About Our Company

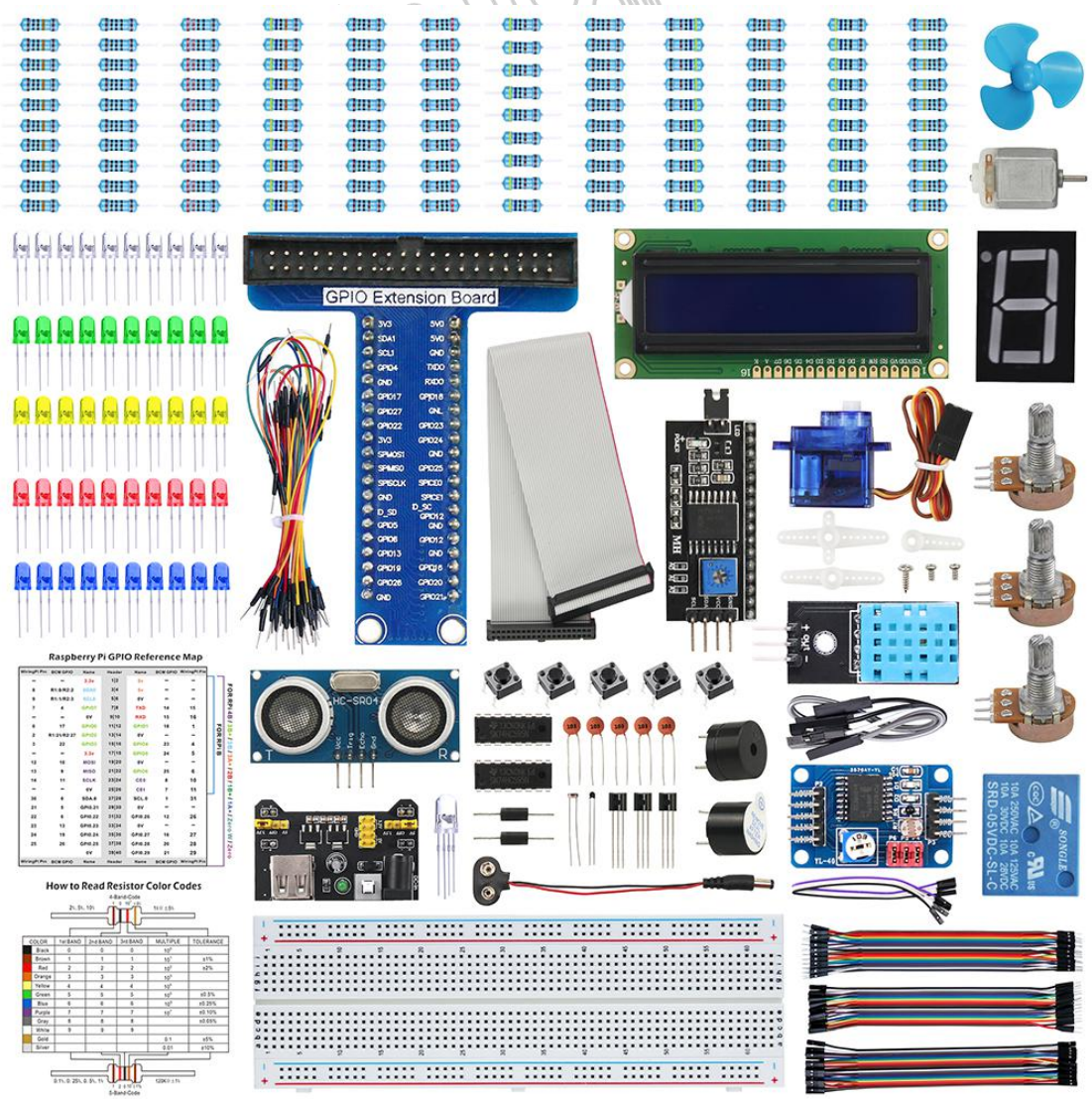
WayinTop, Your Top Way to Inspiration, is a professional manufacturer over 2,000 open source motherboards, modules, and components. From designing PCBs, printing, soldering, testing, debugging, and offering online tutorials, WayinTop has been committed to explore and demystify the wonderful world of embedded electronics, including but not limited to Arduino and Raspberry Pi. We aim to make the best designed products for makers of all ages and skill levels. No matter your vision or skill level, our products and resources are designed to make electronics more accessible. Founded in 2013, WayinTop has grown to over 100+ employees and a 50,000+ sq ft. factory in China by now. With our unremitting efforts, we also have expanded offerings to include tools, equipments, connector kits and various DIY products that we have carefully selected and tested.

US Amazon Store Homepage:
https://www.amazon.com/shops/A22PZZC3JNHS9L
CA Amazon Store Homepage:
https://www.amazon.ca/shops/A22PZZC3JNHS9L
UK Amazon Store Homepage:
https://www.amazon.co.uk/shops/A3F8F97TMOROPI
DE Amazon Store Homepage:
https://www.amazon.de/shops/A3F8F97TMOROPI
FR Amazon Store Homepage:
https://www.amazon.fr/shops/A3F8F97TMOROPI
IT Amazon Store Homepage:
https://www.amazon.it/shops/A3F8F97TMOROPI
ES Amazon Store Homepage:
https://www.amazon.es/shops/A3F8F97TMOROPI
JP Amazon Store Homepage:
https://www.amazon.co.jp/shops/A1F5OUAXY2TP0K

Preface

This tutorial is applicable for The Most Complete Ultimate Starter Kit for Raspberry Pi. If you have learned our C and python tutorials, or you have learned basic electronic circuits and programming, you can start learning this tutorial. Otherwise, we recommend that you had better learn our C and Python tutorials first. Sketch of this tutorial is written by java language in processing software. This tutorial has similar projects to C and python tutorials. And graphical man-machine interface is added to achieve perfect integration of electronic circuits, computer software, images and so on, which will let the readers fully experience the fun of programming and DIY.

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.



Content

Install Processing Software

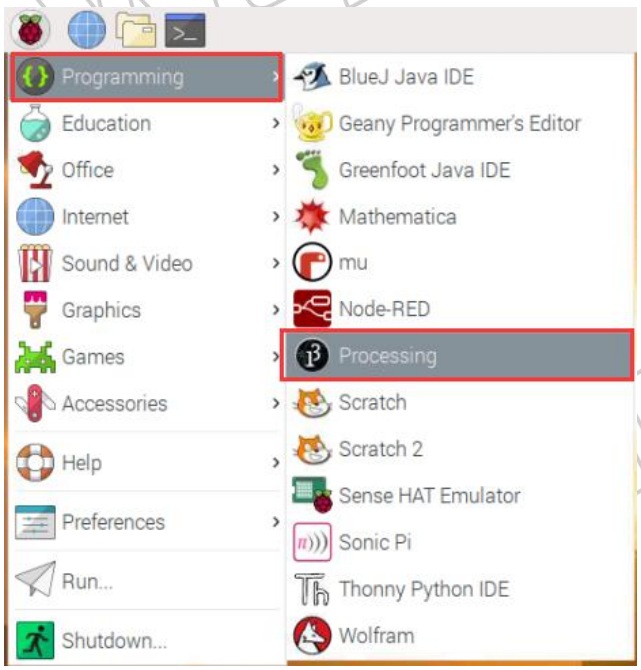
We need to install the programming software 'processing' on the Raspberry Pi first. This software makes programming very easy, and its installation method is also very simple.

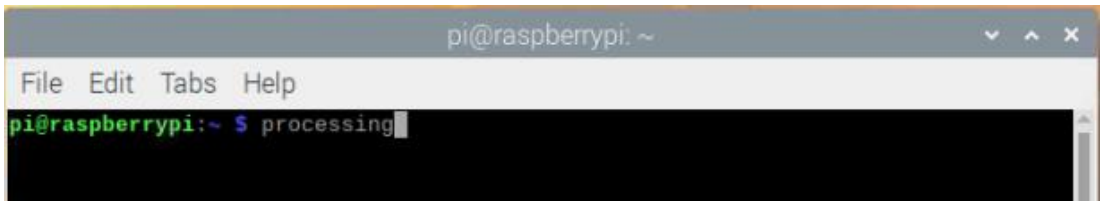
How to install it?

In the Raspberry Pi terminal enter the command ‘curl https://processing.org/download/install-arm.sh | sudo sh’

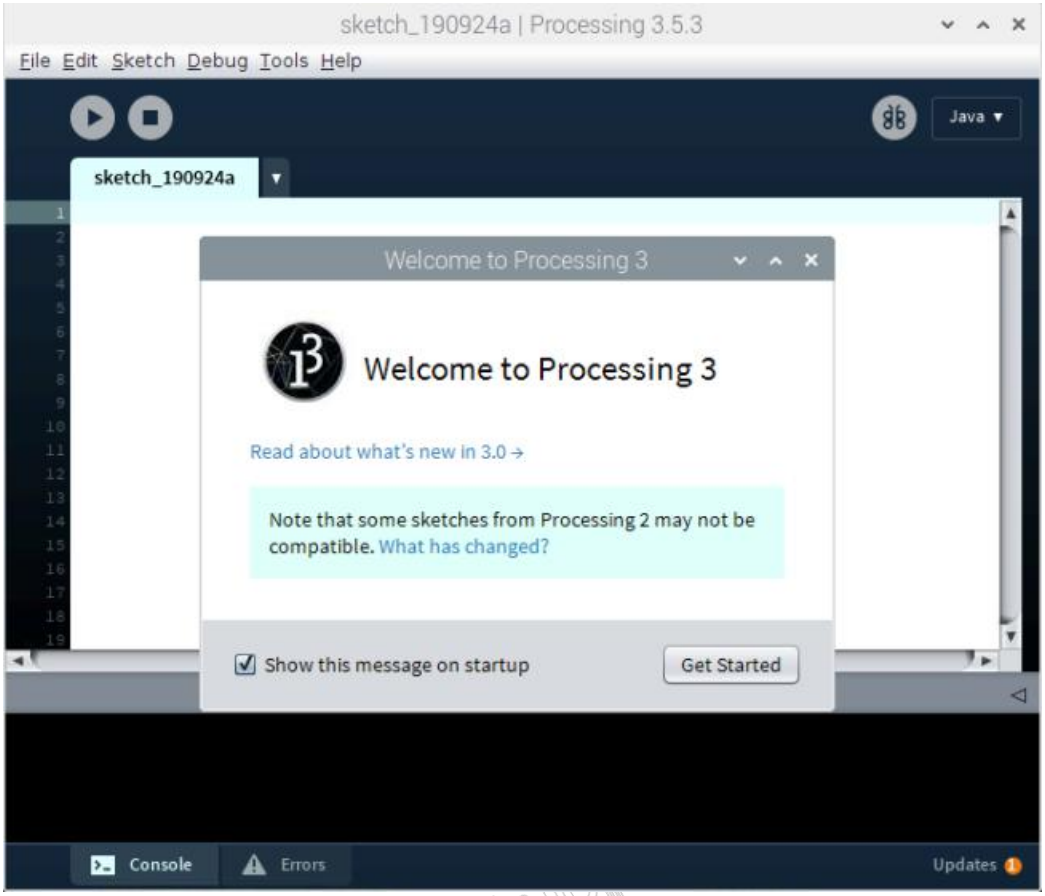
Ensure that your RPi always has the Internet to access in the installation process. You can also download and install the software by visiting the official website: https://processing.org/.

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

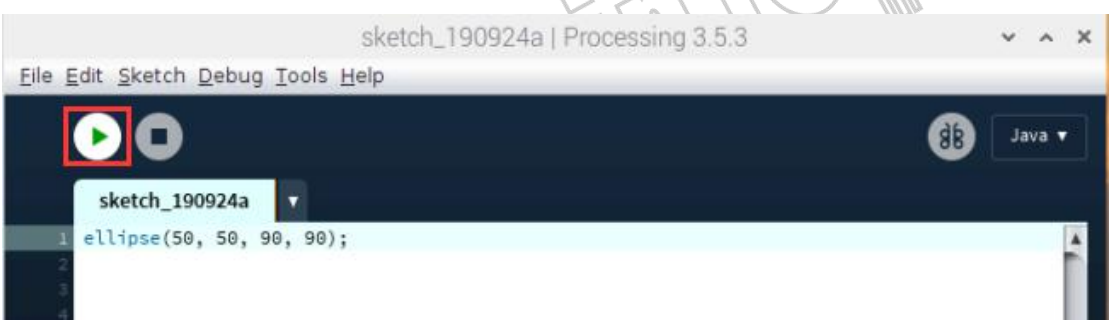




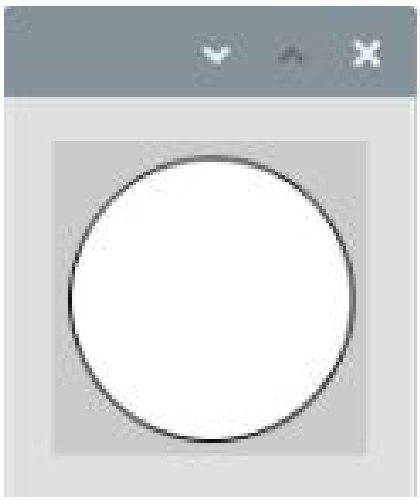
Interface of processing software is shown below:



In the editor, type ‘ellipse(50, 50, 90, 90);’ 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). As shown below:



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 needs to 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 study of the installation and use of the 'processing' software, and then started to use the software to make projects.

Lesson 1 LED

Overview

In this lesson, you will learn how to blink LED and learn the usage of some commonly used functions of processing software. In this project, we will make a Blink LED and let Display window of Processing Blink at the same time.

Parts Required:

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

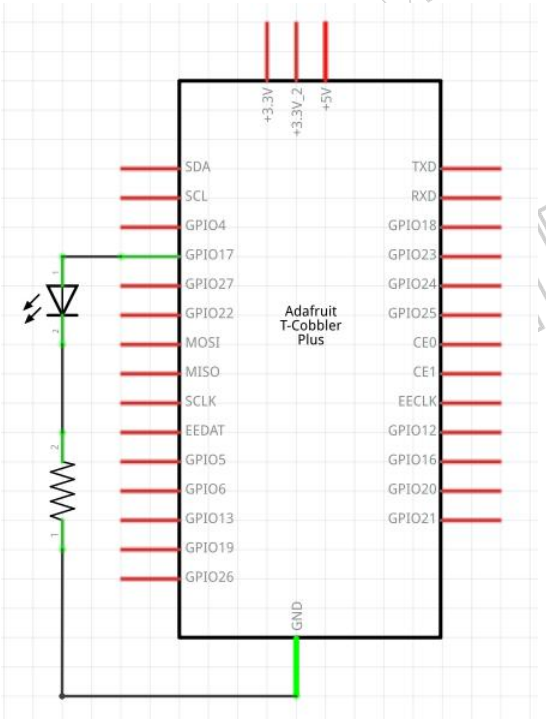
1 x Breadboard

1 x LED

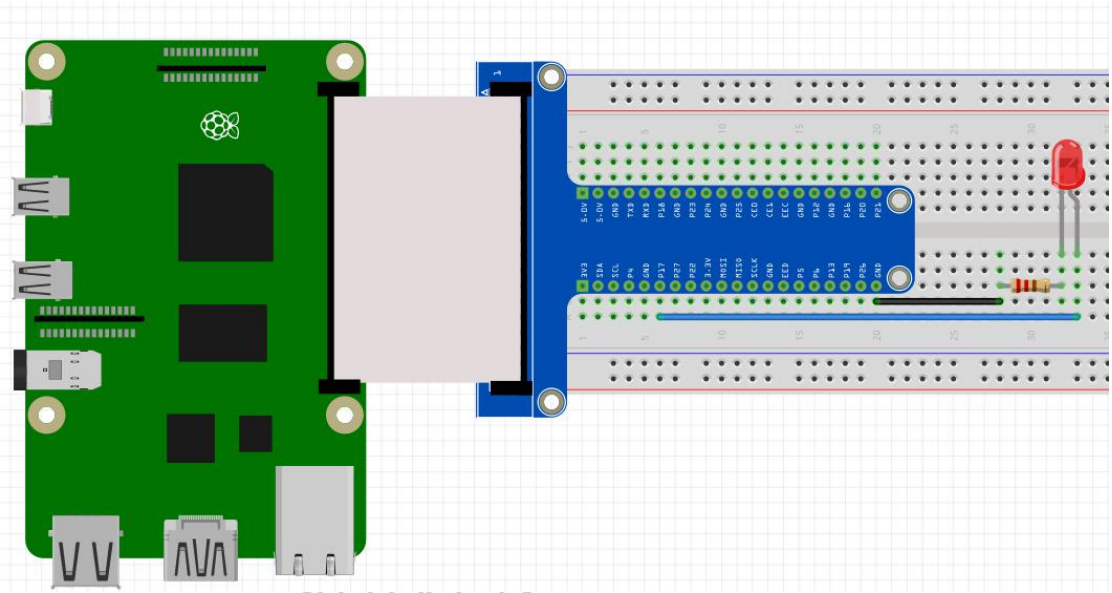
1 x 220 Ohm Resistor

Some Jumper Wires

Connection Schematic



Wiring Diagram



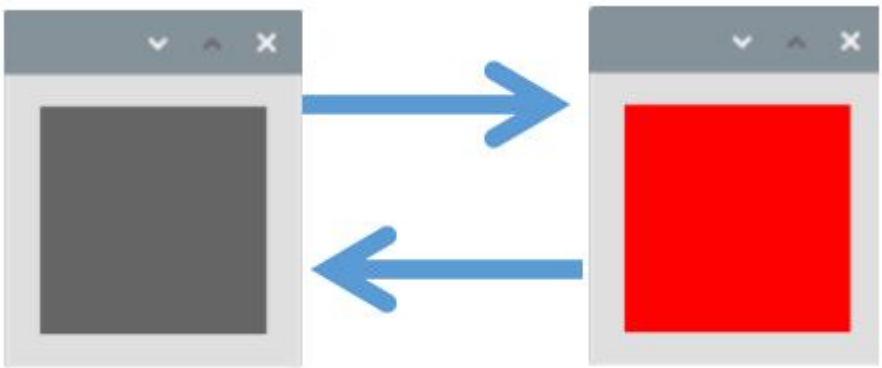
Because Numbering of GPIO Extension Shield is the same as RPi GPIO, latter Hardware connection diagram will only show the part of breadboard and GPIO Extension Shield.

Code

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

1. Enter processing code / Java / 1.LED / LED / LED.pde command in the terminal to open the code;
2. Click "RUN" button in the pop-up "processing" software to run the code;

After the program is executed, LED will start Blinking and background of Display window will change with the changing of LED state. As shown below：



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
   }
}
```

Code Interpretation

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

Processing codes usually have two functions: setup() and draw(), where the function setup() is only executed once, but the function draw() will be executed loop. 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, that's to say, the draw() function will be executed once per second. GPIO.pinMode (ledPin, GPIO.OUTPUT) is used to set 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
  }
}
```

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 loop repeats itself to achieve the effect of blink.

Function introduction

“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()”：Define the dimension of the display window width and height in units of pixels.

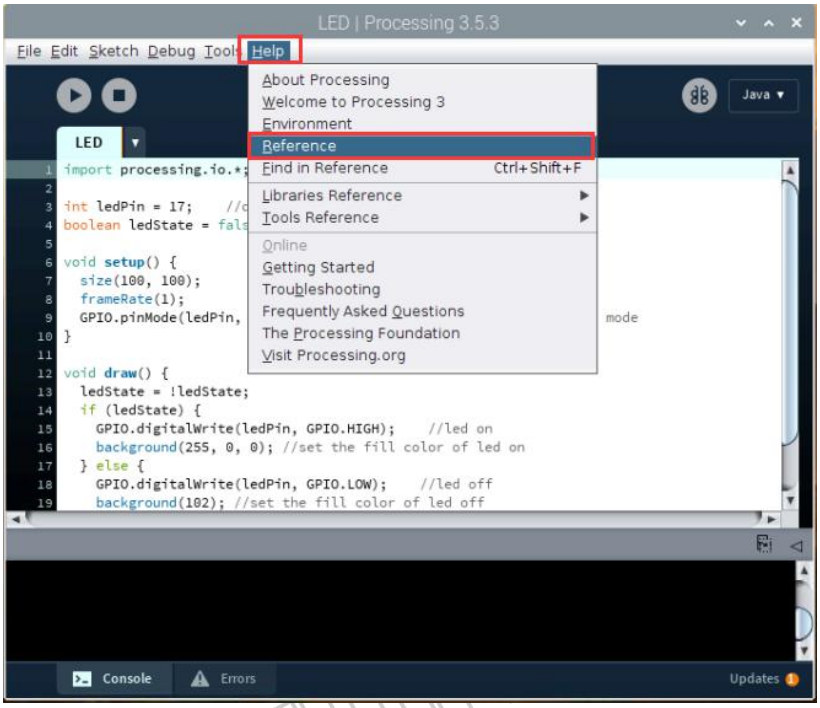
“framerate()”：Specify the number of frames to be displayed every second.

“background()”：Set the color used for the background of the Processing window.

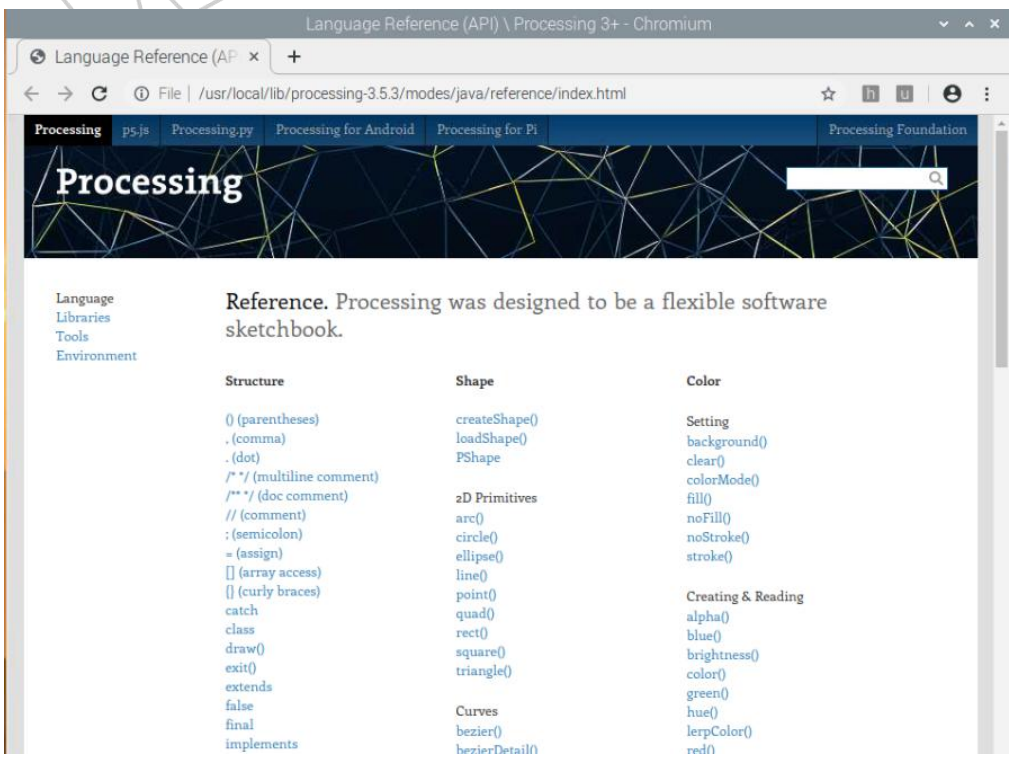
“GPIO.pinMode()”：Configure a pin to act either as input or output.

