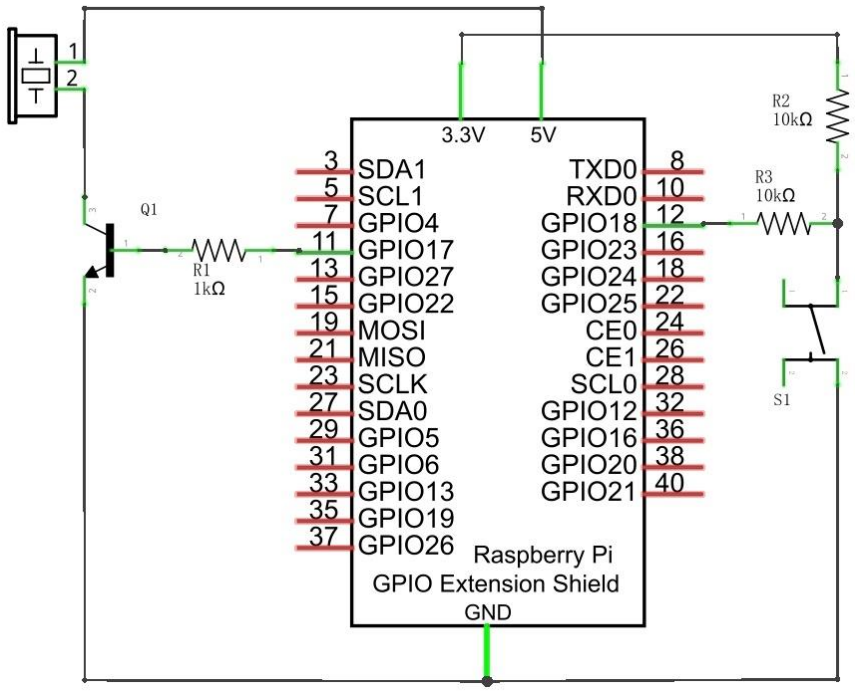
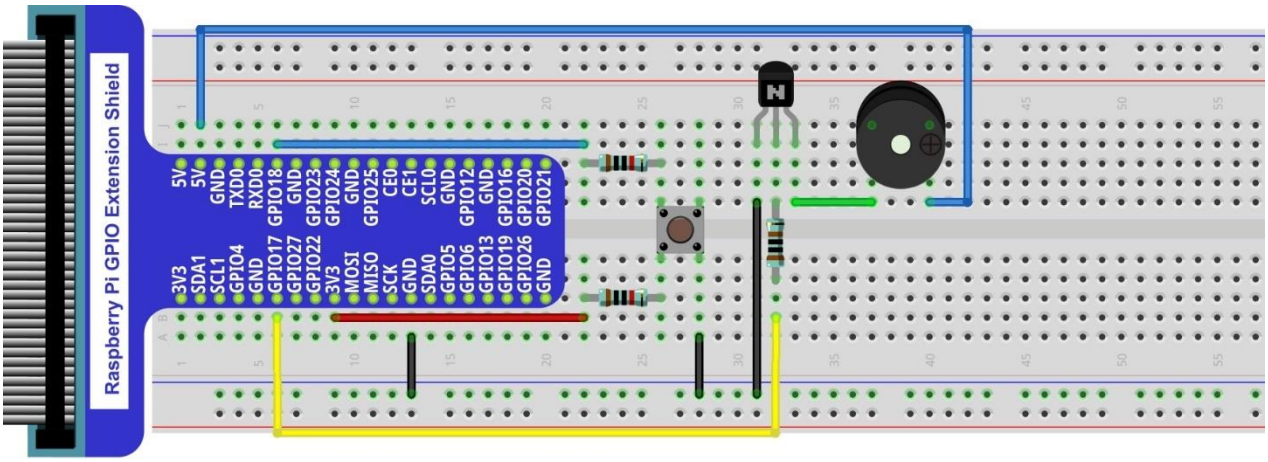
Circuit

Schematic diagram with RPi GPIO Extension Shield



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



Note: in this circuit, the power supply for the buzzer is 5V, and pull-up resistor of the push button switch is connected to the 3.3V power feed. Actually, the buzzer can work when connected to the 3.3V power feed but this will produce a weak sound from the buzzer (not very loud).

Code

In this project, a buzzer will be controlled by a push button switch. When the button switch is pressed, the buzzer sounds and when the button is released, the buzzer stops. It is analogous to our earlier project that controlled an LED ON and OFF.

C Code 6.1.1 Doorbell

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 06.1.1_Doorbell directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/06.1.1_Doorbell
```

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

```
gcc Doorbell.c -o Doorbell -lwiringPi
```

3. Then run the generated file "Doorbell".

```
sudo ./Doorbell
```

After the program is executed, press the push button switch and the will buzzer sound. Release the push button switch and the buzzer will stop.

The following is the program code:

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

#define buzzerPin 0   //define the buzzerPin
#define buttonPin 1       //define the buttonPin

void main(void)
{
    printf("Program is starting ... \n");

    wiringPiSetup();

    pinMode(buzzerPin, OUTPUT);
    pinMode(buttonPin, INPUT);

    pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
    while(1){

        if(digitalRead(buttonPin) == LOW){ //button is pressed
            digitalWrite(buzzerPin, HIGH);   //Turn on buzzer
            printf("buzzer turned on >>> \n");
        }
        else {                //button is released
            digitalWrite(buzzerPin, LOW);   //Turn off buzzer
            printf("buzzer turned off <<< \n");
        }
    }
}
```

The code is exactly the same as when we used a push button switch to control an LED. You can also try using the PNP transistor to achieve the same results.

Python Code 6.1.1 Doorbell

First, observe the project result, 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 06.1.1_Doorbell directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/06.1.1_Doorbell
```

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

```
python Doorbell.py
```

After the program is executed, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.

The following is the program code:

```
import RPi.GPIO as GPIO

buzzerPin = 11    # define buzzerPin
buttonPin = 12    # define buttonPin

def setup():
    GPIO.setmode(GPIO.BOARD)        # use PHYSICAL GPIO Numbering
    GPIO.setup(buzzerPin, GPIO.OUT)   # set buzzerPin to OUTPUT mode
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # set buttonPin to PULL UP
INPUT mode

def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW: # if button is pressed
            GPIO.output(buzzerPin,GPIO.HIGH) # turn on buzzer
            print ('buzzer turned on >>>')
        else : # if button is relessed
            GPIO.output(buzzerPin,GPIO.LOW) # turn off buzzer
            print ('buzzer turned off <<<')

def destroy():
    GPIO.cleanup()                     # Release all GPIO

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

The code is exactly the same as when we used a push button switch to control an LED. You can also try using the PNP transistor to achieve the same results.

Project 6.2 Alertor

Next, we will use a passive buzzer to make an alarm.
The list of components and the circuit is similar to the doorbell project. We only need to take the Doorbell circuit and replace the active buzzer with a passive buzzer.

Code

In this project, our buzzer alarm is controlled by the push button switch. Press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.
As stated before, it is analogous to our earlier project that controlled an LED ON and OFF.
To control a passive buzzer requires PWM of certain sound frequency.

C Code 6.2.1 Alertor

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 06.2.1_Alertor directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/06.2.1_Alertor
```

2. Use following command to compile "Alertor.c" and generate executable file "Alertor". "-lm" and "-lpthread" compiler options need to added here.

```
gcc Alertor.c -o Alertor -lwiringPi -lm -lpthread
```

3. Then run the generated file "Alertor".

```
sudo ./Alertor
```

After the program is executed, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.
The following is the program code:

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

#define buzzerPin    0    //define the buzzerPin
#define buttonPin      1       //define the buttonPin

void alertor(int pin){
    int x;
    double sinVal, toneVal;
    for(x=0;x<360;x++){ // frequency of the alertor is consistent with the sine wave
        sinVal = sin(x * (M_PI / 180));          //Calculate the sine value
        toneVal = 2000 + sinVal * 500;      //Add the resonant frequency and weighted sine
value
        softToneWrite(pin,toneVal);              //output corresponding PWM
        delay(1);
```

```
18      }
19  }
20  void stopAlertor(int pin){
21      softToneWrite(pin,0);
22  }
23  int main(void)
24  {
25      printf("Program is starting ... \n");
26
27      wiringPiSetup();
28
29      pinMode(buzzerPin, OUTPUT);
30      pinMode(buttonPin, INPUT);
31      softToneCreate(buzzerPin); //set buzzerPin
32      pullUpDnControl(buttonPin, PUD_UP);  //pull up to HIGH level
33      while(1){
34          if(digitalRead(buttonPin) == LOW){ //button is pressed
35              alertor(buzzerPin);   // turn on buzzer
36              printf("alertor turned on >>> \n");
37          }
38          else {                  //button is released
39              stopAlertor(buzzerPin);   // turn off buzzer
40              printf("alertor turned off <<< \n");
41          }
42      }
43      return 0;
44  }
```

The code is the same to the active buzzer but the method is different. A passive buzzer requires PWM of a certain frequency, so you need to create a software PWM pin though softToneCreate (buzzeRPin). Here softTone is designed to generate square waves with variable frequency and a duty cycle fixed to 50%, which is a better choice for controlling the buzzer.

```
    softToneCreate(buzzeRPin);
```

In the while loop of the main function, when the push button switch is pressed the subfunction alertor() will be called and the alarm will issue a warning sound. The frequency curve of the alarm is based on a sine curve. We need to calculate the sine value from 0 to 360 degrees and multiplied by a certain value (here this value is 500) plus the resonant frequency of buzzer. We can set the PWM frequency through softToneWrite (pin, toneVal).

```
void alertor(int pin){
    int x;
    double sinVal, toneVal;
    for(x=0;x<360;x++){ //The frequency is based on the sine curve.
        sinVal = sin(x * (M_PI / 180));
        toneVal = 2000 + sinVal * 500;
```

```
            softToneWrite(pin,toneVal);
            delay(1);
        }
    }
```

If you want to stop the buzzer, just set PWM frequency of the buzzer pin to 0.

```
void stopAlertor(int pin){
    softToneWrite(pin,0);
}
```

The related functions of softTone are described as follows:

int softToneCreate (int pin) ;
This creates a software controlled tone pin.
void softToneWrite (int pin, int freq) ;
This updates the tone frequency value on the given pin.

For more details about softTone, please refer to :http://wiringpi.com/reference/software-tone-library/

Python Code 6.2.1 Alertor

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 06.2.1_Alertor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/06.2.1_Alertor
```

2. Use the python command to execute the Python code "Alertor.py".

```
python Alertor.py
```

After the program is executed, press the push button switch and the buzzer will sound. Release the push button switch and the buzzer will stop.

The following is the program code:

```
import RPi.GPIO as GPIO
import time
import math

buzzerPin = 11    # define the buzzerPin
buttonPin = 12    # define the buttonPin

def setup():
    global p
    GPIO.setmode(GPIO.BOARD)          # Use PHYSICAL GPIO Numbering
    GPIO.setup(buzzerPin, GPIO.OUT)   # set RGBLED pins to OUTPUT mode
    GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set buttonPin to INPUT
mode, and pull up to HIGH level, 3.3V
    p = GPIO.PWM(buzzerPin, 1)
    p.start(0);

def loop():
    while True:
        if GPIO.input(buttonPin)==GPIO.LOW:
            alertor()
            print ('alertor turned on >>> ')
        else :
            stopAlertor()
            print ('alertor turned off <<<')
def alertor():
    p.start(50)
    for x in range(0,361):    # Make frequency of the alertor consistent with the sine wave
        sinVal = math.sin(x * (math.pi / 180.0))         # calculate the sine value
        toneVal = 2000 + sinVal * 500  # Add to the resonant frequency with a Weighted
        p.ChangeFrequency(toneVal)     # Change Frequency of PWM to toneVal
        time.sleep(0.001)

def stopAlertor():
```

```
34      p.stop()
35
36  def destroy():
37      GPIO.output(buzzerPin, GPIO.LOW)      # Turn off buzzer
38      GPIO.cleanup()                        # Release GPIO resource
39
40  if __name__ == '__main__':     # Program entrance
41      print ('Program is starting...')
42      setup()
43      try:
44          loop()
45      except KeyboardInterrupt:  # Press ctrl-c to end the program.
46          destroy()
```

The code is the same to the active buzzer but the method is different. A passive buzzer requires PWM of a certain frequency, so you need to create a software PWM pin though softToneCreate (buzzeRPin). The way to create a PWM was introduced earlier in the BreathingLED and RGB LED projects.

```
def setup():
    global p
    GPIO.setmode(GPIO.BOARD)          # Use PHYSICAL GPIO Numbering
    GPIO.setup(buzzerPin, GPIO.OUT)    # set RGBLED pins to OUTPUT mode
    GPIO.setup(buzzerPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)     # Set buttonPin to INPUT
mode, and pull up to HIGH level, 3.3V
    p = GPIO.PWM(buzzerPin, 1)
    p.start(0);
```

In the while loop loop of the main function, when the push button switch is pressed the subfunction alertor() will be called and the alarm will issue a warning sound. The frequency curve of the alarm is based on a sine curve. We need to calculate the sine value from 0 to 360 degrees and multiplied by a certain value (here this value is 500) plus the resonant frequency of buzzer. We can set the PWM frequency through softToneWrite (pin, toneVal).

```
def alertor():
    p.start(50)
    for x in range(0,361):
        sinVal = math.sin(x * (math.pi / 180.0))
        toneVal = 2000 + sinVal * 500
        p.ChangeFrequency(toneVal)
        time.sleep(0.001)
```

When the push button switch is released, the buzzer (in this case our Alarm) will stop.

