Free your innovation

Freenove is an open-source electronics platform.
www.freenove.com

About

Freenove is an open-source electronics platform. Freenove is committed to helping customer quickly realize the creative idea and product prototypes, making it easy to get started for enthusiasts of programing and electronics and launching innovative open source products. Our services include:

- Electronic components and modules
- Learning kits for Arduino
- Learning kits for Raspberry Pi
- Learning kits for Technology
- Robot kits
- Auxiliary tools for creations

Our code and circuit are open source. You can obtain the details and the latest information through visiting the following web sites:
http://www.freenove.com
https://github.com/freenove

Your comments and suggestions are warmly welcomed, please send them to the following email address:
support@freenove.com

Support

Freenove provides free and quick technical support, including but not limited to:

- Quality problems of products
- Problems in using products
- Questions for learning and technology
- Opinions and suggestions
- Ideas and thoughts

Please send email to:
support@freenove.com
On working day, we usually reply to you within 24 hours.

Copyright

Contents

Chapter 0 Procesing

Processing software is used to write programs that can run on computers. Processing software is free and open source, and runs on the Mac, Windows, and GNU/Linux platforms, which is the same with Arduino software. In fact, the development of Arduino software is based on Processing software, and they still have similar interface.

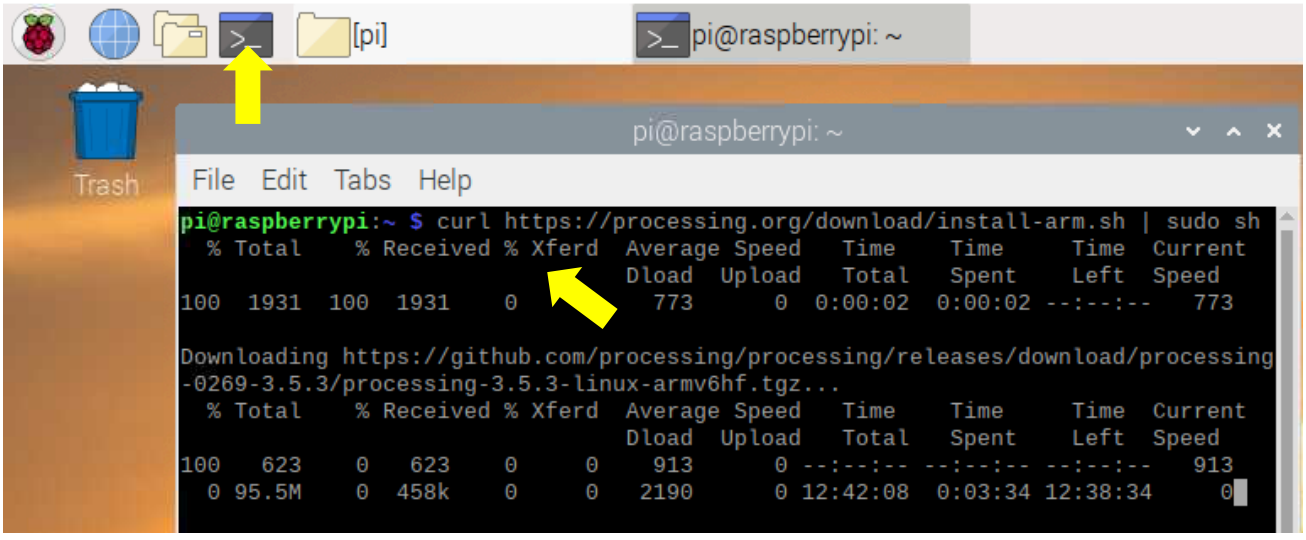
Programs written using Processing are also called sketches, and Java is the default language. Java language and C++ language has many similarities, so the readers who have learned our basic tutorial are able to understand and write simple Processing sketches quickly.

This tutorial will introduce how to install and use processing software on Raspberry Pi through some electronic circuit projects. Chapters and sequence is similar to C and python tutorial. Equally, detailed description and explanation is arranged for the skect in each project. Our elaborate electronic circuits and interactive project with Processing are attached in the end, including virtual instruments, games (2D and 3D versions), etc.

Install Processing Software

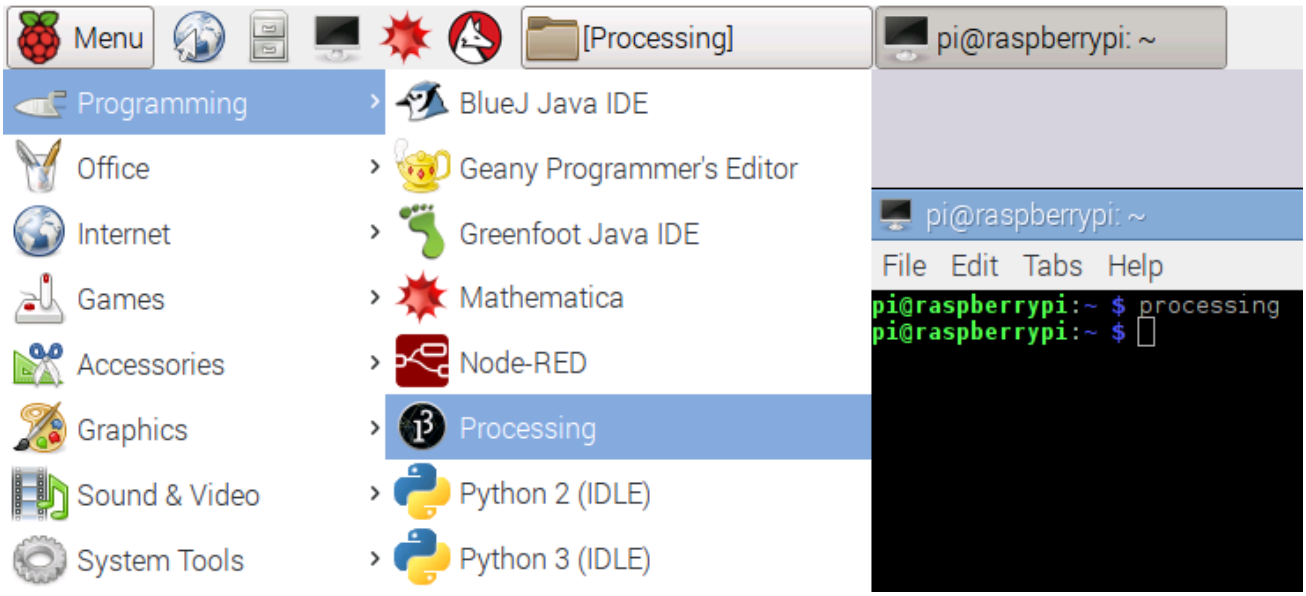
Processing software / Processing Development Environment (PDE) makes it easy to write Processing programs. First install Processing software: type following command in the terminal to start installation:

```
curl https://processing.org/download/install-arm.sh | sudo sh
```

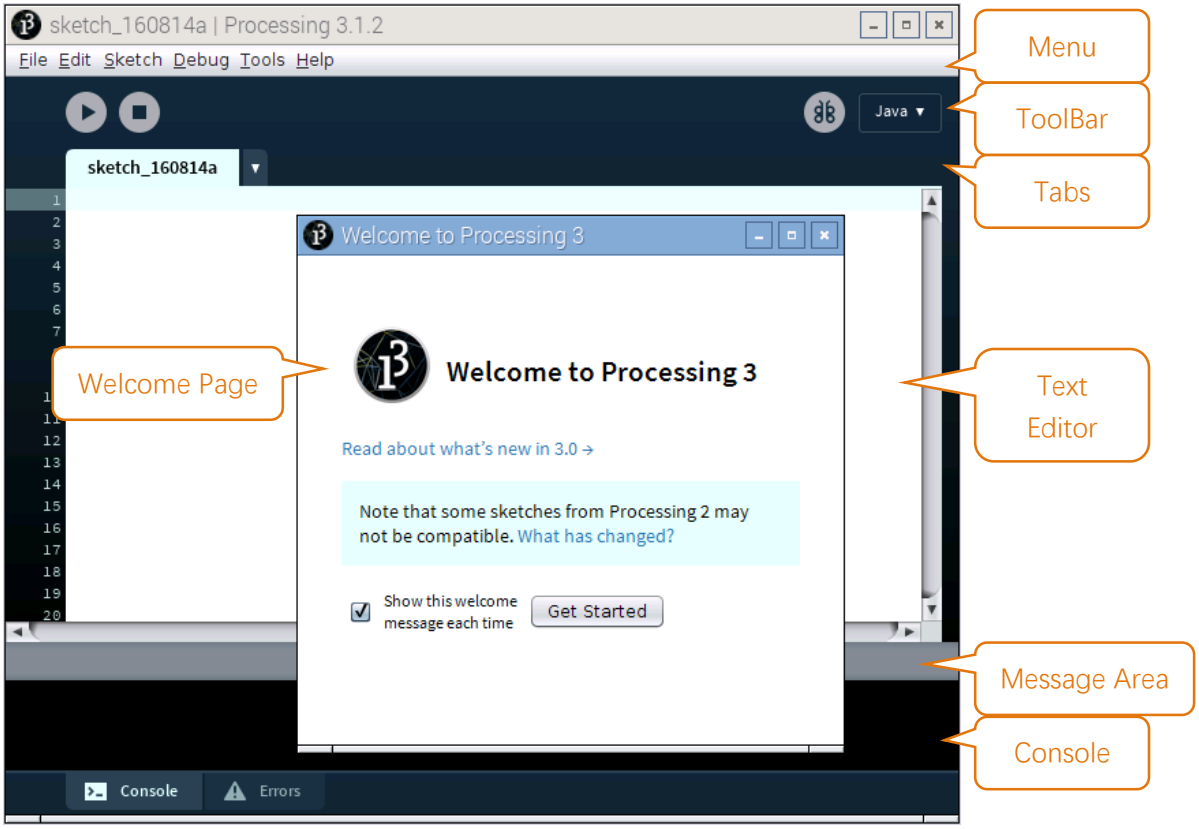


Ensures that your RPi always has the Internet to access in the installation process.
You can also download and install the software by visiting the official website https://processing.org/.

After the installation is completed, you can enter the "processing" to open processing software in any directory of the terminal, or open the software processing in the start menu of the system, as shown below:



Interface of processing software is shown below:



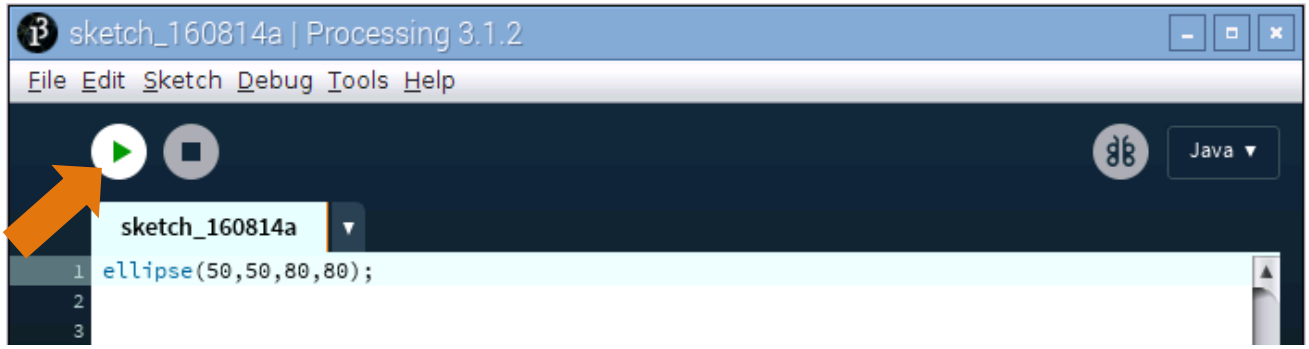
You're now running the Processing Development Environment (or PDE). There's not much to it; the large area is the Text Editor, and there's a row of buttons across the top; this is the toolbar. Below the editor is the Message Area, and below that is the Console. The Message Area is used for one line messages, and the Console is used for more technical details.

First Use

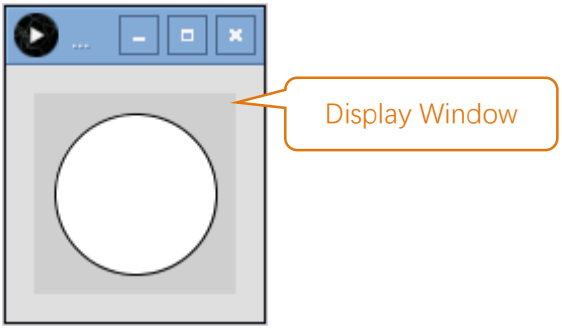
In the editor, type the following:

```
1  ellipse(50, 50, 80, 80);
```

This line of code means "draw an ellipse, with the center 50 pixels over from the left and 50 pixels down from the top, with a width and height of 80 pixels." Click the Run button (the triangle button in the Toolbar).

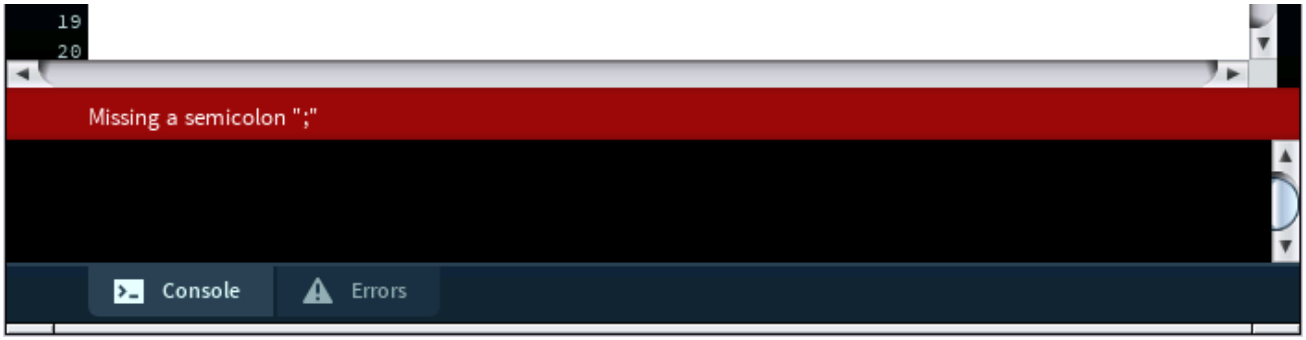


If you've typed everything correctly, you'll see a circle on your screen.



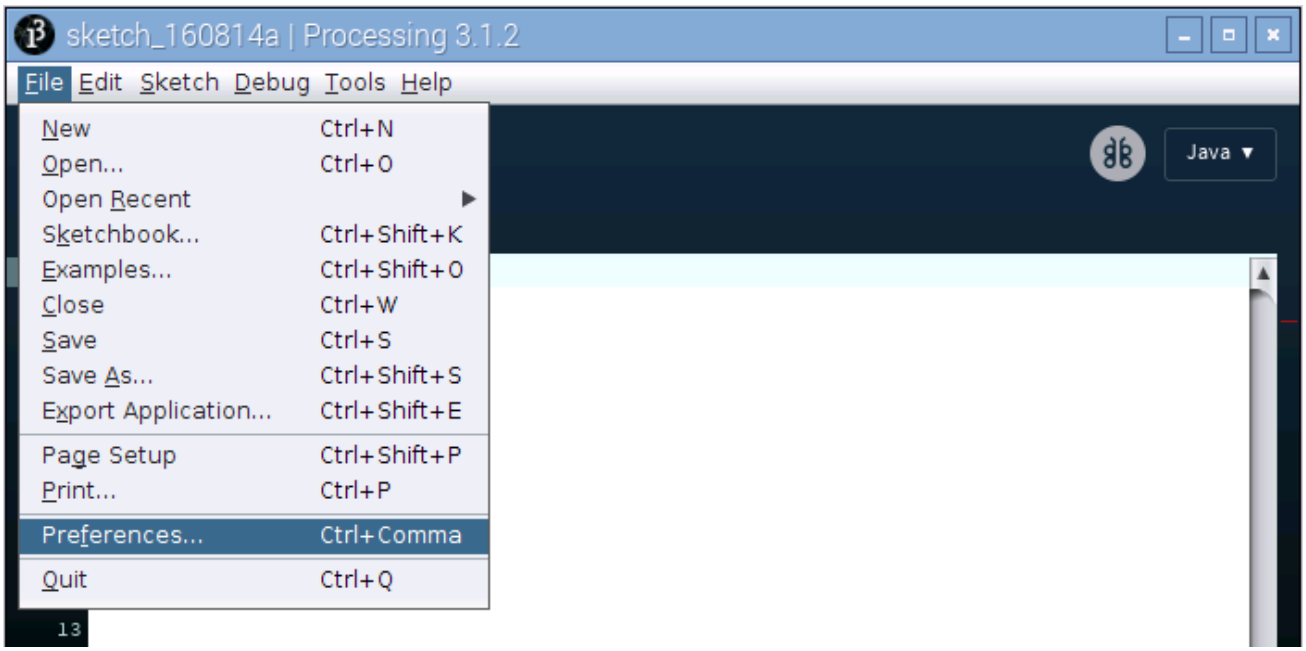
Click on "Stop" (the rectangle button in the Toolbar) or "Close" on Display Window to stop running the program.

If you didn't type it correctly, the Message Area will turn red and complain about an error. If this happens, make sure that you've copied the example code exactly: the numbers should be contained within parentheses and have commas between each of them, and the line should end with a semicolon.



You can export this sketch to an application to run it directly without opening the Processing.

To export the sketch to the application, you must first save it.



So far, we have completed the first use. I believe you have felt the joy of it.

Chapter 1 LED

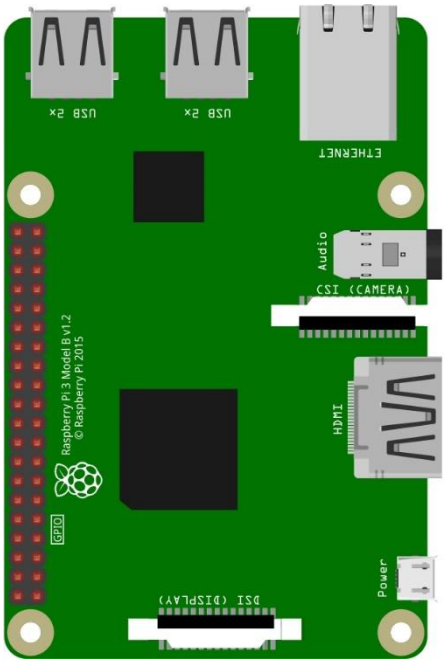
We will still start from Blink LED in this chapter, and also learn the usage of some commonly used functions of Processing Software.

Project 1.1 Blink

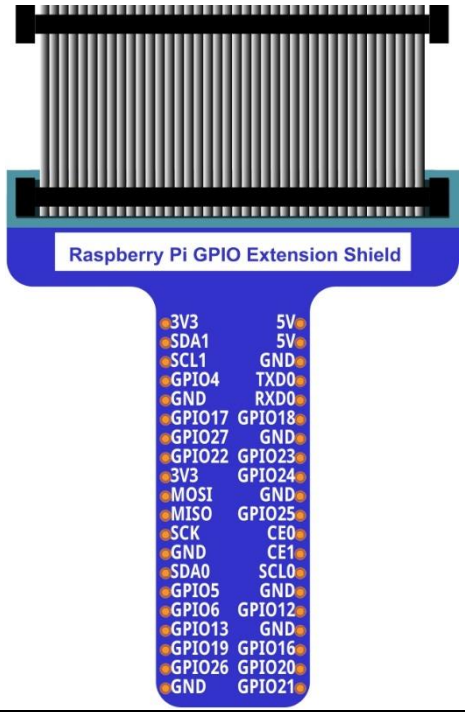
In this project, we will make a Blink LED and let Display window of Processing Blink at the same time.

Component List

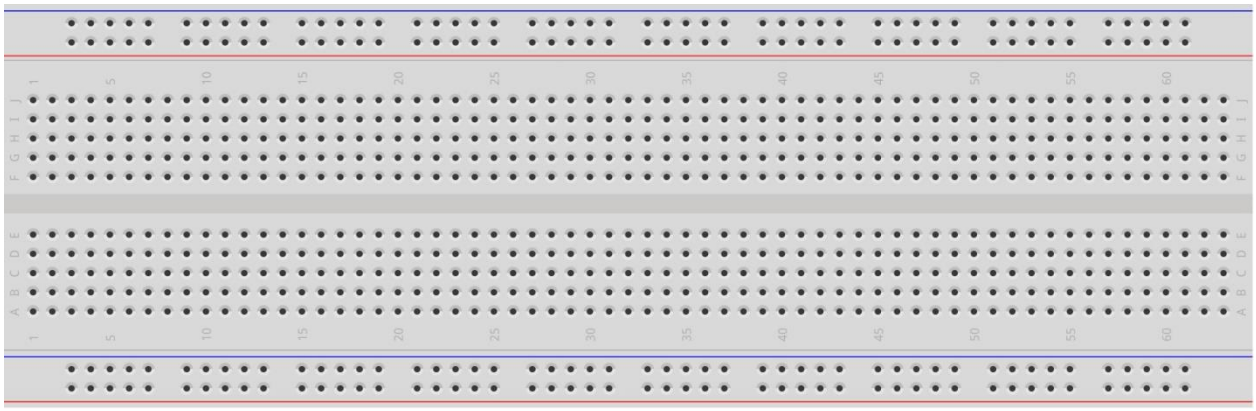
Raspberry Pi x1



GPIO Extension Board & Wire x1



BreadBoard x1



LED x1

Resistor 220Ω x1

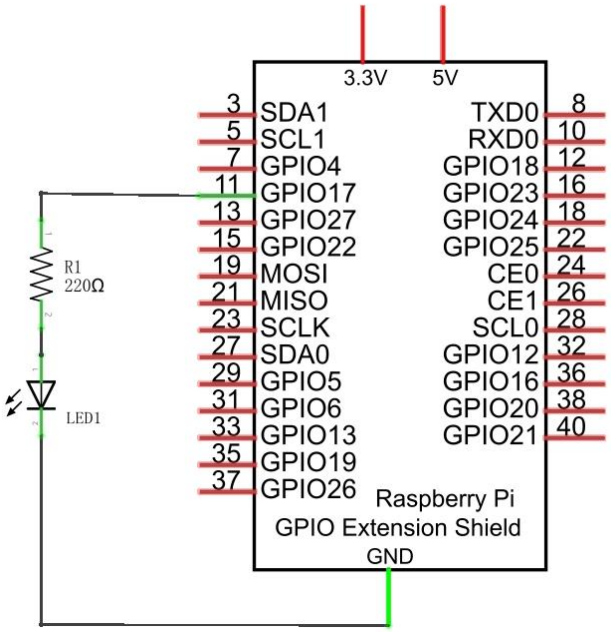
Jumper Wire M/M x2

In the components list, Raspberry Pi, GPIO Extension Shield and Breadboard are necessary for each experiment. They will be listed only in text form.

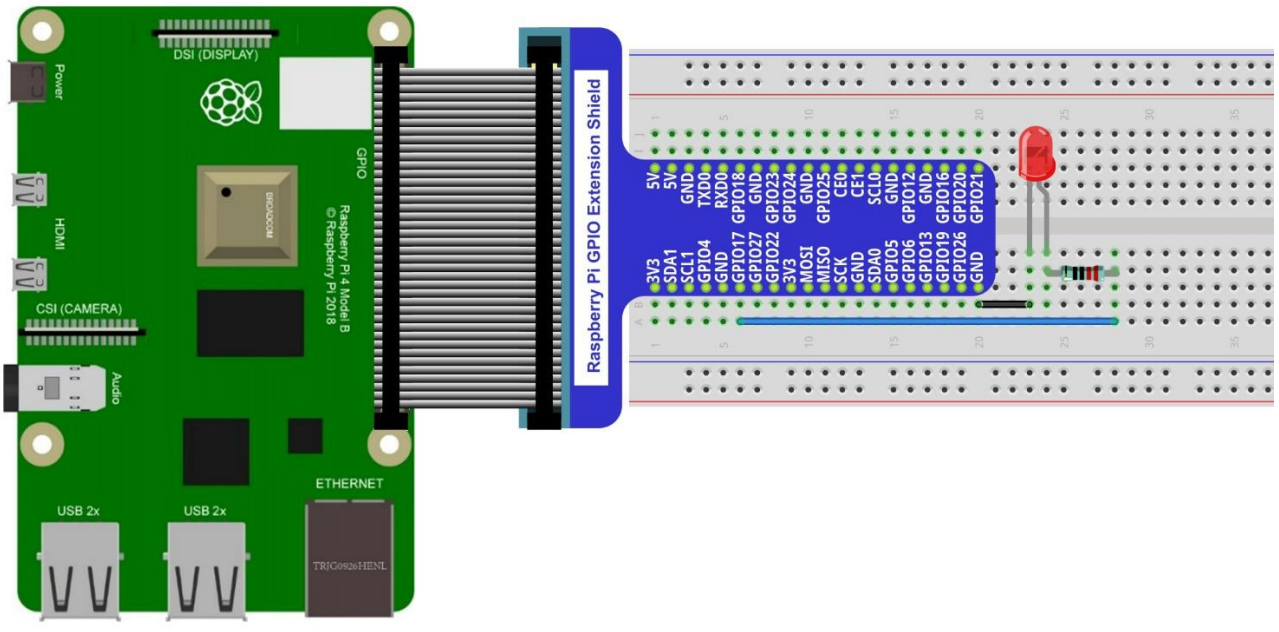
Circuit

Disconnect RPi from GPIO Extension Shield first. Then build the circuit according to the circuit diagram and the hardware connection diagram. After the circuit is built and confirmed, connect RPi to GPIO Extension Shield. In addition, short circuit (especially 5V and GND, 3.3V and GND) should be avoid, because short circuit may cause anormal circuit work, or even damage to PRi.

Schematic diagram



Hardware connection



Because Numbering of GPIO Extension Shield is the same as RPi GPIO, latter Hardware connection diagram will only show the part of breadboard and GPIO Extension Shield.

