```
    
    if(adc->detectI2C(0x48)){    // Detect the pcf8591.
        delete adc;                 // Free previously pointed memory
        adc = new PCF8591();    // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;                // Free previously pointed memory
        adc = new ADS7830();      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    printf("Program is starting ... \n");
    while(1){
        int adcValue = adc->analogRead(0);  //read analog value A0 pin
        float voltage = (float)adcValue / 255.0 * 3.3;    // calculate voltage
        float Rt = 10 * voltage / (3.3 - voltage);        //calculate resistance value of 
thermistor
        float tempK = 1/(1/(273.15 + 25) + log(Rt/10)/3950.0); //calculate temperature 
(Kelvin)
        float tempC = tempK -273.15;        //calculate temperature (Celsius)
        printf("ADC value : %d  ,\tVoltage : %.2fV, 
\tTemperature : %.2fC\n",adcValue,voltage,tempC);
        delay(100);
    }
    return 0;
}
```

In the code, the ADC value of ADC module A0 port is read, and then calculates the voltage and the resistance of Thermistor according to Ohms Law. Finally, it calculates the temperature sensed by the Thermistor, according to the formula.

### Python Code 11.1.1 Thermometer

If you did not configure I2C, please refer to Chapter 7. If you did, please continue.
First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

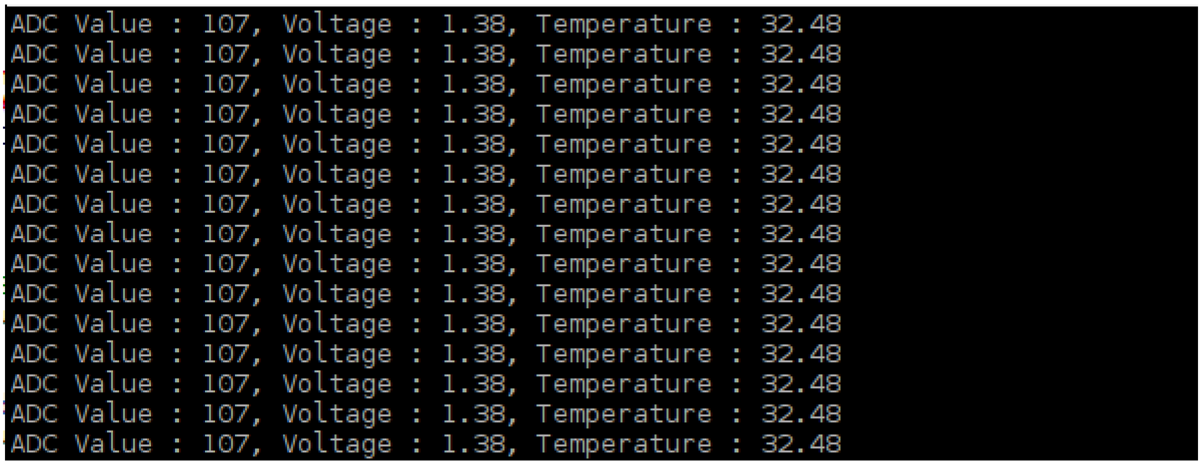
1. Use cd command to enter 11.1.1_Thermometer directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/11.1.1_Thermometer
```

2. Use python command to execute Python code “Thermometer.py”.

```
python Thermometer.py
```

After the program is executed, the Terminal window will display the current ADC value, voltage value and temperature value. Try to “pinch” the thermistor (without touching the leads) with your index finger and thumb for a brief time, you should see that the temperature value increases.



```
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
ADC Value : 107, Voltage : 1.38, Temperature : 32.48
```

The following is the code:

```python
import RPi.GPIO as GPIO
import time
import math
from ADCDevice import *

adc = ADCDevice() # Define an ADCDevice class object

def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = PCF8591()
    elif(adc.detectI2C(0x4b)): # Detect the ads7830
        adc = ADS7830()
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)
```

```

def loop():
    while True:
        value = adc.analogRead(0)        # read ADC value A0 pin
        voltage = value / 255.0 * 3.3        # calculate voltage
        Rt = 10 * voltage / (3.3 - voltage)     # calculate resistance value of thermistor
        tempK = 1/(1/(273.15 + 25) + math.log(Rt/10)/3950.0) # calculate temperature
(Kelvin)
        tempC = tempK -273.15        # calculate temperature (Celsius)
        print ('ADC Value : %d, Voltage : %.2f,
Temperature : %.2f'%(value,voltage,tempC))
        time.sleep(0.01)

def destroy():
    adc.close()
    GPIO.cleanup()

if __name__ == '__main__':  # Program entrance
    print ('Program is starting ... ')
    setup()
    try:
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        destroy()
```

In the code, the ADC value of ADC module A0 port is read, and then calculates the voltage and the resistance of Thermistor according to Ohms Law. Finally, it calculates the temperature sensed by the Thermistor, according to the formula.

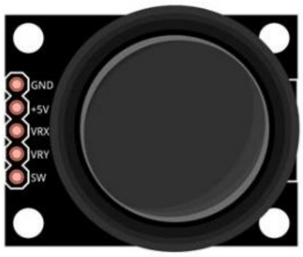
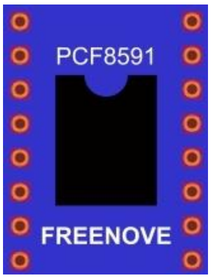
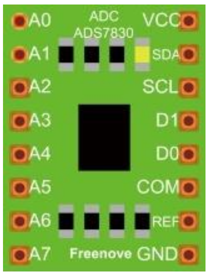
# Chapter 12 Joystick

In an earlier chapter, we learned how to use Rotary Potentiometer. We will now learn about joysticks, which are electronic modules that work on the same principle as the Rotary Potentiometer.

## Project 12.1 Joystick

In this project, we will read the output data of a joystick and display it to the Terminal screen.

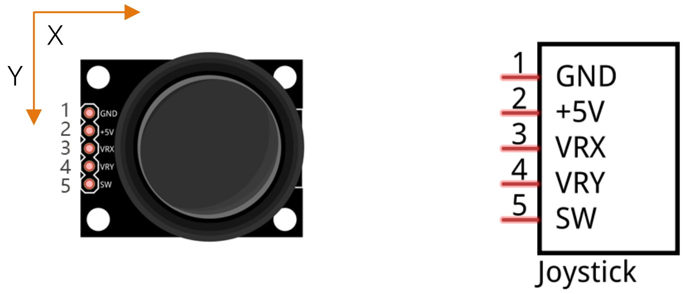
### Component List

<table>
<tr><td colspan="2">Raspberry Pi x1<br>GPIO Extension Board &amp; Ribbon Cable x1<br>Breadboard x1</td><td>Jumper x18<br> </td></tr>
<tr><td>Joystick x1<br><br> </td><td>ADC module x1<br><br> <br>or<br><br> </td><td>Resistor<br>10kΩ x3<br> </td></tr>
</table>

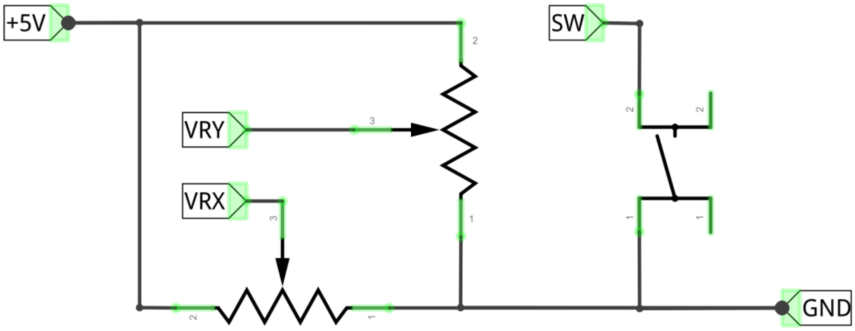
## Component knowledge

### Joystick

A Joystick is a kind of input sensor used with your fingers. You should be familiar with this concept already as they are widely used in gamepads and remote controls. It can receive input on two axes (Y and or X) at the same time (usually used to control direction on a two dimensional plane). And it also has a third direction capability by **pressing down (Z axis/direction)**.



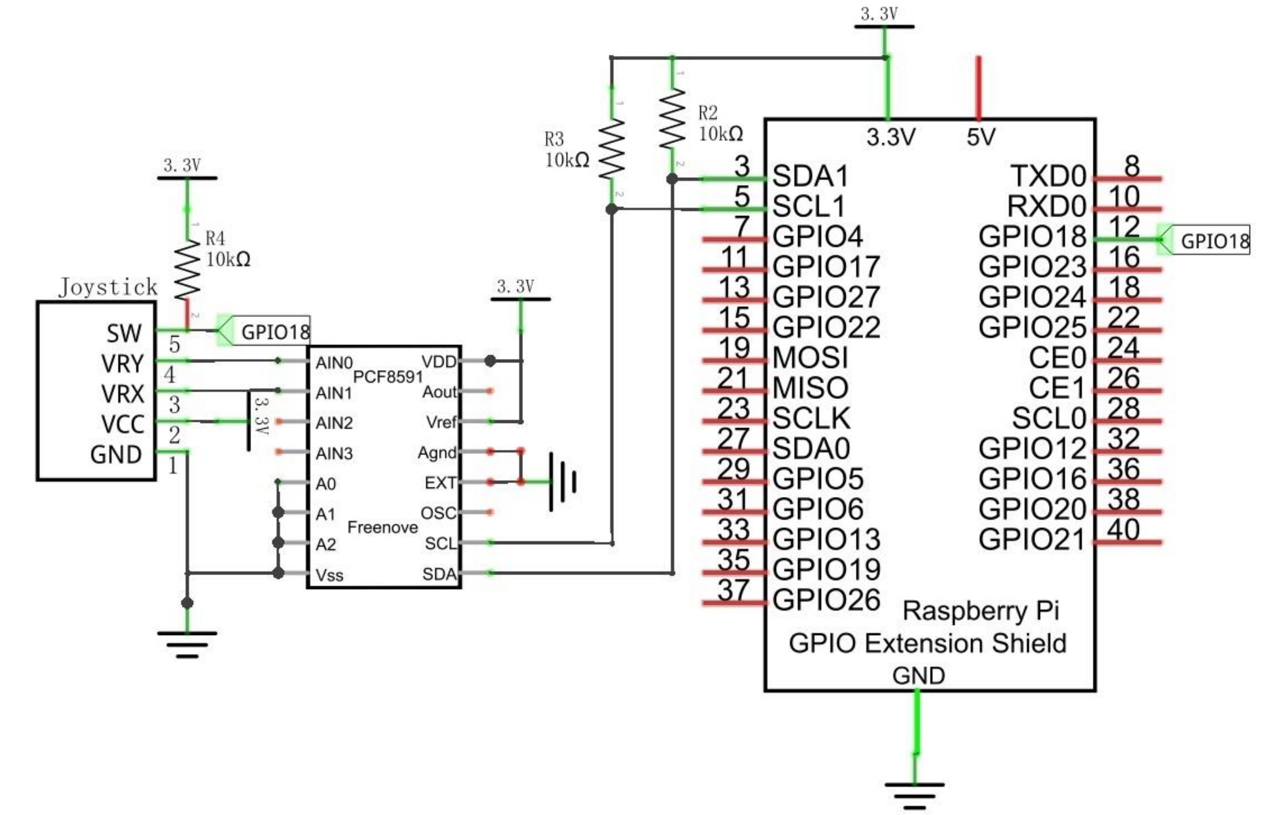
This is accomplished by incorporating two rotary potentiometers inside the Joystick Module at 90 degrees of each other, placed in such a manner as to detect shifts in direction in two directions simultaneously and with a Push Button Switch in the "vertical" axis, which can detect when a User presses on the Joystick.



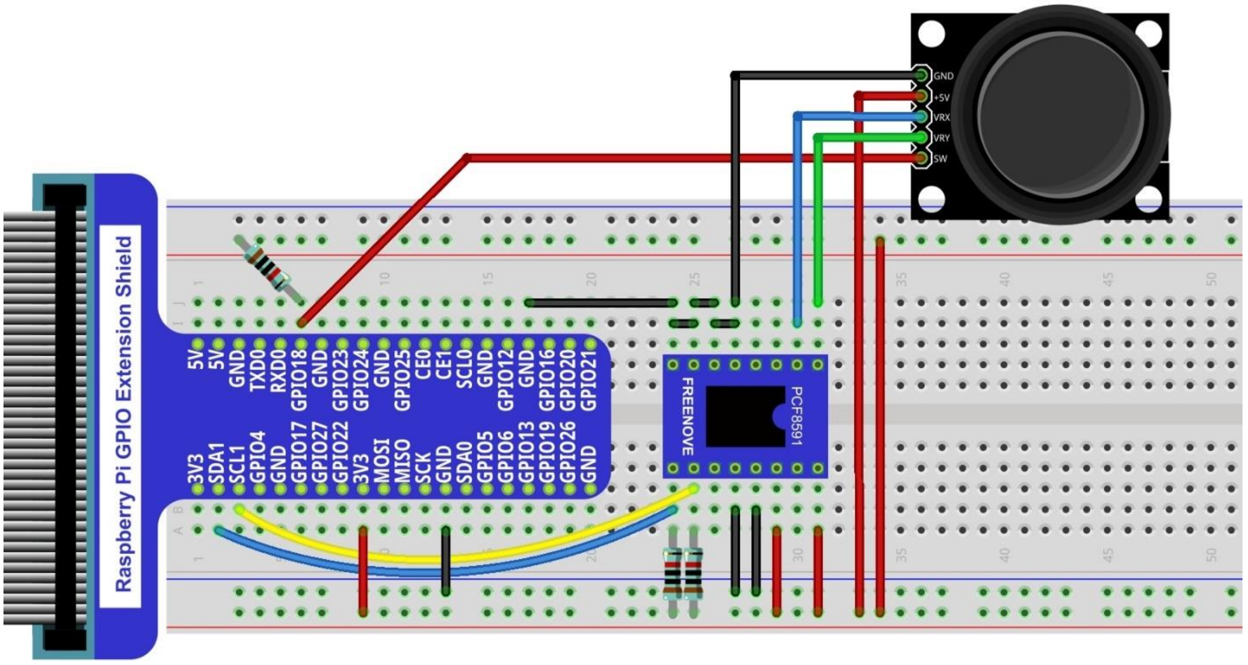
When the Joystick data is read, there are some differences between the axes: data of X and Y axes is analog, which needs to use the ADC. The data of the Z axis is digital, so you can directly use the GPIO to read this data or you have the option to use the ADC to read this.

## Circuit with PCF8591

Schematic diagram

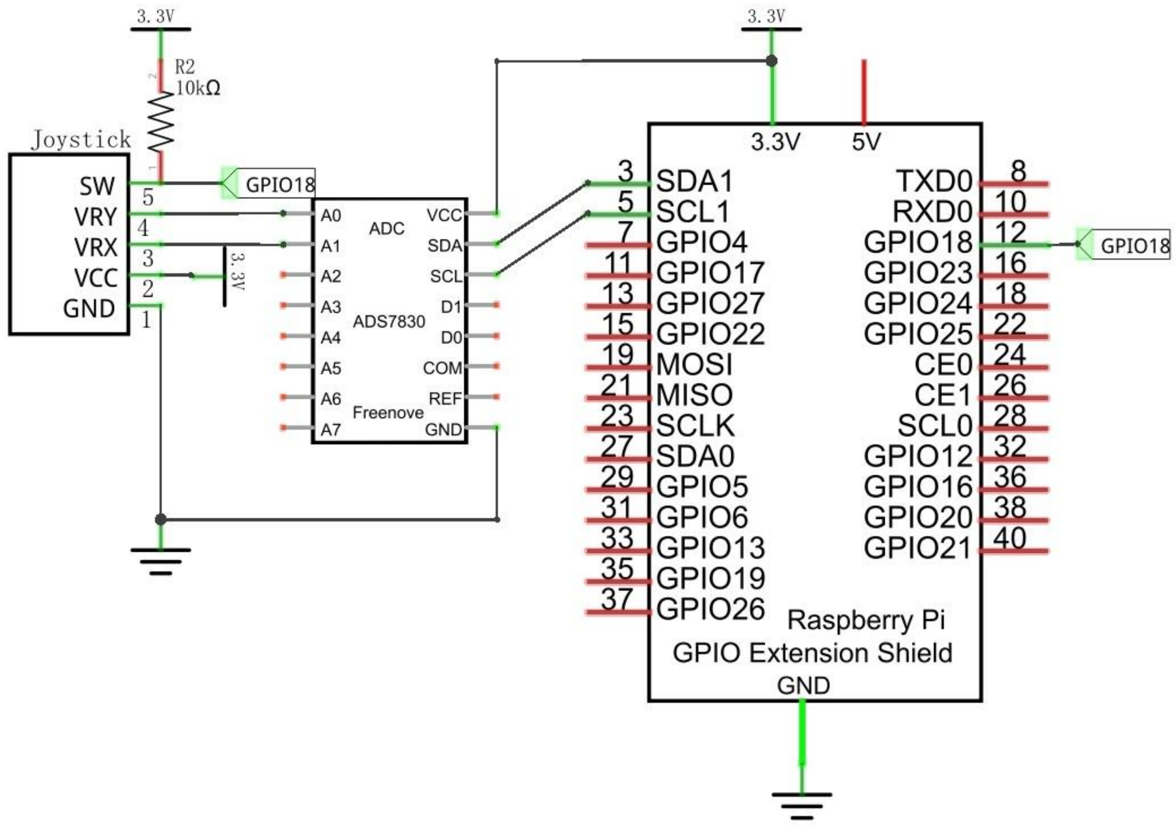


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

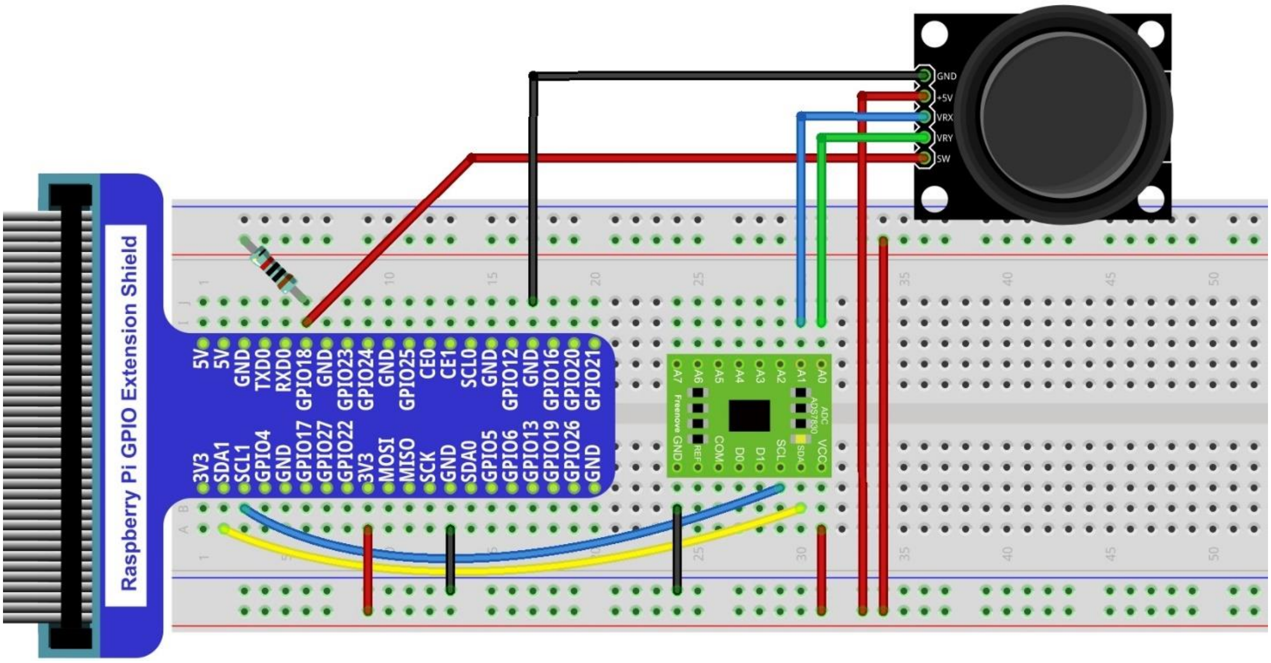


## Circuit with ADS7830

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**



## Code

In this project's code, we will read the ADC values of X and Y axes of the Joystick, and read digital quality of the Z axis, then display these out in Terminal.

C Code 12.1.1 Joystick

If you did not configure I2C, please refer to Chapter 7. If you did, please continue.

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 12.1.1_Joystick directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/12.1.1_Joystick
```