```
def stopAlertor():
    p.stop()
```

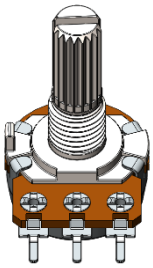
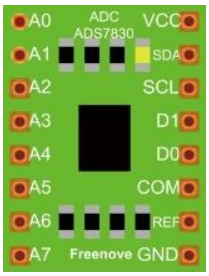
(Important) Chapter 7 ADC

We have learned how to control the brightness of an LED through PWM and that PWM is not a real analog signal. In this chapter, we will learn how to read analog values via an ADC Module and convert these analog values into digital.

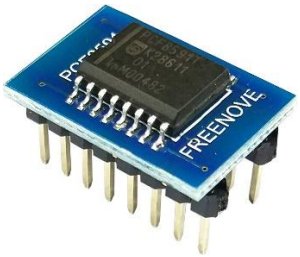
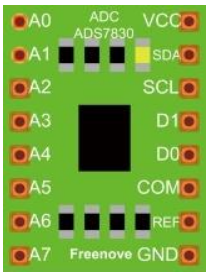
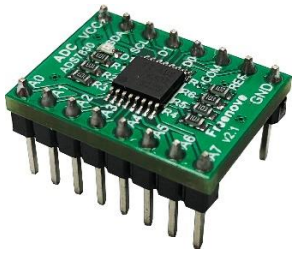
Project 7.1 Read the Voltage of Potentiometer

In this project, we will use the ADC function of an ADC Module to read the voltage value of a potentiometer.

Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	Jumper Wire M/M x16 	
Rotary potentiometer x1 	ADC module x1  or 	Resistor 10kΩ x2 

This product contains **only one ADC module**, there are two types, PCF8591 and ADS7830. For the projects described in this tutorial, they function the same. Please build corresponding circuits according to the ADC module found in your Kit.

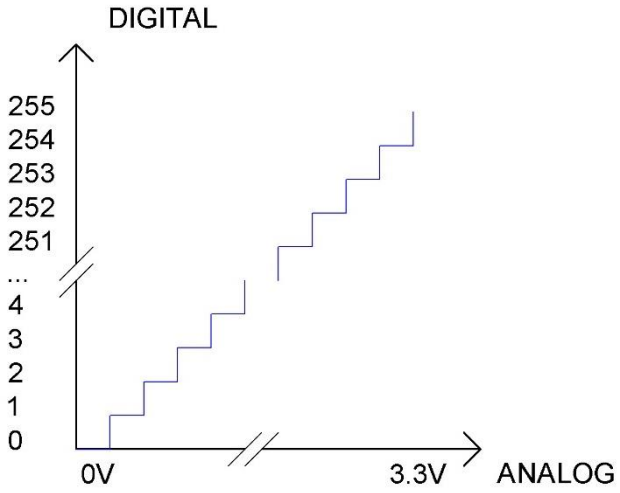
ADC module: PCF8591		**ADC module: ADS7830**	
Model diagram 	Actual Picture 	Model diagram 	Actual Picture 

Circuit knowledge

ADC

An ADC is an electronic integrated circuit used to convert analog signals such as voltages to digital or binary form consisting of 1s and 0s. The range of our ADC module is 8 bits, that means the resolution is 2^8=256, so that its range (at 3.3V) will be divided equally to 256 parts.

Any analog value can be mapped to one digital value using the resolution of the converter. So the more bits the ADC has, the denser the partition of analog will be and the greater the precision of the resulting conversion.



Subsection 1: the analog in range of 0V-3.3/256 V corresponds to digital 0;

Subsection 2: the analog in range of 3.3 /256 V-2*3.3 /256V corresponds to digital 1;

...

The resultant analog signal will be divided accordingly.

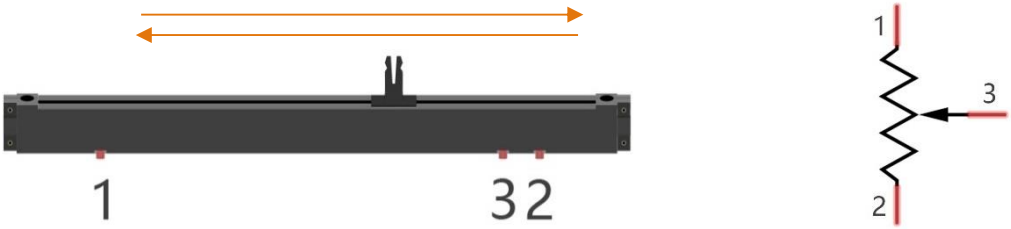
DAC

The reversing this process requires a DAC, Digital-to-Analog Converter. The digital I/O port can output high level and low level (0 or 1), but cannot output an intermediate voltage value. This is where a DAC is useful. The DAC module PCF8591 has a DAC output pin with 8-bit accuracy, which can divide VDD (here is 3.3V) into 2^8=256 parts. For example, when the digital quantity is 1, the output voltage value is 3.3/256 *1 V, and when the digital quantity is 128, the output voltage value is 3.3/256 *128=1.65V, the higher the accuracy of DAC, the higher the accuracy of output voltage value will be.

Component knowledge

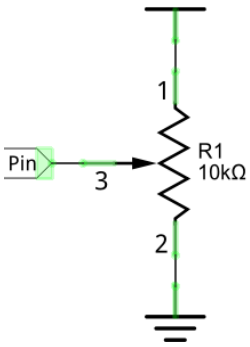
Potentiometer

Potentiometer is a resistive element with three Terminal parts. Unlike the resistors that we have used thus far in our project which have a fixed resistance value, the resistance value of a potentiometer can be adjusted. A potentiometer is often made up by a resistive substance (a wire or carbon element) and movable contact brush. When the brush moves along the resistor element, there will be a change in the resistance of the potentiometer's output side (3) (or change in the voltage of the circuit that is a part). The illustration below represents a linear sliding potentiometer and its electronic symbol on the right.



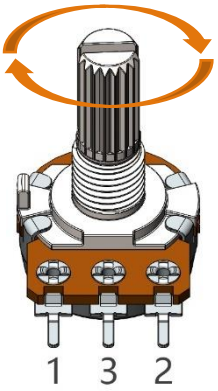
Between potentiometer pin 1 and pin 2 is the resistive element (a resistance wire or carbon) and pin 3 is connected to the brush that makes contact with the resistive element. In our illustration, when the brush moves from pin 1 to pin 2, the resistance value between pin 1 and pin 3 will increase linearly (until it reaches the highest value of the resistive element) and at the same time the resistance between pin 2 and pin 3 will decrease linearly and conversely down to zero. At the midpoint of the slider the measured resistance values between pin 1 and 3 and between pin 2 and 3 will be the same.

In a circuit, both sides of resistive element are often connected to the positive and negative electrodes of power. When you slide the brush "pin 3", you can get variable voltage within the range of the power supply.



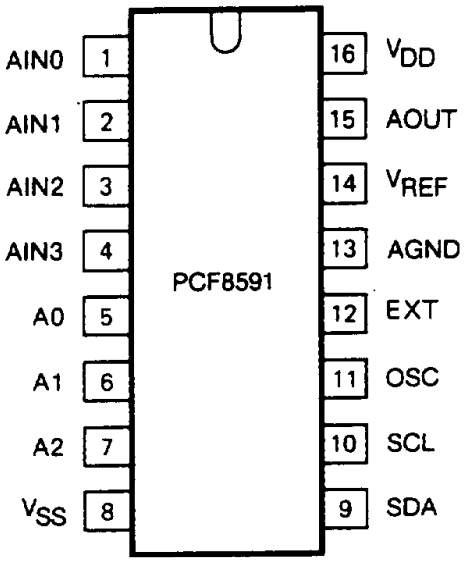
Rotary potentiometer

Rotary potentiometers and linear potentiometers have the same function; the only difference being the physical action being a rotational rather than a sliding movement.



PCF8591

The PCF8591 is a single-chip, single-supply low power 8-bit CMOS data acquisition device with four analog inputs, one analog output and a serial I2C-bus interface. The following table is the pin definition diagram of PCF8591.

SYMBOL	PIN	DESCRIPTION	TOP VIEW
AIN0	1	Analog inputs (A/D converter)	
AIN1	2		
AIN2	3		
AIN3	4		
A0	5	Hardware address	
A1	6		
A2	7		
Vss	8	Negative supply voltage	
SDA	9	I2C-bus data input/output	
SCL	10	I2C-bus clock input	
OSC	11	Oscillator input/output	
EXT	12	external/internal switch for oscillator input	
AGND	13	Analog ground	
Vref	14	Voltage reference input	
AOUT	15	Analog output(D/A converter)	
Vdd	16	Positive supply voltage	

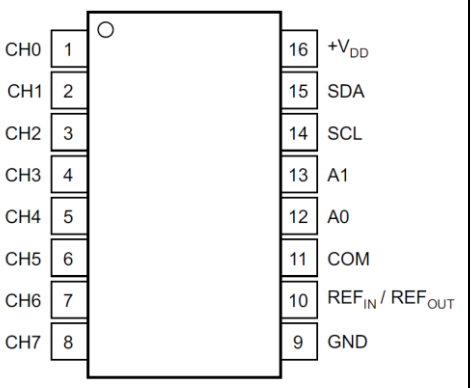
For more details about PCF8591, please refer to the datasheet which can be found on the Internet.

ADS7830

The ADS7830 is a single-supply, low-power, 8-bit data acquisition device that features a serial I2C interface and an 8-channel multiplexer. The following table is the pin definition diagram of ADS7830.

SYMBOL	PIN	DESCRIPTION	TOP VIEW
CH0	1	Analog input channels (A/D converter)	
CH1	2		
CH2	3		
CH3	4		
CH4	5		
CH5	6		

CH6	7	
CH7	8	
GND	9	Ground
REF in/out	10	Internal +2.5V Reference, External Reference Input
COM	11	Common to Analog Input Channel
A0	12	Hardware address
A1	13	
SCL	14	Serial Clock
SDA	15	Serial Sata
+VDD	16	Power Supply, 3.3V Nominal

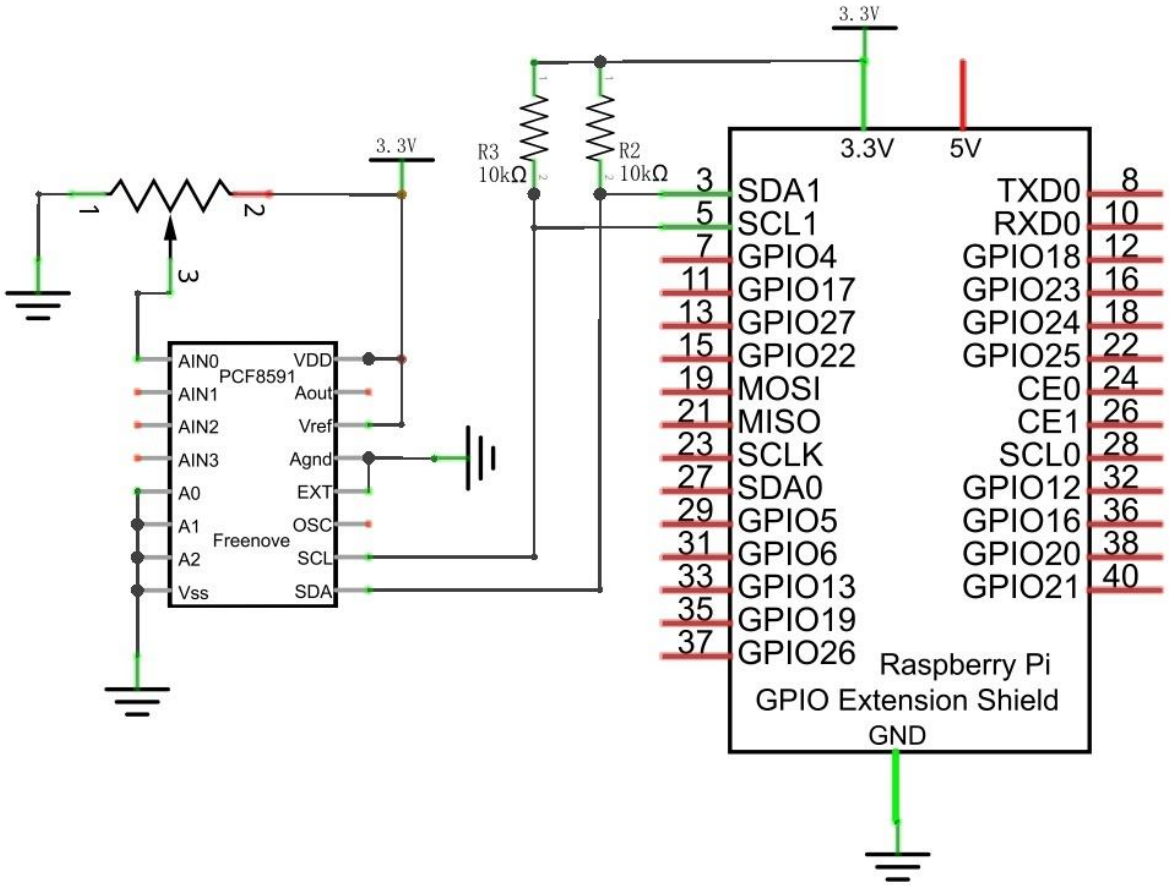


I2C communication

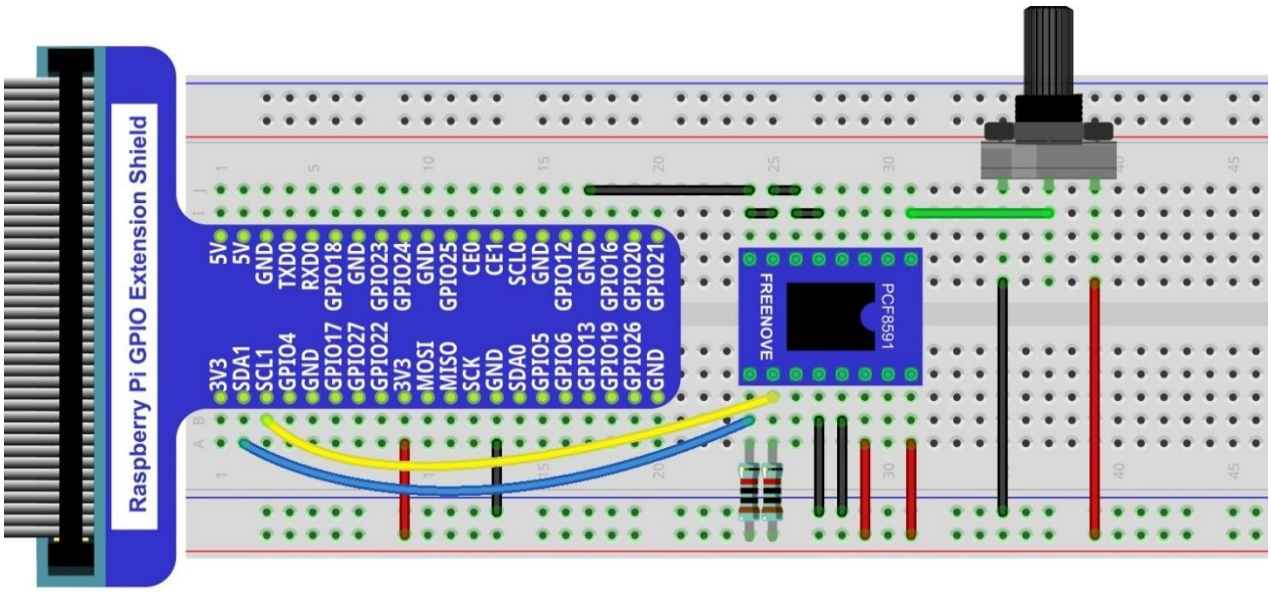
I2C (Inter-Integrated Circuit) has a two-wire serial communication mode, which can be used to connect a micro-controller and its peripheral equipment. Devices using I2C communications must be connected to the serial data line (SDA), and serial clock line (SCL) (called I2C bus). Each device has a unique address which can be used as a transmitter or receiver to communicate with devices connected via the bus.

Circuit with PCF8591

Schematic diagram



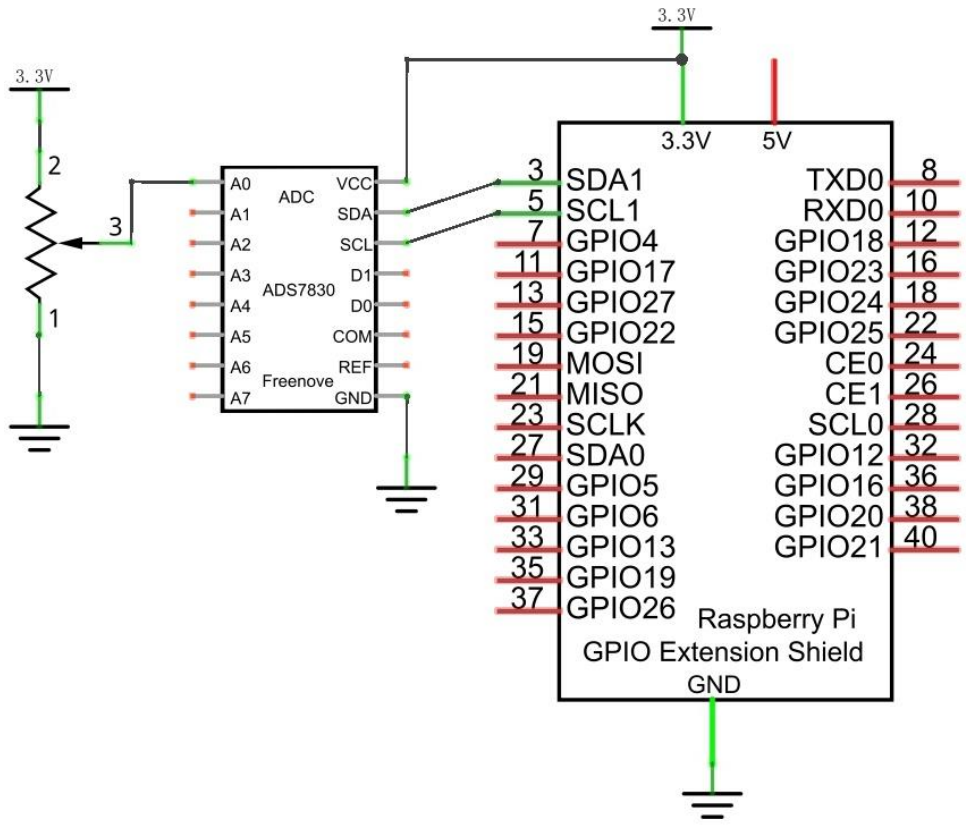
Hardware connection



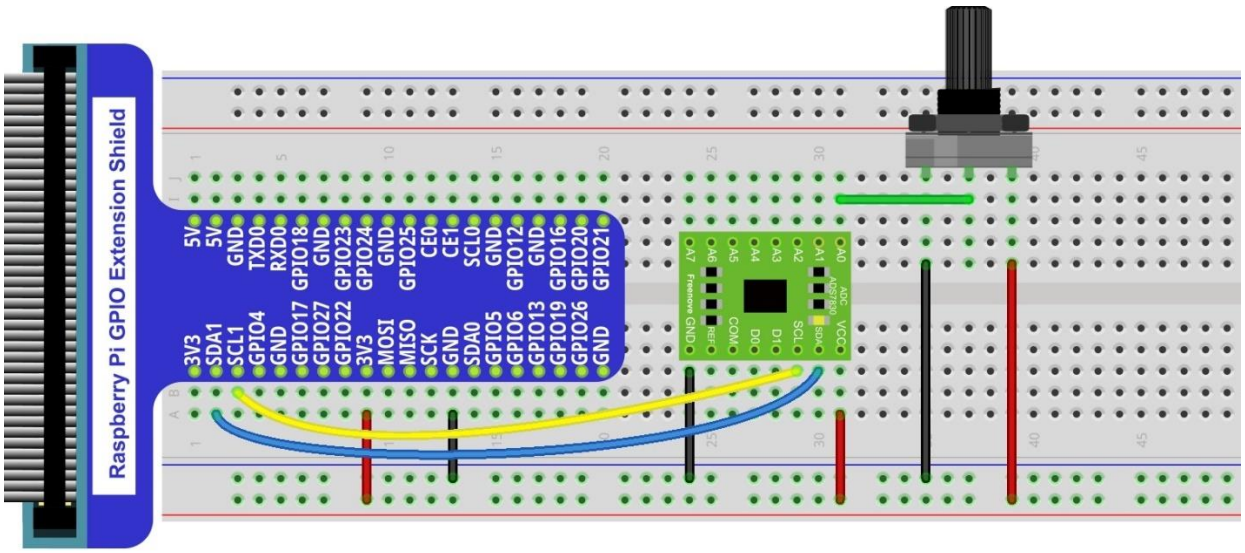
Please keep the **chip mark** consistent to make the chips under right direction and position.

Circuit with ADS7830

Schematic diagram



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



Configure I2C and Install Smbus

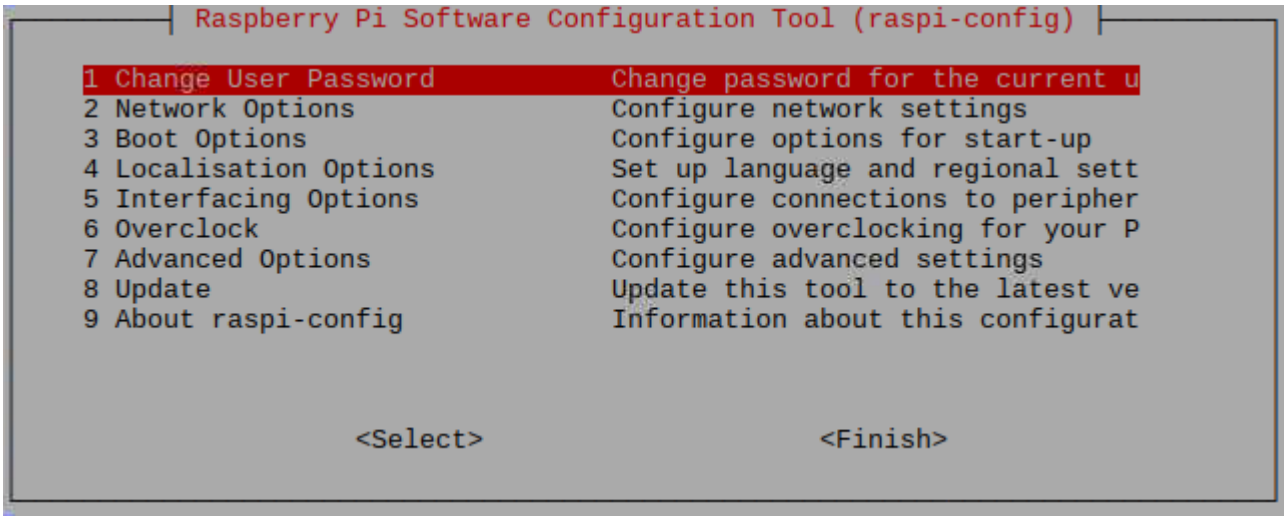
Enable I2C

The I2C interface in Raspberry Pi is disabled by default. You will need to open it manually and enable the I2C interface as follows:

Type command in the Terminal:

```
sudo raspi-config
```

Then open the following dialog box:



Choose “5 Interfacing Options” then “P5 I2C” then “Yes” and then “Finish” in this order and restart your RPi. The I2C module will then be started.

Type a command to check whether the I2C module is started:

```
lsmod | grep i2c
```

If the I2C module has been started, the following content will be shown. “bcm2708" refers to the CPU model. Different models of Raspberry Pi display different contents depending on the CPU installed:

Install I2C-Tools

Next, type the command to install I2C-Tools. It is available with the Raspberry Pi OS by default.

```
sudo apt-get install i2c-tools
```

I2C device address detection:

```
i2cdetect -y 1
```

When you are using the PCF8591 Module, the result should look like this:

```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- 48 -- -- -- -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Here, 48 (HEX) is the I2C address of ADC Module (PCF8591).

When you are using ADS, the result should look like this:

```
pi@raspberrypi:~ $ i2cdetect -y 1
     0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:          -- -- -- -- -- -- -- -- -- -- -- -- --
10: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
20: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
30: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
40: -- -- -- -- -- -- -- -- -- -- -- 4b -- -- -- --
50: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
60: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
70: -- -- -- -- -- -- -- --
```

Here, 4b (HEX) is the I2C address of ADC Module (ADS7830).

Install Smbus Module

```
sudo apt-get install python-smbus
sudo apt-get install python3-smbus
```

Code

C Code 7.1.1 ADC

For C code for the ADC Device, a custom library needs to be installed.

1. Use cd command to enter folder of the ADC Device library.

```
cd ~/Freenove_Kit/Libs/C-Libs/ADCDevice
```

2. Execute command below to install the library.

```
sh ./build.sh
```

A successful installation, without error prompts, is shown below:

```
pi@raspberrypi:~/Freenove_Kit/Libs/C-Libs/ADCDevice $ sh ./build.sh
build completed!
```

Next, we will execute the code for this project.

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 07.1.1_ADC directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/07.1.1_ADC
```

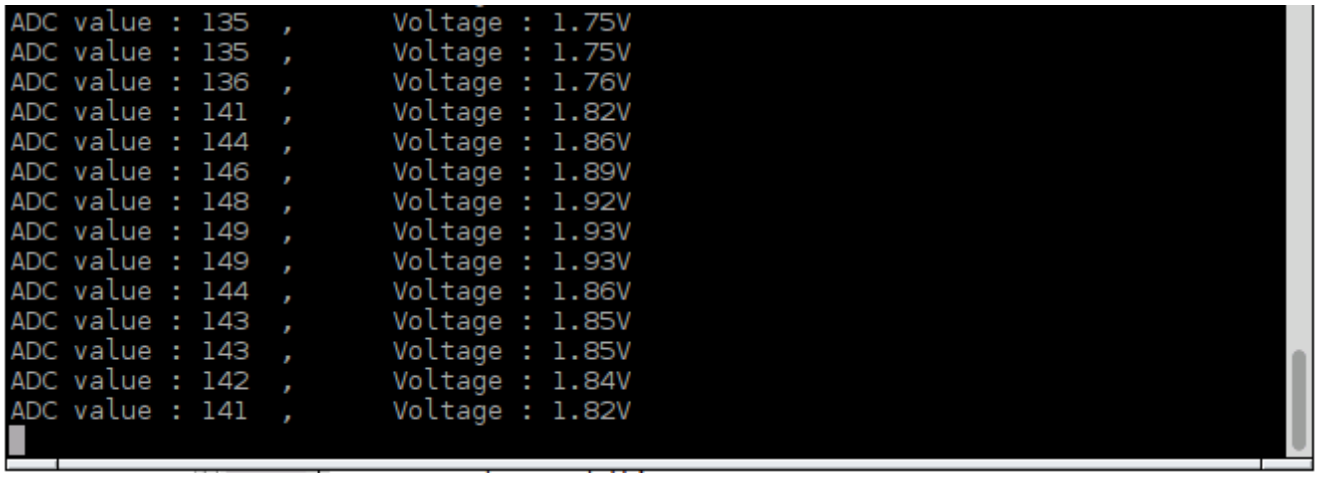
2. Use following command to compile "ADC.cpp" and generate the executable file "ADC".

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

3. Then run the generated file "ADC".

