Lesson 1 Blinking LED

Overview

In this tutorial, we will start the journey of learning Raspberry Pi. In the first lesson, we will learn how to control an LED.

Requirement

- 1* Raspberry Pi
- 1* 220Ω Resistor
- 1* LED
- 1* Breadboard
- 2* Jumper Wires

Principle

In this lesson, we will program the Raspberry Pi to output high(+3.3V) and low level(0V), and then make an LED which is connected to the Raspberry Pi GPIO flicker with a certain frequency.

1. What is LED ?

The LED is the abbreviation of light emitting diode. It is usually made of gallium arsenide, gallium phosphide semiconductor materials. The LED has two electrodes, a positive electrode and a negative electrode, it will light only when a forward current passes, and it can be red, blue, green or yellow light, etc. The color of light depends on the materials it was made.

In general, the drive current for LED is 5-20mA. Therefore, in reality it usually needs an extra resistor for current limitation so as to protect the LED.

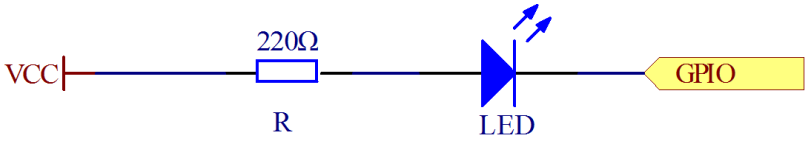
2. What is the resistor ?

The main function of the resistor is to limit current. In the circuit, the character 'R' represents resistor, and the unit of resistor is ohm(Ω).

The band resistor is used in this experiment. A band resistor is one whose surface is coated with some particular color through which the resistance can be identified directly.

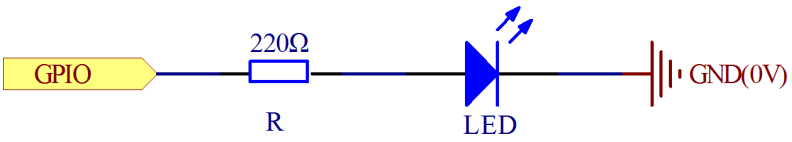
There are two methods for connecting an LED with Raspberry Pi GPIO:

①



As shown in the schematic diagram above, the anode of LED is connected to VCC(+3.3V), and the cathode of LED is connected to the Raspberry Pi GPIO. When the GPIO output low level, the LED is on; when the GPIO output high level, the LED is off.

②



As shown in the schematic diagram above, the anode of LED is connected to Raspberry Pi GPIO via a resistor, and the cathode of LED is connected to the ground (GND). When the GPIO output high level, the LED is on; when the GPIO output low level, the LED is off.

The size of the current-limiting resistor is calculated as follows : 5~20mA current is required to make an LED on, and the output voltage of the Raspberry Pi GPIO is 3.3V, so we can get the resistance：

R = U / I = 3.3V / (5~20mA) = 165Ω~660Ω

In this experiment, we choose a 220ohm resistor.

The experiment is based on method ①, we select pin 11 of Raspberry Pi to control an LED. When the pin 11 of Raspberry Pi is programmed to output low level, then the LED will be lit, next delay for the amount of time, and then programmed the pin 11 to high level to make the LED off. Continue to perform the above process, you can get a blinking LED.

3. Key functions

C language user:

- **int wiringPiSetup (void)**

The function must be called at the start of your program or your program will fail to work correctly. You may experience symptoms from it simply not working to segfaults and timing issues.

Note: This function needs to be called with root privileges.

- **void pinMode (int pin, int mode)**

This sets the mode of a pin to either **INPUT**, **OUTPUT**, **PWM_OUTPUT** or **GPIO_CLOCK**. Note that only wiringPi pin 1 (BCM_GPIO 18) supports PWM output and only wiringPi pin 7 (BCM_GPIO 4) supports CLOCK output modes.

This function has no effect when in Sys mode. If you need to change the pin mode, then you can do it with the gpio program in a script before you start your program.

- **void digitalWrite (int pin, int value)**

Writes the value **HIGH** or **LOW** (1 or 0) to the given pin which must have been previously set as an output. ***WiringPi*** treats any non-zero number as HIGH, however 0 is the only representation of LOW.

- **void delay (unsigned int howLong)**

This causes program execution to pause for at least howLong milliseconds. Due to the multi-tasking nature of Linux it could be longer. Note that the maximum delay is an unsigned 32-bit integer or approximately 49 days.

Python user:

- **GPIO.setmode(GPIO.BOARD)**

There are two ways of numbering the IO pins on a Raspberry Pi within RPi.GPIO. The first is using the **BOARD** numbering system. This refers to the pin numbers on the P1 header of the Raspberry Pi board. The advantage of using this numbering system is that your hardware will always work, regardless of the board revision of the RPi. You will not need to rewire your connector or change your code.

The second numbering system is the **BCM**(GPIO.BCM) numbers. This is a lower level way of working - it refers to the channel numbers on the Broadcom SOC. You have to always work with a diagram of which channel number goes to which pin on the RPi board. Your script could break between revisions of Raspberry Pi boards.

- **GPIO.setup(channel, mode)**

This sets every channel you are using as an input(GPIO.IN) or an output(GPIO.OUT).

