program

$ sudo python ledServer.py (On Raspberry Pi)

$ python ledClient.py (On PC)

Now, input "ON" in the terminal and then press Enter, you will find the LED connected to the Raspberry Pi is on, input "OFF", the LED is off.



Summary

By learning this lesson, you should have understood the basic principles of inter-computer communication. I hope this lesson will help you open the door to learn IoT(Internet of Things).

Lesson 21 Movement Detection Based on PIR

Overview

In this lesson, we will learn how to use the Passive Infrared (PIR) sensor to detect the movement nearby.

Requirement

- 1* Raspberry Pi
- 1* PIR Movement Sensor
- Several Jumper wires

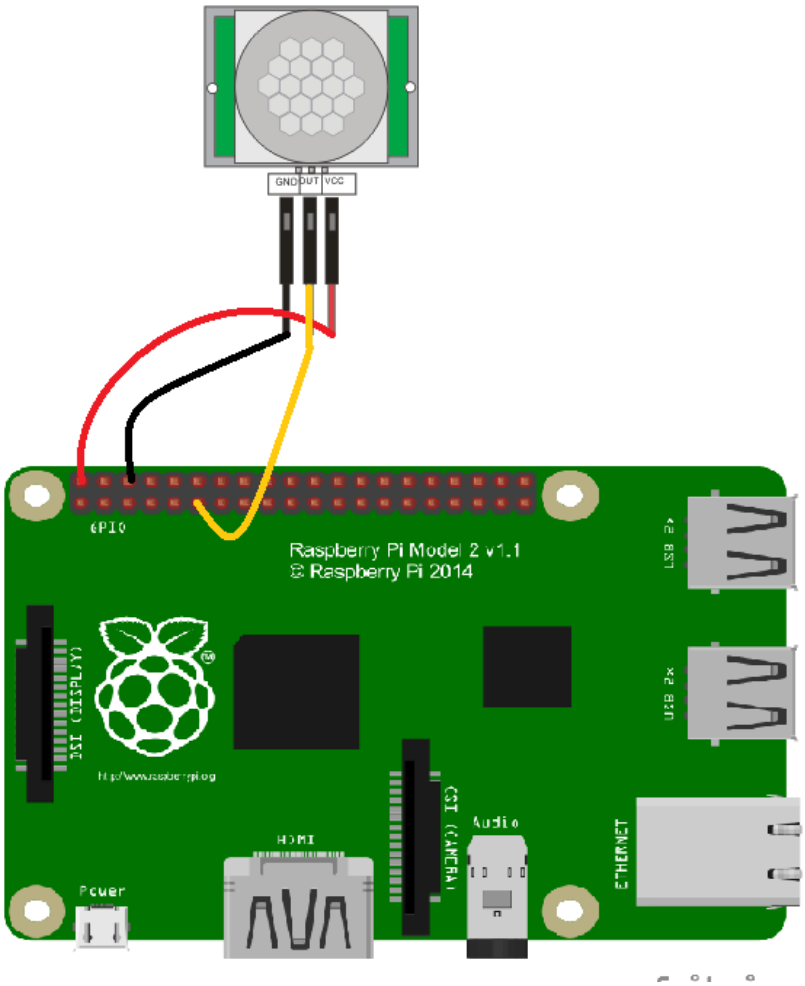
Principle

The sensor has three pins marked “OUT”, “-”, and “+” (for Output, GND, and +5V).

PIR sensors respond to heat and can be triggered by animals such as cats and dogs, as well as by people and other heat sources. The ‘output’ pin of Passive Infrared (PIR) sensor will go HIGH when the motion has been detected.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/21_PIR/PIR.c)

2.2 Compile the program

```
$ gcc PIR.c -o PIR -lwiringPi
```

2.3 Run the program

```
$ sudo ./PIR
```

Python user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/21_PIR.py)

2.2 Run the program

```
$ sudo python 21_PIR.py
```

Lesson 22 DC Motor

Overview

In this comprehensive experiment, we will learn how to control the state of DC motor with Raspberry Pi. The state of DC motor includes its forward, reverse, acceleration, deceleration and stop.

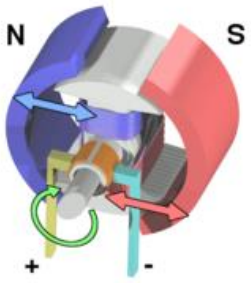
Requirement

- 1* Raspberry Pi
- 1* L9110 DC Motor driver
- 1* DC motor
- 4* Button
- 1* LED
- 1* 220Ω Resistor
- 1* Capacitor(104, 0.1uF)
- 1* Breadboard
- Several Jumper wires

Principle

1. DC motor

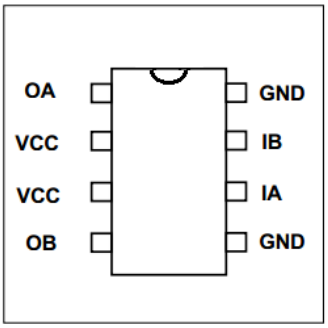
A DC motor is any of a class of electrical machines that converts direct current electrical power into mechanical power. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor. Most types produce rotary motion; a linear motor directly produces force and motion in a straight line.