```
sudo ./ADC
```

After the program is executed, adjusting the potentiometer will produce a readout display of the potentiometer voltage values in the Terminal and the converted digital content.



The following is the code:

```
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <stdio.h>
#include <ADCDevice.hpp>

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;
        adc = new PCF8591();    // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;
        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;
    }

    while(1){
        int adcValue = adc->analogRead(0);  //read analog value of A0 pin
        float voltage = (float)adcValue / 255.0 * 3.3;  // Calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",adcValue,voltage);
        delay(100);
    }
}
```

In this code, a custom class library "ADCDevice" is used. It contains the method of utilizing the ADC Module in this project, through which the ADC Module can easily and quickly be used. In the code, you need to first create a class pointer adc, and then point to an instantiated object. (Note: An instantiated object is given a name and created in memory or on disk using the structure described within a class declaration.)

```
ADCDevice *adc;  // Define an ADC Device class object
......
adc = new ADCDevice();
```

Then use the member function detectIC(addr) in the class to detect the I2C module in the circuit. Different modules have different I2C addresses. Therefore, according to the different addresses, we can determine what the ADC module is in the circuit. When the correct module is detected, the pointer adc will point to the address of the object, and then the previously pointed content will be deleted to free memory. The default address of ADC module PCF8591 is 0x48, and that of ADC module ADS7830 is 0x4b.

```
    if(adc->detectI2C(0x48)){   // Detect the pcf8591.
        delete adc;
        adc = new PCF8591();    // If detected, create an instance of PCF8591.
    }
    else if(adc->detectI2C(0x4b)){// Detect the ads7830
        delete adc;
        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;
    }
```

When you have a class object pointed to a specific device, you can get the ADC value of the specific channel by calling the member function analogRead (chn) in this class

```
int adcValue = adc->analogRead(0);  //read analog value of A0 pin
```

Then according to the formula, the voltage value is calculated and displayed on the Terminal.

```
        float voltage = (float)adcValue / 255.0 * 3.3;  // Calculate voltage
        printf("ADC value : %d  ,\tVoltage : %.2fV\n",adcValue,voltage);
```

Reference

class ADCDevice

This is a base class. All ADC module classes are its derived classes. It has a real function and a virtual function.

`int detectI2C(int addr);`

This is a real function, which is used to detect whether the device with given I2C address exists. If it exists, return 1, otherwise return 0.

`virtual int analogRead(int chn);`

This is a virtual function that reads the ADC value of the specified channel. It is implemented in a derived class.

class PCF8591:public ADCDevice

class ADS7830:public ADCDevice

These two classes are derived from the ADCDevice class and mainly implement the function analogRead(chn).

`int analogRead(int chn);`

This returns the value read on the supplied analog input pin.

Parameter chn: For PCF8591, the range of chn is 0, 1, 2, 3. For ADS7830, the range of is 0, 1, 2, 3, 4, 5, 6, 7.

You can find the source file of this library in the folder below:

~/Freenove_Kit/Libs/C-Libs/ADCDevice/

Python Code 7.1.1 ADC

For Python code, ADCDevice requires a custom module which needs to be installed.

1. Use cd command to enter folder of ADCDevice.

```
cd ~/Freenove_Kit/Libs/Python-Libs/
```

2. Unzip the file.

```
tar zxvf ADCDevice-1.0.3.tar.gz
```

3. Open the unzipped folder.

```
cd ADCDevice-1.0.3
```

4. Install library for python2 and python3.

```
sudo python2 setup.py install
sudo python3 setup.py install
```

A successful installation, without error prompts, is shown below:

```
Installed /usr/local/lib/python3.7/dist-packages/ADCDevice-1.0.2-py3.7.egg
Processing dependencies for ADCDevice==1.0.2
Finished processing dependencies for ADCDevice==1.0.2
```

Execute the following command. 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 07.1.1_ADC directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/07.1.1_ADC
```

2. Use the Python command to execute the Python code "ADC.py".

```
python ADC.py
```

After the program is executed, adjusting the potentiometer will produce a readout display of the potentiometer voltage values in the Terminal and the converted digital content.

```
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
ADC Value : 169, Voltage : 2.19
ADC Value : 168, Voltage : 2.17
ADC Value : 168, Voltage : 2.17
```

The following is the code:

```
1  import time
2  from ADCDevice import *
3
4  adc = ADCDevice() # Define an ADCDevice class object
5
6  def setup():
7      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 the ADC value of channel 0
        voltage = value / 255.0 * 3.3  # calculate the voltage value
        print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
        time.sleep(0.1)

def destroy():
    adc.close()

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

In this code, a custom Python module "ADCDevice" is used. It contains the method of utilizing the ADC Module in this project, through which the ADC Module can easily and quickly be used. In the code, you need to first create an ADCDevice object adc.

```
adc = ADCDevice() # Define an ADCDevice class object
```

Then in setup(), use detecticIC(addr), the member function of ADCDevice, to detect the I2C module in the circuit. Different modules have different I2C addresses. Therefore, according to the address, we can determine which ADC Module is in the circuit. When the correct module is detected, a device specific class object is created and assigned to adc. The default address of PCF8591 is 0x48, and that of ADS7830 is 0x4b.

```
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)
```

When you have a class object of a specific device, you can get the ADC value of the specified channel by calling the member function of this class, analogRead(chn). In loop(), get the ADC value of potentiometer.

```
        value = adc.analogRead(0)    # read the ADC value of channel 0
```

Then according to the formula, the voltage value is calculated and displayed on the terminal monitor.

```
        voltage = value / 255.0 * 3.3  # calculate the voltage value
        print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
        time.sleep(0.1)
```

Reference

About smbus Module:

smbus Module

The System Management Bus Module defines an object type that allows SMBus transactions on hosts running the Linux kernel. The host kernel must support I2C, I2C device interface support, and a bus adapter driver. All of these can be either built-in to the kernel, or loaded from modules.

In Python, you can use help(smbus) to view the relevant functions and their descriptions.

bus=smbus.SMBus(1): Create an SMBus class object.

bus.read_byte_data(address,cmd+chn): Read a byte of data from an address and return it.

bus.write_byte_data(address,cmd,value): Write a byte of data to an address.

class ADCDevice(object)

This is a base class.

```
int detectI2C(int addr);
```

This is a member function, which is used to detect whether the device with the given I2C address exists. If it exists, it returns true. Otherwise, it returns false.

class PCF8591(ADCDevice)

class ADS7830(ADCDevice)

These two classes are derived from the ADCDevice and the main function is analogRead(chn).

```
int analogRead(int chn);
```

This returns the value read on the supplied analog input pin.

Parameter chn: For PCF8591, the range of chn is 0, 1, 2, 3. For ADS7830, the range is 0, 1, 2, 3, 4, 5, 6, 7.

You can find the source file of this library in the folder below:

~/Freenove_Kit/Libs/Python-Libs/ADCDevice-1.0.2/src/ADCDevice/ADCdevice.py

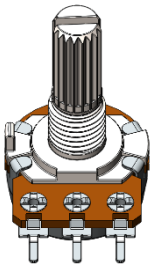
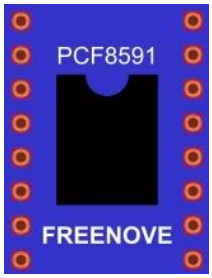
Chapter 8 Potentiometer & LED

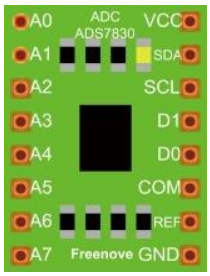
Earlier we learned how to use ADC and PWM. In this chapter, we learn to control the brightness of an LED by using a potentiometer.

Project 8.1 Soft Light

In this project, we will make a soft light. We will use an ADC Module to read ADC values of a potentiometer and map it to duty cycle ratio of the PWM used to control the brightness of an LED. Then you can change the brightness of an LED by adjusting the potentiometer.

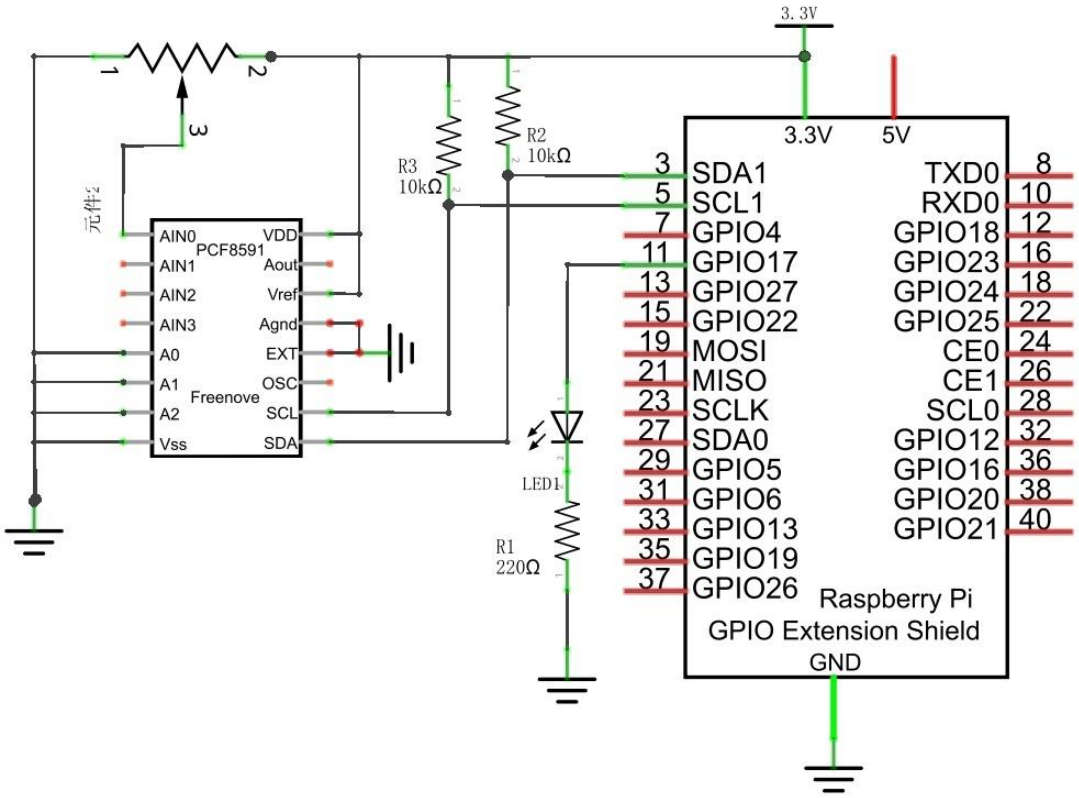
Component List

<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="4">Jumper Wire M/M x17
  </td></tr>
<tr><td>Rotary Potentiometer x1
  </td><td colspan="2">ADC Module x1
 

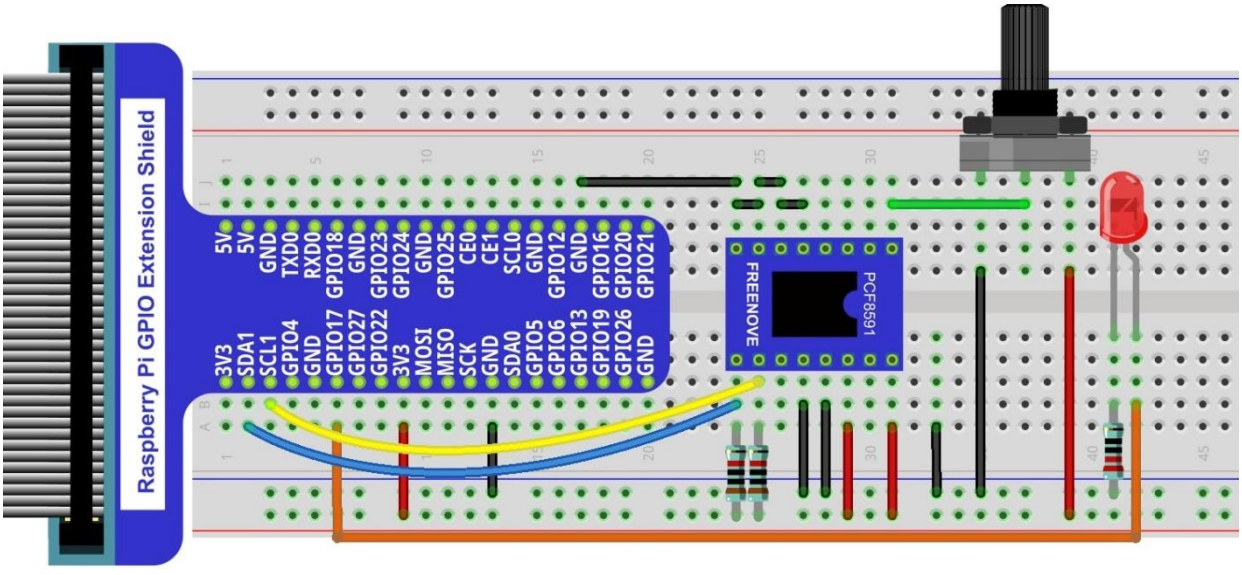
or

</td><td>10kΩ x2
  </td><td>220Ω x1
  </td><td>LED x1
  </td></tr>
</table>

Circuit with PCF8591

Schematic diagram

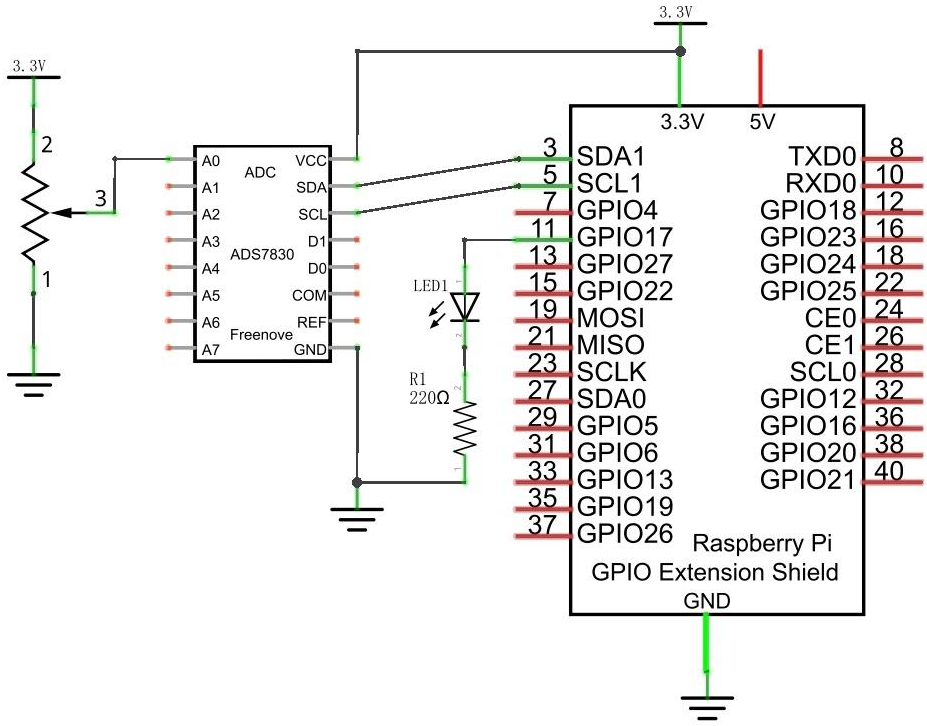


Hardware connection

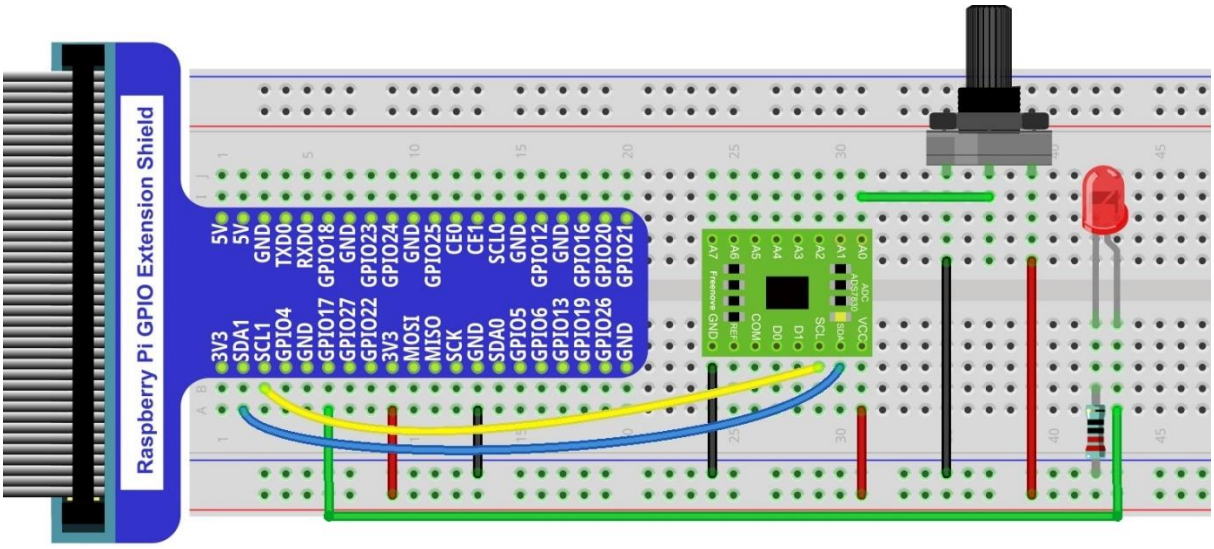


Circuit with ADS7830

Schematic diagram



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



Code

C Code 8.1.1 Softlight

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

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 08.1.1_Softlight directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/08.1.1_Softlight
```