Sketch

Sketch 1.1.1 Blink

Because the resource folder name is too long. For convenience, the folder will be named to "Freenove_Kit". If you have already renamed it, skip this command. Assume the absolute path is "/ home / pi" or "~ /". Execute the following command in the user directory.

```
mv Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/ Freenove_Kit/
```

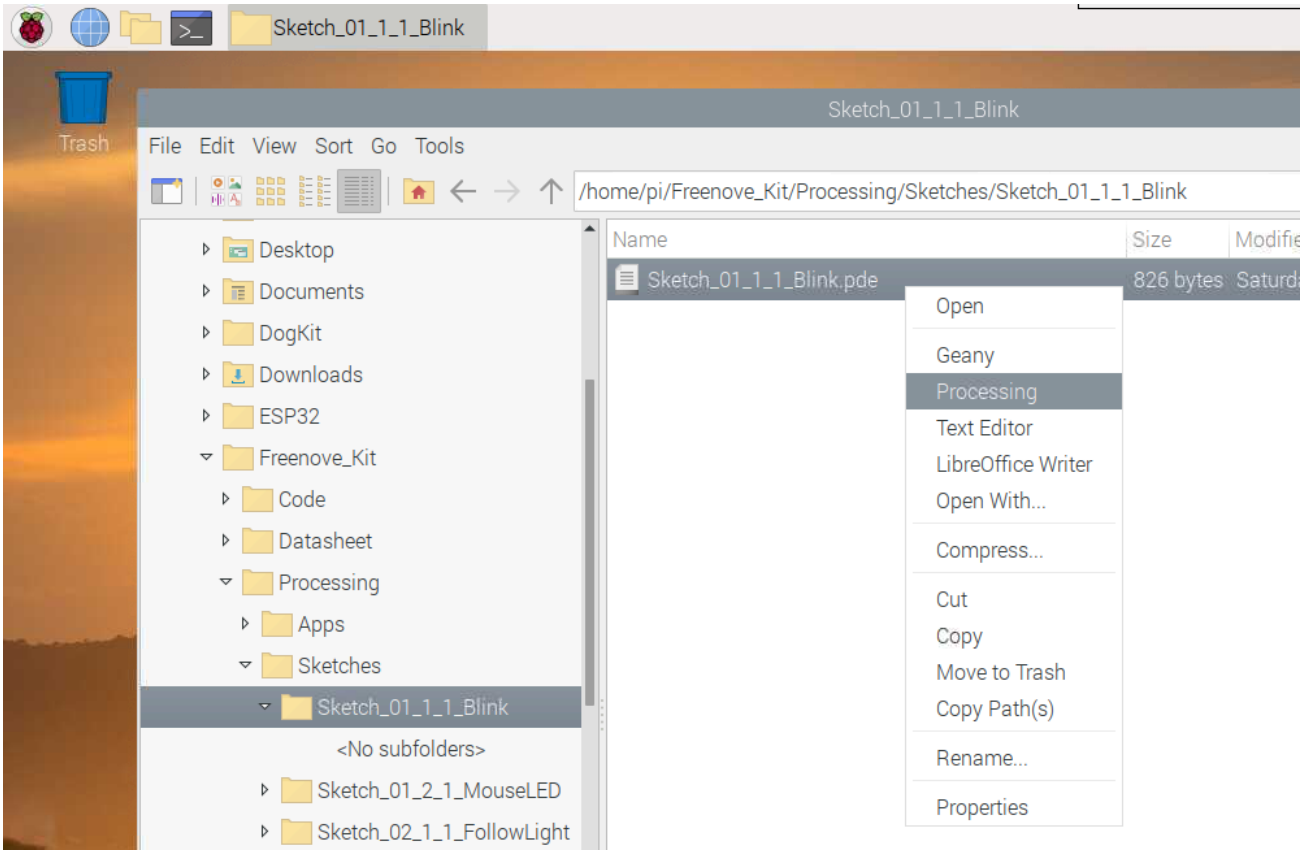
First observe the running result of the sketch, and then analyze the code.

1. Use Processing to open the file Sketch_01_1_1_Blink. (Following is only one line of command. There is a Space after processing.)

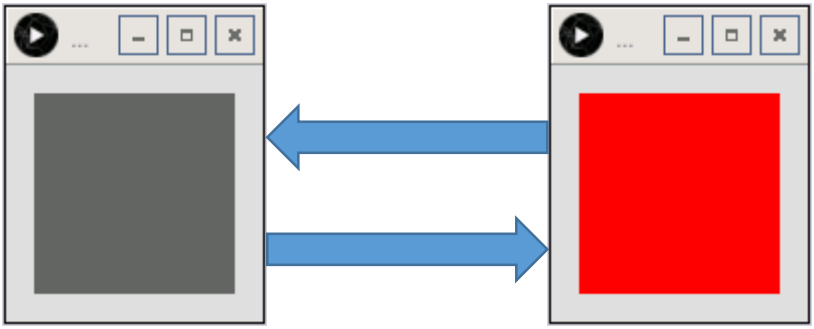
```
Processing ~/Freenove_Kit/Processing/Sketches/Sketch_01_1_1_Blink/Sketch_01_1_1_Blink.pde
```

2. Click on "RUN" to run the code.

You can also open it as below.
Click Raspberry Pi file manerger. Find the file under path below:
/home/pi/Freenove_Kit/Processing/Sketches/Sketch_01_1_1_Blink
Then right click and choose Processing.



After the program is executed, LED will start Blinking and background of Display window will change with the change of LED state.



The following is program code:

```
import processing.io.*;

int ledPin = 17;    //define ledPin
boolean ledState = false;    //define ledState

void setup() {
  size(100, 100);
  frameRate(1);         //set frame rate
  GPIO.pinMode(ledPin, GPIO.OUTPUT);    //set the ledPin to output mode
}

void draw() {
  ledState = !ledState;
  if (ledState) {
    GPIO.digitalWrite(ledPin, GPIO.HIGH);    //led on
    background(255, 0, 0); //set the fill color of led on
  } else {
    GPIO.digitalWrite(ledPin, GPIO.LOW);    //led off
    background(102); //set the fill color of led off
  }
}
```

Processing code usually have two functions: setup() and draw(), where the function setup()is only executed once, but the function draw() will be executed circularly. In the function setup(), size(100, 100) specifies the size of the Display Window to 100x100pixl. FrameRate(1) specifies the refresh rate of Display Window to once per second, namely, the draw() function will be executed once per second. GPIO.pinMode (ledPin, GPIO.OUTPUT) is used to set ledPin to output mode.

```
void setup() {
  size(100, 100);
  frameRate(1);         //set frame rate
  GPIO.pinMode(ledPin, GPIO.OUTPUT);    //set the ledPin to output mode
}
```

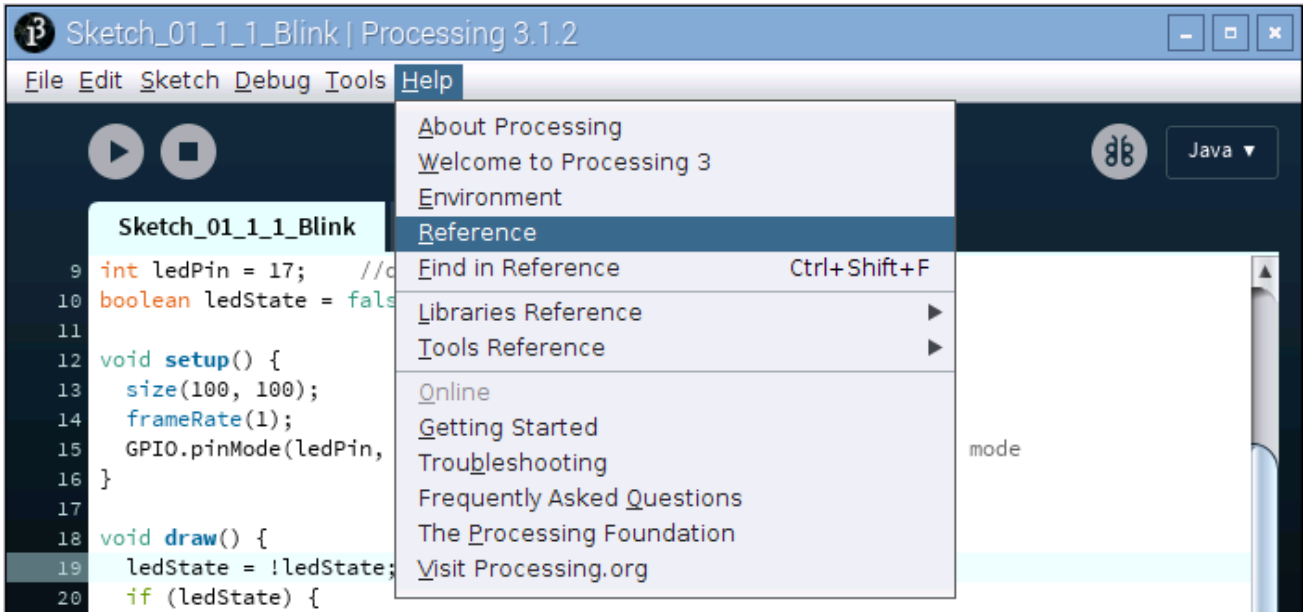
In draw() function, each execution will invert the variable "ledState". When “ledState” is true, LED is turned on, and the background color of display window is set to red. And when the “ledState” is false, the LED will be turned off and the background color of display window is set to gray. Since the function draw() is executed once per second, the background color of Display Window and the state of LED will also change once per second. Such cycle repeats itself to achieve the effect of blink.

```
void draw() {
  ledState = !ledState;
  if (ledState) {
    GPIO.digitalWrite(ledPin, GPIO.HIGH);    //led on
    background(255, 0, 0); //set the fill color of led on
  } else {
    GPIO.digitalWrite(ledPin, GPIO.LOW);    //led off
    background(102); //set the fill color of led off
  }
}
```

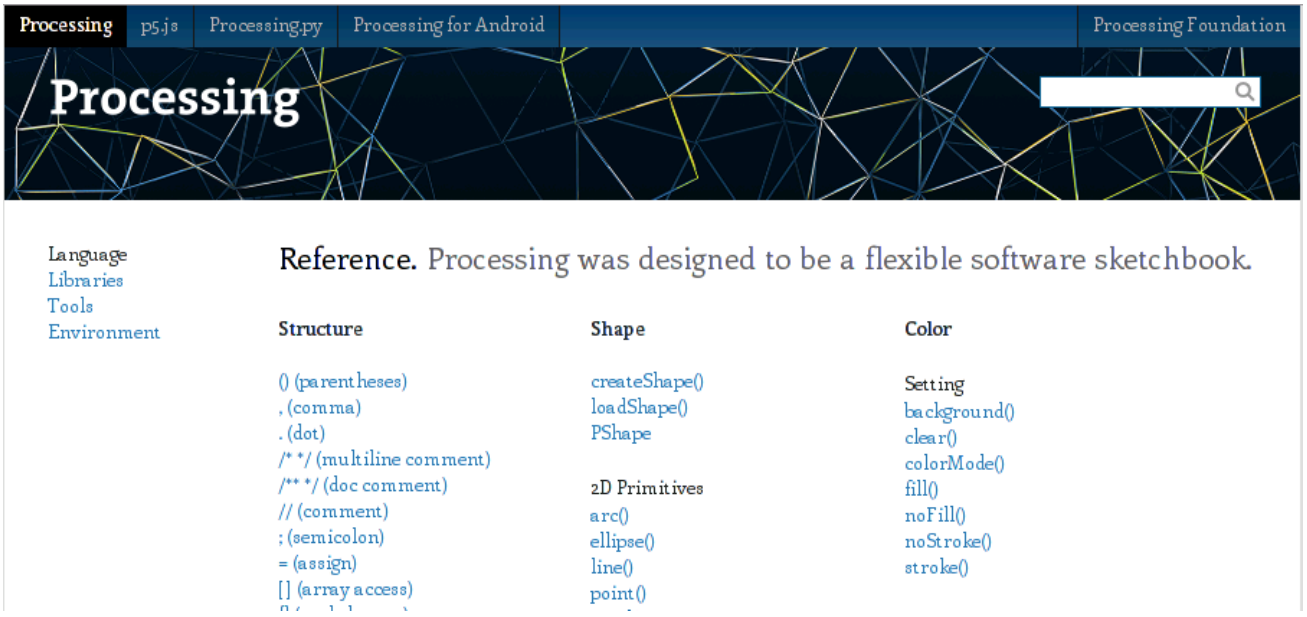
The following is simple description of some functions:

setup()
The setup() function is run once, when the program starts.
draw()
Called directly after setup(), the draw() function continuously executes the lines of code contained inside its block until the program is stopped or noLoop() is called. draw() is called automatically and should never be called explicitly.
size()
Defines the dimension of the display window width and height in units of pixels
framerate()
Specifies the number of frames to be displayed every second
background()
Set the color used for the background of the Processing window.
GPIO.pinMode()
Configures a pin to act either as input or output
GPIO.digitalWrite()
Sets an output pin to be either high or low

All functions used in this code can be found in the Reference of Processing Software, in which built-in functions are described in details, and there are some sample programs. It is recommended for beginners to view more usage and functions of the function. The localization of Reference can be opened by the following steps: click the menu bar "Help"→"Reference".



Then the following page will be displayed in the web browser:



Or directly access to the official website for reference:http://processing.org/reference/

Project 1.2 MouseLED

In this project, we will use the mouse to control the state of LED.
The components and circuits of this project are the same as the last section.

Sketch

Sketch 1.2.1 MouseLED

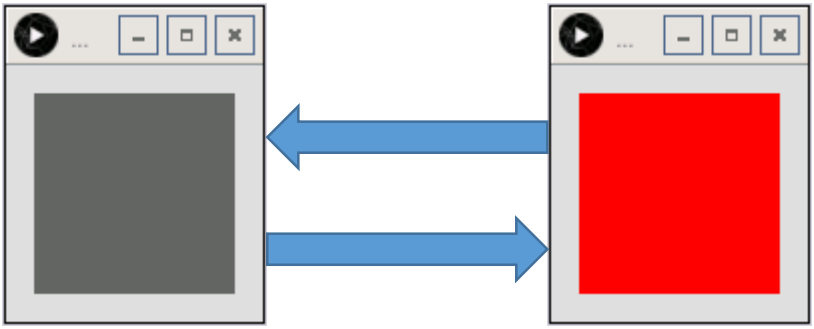
First observe the running result of the sketch, and then analyze the code.

1. Use Processing to open the file Sketch_01_2_1_MouseLED.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_01_2_1_MouseLED/Sketch_01_2_1_MouseLED.pde
```

2. Click on "RUN" to run the code.

After the program is executed, the LED is under off state, and background color of Display window is gray. Click on Display Window with the mouse, then LED is turned on and Display window background color become red. Click on the Display Window again, then the LED is turned off and the background color become gray, as shown below.



The following is program code:

```
import processing.io.*;

int ledPin = 17;
boolean ledState = false;
void setup() {
  size(100, 100);
  GPIO.pinMode(ledPin, GPIO.OUTPUT);
  background(102);
}

void draw() {
  if (ledState) {
    GPIO.digitalWrite(ledPin, GPIO.HIGH);
    background(255,0,0);
  } else {
```

```
16      GPIO.digitalWrite(ledPin, GPIO.LOW);
17      background(102);
18    }
19  }
20
21  void mouseClicked() { //if the mouse Clicked
22    ledState = !ledState;  //Change the led State
23  }
```

The function mouseClicked() is used in this code. The function is used to capture the mouse click events, which is executed when the mouse is clicked on. We can change the state of the variable “ledState” in this function, to realize controlling LED through clicking on the mouse.

```
  void mouseClicked() { //if the mouse Clicked
    ledState = !ledState;  //Change the led State
  }
```

Chapter 2 LEDBar Graph

We have learned how to control a LED, and next we will learn how to control a number of LED.

Project 2.1 FollowLight

In this project, we will use the mouse to control the LEDBar Graph

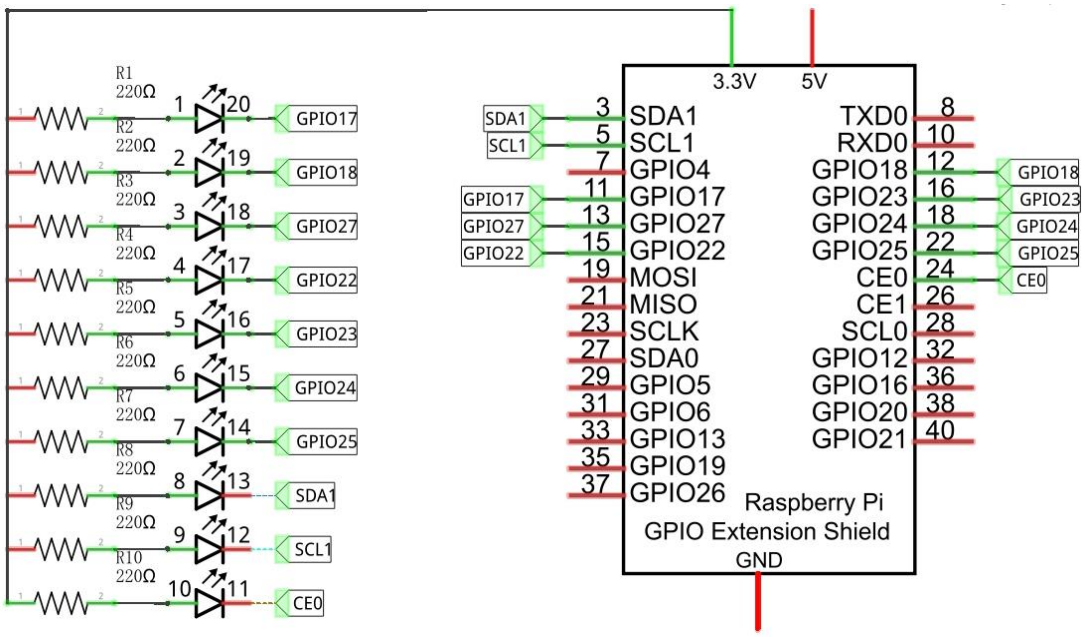
Component List

Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	LED bar graph x1 	Resistor 220Ω x10 
Jumper M/M x11 		

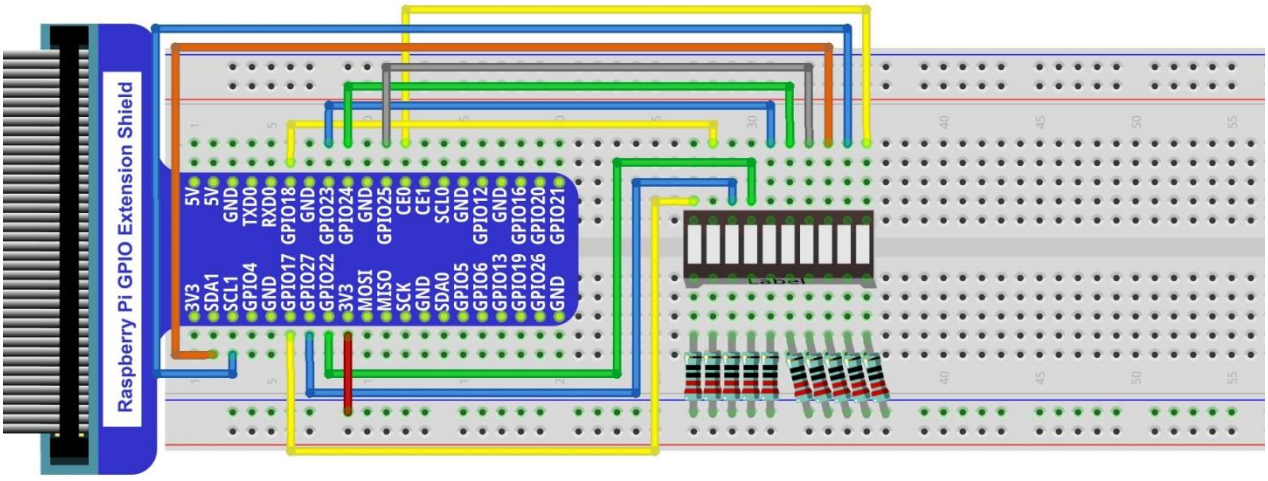
Circuit

The network label is used in the circuit diagram below, and the pins with the same network label are connected togather.

Schematic diagram



Hardware connection



In this circuit, the cathode of LED is connected to GPIO, which is the different from the front circuit. So, LED will be turned on when GPIO output low level in the program.

Sketch

Sketch 2.1.1 FollowLight

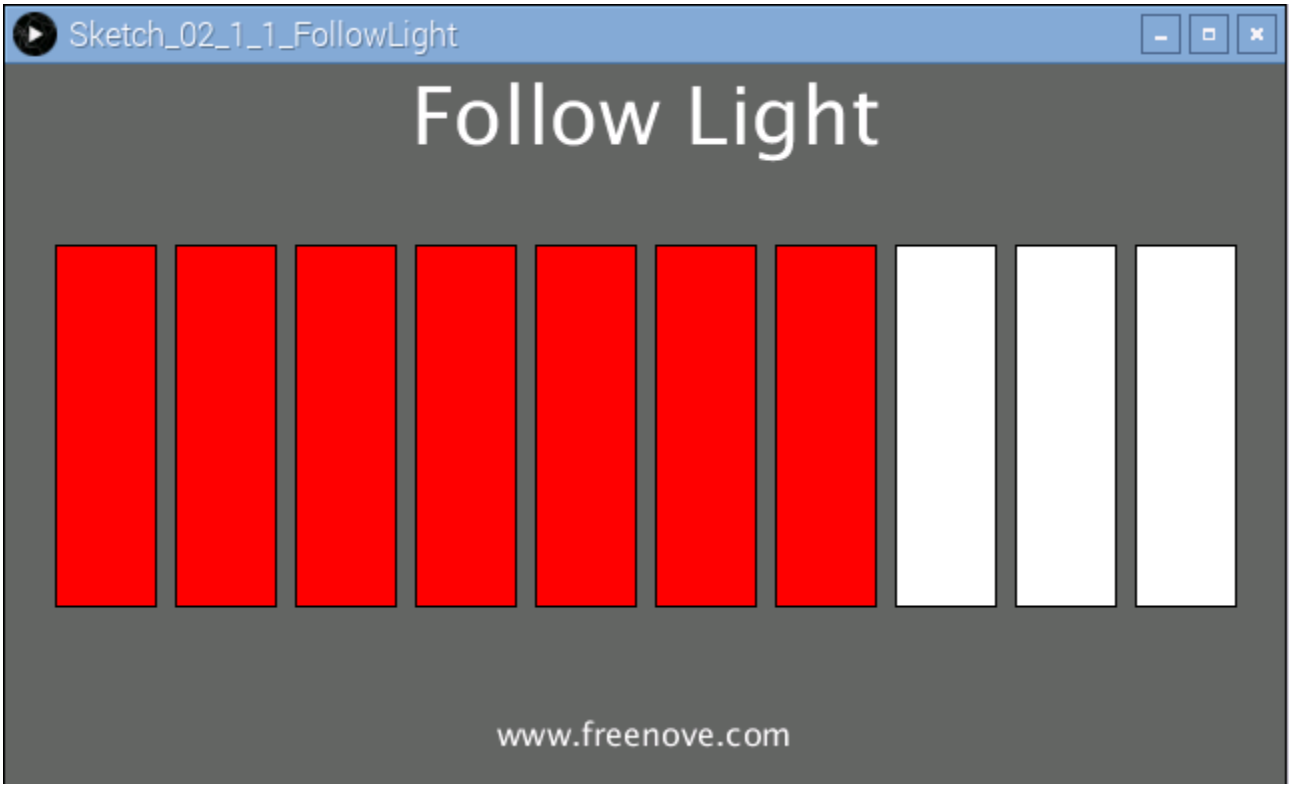
First observe the running result of the sketch, and then analyze the code.

1. Use Processing to open the file Sketch_02_1_1_FollowLight.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_02_1_1_FollowLight/Sketch_02_1_1_FollowLight.pde
```

2. Click on "RUN" to run the code.

After the program is executed, slide the mouse in the Display Window, then the state of LEDBar Graph will be changed, as shown below.



The following is program code:

```
import processing.io.*;

int leds[]={17, 18, 27, 22, 23, 24, 25, 2, 3, 8}; //define ledPins

void setup() {
  size(640, 360);  //display window size
  for (int i=0; i<10; i++) {  //set led Pins to output mode
    GPIO.pinMode(leds[i], GPIO.OUTPUT);
  }
  background(102);
```

```
11    textAlign(CENTER);    //set the text centered
12    textSize(40);         //set text size
13    text("Follow Light", width / 2, 40);    //title
14    textSize(16);
15    text("www.freenove.com", width / 2, height - 20);    //site
16  }
17
18  void draw() {
19    for (int i=0; i<10; i++) {    //draw 10 rectanglar box
20      if (mouseX>(25+60*i)) {    //if the mouse cursor on the right of rectanglar box
21        fill(255, 0, 0);         //fill the rectanglar box in red color
22        GPIO.digitalWrite(leds[i], GPIO.LOW);  //turn on the corresponding led
23      } else {
24        fill(255, 255, 255);  //else fill the rectanglar box in white color
25        GPIO.digitalWrite(leds[i], GPIO.HIGH);  //and turn off the led
26      }
27      rect(25+60*i, 90, 50, 180);    //draw a rectanglar box
28    }
29  }
```

In the function draw(), we draw 10 rectangles to represent 10 LEDs of LEDBar Graph. We make rectangles on the left of mouse filled with red, corresponding LEDs turned on. And make We make rectangles on the right of mouse filled with red, corresponding LEDs turned off. In this way, when slide the mouse to right, the more LEDs on the left of mouse will be turned on. When to the left, the reverse is the case.

```
  void draw() {
    for (int i=0; i<10; i++) {    //draw 10 rectanglar box
      if (mouseX>(25+60*i)) {    //if the mouse cursor on the right of rectanglar box
        fill(255, 0, 0);         //fill the rectanglar box in red color
        GPIO.digitalWrite(leds[i], GPIO.LOW);  //turn on the corresponding led
      } else {
        fill(255, 255, 255);  //else fill the rectanglar box in white color
        GPIO.digitalWrite(leds[i], GPIO.HIGH);  //and turn off the led
      }
      rect(25+60*i, 90, 50, 180);    //draw a rectanglar box
    }
  }
```

Chapter 3 PWM

In this chapter, we will learn how to use PWM.

Project 3.1 BreathingLED

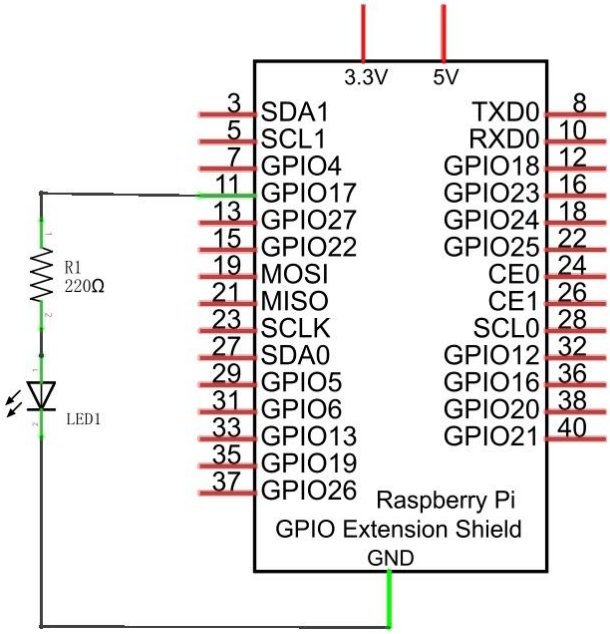
In this project, we will make a breathing LED, and the Display Window will show a breathing LED pattern and a progress bar, at the same time.