DC motors were the first type widely used, since they could be powered from existing direct-current lighting power distribution systems. A DC motor's speed can be controlled over a wide range, using either a variable supply voltage or by changing the strength of current in its field windings. Small DC motors are used in tools, toys, and appliances. The universal motor can operate on direct current but is a lightweight motor used for portable power tools and appliances.

2. L9110

L9110 is a driver chip which is used to control and drive motor. The chip has two TTL/CMOS compatible input terminals, and possesses the property of anti-interference: it has high current driving capability, two output terminals that can directly drive DC motor, each output port can provide 750~800mA dynamic current, and its peak current can reach 1.5~2.0A; L9110 is widely applied to various motor drives, such as toy cars, stepper motor, power switches and other electric circuits.



OA, OB: These are used to connect the DC motor.

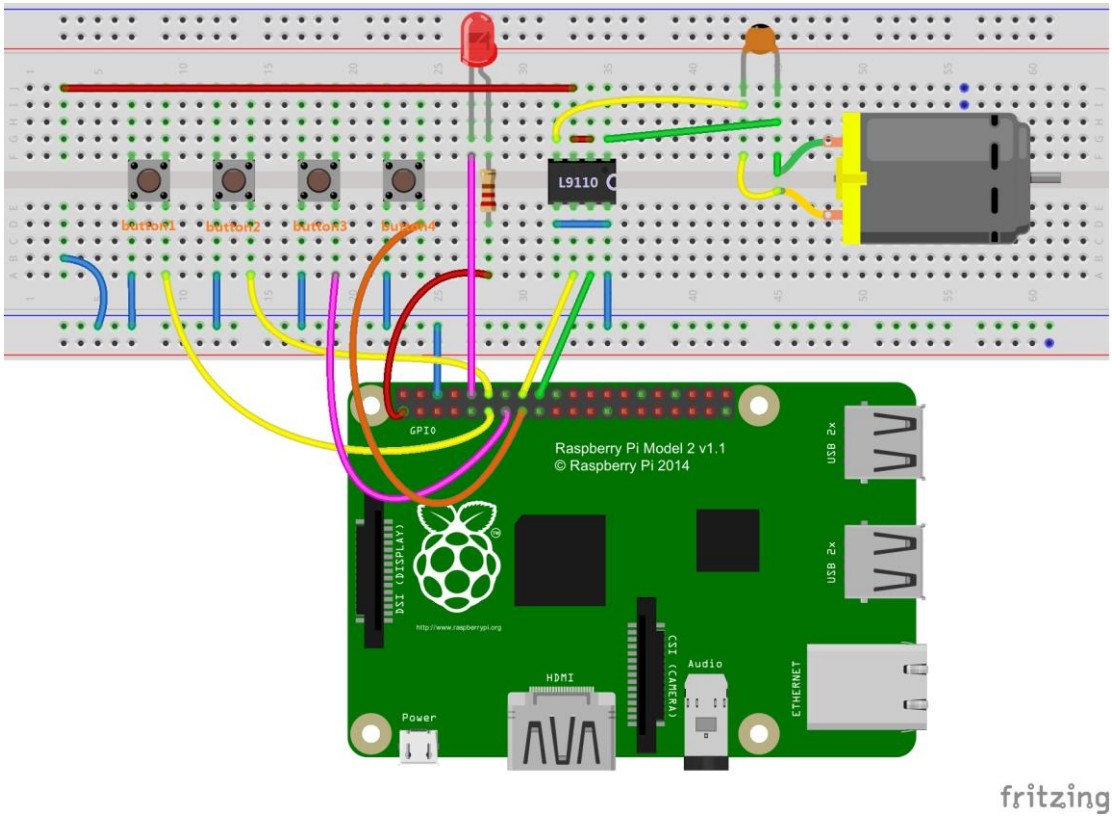
VCC: Power supply (+5V)

GND: The cathode of the power supply (Ground).

IA, IB: The input terminal of drive signal.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/22_motor/motor.c)

2.2 Compile the program

```
$ gcc motor.c -o motor –lwiringPi -lpthread
```

2.3 Run the program

```
$ sudo ./motor
```

Python user:

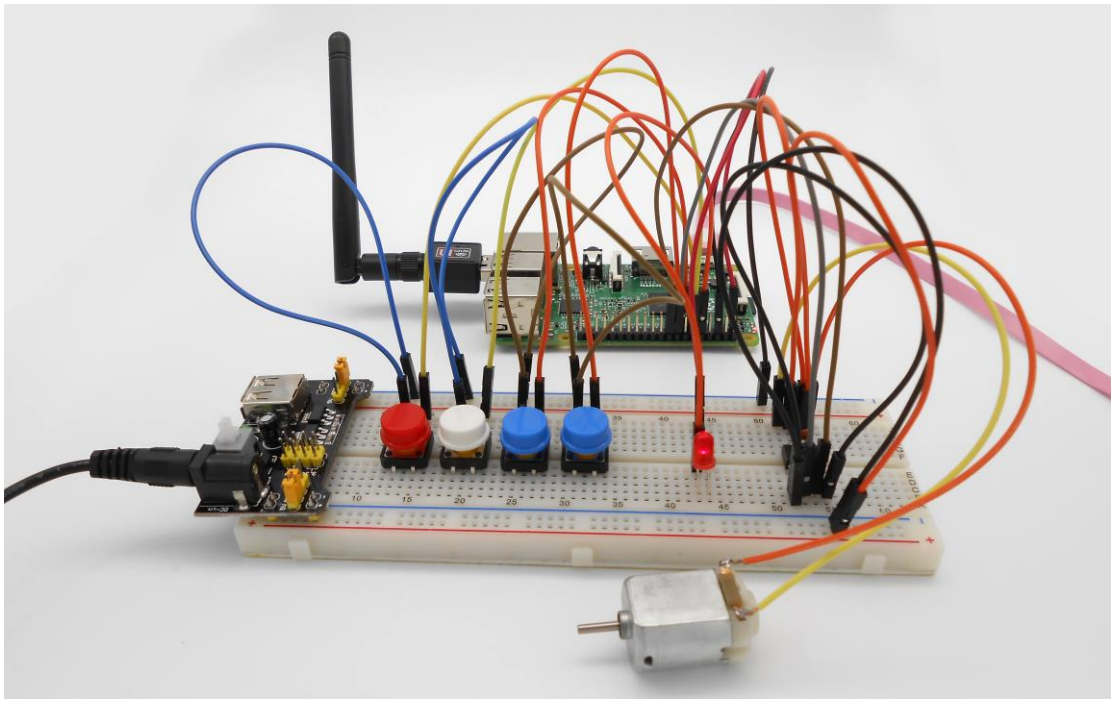
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/22_motor.py)

2.2 Run the program

```
$ sudo python 22_motor.py
```

Press the button 1 to stop or run the DC motor; press the button 2 to forward or reverse the DC motor; Press the button 3 to accelerate the DC motor; Press the button 4 to decelerate the DC motor. When the motor is running, the LED will be lit up. Otherwise, the LED will be extinguished.



Summary

I think you must have grasped the basic theory and programming of the DC motor after studying this experiment. You not only can forward and reverse it, but also can regulate its speed. Besides, you can do some interesting applications with the combination of this course and your prior knowledge.

Lesson 23 How to control a servo

Overview

In this lesson, we will introduce a new electronic device (Servo) to you, and tell you how to control it with Raspberry Pi.

Requirement

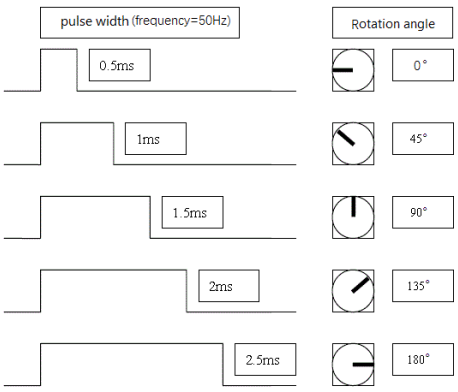
- 1* Raspberry Pi
- 1* Servo
- 1* Breadboard
- Several Jumper wires

Principle

Servo 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.

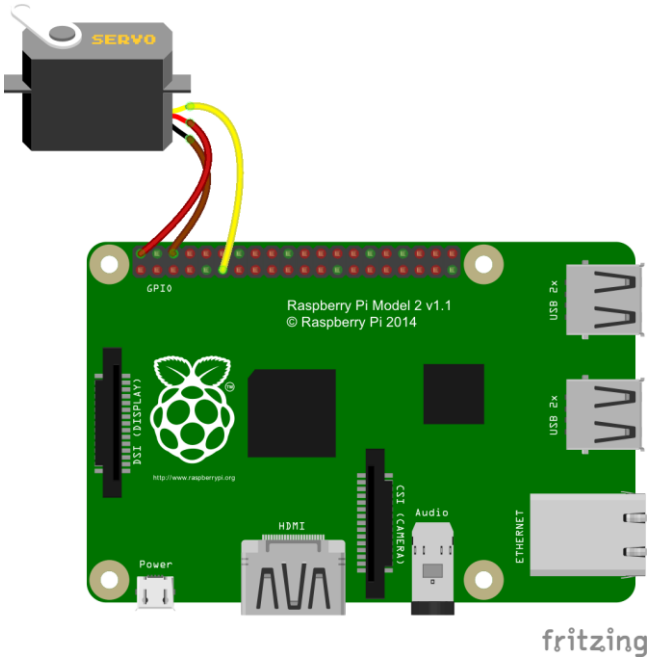
Servo consists of shell, circuit board, non-core motor, gear and location detection. Its working principle is as follow: Raspberry Pi sends PWM signal to servo motor, and then this signal is processed by IC on circuit board to calculate rotation direction to drive motor, and then this driving power is transferred to swing arm by reduction gear. At the same time, position detector returns location signal to judge whether set location is reached or not.