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

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

3. Then run the generated file "Softlight".

```
sudo ./Softlight
```

After the program is executed, adjusting the potentiometer will display the voltage values of the potentiometer in the Terminal window and the converted digital quantity. As a consequence, the brightness of LED will be changed.

The following is the code:

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

#define ledPin 0

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();
    softPwmCreate(ledPin,0,100);
    while(1){
        int adcValue = adc->analogRead(0);     //read analog value of A0 pin
        softPwmWrite(ledPin,adcValue*100/255);     // Mapping to PWM duty cycle
```

```
33          float voltage = (float)adcValue / 255.0 * 3.3;  // Calculate voltage
34          printf("ADC value : %d  ,\tVoltage : %.2fV\n",adcValue,voltage);
35          delay(30);
36      }
37      return 0;
38  }
```

In the code, read the ADC value of potentiometer and map it to the duty cycle of PWM to control LED brightness.

```
            int adcValue = adc->analogRead(0);      //read analog value of A0 pin
            softPwmWrite(ledPin,adcValue*100/255);     // Mapping to PWM duty cycle
```

Python Code 8.1.1 Softlight

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 08.1.1_Softlight directory of Python code

```
cd ~/Freenove_Kit/Code/Python_Code/08.1.1_Softlight
```

2. Use the python command to execute the Python code "Softlight.py".

```
python Softlight.py
```

After the program is executed, adjusting the potentiometer will display the voltage values of the potentiometer in the Terminal window and the converted digital quantity. As a consequence, the brightness of LED will be changed.

The following is the code:

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

ledPin = 11
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(ledPin, GPIO.OUT)
    p = GPIO.PWM(ledPin, 1000)
    p.start(0)

def loop():
    while True:
        value = adc.analogRead(0)     # read the ADC value of channel 0
        p.ChangeDutyCycle(value*100/255)         # Mapping to PWM duty cycle
        voltage = value / 255.0 * 3.3  # calculate the voltage value
```

```
30          print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
31          time.sleep(0.03)
32
33  def destroy():
34      adc.close()
35
36  if __name__ == '__main__':   # Program entrance
37      print ('Program is starting ... ')
38      try:
39          setup()
40          loop()
41      except KeyboardInterrupt: # Press ctrl-c to end the program.
42          destroy()
```

In the code, read ADC value of potentiometers and map it to the duty cycle of the PWM to control LED brightness.

```
            value = adc.analogRead(0)     # read the ADC value of channel 0
            p.ChangeDutyCycle(value*100/255)        # Mapping to PWM duty cycle
```

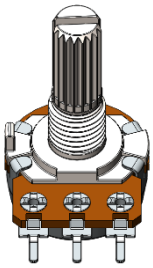
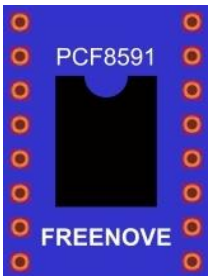
Chapter 9 Potentiometer & RGBLED

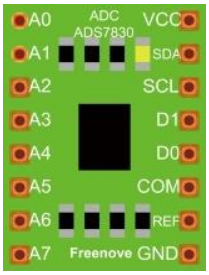
In this chapter, we will use 3 potentiometers to control the brightness of 3 LEDs of RGBLED to create multiple colors.

Project 9.1 Colorful Light

In this project, 3 potentiometers are used to control the RGB LED and in principle it is the same as with the Soft Light. project. Namely, read the voltage value of the potentiometer and then convert it to PWM used to control LED brightness. Difference is that the previous soft light project needed only one LED while this one required (3) RGB LEDs.

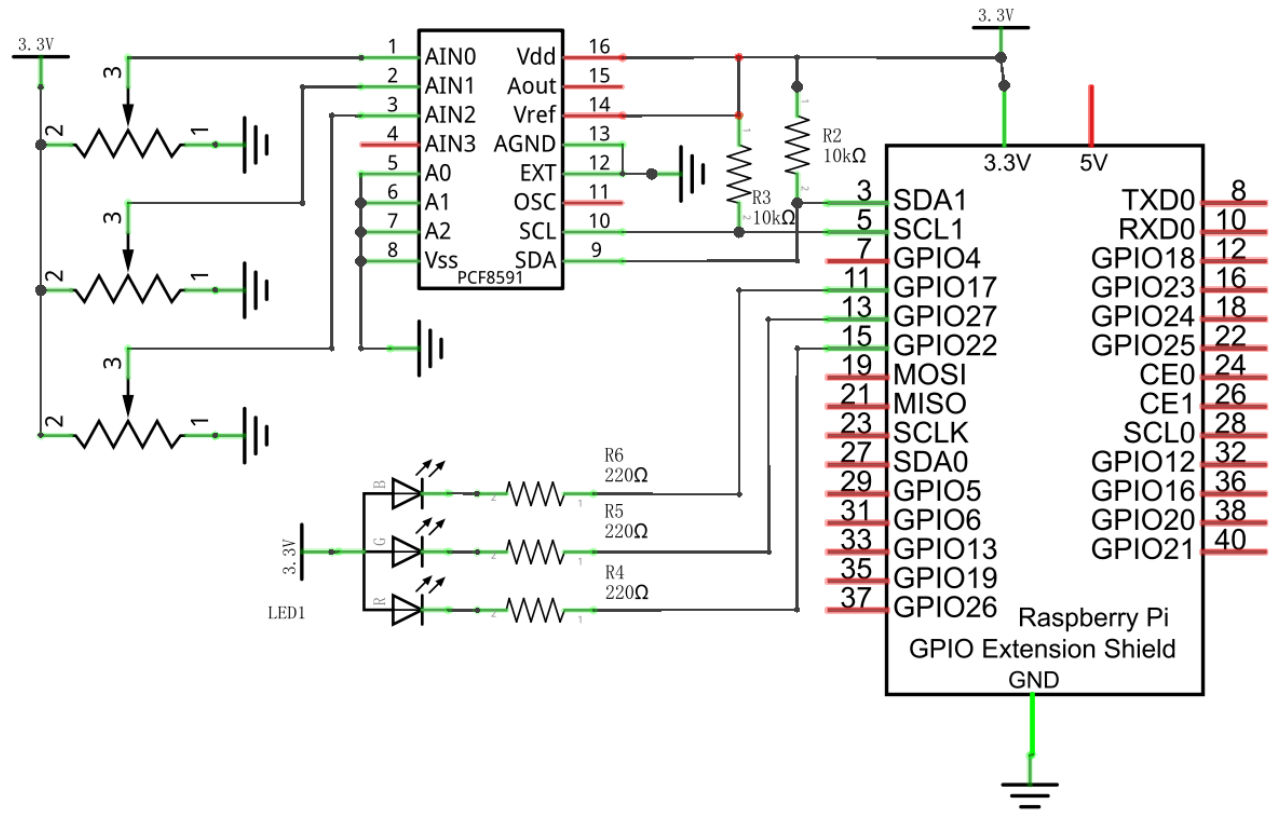
Component List

<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wires M/M x17
  </td></tr>
<tr><td>Rotary potentiometer x3
  </td><td colspan="2">ADC module x1
 