Component List

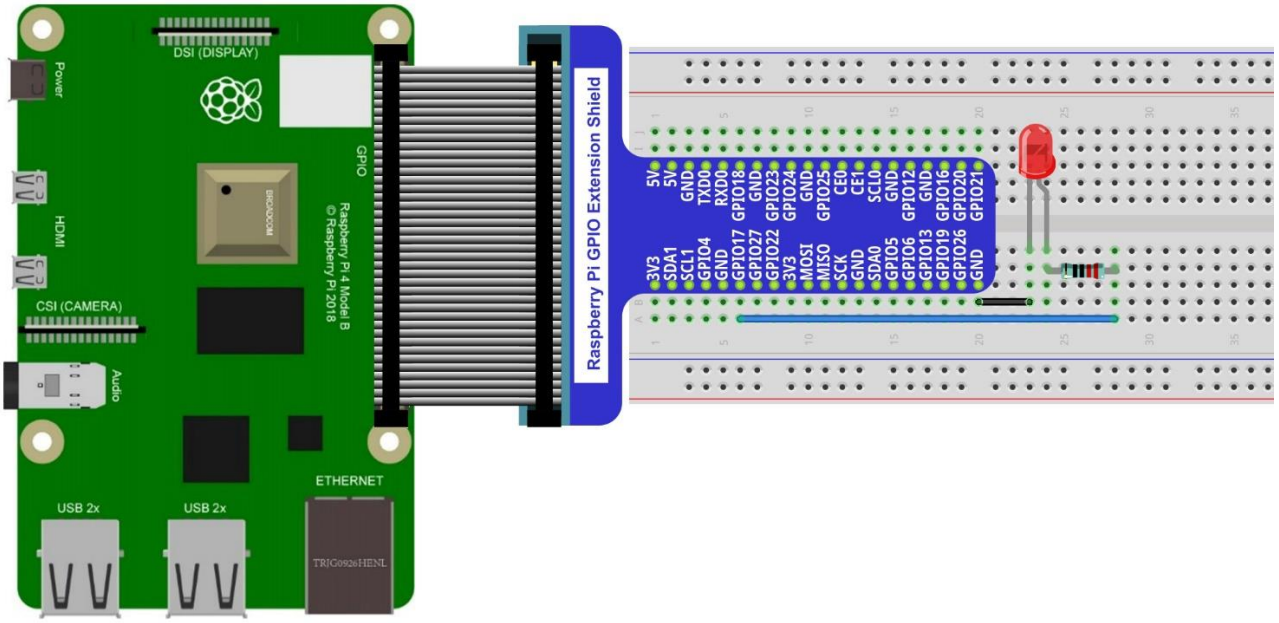
Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	LED x1 	Resistor 220Ω x1 
Jumper M/M x2 		

Circuit

Schematic diagram



Hardware connection



Sketch

Sketch 3.1.1 BreathingLED

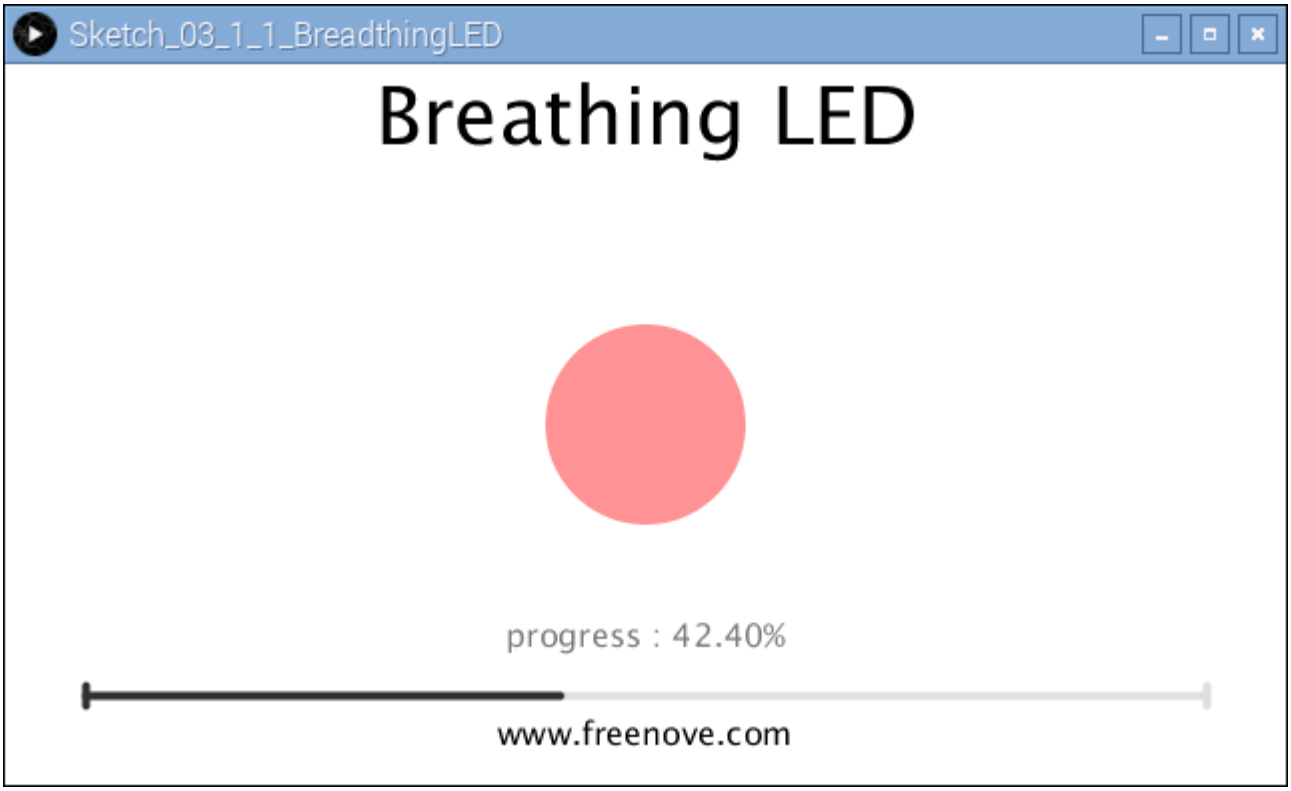
First observe the running result of the sketch, and then analyze the code.

1. Use Processing to open the file Sketch_03_1_1_BreathingLED.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_03_1_1_BreathingLED/Sketch_03_1_1_BreathingLED.pde
```

2. Click on "RUN" to run the code.

After the program is executed, the LED in circuit will be brightened gradually, and the color of LED pattern in Display Window will be deepen gradually at the same time. The progress bar under the patten shows the percentage of completion, and clicking on the inside of window with the mouse can change the progress.



The following is program code:

```
1  import processing.io.*;
2
3  int ledPin = 17;    //led Pin
4  int borderSize = 40;  //
5  float t = 0.0;      //progress percent
6  float tStep = 0.004;    // speed
7  SOFTPWM p = new SOFTPWM(ledPin, 10, 100);    //Create a PWM pin,initialize the duty cycle
8  and period
9  void setup() {
10   size(640, 360);  //display window size
```

```
11    strokeWeight(4);  //stroke Weight
12  }
13
14  void draw() {
15    // Show static value when mouse is pressed, animate otherwise
16    if (mousePressed) {
17      int a = constrain(mouseX, borderSize, width - borderSize);
18      t = map(a, borderSize, width - borderSize, 0.0, 1.0);
19    } else {
20      t += tStep;
21      if (t > 1.0) t = 0.0;
22    }
23    p.softPwmWrite((int)(t*100)); //wirte the duty cycle according to t
24    background(255);  //A white background
25    titleAndSiteInfo();  //title and Site infomation
26
27    fill(255, 255-t*255, 255-t*255);  //cycle
28    ellipse(width/2, height/2, 100, 100);
29
30    pushMatrix();
31    translate(borderSize, height - 45);
32    int barLength = width - 2*borderSize;
33
34    barBgStyle();  //progressbar bg
35    line(0, 0, barLength, 0);
36    line(barLength, -5, barLength, 5);
37
38    barStyle();  //progressbar
39    line(0, -5, 0, 5);
40    line(0, 0, t*barLength, 0);
41
42    barLabelStyle();    //progressbar label
43    text("progress : "+nf(t*100,2,2),barLength/2,-25);
44    popMatrix();
45  }
46
47  void titleAndSiteInfo() {
48    fill(0);
49    textAlign(CENTER);    //set the text centered
50    textSize(40);         //set text size
51    text("Breathing Light", width / 2, 40);    //title
52    textSize(16);
53    text("www.freenove.com", width / 2, height - 20);    //site
54  }
```

```
55  void barBgStyle() {
56    stroke(220);
57    noFill();
58  }
59
60  void barStyle() {
61    stroke(50);
62    noFill();
63  }
64
65  void barLabelStyle() {
66    noStroke();
67    fill(120);
68  }
```

First, use SOFTPWM class to create a PWM pin, which is used to control the brightness of LED. Then define a variable "t" and variable "tStep" to control the PWM duty cycle and add-self rate.

```
float t = 0.0;       //progress percent
float tStep = 0.004;    // speed
SOFTPWM p = new SOFTPWM(ledPin, 10, 100);
```

In the function draw, if there is a click, the coordinate in X direction of mouse will be mapped into the duty cycle "t", otherwise, duty cycle "t" will be increased gradually. Then output PWM with the duty cycle.

```
  if (mousePressed) {
    int a = constrain(mouseX, borderSize, width - borderSize);
    t = map(a, borderSize, width - borderSize, 0.0, 1.0);
  } else {
    t += tStep;
    if (t > 1.0) t = 0.0;
  }
  p.softPwmWrite((int)(t*100)); //wirte the duty cycle according to t
```

The next code is designed to draw a circle filled colors with different depth according to the "t" value, which is used to simulate the of LED with different brightness.

```
  fill(255, 255-t*255, 255-t*255);  //cycle
  ellipse(width/2, height/2, 100, 100);
```

The last code is designed to draw the progress bar and the percentage of the progress.

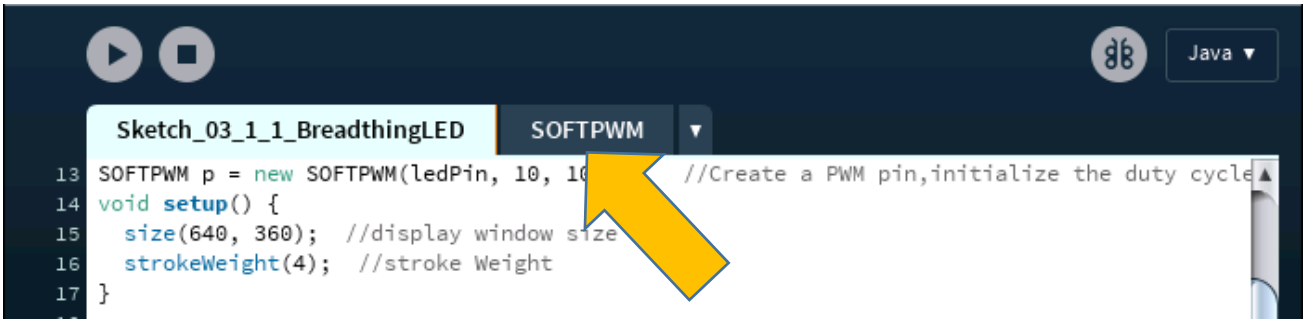
```
  barBgStyle();  //progressbar bg
  line(0, 0, barLength, 0);
  line(barLength, -5, barLength, 5);

  barStyle();  //progressbar
  line(0, -5, 0, 5);
```

```
  line(0, 0, t*barLength, 0);

  barLabelStyle();     //progressbar label
  text("progress : "+nf(t*100,2,2),barLength/2,-25);
```

In processing software, you will see a tag page "SOFTPWM" in addition to above code.



The file contains some information about the SOFTPWM class.

class SOFTPWM

`public SOFTPWM(int iPin, int dc, int pwmRange):`

Constructor, used to create a PWM pin, set the pwmRange and initial duty cycle. The minimum of pwmRange is 0.1ms. So pwmRange=100 means that the PWM cycle is 0.1ms*100=10ms.

`public void softPwmWrite(int value)`

Set PMW duty cycle.

`public void softPwmStop()`

Stop outputting PMW.

Chapter 4 RGBLED

In this chapter, we will learn how to use RGBLED.

Project 4.1 ColorfulLED

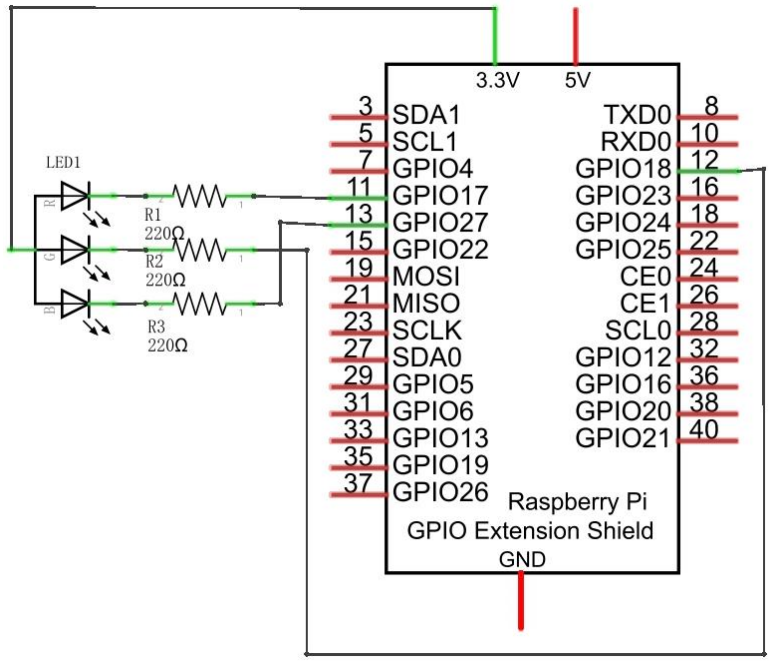
This project will make a ColorfulLED, namely, use Processing to control the color of RGBLED.

Component List

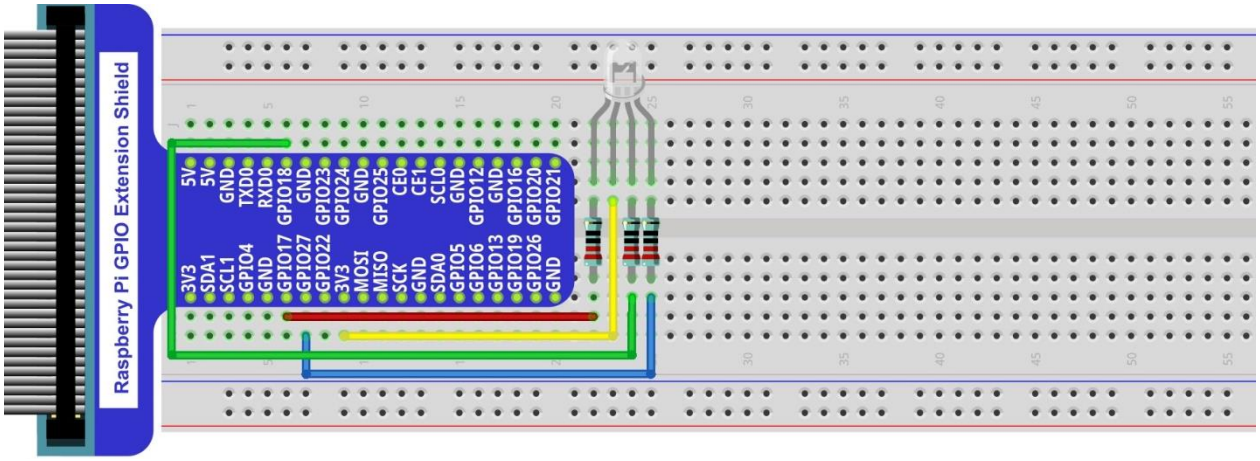
Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	RGBLED x1 	Resistor 220Ω x3 
Jumper M/M x4 		

Circuit

Schematic diagram



Hardware connection



Sketch

Sketch 4.1.1 ColorfulLED

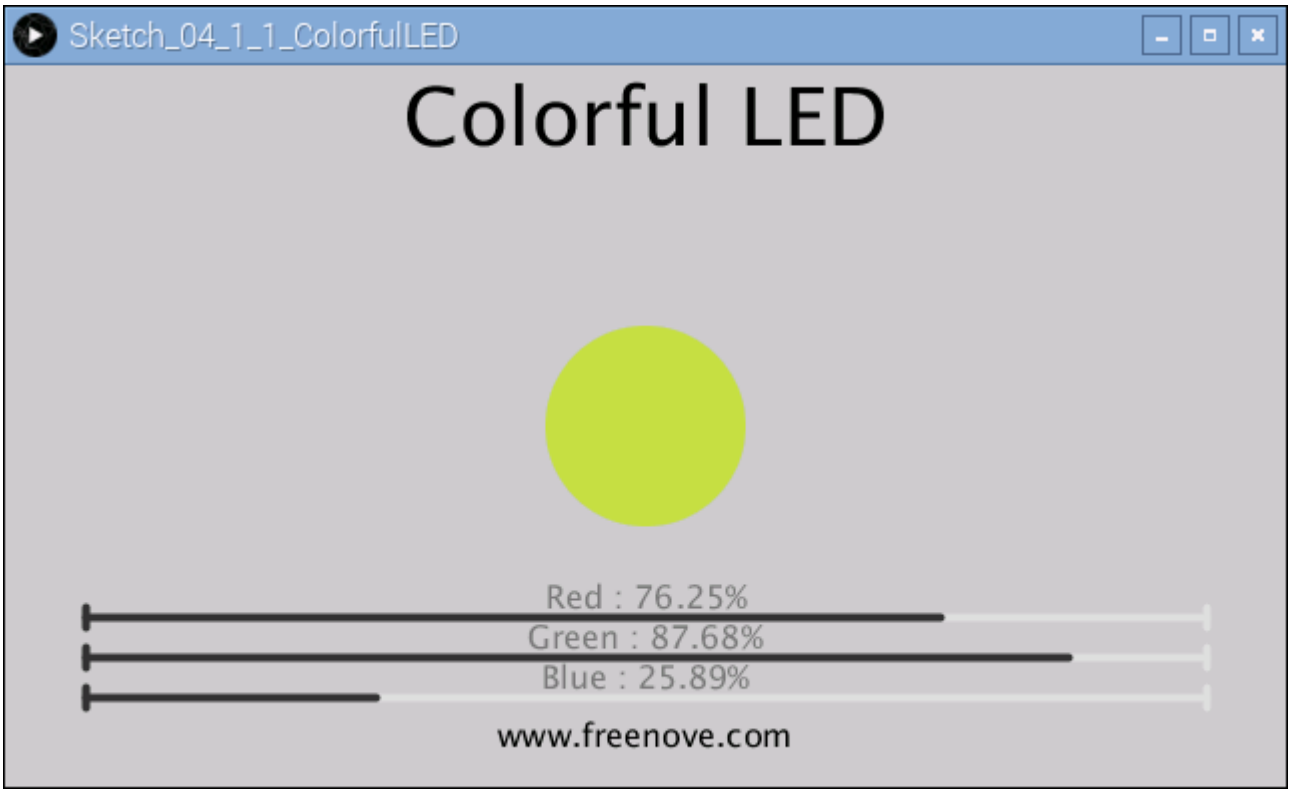
First observe the running result of the sketch, and then analyze the code.

1. Use Processing to open the file Sketch_03_1_1_BreathingLED.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_04_1_1_ColorfulLED/Sketch_04_1_1_ColorfulLED.pde
```

2. Click on "RUN" to run the code.

After the program is executed, RGBLED is under off state. And in Display Window, the pattern used to simulate LED is in black. Red, Green and Blue progress are in 0. By using mouse to click on and drag any progress bar, you can set the PWM duty cycle of color channels, and then RGBLED used in the circuit will show corresponding color. At the same time, the pattern in Display Window will show the same color.



This project contains a lot of code files, and the core code is contained in the file Sketch_04_1_1_ColorfulLED. The other files only contain some custom classes.

The following is program code:

```
import processing.io.*;

int bluePin = 17;    //blue Pin
int greenPin = 18;  //green Pin
int redPin = 27;    //red Pin
int borderSize = 40;  //picture border size
//Create a PWM pin,initialize the duty cycle and period
SOFTPWM pRed = new SOFTPWM(redPin, 100, 100);
SOFTPWM pGreen = new SOFTPWM(greenPin, 100, 100);
SOFTPWM pBlue = new SOFTPWM(bluePin, 100, 100);
//instantiate three ProgressBar Object
ProgressBar rBar, gBar, bBar;
boolean rMouse = false, gMouse = false, bMouse = false;
void setup() {
  size(640, 360);  //display window size
  strokeWeight(4);  //stroke Weight
  //define the ProgressBar length
  int barLength = width - 2*borderSize;
  //Create ProgressBar Object
  rBar = new ProgressBar(borderSize, height - 85, barLength);
  gBar = new ProgressBar(borderSize, height - 65, barLength);
  bBar = new ProgressBar(borderSize, height - 45, barLength);
  //Set ProgressBar's title
  rBar.setTitle("Red");gBar.setTitle("Green");bBar.setTitle("Blue");
}

void draw() {
  background(200);  //A white background
  titleAndSiteInfo();  //title and Site infomation

  fill(rBar.progress*255, gBar.progress*255, bBar.progress*255);  //cycle color
  ellipse(width/2, height/2, 100, 100);  //show cycle

  rBar.create();  //Show progressBar
  gBar.create();
  bBar.create();
}

void mousePressed() {
  if ( (mouseY< rBar.y+5) && (mouseY>rBar.y-5) ) {
    rMouse = true;
  } else if ( (mouseY< gBar.y+5) && (mouseY>gBar.y-5) ) {
    gMouse = true;
```

```
44    } else if ( (mouseY< bBar.y+5) && (mouseY>bBar.y-5) ) {
45      bMouse = true;
46    }
47  }
48  void mouseReleased() {
49    rMouse = false;
50    bMouse = false;
51    gMouse = false;
52  }
53  void mouseDragged() {
54    int a = constrain(mouseX, borderSize, width - borderSize);
55    float t = map(a, borderSize, width - borderSize, 0.0, 1.0);
56    if (rMouse) {
57      pRed.softPwmWrite((int)(100-t*100)); //wirte the duty cycle according to t
58      rBar.setProgress(t);
59    } else if (gMouse) {
60      pGreen.softPwmWrite((int)(100-t*100)); //wirte the duty cycle according to t
61      gBar.setProgress(t);
62    } else if (bMouse) {
63      pBlue.softPwmWrite((int)(100-t*100)); //wirte the duty cycle according to t
64      bBar.setProgress(t);
65    }
66  }
67
68  void titleAndSiteInfo() {
69    fill(0);
70    textAlign(CENTER);    //set the text centered
71    textSize(40);         //set text size
72    text("Colorful LED", width / 2, 40);    //title
73    textSize(16);
74    text("www.freenove.com", width / 2, height - 20);    //site
75  }
```

In the code, first create three PWM pins and three progress bars to control RGBLED.

```
SOFTPWM pRed = new SOFTPWM(redPin, 100, 100);
SOFTPWM pGreen = new SOFTPWM(greenPin, 100, 100);
SOFTPWM pBlue = new SOFTPWM(bluePin, 100, 100);
//instantiate three ProgressBar Object
ProgressBar rBar, gBar, bBar;
```

And then in function setup(), define position and length of progress bar ccording to the size of Display Window, and set the name of each progress bar.

```
void setup() {
  size(640, 360);  //display window size
```

```
  strokeWeight(4);  //stroke Weight
  //define the ProgressBar length
  int barLength = width - 2*borderSize;
  //Create ProgressBar Object
  rBar = new ProgressBar(borderSize, height - 85, barLength);
  gBar = new ProgressBar(borderSize, height - 65, barLength);
  bBar = new ProgressBar(borderSize, height - 45, barLength);
  //Set ProgressBar's title
  rBar.setTitle("Red");gBar.setTitle("Green");bBar.setTitle("Blue");
}
```

In function draw(), first set background, header and other basic information. Then draw a circle and set its color according to the duty cycle of three channel of RGB. Finally draw three progress bars.

```
void draw() {
  background(200);  //A white background
  titleAndSiteInfo();  //title and Site infomation

  fill(rBar.progress*255, gBar.progress*255, bBar.progress*255);  //cycle color
  ellipse(width/2, height/2, 100, 100);  //show cycle

  rBar.create();  //Show progressBar
  gBar.create();
  bBar.create();
}
```

System function mousePressed(), mouseReleased() and mouseDragged() are used to determine whether the mouse drag the progress bar and set the schedule. If the mouse button is pressed in a progress bar, then the mousePressed () the progress of a flag *Mouse is set to true, mouseDragged (mouseX), in the mapping progress value set at the same time, the progress of the corresponding schedule and PWM. When the mouse is released, empty all the flags in the mouseReleased ().

```
void mousePressed() {
  if ( (mouseY< rBar.y+5) && (mouseY>rBar.y-5) ) {
    rMouse = true;
  } else if ( (mouseY< gBar.y+5) && (mouseY>gBar.y-5) ) {
    gMouse = true;
  } else if ( (mouseY< bBar.y+5) && (mouseY>bBar.y-5) ) {
    bMouse = true;
  }
}
void mouseReleased() {
  rMouse = false;
  bMouse = false;
  gMouse = false;
}
```

```
void mouseDragged() {
  int a = constrain(mouseX, borderSize, width - borderSize);
  float t = map(a, borderSize, width - borderSize, 0.0, 1.0);
  if (rMouse) {
    pRed.softPwmWrite((int)(100-t*100)); //wirte the duty cycle according to t
    rBar.setProgress(t);
  } else if (gMouse) {
    pGreen.softPwmWrite((int)(100-t*100)); //wirte the duty cycle according to t
    gBar.setProgress(t);
  } else if (bMouse) {
    pBlue.softPwmWrite((int)(100-t*100)); //wirte the duty cycle according to t
    bBar.setProgress(t);
  }
}
```

About class ProgressBar:

class ProgressBar

This is a custom class that is used to create a progress bar.

```
public ProgressBar(int ix, int iy, int barlen)
```

Constructor used create ProgressBar, the parameters for coordinates X , Y and length of ProgressBar.

```
public void setTitle(String str)
```

Used to set the name of progress bar, which will be displayed in middle of the progress bar.

```
public void setProgress(float pgress)
```

Used to set the progress of progress bar. The parameter: 0<pgress<1.0.

```
public void create() & public void create(float pgress)
```

Used to draw progress bar.

Chapter 5 Buzzer

In this chapter we will learn how to use buzzer.

Project 5.1 ActiveBuzzer

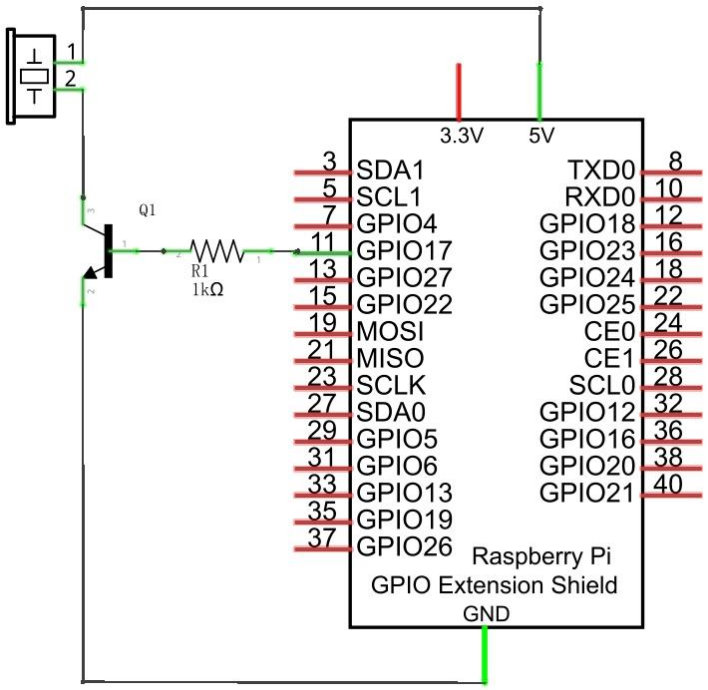
This project will use the mouse to control an active buzzer.

