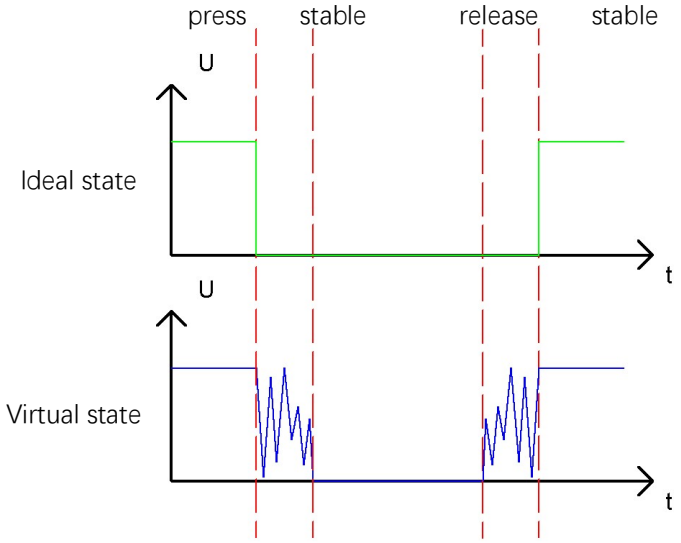
Project 2.2 MINI table lamp

We will also use a button, LED and RPi to make a MINI table lamp. But the function is different: Press the button, the LED will be turned on, and press the button again, the LED goes out.
First, let us learn some knowledge about the button.

Debounce for Push Button

When a Push Button is pressed, it will not change from one state to another state immediately. Due to mechanical vibration, there will be a continuous buffeting before it becomes another state. And the releasing-situation is similar with that process.



Therefore, if we directly detect the state of Push Button, there may be multiple pressing and releasing action in one pressing process. The buffeting will mislead the high-speed operation of the microcontroller to cause a lot of false judgments. So we need to eliminate the impact of buffeting. Our solution is: to judge the state of the button several times. Only when the button state is stable after a period of time, can it indicate that the button is pressed down.
This project needs the same components and circuits with the last section.

Code

In the project, we still detect the state of Button to control LED. Here we need to define a variable to save the state of LED. And when the button is pressed once, the state of LED will be changed once. This has achieved the function of the table lamp.

C Code 2.2.1 Tablelamp

First observe the project result, and then analyze the code.

1. Use cd command to enter 02.2.1_Tablelamp directory of C code.

```
cd ~/Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/C_Code/02.1.1_Tablelamp
```

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

```
gcc Tablelamp.c -o Tablelamp -lwiringPi
```

3. Tablelamp. Then run the generated file "Tablelamp".

```
sudo ./Tablelamp
```

When the program is executed, press the Button once, then LED is turned on. Press the Button another time, then LED is turned off.

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3
4  #define ledPin    0     //define the ledPin
5  #define buttonPin 1     //define the buttonPin
6  int ledState=LOW;        //store the State of led
7  int buttonState=HIGH;    //store the State of button
8  int lastbuttonState=HIGH;//store the lastState of button
9  long lastChangeTime;     //store the change time of button state
10 long captureTime=50;     //set the button state stable time
11 int reading;
12 int main(void)
13 {
14     if(wiringPiSetup() == -1){ //when initialize wiring failed,print message to screen
15         printf("setup wiringPi failed !");
16         return 1;
17     }
18     printf("Program is starting...\n");
19     pinMode(ledPin, OUTPUT);
20     pinMode(buttonPin, INPUT);
21
22     pullUpDnControl(buttonPin, PUD_UP);  //pull up to high level
23     while(1){
24         reading = digitalRead(buttonPin); //read the current state of button
25         if( reading != lastbuttonState){  //if the button state has changed ,record the
26 time point
27             lastChangeTime = millis();
28         }
```

```
29          //if changing-state of the button last beyond the time we set, we considered that
30          //the current button state is an effective change rather than a buffeting
31          if(millis() - lastChangeTime > captureTime){
32              //if button state is changed ,update the data.
33              if(reading != buttonState){
34                  buttonState = reading;
35                  //if the state is low ,the action is pressing
36                  if(buttonState == LOW){
37                      printf("Button is pressed!\n");
38                      ledState = !ledState;
39                      if(ledState){
40                          printf("turn on LED ...\n");
41                      }
42                      else {
43                          printf("turn off LED ...\n");
44                      }
45                  }
46                  //if the state is high ,the action is releasing
47                  else {
48                      printf("Button is released!\n");
49                  }
50              }
51          }
52          digitalWrite(ledPin,ledState);
53          lastbuttonState = reading;
54      }
55      return 0;
56  }
```

This code focuses on eliminating the buffeting of button. We define several variables to save the state of LED and button. Then read the button state in while () constantly, and determine whether the state has changed. If it is, record this time point.