“GPIO.digitalWrite()”：Set 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”. As shown below:



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



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

Lesson 2 Mouse Control LED

Overview

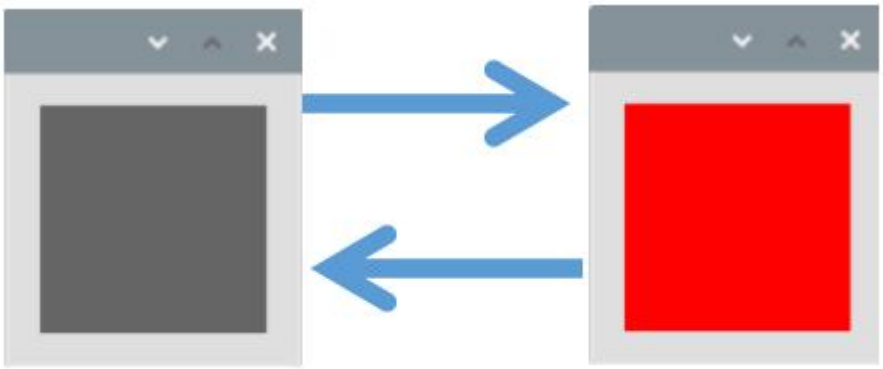
In this lesson, we will change the code based on the hardware of the LED course, and use the mouse to control the state of the LED.

Code

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

1. Enter the processing code / Java / 2.MouseLED / MouseLED / MouseLED.pde command in the terminal to open the code;
2. Click the "RUN" button in the pop-up "processing" software 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 becomes gray, as shown below.



The following is the 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 {
      GPIO.digitalWrite(ledPin, GPIO.LOW);
      background(102);
   }
}

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

Code Interpretation

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

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.

Lesson 3 LEDBar Graph

Overview

In this lesson, we will use the mouse to control the LED Bar Graph.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

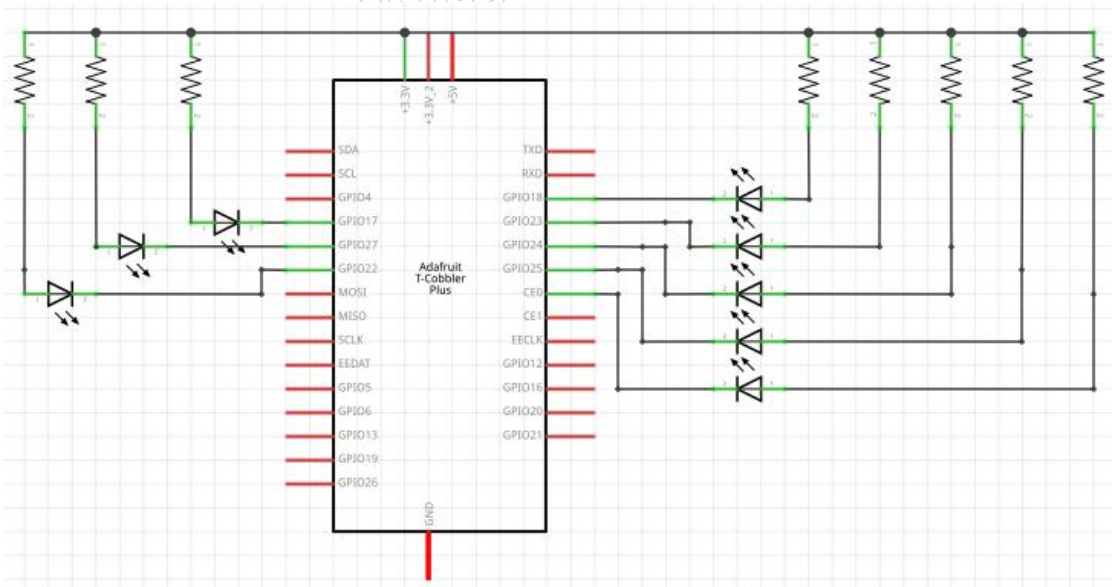
1 x Breadboard

8 x LED

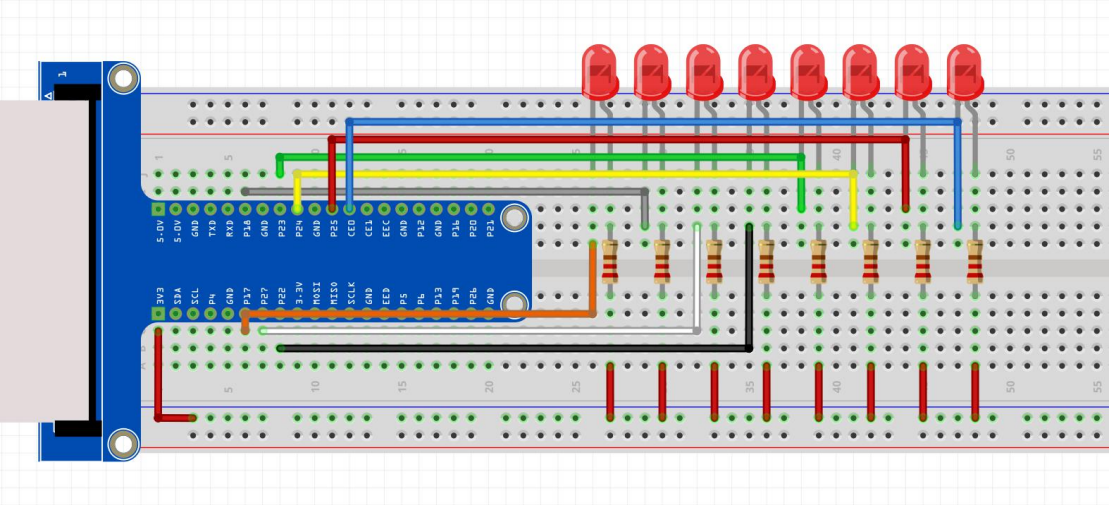
8 x 220 Ohm Resistor

Some Jumper Wires

Connection Diagram



Wiring Diagram



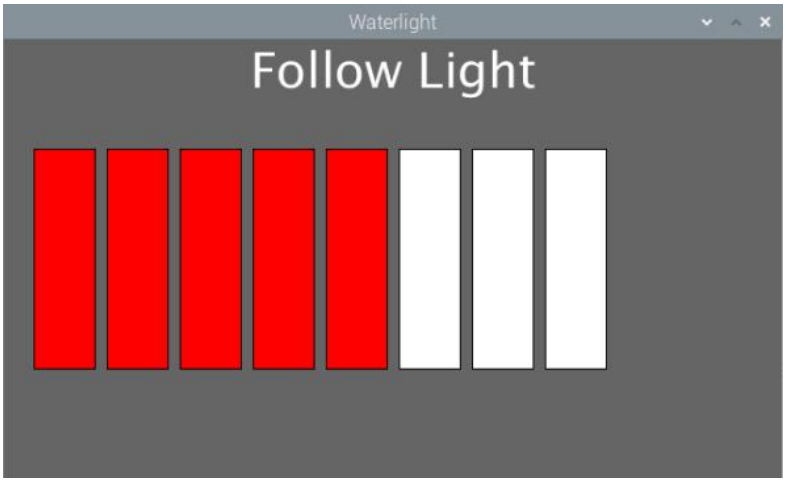
Code

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

1. Enter the processing code / Java / 3.Waterlight / Waterlight / Waterlight.pde command in the Command Window to open the code;

2. Click the "RUN" button in the pop-up "processing" software 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 becomes gray, as shown below.



The following is the program code:

```
import processing.io.*;

int leds[]={17, 18, 27, 22, 23, 24, 25, 8};

void setup() {
   size(640, 360);
   for (int i=0; i<8; i++) {
      GPIO.pinMode(leds[i], GPIO.OUTPUT);
   }
   background(102);
   textAlign(CENTER);
   textSize(40);
   text("Follow Light", width / 2, 40);
   textSize(16);
   text("www.keeyees.com", width / 2, height - 20);
}

void draw() {

for (int i=0; i<8; i++) {
      if (mouseX>(25+60*i)) {
         fill(255, 0, 0);

GPIO.digitalWrite(leds[i], GPIO.LOW);
      } else {
         fill(255, 255, 255);
         GPIO.digitalWrite(leds[i], GPIO.HIGH);
```

```
        }
        rect(25+60*i, 90, 50, 180);
    }
}
```

Code Interpretation

```
void draw() {
   for (int i=0; i<8; i++) {
      if (mouseX>(25+60*i)) {
         fill(255, 0, 0);
         GPIO.digitalWrite(leds[i], GPIO.LOW);
      } else {
         fill(255, 255, 255);

GPIO.digitalWrite(leds[i], GPIO.HIGH);
      }
      rect(25+60*i, 90, 50, 180);
   }
```

In the function draw(), we draw 8 rectangles to represent 8 LEDs of LED Bar Graph. We make rectangles on the left of mouse filled with red, corresponding LEDs turned on. And 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.

Lesson 4 PWM

Overview

In this lesson, we will learn how to make a “breathing LED” and the Display Window will show gradient LED pattern. How to control the brightness of the LED? We will use PWM to achieve this goal.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

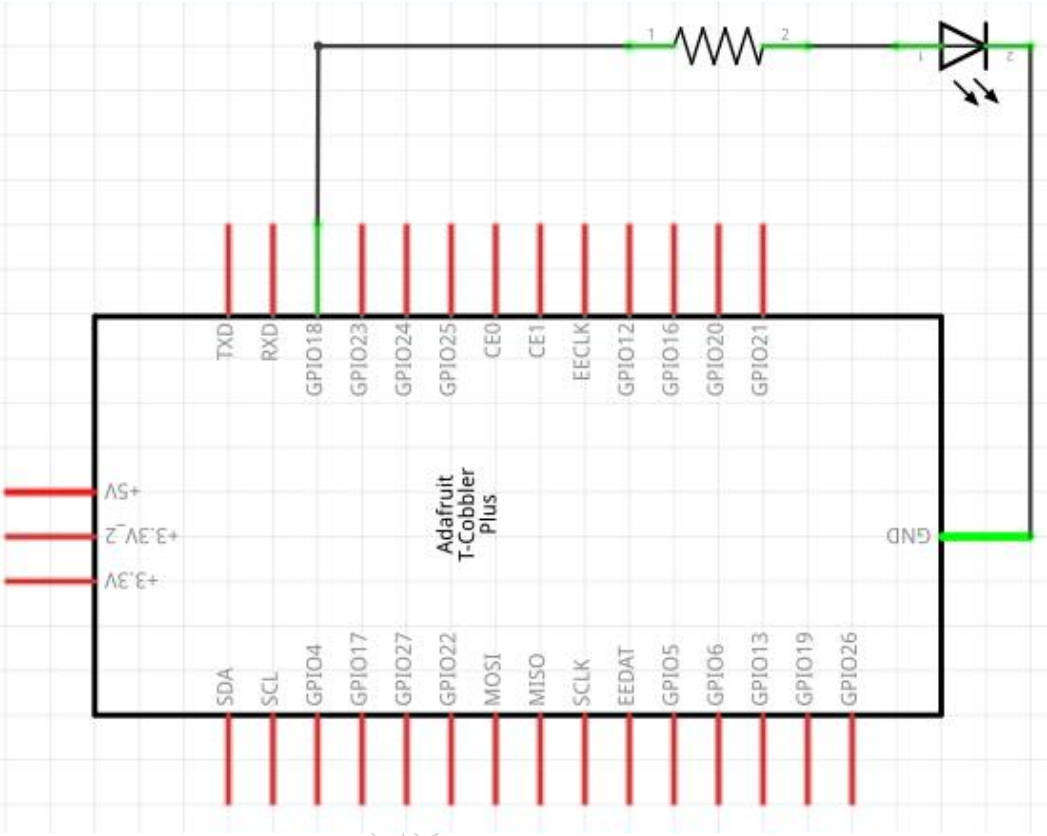
1 x Breadboard

1 x LED

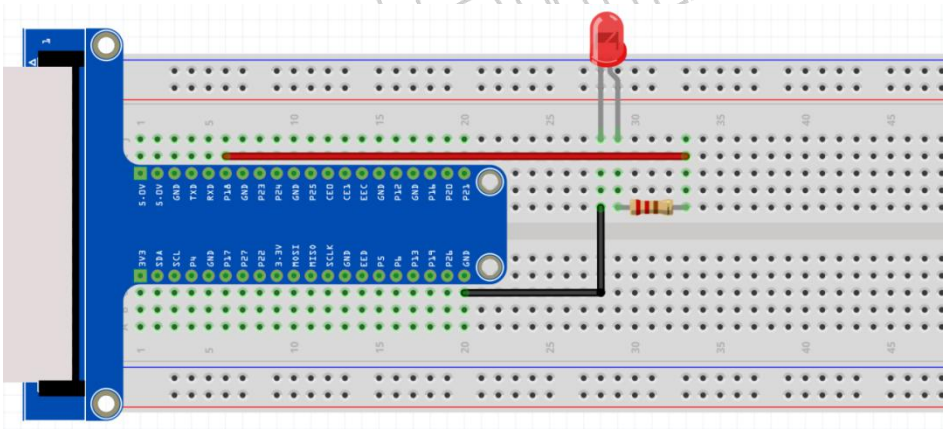
1 x 220 Ohm Resistor

Some Jumper Wires

Connection Diagram



Wiring Diagram

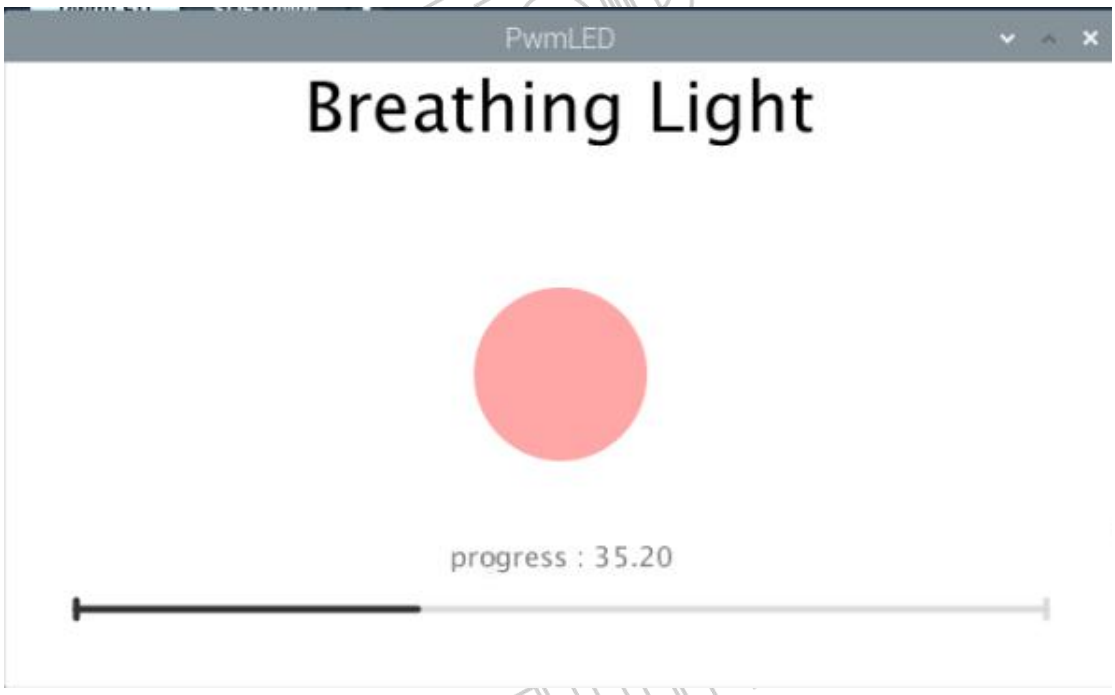


Code

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

1. Enter the processing code / Java / 4.PWMLED / PwmLED / PwmLED.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software 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 pattern shows the percentage of completion, and clicking on interface of the window with mouse can change the progress. As shown below:



The following is the code:

```
import processing.io.*;

int ledPin = 18;
int borderSize = 40;
float t = 0.0;
float tStep = 0.004;
SOFTPWM p = new SOFTPWM(ledPin, 10, 100);
void setup() {
```

```
size(640, 360);
  strokeWeight(4);
}

void draw() {
  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));
  background(255);
  titleAndSiteInfo();

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

  pushMatrix();
  translate(borderSize, height - 45);
  int barLength = width - 2*borderSize;

  barBgStyle();
  line(0, 0, barLength, 0);
  line(barLength, -5, barLength, 5);

  barStyle();
  line(0, -5, 0, 5);
  line(0, 0, t*barLength, 0);

  barLabelStyle();
  text("progress : "+nf(t*100,2,2),barLength/2,-25);
  popMatrix();
}

void titleAndSiteInfo() {
  fill(0);
  textAlign(CENTER);
  textSize(40);
  text("Breathing Light", width / 2, 40);
  textSize(16);
```

```
text("www.keeyees.com", width / 2, height - 20);
}
void barBgStyle() {
  stroke(220);
  noFill();
}

void barStyle() {
  stroke(50);
  noFill();
}

void barLabelStyle() {
  noStroke();
  fill(120);
}
```

Code Interpretation

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

First, use SOFTPWM 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 Self-acceleration rate.

```
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;
  }
```

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.

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

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 LED with different brightness.

```
barBgStyle();
line(0, 0, barLength, 0);
line(barLength, -5, barLength, 5);

barStyle();
line(0, -5, 0, 5);
line(0, 0, t*barLength, 0);

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

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

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

The file contains some information about the SOFTPWM.

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

It’s 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 SOFTPWM(int iPin, int dc, int pwmRange):

Set PMW duty cycle.

public void softPwmStop()

Stop outputting PMW.

Lesson 5 RGBLED

Overview

In this lesson you will learn how to use RGB LED, make a Colorful LED. That's to say, use processing to control the color of RGB LED.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

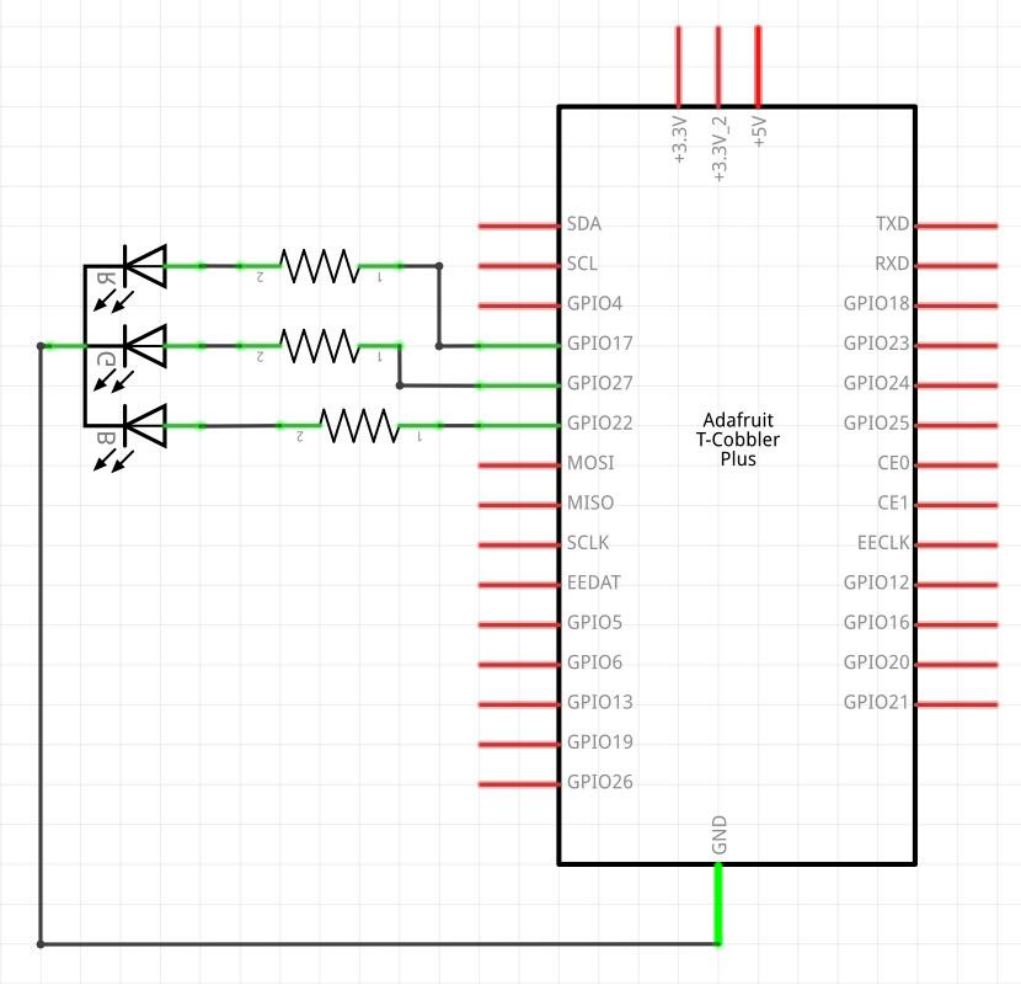
1 x Breadboard

1 x RGBLED

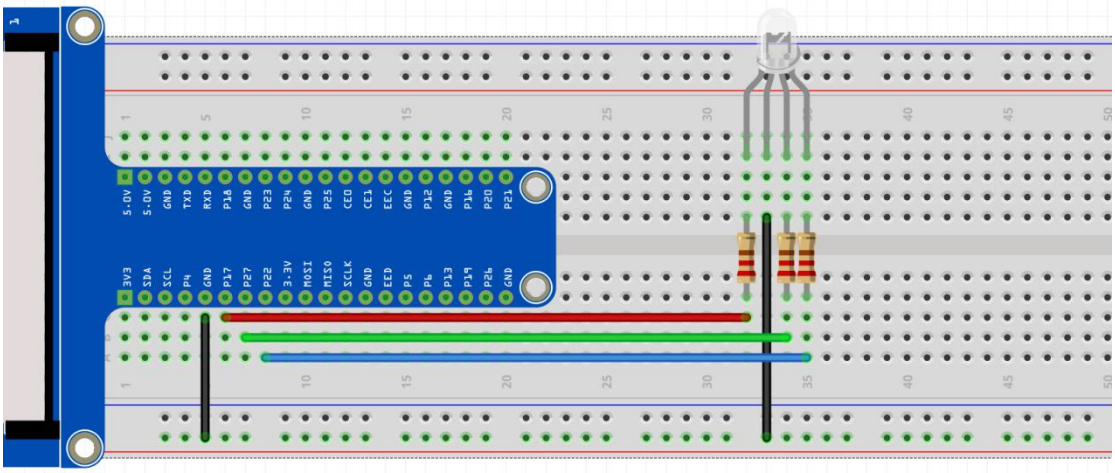
3 x 220 Ohm Resistors

Some Jumper Wires

Connection diagram



Wiring Diagram

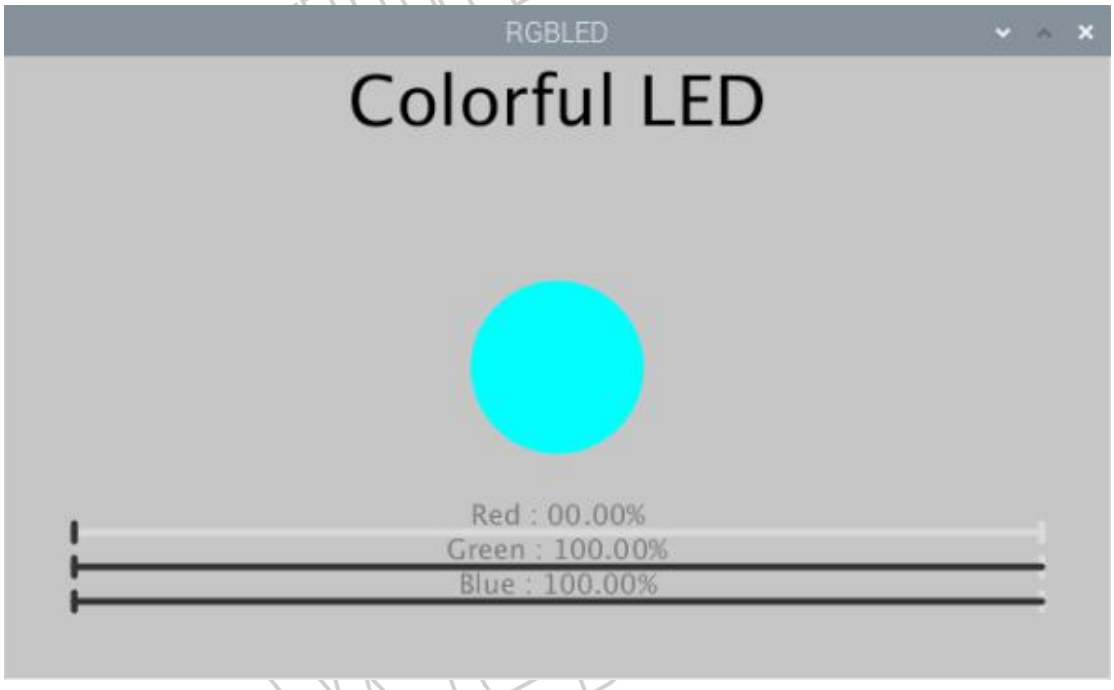


Code

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

1. Enter the processing code / Java / 3.Waterlight / Waterlight / Waterlight.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

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



This project contains a lot of code files, and the core code is contained in the file “RGB LED”. The other files only contain some custom contents.

The following is the code:

```
import processing.io.*;

int bluePin = 17;      //blue Pin
int greenPin = 27;     //green Pin
int redPin = 22;       //red Pin
int borderSize = 40;   //picture border size

SOFTPWM pRed = new SOFTPWM(redPin, 100, 100);
SOFTPWM pGreen = new SOFTPWM(greenPin, 100, 100);
SOFTPWM pBlue = new SOFTPWM(bluePin, 100, 100);

ProgressBar rBar, gBar, bBar;
boolean rMouse = true, gMouse = true, bMouse = true;
void setup() {
   size(640, 360);    //display window size
   strokeWeight(4);    //stroke Weight
   int barLength = width - 2*borderSize;
   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);
   titleAndSiteInfo();
```

```
    fill(rBar.progress*255, gBar.progress*255, bBar.progress*255);

ellipse(width/2, height/2, 100, 100);
    rBar.create();
    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;
    } 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));
        rBar.setProgress(t);
    } else if (gMouse) {
        pGreen.softPwmWrite((int)(100-t*100));
        gBar.setProgress(t);
    } else if (bMouse) {
        pBlue.softPwmWrite((int)(100-t*100));
        bBar.setProgress(t);
    }
}

void titleAndSiteInfo() {
    fill(0);
    textAlign(CENTER);
    textSize(40);               //set text size
    text("Colorful LED", width / 2, 40);       //title
    textSize(16);
```

```
}
```

Code Interpretation

```
SOFTPWM pRed = new SOFTPWM(redPin, 100, 100);
SOFTPWM pGreen = new SOFTPWM(greenPin, 100, 100);
SOFTPWM pBlue = new SOFTPWM(bluePin, 100, 100);

ProgressBar rBar, gBar, bBar;
```

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

```
void setup() {
  size(640, 360);   //display window size
  strokeWeight(4);   //stroke Weight
  int barLength = width - 2*borderSize;
  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");
}
```

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

```
void draw() {
  background(200);
  titleAndSiteInfo();

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

  rBar.create();
  gBar.create();
  bBar.create();
}
```

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 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));
      rBar.setProgress(t);
   } else if (gMouse) {
      pGreen.softPwmWrite((int)(100-t*100));
      gBar.setProgress(t);
   } else if (bMouse) {
      pBlue.softPwmWrite((int)(100-t*100));
      bBar.setProgress(t);
   }
}
```

System function mousePressed(), mouseReleased() and mouseDragged() are used to determine whether the mouse drag the progress bar or not, 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 ().