Component List

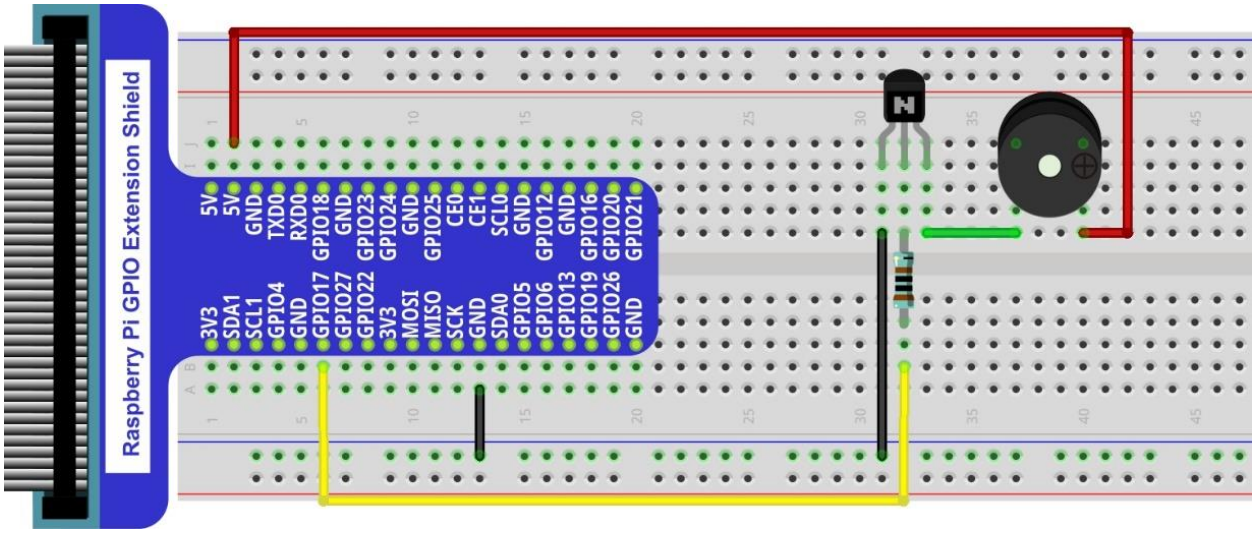
<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x7
  </td></tr>
<tr><td>NPN transistorx1
  </td><td>Active buzzer x1
  </td><td>Push button x1
  </td><td>Resistor 1kΩ x1
  </td><td>Resistor 10kΩ x2
  </td></tr>
</table>

Circuit

Schematic diagram



Hardware connection



Note: in this circuit, the power supply for buzzer is 5V, and pull-up resistor of the button connected to the power 3.3V. The buzzer can work when connected to power 3.3V, but it will reduce the loudness.

Sketch

Sketch 5.1.1 ActiveBuzzer

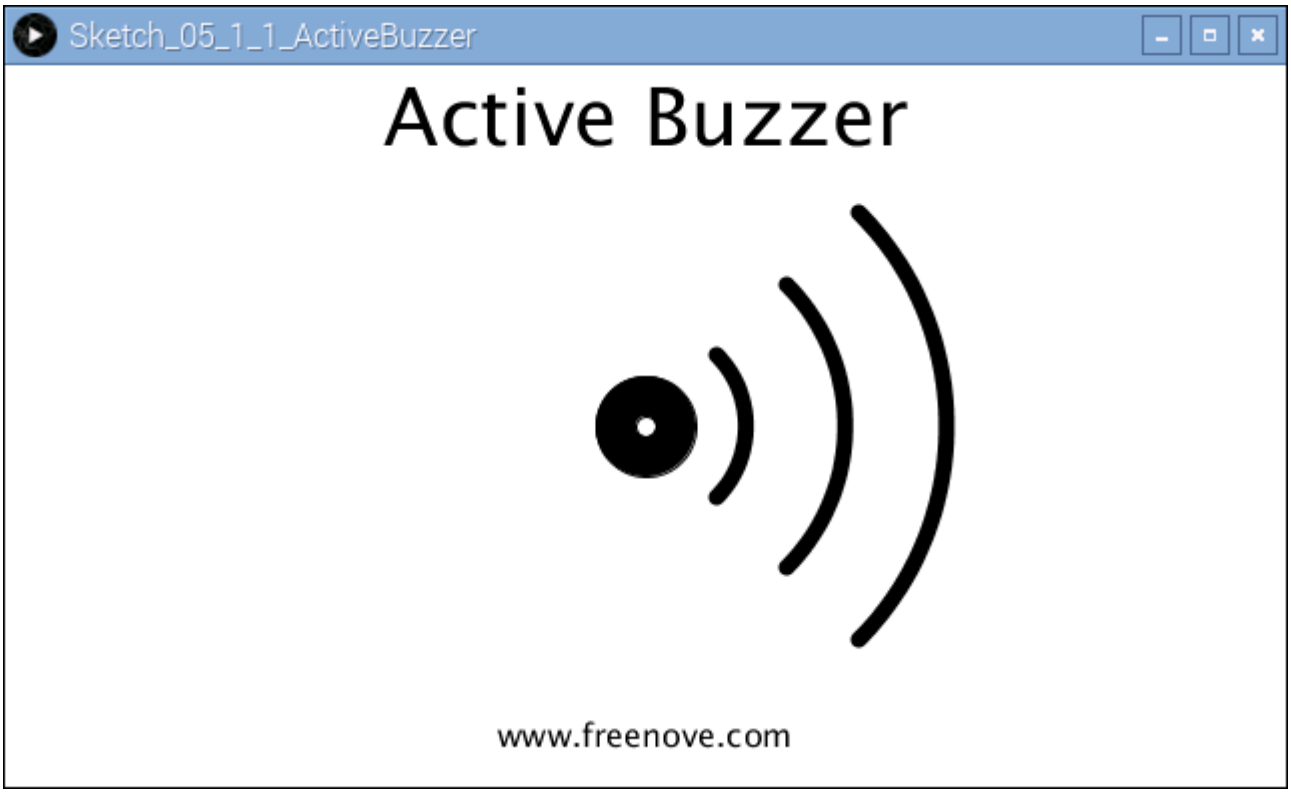
First observe the running result of the sketch, and then analyze the code.

1. Use Processing to open the file Sketch_05_1_1_ActiveBuzzer.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_05_1_1_ActiveBuzzer/Sketch_05_1_1_ActiveBuzzer.pde
```

2. Click on "RUN" to run the code.

After the program is executed, use the mouse to click on any position of the Display Window, then Active Buzzer begins to sound and arc graphics(Schematic of sounding) will appear next to the buzzer pattern in Display Window. Click the mouse again, then Active Buzzer stops sounding and arc graphics disappeare.



The following is program code:

```
import processing.io.*;

int buzzerPin = 17;
boolean buzzerState = false;
void setup() {
  size(640, 360);
  GPIO.pinMode(buzzerPin, GPIO.OUTPUT);
}
```

```
void draw() {
  background(255);
  titleAndSiteInfo();    //title and site infomation
  drawBuzzer();       //buzzer img
  if (buzzerState) {
    GPIO.digitalWrite(buzzerPin, GPIO.HIGH);
    drawArc();      //Sounds waves img
  } else {
    GPIO.digitalWrite(buzzerPin, GPIO.LOW);
  }
}

void mouseClicked() { //if the mouse Clicked
  buzzerState = !buzzerState;  //Change the buzzer State
}
void drawBuzzer() {
  strokeWeight(1);
  fill(0);
  ellipse(width/2, height/2, 50, 50);
  fill(255);
  ellipse(width/2, height/2, 10, 10);
}
void drawArc() {
  noFill();
  strokeWeight(8);
  for (int i=0; i<3; i++) {
    arc(width/2, height/2, 100*(1+i), 100*(1+i), -PI/4, PI/4, OPEN);
  }
}
void titleAndSiteInfo() {
  fill(0);
  textAlign(CENTER);    //set the text centered
  textSize(40);        //set text size
  text("Active Buzzer", width / 2, 40);    //title
  textSize(16);
  text("www.freenove.com", width / 2, height - 20);    //site
}
```

Code in this project is based on similar logic with the previous "MouseLED". And the difference is that this project needs to draw the buzzer pattern and arc graphics after the buzzer sounding.

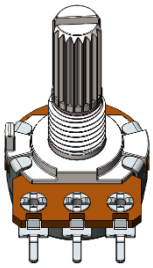
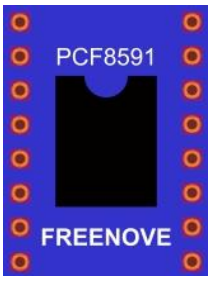
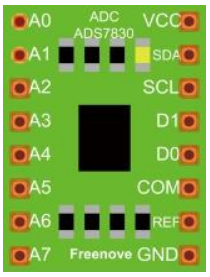
Chapter 6 ADC Module

In this chapter we will learn how to use ADC module.

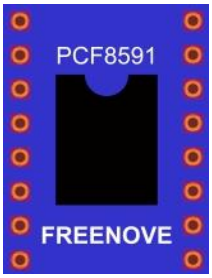
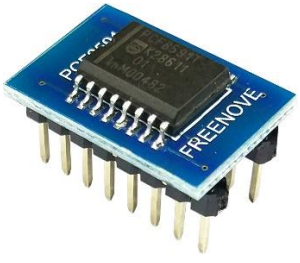
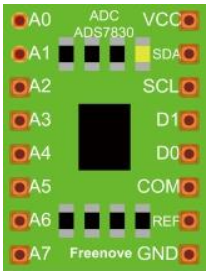
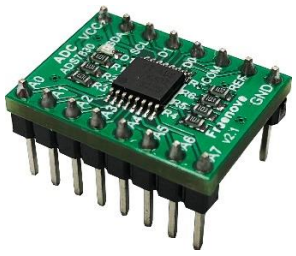
Project 6.1 Voltmeter

This project uses ADC module to read potentiometer voltage value and display the value on Display Window.

Component List

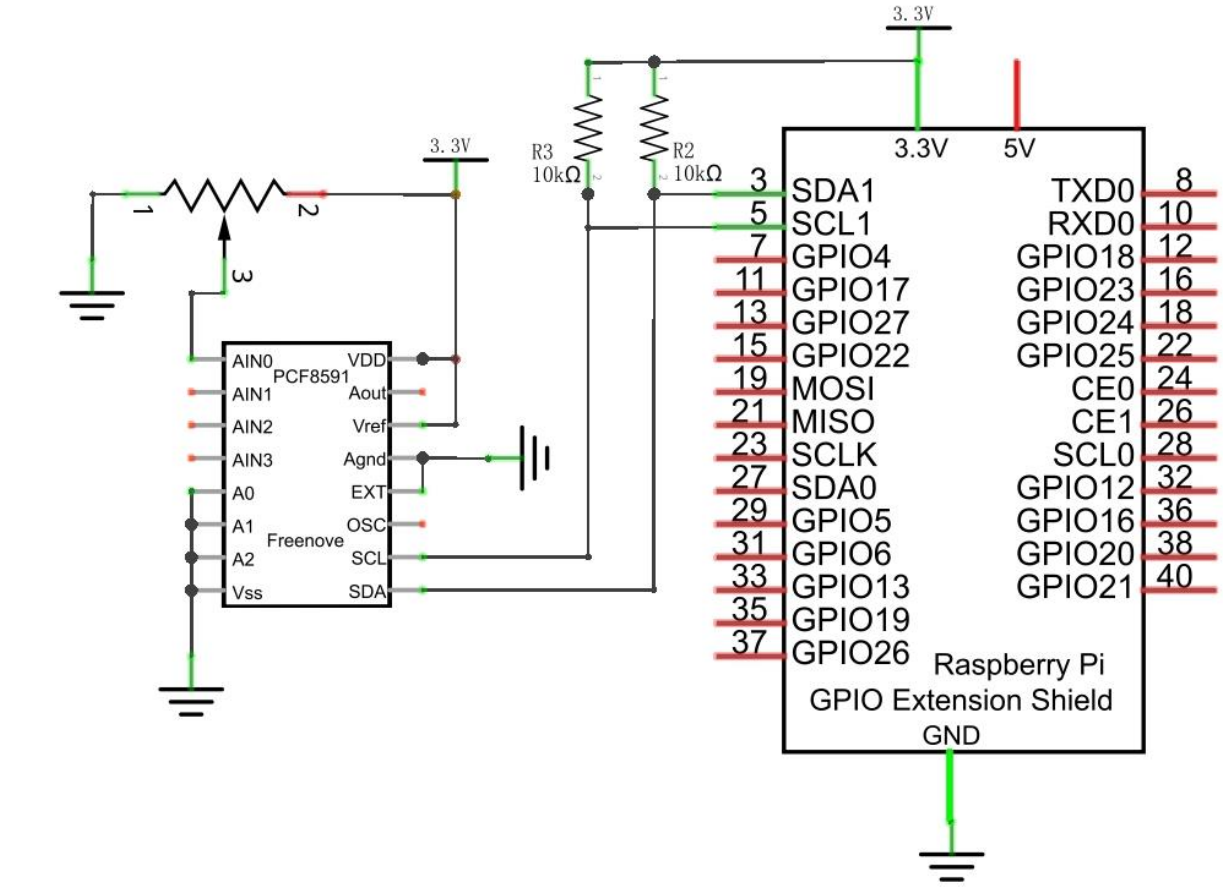
Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	Jumper M/M x16 	
Rotary potentiometer x1 	ADC module x1  or 	Resistor 10kΩ x2 

This product contains **only one ADC module**, which has two different types, PCF8591 and ADS7830. The functions involved in this tutorial are the same. Please build corresponding circuits according to different ADC module in your kit.

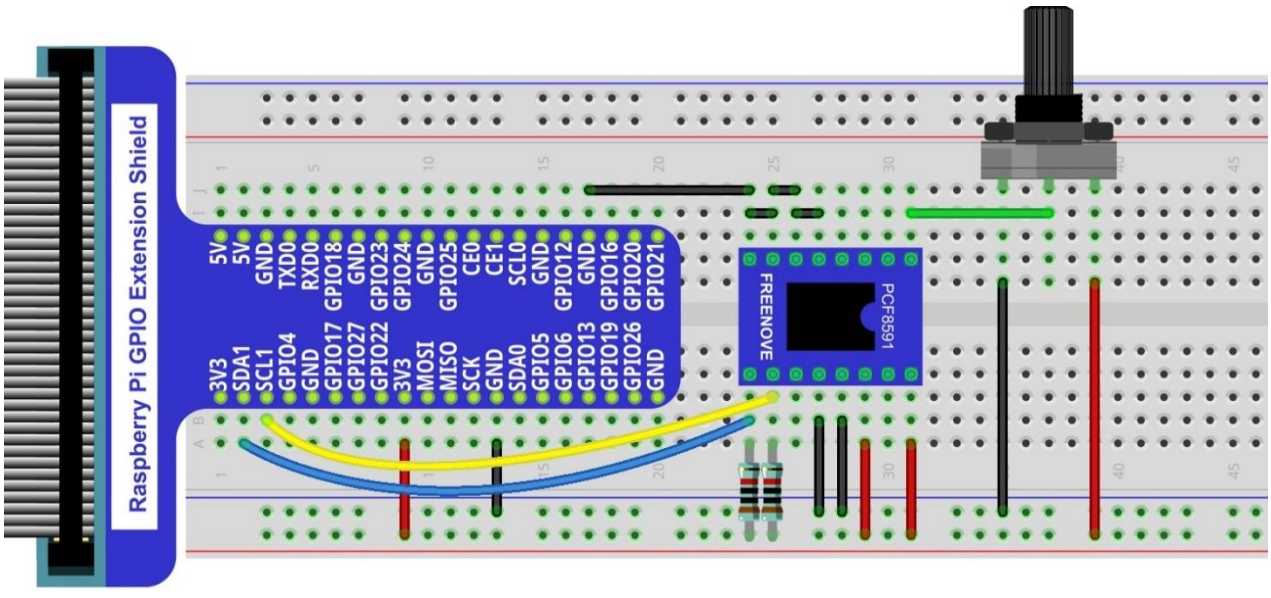
ADC module : PCF8591		ADC module : ADS7830	
			

Circuit with PCF8591

Schematic diagram

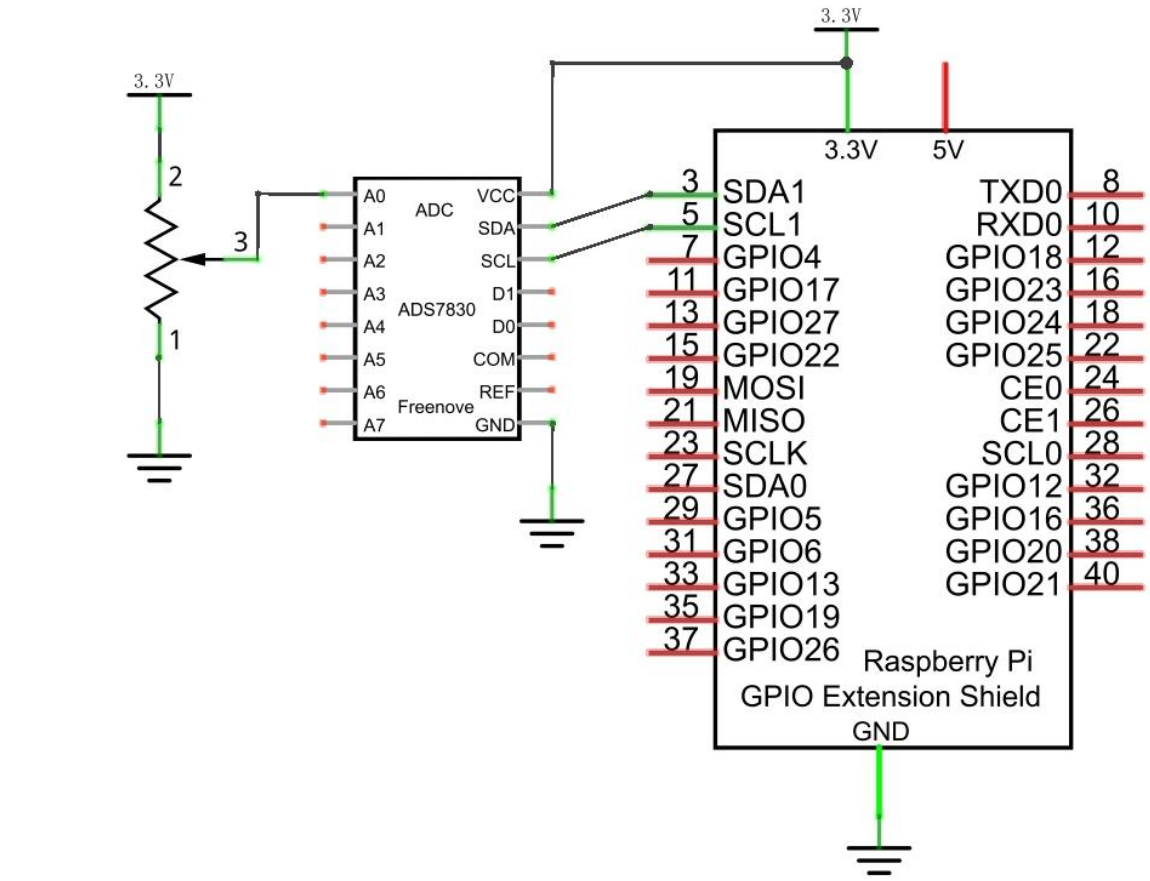


Hardware connection

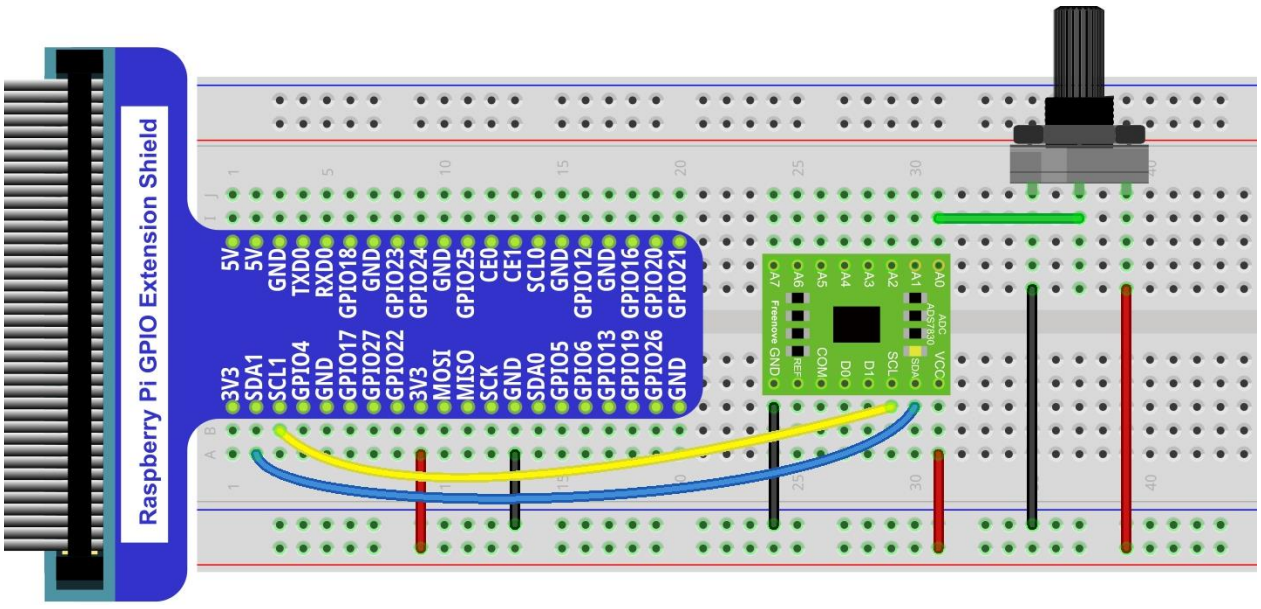


Circuit with ADS7830

Schematic diagram



Hardware connection



Sketch

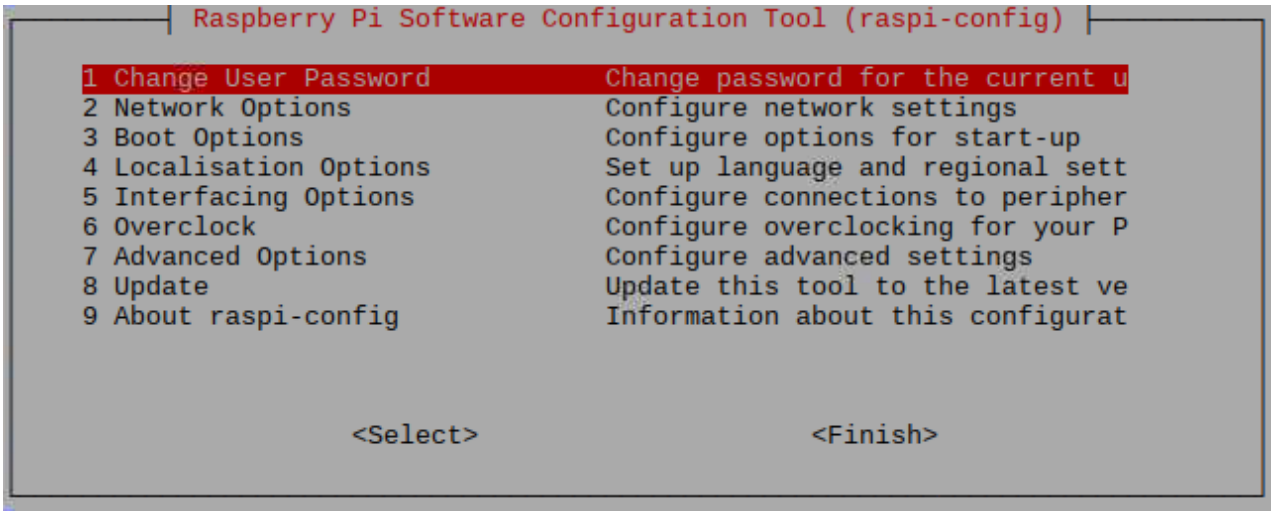
Configure I2C (Necessary)

Enable I2C

There are some I2C chips in this kit like ADC module. The I2C interface of raspberry pi is closed in default. You need to open it manually as below:
Type command in the terminal:

```
sudo raspi-config
```

Then open the following dialog box:



Choose “5 Interfacing Options”→“P5 I2C”→“Yes”→“Finish” in order and restart your RPi later. Then the I2C module is 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:

Install I2C-Tools

Type the command to install I2C-Tools.

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

I2C device address detection:

```
i2cdetect -y 1
```

When you are using PCF8591, the result is below:

```
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 is below:

```
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).

Sketch 6.1.1 ADC

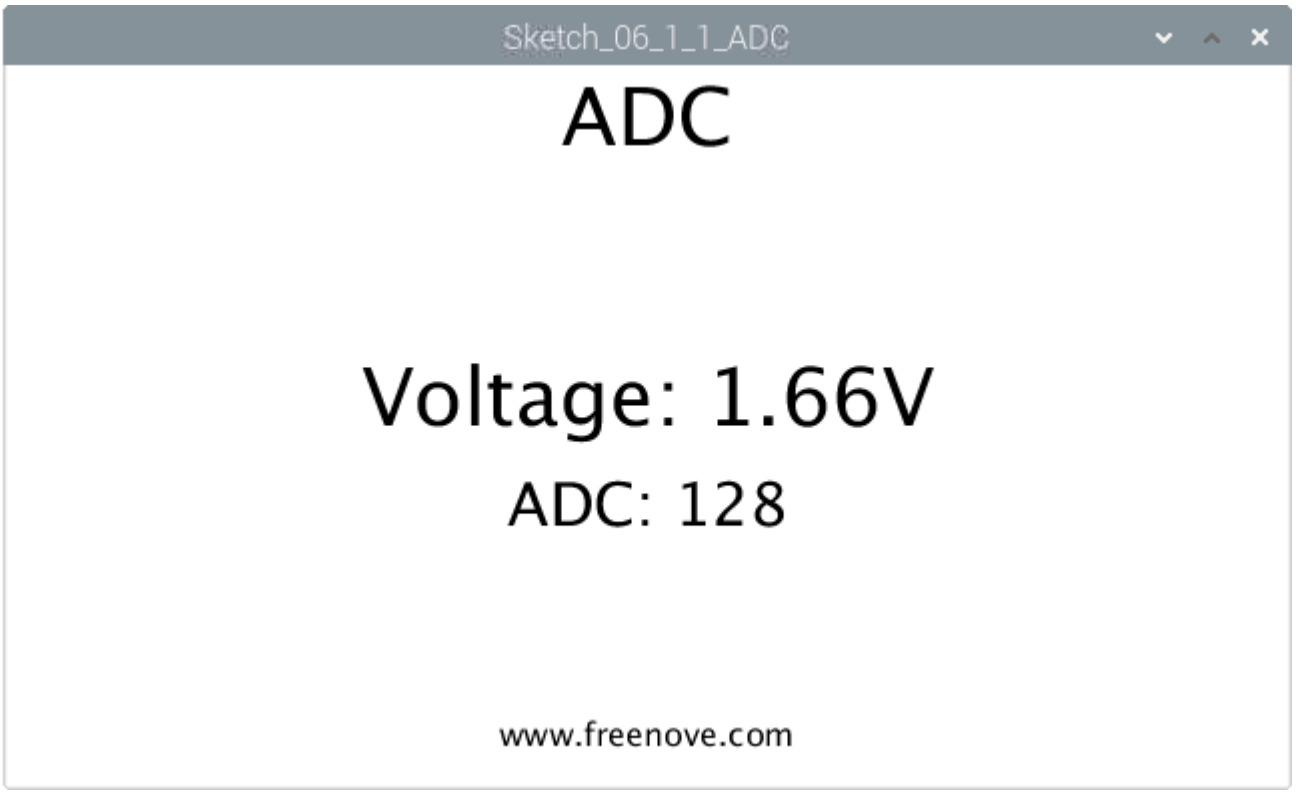
First observe the running result of the sketch, and then analyze the code.

1. Use Processing to open the file Sketch_06_1_1_ADC.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_06_1_1_ADC/Sketch_06_1_1_ADC.pde
```

2. Click on "RUN" to run the code.

After the program is executed, Display Window shows the voltage value of the potentiometer, and the ADC value. Rotate the potentiometer to change the voltage output by potentiometer.