```
reading = digitalRead(buttonPin); //read the current state of button
        if( reading != lastbuttonState){
            lastChangeTime = millis();
        }
```

millis()

This returns a number representing the number of milliseconds since your program called one of the wiringPiSetup functions. It returns an unsigned 32-bit number which wraps after 49 days.

Then according to just recorded time point, judge the duration of the button state change. If the duration exceeds captureTime (buffeting time) we set, it indicates that the state of the button has changed. During that time, the while () is still detecting the state of the button, so if there is a change, the time point of change will be updated. Then duration will be judged again until the duration of there is a stable state exceeds the time we set.

```
        if(millis() - lastChangeTime > captureTime){
            //if button state is changed ,update the data.
            if(reading != buttonState){
                buttonState = reading;
```

Finally, judge the state of Button. And if it is low level, the changing state indicates that the button is pressed, if the state is high level, then the button is released. Here, we change the status of the LED variable, and then update the state of LED.

```
if(buttonState == LOW){
                    printf("Button is pressed!\n");
                    ledState = !ledState;
                    if(ledState){
                        printf("turn on LED ...\n");
                    }
                    else {
                        printf("turn off LED ...\n");
                    }
                }
                //if the state is high ,the action is releasing
                else {
                    printf("Button is released!\n");
                }
```

Python Code 2.2.1 Tablelamp

First observe the project result, and then analyze the code.

1. Use cd command to enter 02.2.1_Tablelamp directory of Python code

```
cd ~/Freenove_Ultimate_Starter_Kit_for_Raspberry_Pi/Code/Python_Code/02.2.1_Tablelamp
```

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

```
python Tablelamp.py
```

When the program is executed, press the Button once, then LED is turned on. Press the Button another time, then LED is turned off.

```
1  import RPi.GPIO as GPIO
2
3  ledPin = 11    # define the ledPin
4  buttonPin = 12    # define the buttonPin
5  ledState = False
6
7  def setup():
8      print ('Program is starting...')
9      GPIO.setmode(GPIO.BOARD)       # Numbers GPIOs by physical location
10     GPIO.setup(ledPin, GPIO.OUT)   # Set ledPin's mode is output
11     GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)    # Set buttonPin's mode is
12 input, and pull up to high
13
14 def buttonEvent(channel):#When the button is pressed, this function will be executed
15     global ledState
16     print ('buttonEvent GPIO%d' %channel)
17     ledState = not ledState
18     if ledState :
19         print ('Turn on LED ... ')
20     else :
21         print ('Turn off LED ... ')
22     GPIO.output(ledPin,ledState)
23
24 def loop():
25     #Button detect
26     GPIO.add_event_detect(buttonPin,GPIO.FALLING,callback = buttonEvent,bouncetime=300)
27     while True:
28         pass
29
30 def destroy():
31     GPIO.output(ledPin, GPIO.LOW)     # led off
32     GPIO.cleanup()                    # Release resource
33
34 if __name__ == '__main__':     # Program start from here
35     setup()
36     try:
```

```
        loop()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the subprogram destroy() will be executed.
        destroy()
```

RPi.GPIO provides us with a simple and effective function to eliminate the jitter, that is GPIO.add_event_detect(). It uses callback function. Once it detect that the buttonPin has a specified action FALLING, execute the specified function buttonEvent(). In the function buttonEvent, each time the ledState is reversed, the state of the LED will be updated.

```
def buttonEvent(channel):
    global ledState
    print 'buttonEvent GPIO%d'%channel
    ledState = not ledState
    if ledState :
        print ('Turn on LED ... ')
    else :
        print ('Turn off LED ... ')
    GPIO.output(ledPin,ledState)

def loop():
    #Button detect
    GPIO.add_event_detect(buttonPin,GPIO.FALLING,callback = buttonEvent,bouncetime=300)
    while True:
        pass
```

Of course, you can also use the same programming idea of C code above to achieve this target.

GPIO.add_event_detect(channel, GPIO.RISING, callback=my_callback, bouncetime=200)

This is an event detection function. The first parameter specifies the IO port to be detected. The second parameter specifies the action to be detected. The third parameter specified a function name, the function will be executed when the specified action is detected. And the fourth parameter is used to set the jitter time.