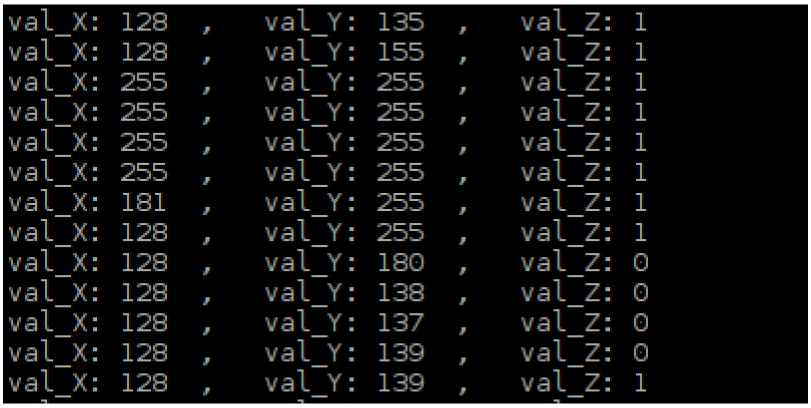
2. Use following command to compile "Joystick.cpp" and generate executable file "Joystick".

```
g++ Joystick.cpp -o Joystick -lwiringPi -lADCDevice
```

3. Then run the generated file "Joystick".

```
sudo ./Joystick
```

After the program is executed, the terminal window will display the data of 3 axes X, Y and Z. Shifting (moving) the Joystick or pressing it down will make the data change.



```
val_X: 128  ,   val_Y: 135  ,   val_Z: 1
val_X: 128  ,   val_Y: 155  ,   val_Z: 1
val_X: 255  ,   val_Y: 255  ,   val_Z: 1
val_X: 255  ,   val_Y: 255  ,   val_Z: 1
val_X: 255  ,   val_Y: 255  ,   val_Z: 1
val_X: 255  ,   val_Y: 255  ,   val_Z: 1
val_X: 181  ,   val_Y: 255  ,   val_Z: 1
val_X: 128  ,   val_Y: 255  ,   val_Z: 1
val_X: 128  ,   val_Y: 180  ,   val_Z: 0
val_X: 128  ,   val_Y: 138  ,   val_Z: 0
val_X: 128  ,   val_Y: 137  ,   val_Z: 0
val_X: 128  ,   val_Y: 139  ,   val_Z: 0
val_X: 128  ,   val_Y: 139  ,   val_Z: 1
```

The flowing is the code:

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

#define Z_Pin 1      //define pin for axis Z

ADCDevice *adc;  // Define an ADC Device class object

int main(void){
    adc = new ADCDevice();
    printf("Program is starting ... \n");

    if(adc->detectI2C(0x48)){    // Detect the pcf8591.
        delete adc;                 // Free previously pointed memory
        adc = new PCF8591();    // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;                // Free previously pointed memory
        adc = new ADS7830();      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    wiringPiSetup();
    pinMode(Z_Pin,INPUT);       //set Z_Pin as input pin and pull-up mode
    pullUpDnControl(Z_Pin,PUD_UP);
    while(1){
        int val_Z = digitalRead(Z_Pin);  //read digital value of axis Z
        int val_Y = adc->analogRead(0);      //read analog value of axis X and Y
        int val_X = adc->analogRead(1);
        printf("val_X: %d  ,\tval_Y: %d  ,\tval_Z: %d \n",val_X,val_Y,val_Z);
        delay(100);
    }
    return 0;
}
```

In the code, configure Z_Pin to pull-up input mode. In the while loop of the main function, use **analogRead** () to read the value of axes X and Y and use **digitalRead** () to read the value of axis Z, then display them.

```
    while(1){
        int val_Z = digitalRead(Z_Pin);  //read digital value of axis Z
        int val_Y = adc->analogRead(0);      //read analog value of axis X and Y
        int val_X = adc->analogRead(1);
        printf("val_X: %d  ,\tval_Y: %d  ,\tval_Z: %d \n",val_X,val_Y,val_Z);
        delay(100);
    }
```

### Python Code 12.1.1 Joystick

If you did not configure I2C, please refer to Chapter 7. If you did, please continue.

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

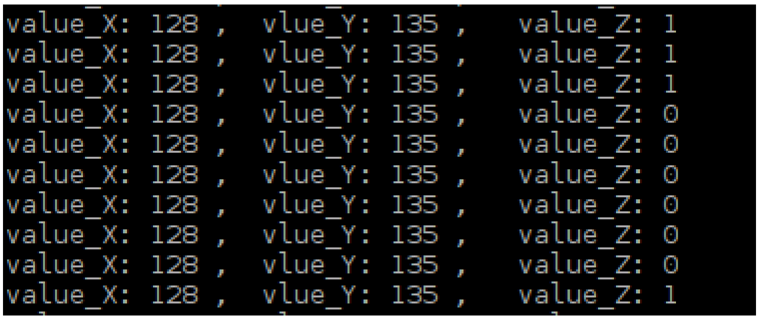
1. Use cd command to enter 12.1.1_Joystick directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/12.1.1_Joystick
```

2. Use Python command to execute Python code "Joystick.py".

```
python Joystick.py
```

After the program is executed, the Terminal window will display the data of 3 axes X, Y and Z. Shifting (moving) the joystick or pressing it down will make the data change.



```
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 1
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 1
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 1
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 0
value_X: 128 ,  vlue_Y: 135 ,   value_Z: 1
```

The following is the program code:

```
import RPi.GPIO as GPIO
import time
from ADCDevice import *

Z_Pin = 12      # define Z_Pin
adc = ADCDevice() # Define an ADCDevice class object

def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = PCF8591()
    elif(adc.detectI2C(0x4b)): # Detect the ads7830
        adc = ADS7830()
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(Z_Pin,GPIO.IN,GPIO.PUD_UP)   # set Z_Pin to pull-up mode
def loop():
    while True:
        val_Z = GPIO.input(Z_Pin)       # read digital value of axis Z
        val_Y = adc.analogRead(0)           # read analog value of axis X and Y
        val_X = adc.analogRead(1)
        print ('value_X: %d ,\tvlue_Y: %d ,\tvalue_Z: %d'%(val_X,val_Y,val_Z))
        time.sleep(0.01)

def destroy():
    adc.close()
    GPIO.cleanup()

if __name__ == '__main__':
    print ('Program is starting ... ') # Program entrance
    setup()
    try:
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        destroy()
```

In the code, configure Z_Pin to pull-up input mode. In while loop, use **analogRead** () to read the value of axes X and Y and use **GPIO.input** () to read the value of axis Z, then display them.

```
    while True:
        val_Z = GPIO.input(Z_Pin)       #read digital quality of axis Z
        val_Y = analogRead(0)           #read analog quality of axis X and Y
        val_X = analogRead(1)
        print ('value_X: %d ,\tvlue_Y: %d ,\tvalue_Z: %d'%(val_X,val_Y,val_Z))
        time.sleep(0.01)
```

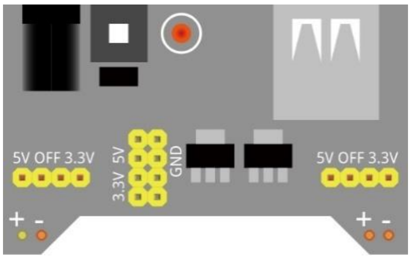
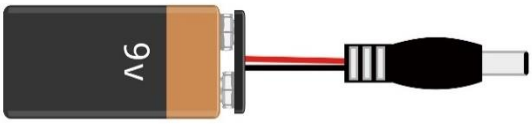
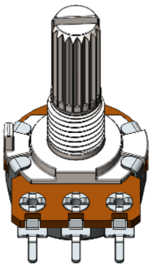
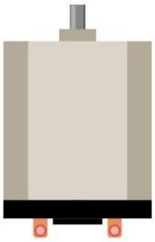
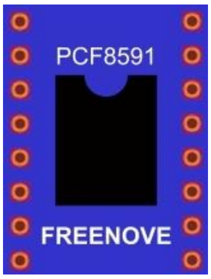
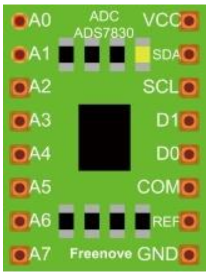
# Chapter 13 Motor & Driver

In this chapter, we will learn about DC Motors and DC Motor Drivers and how to control the speed and direction of a DC Motor.

## Project 13.1 Control a DC Motor with a Potentiometer

In this project, a potentiometer will be used to control a DC Motor. When the Potentiometer is at the midpoint position, the DC Motor will STOP, and when the Potentiometer is turned in either direction of this midpoint, the DC Motor speed increases until it reached the endpoint where the DC Motor achieves its maximum speed. When the Potentiometer is turned "Left" of the midpoint the DC Motor will ROTATE in one direction and when turned "Right" the DC Motor will ROTATE in the opposite direction.