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



Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/23_servo/servo.c)

2.2 Compile the program

```
$ gcc servo.c -o servo -lwiringPi
```

2.3 Run the program

```
$ sudo ./servo
```

Python user:

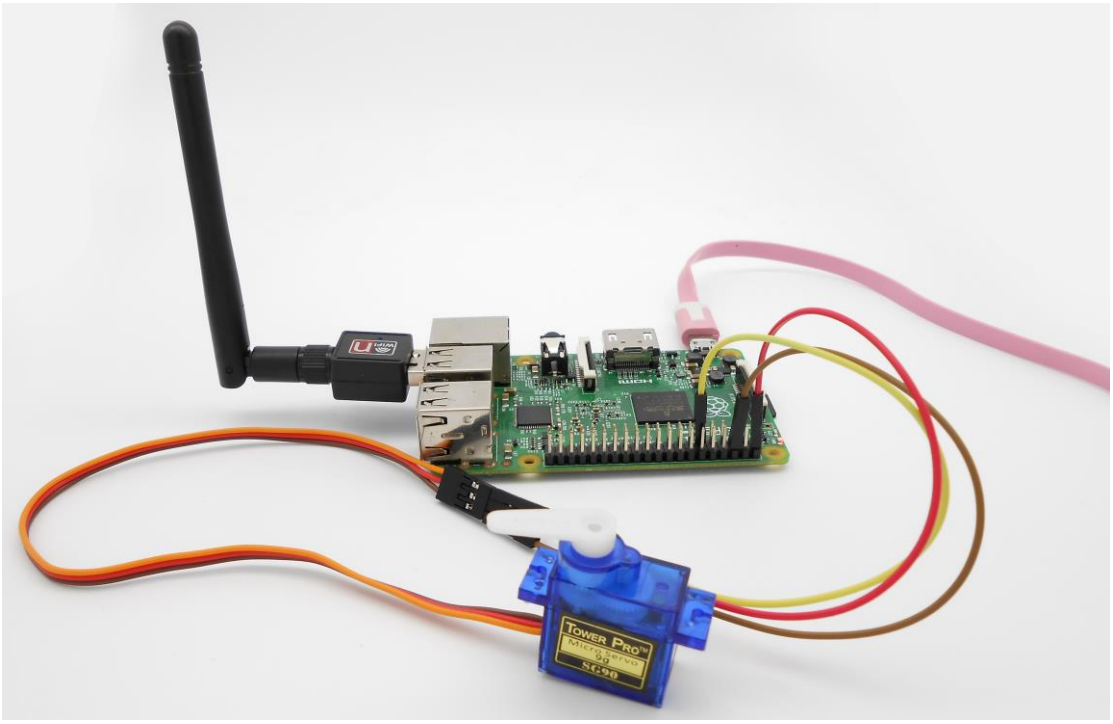
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/23_servo.py)

2.2 Run the program

```
$ sudo python 23_servo.py
```

Press Enter, you should see the servo motor rotate 180 degrees. And then rotate in opposite direction.


TOWER PRO
SG90

Lesson 24 How to control a stepper motor

Overview

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

Requirement

- 1* Raspberry Pi
- 1* Stepper motor
- 1* ULN2003 stepper motor driver module
- Several Jumper wires

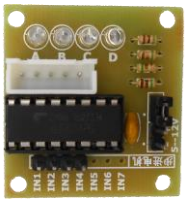
Principle

1. 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.

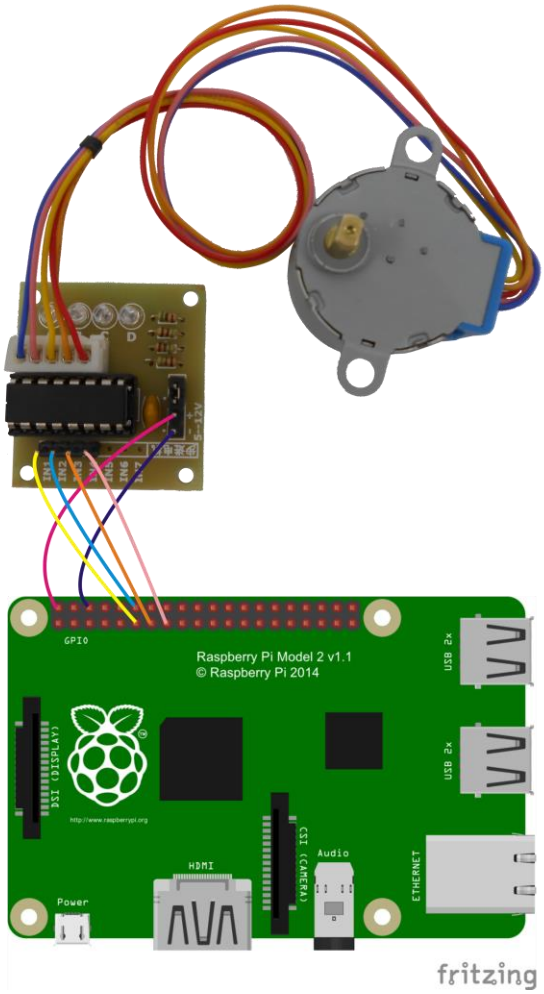
2. ULN2003 driver module



The Raspberry Pi's GPIO cannot directly drive a stepper motor due to the weak current. Therefore, a driver circuit is necessary for controlling a stepper motor. What we used in this experiment is a ULN2003-based 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 connected with the Raspberry Pi.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/24_stepperMotor/stepperMotor.c)

2.2 Compile the program

```
$ gcc stepperMotor.c -o stepperMotor -lwiringPi
```

2.3 Run the program

```
$ sudo ./stepperMotor
```

Python user:

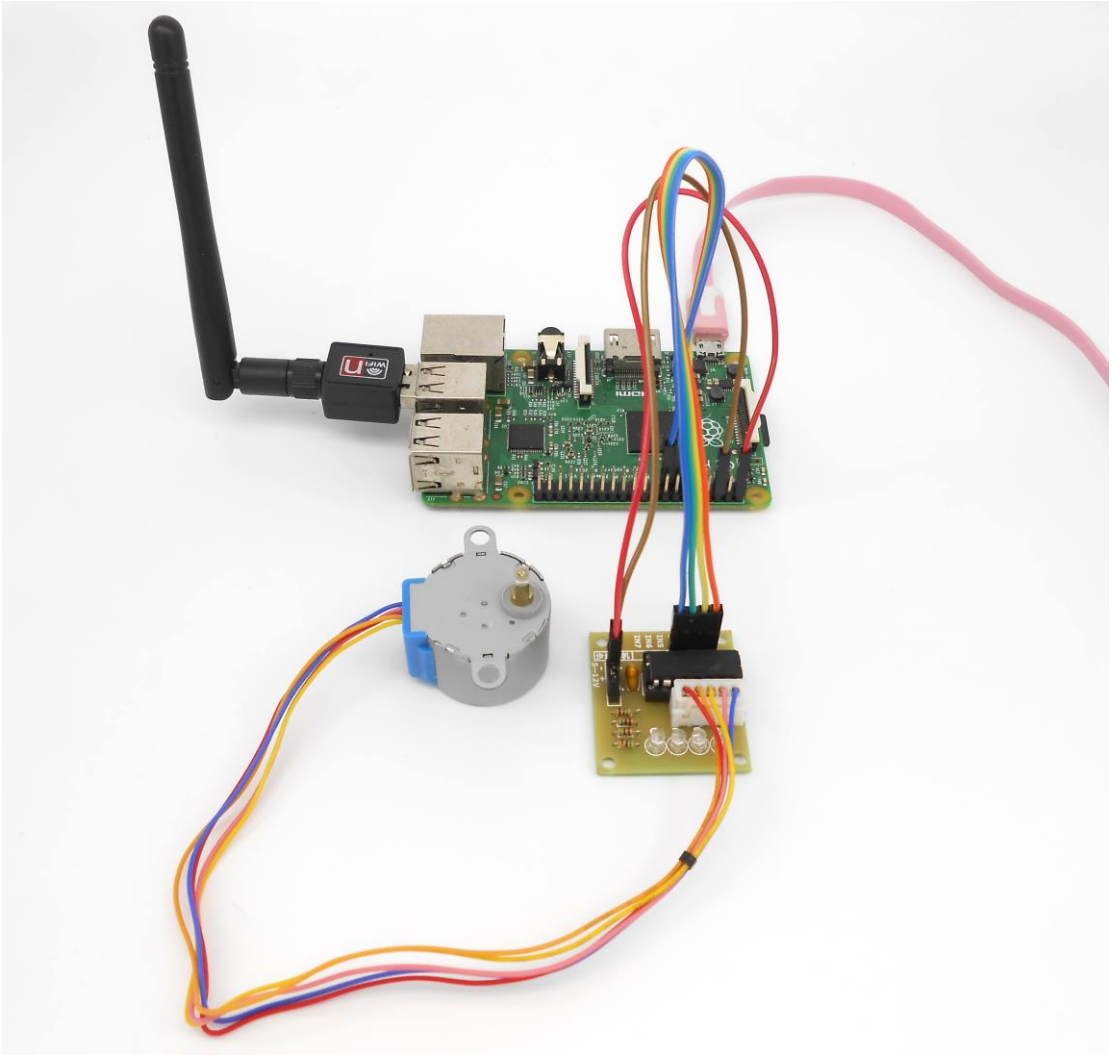
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/24_stepperMotor.py)

2.2 Run the program

```
$ sudo python 24_stepperMotor.py
```

Press Enter, you should see that the stepper motor is running.



Lesson 25 How to use the acceleration sensor ADXL345

Overview

In this lesson, we will learn how to use an acceleration sensor ADXL345 to get acceleration data.

Requirement

- 1* Raspberry Pi
- 1* ADXL345 module
- Several Jumper wires

Principle

1. 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.

2. Key functions

- int wiringPiI2CSetup (int devId)

This initialises the I2C system with your given device identifier. The ID is the I2C number of the device and you can use the **i2cdetect** program to find this out. wiringPiI2CSetup() will work out which revision Raspberry Pi you have and open the appropriate device in /dev.

The return value is the standard Linux filehandle, or -1 if any error – in which case, you can consult errno as usual.

- int wiringPiI2CRead (int fd)

Simple device read. Some devices present data when you read them without having to do any register transactions.

