Lesson 5

Light Mode

Introduction:

There are two stages in this lesson. Each stage has three parts. First of all, we will teach you how to light the color lights on the car and how to change the brightness and color of the lights.

Key control light effect

2.1 Hardware Design
2.2 Software Design
2.3 Upload Validation

Preparations:

one car (with a battery)
one USB cable

1.1 Hardware Design

In our kit, the lighting effects are implemented using the WS2812B module.WS2812B is a intelligent control LED light source that the control circuit and RGB chip are integrated in a package of 5050 components.The data transfer protocol use single NZR communication mode. After the pixel power-on reset, the DIN port receive data from controller, the first pixel collect initial 24bit data then sent to the internal data latch, the other data which reshaping by the internal signal reshaping amplification circuit sent to the next cascade pixel through the DO port. After transmission for each pixel, the signal to reduce 24bit. pixel adopt auto resha -ping transmit technology, making the pixel cascade number is not limited the signal transmission, only depend on the speed of signal transmission.

LED with low driving voltage, environmental protection and energy saving, high brightness, scattering angle is large, good consistency, low power, long life and other advantages. The control chip integrated in LED above becoming more simple circuit, small volume, convenient installation.

In summary, WS2812B is connected by cascading, so we only need to control one IO port to control multiple WS2812B.
(As shown in figure 1.1.1~1.1.2)

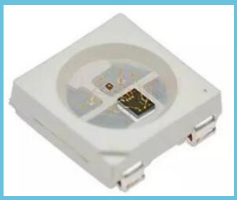
Figure 1.1.1 The real object of the WS2812B

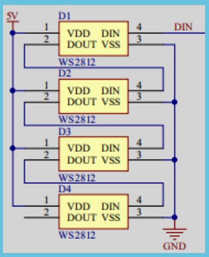

Figure 1.1.2 The schematic of the WS2812B
(Connect in cascade)

Finally, let's see which pin of Nano is connected to Din and start to write programs.
(As shown in figure 1.1.3)

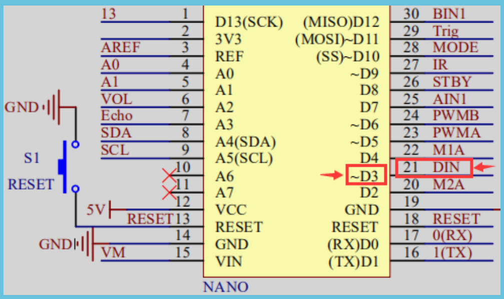

Figure 1.1.3 Din is connected to D3

RGB light principle: there are three smaller red LED, blue LED and green LED in the light bead of RGB color light. We can mix their colors to produce different colors by lighting the three small lights respectively or simultaneously and adjusting their brightness.

1.2 Software Design

Please open ELEGOO Tumbller Self-Balancing Car Tutorial -> Lesson 5 Light Mode->Light->Light.ino in the current folder

In this course, we will focus on learning how to use Adafruit_NeoPixel library. So we are going to add Adafruit_NeoPixel.h first.

```
//in Rgb.h
#include "Adafruit_NeoPixel.h"
```

Because our RGB class inherits the Adafruit_NeoPixel class in the Adafruit_NeoPixel library, we can create our own lighting effects based on the original basic functions.

```
//in Rgb.h
class RGB : public Adafruit_NeoPixel
{}rgb;
```

First of all, let 's look at the definition and initialization of related pins.

```
//in Rgb.h
#define RGB_PIN 3
#define NUMPIXELS 4
```

```
//in Light
void setup() {
 rgb.initialize();
}
```

```
//in Light
void initialize()
 {
  begin();
  setBrightness(brightness);
  show();
 }
```

```
//in Adafruit_NeoPixel.cpp
void Adafruit_NeoPixel::begin(void)
{......}
void Adafruit_NeoPixel::setBrightness(uint8_t b)
{......}
void Adafruit_NeoPixel::show(void)
{......}
```

Next, let's learn the functions that we use.

void Adafruit_NeoPixel::setPixelColor(uint16_t n, uint8_t r, uint8_t g, uint8_t b)

n: which lamp to choose

r: The brightness of the red LED in that lamp

g: The brightness of the green LED in that lamp

b: The brightness of the blue LED in that lamp

void Adafruit_NeoPixel::setPixelColor(uint16_t n, uint32_t c)

n: which lamp to choose

c: Set pixel color from 'packed' 32-bit RGB color:

```
// Convert separate R,G,B into packed 32-bit RGB color.
// Packed format is always RGB, regardless of LED strand color order.
uint32_t Adafruit_NeoPixel::Color(uint8_t r, uint8_t g, uint8_t b)
```

show()：Use this function to turn on the light after setting it

Finally, we can make different lighting effects according to these basic functions and our preferences. Please check how to realize the following lighting effect by yourself in rgb.h

```
//in Light
void loop() {

  rgb.theaterChaseRainbow(50);

  rgb.rainbowCycle(20);

  rgb.theaterChase(127, 127, 127, 50);

  rgb.rainbow(20);

  rgb.whiteOverRainbow(20, 30, 4);

  rgb.rainbowFade2White(3, 50, 50);
}
```

Finally, we can make different lighting effects according to these basic functions and our preferences. Please check how to realize the following lighting effect by yourself in rgb.h

```
//in Light
void loop() {

  rgb.theaterChaseRainbow(50);

  rgb.rainbowCycle(20);

  rgb.theaterChase(127, 127, 127, 50);

  rgb.rainbow(20);

  rgb.whiteOverRainbow(20, 30, 4);

  rgb.rainbowFade2White(3, 50, 50);
}
```

1.3 Upload Validation

After uploading the program, you will see all kinds of nice lighting effects.