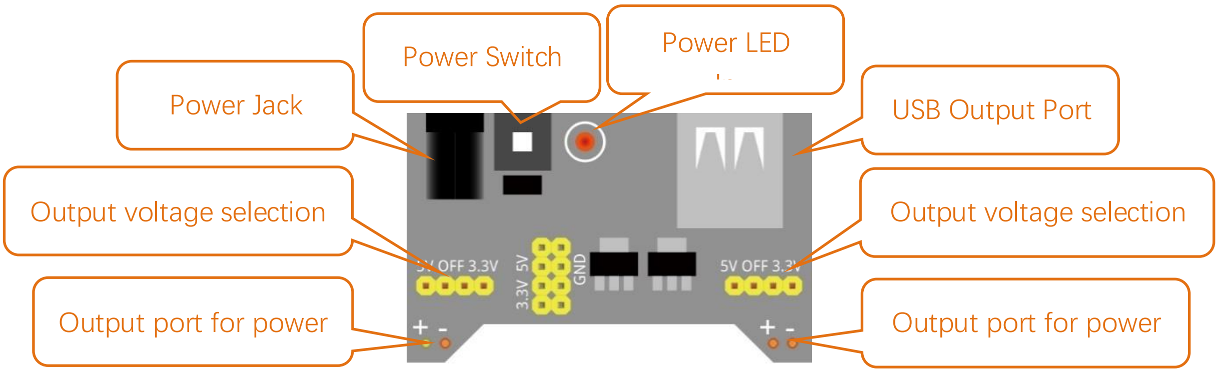
### Component List

<table>
<tr><td colspan="3">Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Ribbon Cable x1<br>Breadboard x1</td><td colspan="3">Jumper Wires x23<br> </td></tr>
<tr><td colspan="3">Breadboard Power Module x1<br> </td><td colspan="3">9V Battery (you provide) & 9V Battery Cable<br> </td></tr>
<tr><td>Rotary Potentiometer x1<br></td><td>DC Motor x1<br></td><td>10kΩ x2<br> </td><td colspan="2">ADC Module x1<br> <br>or<br> </td><td>L293D IC Chip<br> </td></tr>
</table>

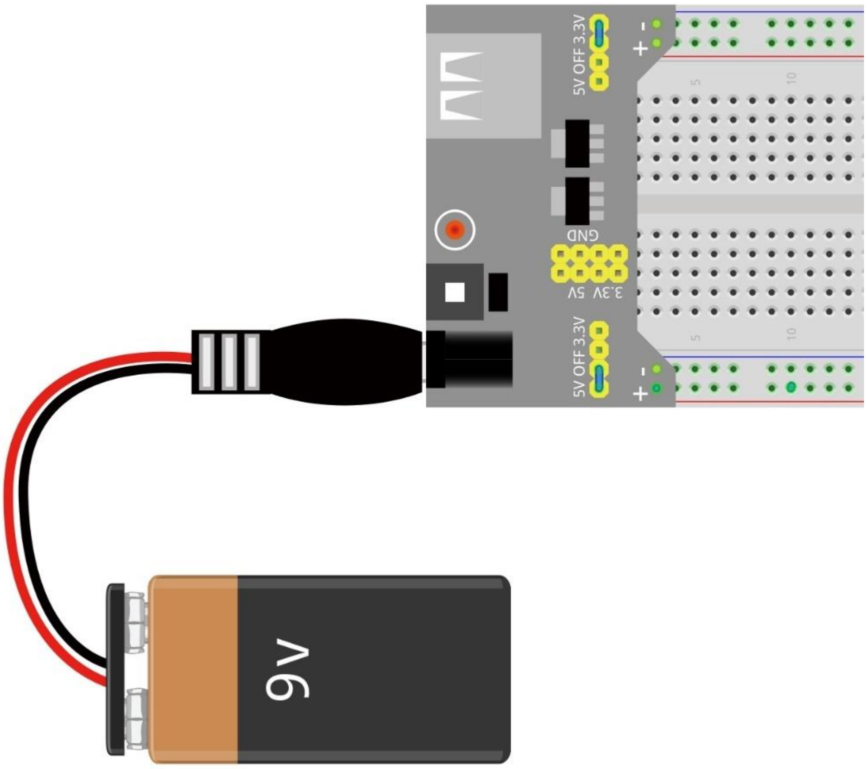
## Component knowledge

### Breadboard Power Module

Breadboard Power Module is an independent circuit board, which can provide independent 5V or 3.3V power to the breadboard when building circuits. It also has built-in power protection to avoid damaging your RPi module. The schematic diagram below identifies the important features of this Power Module:

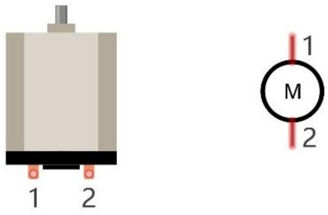


Here is an acceptable connection between Breadboard Power Module and Breadboard using a 9V battery and the provided power harness:

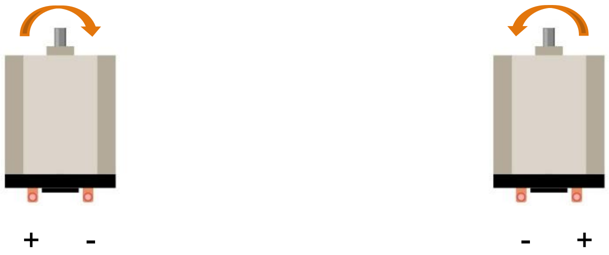


## DC Motor

DC Motor is a device that converts electrical energy into mechanical energy. DC Motors consist of two major parts, a Stator and the Rotor. The stationary part of a DC Motor is the Stator and the part that Rotates is the Rotor. The Stator is usually part of the outer case of motor (if it is simply a pair of permanent magnets), and it has terminals to connect to the power if it is made up of electromagnet coils. Most Hobby DC Motors only use Permanent Magnets for the Stator Field. The Rotor is usually the shaft of motor with 3 or more electromagnets connected to a commutator where the brushes (via the terminals 1 & 2 below) supply electrical power, which can drive other mechanical devices. The diagram below shows a small DC Motor with two terminal pins.

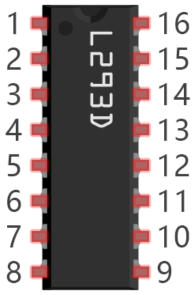


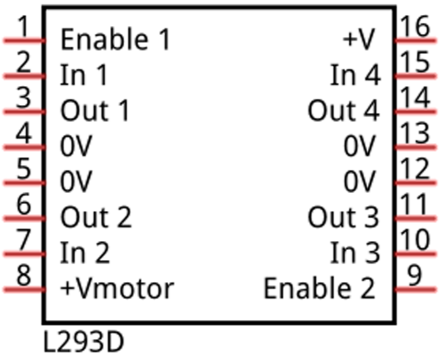
When a DC Motor is connected to a power supply, it will rotate in one direction. If you reverse the polarity of the power supply, the DC Motor will rotate in opposite direction. This is important to note.



## L293D

L293D is an IC Chip (Integrated Circuit Chip) with a 4-channel motor drive. You can drive a Unidirectional DC Motor with 4 ports or a Bi-Directional DC Motor with 2 ports or a Stepper Motor (Stepper Motors are covered later in this Tutorial).





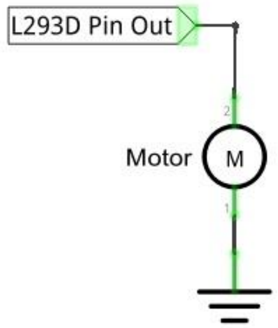
Port description of L293D module is as follows:

| Pin name | Pin number | Description |
|---|---|---|
| In x | 2, 7, 10, 15 | Channel x digital signal input pin |
| Out x | 3, 6, 11, 14 | Channel x output pin, input high or low level according to In x pin, gets connected to +Vmotor or 0V |
| Enable1 | 1 | Channel 1 and Channel 2 enable pin, high level enable |
| Enable2 | 9 | Channel 3 and Channel 4 enable pin, high level enable |
| 0V | 4, 5, 12, 13 | Power Cathode (GND) |
| +V | 16 | Positive Electrode (VCC) of power supply, supply voltage 4.5~36V |
| +Vmotor | 8 | Positive Electrode of load power supply, provide power supply for the Out pin x, the supply voltage is +V~36V |

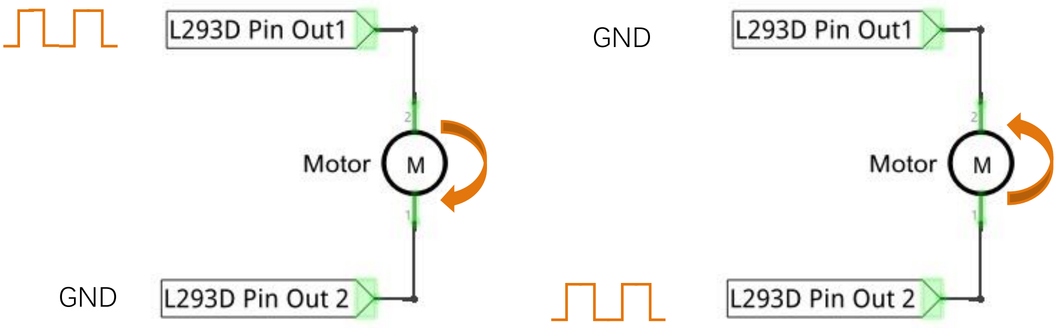
For more details, please see the datasheet for this IC Chip.

When using the L293D to drive a DC Motor, there are usually two connection options.

The following connection option uses one channel of the L239D, which can control motor speed through the PWM, However the motor then can only rotate in one direction.



The following connection uses two channels of the L239D: one channel outputs the PWM wave, and the other channel connects to GND. Therefore, you can control the speed of the motor. When these two channel signals are exchanged, not only controls the speed of motor, but also can control the direction of the motor.

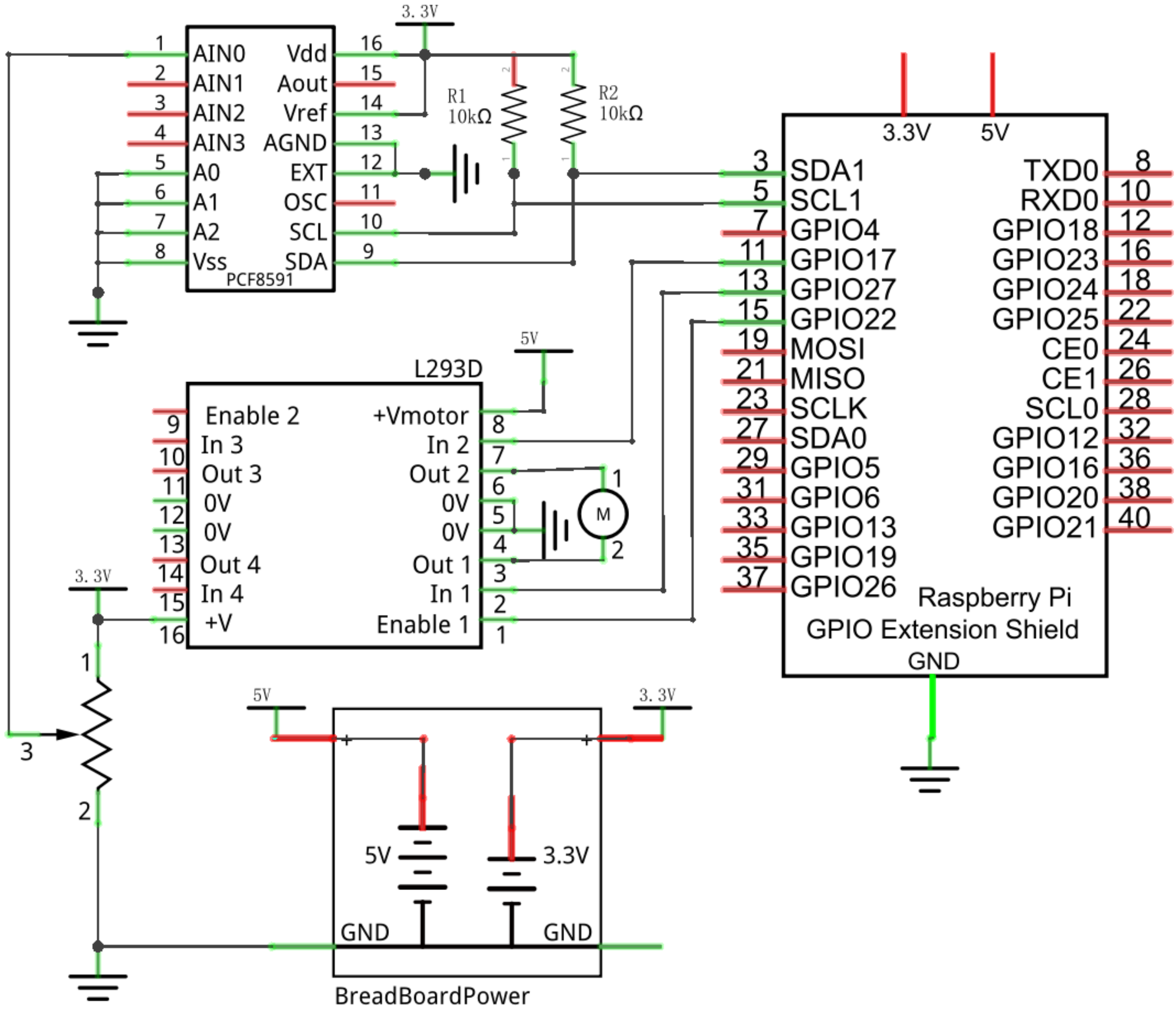


In practical use the motor is usually connected to channel 1 and by outputting different levels to in1 and in2 to control the rotational direction of the motor, and output to the PWM wave to Enable1 port to control the motor's rotational speed. If the motor is connected to channel 3 and 4 by outputting different levels to in3 and in4 to control the motor's rotation direction, and output to the PWM wave to Enable2 pin to control the motor's rotational speed.