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)

Construction function 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.

Lesson 6 ActiveBuzzer

Overview

In this course, you will learn how to use active buzzer.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

1 x Breadboard

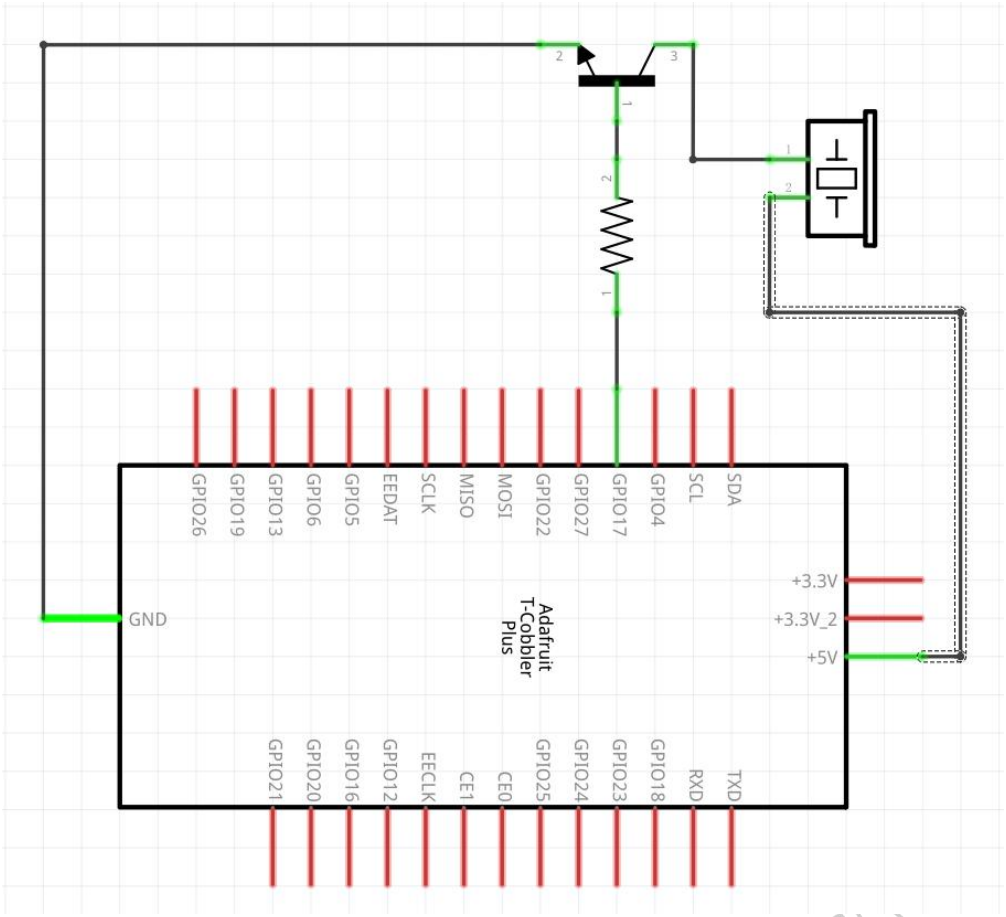
1 x Active Buzzer

1 x 1K Ohm Resistor

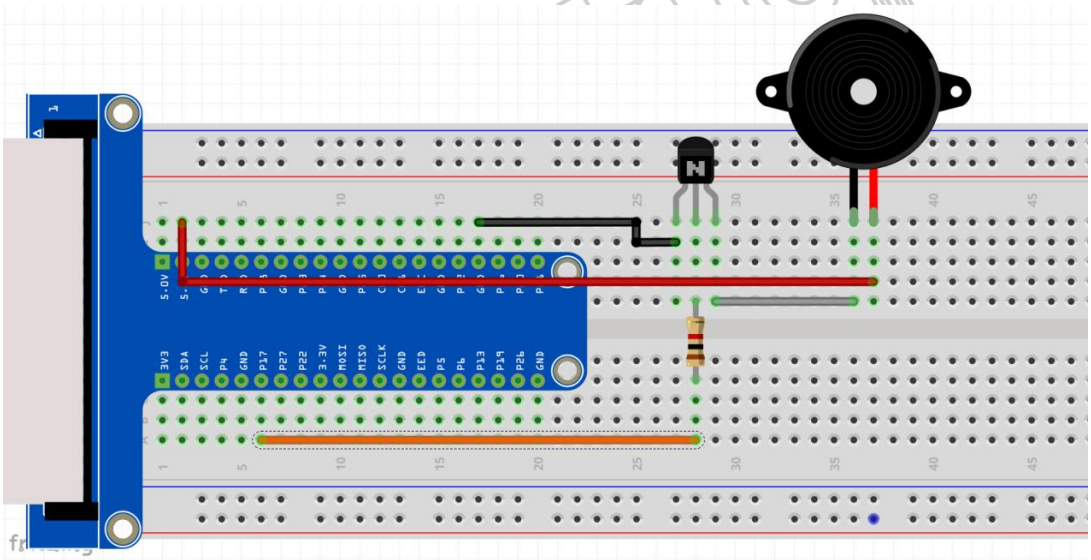
1 x S8050 Transistor

Some Jumper Wires

Connection Diagram



Wiring Diagram



Code

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

1. Enter the processing code / Java / 6.Buzzer / Buzzer / Buzzer.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software 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, and then Active Buzzer stops sounding and arc graphics disappear.

The following is the 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() {
  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);


}
```

Code Interpretation

Code in this project is based on similar logic with the previous lesson 2 "MouseLED". And the difference is that this project needs to draw the buzzer pattern and arc graphics after the buzzer sounding.

Lesson 7 PCF8591

Overview

In this lesson you will learn how to use AD/DA PCF8591 module.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

1 x Breadboard

1 x 220 Ohm Resistor

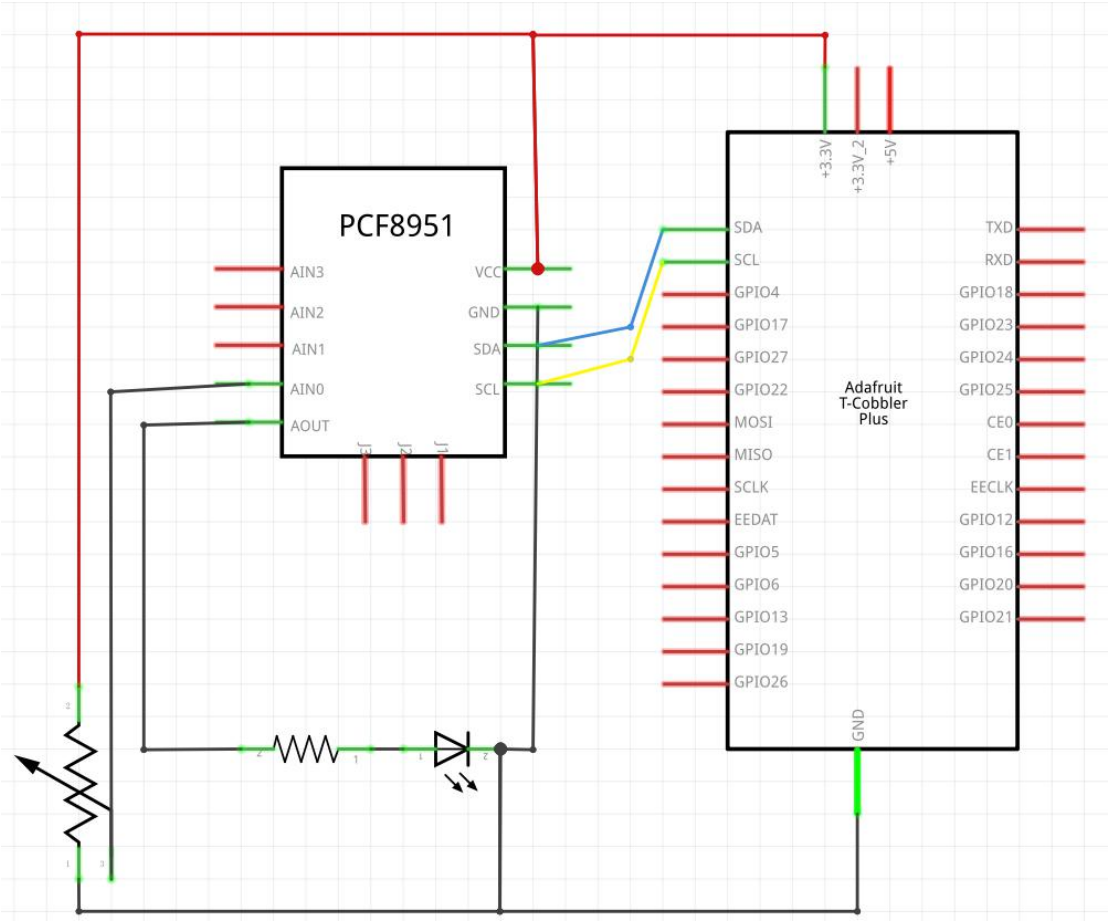
1 x PCF8591 Module

1 x Potentiometer

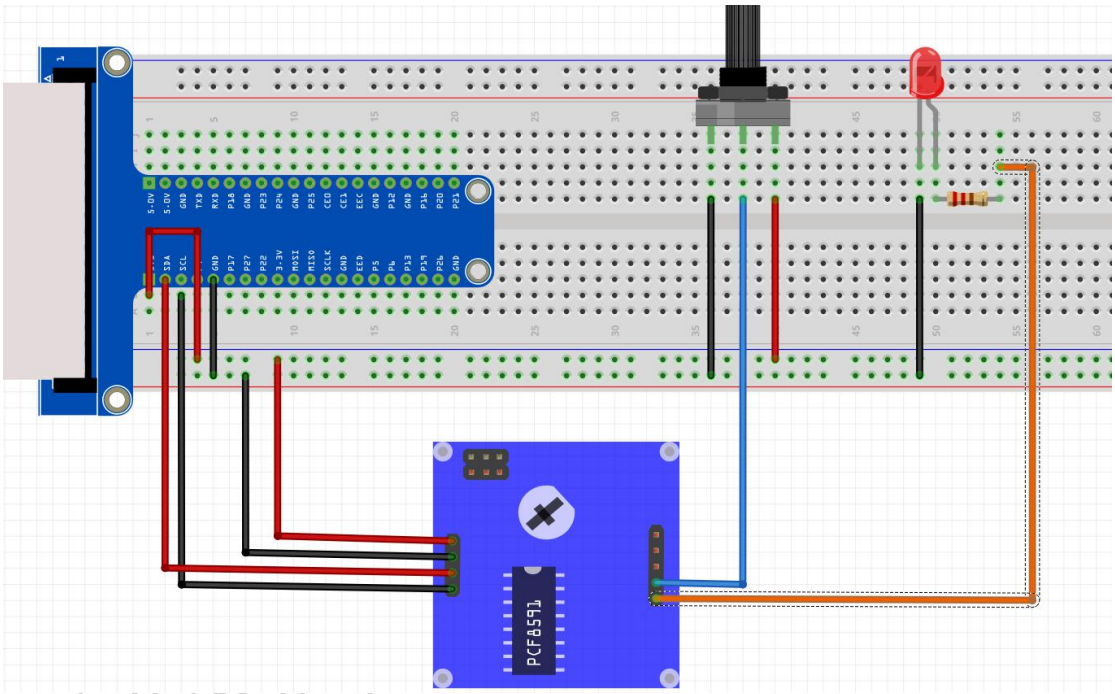
1 x LED

Some Jumper Wires

Connection Diagram



Wiring Diagram

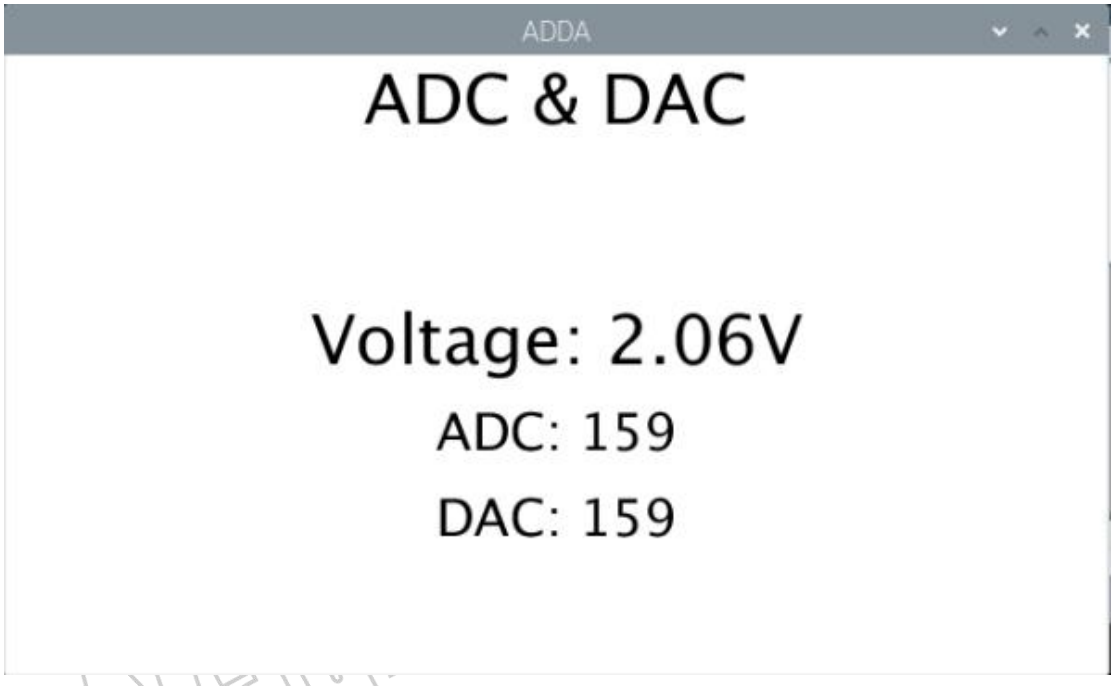


Code

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

1. Enter the processing code / Java / 7.ADDA / ADDA / ADDA.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After the program is executed, Display Window shows the voltage value of the potentiometer, the ADC and DAC value. And ADC value and DAC value are the same. Rotate the potentiometer to change the voltage output. Because the turn-on voltage for red LED is about 1.6V, when the voltage is less than 1.6V, the LED is in off state. When the voltage is greater than 1.6V, the LED is turned on. Continue to increase the DAC value, and then the LED brightness will increase accordingly.



The following is the code:

```
import processing.io.*;
PCF8591 pcf = new PCF8591(0x48);
void setup() {
  size(640, 360);
}
void draw() {
  int adc = pcf.analogRead(0);       //Read the ADC value of channel 0
  float volt = adc*3.3/255.0;        //calculate the voltage
  pcf.analogWrite(adc);              //Write the DAC
  background(255);
  titleAndSiteInfo();

  fill(0);
  textAlign(CENTER);       //set the text centered
  textSize(30);
  text("ADC: "+nf(adc, 3, 0), width / 2, height/2+50);
  textSize(30);
  text("DAC: "+nf(adc, 3, 0), width / 2, height/2+100);
  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 & DAC", width / 2, 40);       //title
  textSize(16);


}
```

Code Interpretation

```
int adc = pcf.analogRead(0);
float volt = adc*3.3/255.0;
pcf.analogWrite(adc);
```

The project code mainly use function analogRead() and analogWrite() to read ADC and write DAC.

About class PCF8591：

class PCF8591

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

public PCF8591(int addr)

Construction Function, 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.

Lesson 8 ADDA&LED

Overview

In this lesson, you will learn how to combine ADC and PWM to control the brightness of LED.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

1 x Breadboard

1 x 220 Ohm Resistor

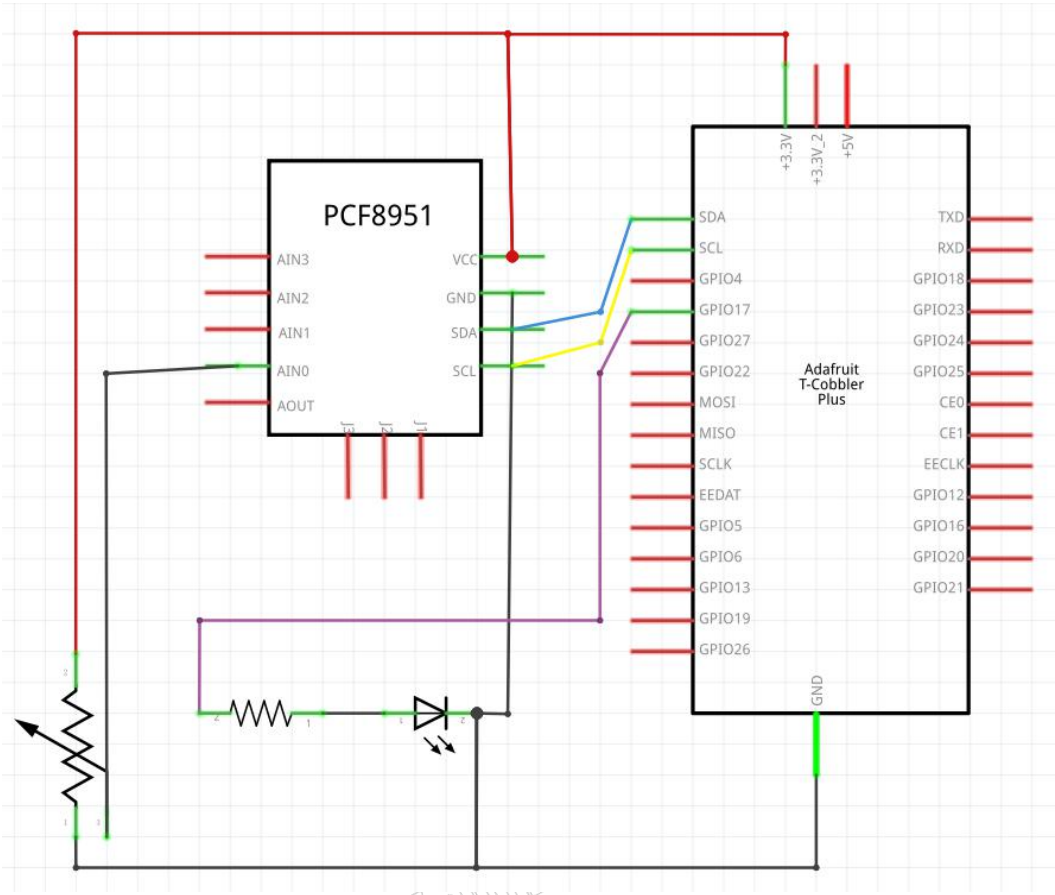
1 x PCF8591 Module

1 x Potentiometer

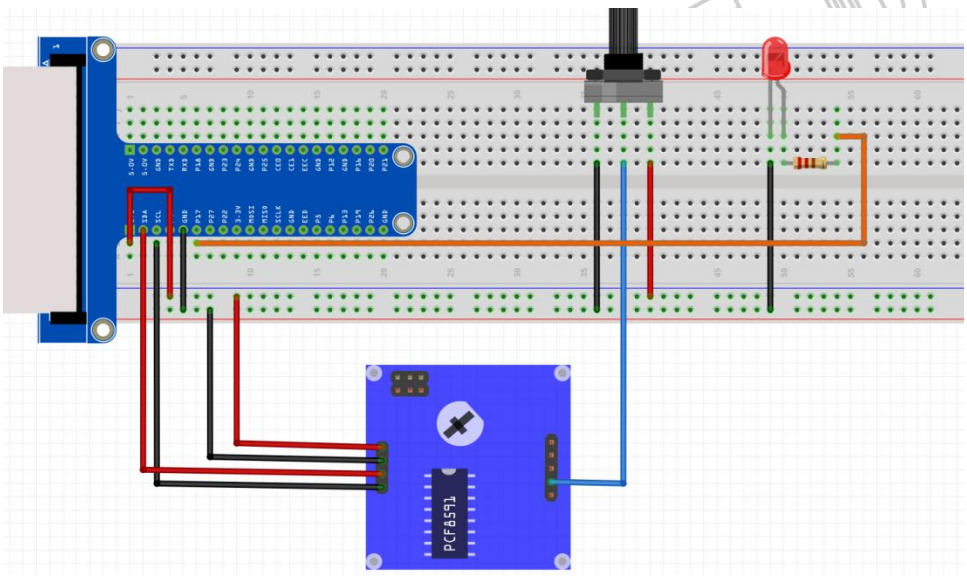
1 x LED

Some Jumper Wires

Connection Diagram



Wiring Diagram

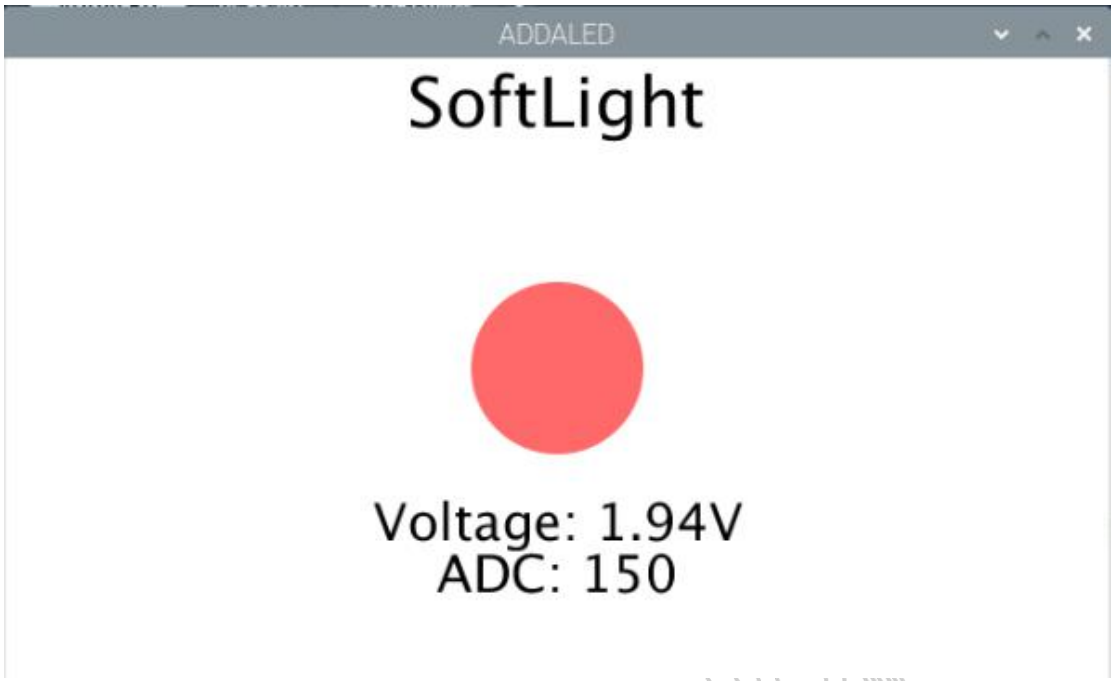


Code

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

1. Enter the processing code / Java / 8.ADDALED / ADDALED / ADDALED.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software 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.



The following is the code:

```
import processing.io.*;

int ledPin = 17;
PCF8591 pcf = new PCF8591(0x48);
SOFTPWM p = new SOFTPWM(ledPin, 0, 100);
void setup() {
   size(640, 360);
}
void draw() {
   int adc = pcf.analogRead(0);        //Read the ADC value of channel 0
```

```
float volt = adc*3.3/255.0;        //calculate the voltage
    float dt = adc/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);
    textAlign(CENTER);        //set the text centered
    textSize(30);
    text("ADC: "+nf(adc, 3, 0), width / 2, height/2+130);
    text("Voltage: "+nf(volt, 0, 2)+"V", width / 2, height/2+100);
}
void titleAndSiteInfo() {
    fill(0);
    textAlign(CENTER);        //set the text centered
    textSize(40);             //set text size
    text("SoftLight", width / 2, 40);
    textSize(16);

}
```

Code Interpretation

```
int adc = pcf.analogRead(0);
float volt = adc*3.3/255.0;
float dt = adc/255.0;
p.softPwmWrite((int)(dt*100));
```

In this project code, get the ADC value of the potentiometer, and 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.

Lesson 9 Photoresistance

Overview

In this course, you will learn how to use Photo resistor, used to intense ambient light, to make a Night Lamp. When the ambient light get dark, LED brightness will be enhanced automatically. Conversely, the LED brightness will be weakened automatically.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

1 x Breadboard

1 x 10 K Resistors

1 x 220 Ohm Resistor

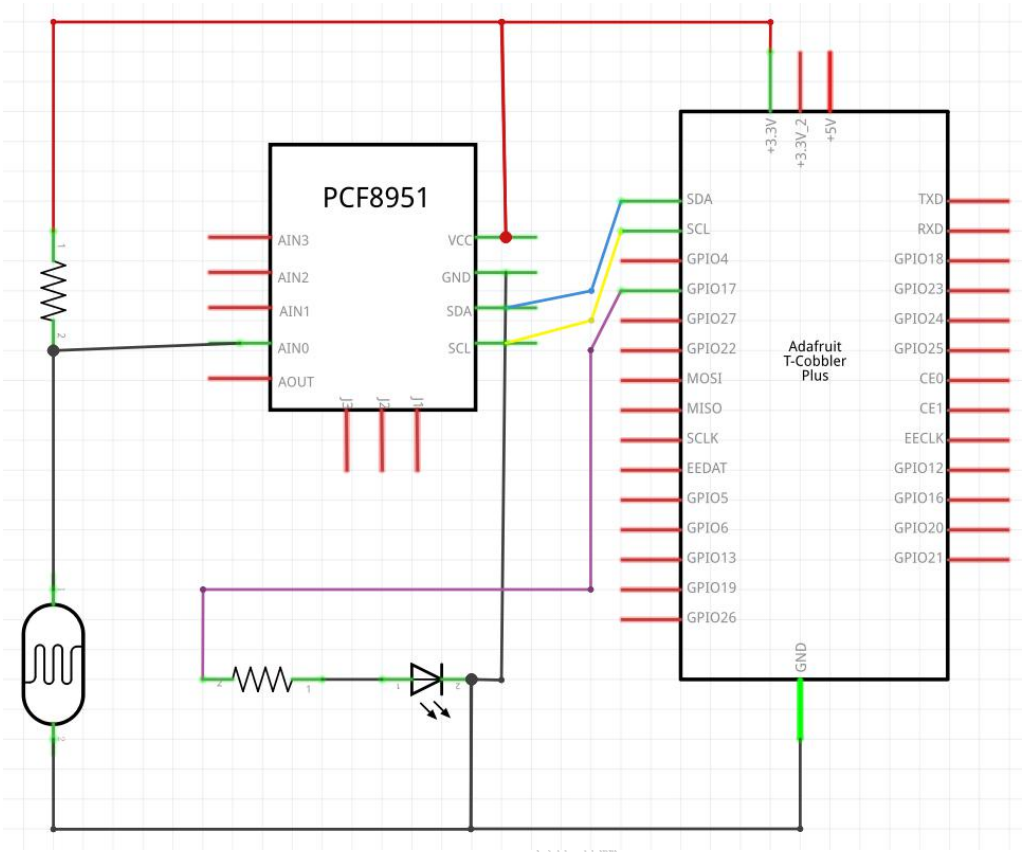
1 x PCF8591 Module

1 x Photo resistor

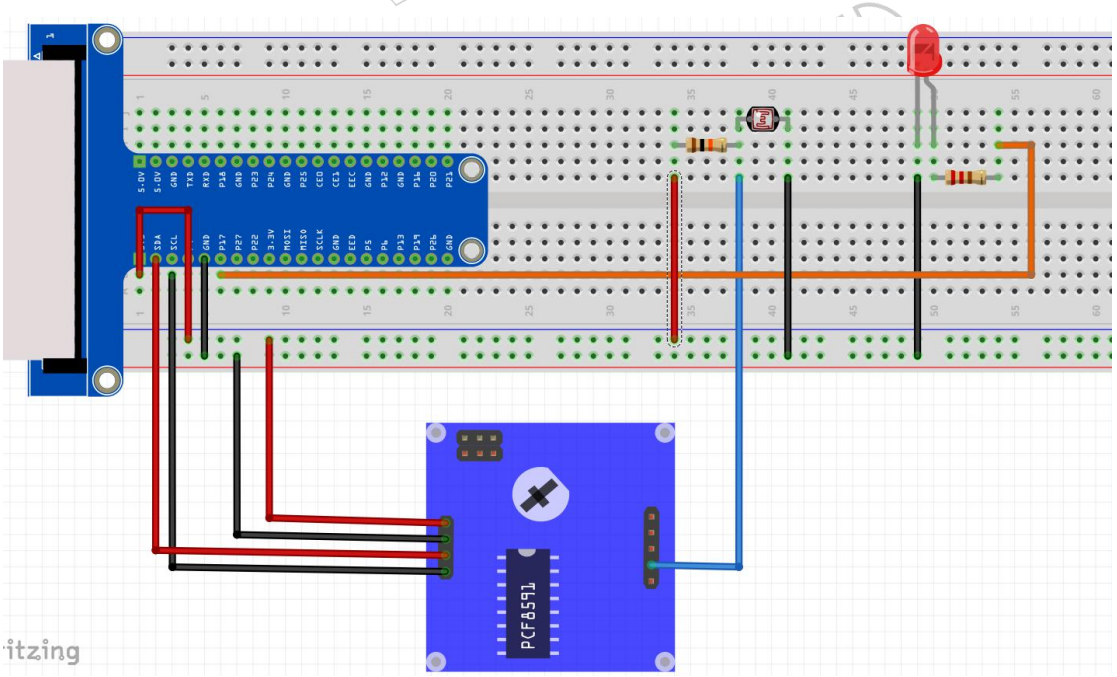
1 x LED

Some Jumper Wires

Connection diagram



Wiring diagram

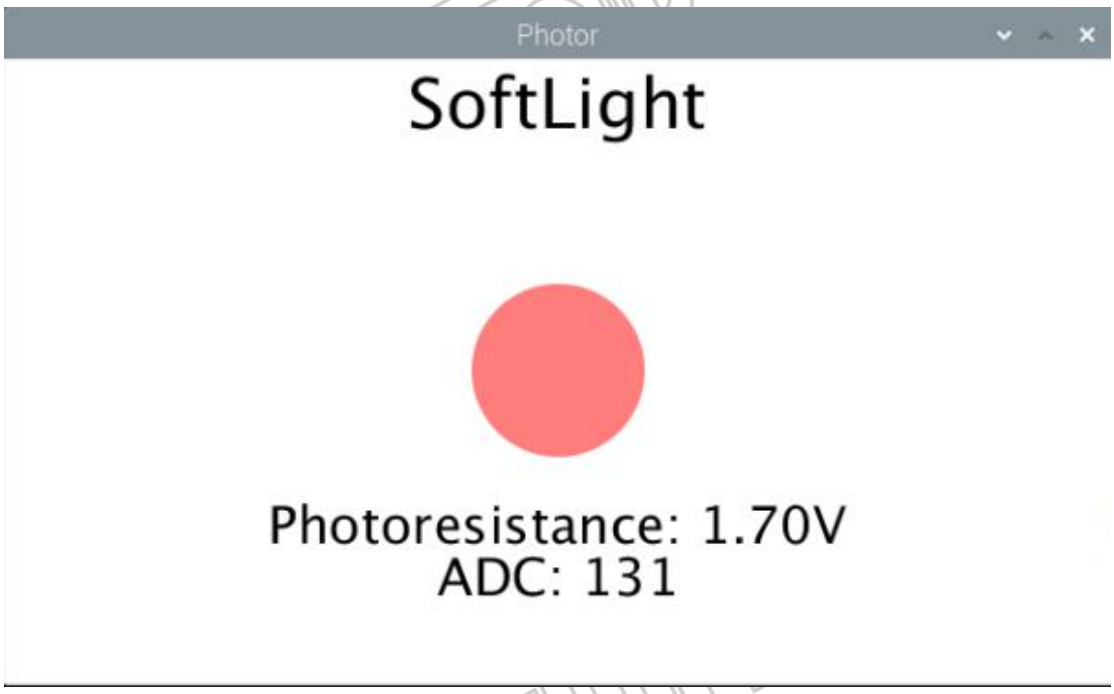


Code

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

1. Enter the processing code / Java / 9.Photoresistance / Photor / Photor.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After the program is executed, the Display Window will show the voltage value of the photoresistor, the ADC value and a LED pattern. Illuminate the photoresistor with light to change the voltage value and the brightness of the LED.



The following is the code:

```
import processing.io.*;

int ledPin = 17;
PCF8591 pcf = new PCF8591(0x48);
SOFTPWM p = new SOFTPWM(ledPin, 0, 100);
void setup() {
   size(640, 360);
}
void draw() {
```

```
    int adc = pcf.analogRead(0);        //Read the ADC value of channel 0

float volt = adc*3.3/255.0;         //calculate the voltage
    float dt = adc/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);
    textAlign(CENTER);        //set the text centered
    textSize(30);
    text("ADC: "+nf(adc, 3, 0), width / 2, height/2+130);
    text("Photoresistance: "+nf(volt, 0, 2)+"V", width / 2, height/2+100);
}
void titleAndSiteInfo() {
    fill(0);
    textAlign(CENTER);        //set the text centered
    textSize(40);              //set text size
    text("SoftLight", width / 2, 40);
    textSize(16);

}
```

Code Interpretation

```
int adc = pcf.analogRead(0);
float volt = adc*3.3/255.0;
float dt = adc/255.0;
p.softPwmWrite((int)(dt*100));
```

The project code is the same as the "Lesson 8 ADDA & LED" code. The only difference is that the input signal of the "AIN0" pin of the "PCF8591" is changed from a potentiometer to a combination of a photo resistor and a resistor.

Lesson 10 Thermistor

Overview

In this lesson, we will learn how to use a thermistor to make a thermometer.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

1 x Breadboard

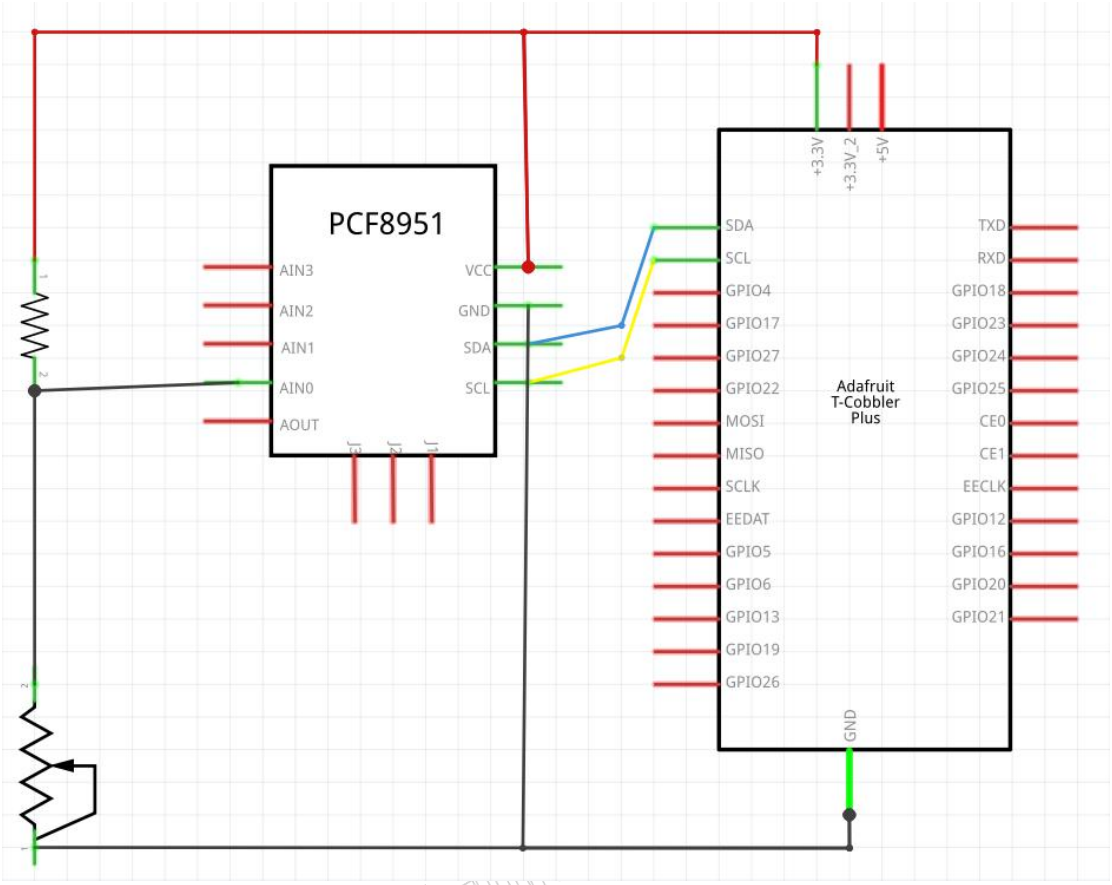
1 x 10K Resistors

1 x PCF8591 Module

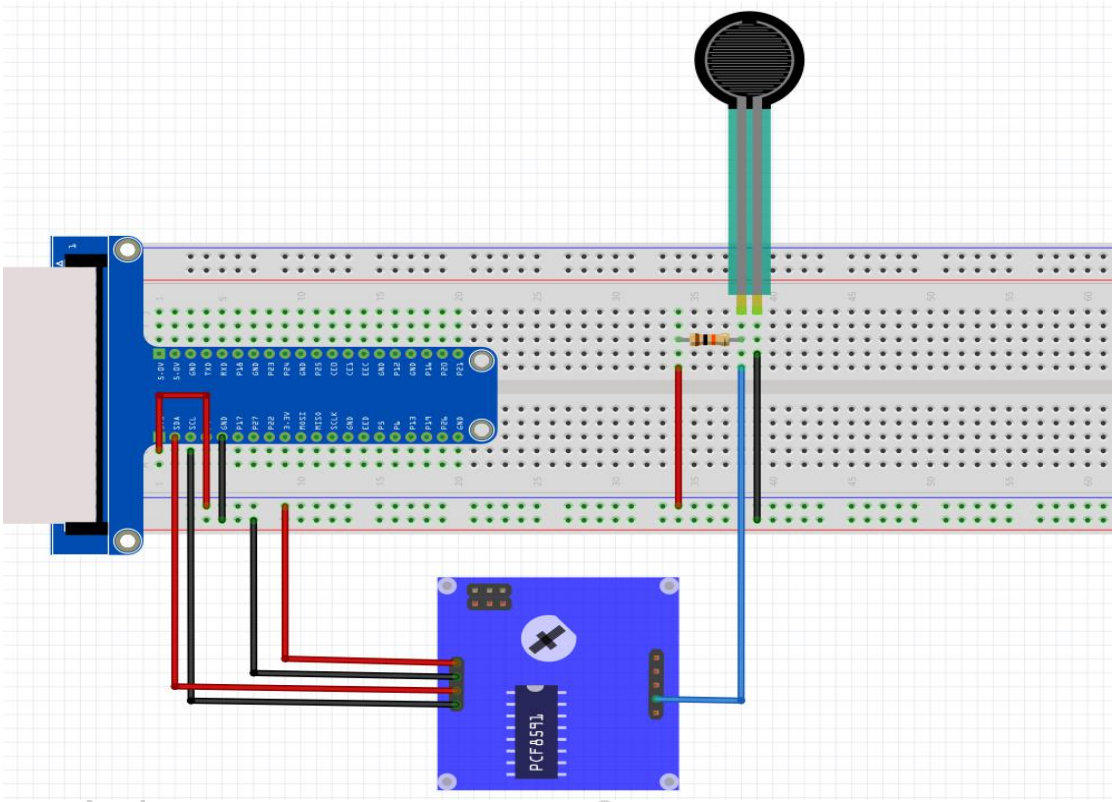
1 x Thermistor

Some Jumper Wires

Connection Diagram



Wiring Diagram

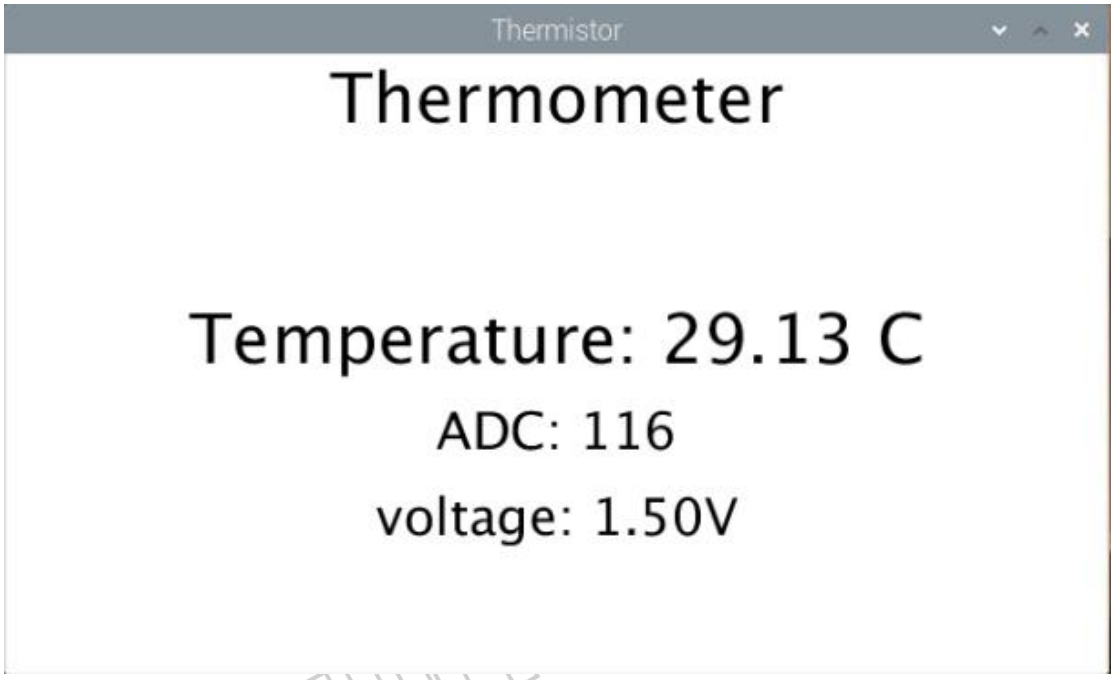


Code

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

1. Enter the processing code / Java / 10.Thermistor / Thermistor / Thermistor.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

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



The following is the code:

```
import processing.io.*;
PCF8591 pcf = new PCF8591(0x48);
void setup() {
  size(640, 360);
}
void draw() {
  int adc = pcf.analogRead(0);
  float volt = adc*3.3/255.0;
  float tempK,tempC,Rt;
  Rt = 10*volt / (3.3-volt);
  tempK = 1/(1/(273.15+25) + log(Rt/10)/3950);
  tempC = tempK - 273.15;

  background(255);
  titleAndSiteInfo();

  fill(0);
  textAlign(CENTER);
  textSize(30);
  text("ADC: "+nf(adc, 0, 0), width / 2, height/2+50);
  textSize(30);

text("voltage: "+nf(volt, 0, 2)+"V", width / 2, height/2+100);
```

```
    textSize(40);
    text("Temperature: "+nf(tempC, 0, 2)+" C", width / 2, height/2);
}
void titleAndSiteInfo() {
    fill(0);
    textAlign(CENTER);
    textSize(40);
    text("Thermometer", width / 2, 40);
    textSize(16);