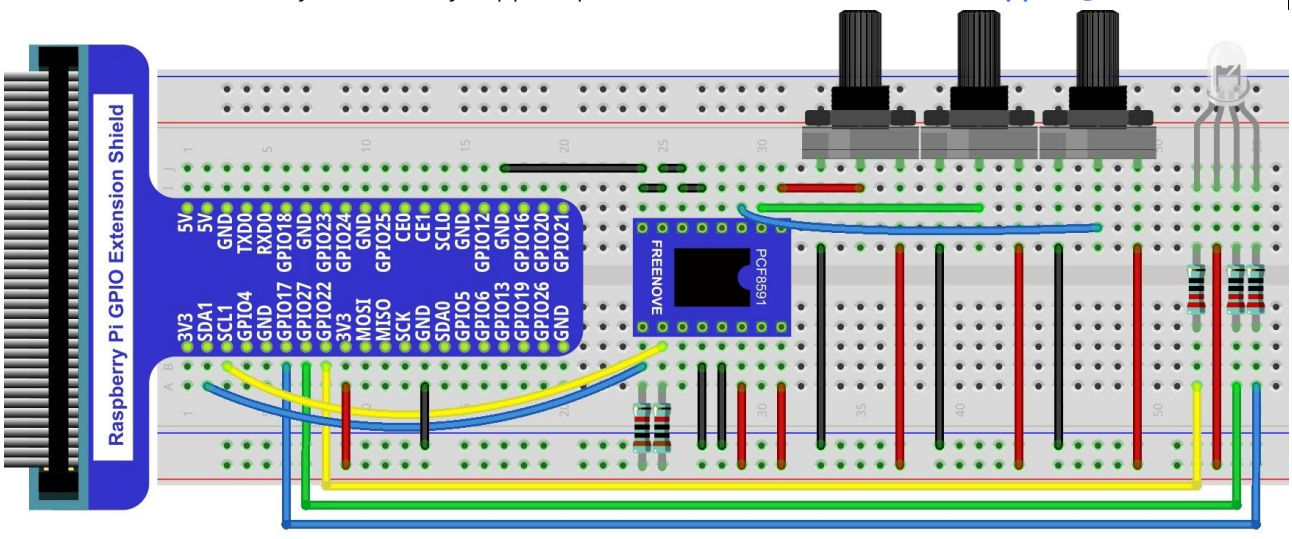
or

</td><td>10kΩ x2
  </td><td>220Ω x3
  </td><td>RGB LEDx1
  </td></tr>
</table>

Circuit with PCF8591

Schematic diagram

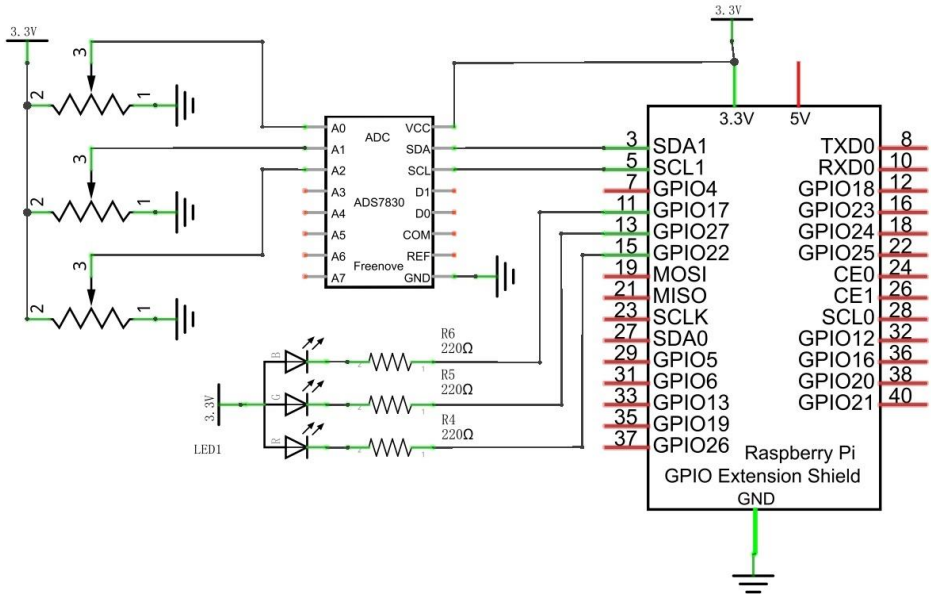


Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com

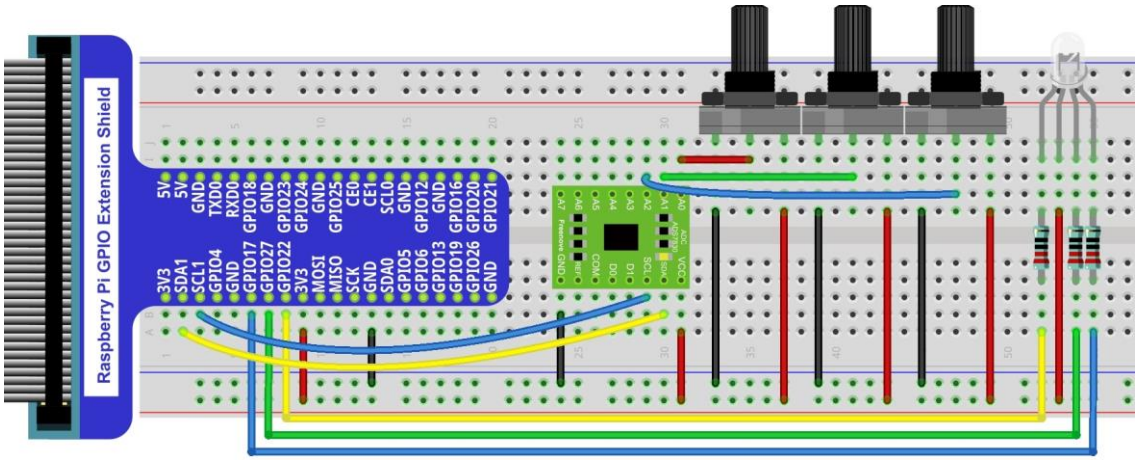


Circuit with ADS7830

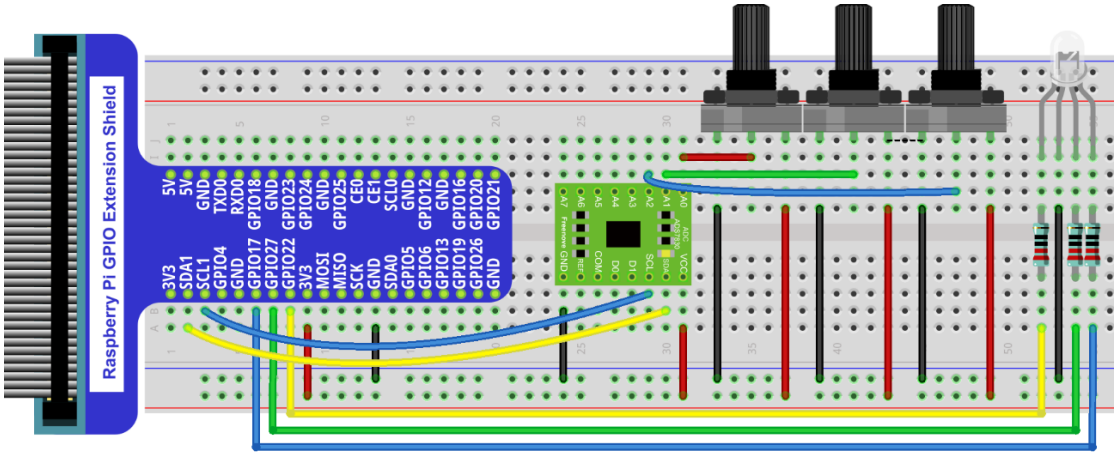
Schematic diagram



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



If circuit above doesn't work, please try following wiring.



Code

C Code 9.1.1 Colorful Softlight

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 09.1.1_ColorfulSoftlight directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/09.1.1_ColorfulSoftlight
```

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

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

3. Then run the generated file "ColorfulSoftlight".

```
sudo ./ColorfulSoftlight
```

After the program is executed, rotate one of the potentiometers, then the color of RGB LED will change. The Terminal window will display the ADC value of each potentiometer.

```
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 238
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 206
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 174
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 152
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 139
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 138
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 138
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 138
ADC value val_Red: 147  ,       val_Green: 192  ,       val_Blue: 138
```

The following is the program code:

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

#define ledRedPin 3         //define 3 pins for RGBLED
#define ledGreenPin 2
#define ledBluePin 0

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
```

```
18          adc = new PCF8591();    // If detected, create an instance of PCF8591.
19      }
20      else if(adc->detectI2C(0x4b)){// Detect the ads7830
21          delete adc;               // Free previously pointed memory
22          adc = new ADS7830();      // If detected, create an instance of ADS7830.
23      }
24      else{
25          printf("No correct I2C address found, \n"
26          "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
27          "Program Exit. \n");
28          return -1;
29      }
30      wiringPiSetup();
31      softPwmCreate(ledRedPin,0,100);     //creat 3 PMW output pins for RGBLED
32      softPwmCreate(ledGreenPin,0,100);
33      softPwmCreate(ledBluePin,0,100);
34      while(1){
35          int val_Red = adc->analogRead(0);  //read analog value of 3 potentiometers
36          int val_Green = adc->analogRead(1);
37          int val_Blue = adc->analogRead(2);
38          softPwmWrite(ledRedPin,val_Red*100/255);    //map the read value of
39  potentiometers into PWM value and output it
40          softPwmWrite(ledGreenPin,val_Green*100/255);
41          softPwmWrite(ledBluePin,val_Blue*100/255);
42          //print out the read ADC value
43          printf("ADC value val_Red: %d  ,\tval_Green: %d  ,\tval_Blue: %d
44  \n",val_Red,val_Green,val_Blue);
45          delay(100);
46      }
47      return 0;
48  }
```

In the code you can read the ADC values of the 3 potentiometers and map it into a PWM duty cycle to control the 3 LED elements to vary the color of their respective RGB LED.

Python Code 9.1.1 ColorfulSoftlight

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 09.1.1_ColorfulSoftlight directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/09.1.1_ColorfulSoftlight
```

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