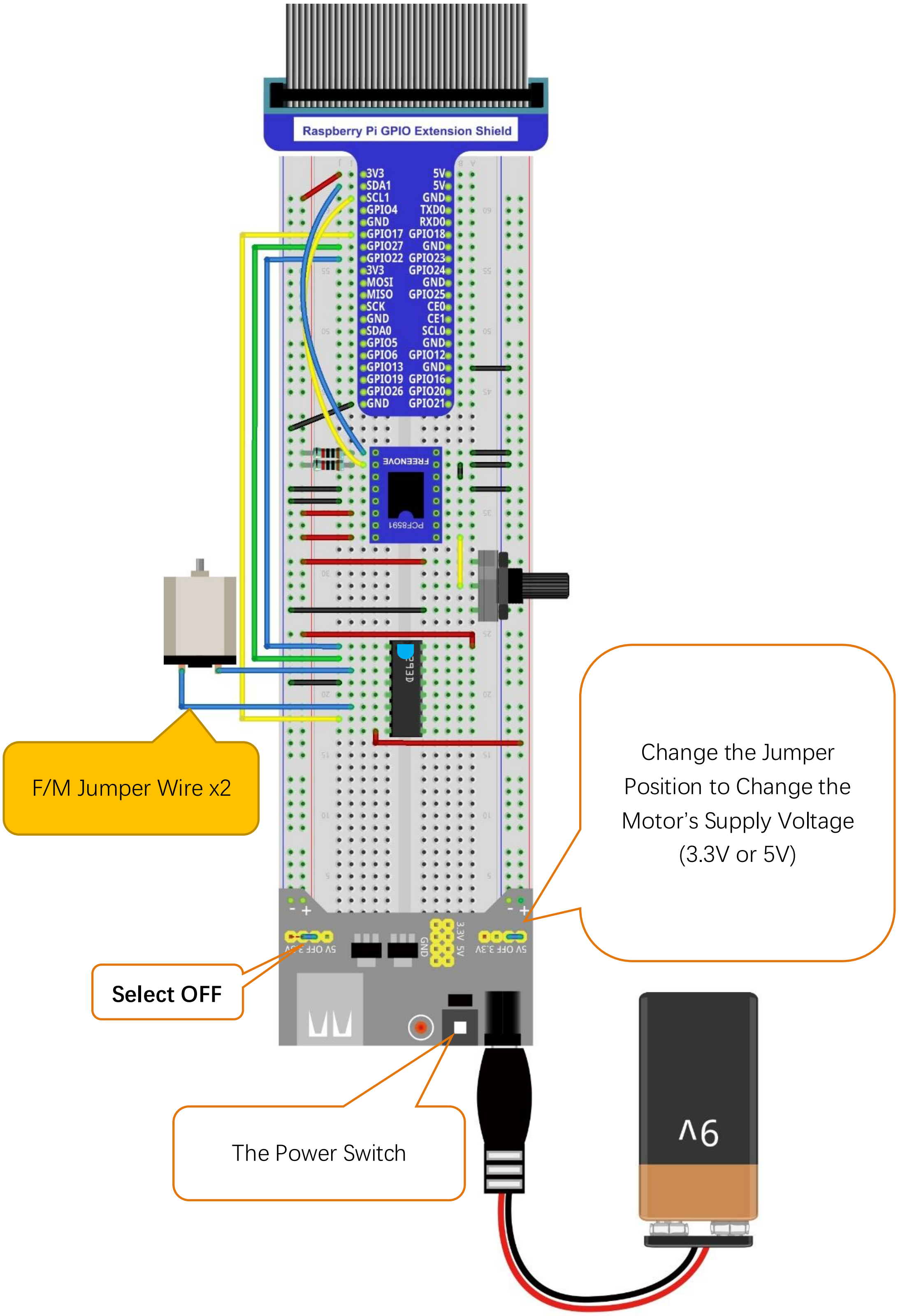
## Circuit with PCF8591

Use caution when connecting this circuit because the DC Motor is a high-power component. **Do not use the power provided by the RPi to power the motor directly, as this may cause permanent damage to your RPi!** The logic circuit can be powered by the RPi's power or an external power supply, which should share a common ground with RPi.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**



F/M Jumper Wire x2

Change the Jumper Position to Change the Motor's Supply Voltage (3.3V or 5V)

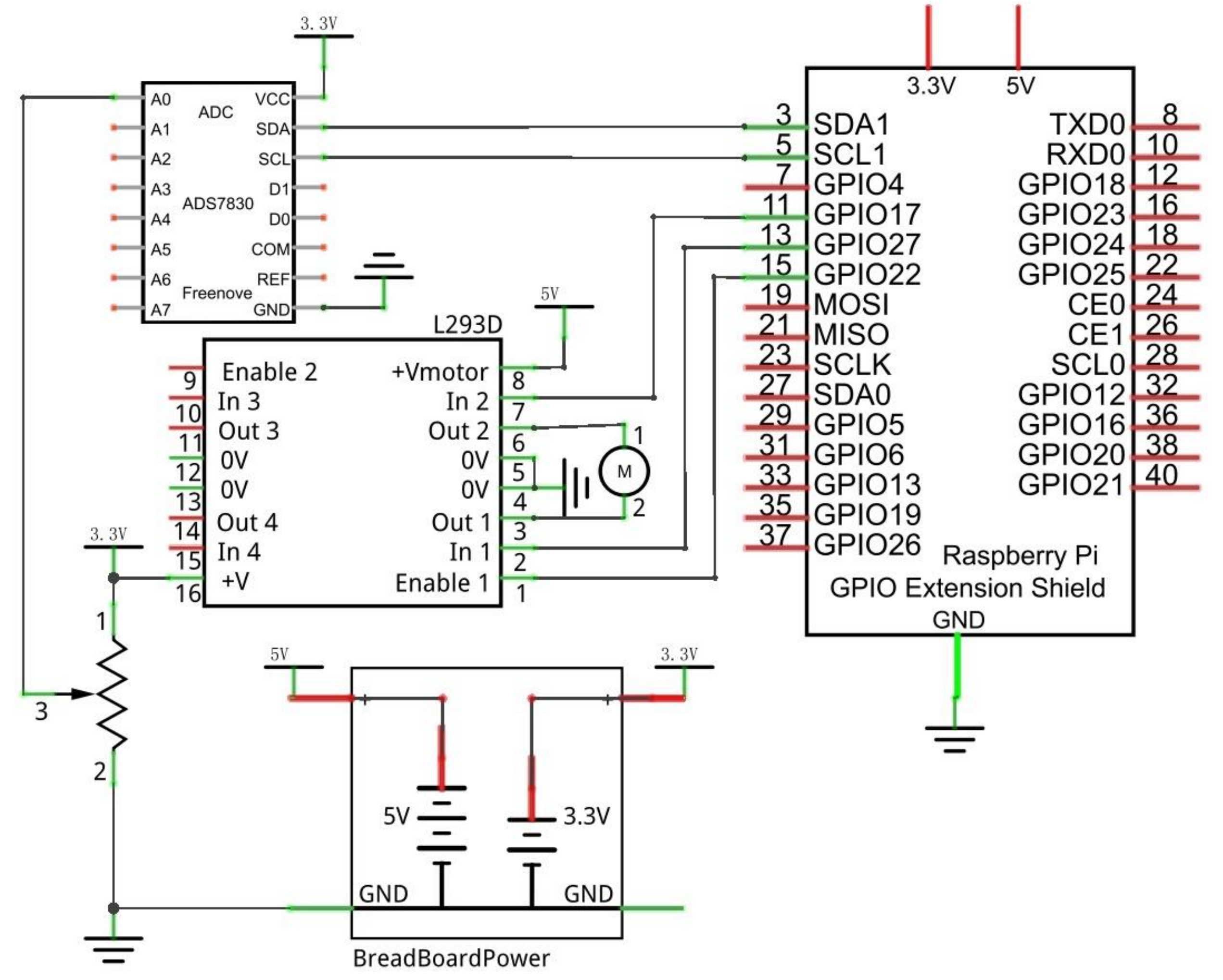
Select OFF

The Power Switch

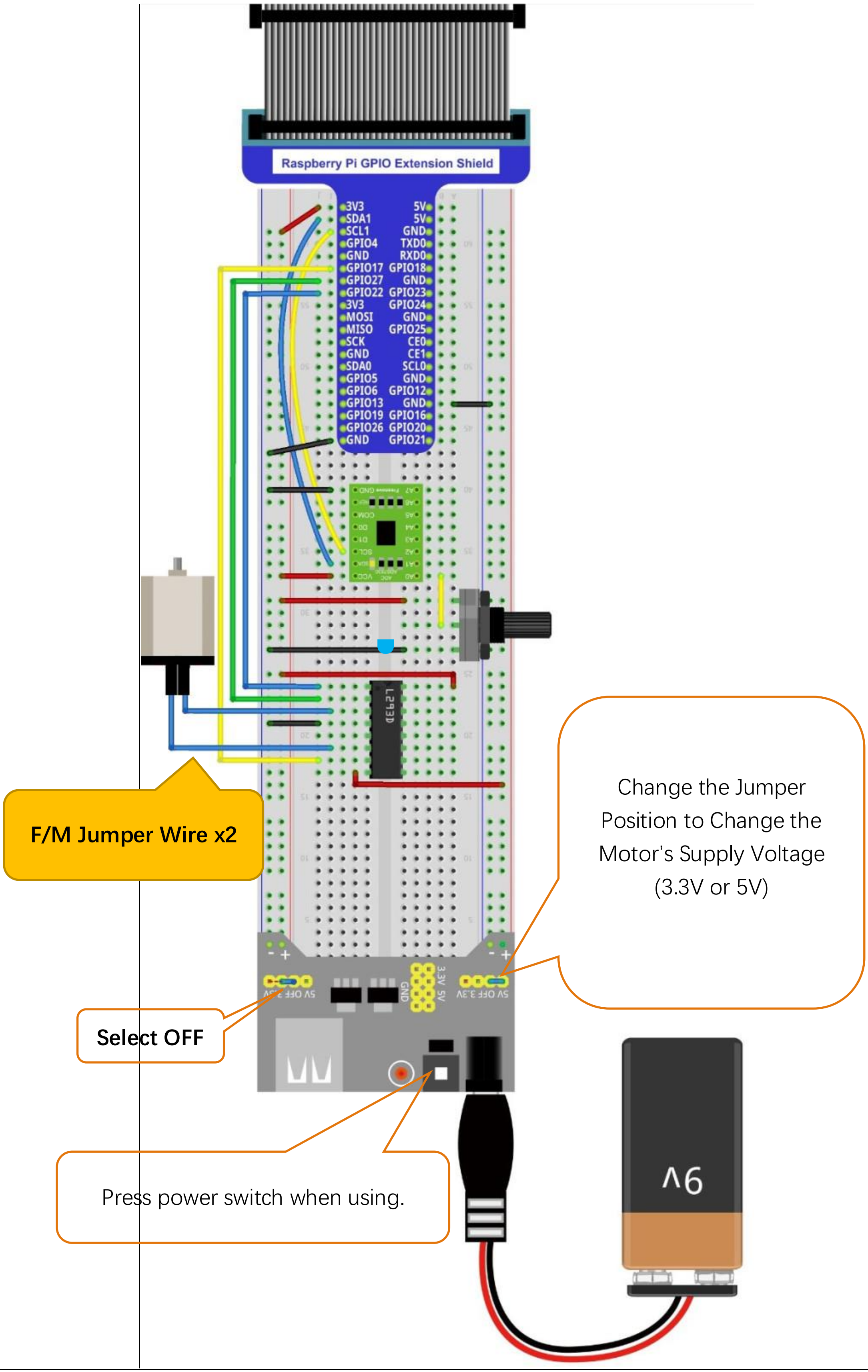
## Circuit with ADS7830

Use caution when connecting this circuit because the DC Motor is a high-power component. **Do not use the power provided by the RPi to power the motor directly, as this may cause permanent damage to your RPi!** The logic circuit can be powered by the RPi's power or an external power supply, which should share a common ground with RPi.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: **support@freenove.com**



F/M Jumper Wire x2

Change the Jumper Position to Change the Motor's Supply Voltage (3.3V or 5V)

Select OFF

Press power switch when using.

# Code

In code for this project, first read the ADC value and then control the rotation direction and speed of the DC Motor according to the value of the ADC.

C Code 13.1.1 Motor

If you did not configure I2C, please refer to Chapter 7. If you did, please continue.

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 13.1.1_Motor directory of the C code.

```
cd ~/Freenove_Kit/Code/C_Code/13.1.1_Motor
```

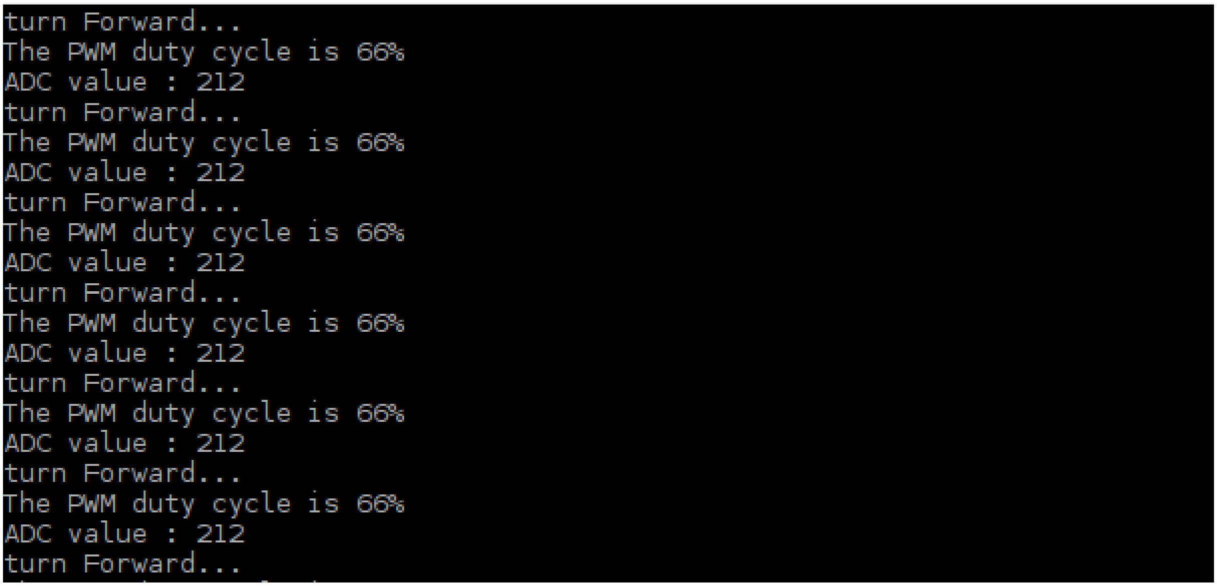
2. Use the following command to compile "Motor.cpp" and generate the executable file "Motor".

```
g++ Motor.cpp -o Motor -lwiringPi -lADCDevice
```

3. Then run the generated file "Motor".

```
sudo ./Motor
```

After the program is executed, you can use the Potentiometer to control the DC Motor. When the Potentiometer is at the midpoint position, the DC Motor will STOP, and when the Potentiometer is turned in either direction of this midpoint, the DC Motor speed increases until it reaches the endpoint where the DC Motor achieves its maximum speed. When the Potentiometer is turned "Left" of the midpoint the DC Motor will ROTATE in one direction and when turned "Right" the DC Motor will ROTATE in the opposite direction. You will also see the ADC value of the potentiometer displayed in the Terminal with the motor direction and the PWM duty cycle used to control the DC Motor's speed.



```
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
The PWM duty cycle is 66%
ADC value : 212
turn Forward...
```

The following is the code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <softPwm.h>
#include <math.h>
#include <stdlib.h>
#include <ADCDevice.hpp>

```

```
#define motorPin1    2        //define the pin connected to L293D
#define motorPin2    0
#define enablePin    3

ADCDevice *adc;  // Define an ADC Device class object

//Map function: map the value from a range to another range.
long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
//motor function: determine the direction and speed of the motor according to the ADC
void motor(int ADC){
    int value = ADC -128;
    if(value>0){
        digitalWrite(motorPin1,HIGH);
        digitalWrite(motorPin2,LOW);
        printf("turn Forward...\n");
    }
    else if (value<0){
        digitalWrite(motorPin1,LOW);
        digitalWrite(motorPin2,HIGH);
        printf("turn Back...\n");
    }
    else {
        digitalWrite(motorPin1,LOW);
        digitalWrite(motorPin2,LOW);
        printf("Motor Stop...\n");
    }
    softPwmWrite(enablePin,map(abs(value),0,128,0,100));
    printf("The PWM duty cycle is %d%%\n",abs(value)*100/127);//print the PMW duty cycle
}
int main(void){
    adc = new ADCDevice();
    printf("Program is starting ... \n");

    if(adc->detectI2C(0x48)){    // Detect the pcf8591.
        delete adc;                 // Free previously pointed memory
        adc = new PCF8591();    // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;                // Free previously pointed memory
        adc = new ADS7830();      // If detected, create an instance of ADS7830.
    }
    else{
        printf("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        return -1;
    }
    wiringPiSetup();
    pinMode(enablePin,OUTPUT);//set mode for the pin
    pinMode(motorPin1,OUTPUT);
    pinMode(motorPin2,OUTPUT);
    softPwmCreate(enablePin,0,100);//define PMW pin
    while(1){
        int value = adc->analogRead(0);  //read analog value of A0 pin
        printf("ADC value : %d \n",value);
        motor(value);        //make the motor rotate with speed(analog value of A0 pin)
        delay(100);
    }
    return 0;
}
```

Now that we have familiarity with reading ADC values, let's learn the subfunction void motor (int ADC): first, compare the ADC value with 128 (value corresponding to midpoint). When the current ADC value is higher, motoRPin1 outputs high level and motoRPin2 outputs low level to control the DC Motor to run in the "Forward" Rotational Direction. When the current ADC value is lower, motoRPin1 outputs low level and motoRPin2 outputs high level to control the DC Motor to run in the "Reverse" Rotational Direction. When the ADC value is equal to 128, motoRPin1 and motoRPin2 output low level, the motor STOPS. Then determine the PWM duty cycle according to the difference (delta) between ADC value and 128. Because the absolute delta value stays within 0-128, we need to use the map() subfunction mapping the delta value to a range of 0-255. Finally, we see a display of the duty cycle in Terminal.