- int wiringPiI2CWriteReg8 (int fd, int reg, int data)
- int wiringPiI2CWriteReg16 (int fd, int reg, int data)

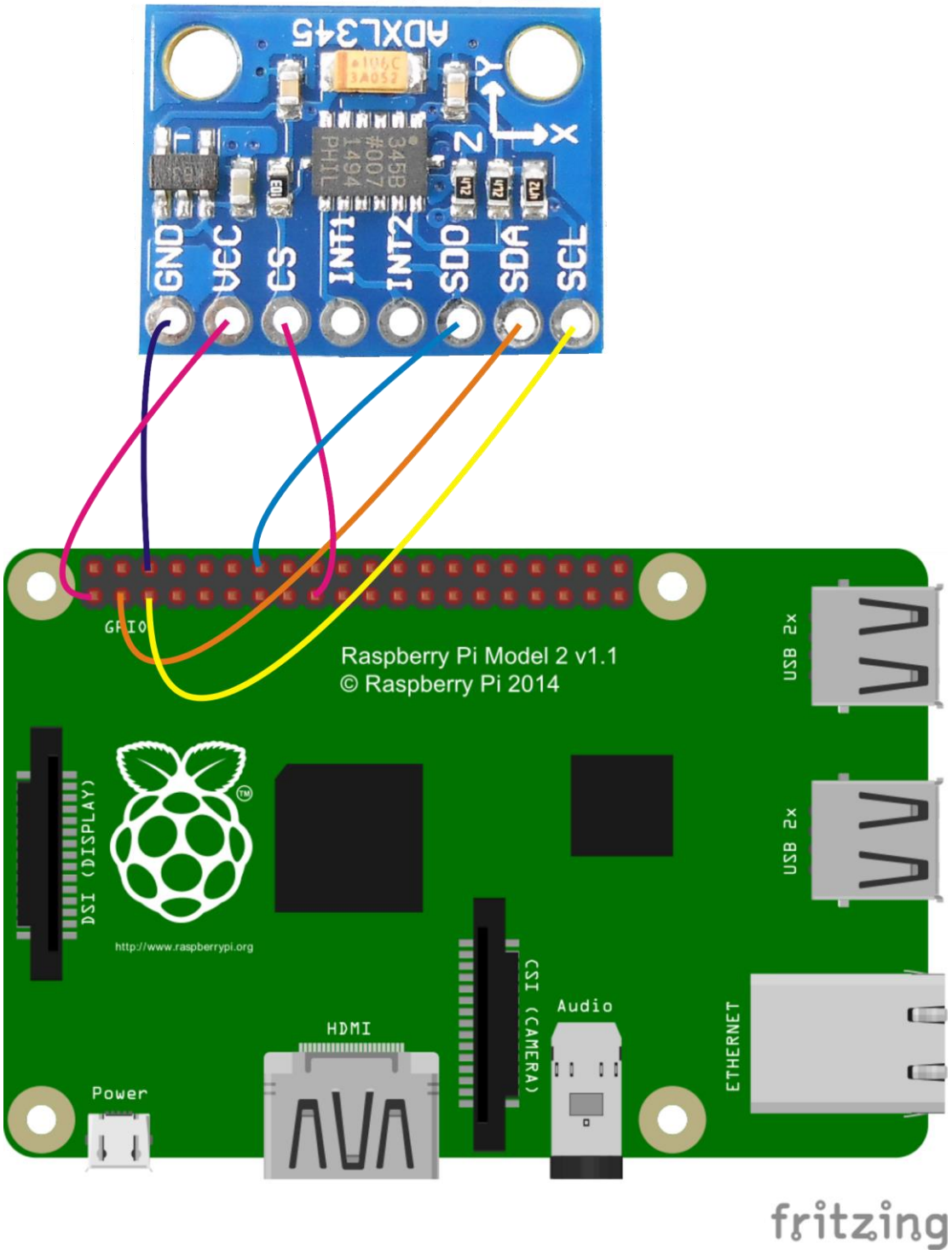
These write an 8 or 16-bit data value into the device register indicated.

- int wiringPiI2CReadReg8 (int fd, int reg)
- int wiringPiI2CReadReg16 (int fd, int reg)

These read an 8 or 16-bit value from the device register indicated.

Procedures

1. Build the circuit



2. Program

NOTE:

The following program use I2C interface. Before you run the program, please make sure the I2C driver

module of Raspberry Pi has loaded normally.

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/25_ADXL345/adxl345.c)

2.2 Compile the program

```
$ gcc adxl345.c -o adxl345 -lwiringPi
```

2.3 Run the program

```
$ sudo ./adxl345
```

Python user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/25_ADXL345/Adafruit_ADXL345/Adafruit_ADXL345.py)

2.2 Run the program

```
$ sudo python Adafruit_ADXL345.py
```

Press Enter, you should see that the acceleration data will be displayed on the terminal.

Lesson 26 PS2 Joystick

Overview

In this lesson, we will learn the usage of joystick. We program the Raspberry Pi to detect the state of joystick.

Requirement

- 1* Raspberry Pi
- 1* ADC0832
- 1* PS2 Joystick
- 1* Breadboard
- Several Jumper wires