This project contains a lot of code files, and the core code is contained in the file Sketch_06_1_1_ADC. The other files only contain some custom classes.

The following is program code:

```
import processing.io.*;
//Create a object of class ADCDevice
ADCDevice adc = new ADCDevice();
void setup() {
  size(640, 360);
  if (adc.detectI2C(0x48)) {
    adc = new PCF8591(0x48);
  } else if (adc.detectI2C(0x4b)) {
    adc = new ADS7830(0x4b);
  } else {
    println("Not found ADC Module!");
    System.exit(-1);
  }
}
void draw() {
  int adcValue = adc.analogRead(0);    //Read the ADC value of channel 0
  float volt = adcValue*3.3/255.0;    //calculate the voltage
  background(255);
  titleAndSiteInfo();

  fill(0);
  textAlign(CENTER);    //set the text centered
  textSize(30);
  text("ADC: "+nf(adcValue, 3, 0), width / 2, height/2+50);
  textSize(40);         //set text size
  text("Voltage: "+nf(volt, 0, 2)+"V", width / 2, height/2);    //
}
void titleAndSiteInfo() {
  fill(0);
  textAlign(CENTER);    //set the text centered
  textSize(40);         //set text size
  text("ADC", width / 2, 40);    //title
  textSize(16);
  text("www.freenove.com", width / 2, height - 20);    //site
}
```

The project code mainly use PCF8591 class member function analogRead() to read ADC.

```
int adcValue = adc.analogRead(0);     //Read the ADC value of channel 0
float volt = adcValue*3.3/255.0;     //calculate the voltage
```

About class ADCDevice, PCF8591, ADS7830:

class ADCDevice

This is a base class, and all ADC module classes are subclasses of it. It provides two basic member functions.

`public int analogRead(int chn)`

This is a unified function name. Different chips have different implement methods. Therefore, specific method is implemented in subclasses.

`public boolean detectI2C(int addr)`

Used to detect I2C device with a given address. If it exists, it returns true, otherwise it returns false.

class PCF8591 extends ADCDevice

This is a custom class that is used to operate the ADC and DAC of PCF8591.

`public PCF8591(int addr)`

Constructor, used to create a PCF8591 class object, parameters for the I2C PCF8591 device address.

`public int analogRead(int chn)`

Used to read ADC value of one channel of PCF8591, the parameter CHN indicates the channel number: 0,1,2,3.

`public byte[] analogRead()`

Read ADC values of all channels of PCF8591.

`public void analogWrite(int data)`

Write a DAC value to PCF8591.

class ADS7830 extends ADCDevice

This is a custom class that is used to operate the ADC of ADS7830.

`public ADS7830(int addr)`

Constructor, used to create a ADS7830 class object, parameters for the I2C ADS7830 device address.

`public int analogRead(int chn)`

Used to read ADC value of one channel of ADS7830, the parameter CHN indicates the channel number: 0,1,2,3,4,5,6,7.

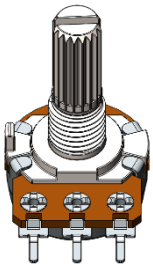
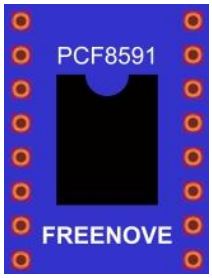
Chapter 7 ADC & LED

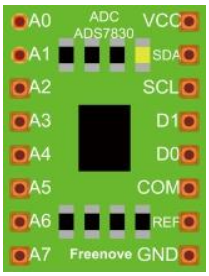
In this chapter, we will combine ADC and PWM to control the brightness of LED.

Project 7.1 SoftLight

In this project, we will make a softlight, namely, using a potentiometer to control the brightness of LED.

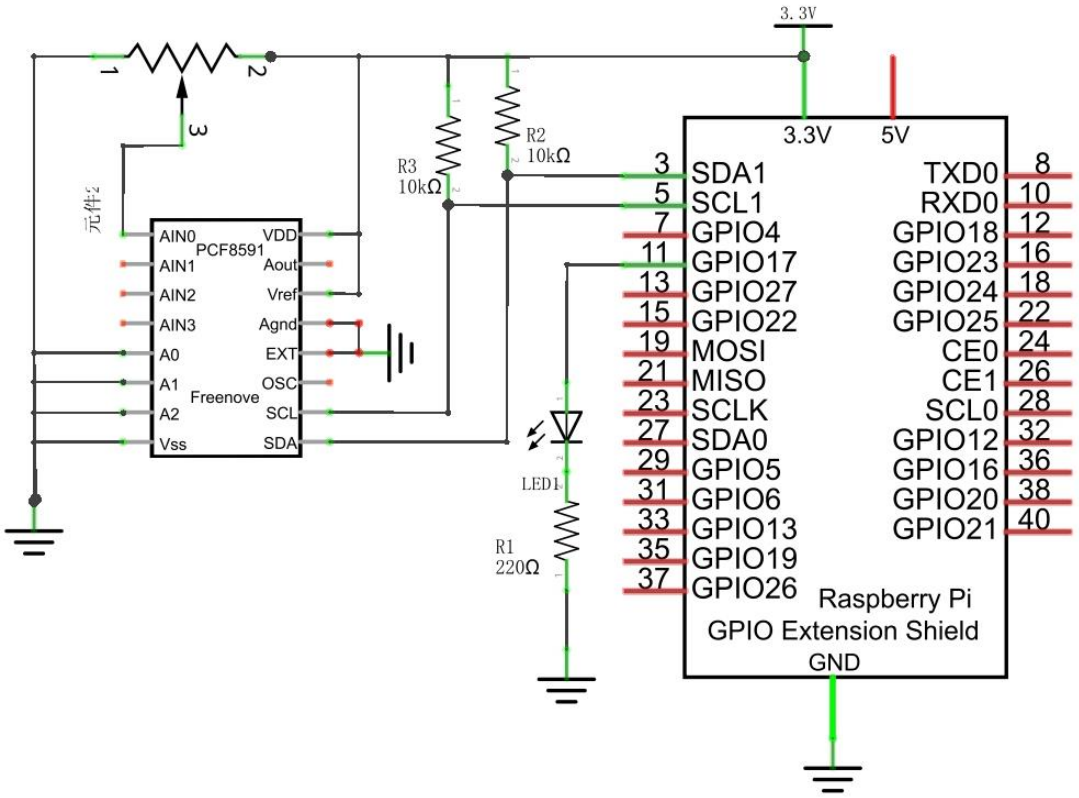
Component List

<table>
<tr><td colspan="3">Raspberry Pi x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M x17
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td colspan="2">ADC module x1
 

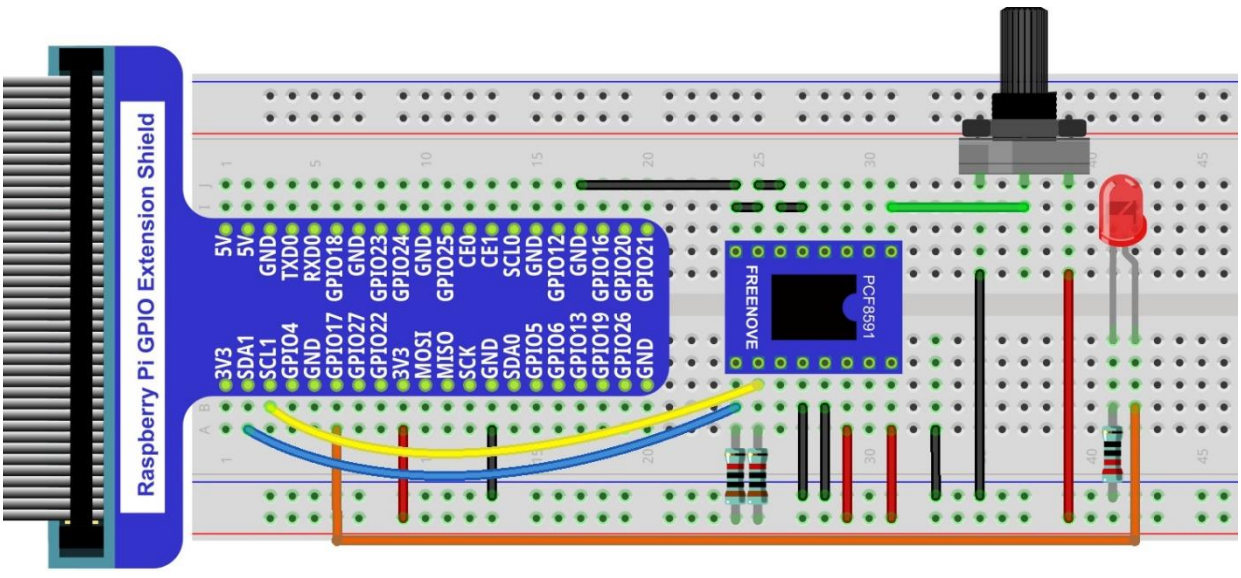
or

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

Circuit with PCF8591

Schematic diagram

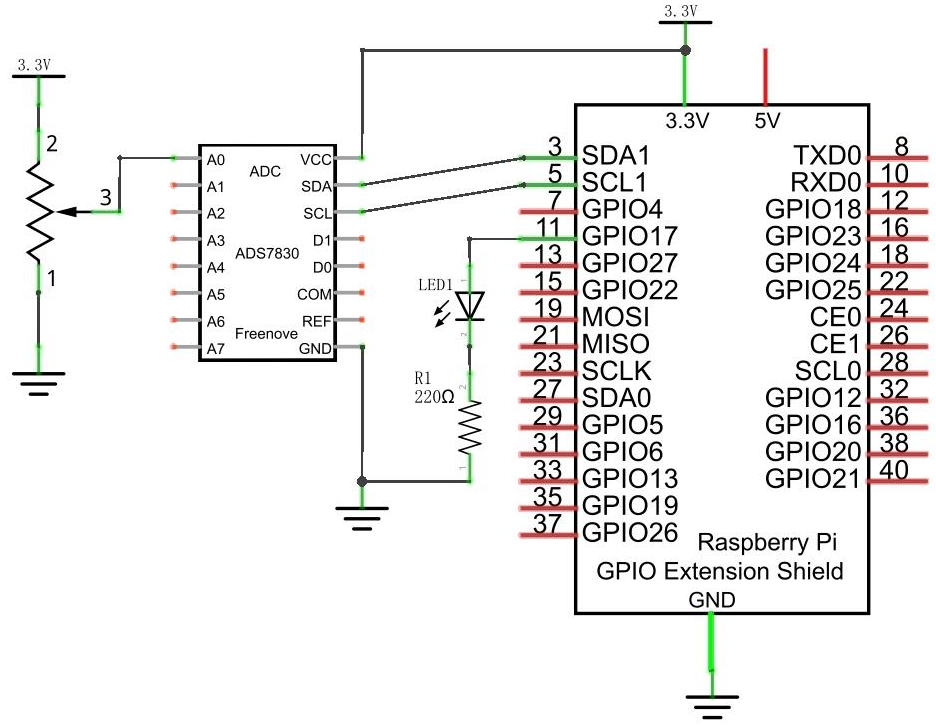


Hardware connection

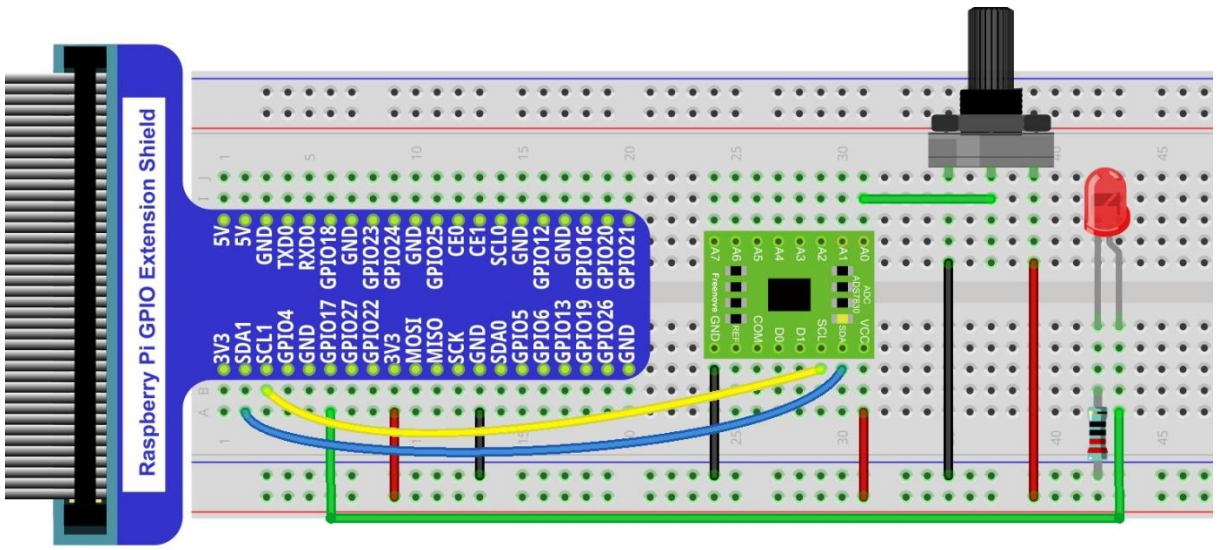


Circuit with ADS7830

Schematic diagram



Hardware connection



Sketch

If you did not configure I2C, please refer to Chapter 6. If you did, please move on.

Sketch 7.1.1 SoftLight

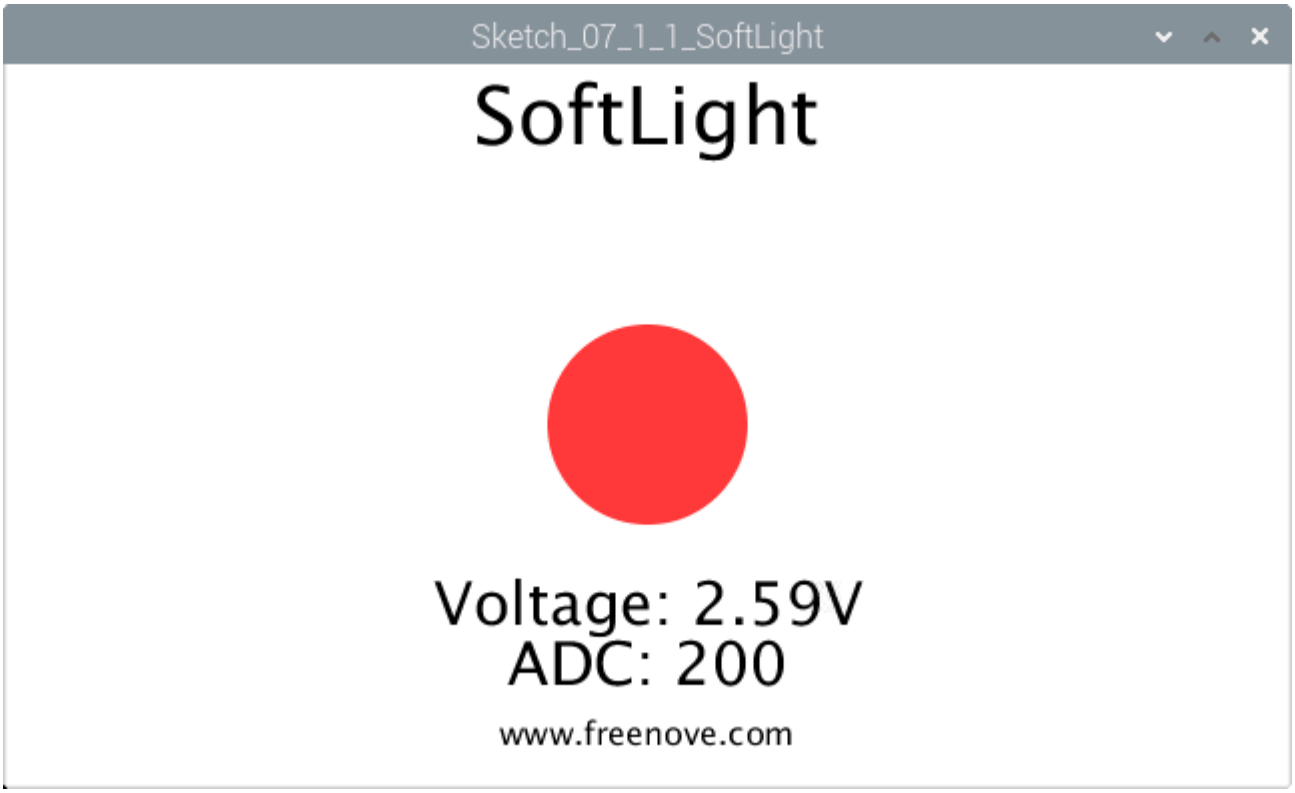
First observe the running result of the sketch, and then analyze the code.

1. Use Processing to open the file Sketch_07_1_1_SoftLight.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_07_1_1_SoftLight/Sketch_07_1_1_SoftLight.pde
```

2. Click on "RUN" to run the code.

After the program is executed, the Display Window will show the voltage value of potentiometer, the ADC value and a LED pattern. Rotate potentiometer to change the voltage value and the brightness of the LED.



This project contains a lot of code files, and the core code is contained in the file Sketch_07_1_1_SoftLight. The other files only contain some custom classes.

The following is program code:

```
import processing.io.*;

int ledPin = 17;    //led
//Create a object of class ADCDevice
ADCDevice adc = new ADCDevice();
SOFTPWM p = new SOFTPWM(ledPin, 0, 100);
void setup() {
  size(640, 360);
  if (adc.detectI2C(0x48)) {
    adc = new PCF8591(0x48);
  } else if (adc.detectI2C(0x4b)) {
    adc = new ADS7830(0x4b);
  } else {
    println("Not found ADC Module!");
    System.exit(-1);
  }
}
void draw() {
  int adcValue = adc.analogRead(0);    //Read the ADC value of channel 0
  float volt = adcValue*3.3/255.0;    //calculate the voltage
  float dt = adcValue/255.0;
  p.softPwmWrite((int)(dt*100));  //output the pwm
  background(255);
  titleAndSiteInfo();

  fill(255, 255-dt*255, 255-dt*255);  //cycle
  noStroke();  //no border
  ellipse(width/2, height/2, 100, 100);