```
void motor(int ADC){
    int value = ADC -128;
    if(value>0){
        digitalWrite(motoRPin1,HIGH);
        digitalWrite(motoRPin2,LOW);
        printf("turn Forward...\n");
    }
    else if (value<0){
        digitalWrite(motoRPin1,LOW);
        digitalWrite(motoRPin2,HIGH);
        printf("turn Backward...\n");
    }
    else {
        digitalWrite(motoRPin1,LOW);
        digitalWrite(motoRPin2,LOW);
        printf("Motor Stop...\n");
    }
    softPwmWrite(enablePin,map(abs(value),0,128,0,100));
    printf("The PWM duty cycle is %d%%\n",abs(value)*100/127);// print out PWM duty cycle.
}
```

### Python Code 13.1.1 Motor

If you did not configure I2C and install Smbus, please refer to Chapter 7. If you did, please Continue.
First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

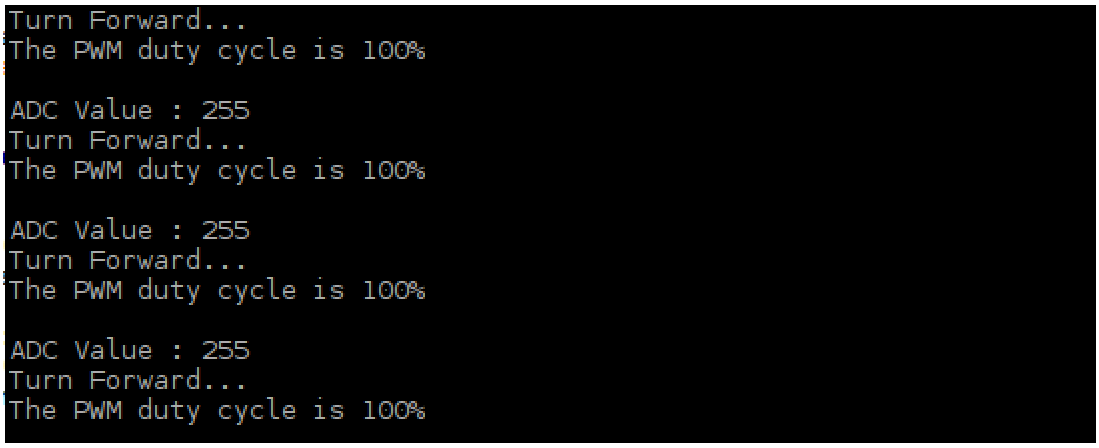
1. Use cd command to enter 13.1.1_Motor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/13.1.1_Motor
```

2. Use python command to execute the Python code “Motor.py”.

```
python Motor.py
```

After the program is executed, you can use the Potentiometer to control the DC Motor. When the Potentiometer is at the midpoint position, the DC Motor will STOP, and when the Potentiometer is turned in either direction of this midpoint, the DC Motor speed increases until it reaches the endpoint where the DC Motor achieves its maximum speed. When the Potentiometer is turned “Left” of the midpoint the DC Motor will ROTATE in one direction and when turned “Right” the DC Motor will ROTATE in the opposite direction. You will also see the ADC value of the potentiometer displayed in the Terminal with the motor direction and the PWM duty cycle used to control the DC Motor’s speed.



```
Turn Forward...
The PWM duty cycle is 100%

ADC Value : 255
Turn Forward...
The PWM duty cycle is 100%

ADC Value : 255
Turn Forward...
The PWM duty cycle is 100%

ADC Value : 255
Turn Forward...
The PWM duty cycle is 100%
```

The following is the code:

```python
import RPi.GPIO as GPIO
import time
from ADCDevice import *

# define the pins connected to L293D
motoRPin1 = 13
motoRPin2 = 11
enablePin = 15
adc = ADCDevice() # Define an ADCDevice class object

def setup():
    global adc
    if(adc.detectI2C(0x48)): # Detect the pcf8591.
        adc = PCF8591()
    elif(adc.detectI2C(0x4b)): # Detect the ads7830
        adc = ADS7830()
    else:
        print("No correct I2C address found, \n"
        "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
        "Program Exit. \n");
        exit(-1)
    global p
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(motoRPin1,GPIO.OUT)   # set pins to OUTPUT mode
    GPIO.setup(motoRPin2,GPIO.OUT)
    GPIO.setup(enablePin,GPIO.OUT)

    p = GPIO.PWM(enablePin,1000) # creat PWM and set Frequence to 1KHz
    p.start(0)

# mapNUM function: map the value from a range of mapping to another range.
def mapNUM(value,fromLow,fromHigh,toLow,toHigh):
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

# motor function: determine the direction and speed of the motor according to the input
ADC value input
def motor(ADC):
    value = ADC -128
    if (value > 0):  # make motor turn forward
        GPIO.output(motoRPin1,GPIO.HIGH)  # motoRPin1 output HIHG level
        GPIO.output(motoRPin2,GPIO.LOW)   # motoRPin2 output LOW level
        print ('Turn Forward...')
    elif (value < 0): # make motor turn backward
        GPIO.output(motoRPin1,GPIO.LOW)
        GPIO.output(motoRPin2,GPIO.HIGH)
        print ('Turn Backward...')
    else :
        GPIO.output(motoRPin1,GPIO.LOW)
        GPIO.output(motoRPin2,GPIO.LOW)
        print ('Motor Stop...')
    p.start(mapNUM(abs(value),0,128,0,100))
    print ('The PWM duty cycle is %d%%\n'%(abs(value)*100/127))   # print PMW duty cycle.

def loop():
    while True:
        value = adc.analogRead(0) # read ADC value of channel 0
        print ('ADC Value : %d'%(value))
        motor(value)
        time.sleep(0.01)

def destroy():
    GPIO.cleanup()

if __name__ == '__main__':  # Program entrance
    print ('Program is starting ... ')
    setup()
    try:
        loop()
    except KeyboardInterrupt: # Press ctrl-c to end the program.
        destroy()
```

Now that we have familiarity with reading ADC values, let's learn the subfunction void motor (int ADC): first, compare the ADC value with 128 (value corresponding to midpoint). When the current ADC value is higher, motoRPin1 outputs high level and motoRPin2 outputs low level to control the DC Motor to run in the "Forward" Rotational Direction. When the current ADC value is lower, motoRPin1 outputs low level and motoRPin2 outputs high level to control the DC Motor to run in the "Reverse" Rotational Direction. When the ADC value is equal to 128, motoRPin1 and motoRPin2 output low level, the motor STOPS. Then determine the PWM duty cycle according to the difference (delta) between ADC value and 128. Because the absolute delta value stays within 0-128. We need to use the map() subfunction mapping the delta value to a range of 0-255. Finally, we see a display of the duty cycle in Terminal.

```
def motor(ADC):
    value = ADC -128
    if (value > 0):
        GPIO.output(motoRPin1,GPIO.HIGH)
        GPIO.output(motoRPin2,GPIO.LOW)
        print ('Turn Forward...')
    elif (value < 0):
        GPIO.output(motoRPin1,GPIO.LOW)
        GPIO.output(motoRPin2,GPIO.HIGH)
        print ('Turn Backward...')
    else :
        GPIO.output(motoRPin1,GPIO.LOW)
        GPIO.output(motoRPin2,GPIO.LOW)
        print ('Motor Stop...')
    p.start(mapNUM(abs(value),0,128,0,100))
    print ('The PWM duty cycle is %d%%\n'%(abs(value)*100/127))   #print PMW duty cycle.
```

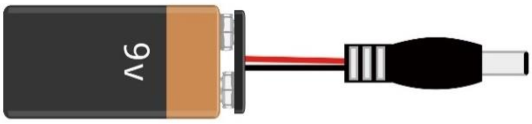
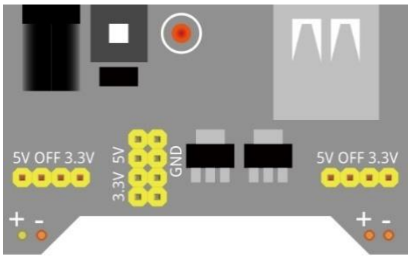
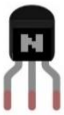
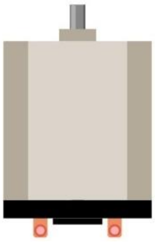
# Chapter 14 Relay & Motor

In this chapter, we will learn a kind of special switch module, Relay Module.

## Project 14.1.1 Relay & Motor

In this project, we will use a Push Button Switch indirectly to control the DC Motor via a Relay.

### Component List

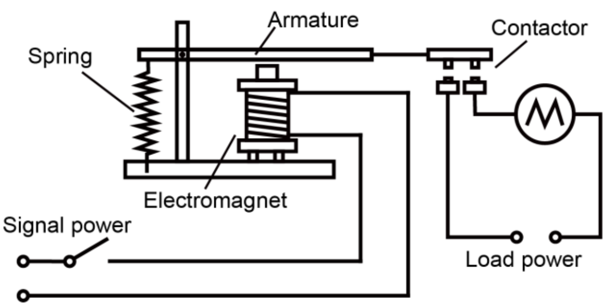
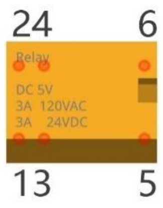
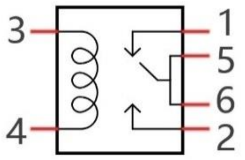
<table>
<tr><td colspan="3">Raspberry Pi (with 40 GPIO) x1<br>GPIO Expansion Board & Ribbon Cable x1<br>Breadboard x1</td><td colspan="3">Jumper Wire x11<br> </td></tr>
<tr><td colspan="6">9V battery (prepared by yourself) & battery line<br><br> </td></tr>
<tr><td colspan="3">Breadboard Power module x1<br><br> </td><td>Resistor 10kΩ  x2<br> </td><td>Resistor 1kΩ  x1<br> </td><td>Resistor 220Ω  x1<br> </td></tr>
<tr><td>NPN transistor x1<br><br> </td><td>Relay x1<br> <br> </td><td>Motor x1<br></td><td>Push button x1<br> </td><td>LED x1<br> </td><td>Diode x1<br> </td></tr>
</table>

## Component knowledge

### Relay

Relays are a type of Switch that open and close circuits electromechanically or electronically. Relays control one electrical circuit by opening and closing contacts in another circuit using an electromagnet to initiate the Switch action. When the electromagnet is energized (powered), it will attract internal contacts completing a circuit, which act as a Switch. Many times Relays are used to allow a low powered circuit (and a small low amperage switch) to safely turn ON a larger more powerful circuit. They are commonly found in automobiles, especially from the ignition to the starter motor.

The following is a basic diagram of a common Relay and the image and circuit symbol diagram of the 5V relay used in this project:

| Diagram | Feature： | Symbol |
|---|---|---|
|   Armature<br>Contactor<br>Spring<br>Electromagnet<br>Signal power<br>Load power<br>M |   24<br>6<br>Relay<br>DC 5V<br>3A 120VAC<br>3A 24VDC<br>13<br>5 |  |

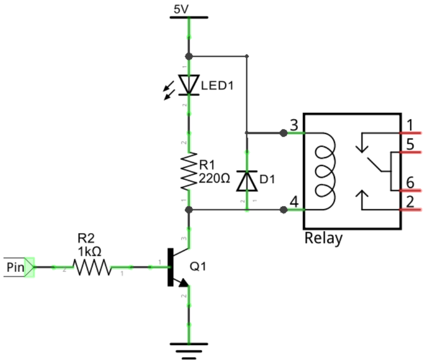
Pin 5 and pin 6 are internally connected to each other. When the coil pin3 and pin 4 are connected to a 5V power supply, pin 1 will be disconnected from pins 5 & 6 and pin 2 will be connected to pins 5 & 6. Pin 1 is called Closed End and pin 2 is called the Open End.

### Inductor

The symbol of Inductance is "L" and the unit of inductance is the "Henry" (H). Here is an example of how this can be encountered: 1H=1000mH, 1mH=1000μH.

An Inductor is a passive device that stores energy in its Magnetic Field and returns energy to the circuit whenever required. An Inductor is formed by a Cylindrical Core with many Turns of conducting wire (usually copper wire). Inductors will hinder the changing current passing through it. When the current passing through the Inductor increases, it will attempt to hinder the increasing movement of current; and when the current passing through the inductor decreases, it will attempt to hinder the decreasing movement of current. So the current passing through an Inductor is not transient.