- **GPIO.output(channel, state)**

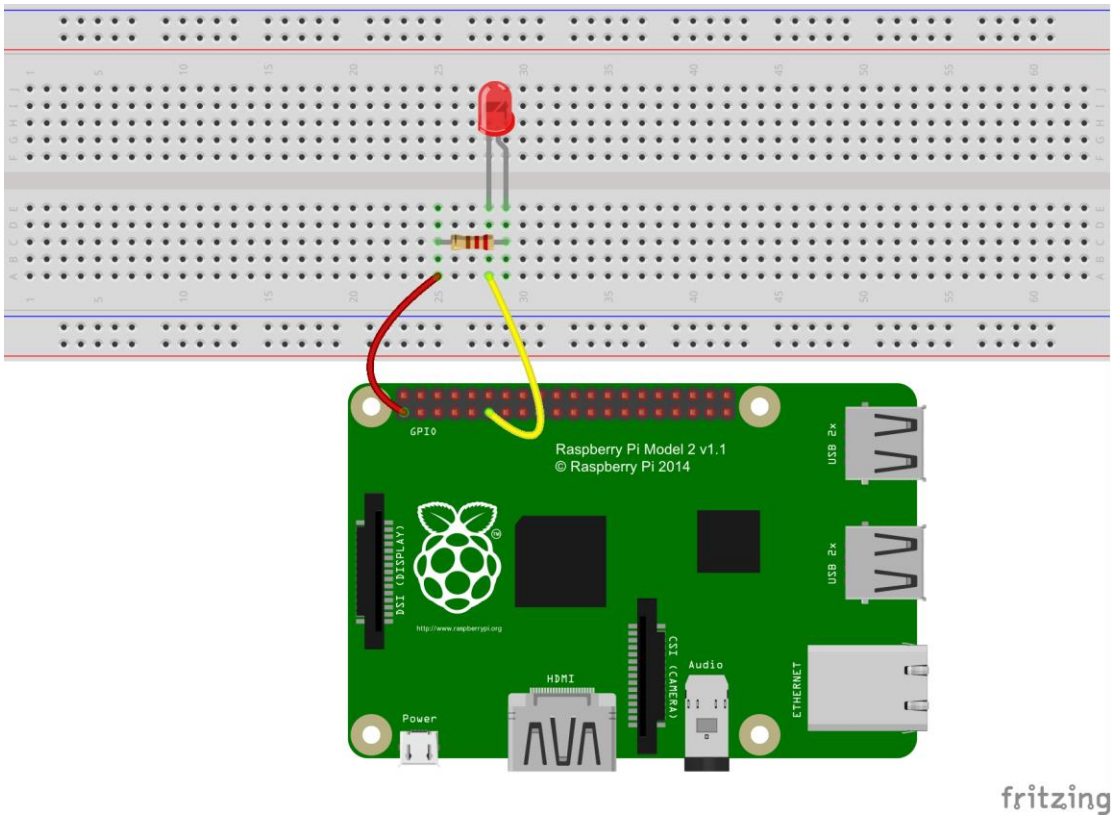
This sets the output state of a GPIO pin. Where channel is the channel number based on the numbering system you have specified (BOARD or BCM). **State** can be 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.

- **GPIO.cleanup( )**

At the end any program, it is good practice to clean up any resources you might have used. This is no different with RPi.GPIO. By returning all channels you have used back to inputs with no pull up/down, you can avoid accidental damage to your RPi by shorting out the pins. Note that this will only clean up GPIO channels that your script has used. Note that GPIO.cleanup() also clears the pin numbering system in use.

Procedures

1. Build the circuit



2. Program

C user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/01_blinkingLed/blinkingLed.c)

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

#define  LedPin    0

int main(void)
{
    if(wiringPiSetup() == -1){ //when initialize wiringPi failed, print message to screen
        printf("setup wiringPi failed !\n");
        return -1;
    }

    pinMode(LedPin, OUTPUT);

    while(1){
        digitalWrite(LedPin, LOW);   //led on
        printf("led on...\n");
        delay(500);
        digitalWrite(LedPin, HIGH);  //led off
        printf("...led off\n");
        delay(500);
    }

    return 0;
}
```

2.2 Compile the program

```
$ gcc blinkingLed.c -o led -lwiringPi
```

Note : The parameter '-o' is to specify a file name for the compiled executable program. If you do not use this parameter, the default file name is *a.out*.

2.3 Run the program

```
$ sudo ./led
```

Python user:

2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/01_blinkingLed_1.py)

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time

LedPin = 11    # pin11

def setup():
    GPIO.setmode(GPIO.BOARD)       # Numbers GPIOs by physical location
    GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode is output
    GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led

def loop():
    while True:
        print '...led on'
        GPIO.output(LedPin, GPIO.LOW)  # led on
        time.sleep(0.5)
        print 'led off...'
        GPIO.output(LedPin, GPIO.HIGH) # led off
        time.sleep(0.5)

def destroy():
    GPIO.output(LedPin, GPIO.HIGH)     # led off
    GPIO.cleanup()                     # Release resource

if __name__ == '__main__':     # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # When 'Ctrl+C' is pressed, the child program destroy() will
be  executed.
        destroy()
```

2.2 Run the program

```
$ sudo python 01_blinkingLed_1.py
```

Press Enter, and then you can see the LED is blinking.