}
```

Code Interpretation

```
int adc = pcf.analogRead(0);
float volt = adc*3.3/255.0;
float tempK,tempC,Rt;
Rt = 10*volt / (3.3-volt);
tempK = 1/(1/(273.15+25) + log(Rt/10)/3950);
tempC = tempK - 273.15;
```

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.

Lesson 11 74HC595 & LED

Overview

In this lesson you will learn how to use 74HC595 and how to use it to control LEDs.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

1 x Breadboard

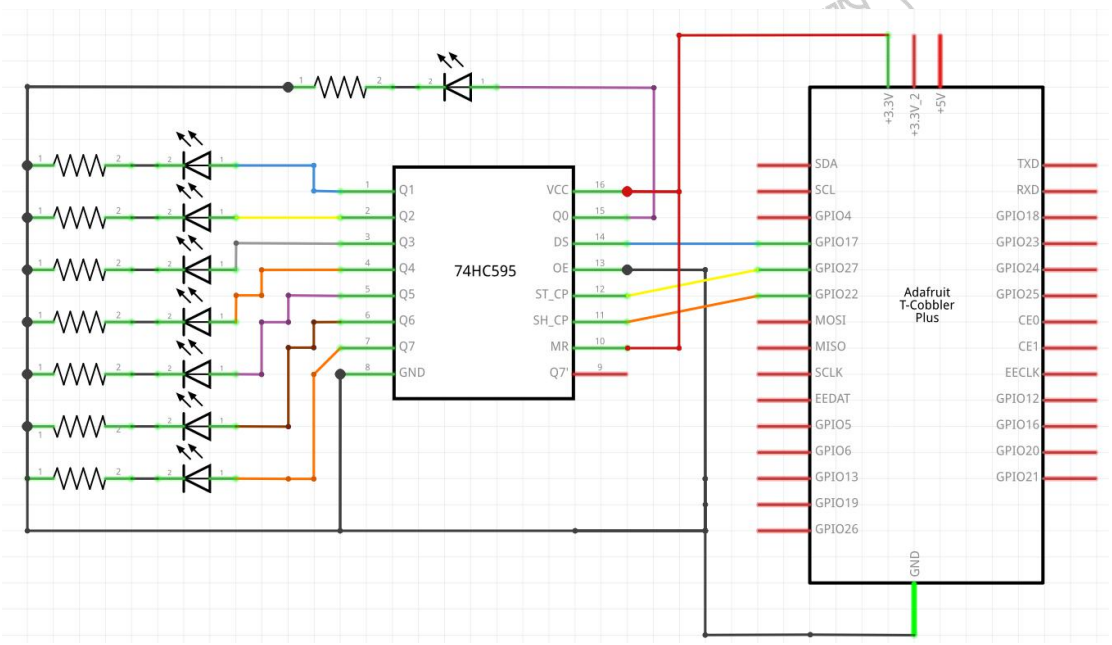
8 x 220 Ohm Resistor

8 x LED

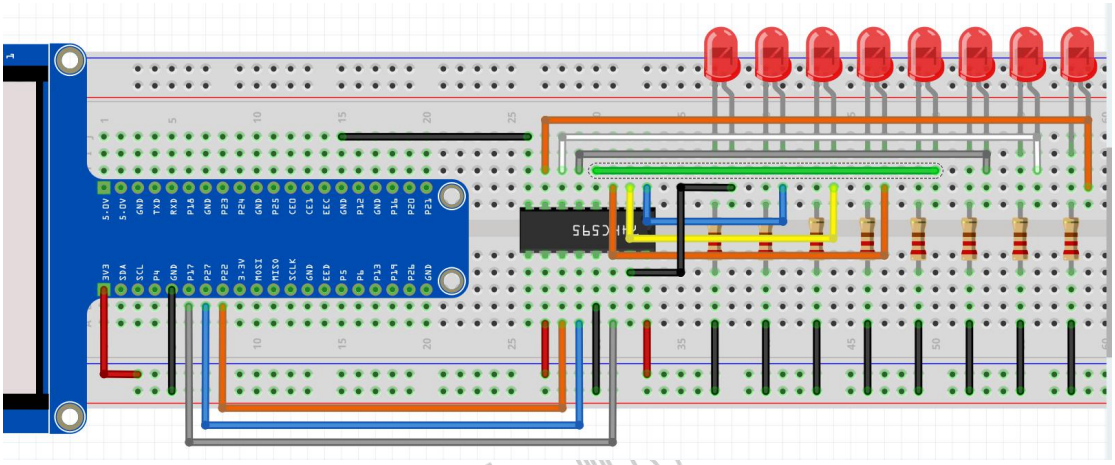
1 x 74HC595 Chip

Some Jumper Wires

Connection diagram



Wiring diagram

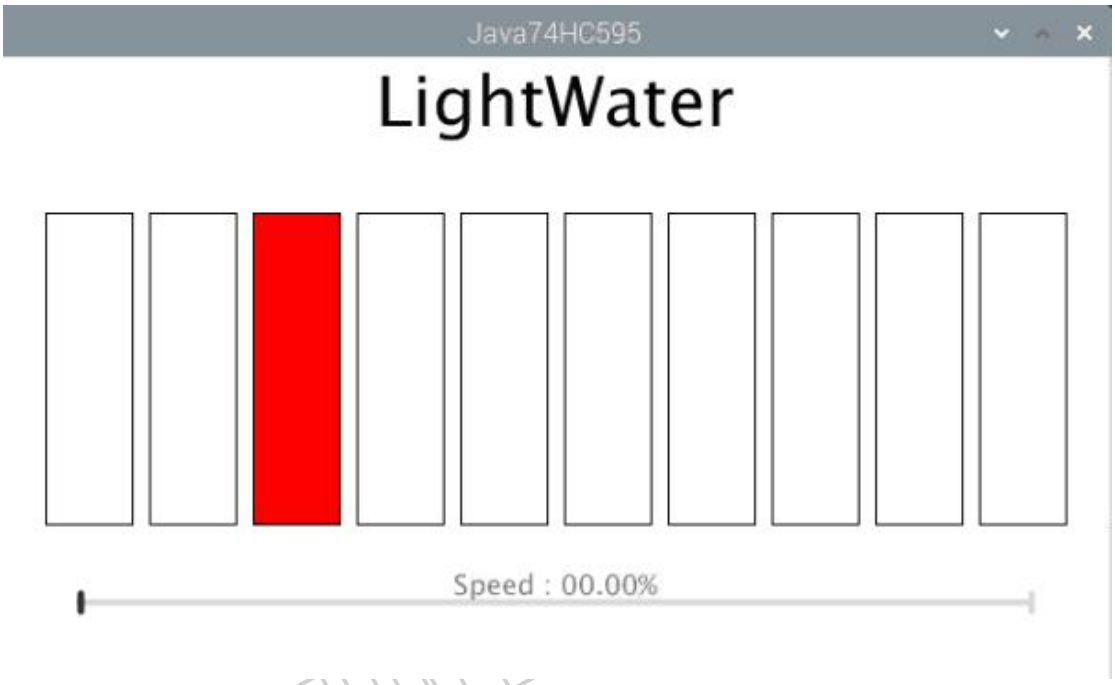


Code

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

1. Enter the processing code / Java / 12.Java74HC595LED / Java74HC595 / Java74HC595.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After executing the program, the display window will display a virtual LEDBar Graph, which will light up at the same rate and manner as the LEDBar Graph. Drag the progress bar to adjust the flow rate of light water. As follows:



The following is the code:

```
import processing.io.*;

int dataPin = 17;
int latchPin = 27;
int clockPin = 22;
final int borderSize = 45;
ProgressBar mBar;
IC74HC595 ic;
boolean mMouse = false;
int leds = 0x01;
int lastMoveTime = 0;
void setup() {
    size(640, 360);
    mBar = new ProgressBar(borderSize, height-borderSize, width-borderSize*2);
    mBar.setTitle("Speed");
    ic = new IC74HC595(dataPin, latchPin, clockPin);
}

void draw() {
    background(255);
    titleAndSiteInfo();
    strokeWeight(4);
    mBar.create();
```

```
    if (millis() - lastMoveTime > 50/(0.05+mBar.progress)) {
      lastMoveTime = millis();
      leds<<=1;
      if (leds == 0x100)
        leds = 0x01;
    }
    ic.write(ic.LSBFIRST, leds);

    stroke(0);
    strokeWeight(1);
    for (int i=0; i<10; i++) {
      if (leds == (1<<i)) {
        fill(255, 0, 0);
      } else {
        fill(255, 255, 255);
      }
      rect(25+60*i, 90, 50, 180);
    }
}

void mousePressed() {
    if ( (mouseY< mBar.y+5) && (mouseY>mBar.y-5) ) {
      mMouse = true;
    }
}
void mouseReleased() {
    mMouse    = false;
}
void mouseDragged() {
    int a = constrain(mouseX, borderSize, width - borderSize);
    float t = map(a, borderSize, width - borderSize, 0.0, 1.0);
    if (mMouse) {
      mBar.setProgress(t);
    }
}
void titleAndSiteInfo() {

fill(0);
    textAlign(CENTER);
    textSize(40);
    text("LightWater", width / 2, 40);
    textSize(16);
```

```
}
```

Code Interpretation

```
int dataPin = 17;
int latchPin = 27;
int clockPin = 22;
final int borderSize = 45;
ProgressBar mBar;
IC74HC595 ic;
boolean mMouse = false;
int leds = 0x01;
int lastMoveTime = 0;
```

First define the GPIO pin connected to 74HC595, the ProgressBar class object, IC74HC595 class object, and some variables.

```
mBar = new ProgressBar(borderSize, height-borderSize, width-borderSize*2);
 mBar.setTitle("Speed");
 ic = new IC74HC595(dataPin, latchPin, clockPin);
```

In the function setup(), instantiate ProgressBar class object and IC74HC595 class object.

```
background(255);
titleAndSiteInfo();
strokeWeight(4);
mBar.create();
```

In the function draw(), set the background, text, and other information and draw the progress bar.

```
if (millis() - lastMoveTime > 50/(0.05+mBar.progress)) {
      lastMoveTime = millis();
      leds<<=1;
      if (leds == 0x100)
         leds = 0x01;
  }

ic.write(ic.LSBFIRST, leds);
```

Then according to the speed of the breathing light, calculate the data “leds” for 74HC595, and make it written to 74HC595, then LEDBar Graph is turned on.

```
stroke(0);
strokeWeight(1);
for (int i=0; i<10; i++) {
      if (leds == (1<<i)) {
         fill(255, 0, 0);
      } else {
         fill(255, 255, 255);
      }
      rect(25+60*i, 90, 50, 180);
 }
```

According to the variable leds, draw the virtual LEDBar Graph in Display Window.

About class IC74HC595：

class IC74HC595

This is a custom class that is used to operate integrated circuit 74HC595.

public IC74HC595(int dPin, int lPin, int cPin)

Construction Function, the parameter for the GPIO pin connected to 74HC595.

public void write(int order,int value)

Used to write data to 74HC595, and the 74HC595 output port will output these data immediately.

Lesson 12 74HC595 & Seven-segment display

Overview

In this lesson, we will learn a new component, Seven-segment display (SSD).

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

1 x Breadboard

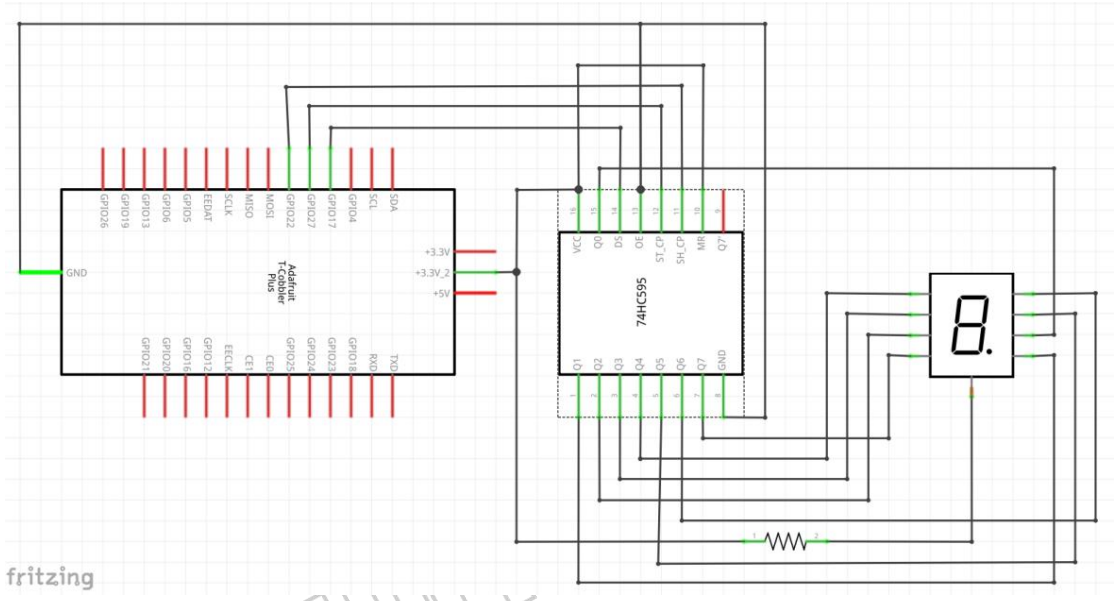
1 x 220 Ohm Resistor

1 x One 7-segment Digital Tube

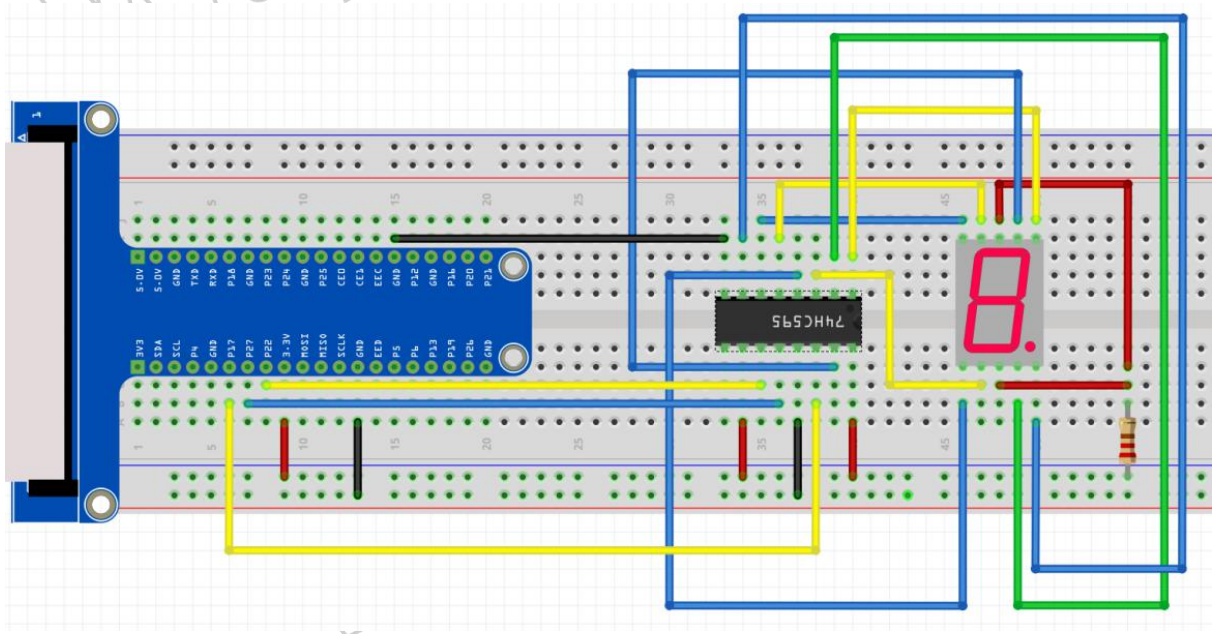
1 x 74HC595 Chip

Some Jumper Wires

Connection Diagram



Wiring Diagram

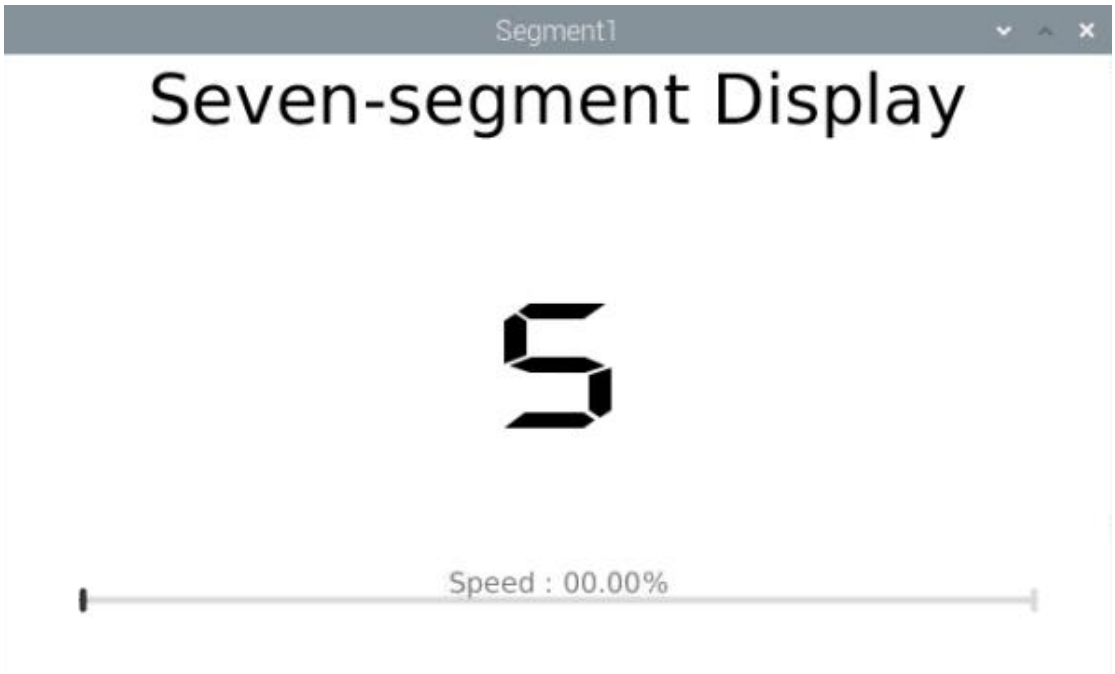


Code

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

1. Enter the processing code / Java / 12.Segment1 / Segment1 / Segment1.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After the program is executed, both Display Window and SSD in the circuit show the same number. And they have the same self-acceleration rate to display number "0-9" constantly. Dragging the progress bar can adjust the self-acceleration rate.



The following is the code:

```
import processing.io.*;

int dataPin = 17;
int latchPin = 27;
int clockPin = 22;
final int borderSize = 45;
ProgressBar mBar;
IC74HC595 ic;
boolean mMouse = false;
int index = 0;
```

```
int lastMoveTime = 0;
final int[] numCode = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};
PFont mFont;

void setup() {
  size(640, 360);
  mBar = new ProgressBar(borderSize, height-borderSize, width-borderSize*2);
  mBar.setTitle("Speed");
  ic = new IC74HC595(dataPin, latchPin, clockPin);
  mFont = loadFont("DigifaceWide-100.vlw");
}

void draw() {
  background(255);
  titleAndSiteInfo();
  strokeWeight(4);
  mBar.create();
  if (millis() - lastMoveTime > 50/(0.05+mBar.progress)) {
    lastMoveTime = millis();
    index++;
    if (index > 9) {
      index = 0;
    }
  }
  ic.write(ic.MSBFIRST, numCode[index]);
  showNum(index);
}
void showNum(int num) {
  fill(0);
  textSize(100);
  textFont(mFont);
  textAlign(CENTER, CENTER);
  text(num, width/2, height/2);
}
void mousePressed() {
  if ( (mouseY< mBar.y+5) && (mouseY>mBar.y-5) ) {
    mMouse = true;
  }
}
void mouseReleased() {
  mMouse    = false;
}
```

```
void mouseDragged() {

int a = constrain(mouseX, borderSize, width - borderSize);
   float t = map(a, borderSize, width - borderSize, 0.0, 1.0);
   if (mMouse) {
      mBar.setProgress(t);
   }
}
void titleAndSiteInfo() {
   fill(0);
   textAlign(CENTER);
   textFont(createFont("", 100));
   textSize(40);
   text("Seven-segment Display", width / 2, 40);
   textSize(16);

}
```

Code Interpretation

```
final int[] numCode = {0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90};
```

The project code is similar to the last chapter. The difference is that in this project the data output by 74HC595 is the fixed coding information of SSD. First, the character "0-9" is defined as code of common anode SSD.

```
if (millis() - lastMoveTime > 50/(0.05+mBar.progress)) {
      lastMoveTime = millis();
      index++;
      if (index > 9) {
         index = 0;
      }
   }
 ic.write(ic.MSBFIRST, numCode[index]);
 showNum(index);
```

In the function draw(), the data is output at a certain speed. At the same time the Display Window outputs the same character.

```
PFont mFont;

mFont = loadFont("DigifaceWide-100.vlw");
```

By creating the font "mFont", we can change the font of the characters in Display Window. The font ".vlw" file is created by clicking the "Create Font" of the menu bar, which is saved in the data folder of current "Segment1".

```
textFont(createFont("", 100));
```

By creating an empty font, you can reset the font to the default font.

For more details about loadFont (), please refer to the official website:

https://processing.org/reference/loadFont_.html

Lesson 13 I2C-LCD1602

Overview

In this lesson you will learn LCD1602 display screen. The current time and date will be displayed on the LCD1602 and Display Window.

Parts Required

1 x Raspberry Pi

1 x GPIO T-type Expansion Board

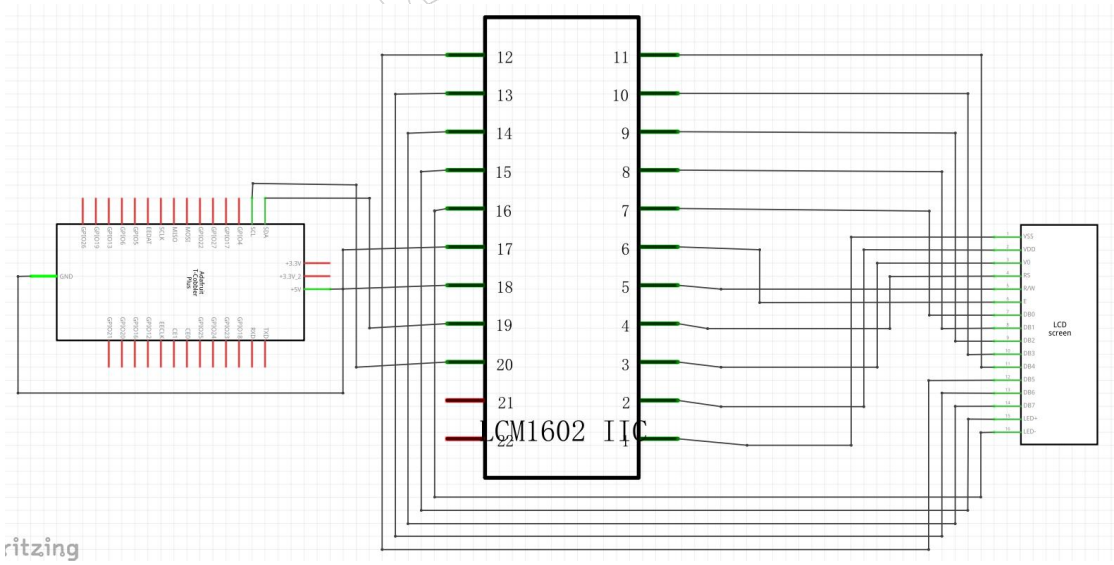
1 x Breadboard

1 x LCD1602

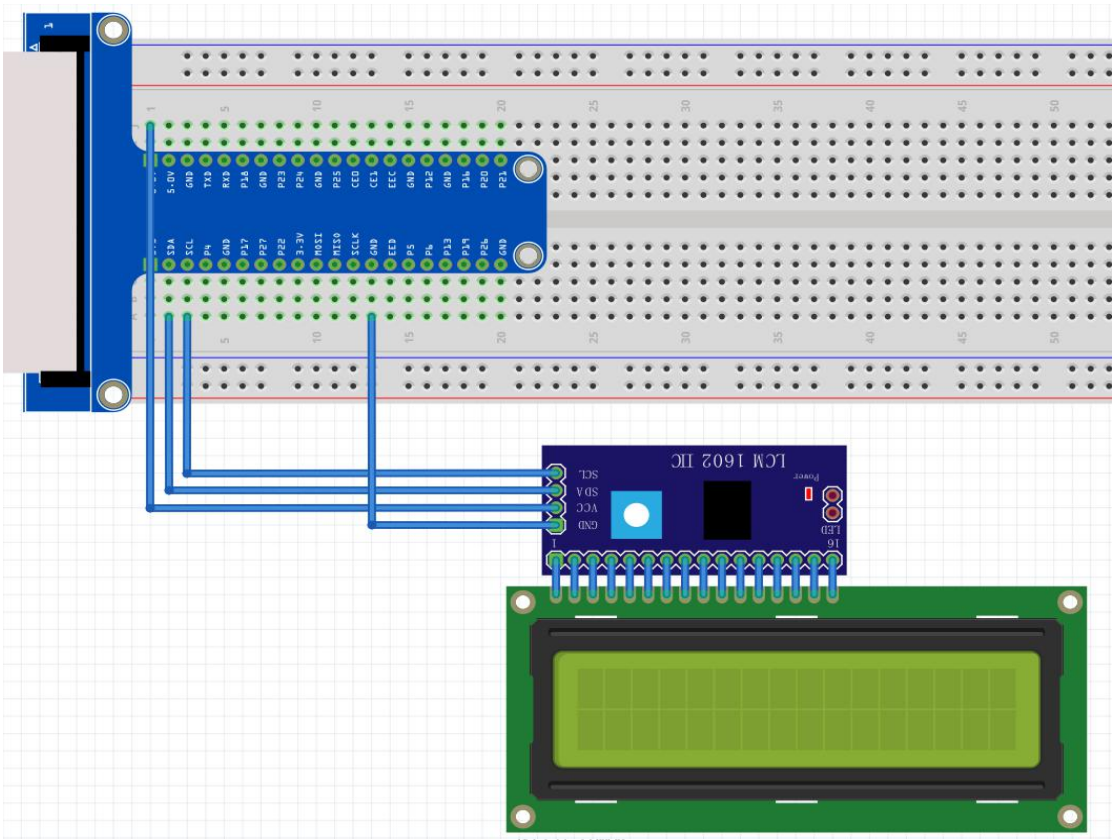
1 x PCF8574

Some Jumper Wires

Connection Diagram



Wiring Diagram



Code

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

1. Enter the processing code / Java / 15.LCD1602 / LCD1602 / LCD1602.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After executing the program, the LCD1602 displays the time and date, and the display window also displays the contents of the LCD1602. As shown below:

The following is the program code:

```
import processing.io.*;
//Create a object of class PCF8574
PCF8574 pcf = new PCF8574(0x27);
LCD_LCD1602 lcd;    //Create a lcd object
String time = "";
String date = "";
void setup() {
   size(640, 360);
   lcd = new LCD_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);
}
```

Code Interpretation

```
PCF8574 pcf = new PCF8574(0x27);
LCD_LCD1602 lcd;    //Create a lcd object
String time = "";
String date = "";
void setup() {
    size(640, 360);
    lcd = new LCD_LCD1602(pcf);
    frameRate(2);       //set display window frame rate for 2 HZ
}
```

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 2Hz.