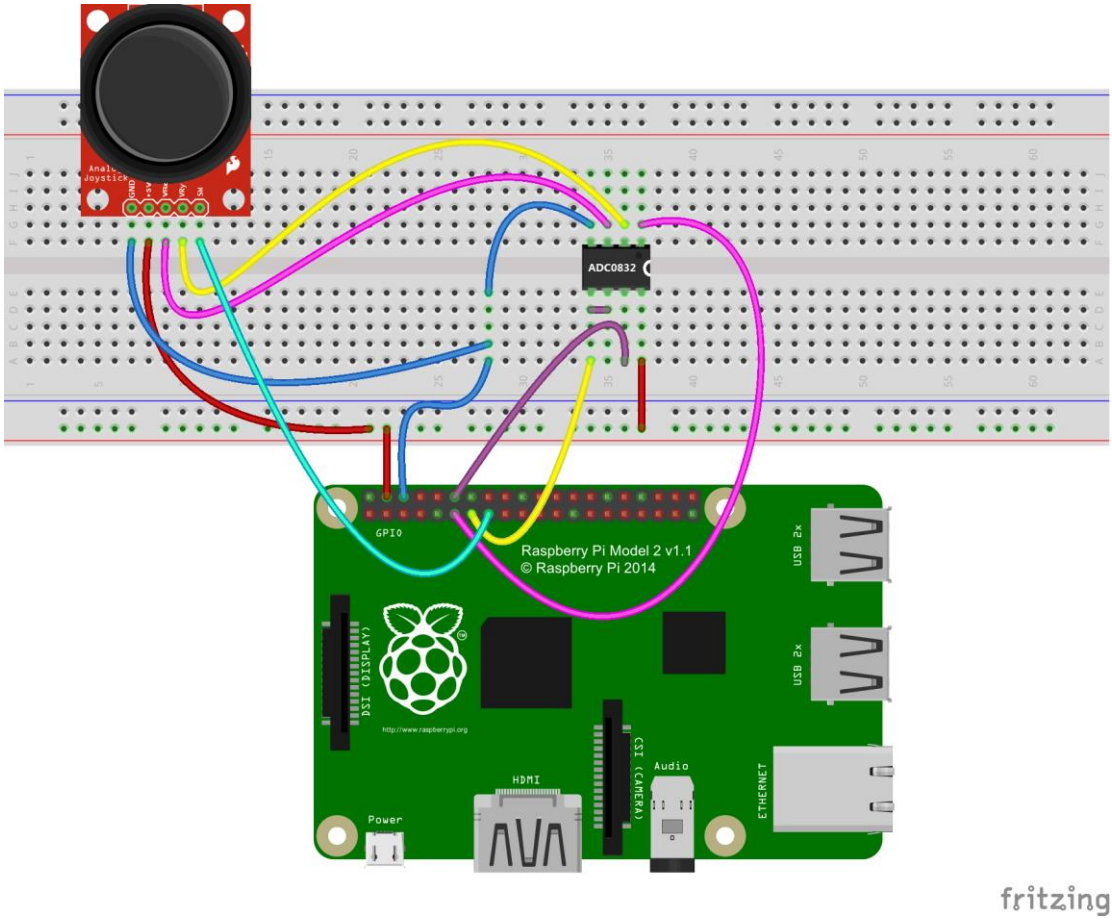
Principle

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.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/26_ps2Joystick/joystick.c)

2.2 Compile the program

```
$ gcc joystick.c -o joystick -lwiringPi
```

2.3 Run the program

```
$ sudo ./joystick
```

Python user:

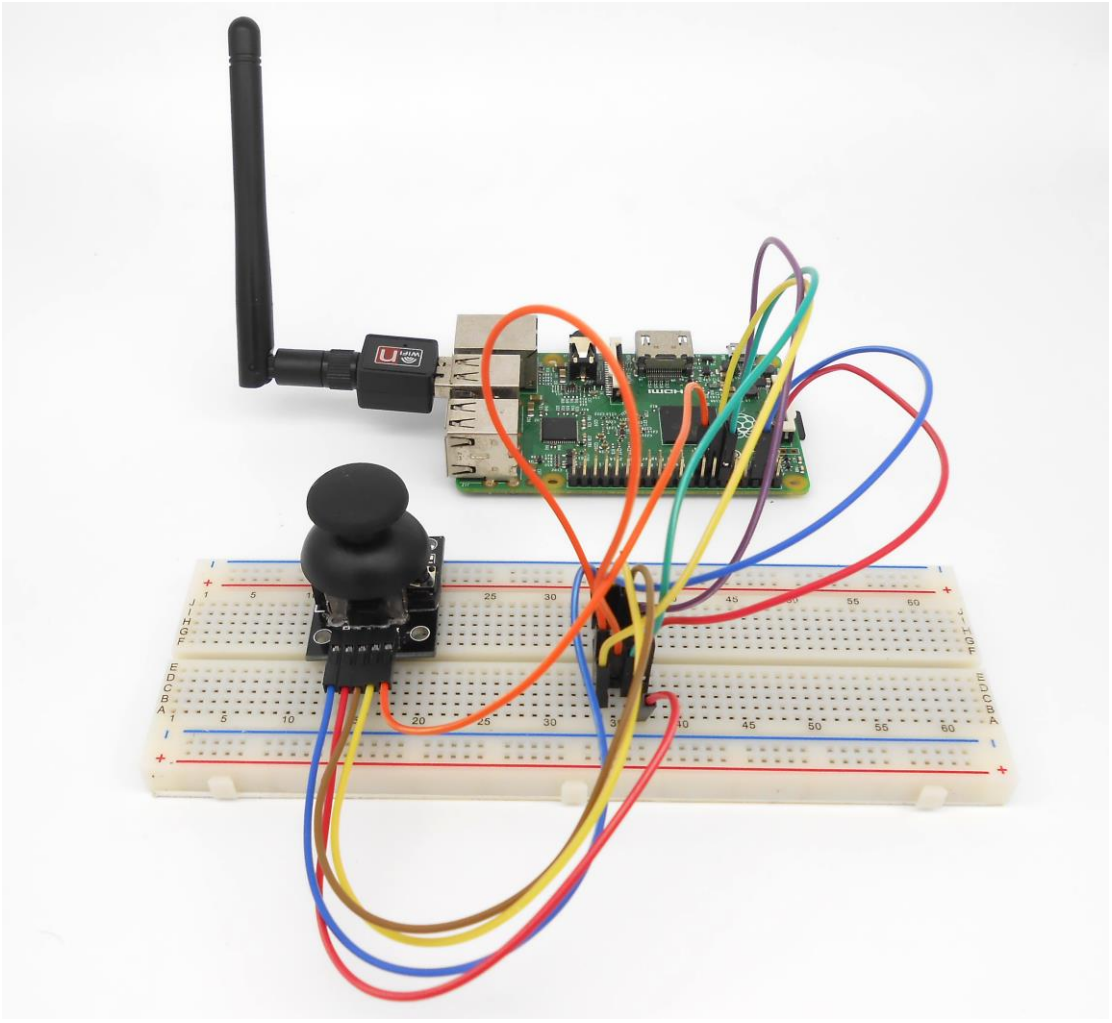
2.1 Edit and save the code with vim or nano.