2.1 Hardware Design

Please open ELEGOO Tumbller Self-Balancing Car Tutorial -> Lesson 5 Light Mode->KeyControl_light-> KeyControl_light.ino in the current folder.

After learning to set the lighting effect, we will learn how to switch our favorite lighting effect by pressing the key. First, look at the location of the key connection on the schematic diagram. (As shown in figure 2.1.1~2.1.3).

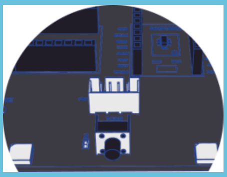

Figure 2.1.1 The real object of the button

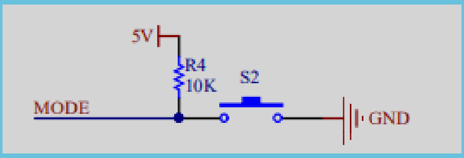

Figure 2.1.2 The schematic of the button

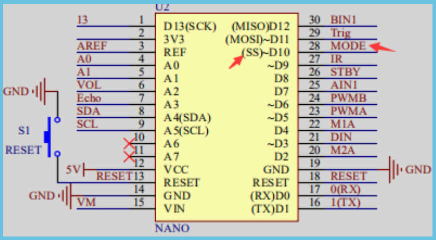

Figure 2.1.3 Pins connected to button on Nano

2.2 Software Design

First of all, we define the pin of the key. Because the key interrupt is used, we need to add the corresponding library:"pinchangeint. H", and initialize the corresponding pin interrupt.

```
//in keyControl_Light
#include "PinChangeInt.h"
#define BUTTON 10

void setup() {
   attachPinChangeInterrupt(BUTTON, Mode_KeyControl, FALLING );
}
```

Because of jitter in our mechanical keys, we need to eliminate jitter. (As shown in figure 2.1.4).

Mechanical elastic switch:

When the mechanical contact is disconnected and connected, because of the elastic effect of the mechanical contact, a switch key will not be connected stably immediately when it is closed, nor will it be disconnected completely at once when it is disconnected. Instead, a series of jitters accompanied by the moment of connection and disconnection causes the reality to be inconsistent with the ideal effect. In order not to produce this, the measure of this phenomenon is to eliminate the jitters of the button.

The mechanical dithering schematic diagram is as follows:

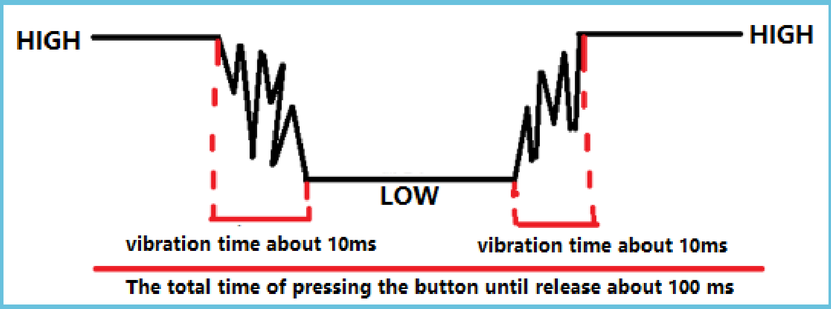
HIGH
HIGH
LOW
vibration time about 10ms
vibration time about 10ms
The total time of pressing the button until release about 100 ms

Figure 2.1.4 The principle of key jitters

Anti shake processing：

Because of jitter, we need to add a short delay to eliminate the jitter.

But the use of delay in interruption results in lower overall efficiency, which means the machine running idle and doing nothing.

So we use millis to record the time, when the interrupt is triggered twice, when the button is pressed twice.
In this case, the descent edge is used to trigger the interrupt.
According to the figure, the interval between the two descent edges is about 200 ms.

The recording time of the second key interruption - the recording time of the last key interruption > 200 determines that the key is pressed rather than shaken.

Finally, press the key to switch the light effect

```
void Mode_KeyControl()
{
  static int time_button,last_time_button;
  static int mode;
  time_button=millis();
  if(200 < (abs(time_button - last_time_button)))
  {
   ......
  }
}
```

```
  mode++;
  switch(mode)
  {
   case 1:rgb.theaterChaseRainbow(50);break;
   case 2:rgb.rainbowCycle(20);break;
   case 3:rgb.theaterChase(127, 127, 127, 50);break;
   case 4:rgb.rainbow(20);break;
   case 5:rgb.whiteOverRainbow(20, 30, 4);break;
   case 6:rgb.rainbowFade2White(3, 50, 50); break;
   default:break;
  }
  if(mode>6)
   mode=0;
```

1.3 Upload Validation

After uploading the program, you can switch the effect of the light every time you press the key.