```
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
}
```

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

About class PCF8574：

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

public PCF8574(int addr)

Construction Function, 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 LCD_LCD:

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

Public LCD_LCD1602(PCF8574 ipcf)

Construction Function, used to create 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.

Lesson 14 Relay & Motor

Overview

In this course, you will learn a kind of special switch module, Relay Module. We will use a push button to control a relay and drive the motor.

Parts Required

1 x Raspberry Pi

1 x GPIO T-type Expansion Board

1 x Breadboard

1 x 1kΩ Resistor

1 x 220Ω Resistor

1 x LED

1 x Button

1 x Relay

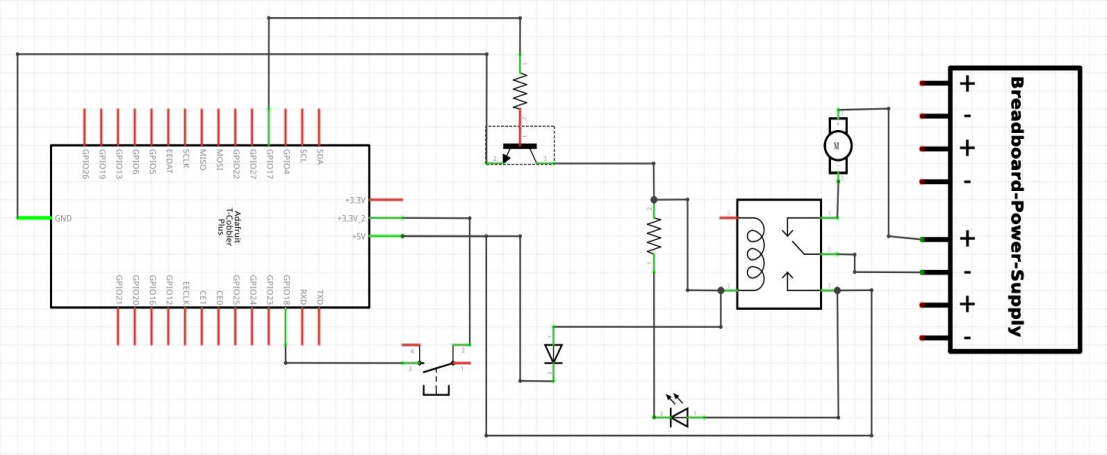
1 x Motor

1 x NPN Transistor

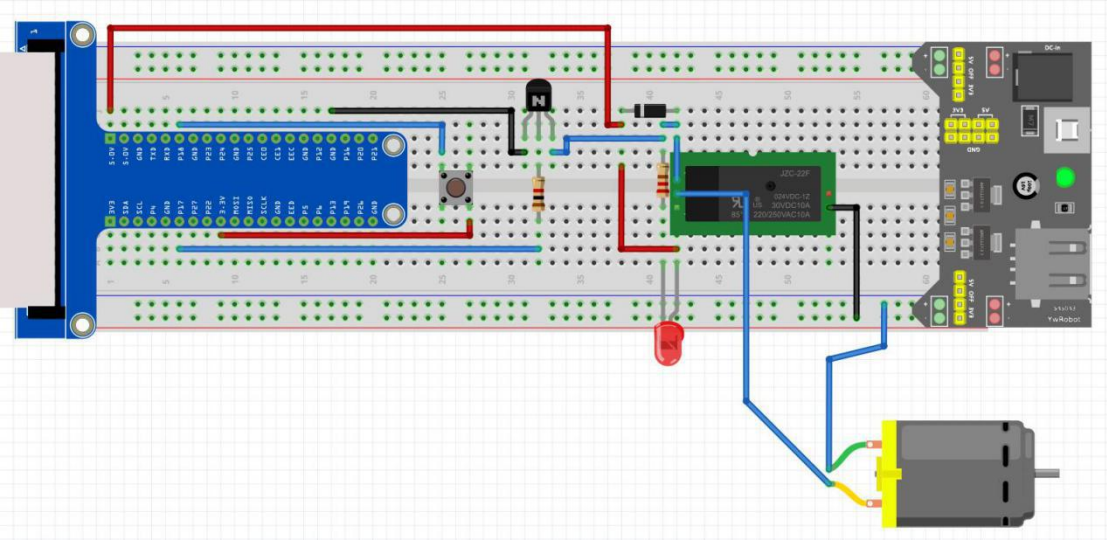
1 x Diode

Some Jumper Wires

Connection diagram



Wiring diagram



Code

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

1. Enter the processing code / Java / 17.Relay / Relay / Relay.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After the program is executed, press the button once, the motor starts to rotate, and the fan on the screen rotates with it, the operation effect is shown in the figure below:

The following is program code:

```
import processing.io.*;

BUTTON btn;
int relayPin = 17; //define relayPin
int buttonPin = 18;
boolean ledState = false; //define ledState
int count=0;
int fign=0;
float rotaSpeed = 0.02 * PI; //virtual fan's rotating speed,
float rotaPosition = 0; //motor position
void setup() {
   size(640, 360);
   GPIO.pinMode(relayPin, GPIO.OUTPUT); //set the ledPin to output mode
   GPIO.pinMode(buttonPin, GPIO.INPUT);
   btn = new BUTTON(90, height - 90, 50, 30); //define the button
   btn.setBgColor(0, 255, 0); //set button color
   btn.setText("OFF"); //set button text
}
void draw() {
    background(255, 255, 255);
   titleAndSiteInfo();

if(GPIO.digitalRead(buttonPin)==GPIO.HIGH && fign==0)
```

```
  {
      delay(20);
      if(GPIO.digitalRead(buttonPin)==GPIO.HIGH)
      {
          count++;
      }
      count=count%2;
      if(count==1)
          ledState=true;
      else
          ledState=false;
      fign=1;
  }
  else if(GPIO.digitalRead(buttonPin)==GPIO.LOW)
  {
      fign=0;
  }
  textAlign(RIGHT, CENTER);
  text("RelayState: ", btn.x, btn.y+btn.h/2);
  btn.create(); //create the button
  if (ledState) {
  GPIO.digitalWrite(relayPin, GPIO.HIGH); //led on
  rotaPosition += rotaSpeed;
  btn.setBgColor(0, 255, 0);
  btn.setText("ON");
  } else {
  GPIO.digitalWrite(relayPin, GPIO.LOW); //led off
  btn.setBgColor(255, 0, 0);
  btn.setText("OFF");
  }
  if (rotaPosition >= 2*PI) {
     rotaPosition = 0;
  }
  drawFan(rotaPosition);
}
void drawFan(float angle) {
  constrain(angle, 0, 2*PI);
  fill(0);
  for (int i=0; i<3; i++) {
  arc(width/2, height/2, 200, 200, 2*i*PI/3+angle, (2*i+0.3)*PI/3+angle, PIE);
  }

fill(0);
```

```
  ellipse(width/2, height/2, 30, 30);
  fill(128);
  ellipse(width/2, height/2, 15, 15);
}
void titleAndSiteInfo() {
  fill(0);
  textAlign(CENTER); //set the text centered
  textSize(40); //set text size
  text("Relay & Motor", width / 2, 40); //title
  textSize(16);
}
```

Code Interpretation

```
void setup() {
   size(640, 360);
   GPIO.pinMode(relayPin, GPIO.OUTPUT); //set the ledPin to output mode
   GPIO.pinMode(buttonPin, GPIO.INPUT);
   btn = new BUTTON(90, height - 90, 50, 30); //define the button
  btn.setBgColor(0, 255, 0); //set button color
   btn.setText("OFF"); //set button text
}
```

In the function setup(), Window Display and virtual button are initialized.

```
void draw() {
     background(255, 255, 255);
   titleAndSiteInfo();
 if(GPIO.digitalRead(buttonPin)==GPIO.HIGH && fign==0)
 {
     delay(20);
     if(GPIO.digitalRead(buttonPin)==GPIO.HIGH)
     {
          count++;
     }
     count=count%2;
     if(count==1)
          ledState=true;
     else
          ledState=false;
     fign=1;
 }
```

```
else if(GPIO.digitalRead(buttonPin)==GPIO.LOW)
  {
      fign=0;
  }
  textAlign(RIGHT, CENTER);
  text("RelayState: ", btn.x, btn.y+btn.h/2);
  btn.create(); //create the button
  if (ledState) {
  GPIO.digitalWrite(relayPin, GPIO.HIGH); //led on
  rotaPosition += rotaSpeed;
  btn.setBgColor(0, 255, 0);
  btn.setText("ON");
  } else {
  GPIO.digitalWrite(relayPin, GPIO.LOW); //led off
  btn.setBgColor(255, 0, 0);
  btn.setText("OFF");
  }
  if (rotaPosition >= 2*PI) {
     rotaPosition = 0;
  }
  drawFan(rotaPosition);

}
```

In the function draw(),Whether the scan button is pressed. If the button is pressed, it is judged whether to open or close the relay, and the state of the relay and the virtual button will be changed. Then draw virtual buttons and fan blades.

Lesson 15 Oscilloscope

Overview

We have used the PCF8591 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 changing process.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

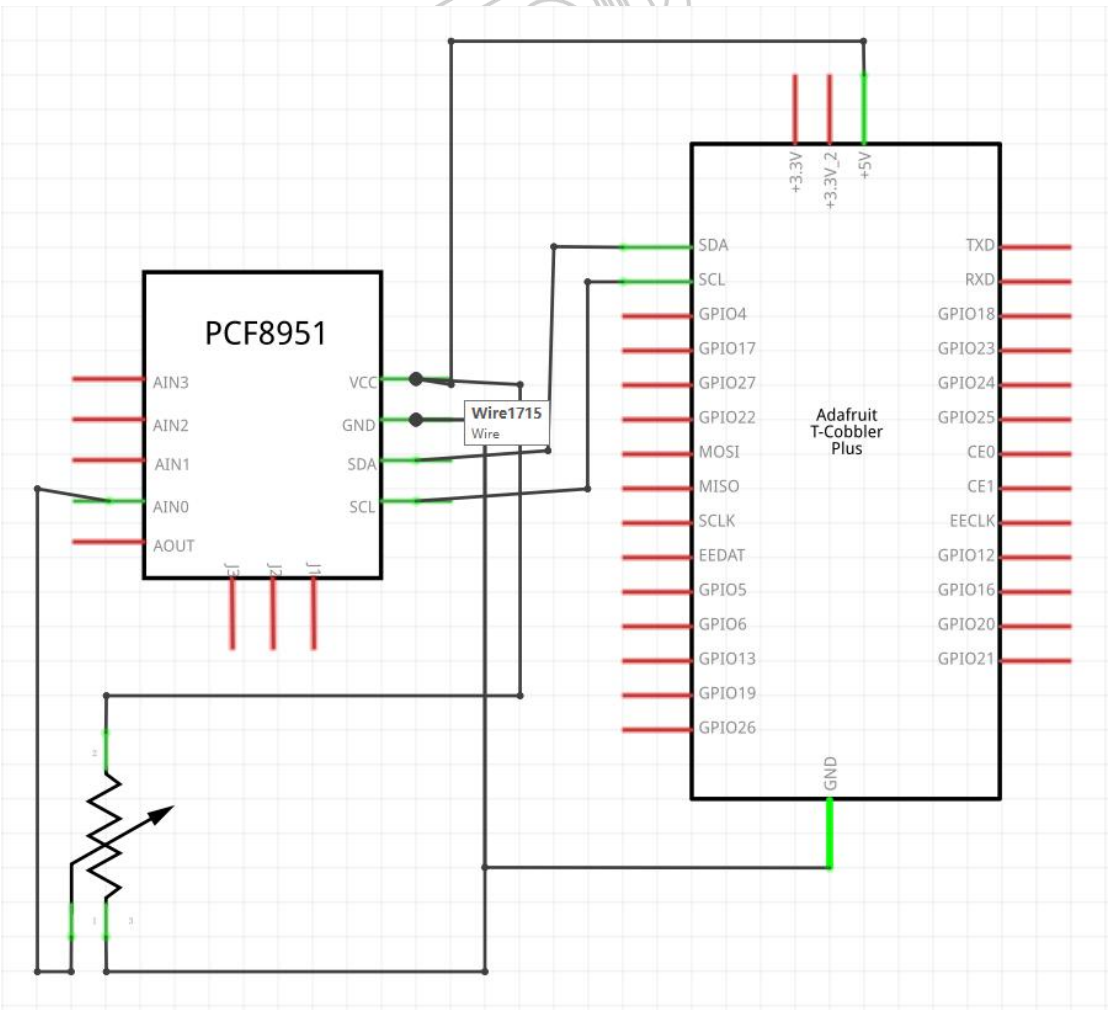
1 x Breadboard

1 x Potentiometer

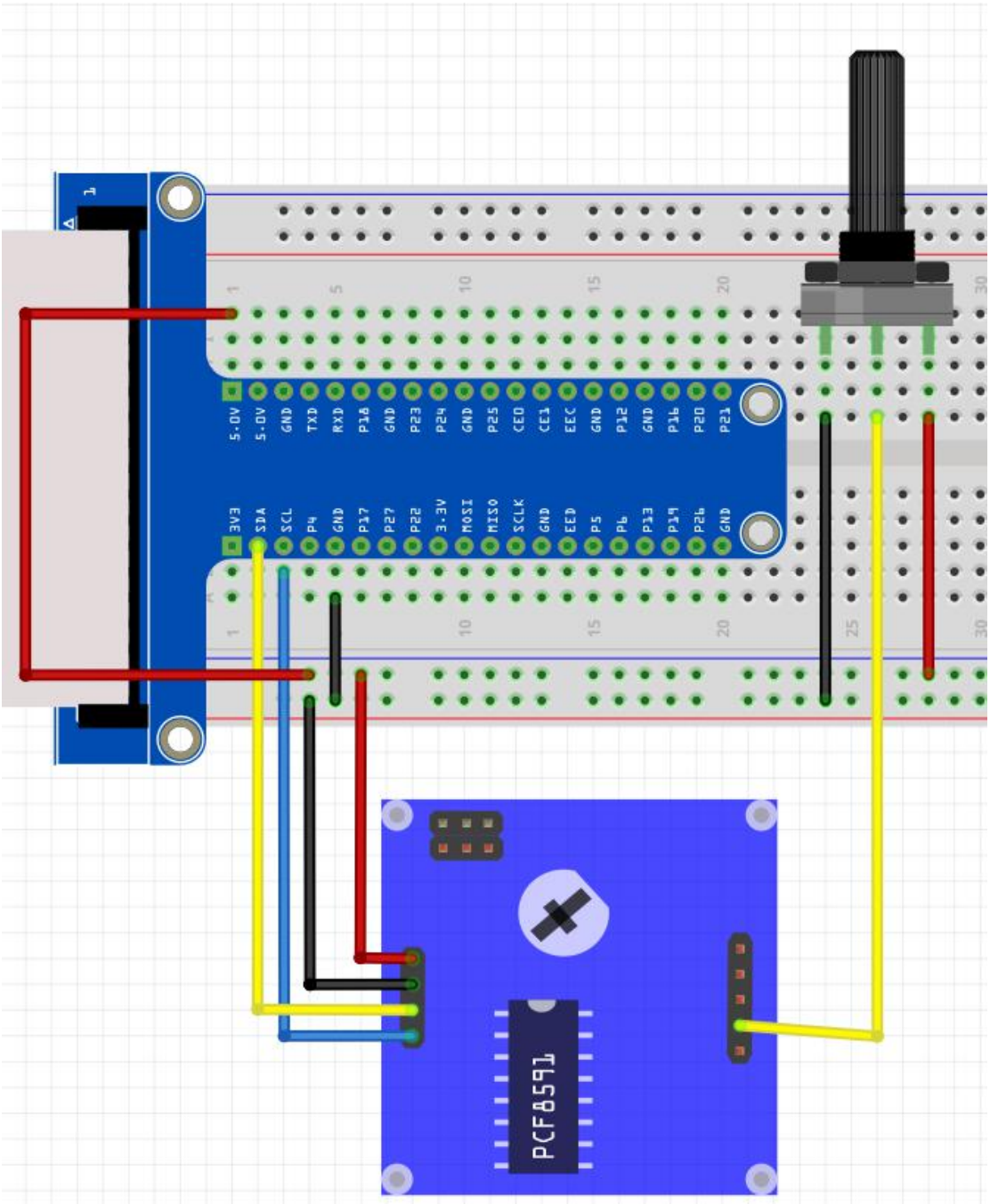
1 x PCF8591 Module

Some Jumper Wires

Connection Diagram



Wiring Diagram

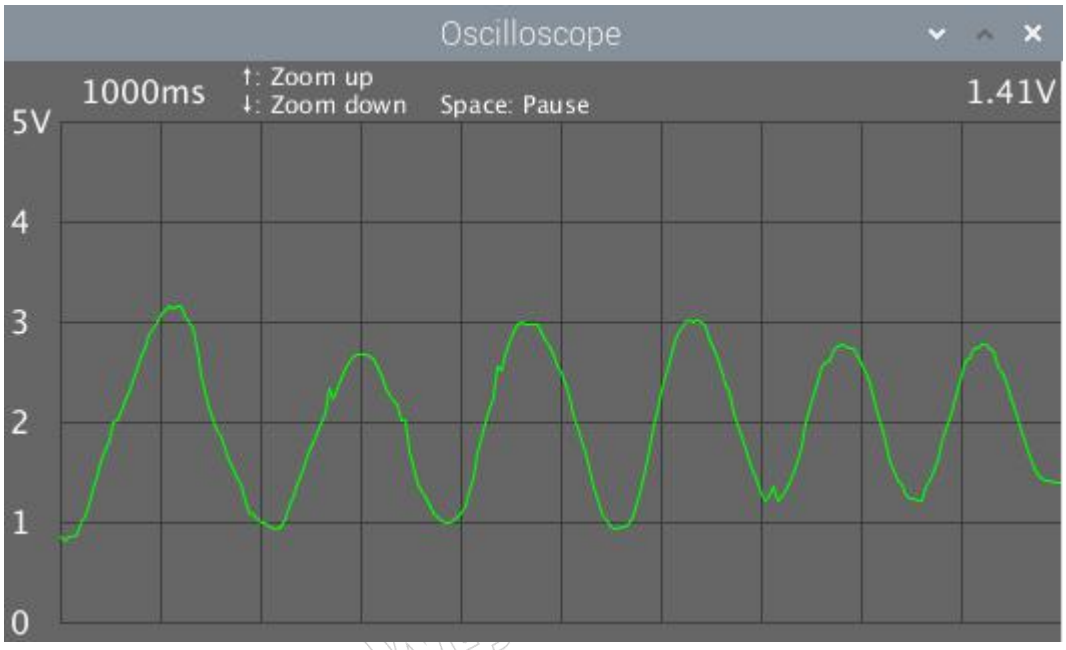

5.0V
5.0V
GND
TXD
RXD
P18
GND
P23
P24
GND
P25
CE0
CE1
EEC
GND
P12
GND
P16
P20
P21
3V3
SDA
SCL
P4
GND
P17
P27
P22
3.3V
MOSI
MISO
SCLK
GND
EED
P5
P6
P13
P19
P26
GND
PCF8591

Code

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

1. Enter the processing code / Java / 20.Oscilloscope / Oscilloscope / Oscilloscope.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After the program is executed, rotate the potentiometer. The waveform of the display window will change as the potentiometer rotates



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 "1.41V" 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.

The following is program code:

```
import processing.io.*;


PCF8591 pcf = new PCF8591(0x48);
int[] analogs;                  // Analog data send from serial device
int analogsCount;               // Length of analogs[] array
```

```
int voltage = 0;              // Voltage
int hMult = 1;                // Horizontal zoom ratio, relative to 1 second
boolean pause = false;    // Storage is suspended display

void setup()
{
  size(530, 290);
  background(102);
  textAlign(CENTER, CENTER);
  textSize(64);
  text("Starting...", width / 2, (height - 40) / 2);
  textSize(16);
  textAlign(LEFT, CENTER);

  analogsCount = width / 2;
  analogs = new int[analogsCount];
  for (int i = 0; i < analogsCount; i++)
    analogs[i] = -1;
}

void draw()
{

  int analog = pcf.analogRead(0);  //serialDevice.requestAnalog();
  if (analog != -1)
  {
    // GUI
    background(102);
    textSize(12);
    text("↑: Zoom up", 120, 6);
    text("↓: Zoom down", 120, 20);
    text("Space: Pause", 220, 20);
    textSize(16);
    // Voltage scale text
    for (int i = 5; i >= 0; i--)
    {
      text(i, 5, 280 - i * 50 - 2);
      if (i == 5)
        text("V", 15, 280 - i * 50 - 2);

  }
    // Horizontal line
    stroke(64, 64, 64);
```

```
        for (int i = 0; i < 6; i++)
            line(30, 30 + i * 50, width, 30 + i * 50);
        // Vertical line time text
        text(1000 / hMult + "ms", 40, 15 - 2);
        // Vertical line
        for (int i = 0; i < (width - 30) / 50 + 1; i++)
            line(30 + i * 50, 30, 30 + i * 50, height - 10);

        if (!pause)
        {
            // Prepare wave data
            for (int i = 0; i < analogsCount - 1; i++)
                analogs[i] = analogs[i + 1];
            analogs[analogsCount - 1] = height - 10 - analog * (height - 10 - 30) / 255;
            // Voltage text
            voltage = analog * 500 / 255;
        }
        String sVoltage = voltage / 100 + "." + voltage / 10 % 10 + voltage % 10;
        text(sVoltage + "V", width - 48, 15 - 2);
        // Wave line
        stroke(0, 255, 0);
        for (int i = width; i > 30; i -= hMult * width / analogsCount)
        {
            int a = i / hMult + width * (hMult - 1) / hMult;
            a = a * analogsCount / width - 1;
            if (analogs[a] >= 0 && analogs[a - 1] >= 0)
                line(i, analogs[a], i - hMult * width / analogsCount, analogs[a - 1]);
        }
    }