```
python ColorfulSoftlight.py
```

After the program is executed, rotate one of the potentiometers, then the color of RGB LED will change. The Terminal window will display the ADC value of each potentiometer.

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import time
3  from ADCDevice import *
4
5  ledRedPin = 15      # define 3 pins for RGBLED
6  ledGreenPin = 13
7  ledBluePin = 11
8  adc = ADCDevice() # Define an ADCDevice class object
9
10 def setup():
11     global adc
12     if(adc.detectI2C(0x48)): # Detect the pcf8591.
13         adc = PCF8591()
14     elif(adc.detectI2C(0x4b)): # Detect the ads7830
15         adc = ADS7830()
16     else:
17         print("No correct I2C address found, \n"
18         "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
19         "Program Exit. \n");
20         exit(-1)
21
22     global p_Red, p_Green, p_Blue
23     GPIO.setmode(GPIO.BOARD)
24     GPIO.setup(ledRedPin, GPIO.OUT)       # set RGBLED pins to OUTPUT mode
25     GPIO.setup(ledGreenPin, GPIO.OUT)
26     GPIO.setup(ledBluePin, GPIO.OUT)
27
28     p_Red = GPIO.PWM(ledRedPin, 1000)     # configure PMW for RGBLED pins, set PWM
29 Frequence to 1kHz
30     p_Red.start(0)
31     p_Green = GPIO.PWM(ledGreenPin, 1000)
```

```
    p_Green.start(0)
    p_Blue = GPIO.PWM(ledBluePin,1000)
    p_Blue.start(0)

def loop():
    while True:
        value_Red = adc.analogRead(0)        # read ADC value of 3 potentiometers
        value_Green = adc.analogRead(1)
        value_Blue = adc.analogRead(2)
        p_Red.ChangeDutyCycle(value_Red*100/255)  # map the read value of potentiometers
into PWM value and output it
        p_Green.ChangeDutyCycle(value_Green*100/255)
        p_Blue.ChangeDutyCycle(value_Blue*100/255)
        # print read ADC value
        print ('ADC Value
value_Red: %d ,\tvlue_Green: %d ,\tvalue_Blue: %d'%(value_Red,value_Green,value_Blue))
        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 you can read the ADC values of the 3 potentiometers and map it into a PWM duty cycle to control the 3 LED elements to vary the color of their respective RGB LED.

Chapter 10 Photoresistor & LED

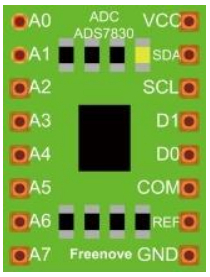
In this chapter, we will learn how to use a photoresistor to make an automatic dimming nightlight.

Project 10.1 NightLamp

A Photoresistor is very sensitive to the amount of light present. We can take advantage of the characteristic to make a nightlight with the following function. When the ambient light is less (darker environment), the LED will automatically become brighter to compensate and when the ambient light is greater (brighter environment) the LED will automatically dim to compensate.

Component List

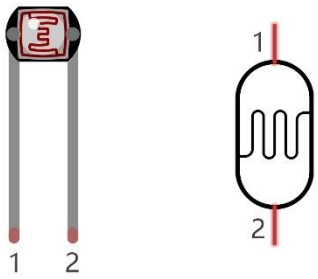
<table>
<tr><td colspan="3">Raspberry Pi x1
GPIO Extension Board & Ribbon Cable x1
Breadboard x1</td><td colspan="3">Jumper Wires M/M x15
  </td></tr>
<tr><td>Photoresistor x1
  </td><td colspan="2">ADC module x1
 

or

</td><td>10kΩ x3
  </td><td>220Ω x1
  </td><td>LED x1
  </td></tr>
</table>

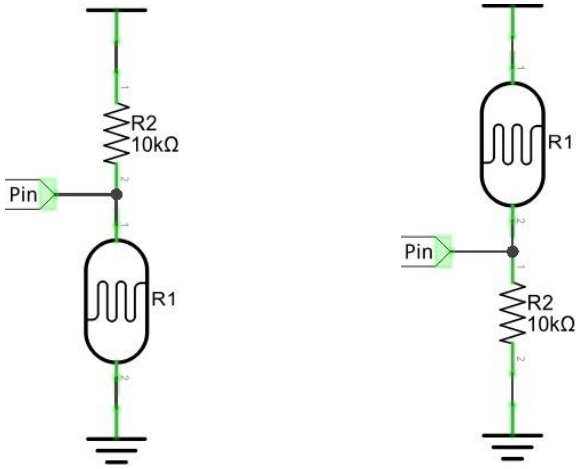
Component knowledge

Photoresistor

A Photoresistor is simply a light sensitive resistor. It is **an active component that decreases resistance with respect to receiving luminosity (light) on the component's light sensitive surface.** A Photoresistor's resistance value will change in proportion to the ambient light detected. With this characteristic, we can use a Photoresistor to detect light intensity. The Photoresistor and its electronic symbol are as follows.



The circuit below is used to detect the change of a Photoresistor's resistance value:

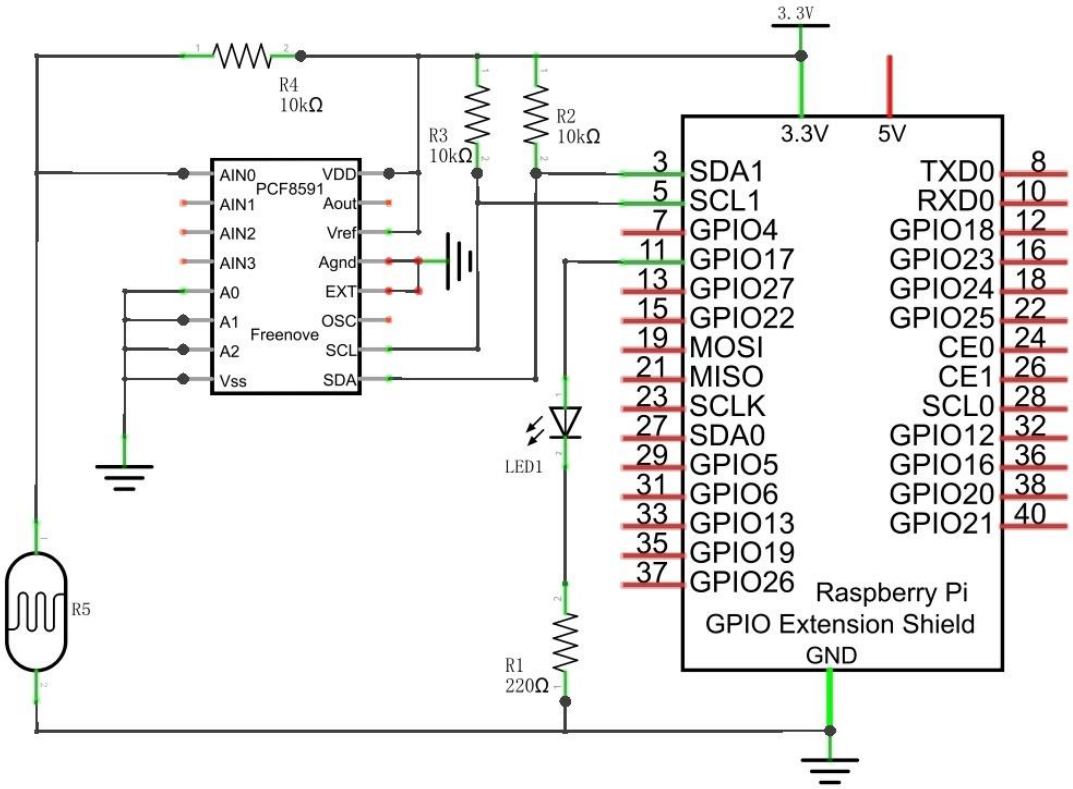


In the above circuit, when a Photoresistor's resistance vale changes due to a change in light intensity, the voltage between the Photoresistor and Resistor R1 will also change. Therefore, the intensity of the light can be obtained by measuring this voltage.

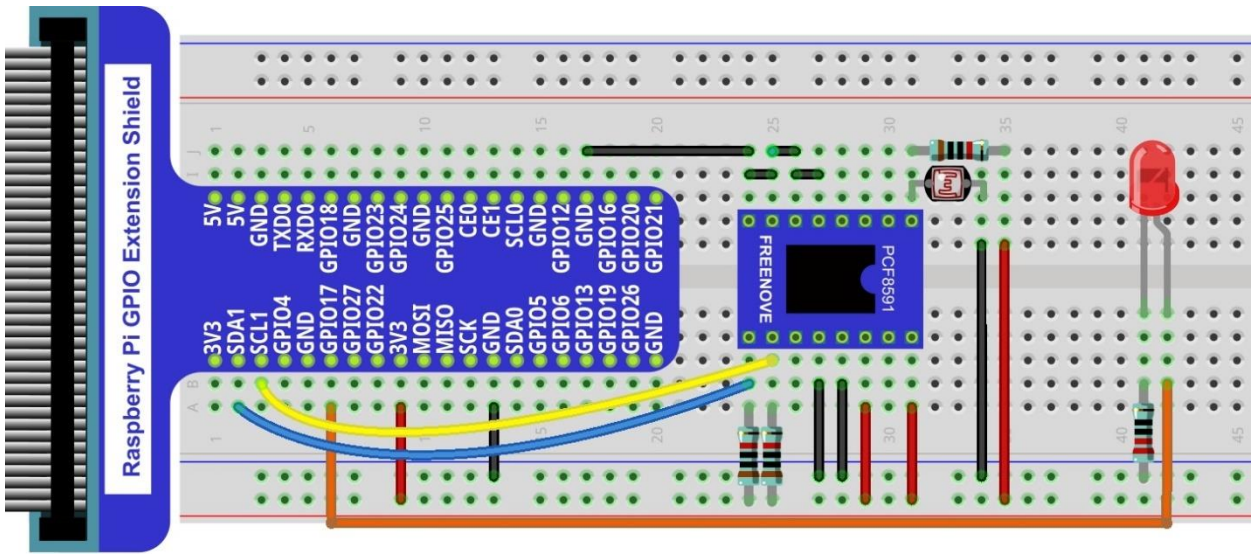
Circuit with PCF8591

The circuit used is similar to the Soft light project. The only difference is that the input signal of the AIN0 pin of ADC changes from a Potentiometer to a combination of a Photoresistor and a Resistor.

Schematic diagram



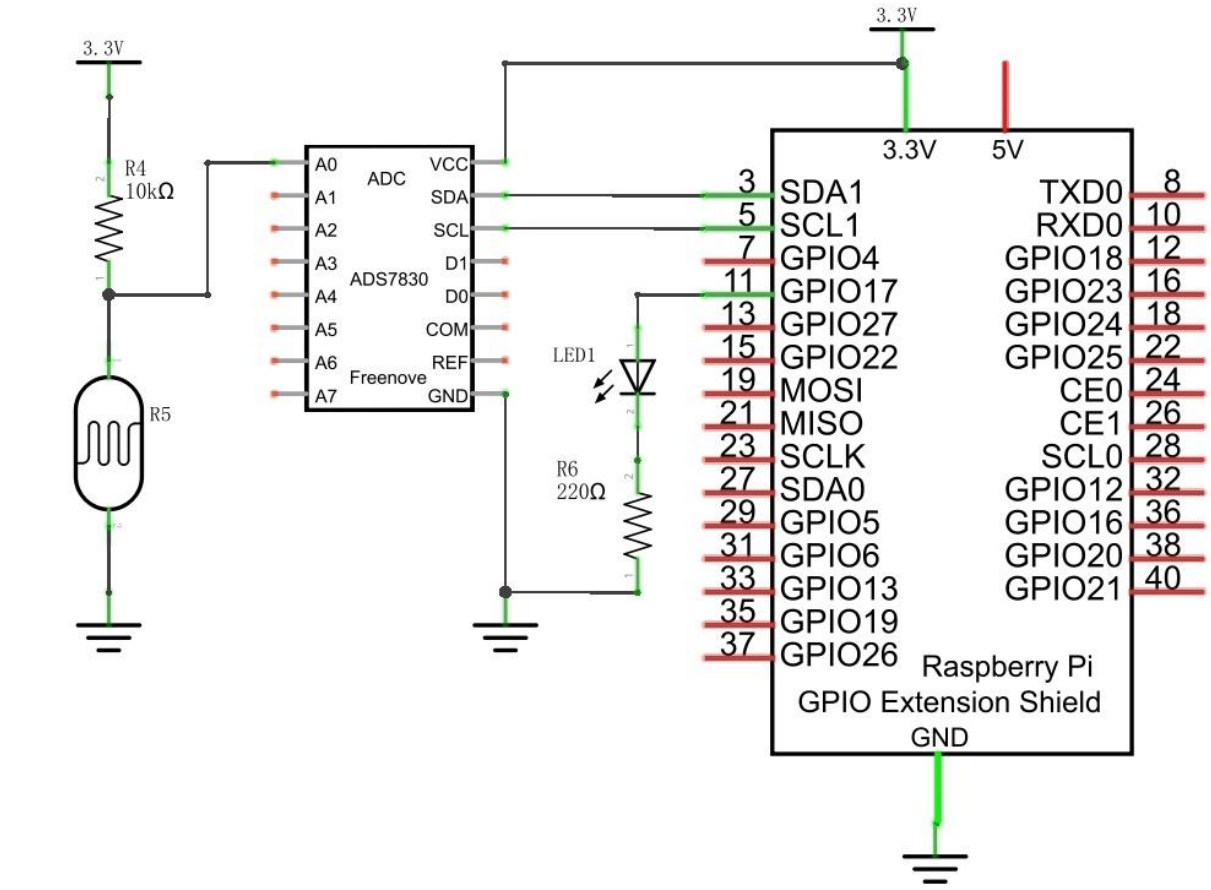
Hardware connection



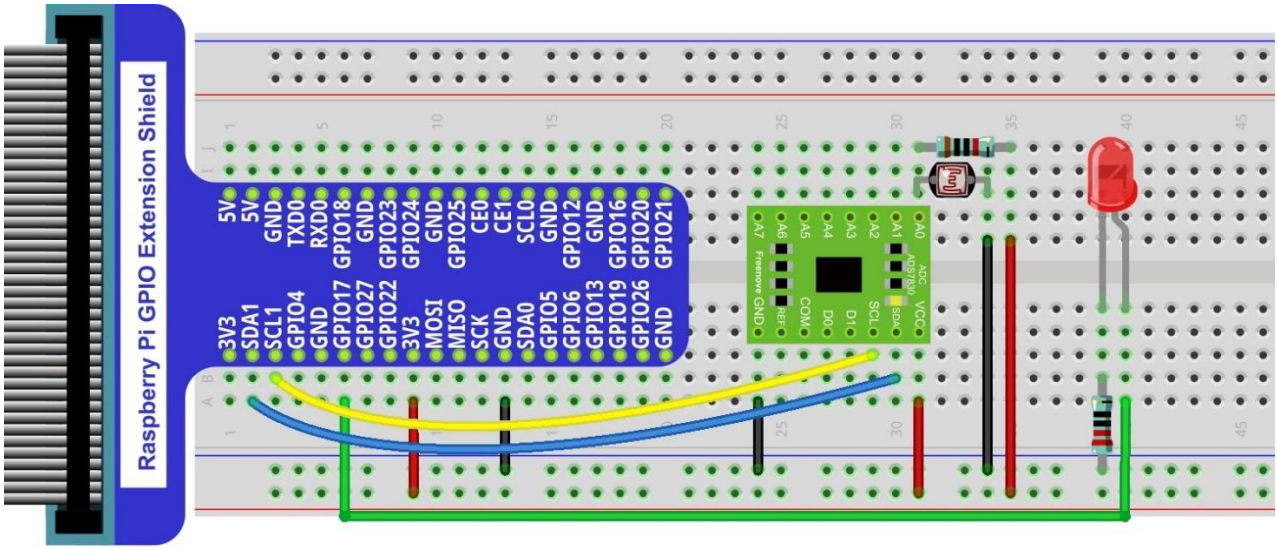
Circuit with ADS7830

The circuit used is similar to the Soft light project. The only difference is that the input signal of the AIN0 pin of ADC changes from a Potentiometer to a combination of a Photoresistor and a Resistor.

Schematic diagram



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



Code

The code used in this project is identical with what was used in the last chapter.

C Code 10.1.1 Nightlamp

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 10.1.1_Nightlamp directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/10.1.1_Nightlamp
```

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

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

3. Then run the generated file "Nightlamp".

```
sudo ./Nightlamp
```

After the program is executed, if you cover the Photoresistor or increase the light shining on it, the brightness of the LED changes accordingly. As in previous projects the Terminal window will display the current input voltage value of ADC module A0 pin and the converted digital quantity.

The following is the program code:

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

#define ledPin 0

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");
```

```
26          return -1;
27      }
28      wiringPiSetup();
29      softPwmCreate(ledPin,0,100);
30      while(1){
31          int value = adc->analogRead(0);  //read analog value of A0 pin
32          softPwmWrite(ledPin,value*100/255);
33          float voltage = (float)value / 255.0 * 3.3;  // calculate voltage
34          printf("ADC value : %d  ,\tVoltage : %.2fV\n",value,voltage);
35          delay(100);
36      }
37      return 0;
38  }
```

Python Code 10.1.1 Nightlamp

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 10.1_Nightlamp directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/10.1.1_Nightlamp
```

2. Use the python command to execute the Python code "Nightlamp.py".

```
python Nightlamp.py
```

After the program is executed, if you cover the Photoresistor or increase the light shining on it, the brightness of the LED changes accordingly. As in previous projects the Terminal window will display the current input voltage value of ADC module A0 pin and the converted digital quantity.

The following is the program code:

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

ledPin = 11 # define ledPin
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(ledPin, GPIO.OUT)   # set ledPin to OUTPUT mode
    GPIO.output(ledPin, GPIO.LOW)

    p = GPIO.PWM(ledPin, 1000) # set PWM Frequence to 1kHz
    p.start(0)

def loop():
    while True:
        value = adc.analogRead(0)    # read the ADC value of channel 0
        p.ChangeDutyCycle(value*100/255)
        voltage = value / 255.0 * 3.3
```