  fill(0);
```

```
31    textAlign(CENTER);    //set the text centered
32    textSize(30);
33    text("ADC: "+nf(adcValue, 3, 0), width / 2, height/2+130);
34    text("Voltage: "+nf(volt, 0, 2)+"V", width / 2, height/2+100);    //
35  }
36  void titleAndSiteInfo() {
37    fill(0);
38    textAlign(CENTER);    //set the text centered
39    textSize(40);         //set text size
40    text("SoftLight", width / 2, 40);    //title
41    textSize(16);
42    text("www.freenove.com", width / 2, height - 20);    //site
43  }
```

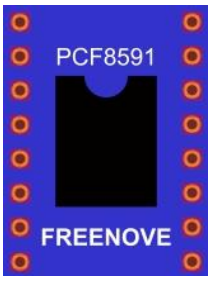
In this project code, get the ADC value of the potentiometer, then map it into the PWM duty cycle of LED to control its brightness. In Display Window, the color filled in LED pattern changes to simulate the brightness change of LED.

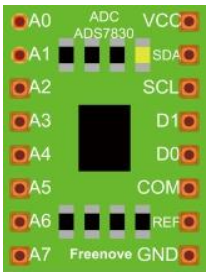
```
    int adcValue = adc.analogRead(0);    //Read the ADC value of channel 0
    float volt = adcValue*3.3/255.0;    //calculate the voltage
    float dt = adcValue/255.0;
    p.softPwmWrite((int)(dt*100));  //output the pwm
```

Project 7.2 NightLamp

In this project, we will use Photoresistor, used to intense ambient light, to make a NightLamp. When the ambient light get dark, LED brightness wil be enhanced automatically. Conversely, the LED brightness will be weakened automatically.

Component List

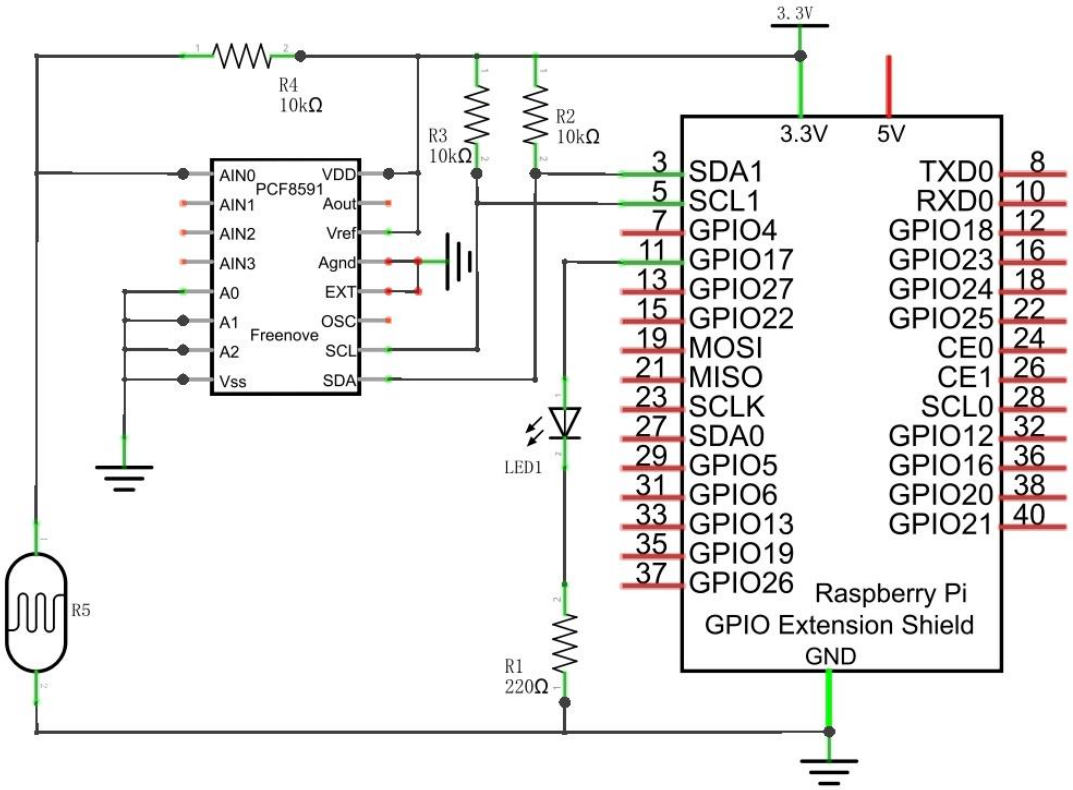
<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td colspan="3">Jumper M/M
  </td></tr>
<tr><td>Photoresistor x1
  </td><td colspan="2">ADC module x1
 

or

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

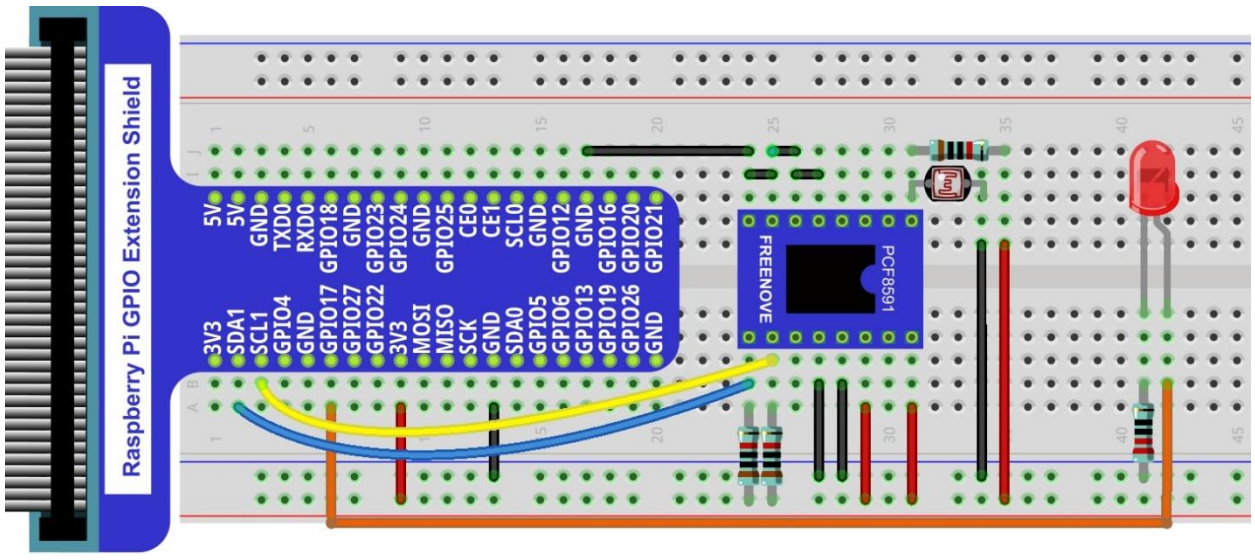
Circuit with PCF8591

The circuit of this experiment is similar to the one in last chapter. The only difference is that the input signal of the AIN0 pin of ADC is changed from a potentiometer to combination of a photoresistor and a resistor.

Schematic diagram



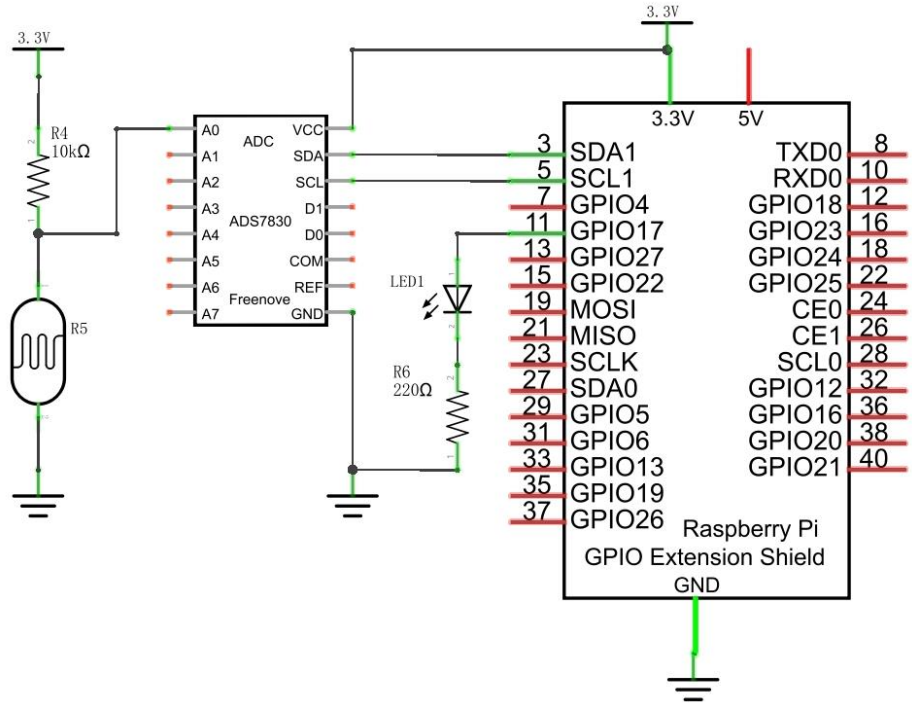
Hardware connection



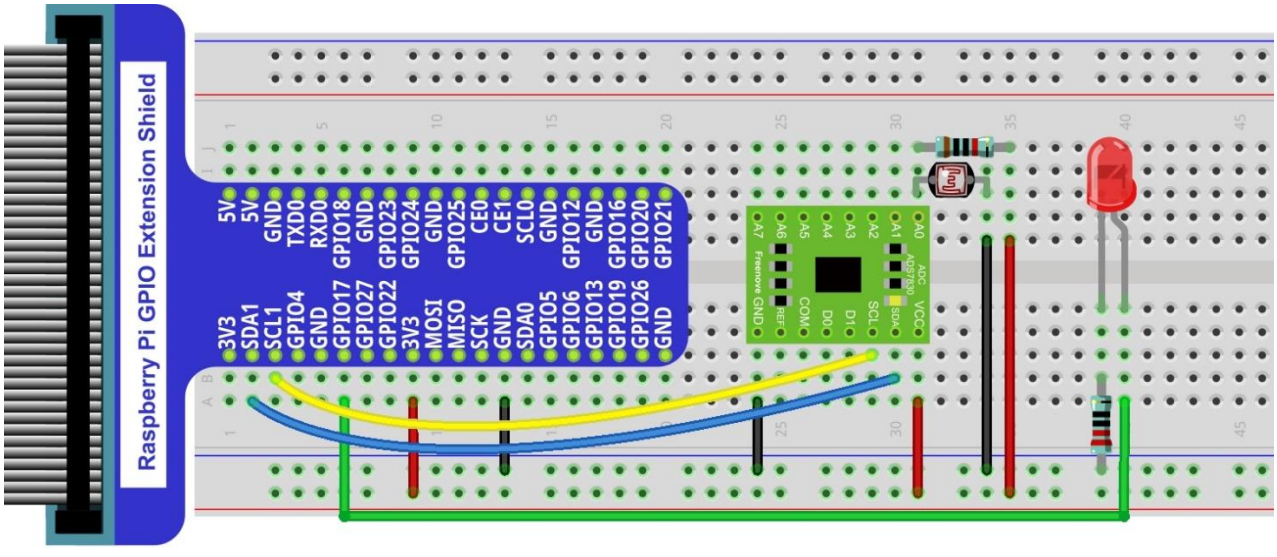
Circuit with ADS7830

The circuit of this experiment is similar to the one in last chapter. The only difference is that the input signal of the AIN0 pin of ADC is changed from a potentiometer to combination of a photoresistor and a resistor.

Schematic diagram



Hardware connection



Sketch

The project code is the same as the last section "SoftLight" in addition to the title.

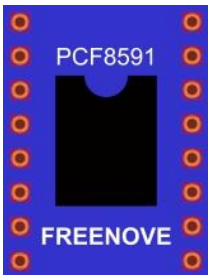
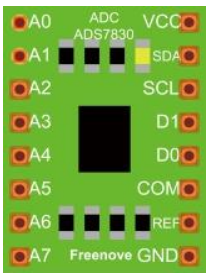
Chapter 8 Thermistor

In this chapter, we will learn how to use thermistor.

Project 8.1 Thermometer

In this project, we will use a thermistor to make a thermometer.

Component List

Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	Jumper M/M 	
Thermistor x1 	ADC module x1  or 	Resistor 10kΩ x3 

Component knowledge

First Review the knowledge of thermistor. The relationship between resistance value and temperature of 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;

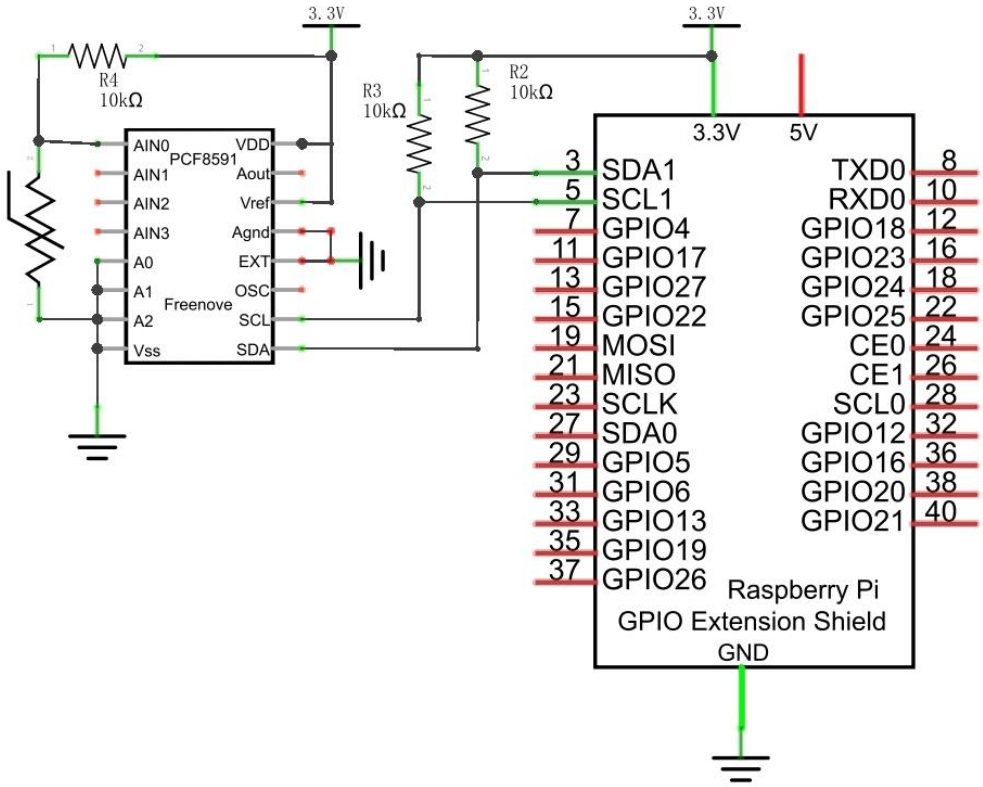
T1, T2 is Kelvin temperature (absolute temperature). Kelvin temperature=273.15+celsius temperature.

Parameters of the thermistor we use is: B=3950, R=10k, T1=25.

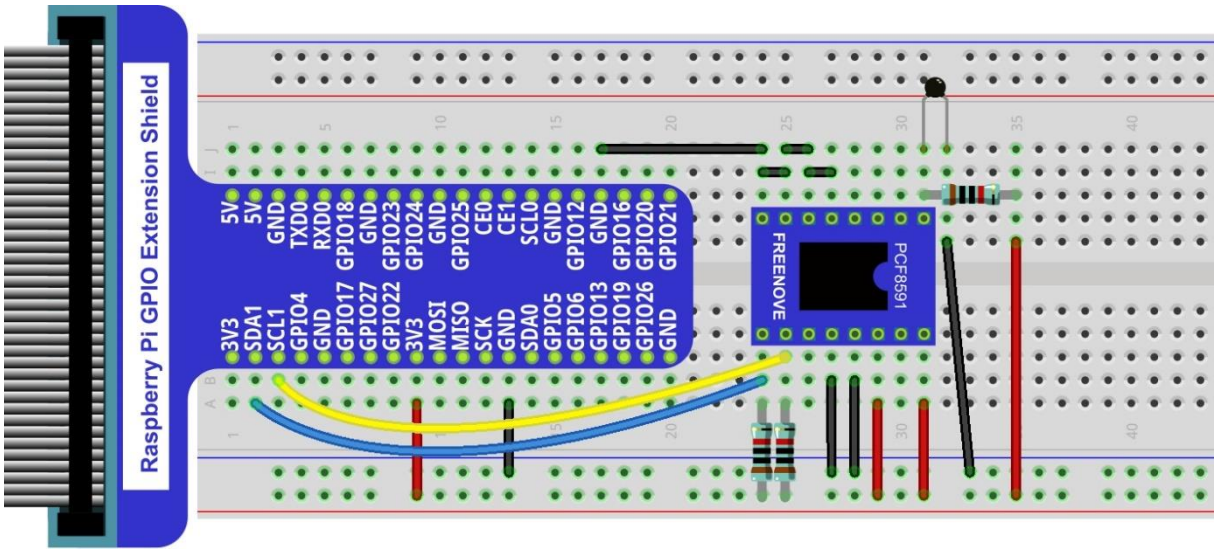
Circuit with PCF8591

The circuit of this experiment 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



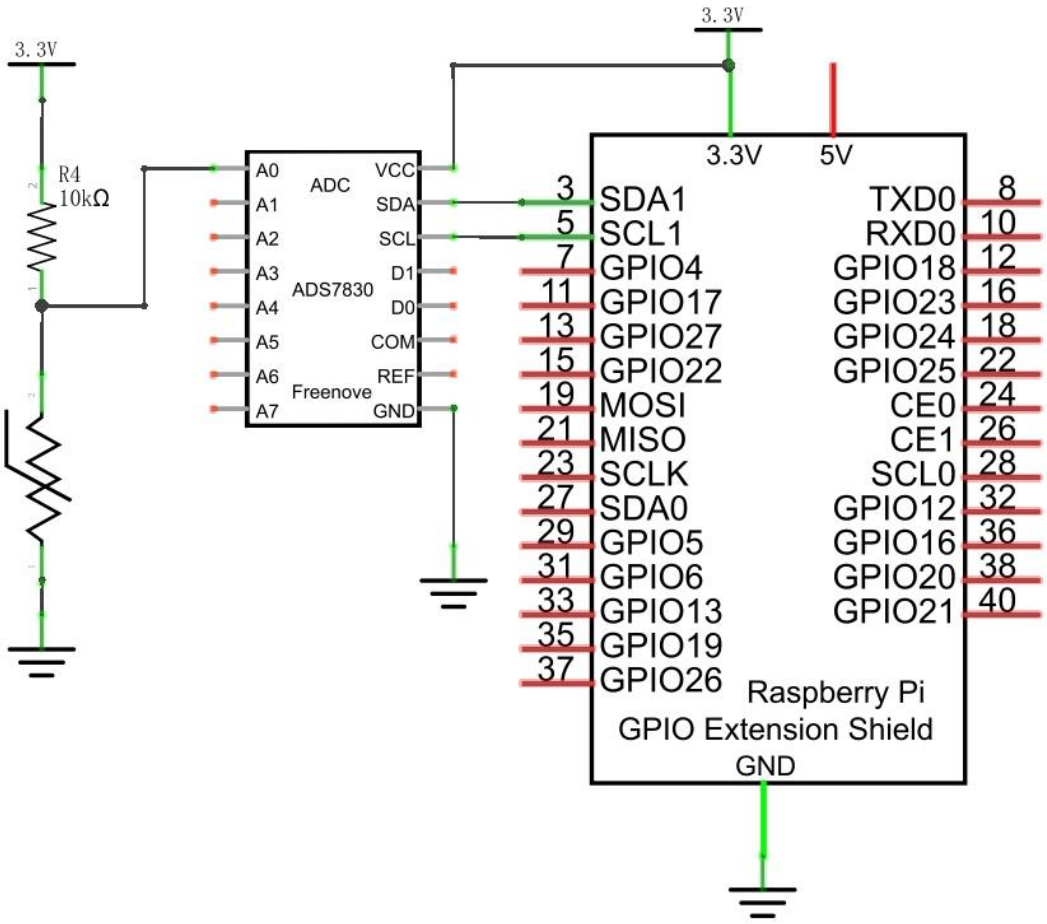
The formula for calculating temperature according to the circuit is shown below:

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

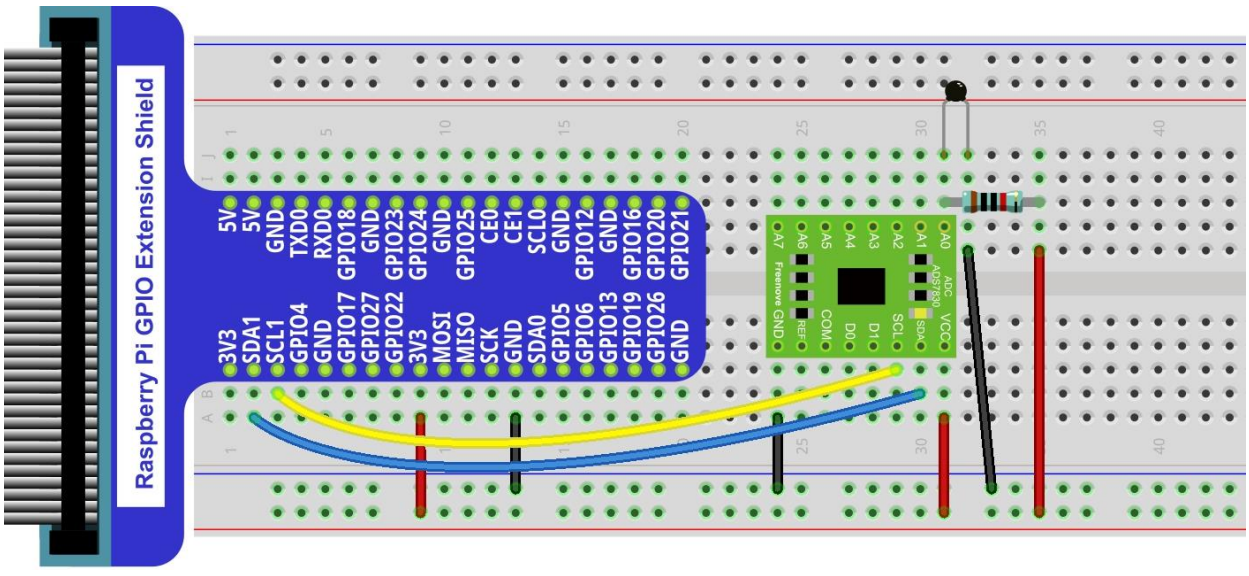
Circuit with ADS7830

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



Sketch

Sketch 8.1.1 Thermometer

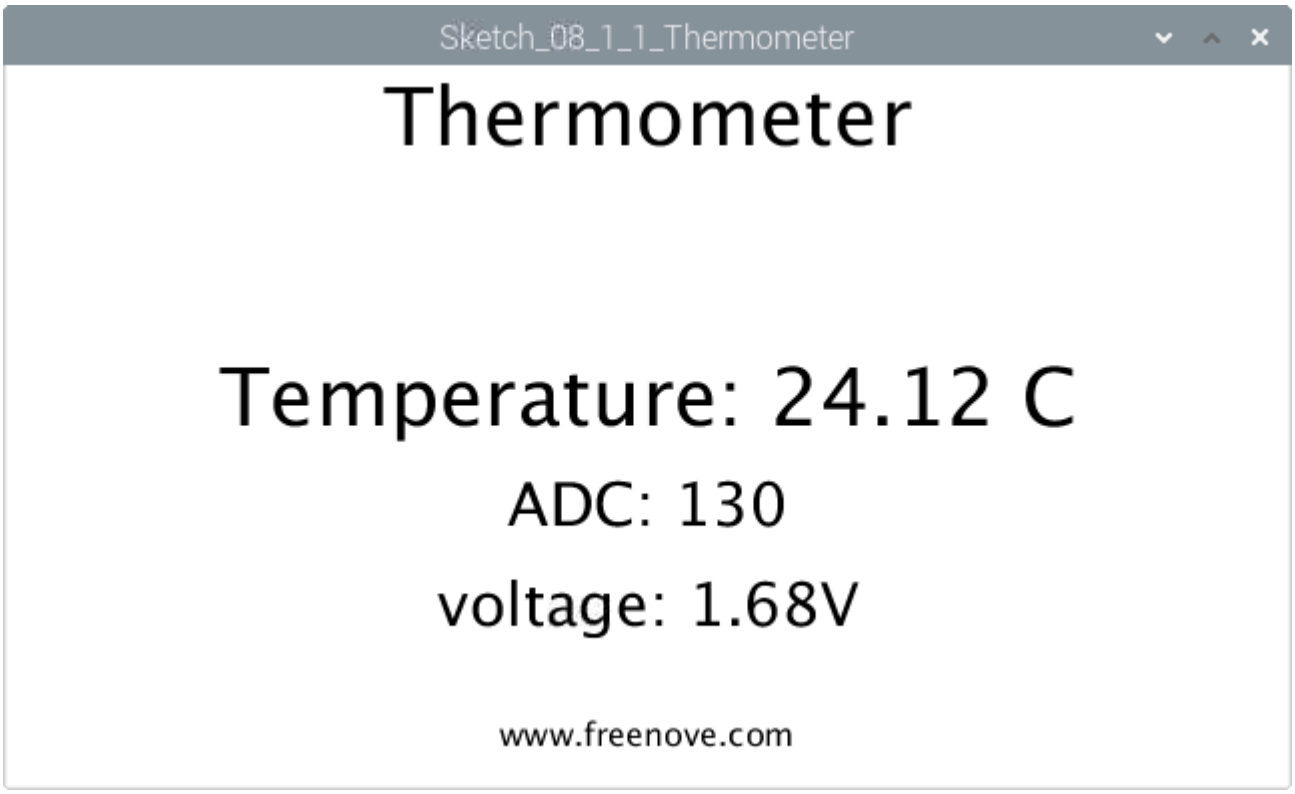
First observe the running result of the sketch, and then analyze the code.

1. Use Processing to open the file Sketch_08_1_1_Thermometer.

```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_08_1_1_Thermometer/Sketch_08_1_1_Thermometer.pde
```

2. Click on "RUN" to run the code.

After the program is executed, the Display Window will show the current temperature, the ADC value and the voltage value.



This project contains a lot of code files, and the core code is contained in the file Sketch_08_1_1_Thermometer. The other files only contain some custom classes.

The following is program code:

```
import processing.io.*;
//Create a object of class ADCDevice
ADCDevice adc = new ADCDevice();
void setup() {
  size(640, 360);
  if (adc.detectI2C(0x48)) {
    adc = new PCF8591(0x48);
  } else if (adc.detectI2C(0x4b)) {
    adc = new ADS7830(0x4b);
  } else {
    println("Not found ADC Module!");
    System.exit(-1);
  }
}
void draw() {
  int adcValue = adc.analogRead(0);    //Read the ADC value of channel 0
  float volt = adcValue*3.3/255.0;    //calculate the voltage
  float tempK,tempC,Rt;        //
  Rt = 10*volt / (3.3-volt);    //calculate the resistance value of thermistor
  tempK = 1/(1/(273.15+25) + log(Rt/10)/3950);  //calaulate temperature(Kelvin)
  tempC = tempK - 273.15;    //calaulate temperature(Celsius)

  background(255);
  titleAndSiteInfo();

  fill(0);
  textAlign(CENTER);    //set the text centered
  textSize(30);
  text("ADC: "+nf(adcValue, 0, 0), width / 2, height/2+50);
  textSize(30);
  text("voltage: "+nf(volt, 0, 2)+"V", width / 2, height/2+100);
  textSize(40);        //set text size
  text("Temperature: "+nf(tempC, 0, 2)+" C", width / 2, height/2);    //
}
void titleAndSiteInfo() {
  fill(0);
  textAlign(CENTER);    //set the text centered
  textSize(40);        //set text size
  text("Thermometer", width / 2, 40);    //title
  textSize(16);
  text("www.freenove.com", width / 2, height - 20);    //site
}
```

In this project code, first read ADC, and then calculate the current temperature according to the Ohm's law and temperature formula mentioned before, finally display them on Display Window.

```
int adc = pcf.analogRead(0);    //Read the ADC value of channel 0
float volt = adc*3.3/255.0;    //calculate the voltage
float tempK,tempC,Rt;          //
Rt = 10*volt / (3.3-volt);    //calculate the resistance value of thermistor
tempK = 1/(1/(273.15+25) + log(Rt/10)/3950);  //calaulate temperature(Kelvin)
tempC = tempK - 273.15;    //calaulate temperature(Celsius)
```

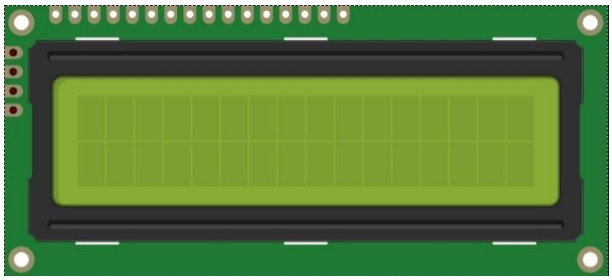
Chapter 9 I2C-LCD1602

In this chapter, we will learn a display screen, LCD1602.

Project 9.1 LCD

In the project, the current time and date will be displayed on the LCD1602 and Display Window.

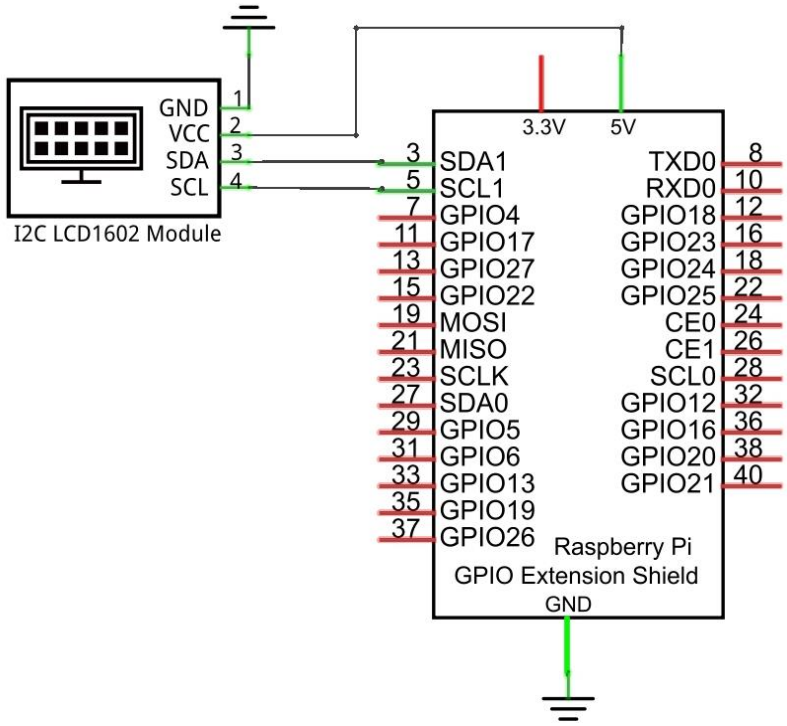
Component List

Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	Jumper M/M x4 
I2C LCD1602 Module x1 	

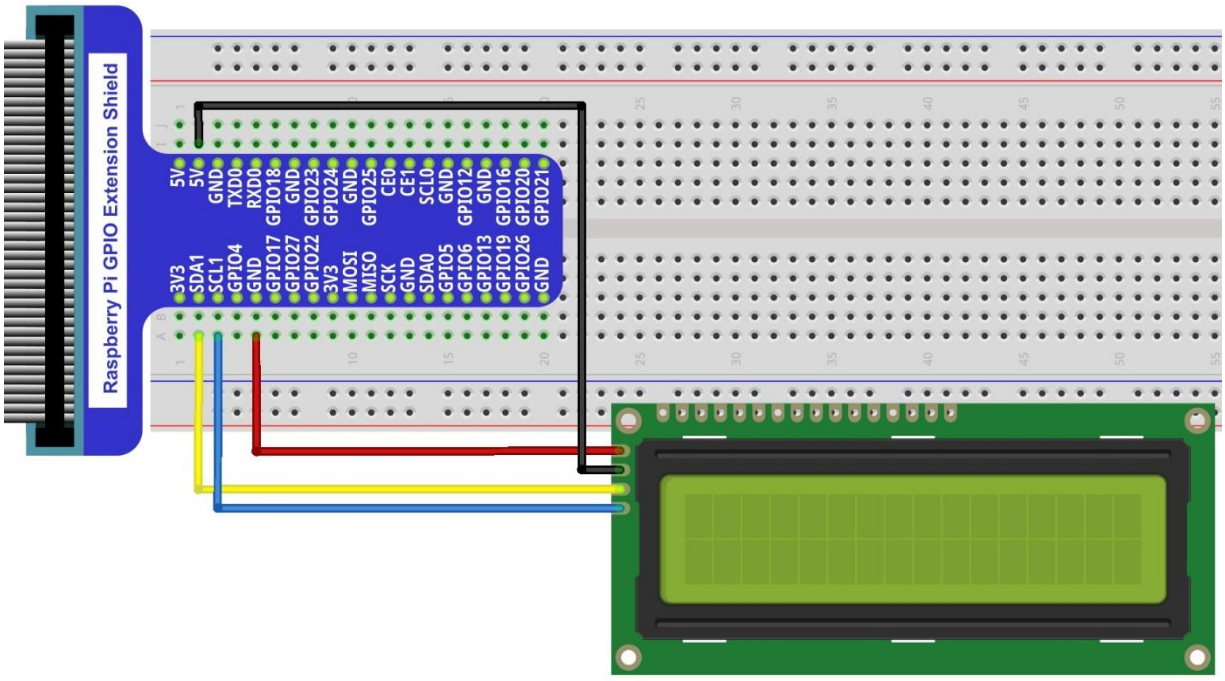
Circuit

Note that the power supply for I2CLCD1602 in this circuit is 5V.

Schematic diagram



Hardware connection



Sketch

Sketch 9.1.1 LCD

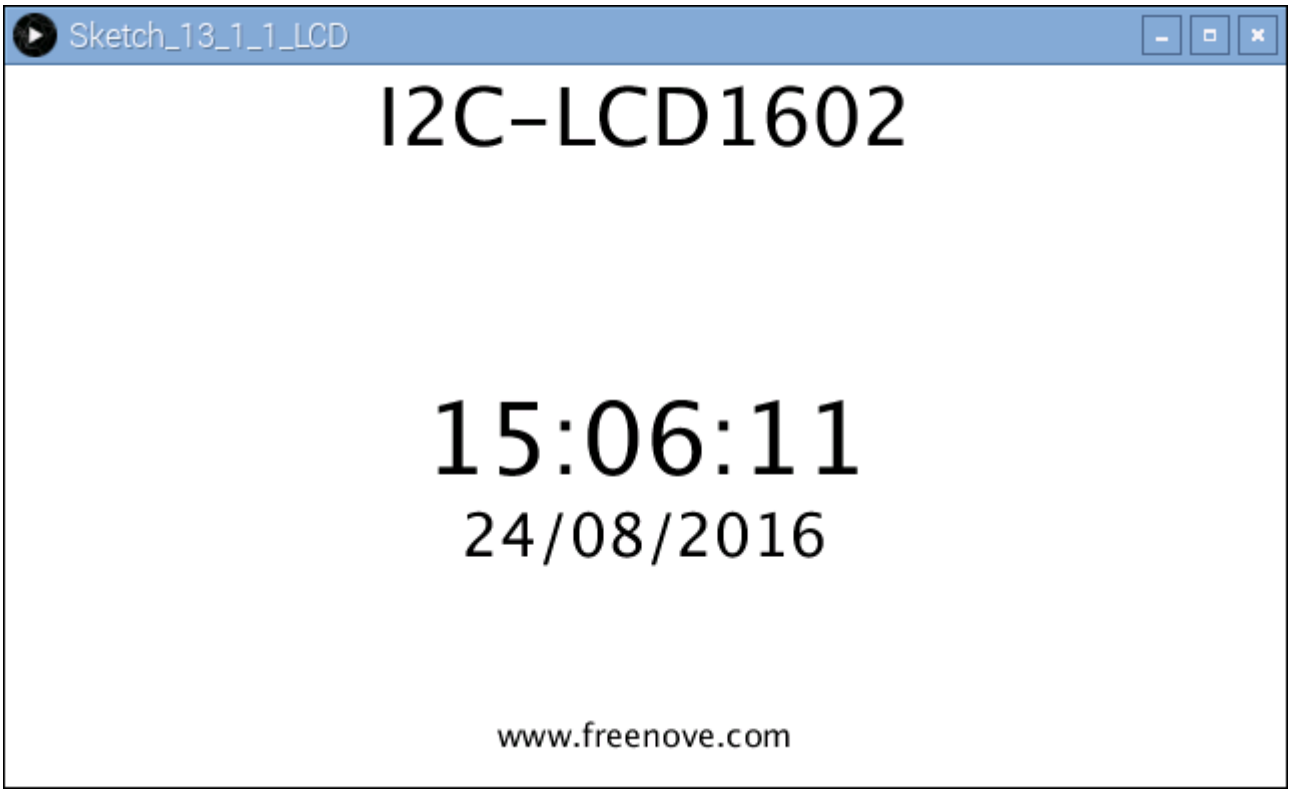
First observe the results of the code and the phenomenon, and then analyze the code.

1. Use Processing to open the file Sketch_9_1_1_LCD.

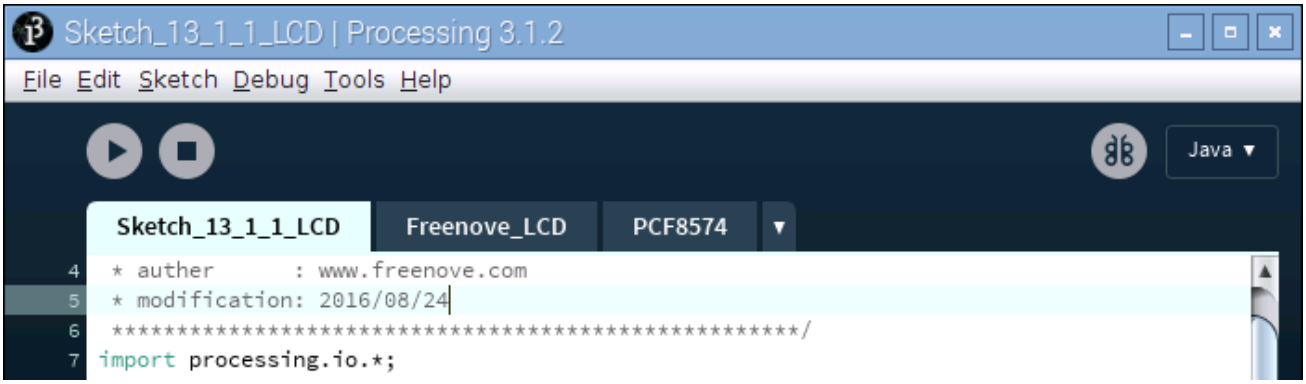
```
processing ~/Freenove_Kit/Processing/Sketches/Sketch_9_1_1_LCD/Sketch_9_1_1_LCD.pde
```

2. Click on "RUN" to run the code.

After the program is executed, both LCD in the circuit and the Display Window will show the current time and date.



This project contains a lot of code files, and the core code is contained in the file Sketch_9_1_1_LCD. The other files only contain some custom classes.



The following is program code:

```
import processing.io.*;
//Create a object of class PCF8574
PCF8574 pcf = new PCF8574(0x27);
Freenove_LCD1602 lcd;  //Create a lcd object
String time = "";
String date = "";
void setup() {
  size(640, 360);
  lcd = new Freenove_LCD1602(pcf);
  frameRate(2);    //set display window frame rate for 2 HZ
}
void draw() {
  background(255);
  titleAndSiteInfo();
  //get current time
  time = nf(hour(), 2, 0) + ":" + nf(minute(), 2, 0) + ":" + nf(second(), 2, 0);
  //get current date
  date = nf(day(), 2, 0)+"/"+nf(month(), 2, 0)+"/"+nf(year(), 2, 0);
  lcd.position(4, 0);  //show time on the lcd display
  lcd.puts(time);
  lcd.position(3, 1);  //show date on the lcd display
  lcd.puts(date);
  showTime(time, date);  //show time/date on the display window
}
void showTime(String time, String date) {
  fill(0);
  textAlign(CENTER, CENTER);
  textSize(50);
  text(time, width/2, height/2);
  textSize(30);
  text(date, width/2, height/2+50);
}
void titleAndSiteInfo() {
  fill(0);
  textAlign(CENTER);    //set the text centered
  textSize(40);         //set text size
  text("I2C-LCD1602", width / 2, 40);    //title
  textSize(16);
  text("www.freenove.com", width / 2, height - 20);    //site
}
```

First create a PCF8574 class object "pcf", and take "pcf" as a parameter to create a LCD1602 class object. And then define the variable time to store date and time. Display window need not refresh frequently. Therefore, the frame rate can be set to 1Hz or 2Hz.

```
PCF8574 pcf = new PCF8574(0x27);
Freenove_LCD1602 lcd;  //Create a lcd object
String time = "";
String date = "";
void setup() {
  size(640, 360);
  lcd = new Freenove_LCD1602(pcf);
  frameRate(2);    //set display window frame rate for 2 HZ
}
```

In the function draw(), get the current time and date, and display them on the LCD1602 and Display Window.

```
void draw() {
  background(255);
  titleAndSiteInfo();
  //get current time
  time = nf(hour(), 2, 0) + ":" + nf(minute(), 2, 0) + ":" + nf(second(), 2, 0);
  //get current date
  date = nf(day(), 2, 0)+"/"+nf(month(), 2, 0)+"/"+nf(year(), 2, 0);
  lcd.position(4, 0);  //show time on the lcd display
  lcd.puts(time);
  lcd.position(3, 1);  //show date on the lcd display
  lcd.puts(date);
  showTime(time, date);  //show time/date on the display window
}
```

About class PCF8574:

class PCF8574

This is a custom class that is used to control the integrated circuit PCF8574.

```
public PCF8574(int addr)
```

Constructor, used to create a PCF8574 class object, parameters for the I2C device address of PCF8574.

```
public int digitalRead(int pin)
```

Used to read the value(HIGH/LOW) of one of the ports.

```
public int readByte()
```

Used to read values of all ports.

```
public void digitalWrite(int pin, int val)
```

Write data(HIGH/LOW) to a port.

```
public void writeByte(int data)
```

Write data to all ports.

About class Freenove_LCD:

class Freenove_LCD

This is a custom class that is currently only used to control the I2C-LCD1602 connected to PCF8574.

```
public Freenove_LCD1602(PCF8574 ipcf)
```

Constructor, used to create Freenove_LCD1602 class object, the parameter for PCF8574 class object.

```
public void putChar(char data)
```

Write a character to the LCD screen.

```
public void puts(String str)
```

Write a string to the LCD screen.

```
public void display(boolean state)
```

Turn on/off LCD.

```
public void lcdCursor(boolean state)
```

Turn on/off Cursor.

```
public void cursorBlink(boolean state)
```

Turn on/off Cursor Blink.

```
public void position(int x, int y)
```

Set the location of Cursor.

```
public void home()
```

Set the Cursor to home.

```
public void lcdClear()
```

Clear the screen.

```
public void backLightON()  &  public void backLightOFF()
```

Turn on/off the backlight.

```
public void scrollDisplayLeft()  &  public void scrollDisplayRight()
```

Shift screen of a unit to left/right.

```
public void leftToRight()  &  public void rightToLeft()
```

Set text direction as form left to right / from right to left.

```
public void autoScroll()  &  public void noAutoScroll()
```

Automatic shifting screen/turn off automatic shifting screen.

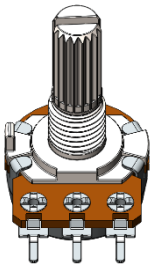
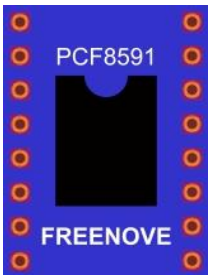
App 1 Oscilloscope

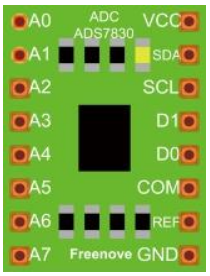
We have used the ADC module to read the voltage of potentiometer to realize function of the voltmeter, before. In this chapter, we will make a more complex virtual instrument, oscilloscope. Oscilloscope is a widely used electronic measuring instrument. It can get the electrical signals not directly observed into visible image to facilitate the analysis and study of various electrical signals change process.

App 1.1 Oscilloscope

Now, let's make an oscilloscope.

Component List

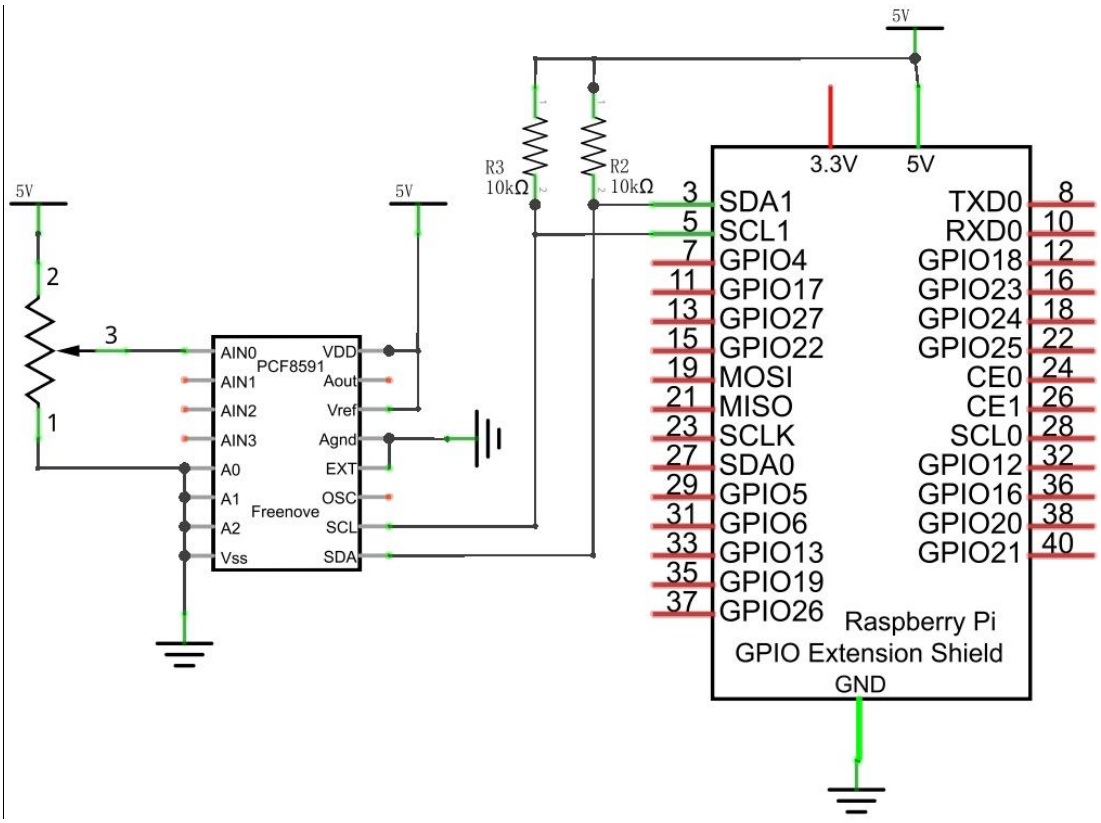
<table>
<tr><td colspan="2">Raspberry Pi x1
GPIO Extension Board & Wire x1
BreadBoard x1</td><td>Jumper M/M
  </td></tr>
<tr><td>Rotary potentiometer x1
  </td><td colspan="2">ADC module x1
 

or

</td><td>Resistor 10kΩ x2
  </td></tr>
</table>

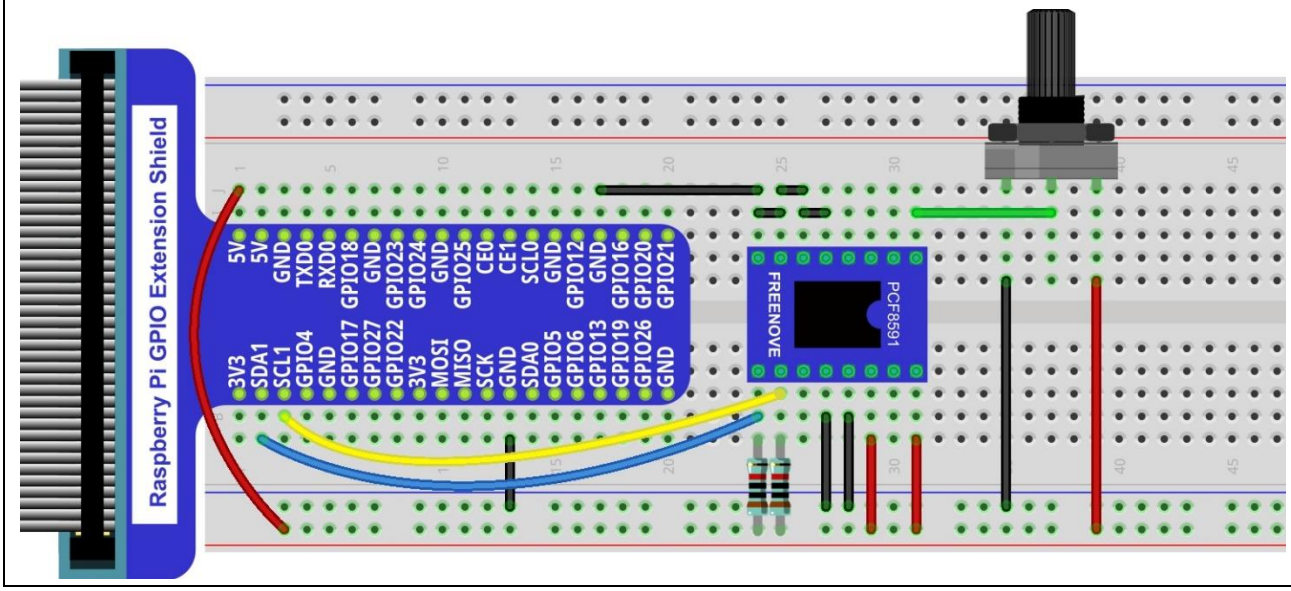
Circuit with PCF8591

Note that the power supply voltage of ADC module in this circuit is 5V.

Schematic diagram



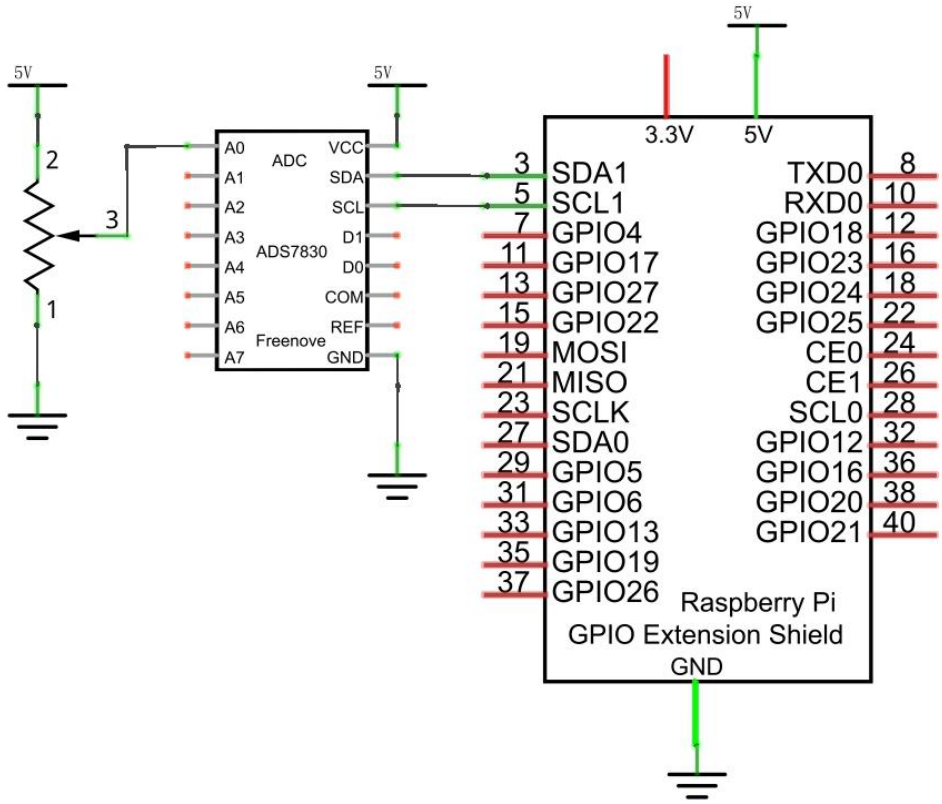
Hardware connection



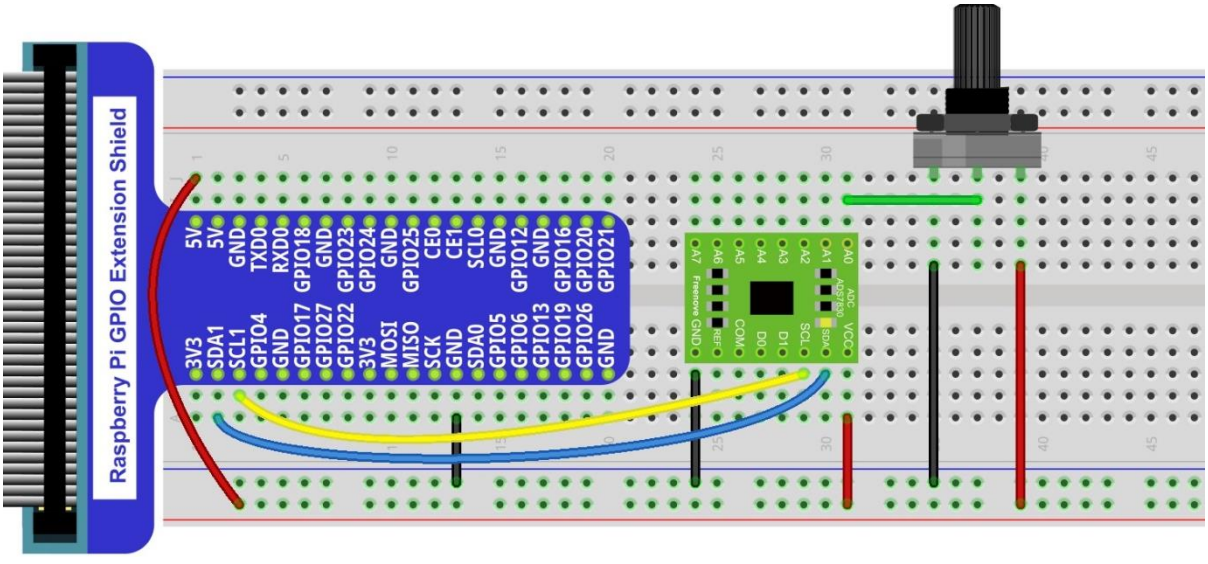
Circuit with ADS7830

Note that the power supply voltage of ADC module in this circuit is 5V.

Schematic diagram



Hardware connection



App

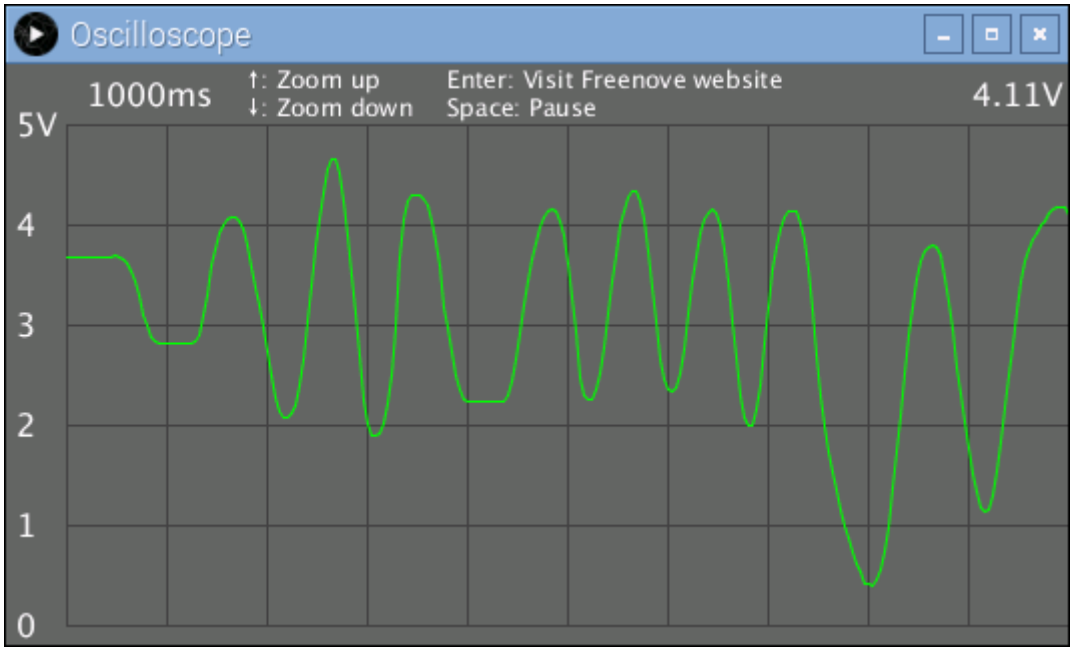
App 1.1.1 Oscilloscope

1. Use Processing to open the file App_01_1_1_ Oscilloscope.