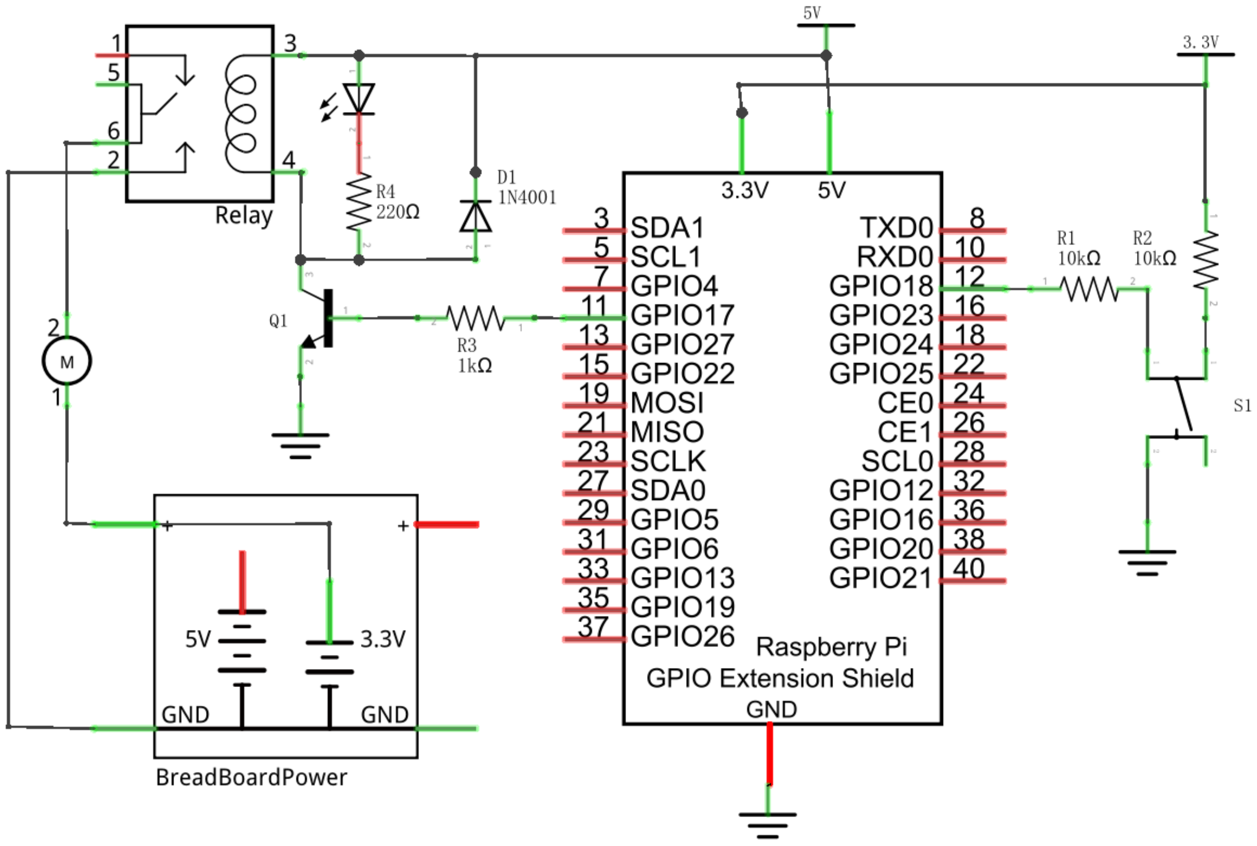
The circuit for a Relay is as follows: The coil of Relay can be equivalent to an Inductor, when a Transistor is present in this coil circuit it can disconnect the power to the relay, the current in the Relay's coil does not stop immediately, which affects the power supply adversely. To remedy this, diodes in parallel are placed on both ends of the Relay coil pins in opposite polar direction. Having the current pass through the diodes will avoid any adverse effect on the power supply.



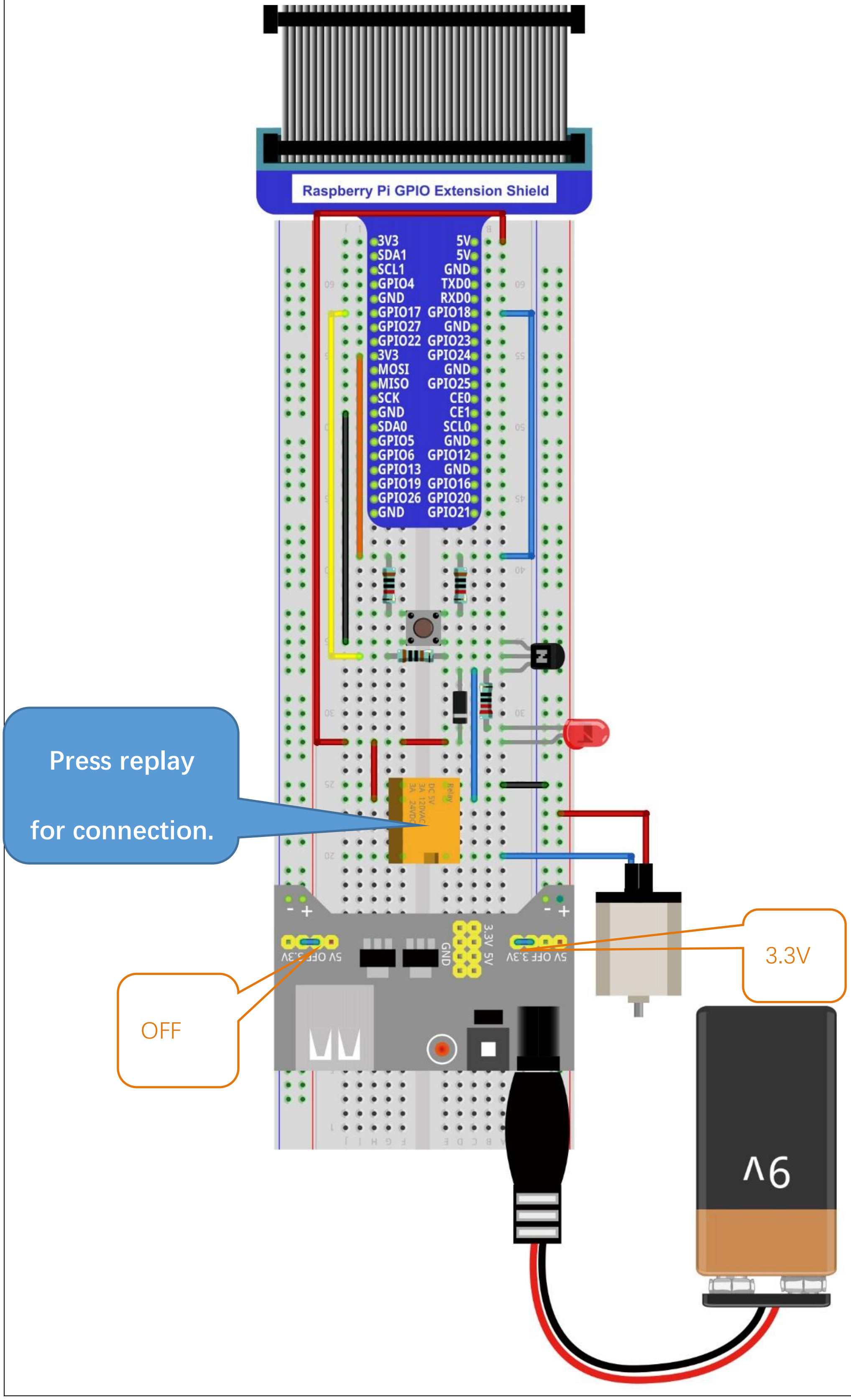
## Circuit

Use caution with the power supply voltage needed for the components in this circuit. The Relay requires a power supply voltage of 5V, and the DC Motor only requires 3.3V. Additionally, there is an LED present, which acts as an indicator (ON or OFF) for the status of the Relay's active status.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



## Code

The project code is in the same as we used earlier in the Table Lamp project. Pressing the Push Button Switch activates the transistor. Because the Relay and the LED are connected in parallel, they will be powered ON at the same time. Press the Push Button Switch again will turn them both OFF.

### C Code 14.1.1 Relay

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 14.1.1_Relay directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/14.1.1_Relay
```

2. Use following command to compile "Relay.c" and generate executable file "Relay".

```
gcc Relay.c -o Relay -lwiringPi
```

3. Run the generated file "Relay".

```
sudo ./Relay
```

After the program is executed, pressing the Push Button Switch activates the Relay (the internal switch is closed), which powers the DC Motor to rotate and simultaneously powers the LED to turn ON. If you press the Push Button Switch again, the Relay is deactivated (the internal switch opens), the Motor STOPS and the LED turns OFF.

The following is the program code:

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

#define relayPin     0      //define the relayPin
#define buttonPin 1         //define the buttonPin
int relayState=LOW;         //store the State of relay
int buttonState=HIGH; //store the State of button
int lastbuttonState=HIGH;//store the lastState of button
long lastChangeTime;  //store the change time of button state
long captureTime=50;  //set the button state stable time
int reading;
int main(void)
{
    printf("Program is starting...\n");

    wiringPiSetup();

    pinMode(relayPin, OUTPUT);
    pinMode(buttonPin, INPUT);
    pullUpDnControl(buttonPin, PUD_UP);  //pull up to high level
    while(1){
        reading = digitalRead(buttonPin); //read the current state of button
        if( reading != lastbuttonState){  //if the button state changed ,record the time
point
```

```
            lastChangeTime = millis();
        }
        //if changing-state of the button last beyond the time we set, we considered that 
        //the current button state is an effective change rather than a buffeting
        if(millis() - lastChangeTime > captureTime){
            //if button state is changed, update the data.
            if(reading != buttonState){
                buttonState = reading;
                //if the state is low, the action is pressing.
                if(buttonState == LOW){
                    printf("Button is pressed!\n");
                    relayState = !relayState;
                    if(relayState){
                        printf("turn on relay ...\n");
                    }
                    else {
                        printf("turn off relay ...\n");
                    }
                }
                //if the state is high, the action is releasing.
                else {
                    printf("Button is released!\n");
                }
            }
        }
        digitalWrite(relayPin,relayState);
        lastbuttonState = reading;
    }

    return 0;
}
```

The project code is in the same as we used earlier in the Table Lamp project.

### Python Code 14.1.1 Relay

First observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 14.1.1_Relay directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/14.1.1_Relay
```

2. Use python command to execute code "Relay.py".

```
python Relay.py
```

After the program is executed, press the button, then the relay is opened, the Motor starts to rotate and LED turns ON

### Python Code 14.1.1 Relay

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

3. Use cd command to enter 14.1.1_Relay directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/14.1.1_Relay
```

4. Use Python command to execute code "Relay.py".

```
python Relay.py
```

After the program is executed, pressing the Push Button Switch activates the Relay (the internal switch is closed), which powers the DC Motor to rotate and simultaneously powers the LED to turn ON. If you press the Push Button Switch again, the Relay is deactivated (the internal switch opens), the Motor STOPS and the LED turns OFF.

The following is the program code:

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

relayPin = 11     # define the relayPin
buttonPin = 12    # define the buttonPin
debounceTime = 50

def setup():
    GPIO.setmode(GPIO.BOARD)
    GPIO.setup(relayPin, GPIO.OUT)   # set relayPin to OUTPUT mode
    GPIO.setup(buttonPin, GPIO.IN)   # set buttonPin to INTPUT mode

def loop():
    relayState = False
    lastChangeTime = round(time.time()*1000)
    buttonState = GPIO.HIGH
    lastButtonState = GPIO.HIGH
    reading = GPIO.HIGH
    while True:
        reading = GPIO.input(buttonPin)
        if reading != lastButtonState :
            lastChangeTime = round(time.time()*1000)
        if ((round(time.time()*1000) - lastChangeTime) > debounceTime):
            if reading != buttonState :
                buttonState = reading;
                if buttonState == GPIO.LOW:
                    print("Button is pressed!")
                    relayState = not relayState
                    if relayState:
                        print("Turn on relay ...")
                    else :
                        print("Turn off relay ... ")
                else :
                    print("Button is released!")
        GPIO.output(relayPin,relayState)
        lastButtonState = reading # lastButtonState store latest state

def destroy():
    GPIO.cleanup()

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:   # Press ctrl-c to end the program.
        destroy()
```

The project code is in the same as we used earlier in the Table Lamp project.

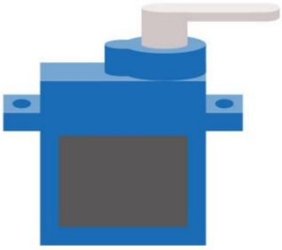
# Chapter 15 Servo

Previously, we learned how to control the speed and rotational direction of a DC Motor. In this chapter, we will learn about Servos which are a rotary actuator type motor that can be controlled rotate to specific angles.

## Project 15.1 Servo Sweep

First, we need to learn how to make a Servo rotate.

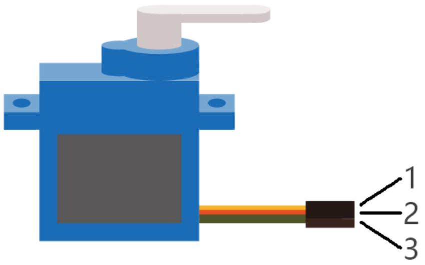
### Component List

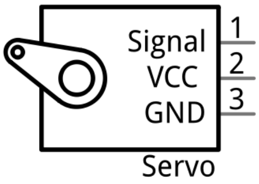
| Raspberry Pi (with 40 GPIO) x1<br>GPIO Expansion Board & Ribbon Cable x1<br>Breadboard x1 | Jumper Wire x3<br> |
|---|---|
| Servo x1<br> | |

## Component knowledge

### Servo

Servo is a compact package which consists of a DC Motor, a set of reduction gears to provide torque, a sensor and control circuit board. Most Servos only have a 180-degree range of motion via their "horn". Servos can output higher torque than a simple DC Motor alone and they are widely used to control motion in model cars, model airplanes, robots, etc. Servos have three wire leads which usually terminate to a male or female 3-pin plug. Two leads are for electric power: Positive (2-VCC, Red wire), Negative (3-GND, Brown wire), and the signal line (1-Signal, Orange wire) as represented in the Servo provided in your Kit.





We will use a 50Hz PWM signal with a duty cycle in a certain range to drive the Servo. The lasting time 0.5ms-2.5ms of PWM single cycle high level corresponds to the Servo angle 0 degrees - 180 degree linearly. Part of the corresponding values are as follows:

Note: the lasting time of high level corresponding to the servo angle is absolute instead of accumulating. For example, the high level time lasting for 0.5ms correspond to the 0 degree of the servo. If the high level time lasts for another 1ms, the servo rotates to 45 degrees.

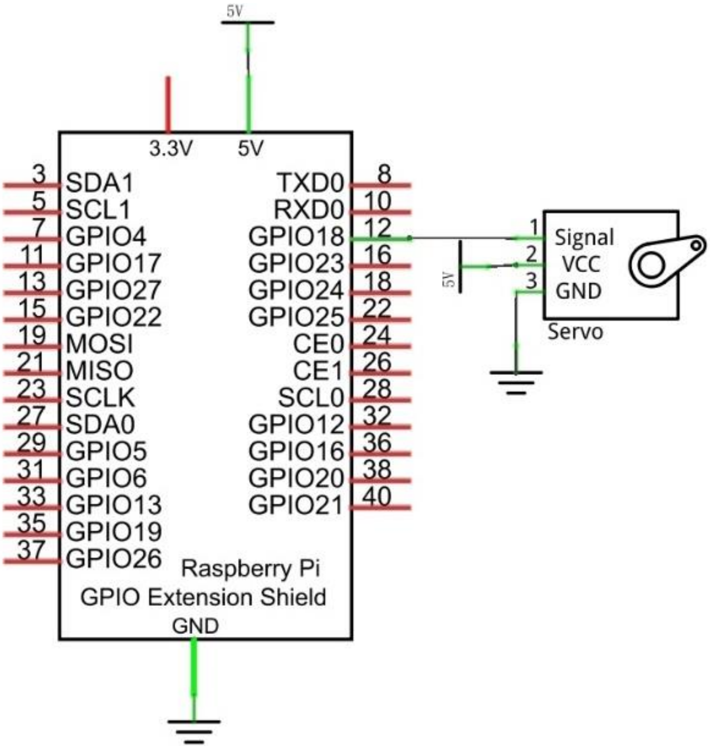
| High level time | Servo angle |
|---|---|
| 0.5ms | 0 degree |
| 1ms | 45 degree |
| 1.5ms | 90 degree |
| 2ms | 135 degree |
| 2.5ms | 180 degree |

When you change the Servo signal value, the Servo will rotate to the designated angle.

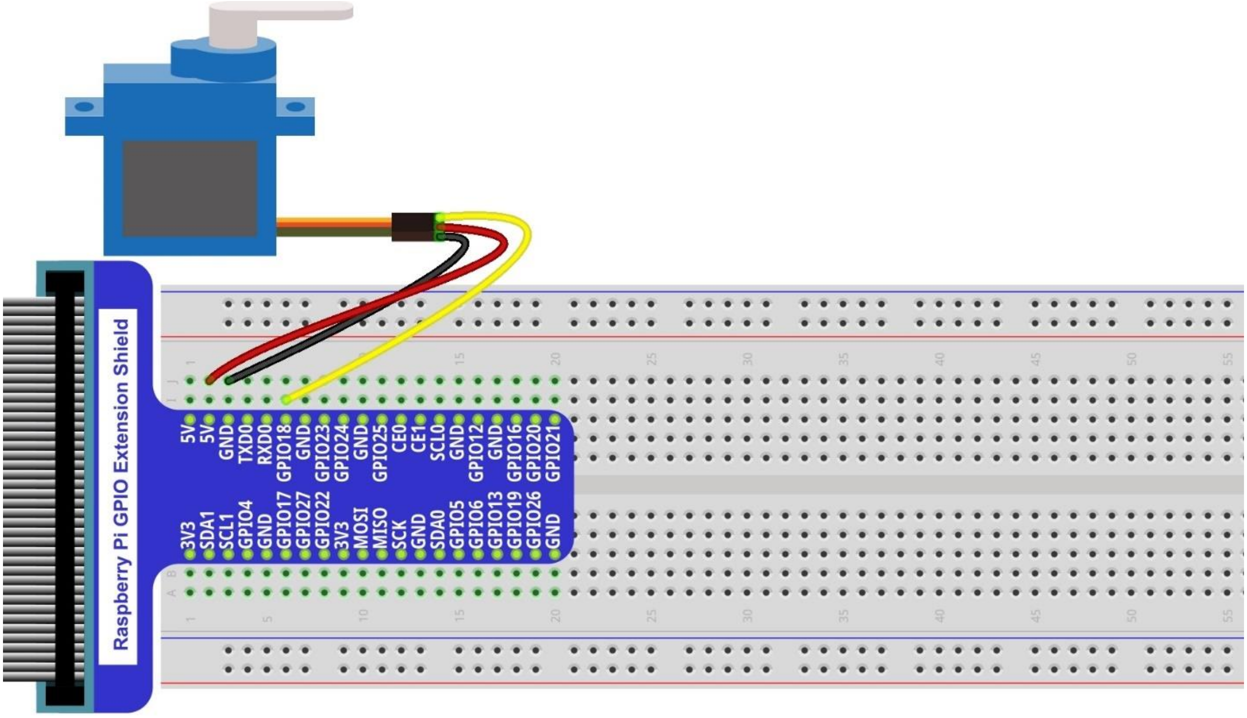
## Circuit

Use caution when supplying power to the Servo it should be 5V. Make sure you do not make any errors when connecting the Servo to the power supply.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



## Code

In this project, we will make a Servo rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.

C Code 15.1.1 Sweep

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 15.1.1_Sweep directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/15.1.1_Sweep
```