}

void keyPressed()
{
    if (key == CODED)
    {
        if (keyCode == UP)
        {
            if (hMult == 1)

hMult = 2;
            else if (hMult == 2)
                hMult = 5;
```

```
            else if (hMult == 5)
                hMult = 10;
        }
        else if (keyCode == DOWN)
        {
            if (hMult == 10)
                hMult = 5;
            else if (hMult == 5)
                hMult = 2;
            else if (hMult == 2)
                hMult = 1;
        }
    }
    else
    {
        if (key == ' ')
        {
            pause = !pause;
            if (!pause)
            {
                for (int i = 0; i < analogsCount; i++)
                    analogs[i] = -1;
            }
        }
    }
}
```

Lesson 16 Control Graphics

Overview

In this lesson, we will learn to change the graph using a potentiometer.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

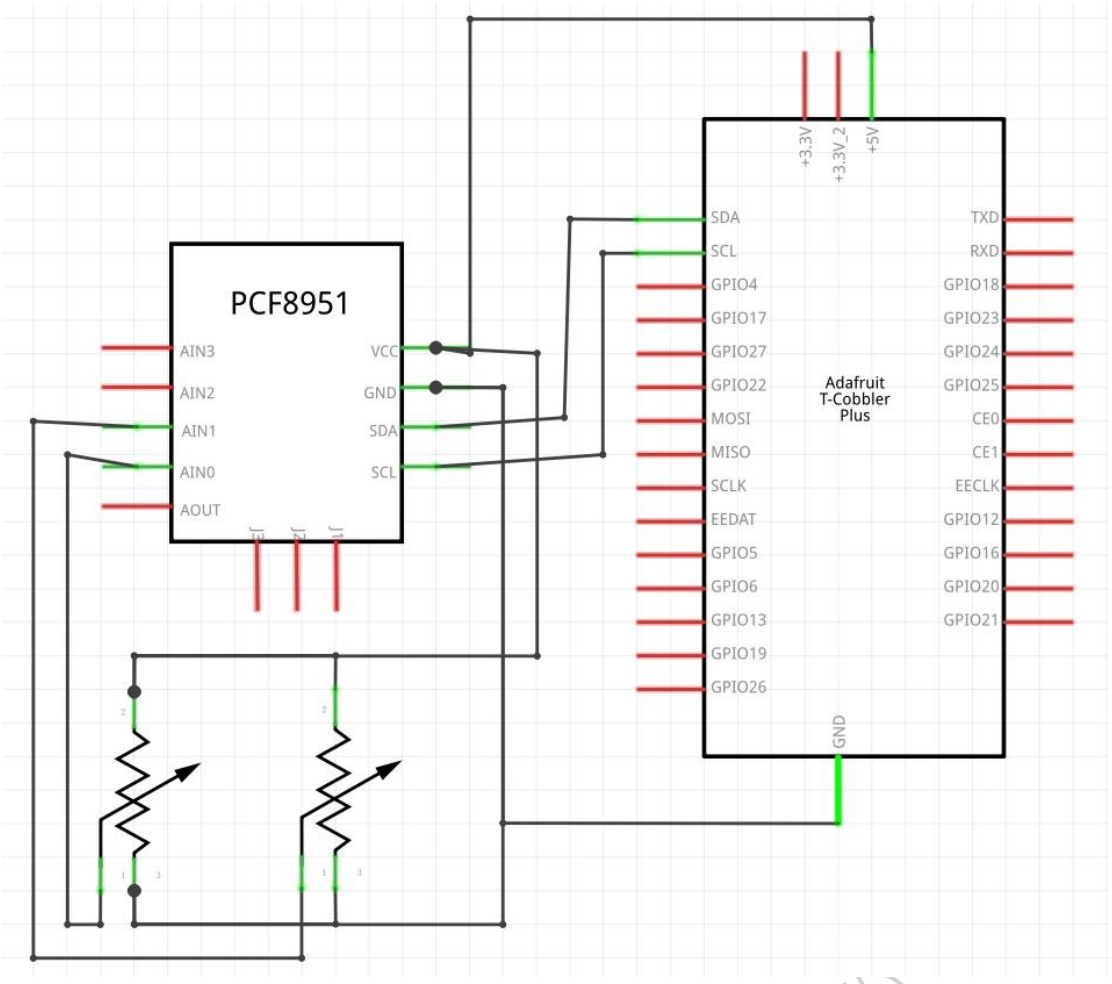
1 x Breadboard

2x Potentiometers

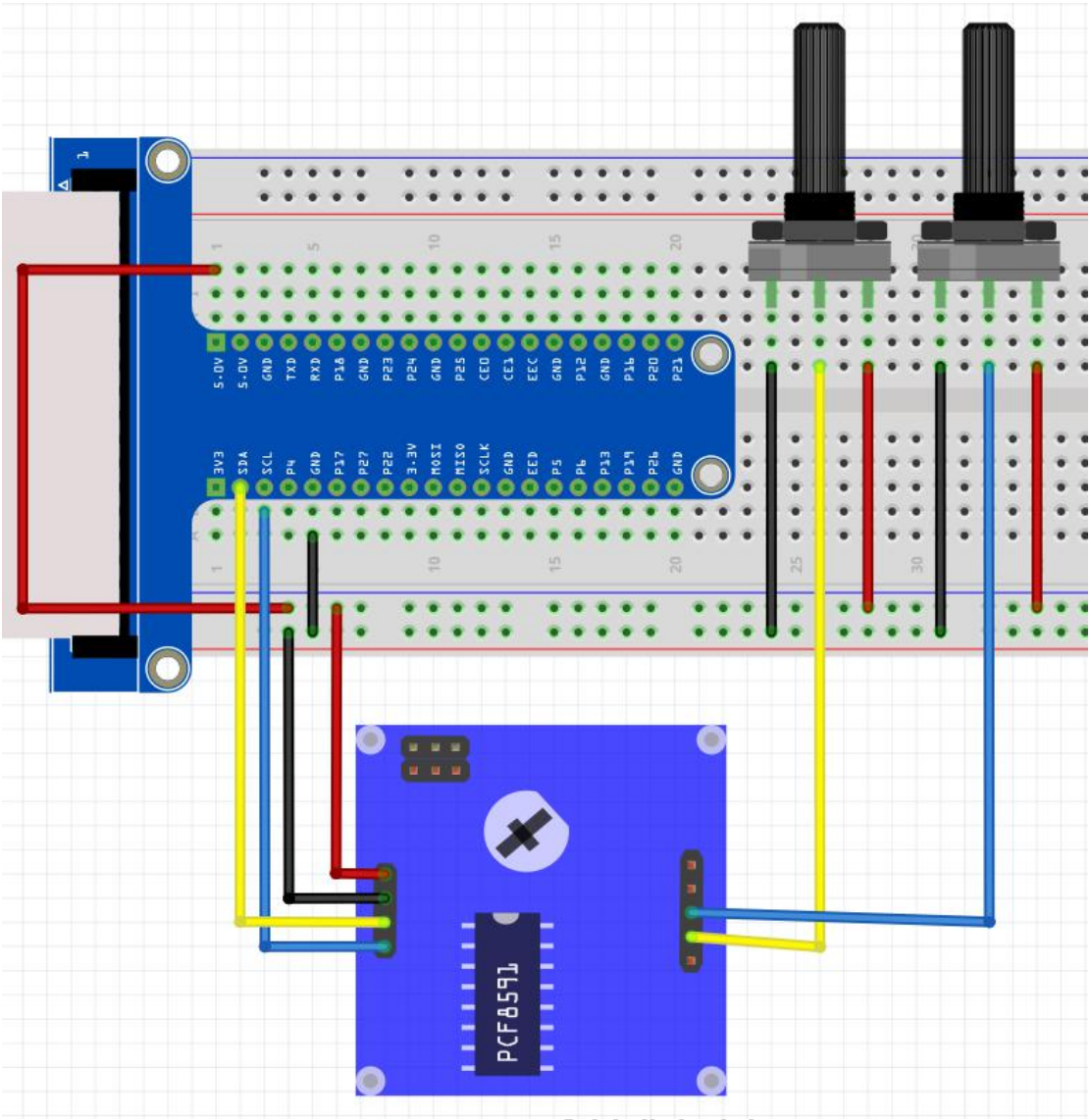
1 x PCF8591 module

Some Jumper Wires

Connection Diagram



Wiring Diagram

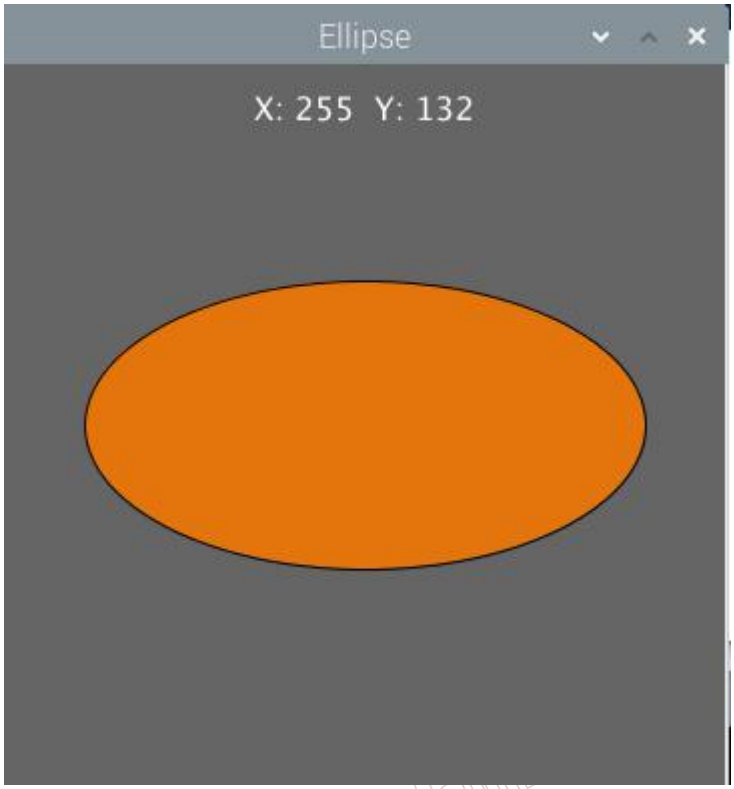


Code

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

1. Enter the processing code / Java / 21.Ellipse / Ellipse / Ellipse.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After the program is executed, rotating potentiometer can change the shape and size of the ellipse. When the voltages of the two positioners are the same, a circle will be displayed, and if not, an ellipse will be displayed.



The following is program code:

```
import processing.io.*;
PCF8591 pcf = new PCF8591(0x48);
void setup()
{
   size(360, 360);
   background(102);
   textAlign(CENTER, CENTER);
   textSize(64);
   text("Starting...", width / 2, (height - 40) / 2);
   textSize(16);
}

void draw()
{
   int[] analogs = new int[2];
```

```
analogs[0] = pcf.analogRead(0);
    analogs[1] = pcf.analogRead(1);
    if (analogs != null)
    {
        background(102);
        drawEllipse(analogs[0], analogs[1]);
    }
}

void drawEllipse(int x, int y)
{
    int maxDiameter = 280;

    fill(255, 255, 255);
    textAlign(CENTER, CENTER);
    textSize(16);
    text("X: " + x, width / 2 - 30, 20);
    text("Y: " + y, width / 2 + 30, 20);

    x = x * maxDiameter / 255;
    y = y * maxDiameter / 255;
    fill(227, 118, 12);
    ellipse(width / 2, height / 2, x, y);
}
```

Lesson 17 PingPong Game

Overview

In this lesson, we will learn to play a table tennis game with a rotary potentiometer on the display interface.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

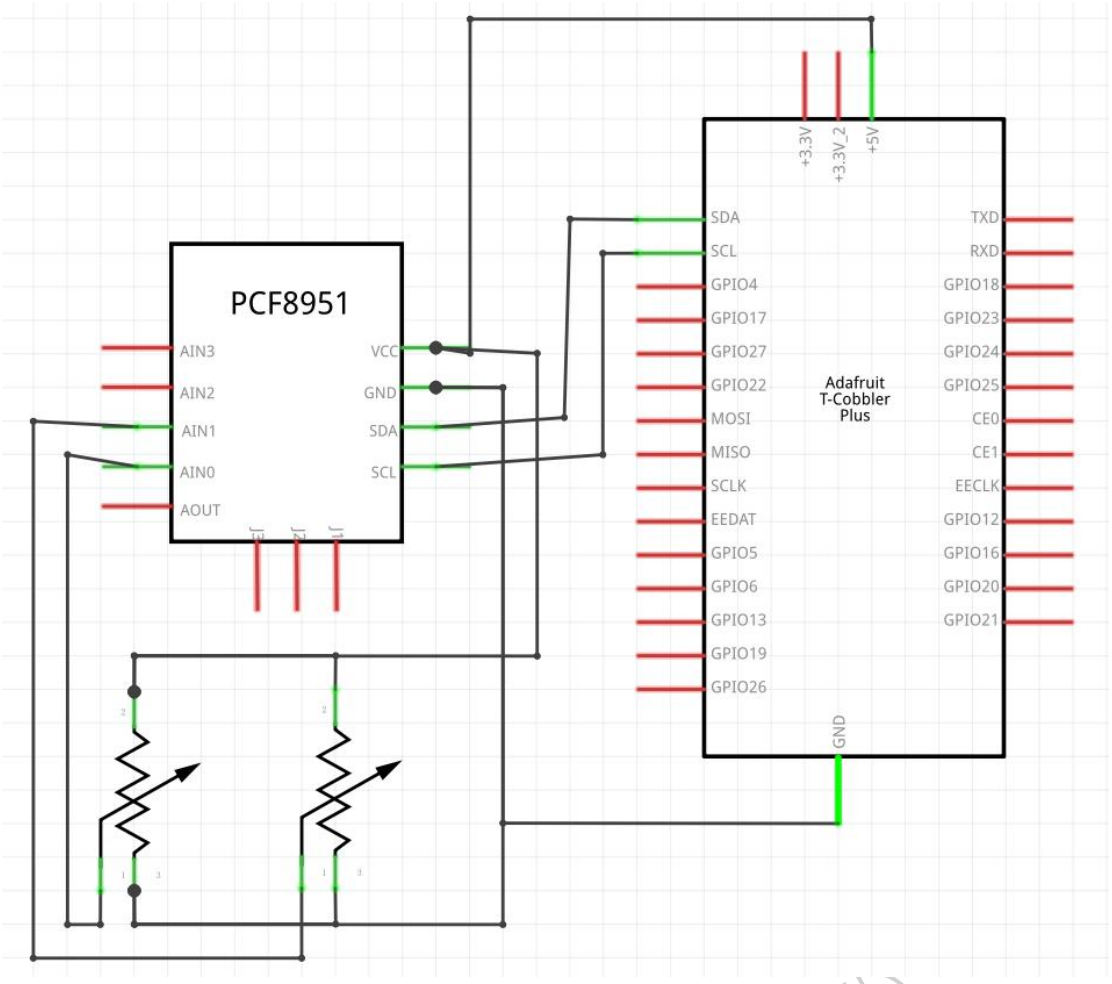
1 x Breadboard

2x Potentiometers

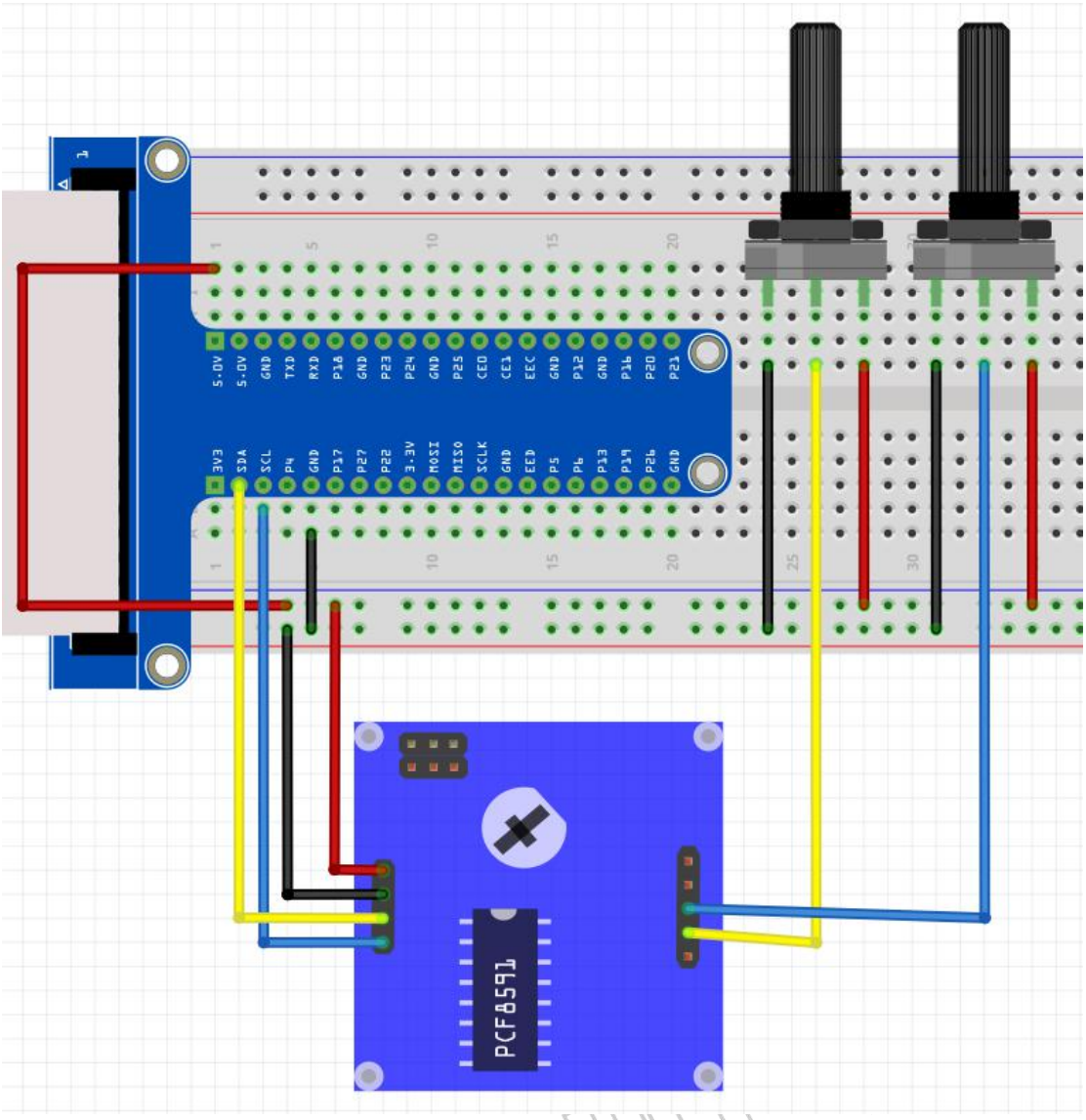
1 x PCF8591 Module

Some Jumper Wires

Connection Diagram



Wiring Diagram

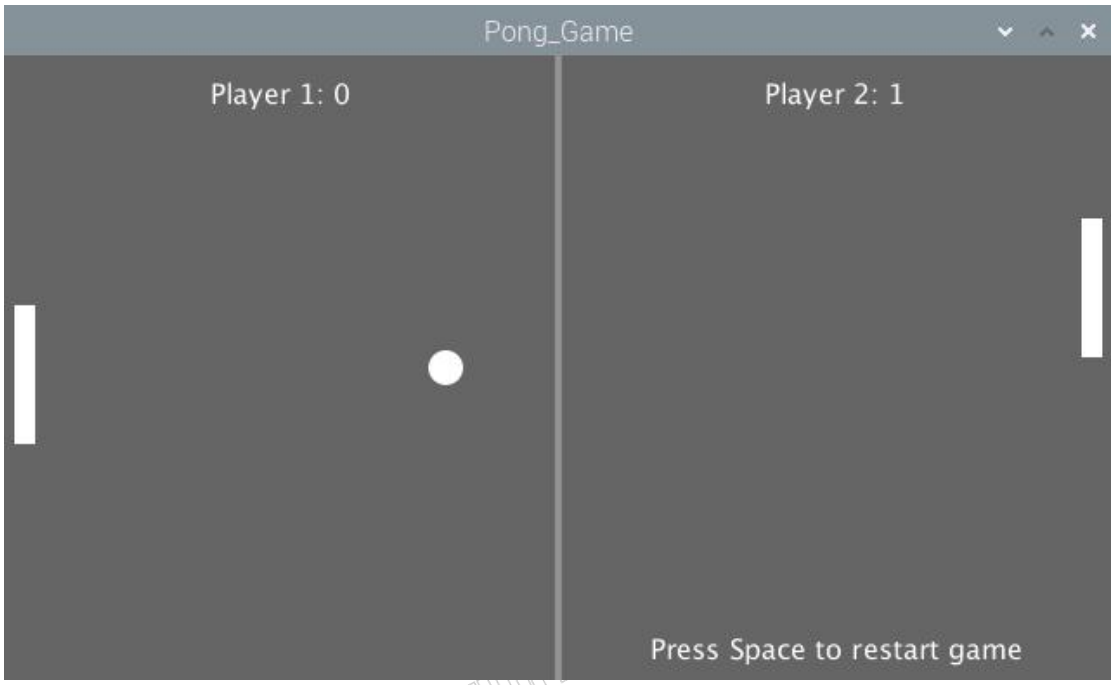


Code

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

1. Enter the processing code / Java / 22.Pong_Game / Pong_Game / Pong_Game.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

Pressing the space bar keyboard can start the game. Then you can try to turn the potentiometer to control the movement of paddle. Use potentiometer to control the movement of paddle to turn back the ball. The rules are the same as the classic PingPong game.



The following is program code:

```
import processing.io.*;

PCF8591 pcf = new PCF8591(0x48);

int winScore = 3;
float acceleration = 0.5;
float deviate = 1;
/* Private variables ---------------------------------------------------*/

Ball ball;
Paddle lPaddle, rPaddle;
int gameState = GameState.WELCOME;
int lScore, rScore;

void setup() {
   size(640, 360);
```

```
    background(102);

textAlign(CENTER, CENTER);
    textSize(64);
    text("Starting...", width / 2, (height - 40) / 2);
    textSize(16);

    ball = new Ball(10);
    lPaddle = new Paddle(new Size(12, 80), 12);
    rPaddle = new Paddle(new Size(12, 80), width - 12);

}

void draw() {
    int[] analogs = new int[2];
    analogs[0] = pcf.analogRead(0);
    analogs[1] = pcf.analogRead(1);
    if (analogs != null)
    {
        lPaddle.position.y = analogs[0] * height /255;
        rPaddle.position.y = analogs[1] * height /255;
    }

    background(102);
    if (gameState == GameState.WELCOME)
    {
        showGUI();
        lPaddle.display();
        rPaddle.display();
        showInfo("Pong Game");
    }
    else if (gameState == GameState.PLAYING)
    {
        ball.updata();
        calculateGame();
        showGUI();
        ball.display();
        lPaddle.display();
        rPaddle.display();
    }
    else if (gameState == GameState.PLAYER1WIN)
    {
        showGUI();
```

```
        lPaddle.display();

rPaddle.display();
        showInfo("Player 1 win!");
    }
    else if (gameState == GameState.PLAYER2WIN)
    {
        showGUI();
        lPaddle.display();
        rPaddle.display();
        showInfo("Player 2 win!");
    }

}

void showInfo(String info)
{
    rectMode(CENTER);
    stroke(0, 0, 0);
    fill(0, 0, 0, 50);
    rect(width / 2, height / 2, width / 2, height / 3);
    fill(255, 255, 255);
    textSize(24);
    textAlign(CENTER, CENTER);
    text(info, width / 2, height / 2 - 24);
    text("Press Space to start", width / 2, height / 2 + 24);
}

void calculateGame()
{
    if (ball.position.x - ball.radius < lPaddle.position.x + lPaddle.size.width / 2)
    {
        if (ball.position.y < lPaddle.position.y - lPaddle.size.height / 2 - ball.radius||
            ball.position.y > lPaddle.position.y + lPaddle.size.height / 2 + ball.radius)
        {
            rScore++;
            ball.reset();
        }
        else
        {
            ball.speed.getSpeed();
            ball.speed.speed += acceleration;
            ball.speed.getXYSpeed((ball.position.y - lPaddle.position.y) /
```

```
(lPaddle.size.height / 2) * deviate);