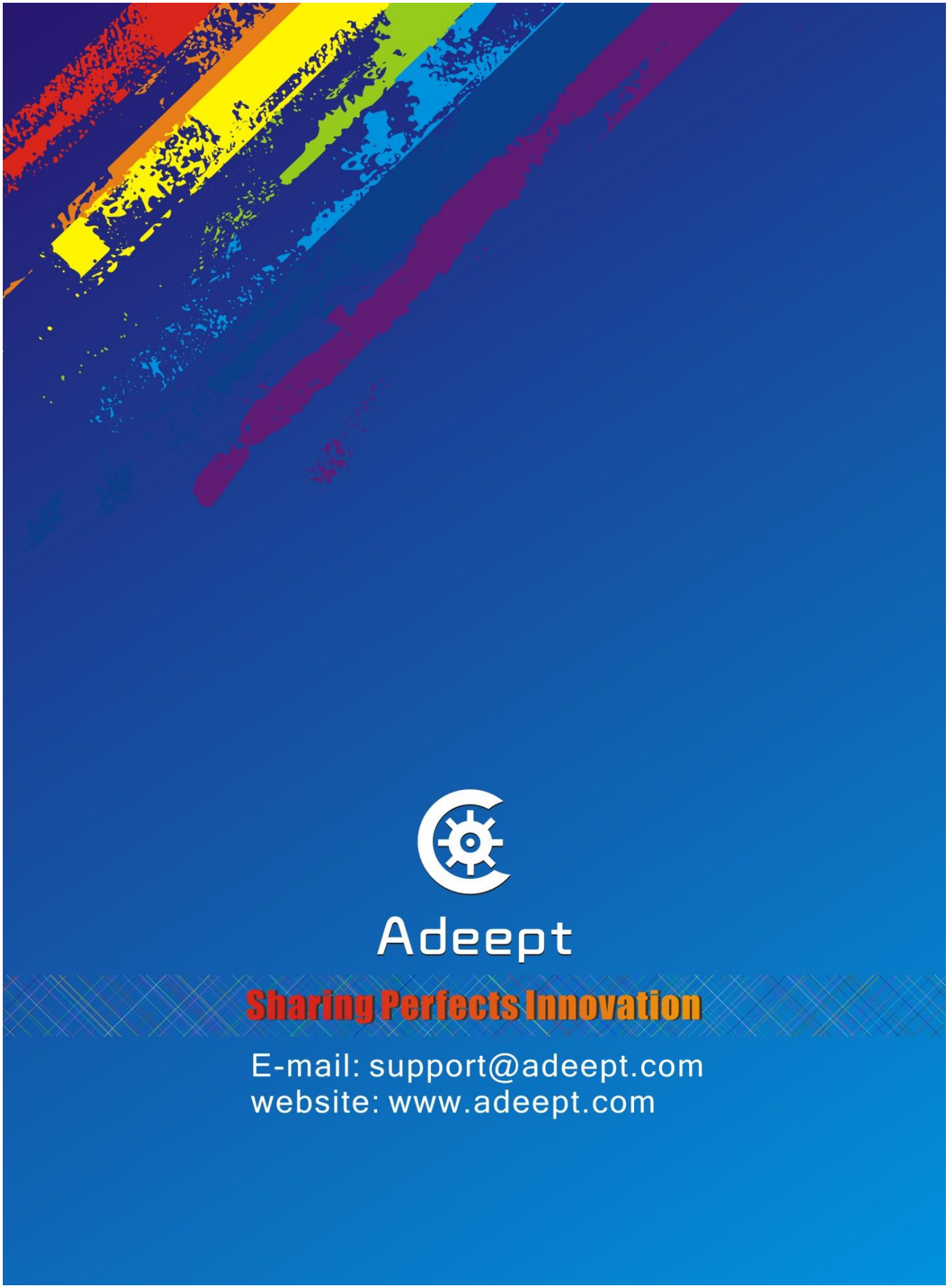
(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/26_joystick.py)

2.2 Run the program

```
$ sudo python 26_joystick.py
```

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




Adeept
Sharing Perfects Innovation
E-mail: support@adeept.com
website: www.adeept.com