```
32          print ('ADC Value : %d, Voltage : %.2f'%(value,voltage))
33          time.sleep(0.01)
34
35  def destroy():
36      adc.close()
37      GPIO.cleanup()
38
39  if __name__ == '__main__':   # Program entrance
40      print ('Program is starting ... ')
41      setup()
42      try:
43          loop()
44      except KeyboardInterrupt:  # Press ctrl-c to end the program.
45          destroy()
```

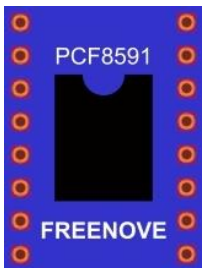
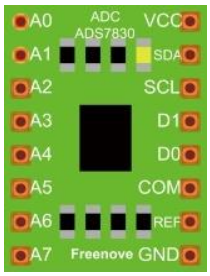
Chapter 11 Thermistor

In this chapter, we will learn about Thermistors which are another kind of Resistor.

Project 11.1 Thermometer

A Thermistor is a type of Resistor whose resistance value is dependent on temperature and changes in temperature. Therefore, we can take advantage of this characteristic to make a Thermometer.

Component List

Raspberry Pi x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1		Jumper Wire M/M x14 
Thermistor x1 	ADC module x1  or 	Resistor 10kΩ x3 

Component knowledge

Thermistor

Thermistor is a temperature sensitive resistor. When it senses a change in temperature, the resistance of the Thermistor will change. We can take advantage of this characteristic by using a Thermistor to detect temperature intensity. A Thermistor and its electronic symbol are shown below.


The relationship between resistance value and temperature of a thermistor is:

$$Rt=R*EXP\ [B*(1/T2-1/T1)]$$

Where:

Rt is the thermistor resistance under T2 temperature;

R is in the nominal resistance of thermistor under T1 temperature;

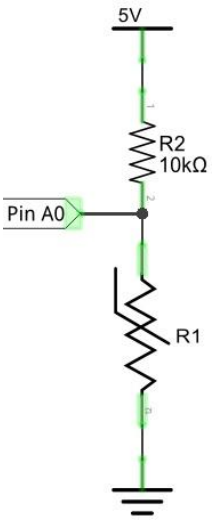
EXP[n] is nth power of e;

B is for thermal index;

T1, T2 is Kelvin temperature (absolute temperature). Kelvin temperature=273.15 + Celsius temperature.

For the parameters of the Thermistor, we use: B=3950, R=10k, T1=25.

The circuit connection method of the Thermistor is similar to photoresistor, as the following:



We can use the value measured by the ADC converter to obtain the resistance value of Thermistor, and then we can use the formula to obtain the temperature value.

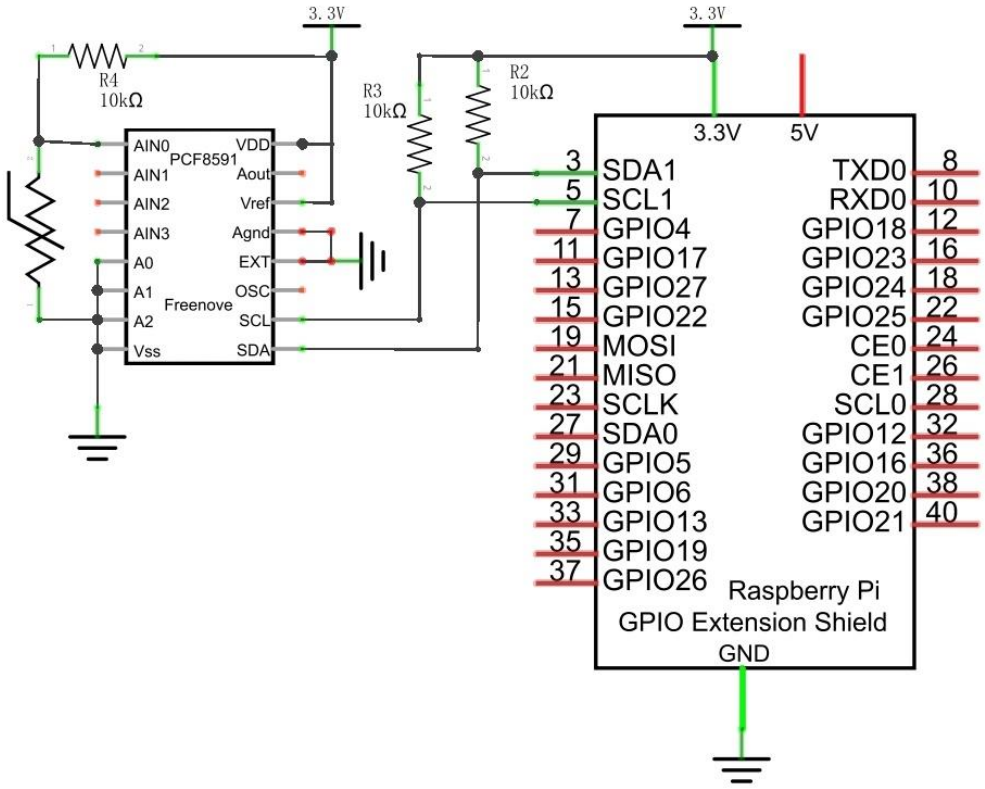
Therefore, the temperature formula can be derived as:

$$T2 = 1/(1/T1 + \ln(Rt/R)/B)$$

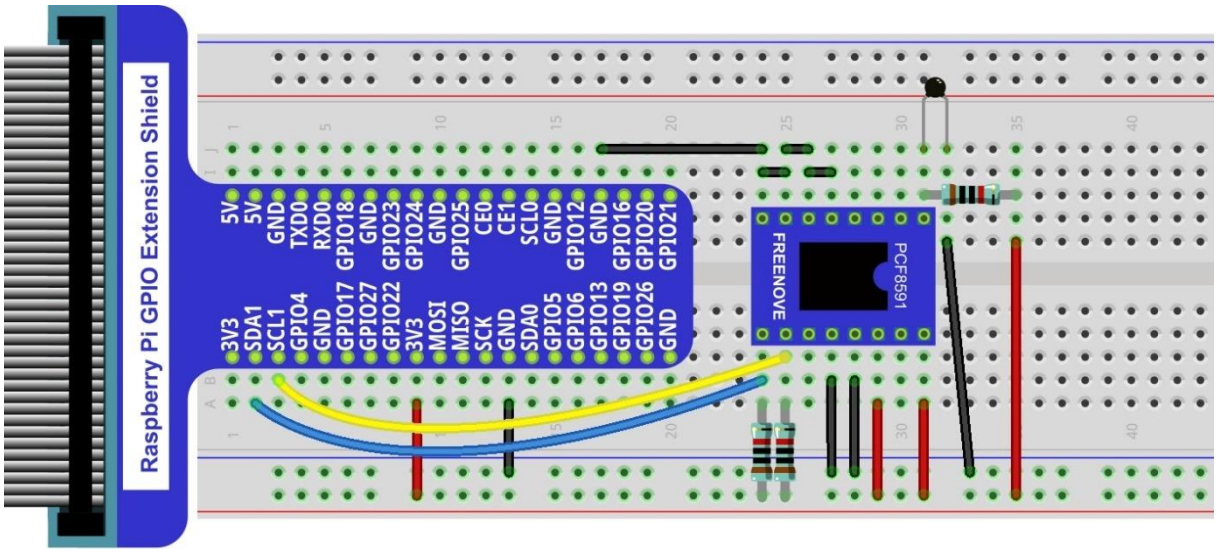
Circuit with PCF8591

The circuit of this project is similar to the one in the last chapter. The only difference is that the Photoresistor is replaced by the Thermistor.

Schematic diagram



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

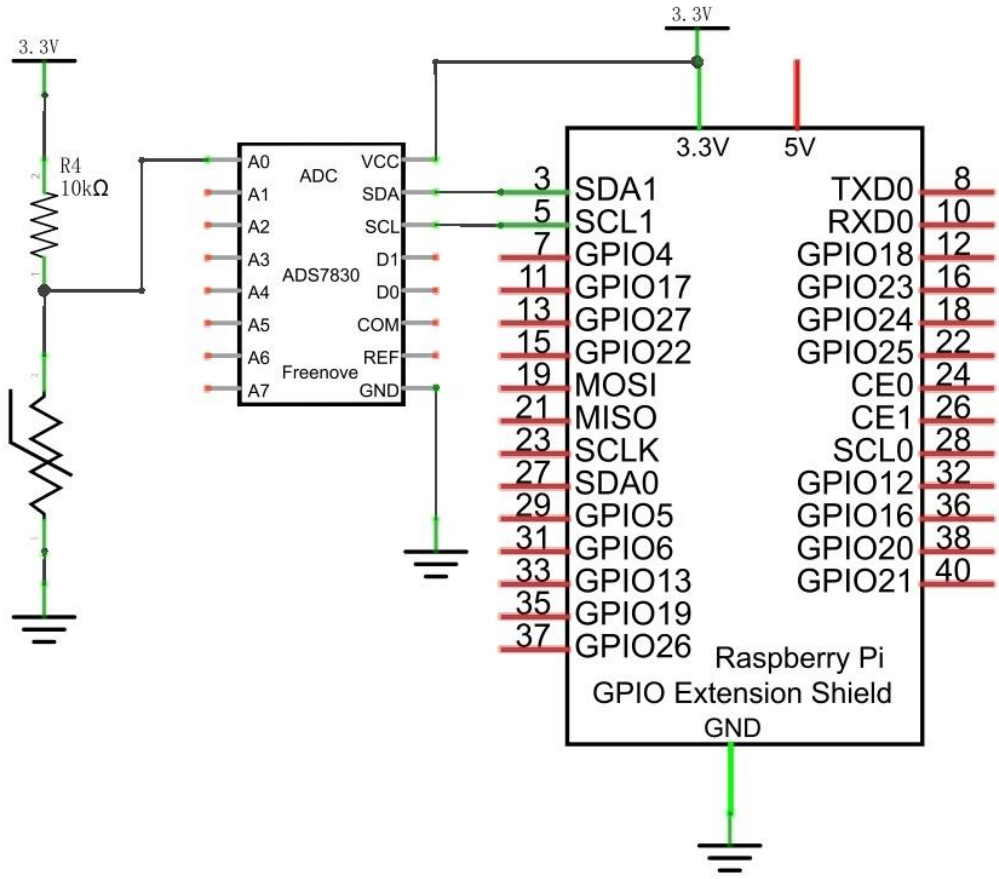


Thermistor has longer pins than the one shown in circuit.

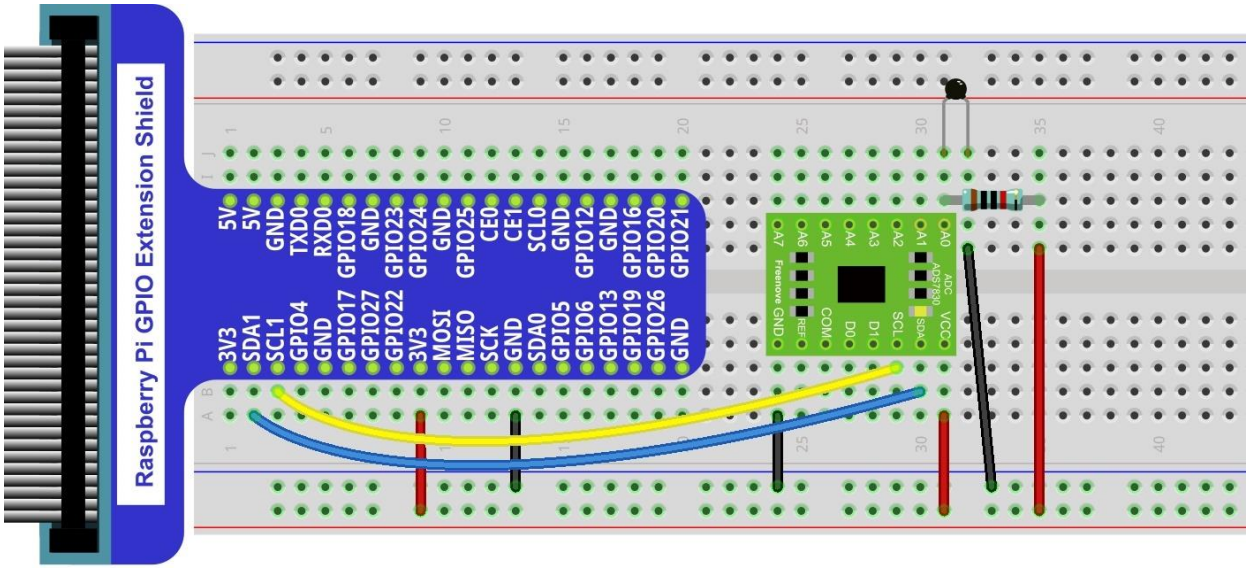
Circuit with ADS

The circuit of this project is similar to the one in last chapter. The only difference is that the Photoresistor is replaced by the Thermistor.

Schematic diagram



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



Thermistor has longer pins than the one shown in circuit.

Code

In this project code, the ADC value still needs to be read, but the difference here is that a specific formula is used to calculate the temperature value.

C 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 C code.

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

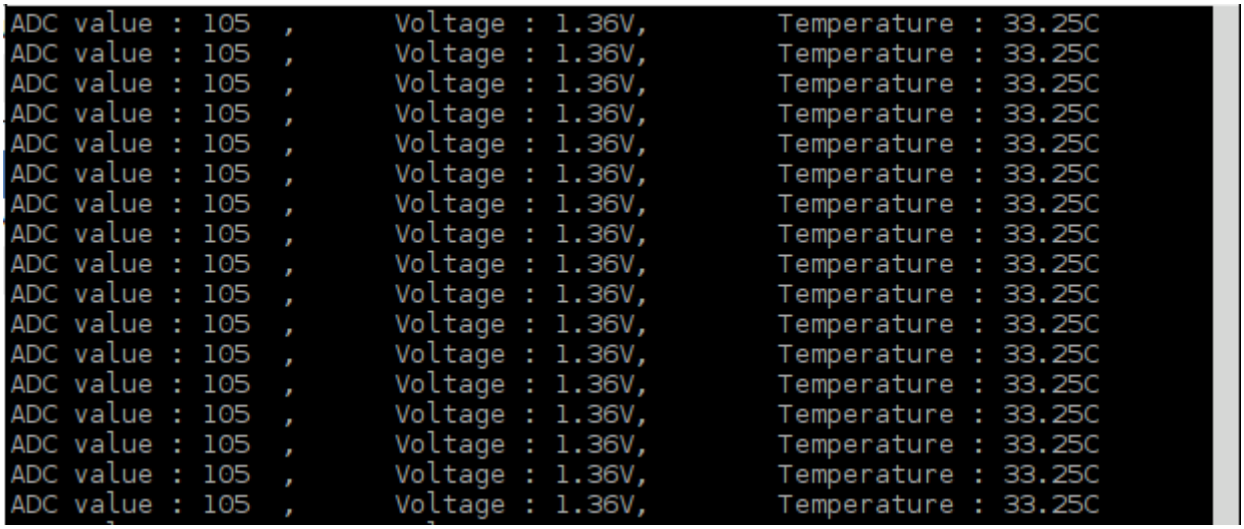
2 Use following command to compile "Thermometer.cpp" and generate executable file "Thermometer".

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

3 Then run the generated file "Thermometer".

```
sudo ./Thermometer
```

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.



The following is the code:

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

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

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