```
processing ~/Freenove_Kit/Processing/Apps/App_01_1_1_Oscilloscope/App_01_1_1_Oscilloscope.pde
```

2. Click on "RUN" to run the code.

After the program is executed, Display Window display as below. Rotating potentiometer can make following waveform.



The left side of the software interface is a voltage scale, which is used to indicate the voltage of the waveform. The "1000ms" on top left corner is the time of a square, and you can press "↑" and "↓" key on keyboard to adjust it.

The "0.00V" on top right corner is the voltage value of current signal.

You can press the space bar on keyboard to pause the display waveform, which is easy to view and analysis.

We believe that with the help of this oscilloscope, you can obtain more intuitive understanding of the actual work of some electronic circuits. It will help you complete the project and eliminate the trouble. You can export this App to an application used as a tool.

App 2 Snake Game

In this chapter, we will play a classic game, snake.

App 2.1 Snake Game

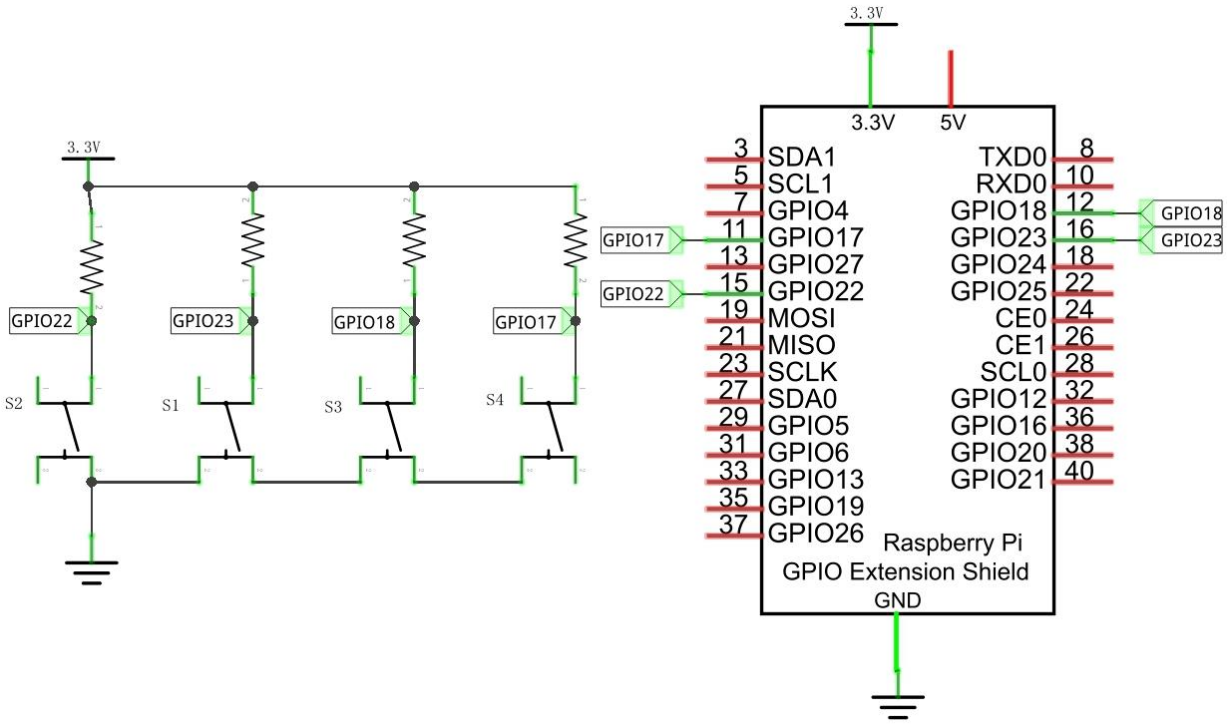
Now, let's create and experience our own game.

Component List

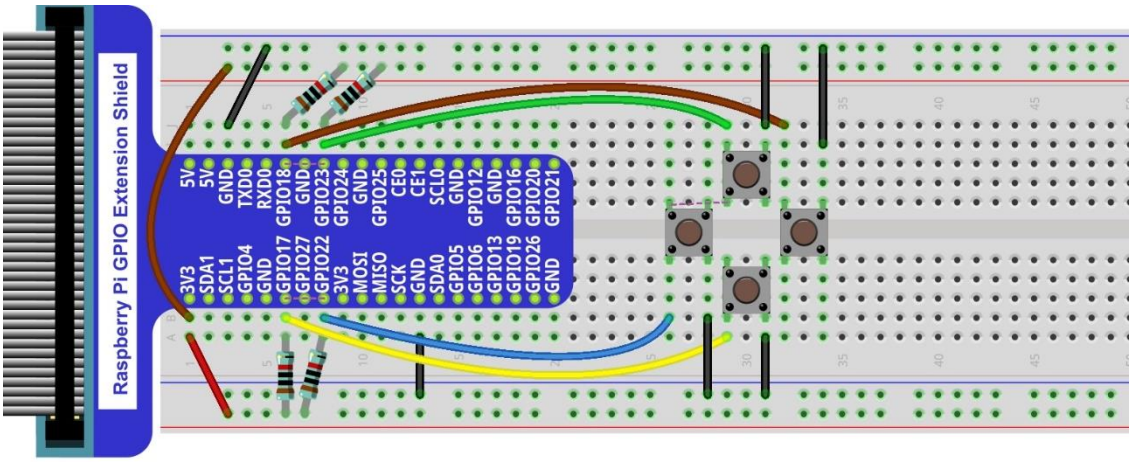
Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	Resistor 10KΩ x4 	Push button x4 
Jumper M/M x12 		

Circuit

Schematic diagram



Hardware connection



App

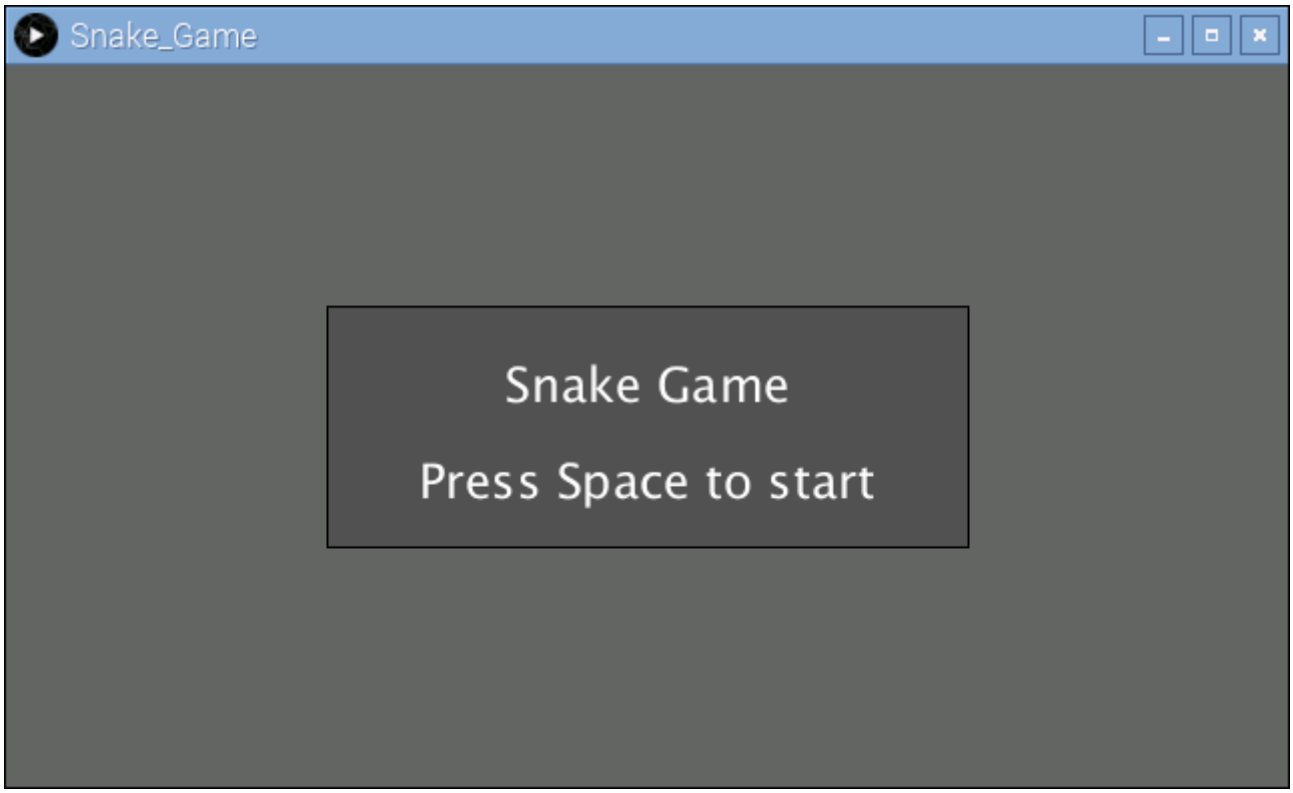
App 2.1.1 SnakeGame

1. Use Processing to open the file App_02_1_1_SnakeGame.

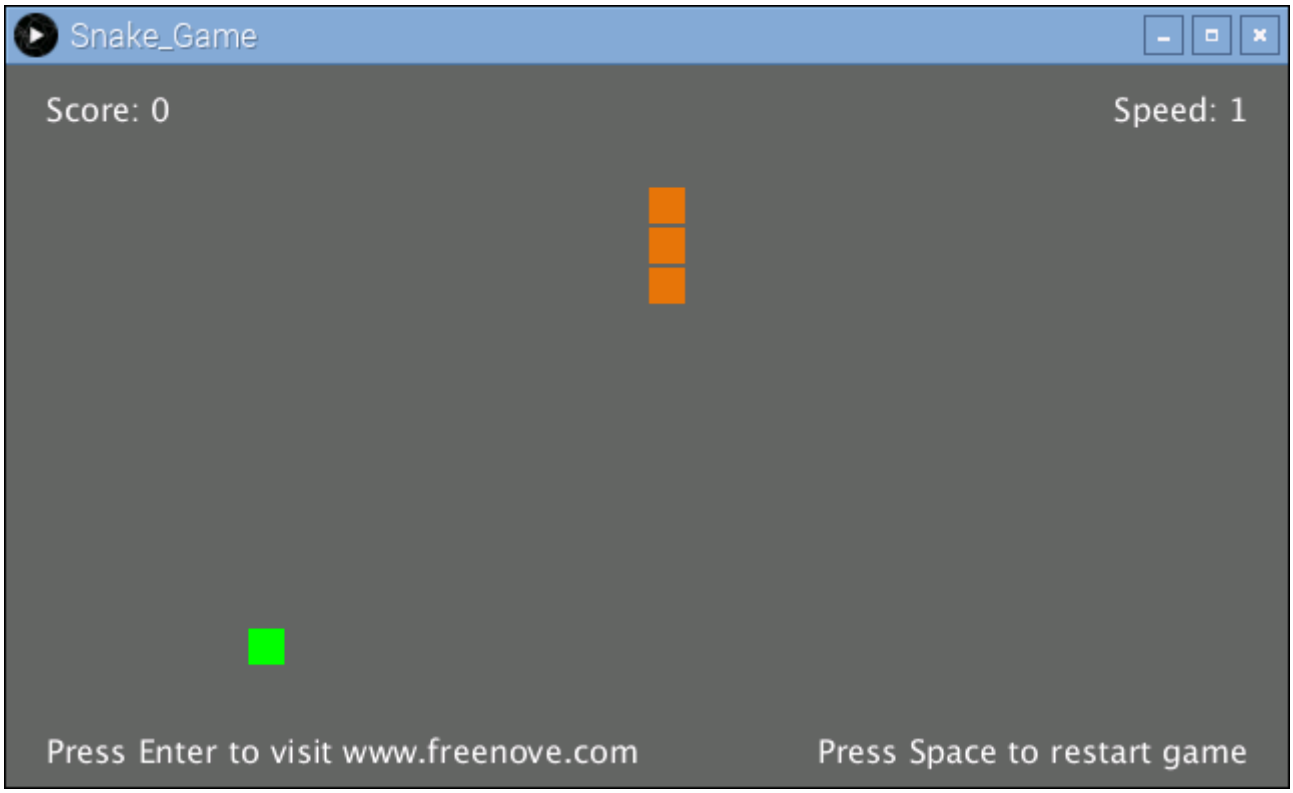
```
processing ~/Freenove_Kit/Processing/Apps/App_02_1_1_SnakeGame/App_02_1_1_SnakeGame.pde
```

2. Click on "RUN" to run the code.

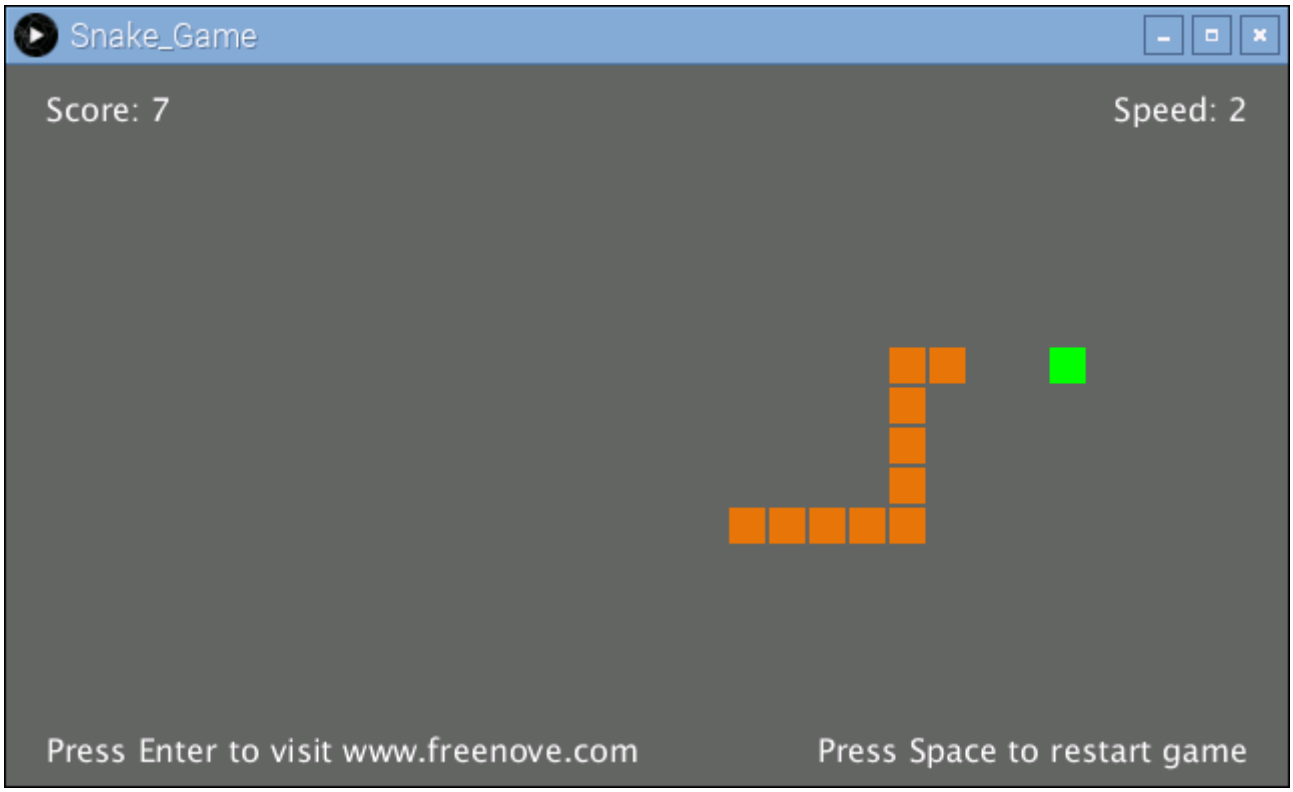
After the program is executed, Display Window displays as below.



Pressing the space can start the game:



You can control the movement direction of the snake through the four buttons in circuit or four arrow keys on the keyboard. The rules are the same as the classic Snake game:



When game is over, pressing the space can restart the game:

You can restart the game by pressing the space bar at any time during the game.

App 3 Tetris Game

In this chapter, we will play a game, tetris game.

App 3.1 Tetris Game

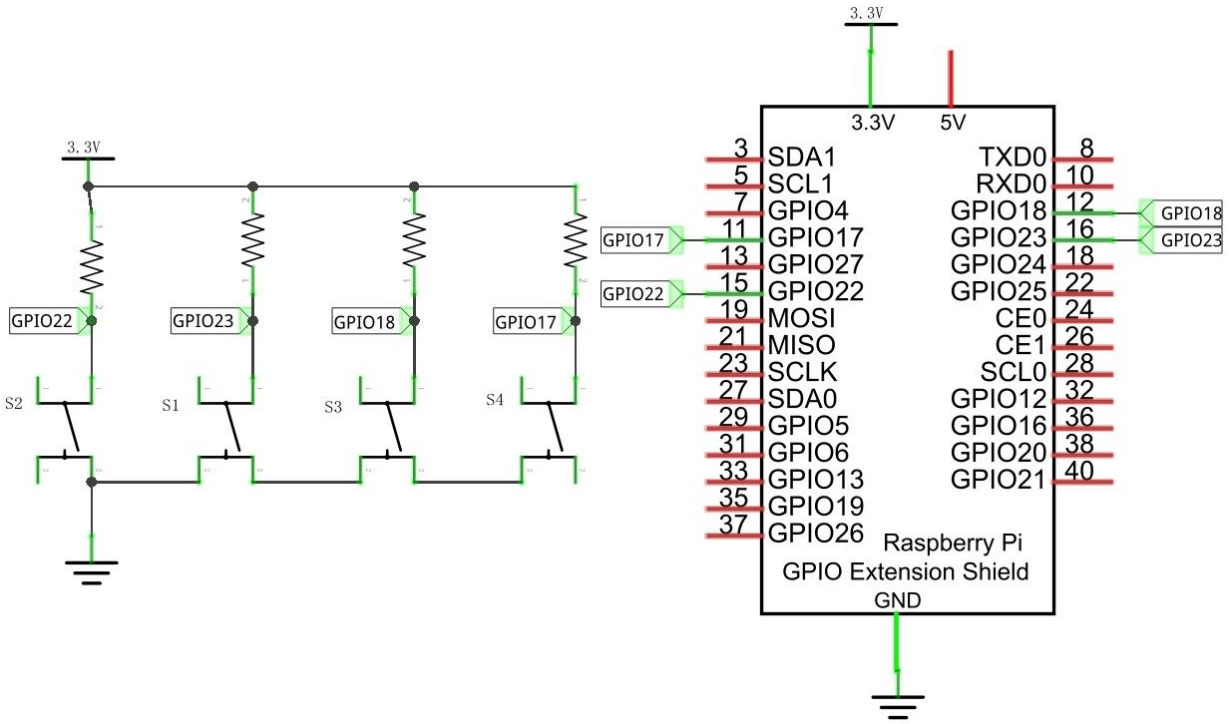
Now, let's create and experience our own game.

Component List

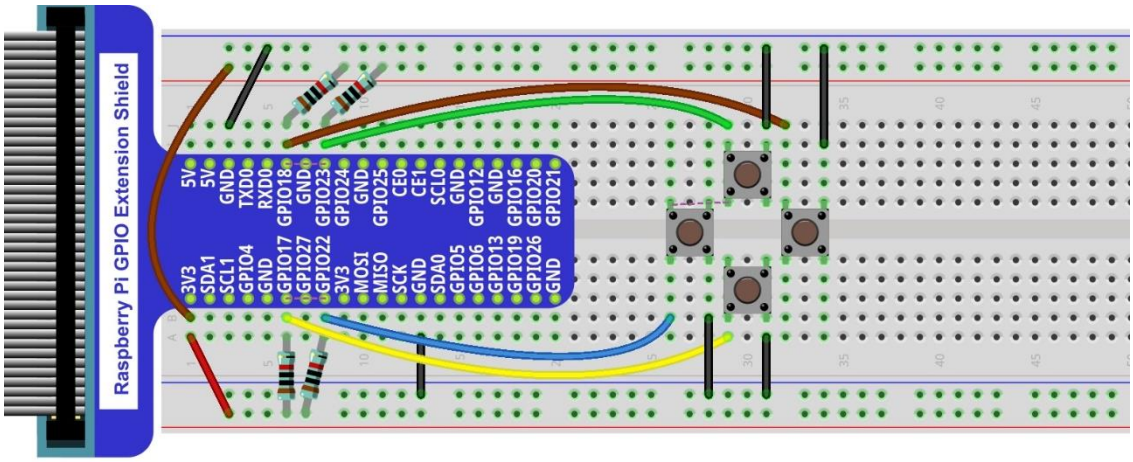
Raspberry Pi x1 GPIO Extension Board & Wire x1 BreadBoard x1	Resistor 10KΩ x4 	Push button x4 
Jumper M/M x12 		

Circuit

Schematic diagram



Hardware connection



App

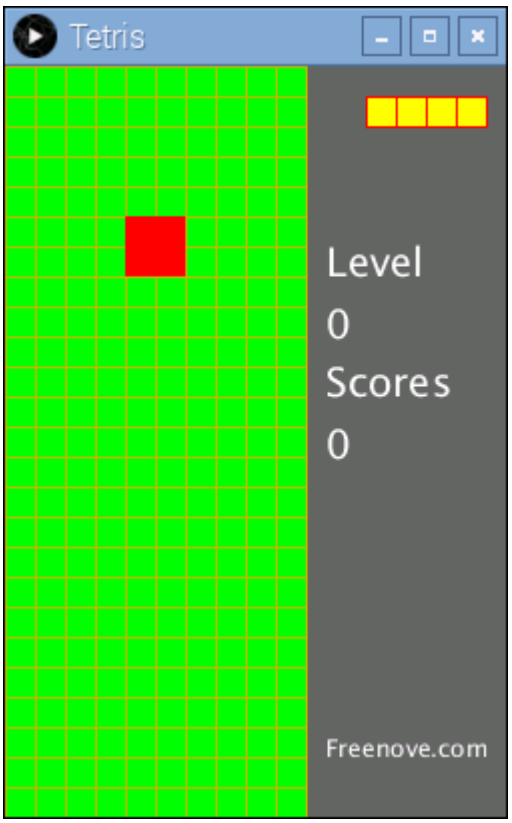
App 3.1.1 TetrisGame

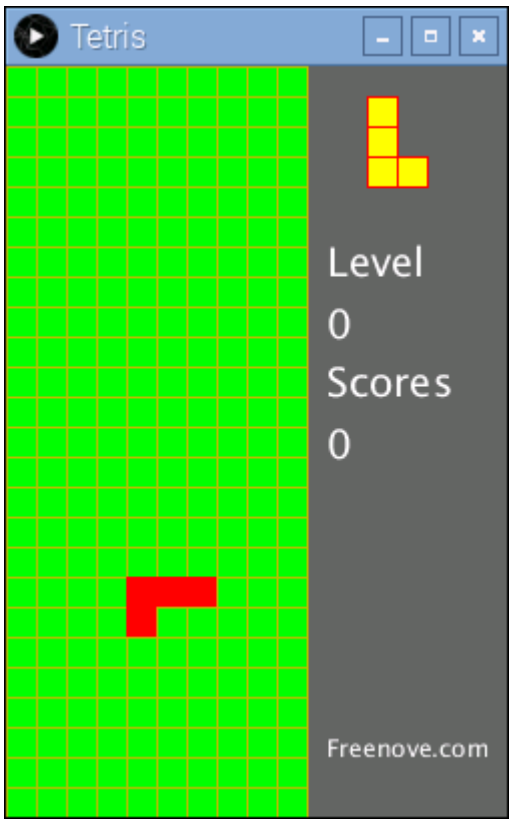
1. Use Processing to open the file App_03_1_1_TetrisGame.

```
processing ~/Freenove_Kit/Processing/Apps/App_03_1_1_TetrisGame/App_03_1_1_TetrisGame.pde
```

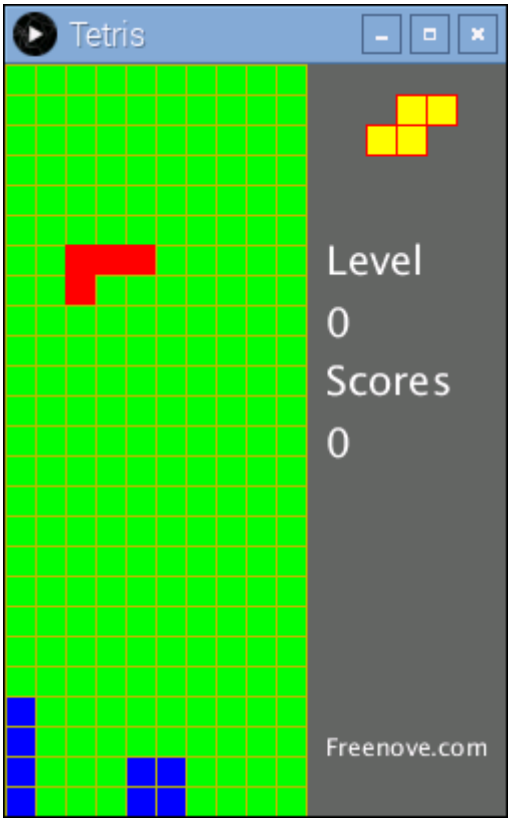
2. Click on "RUN" to run the code.

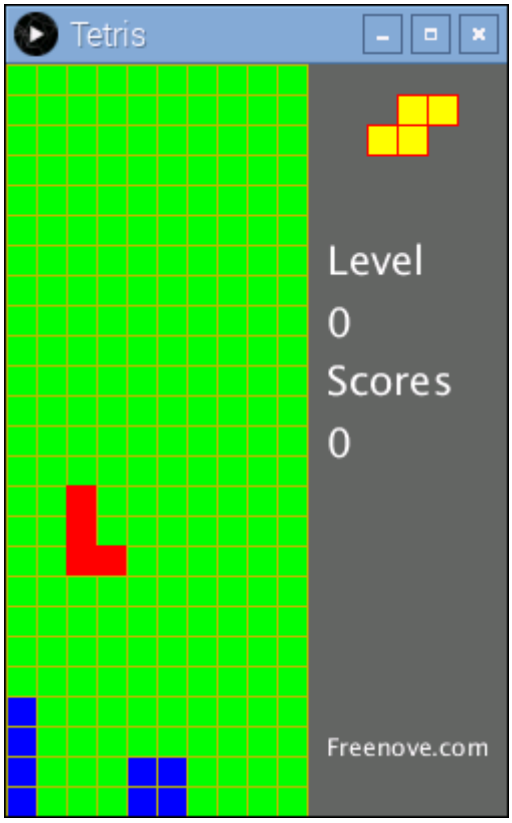
After the program is executed, Display Window displays as below.



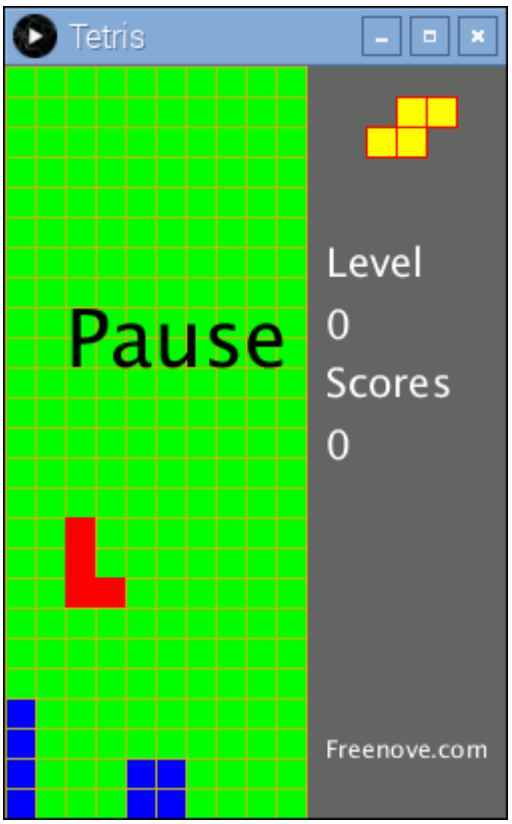


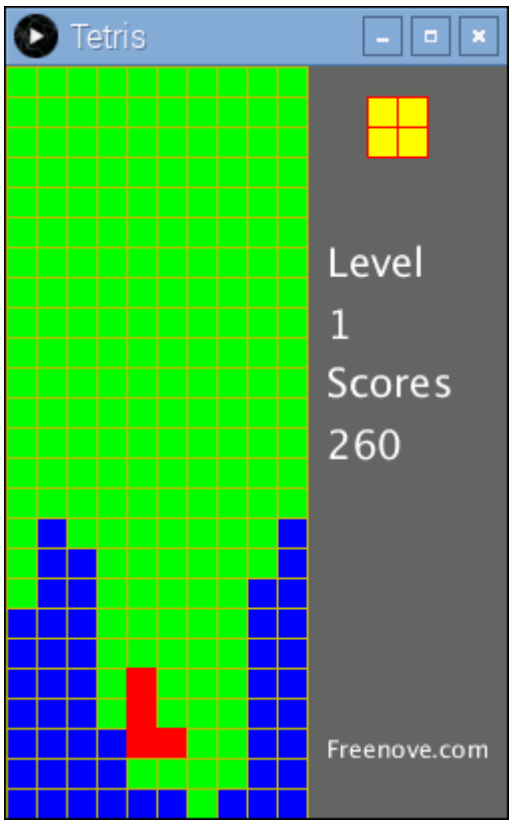
The left and right button in the circuit can control the moving of the falling block to left or right. And the button below can accelerate falling of the block. The button above is used for rotating of the block. Four direction keys on keboard can also be used to play the game.



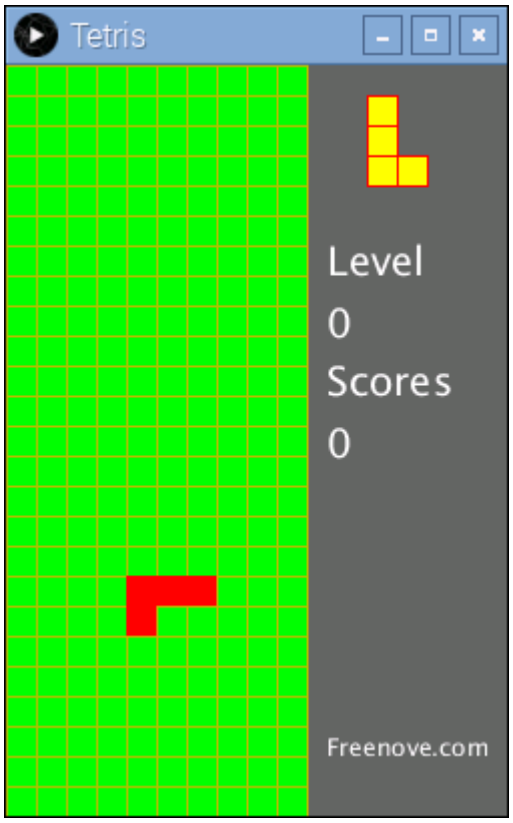


In the process of game, pressing the space bar on the keyboard can pause the game. The right side of the Display Window shows the next upcoming block, the current game speed and the current score. The more lines you eliminate once, the higher the scores. If you eliminate one line once, you will get 10 points. If you eliminate 4 lines once, you will get 70 points.





When the blocks are beyond the screen, the game is over. After the game is over, press the space bar to start a new game.



What's next?

Thanks for your reading.

This tutorial is all over here. If you find any mistakes, missions or you have other ideas and questions about contents of this tutorial or the kit and ect, please feel free to contact us, and we will check and correct it as soon as possible.

If you want to learn more about Arduino, Raspberry Pi, smart cars, robots and orther interesting products in science and technology, please continue to focus on our website. We will continue to launch cost-effective, innovative and exciting products.

Thank you again for choosing Freenove products.