}
  }

  if (ball.position.x + ball.radius > rPaddle.position.x - rPaddle.size.width / 2)
  {
    if (ball.position.y < rPaddle.position.y - rPaddle.size.height / 2 - ball.radius||
      ball.position.y > rPaddle.position.y + rPaddle.size.height / 2 + ball.radius)
    {
      lScore++;
      ball.reset();
    }
    else
    {
      ball.speed.getSpeed();
      ball.speed.speed += acceleration;
      ball.speed.getXYSpeed((ball.position.y - rPaddle.position.y) /
(rPaddle.size.height / 2) * deviate);
      ball.speed.x = - ball.speed.x;
    }
  }

  if (lScore == winScore)
    gameState = GameState.PLAYER1WIN;
  if (rScore == winScore)
    gameState = GameState.PLAYER2WIN;
}

void showGUI()
{
  fill(255, 255, 255);
  textSize(16);
  textAlign(CENTER, CENTER);
  text("Press Space to restart game", width * 3 / 4, height - 20);
  text("Player 1: " + lScore, width / 4, 20);
  text("Player 2: " + rScore, width * 3 / 4, 20);

  rectMode(CENTER);
  noStroke();
  fill(144, 144, 144);
  rect(width / 2, height / 2, 4, height);
}
```

```
void keyPressed() {
  if (key == ' ')
    {
       lScore = 0;
       rScore = 0;
       ball.reset();
       gameState = GameState.PLAYING;
    }
}
```

Lesson 18 Snake Game

Overview

In this lesson, we will learn to play the snake game with the keys on the display interface.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

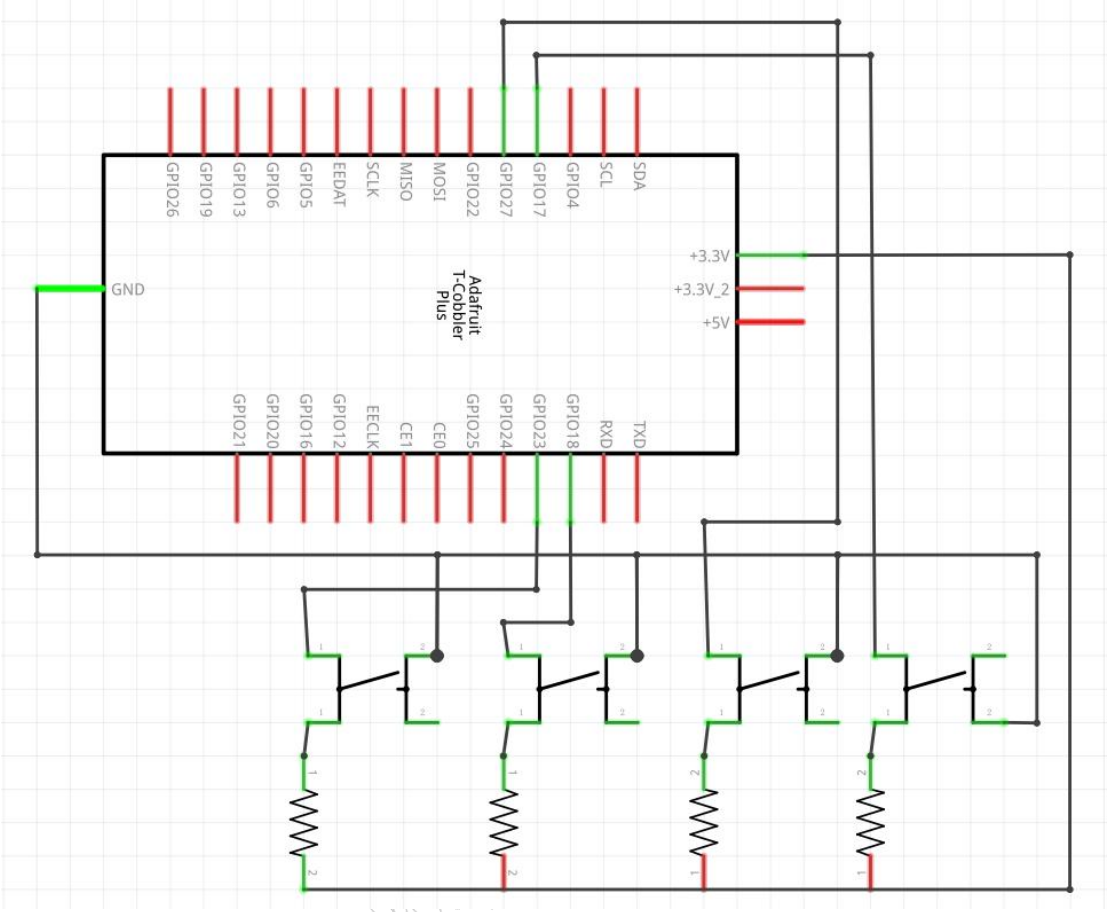
1 x Breadboard

4x 10K Resistor

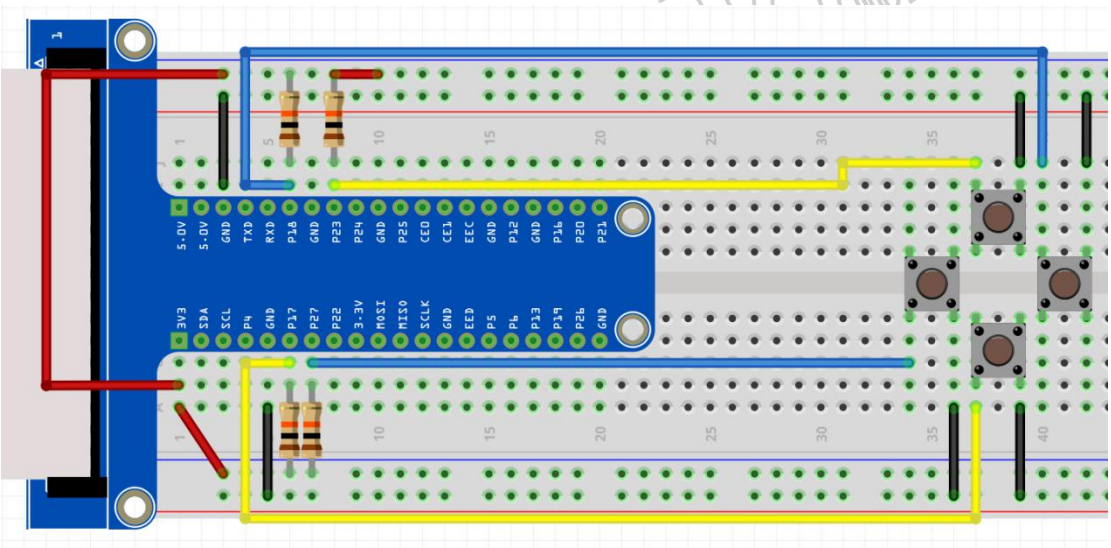
4 x Buttons

Some Jumper Wires

Connection Diagram



Wiring Diagram

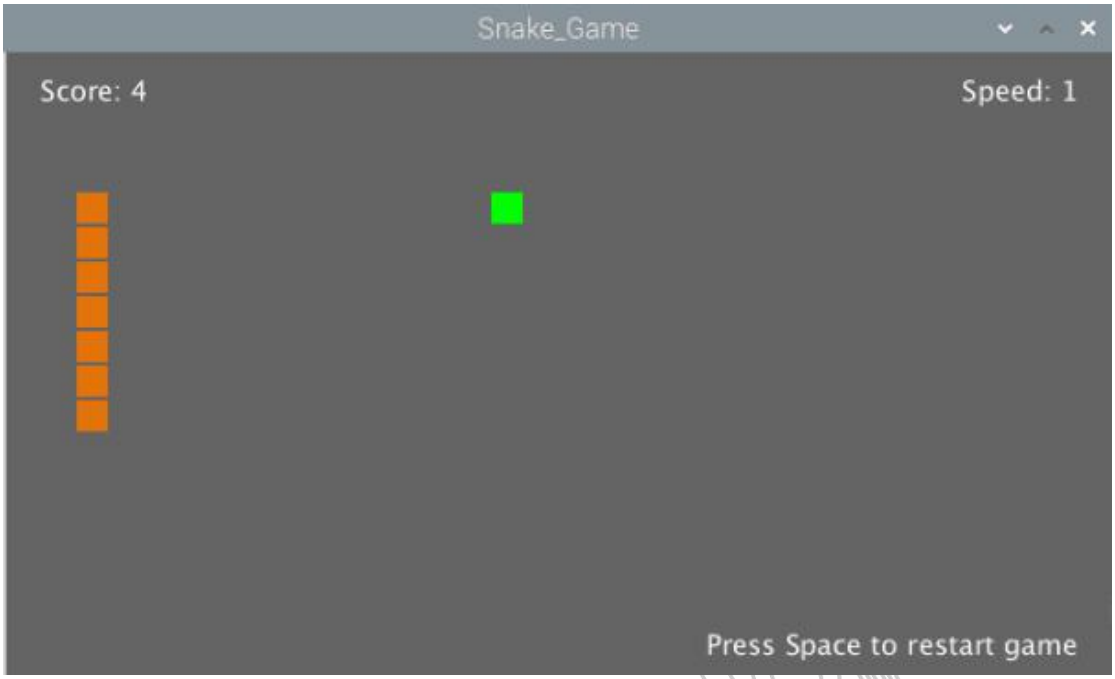


Code

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

1. Enter the processing code / Java / 23.Snake_Game / Snake_Game / Snake_Game.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After the program is executed, 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 following is program code:

```
import processing.io.*;
int threshold = 400;

KeyPad keyUp = new KeyPad(23);
KeyPad keyDown = new KeyPad(17);
KeyPad keyLeft = new KeyPad(22);
KeyPad keyRight = new KeyPad(18);


Snake snake;
```

```
Food food;

void setup() {
  print("Starting ... \n");
  size(640, 360);
  background(102);
  textAlign(CENTER, CENTER);
  textSize(64);
  text("Starting...", width / 2, (height - 40) / 2);
  textSize(16);

  food = new Food(new GridMap(new Size(width, height), 20, 2));
  snake = new Snake(new GridMap(new Size(width, height), 20, 2));
  thread("keypadDetect");
}

void draw() {
  background(102);
  if (snake.gameState == GameState.WELCOME)
  {
    rectMode(CENTER);
    stroke(0, 0, 0);
    fill(0, 0, 0, 50);
    rect(width / 2, height / 2, width / 2, height / 3);
    fill(255, 255, 255);
    textSize(24);
    textAlign(CENTER, CENTER);
    text("Snake Game", width / 2, height / 2 - 24);
    text("Press Space to start", width / 2, height / 2 + 24);
  } else if (snake.gameState == GameState.PLAYING)
  {

    if (snake.body[0].x == food.position.x && snake.body[0].y == food.position.y)
    {
      snake.grow();
      food.generate(snake.body, snake.length);
      snake.speedUp();
    }
    snake.step();
    showGame();
  } else if (snake.gameState == GameState.LOSE)

{
```

```
        showGame();
        rectMode(CENTER);
        stroke(0, 0, 0);
        fill(0, 0, 0, 50);
        rect(width / 2, height / 2, width / 2, height / 3);
        fill(255, 255, 255);
        textSize(24);
        textAlign(CENTER, CENTER);
        text("You lose!", width / 2, height / 2 - 24);
        text("Press Space to start", width / 2, height / 2 + 24);
    }
}

void showGame()
{
    snake.display();
    food.display();

    fill(255, 255, 255);
    textSize(16);
    textAlign(LEFT, CENTER);
    textAlign(RIGHT, CENTER);
    text("Press Space to restart game", width - 20, height - 20);
    textAlign(LEFT, CENTER);
    text("Score: " + (snake.length - 3), 20, 20);
    textAlign(RIGHT, CENTER);
    text("Speed: " + ((snake.initSpeed - snake.speed) / 5 + 1), width - 20, 20);
}

void keyPressed() {
    if ((key == CODED) || (keyValue != -1))
    {
        if ((keyCode == UP) ||((keyValue == keyUp.pin)))
        {
            if (snake.direction != Direction.DOWN)
                snake.nextDirection = Direction.UP;
        } else if ((keyCode == DOWN)||((keyValue == keyDown.pin))) {
            if (snake.direction != Direction.UP)
                snake.nextDirection = Direction.DOWN;
        } else if ((keyCode == LEFT)||((keyValue == keyLeft.pin))) {
            if (snake.direction != Direction.RIGHT)

snake.nextDirection = Direction.LEFT;
```

```
    } else if ((keyCode == RIGHT)||((keyValue ==    keyRight.pin))) {
      if (snake.direction != Direction.LEFT)
        snake.nextDirection = Direction.RIGHT;
    }
    //keyValue = -1;
    println(keyValue);
  } else
  {
    if (key == ' ')
    {
      snake.reset();
      food.generate(snake.body, snake.length);
      snake.gameState = GameState.PLAYING;
    }
  }
}
void keypadDetect() {
  while (true) {
    keyUp.keyScan();
    keyDown.keyScan();
    keyLeft.keyScan();
    keyRight.keyScan();
    transAction();
    try {
      Thread.sleep(10);
    }
    catch(Exception e) {
    }
  }
}
void transAction() {
  if ((keyValue !=   -1))
  {
    if (keyValue ==   keyUp.pin)
    {
      if (snake.direction != Direction.DOWN)
        snake.nextDirection = Direction.UP;
    } else if (((keyValue ==   keyDown.pin))) {
      if (snake.direction != Direction.UP)
        snake.nextDirection = Direction.DOWN;
    } else if (((keyValue ==   keyLeft.pin))) {

if (snake.direction != Direction.RIGHT)
```

```
            snake.nextDirection = Direction.LEFT;
        } else if (((keyValue ==    keyRight.pin))) {
            if (snake.direction != Direction.LEFT)
                snake.nextDirection = Direction.RIGHT;
        }
        keyValue = -1;
    }
}
```

Lesson 19 Tetris Game

Overview

In this lesson, we will learn to play the tetris game with the keys on the display interface.

Parts Required

1 x Raspberry Pi

1 x Raspberry Pi GPIO Expansion Board

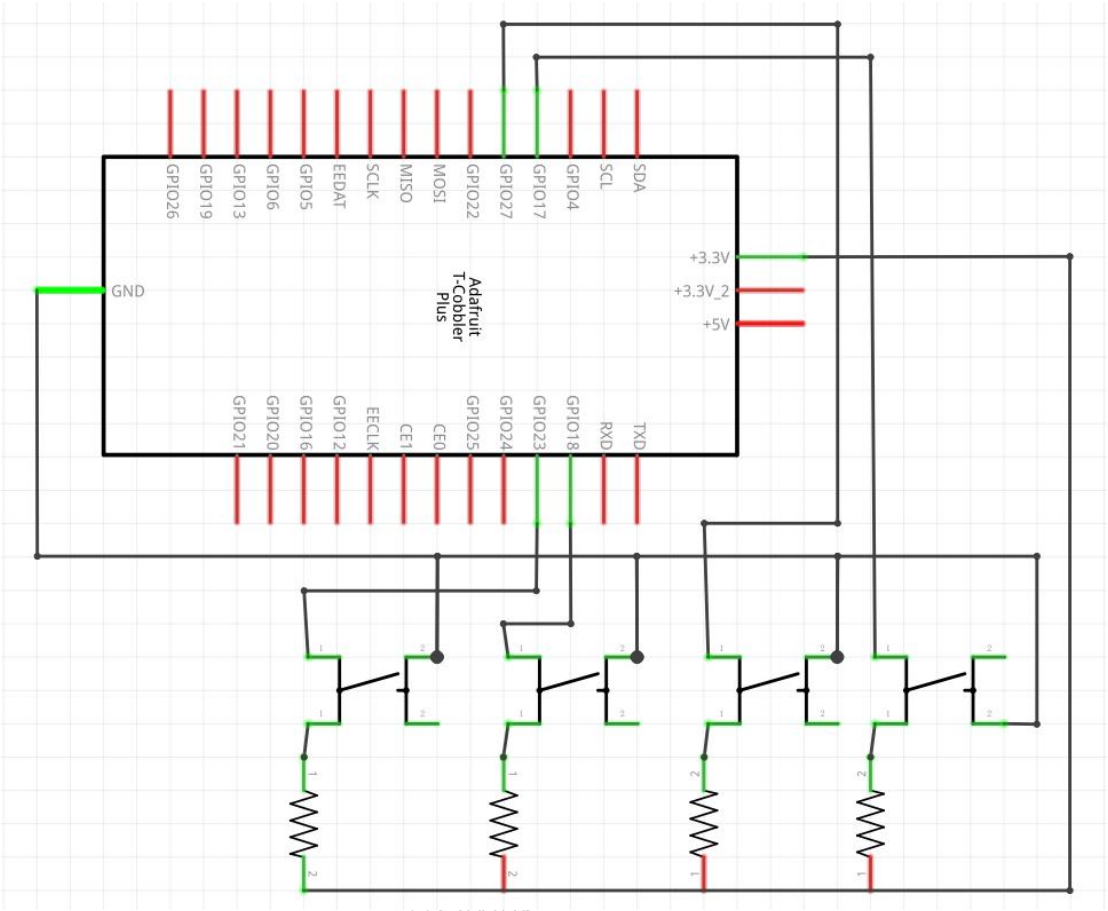
1 x Breadboard

4x 10K Resistors

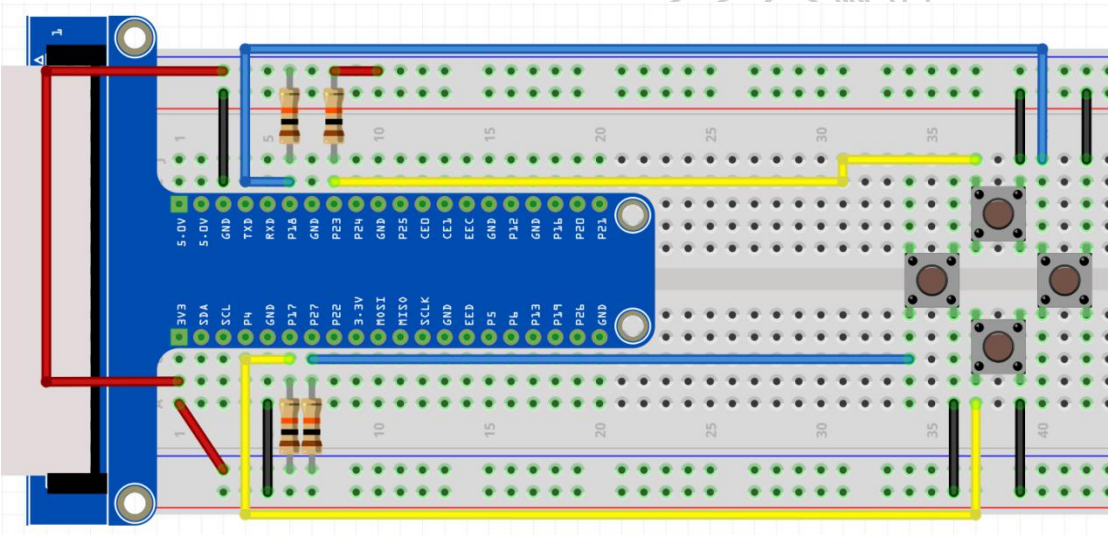
4 x Buttons

Some Jumper Wires

Connection Diagram



Wiring Diagram

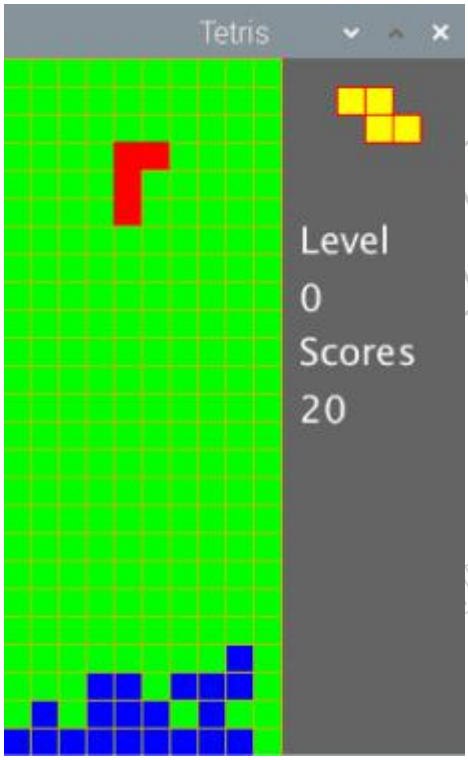


Code

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

1. Enter the processing code / Java / 24.Tetris / Tetris / Tetris.pde command to open the code;
2. Click the "RUN" button in the pop-up "processing" software to run the code;

After executing the program, press the computer spacebar to start the game.



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 keyboard 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. 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.

The following is program code:

```
import processing.io.*;

static final int w = 10; // 4
static final int h = 25; // 60
static final int framesInSecond = 30;
static float gameInitSpeed = 10;
static float gameSpeed = 10;
static final int BlockScale = 15;
static final int sizeWidth = w*BlockScale+100;
static final int sizeHeight = h*BlockScale;


KeyPad keyUp = new KeyPad(23);
KeyPad keyDown = new KeyPad(17);
KeyPad keyLeft = new KeyPad(22);
KeyPad keyRight = new KeyPad(18);

boolean isPaused = false;
boolean keyAllow = true;
float updatingThreshold = 0;
Game game;

float recalculateUpdatingThreshold(float threshold) {
  return threshold + 1;
}
void settings() {
  size(sizeWidth, sizeHeight);
}
void setup() {
  game = new Game(w, h);
  generateRandomBlock(game);
  frameRate(framesInSecond);
  thread("keypadDetect");
}

void draw() {

  background(102);
  Game newGame = game;

  updatingThreshold = recalculateUpdatingThreshold(updatingThreshold);
```

```
    if (updatingThreshold > gameSpeed) {
      if (isGameOver(newGame)) {
      } else if (isPaused) {
      } else {
        newGame = updateGameState(game);
      }
      updatingThreshold = 0;
    }
    drawGameState(newGame);
    if (!isGameOver(newGame)&& (isPaused)) {    //pause
      textSize(40);
      fill(0);
      text("Pause", BlockScale*2, 150);
      keyAllow = false;
    } else if (isGameOver(newGame)&& (isPaused)) {       //restart game
      game = new Game(w, h);
      generateRandomBlock(game);
      isPaused = false;
      keyAllow = false;
    } else if (isGameOver(newGame)) {       //game over
      textSize(40);
      fill(0);
      text("Game \nOver", BlockScale*2, 150);
      keyAllow = false;
    } else {              //playing
      keyAllow = true;
    }

    //level,score information
    pushMatrix();
    translate(w*BlockScale, 0);
    fill(255);
    textSize(20);
    text("Level\n"+game.level, 10, BlockScale*7);
    text("Scores\n"+game.score, 10, BlockScale*11);
    textSize(12);
    drawNextBlock(game.nextBlock, BlockScale*2, BlockScale*1);
    popMatrix();
}

void keyPressed() {

if (key == CODED) {
```

```
        if (keyAllow) {
            switch (keyCode) {
            case LEFT:
                moveBlock(game, MoveLeft);
                break;
            case RIGHT:
                moveBlock(game, MoveRight);
                break;
            case DOWN:
                makeBlockFall(game);
                break;
            case UP:
                rotateBlock(game);
                break;
            }
        }
    } else if (key == ' ') { // SPACE
        isPaused\=! isPaused;
    }
}
void keypadDetect() {
    while (true) {
        keyUp.keyScan();
        keyDown.keyScan();
        keyLeft.keyScan();
        keyRight.keyScan();
        transAction();
        try {
            Thread.sleep(10);
        }
        catch(Exception e) {
        }
    }
}
void transAction() {
    if ((keyValue !=   -1))
    {
        if (keyAllow) {
            if (keyValue == keyLeft.pin) {
                moveBlock(game, MoveLeft);
            } else if (keyValue == keyRight.pin) {

moveBlock(game, MoveRight);
```

```
            } else if (keyValue == keyDown.pin) {
                makeBlockFall(game);
            } else if (keyValue == keyUp.pin) {
                rotateBlock(game);
            }
        }
        try {
            Thread.sleep(50);
        }
        catch(Exception e) {
        }
        keyValue =    -1;
    }
}
```