2. Use following command to compile "Sweep.c" and generate executable file "Sweep".

```
gcc Sweep.c -o Sweep -lwiringPi
```

3. Run the generated file "Sweep".

```
sudo ./Sweep
```

After the program is executed, the Servo will rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.

The following is the program code:

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>
#define OFFSET_MS 3     //Define the unit of servo pulse offset: 0.1ms
#define SERVO_MIN_MS 5+OFFSET_MS        //define the pulse duration for minimum angle of servo
#define SERVO_MAX_MS 25+OFFSET_MS       //define the pulse duration for maximum angle of servo

#define servoPin    1       //define the GPIO number connected to servo
long map(long value,long fromLow,long fromHigh,long toLow,long toHigh){
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow;
}
void servoInit(int pin){        //initialization function for servo PMW pin
    softPwmCreate(pin,  0, 200);
}
void servoWrite(int pin, int angle){    //Specific a certain rotation angle (0-180) for the
servo
    if(angle > 180)
        angle = 180;
    if(angle < 0)
        angle = 0;
    softPwmWrite(pin,map(angle,0,180,SERVO_MIN_MS,SERVO_MAX_MS));
}
void servoWriteMS(int pin, int ms){     //specific the unit for pulse(5-25ms) with specific
duration output by servo pin: 0.1ms
    if(ms > SERVO_MAX_MS)
        ms = SERVO_MAX_MS;
    if(ms < SERVO_MIN_MS)
```

```
        ms = SERVO_MIN_MS;
    softPwmWrite(pin,ms);
}

int main(void)
{
    int i;

    printf("Program is starting ...\n");

    wiringPiSetup();
    servoInit(servoPin);        //initialize PMW pin of servo
    while(1){
        for(i=SERVO_MIN_MS;i<SERVO_MAX_MS;i++){  //make servo rotate from minimum angle to 
maximum angle
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(500);
        for(i=SERVO_MAX_MS;i>SERVO_MIN_MS;i--){  //make servo rotate from maximum angle to 
minimum angle
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(500);
    }
    return 0;
}
```

A 50 Hz pulse for a 20ms cycle is required to control the Servo. In function **softPwmCreate** (int pin, int initialValue, int pwmRange), the unit of the third parameter pwmRange is 100US, specifically 0.1ms. In order to get the PWM with a 20ms cycle, the pwmRange shoulde be set to 200. So in the subfunction of servoInit (), we create a PWM pin with a pwmRange of 200.

```
void servoInit(int pin){        //initialization function for servo PWM pin
    softPwmCreate(pin,  0, 200);
}
```

Since 0-180 degrees of the Servo's motion corresponds to the PWM pulse width of 0.5-2.5ms, with a PwmRange of 200 ms. We then need the function **softPwmWrite** (int pin, int value) and the scope 5-25 of the parameter values to correspond to 0-180 degrees' motion of the Servo. What's more, the number written in subfunction **servoWriteMS** () should be within the range of 5-25. However, in practice, due to the inherent error manufactured into each Servo, the pulse width will have a deviation. So we need to define a minimum and maximum pulse width and an error offset (this is essential in robotics).

```
#define OFFSET_MS 3     //Define the unit of servo pulse offset: 0.1ms
```

```
#define SERVO_MIN_MS 5+OFFSET_MS        //define the pulse duration for minimum angle of
servo
#define SERVO_MAX_MS 25+OFFSET_MS       //define the pulse duration for maximum angle of
servo
……
void servoWriteMS(int pin, int ms){
    if(ms > SERVO_MAX_MS)
        ms = SERVO_MAX_MS;
    if(ms < SERVO_MIN_MS)
        ms = SERVO_MIN_MS;
    softPwmWrite(pin,ms);
}
```

In subfunction **servoWrite** (), directly input an angle value (0-180 degrees), map the angle to the pulse width and then output it.

```
void servoWrite(int pin, int angle){    //Specif a certain rotation angle (0-180) for the
servo
    if(angle > 180)
        angle = 180;
    if(angle < 0)
        angle = 0;
    softPwmWrite(pin,map(angle,0,180,SERVO_MIN_MS,SERVO_MAX_MS));
}
```

Finally, in the "while" loop of the main function, use two "for" cycle to make servo rotate from 0 degrees to 180 degrees, and then from 180 degrees to 0 degrees.

```
    while(1){
        for(i=SERVO_MIN_MS;i<SERVO_MAX_MS;i++){  //make servo rotate from minimum angle
to maximum angle
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(500);
        for(i=SERVO_MAX_MS;i>SERVO_MIN_MS;i--){  //make servo rotate from maximum angle
to minimum angle
            servoWriteMS(servoPin,i);
            delay(10);
        }
        delay(500);
    }
```

### Python Code 15.1.1 Sweep

First observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 15.1.1_Sweep directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/15.1.1_Sweep
```

2. Use python command to execute code "Sweep.py".

```
python Sweep.py
```

After the program is executed, the Servo will rotate from 0 degrees to 180 degrees and then reverse the direction to make it rotate from 180 degrees to 0 degrees and repeat these actions in an endless loop.

The following is the program code:

```python
import RPi.GPIO as GPIO
import time
OFFSE_DUTY = 0.5        #define pulse offset of servo
SERVO_MIN_DUTY = 2.5+OFFSE_DUTY     #define pulse duty cycle for minimum angle of servo
SERVO_MAX_DUTY = 12.5+OFFSE_DUTY    #define pulse duty cycle for maximum angle of servo
servoPin = 12

def map( value, fromLow, fromHigh, toLow, toHigh):  # map a value from one range to another
range
    return (toHigh-toLow)*(value-fromLow) / (fromHigh-fromLow) + toLow

def setup():
    global p
    GPIO.setmode(GPIO.BOARD)         # use PHYSICAL GPIO Numbering
    GPIO.setup(servoPin, GPIO.OUT)   # Set servoPin to OUTPUT mode
    GPIO.output(servoPin, GPIO.LOW)  # Make servoPin output LOW level

    p = GPIO.PWM(servoPin, 50)     # set Frequece to 50Hz
    p.start(0)                     # Set initial Duty Cycle to 0

def servoWrite(angle):      # make the servo rotate to specific angle, 0-180
    if(angle<0):
        angle = 0
    elif(angle > 180):
        angle = 180
    p.ChangeDutyCycle(map(angle,0,180,SERVO_MIN_DUTY,SERVO_MAX_DUTY)) # map the angle to duty
cycle and output it

def loop():
    while True:
        for dc in range(0, 181, 1):   # make servo rotate from 0 to 180 deg
            servoWrite(dc)      # Write dc value to servo
            time.sleep(0.001)
        time.sleep(0.5)
        for dc in range(180, -1, -1): # make servo rotate from 180 to 0 deg
            servoWrite(dc)
            time.sleep(0.001)
        time.sleep(0.5)

def destroy():
    p.stop()
    GPIO.cleanup()

if __name__ == ’__main__’:     # Program entrance
    print (’Program is starting...’)
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

A 50 Hz pulse for a 20ms cycle is required to control the Servo. So we need to set the PWM frequency of servoPin to 50Hz.

```
p = GPIO.PWM(servoPin, 50)     # Set Frequency to 50Hz
```

As 0-180 degrees of the Servo’s rotation corresponds to the PWM pulse width 0.5-2.5ms within cycle 20ms and to duty cycle 2.5%-12.5%. In subfunction **servoWrite** (angle), map the angle to duty cycle to output the PWM, then the Servo will rotate to specifically determined angle. However, in practice, due to the inherent error manufactured into each Servo, the pulse width will have a deviation. So we need to define a minimum and maximum pulse width and an error offset (this is essential in robotics).

```
OFFSE_DUTY = 0.5        #define pulse offset of servo
SERVO_MIN_DUTY = 2.5+OFFSE_DUTY     #define pulse duty cycle for minimum angle of servo
SERVO_MAX_DUTY = 12.5+OFFSE_DUTY    #define pulse duty cycle for maximum angle of servo
……
def servoWrite(angle):      #make the servo rotate to specific angle (0-180 degrees)
    if(angle<0):
        angle = 0
    elif(angle > 180):
        angle = 180
    p.ChangeDutyCycle(map(angle,0,180,SERVO_MIN_DUTY,SERVO_MAX_DUTY))
```

Finally, in the "while" cycle of main function, we need to use two separate cycles to make servo rotate from 0 degrees to 180 degrees and then from 180 degrees to 0 degrees.

```
def loop():
    while True:
        for dc in range(0, 181, 1):   #make servo rotate from 0° to 180°
            servoWrite(dc)     # Write to servo
            time.sleep(0.001)
        time.sleep(0.5)
        for dc in range(180, -1, -1): #make servo rotate from 180° to 0°
            servoWrite(dc)
            time.sleep(0.001)
        time.sleep(0.5)
```

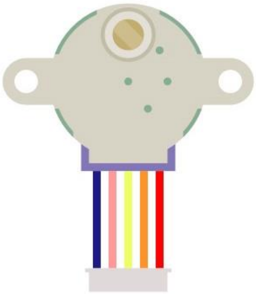
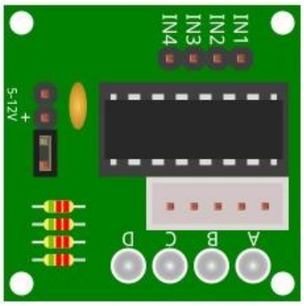
# Chapter 16 Stepper Motor

Thus far, we have learned about DC Motors and Servos. A DC motor can rotate constantly in on direction but we cannot control the rotation to a specific angle. On the contrary, a Servo can rotate to a specific angle but cannot rotate constantly in one direction. In this chapter, we will learn about a Stepper Motor which is also a type of motor. A Stepper Motor can rotate constantly and also to a specific angle. Using a Stepper Motor can easily achieve higher accuracies in mechanical motion.

## Project 16.1 Stepper Motor

In this project, we will learn how to drive a Stepper Motor, and understand its working principle.

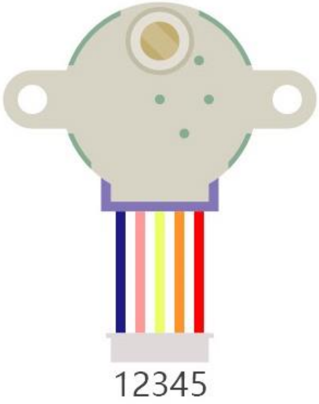
### Component List

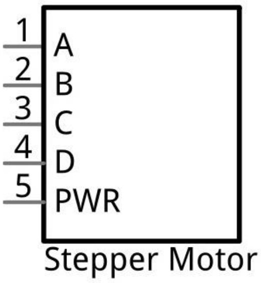
| | |
|---|---|
| Raspberry Pi (with 40 GPIO) x1<br>GPIO Expansion Board & Ribbon Cable x1<br>Breadboard x1 | Jumper Wire x12<br> |
| Stepper Motor x1<br> | ULN2003 Stepper Motor Driver x1<br> |

## Component knowledge

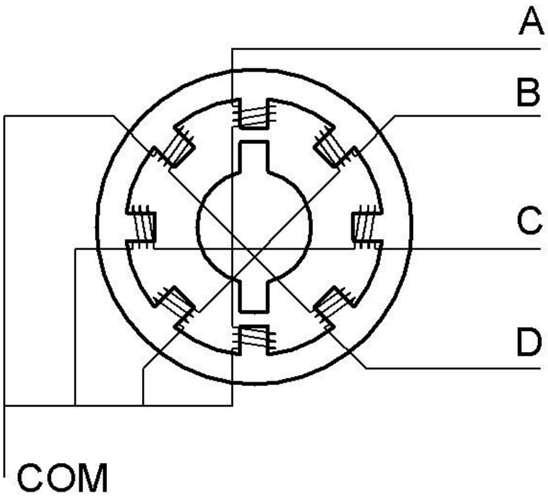
### Stepper Motor

Stepper Motors are an open-loop control device, which converts an electronic pulse signal into angular displacement or linear displacement. In a non-overload condition, the speed of the motor and the location of the stops depends only on the pulse signal frequency and number of pulses and is not affected by changes in load as with a DC Motor. A small Four-Phase Deceleration Stepper Motor is shown here:



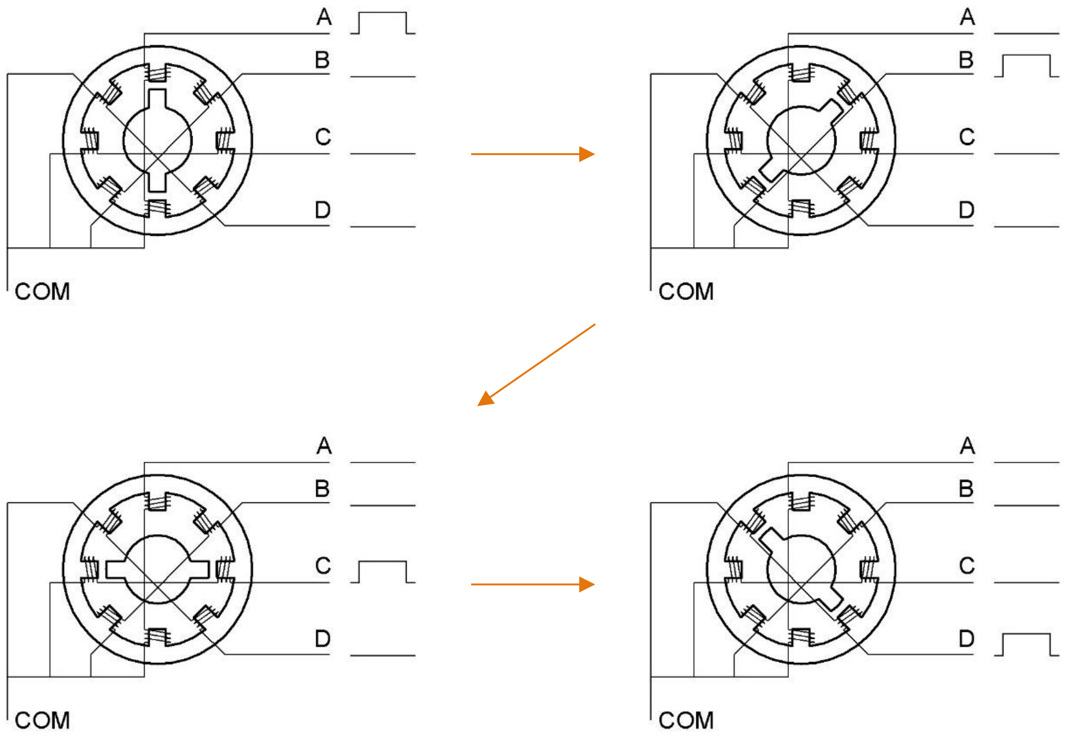


The electronic schematic diagram of a Four-Phase Stepper Motor is shown below:



The outside case or housing of the Stepper Motor is the Stator and inside the Stator is the Rotor. There is a specific number of individual coils, usually an integer multiple of the number of phases the motor has, when the Stator is powered ON, an electromagnetic field will be formed to attract a corresponding convex diagonal groove or indentation in the Rotor’s surface. The Rotor is usually made of iron or a permanent magnet. Therefore, the Stepper Motor can be driven by powering the coils on the Stator in an ordered sequence (producing a series of “steps” or stepped movements).

A common driving sequence is shown here:



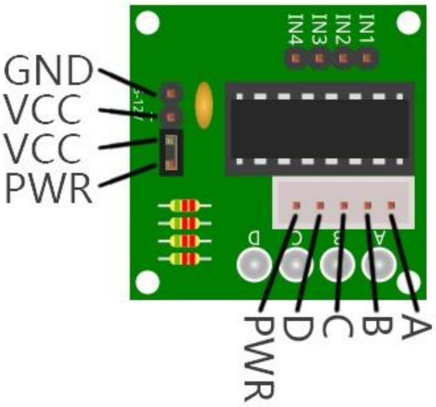
In the sequence above, the Stepper Motor rotates by a certain angle at once, which is called a “step”. By controlling the number of rotational steps, you can then control the Stepper Motor’s rotation angle. By defining the time between two steps, you can control the Stepper Motor’s rotation speed. When rotating clockwise, the order of coil powered on is: A → B → C → D → A →…… . And the rotor will rotate in accordance with this order, step by step, called four-steps, four-part. If the coils is powered ON in the reverse order, D → C → B → A → D →… , the rotor will rotate in counter-clockwise direction.

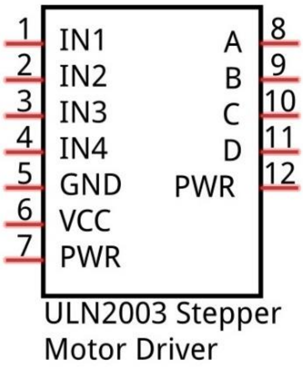
There are other methods to control Stepper Motors, such as: connect A phase, then connect A B phase, the stator will be located in the center of A B, which is called a half-step. This method can improve the stability of the Stepper Motor and reduces noise. Tise sequence of powering the coils looks like this: A → AB → B → BC → C → CD → D → DA → A →……, the rotor will rotate in accordance to this sequence ar, a half-step at a time, called four-steps, eight-part. Conversely, if the coils are powered ON in the reverse order the Stepper Motor will rotate in the opposite direction.

The stator in the Stepper Motor we have supplied has 32 magnetic poles. Therefore, to complete one full revolution requires 32 full steps. The rotor (or output shaft) of the Stepper Motor is connected to a speed reduction set of gears and the reduction ratio is 1:64. Therefore, the final output shaft (exiting the Stepper Motor’s housing) requires 32 X 64 = 2048 steps to make one full revolution.

### ULN2003 Stepper Motor driver

A ULN2003 Stepper Motor Driver is used to convert weak signals into more powerful control signals in order to drive the Stepper Motor. In the illustration below, the input signal IN1-IN4 corresponds to the output signal A-D, and 4 LEDs are integrated into the board to indicate the state of these signals. The PWR interface can be used as a power supply for the Stepper Motor. By default, PWR and VCC are connected.

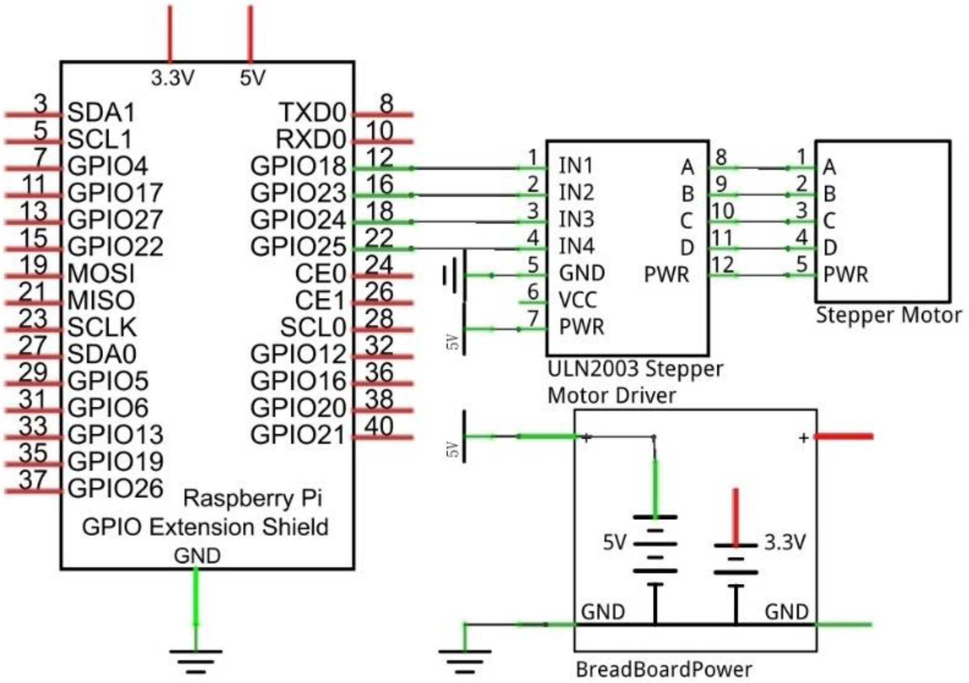




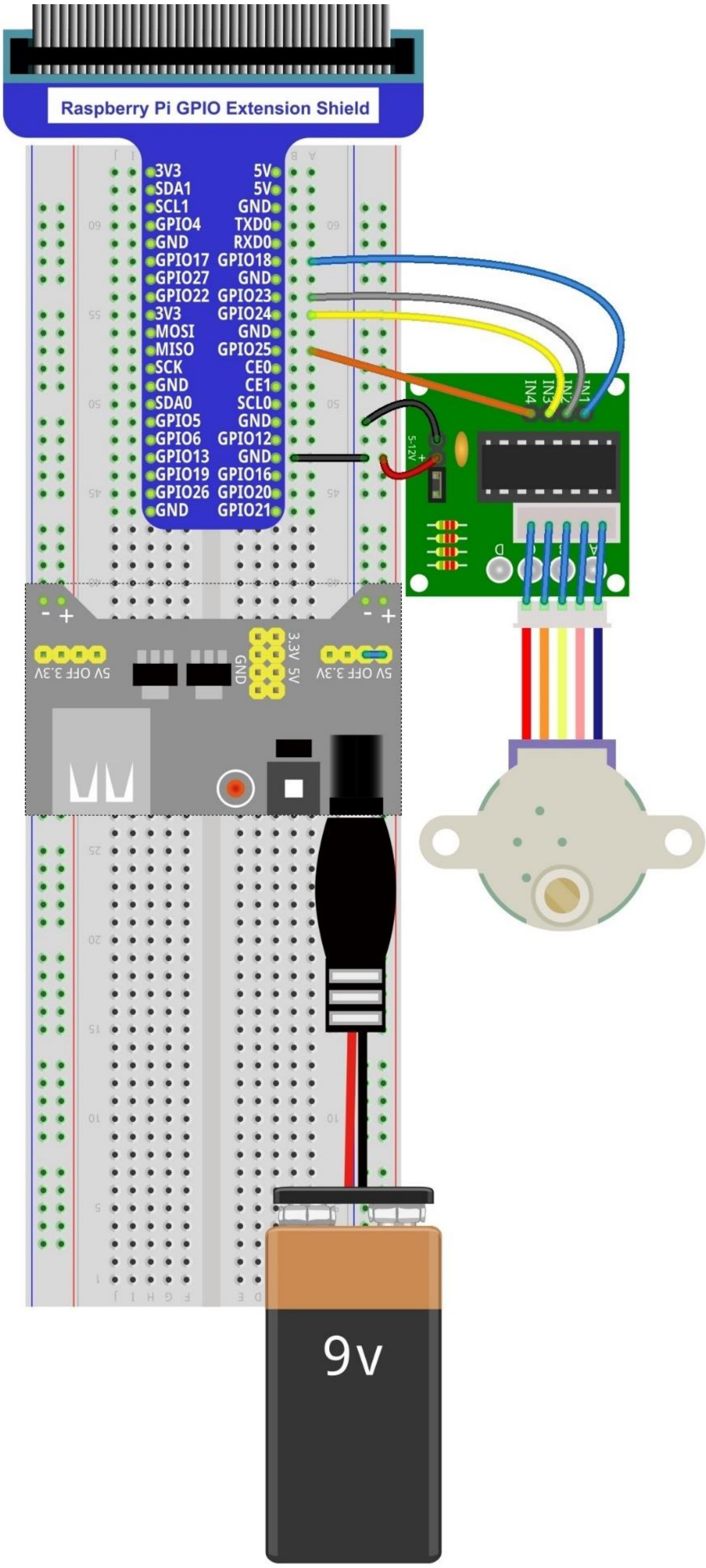
## Circuit

When building the circuit, note that rated voltage of the Stepper Motor is 5V, and we need to use the breadboard power supply independently, (**Caution do not use the RPi power supply**). Additionally, the breadboard power supply needs to share Ground with Rpi.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



## Code

This code uses the four-step, four-part mode to drive the Stepper Motor in the clockwise and anticlockwise directions.

C Code 16.1.1 SteppingMotor

First, observe the project result, and then learn about the code in detail.

**If you have any concerns, please contact us via: support@freenove.com**

1. Use cd command to enter 16.1.1_SteppingMotor directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/16.1.1_SteppingMotor
```

2. Use following command to compile "SteppingMotor.c" and generate executable file "SteppingMotor".

```
gcc SteppingMotor.c -o SteppingMotor -lwiringPi
```

3. Run the generated file "SteppingMotor".

```
sudo ./SteppingMotor
```

After the program is executed, the Stepper Motor will rotate 360° clockwise and then 360° anticlockwise and repeat this action in an endless loop.

The following is the program code:

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

const int motorPins[]={1,4,5,6};    //define pins connected to four phase ABCD of stepper 
motor
const int CCWStep[]={0x01,0x02,0x04,0x08};  //define power supply order for coil for rotating 
anticlockwise
const int CWStep[]={0x08,0x04,0x02,0x01};   //define power supply order for coil for rotating 
clockwise
//as for four phase Stepper Motor, four steps is a cycle. the function is used to drive the 
Stepper Motor clockwise or anticlockwise to take four steps
void moveOnePeriod(int dir,int ms){
    int i=0,j=0;
    for (j=0;j<4;j++){  //cycle according to power supply order 
        for (i=0;i<4;i++){  //assign to each pin, a total of 4 pins
            if(dir == 1)    //power supply order clockwise
                digitalWrite(motorPins[i],(CCWStep[j] == (1<<i)) ? HIGH : LOW);
            else        //power supply order anticlockwise
                digitalWrite(motorPins[i],(CWStep[j] == (1<<i)) ? HIGH : LOW);
            printf("motorPin %d, %d \n",motorPins[i],digitalRead(motorPins[i]));
        }
        printf("Step cycle!\n");
        if(ms<3)        //the delay can not be less than 3ms, otherwise it will exceed speed 
limit of the motor
            ms=3;
        delay(ms);
    }
}
//continuous rotation function, the parameter steps specifies the rotation cycles, every four
steps is a cycle
void moveSteps(int dir, int ms, int steps){
    int i;
    for(i=0;i<steps;i++){
        moveOnePeriod(dir,ms);
    }
}
void motorStop(){   //function used to stop rotating
    int i;
    for(i=0;i<4;i++){
        digitalWrite(motorPins[i],LOW);
    }
}
int main(void){
    int i;

    printf("Program is starting ...\n");

    wiringPiSetup();

    for(i=0;i<4;i++){
        pinMode(motorPins[i],OUTPUT);
    }

    while(1){
        moveSteps(1,3,512);     //rotating 360° clockwise, a total of 2048 steps in a circle,
namely, 512 cycles.
        delay(500);
        moveSteps(0,3,512);     //rotating 360° anticlockwise
        delay(500);
    }
    return 0;
}
```

In the code we define the four pins of the Stepper Motor and the order to supply power to the coils for a four-step rotation mode.

```
const int motorPins[]={1,4,5,6};    //define pins connected to four phase ABCD of stepper
motor
const int CCWStep[]={0x01,0x02,0x04,0x08};  //define power supply order for coil for
rotating anticlockwise
const int CWStep[]={0x08,0x04,0x02,0x01};   //define power supply order for coil for
rotating clockwise
```

Subfunction **moveOnePeriod** ((int dir,int ms) will drive the Stepper Motor rotating four-step clockwise or