Lesson 4 Controlling an LED with a button

Overview

In this lesson, we will learn how to detect the status of a button, and then toggle the status of LED based on the status of the button.

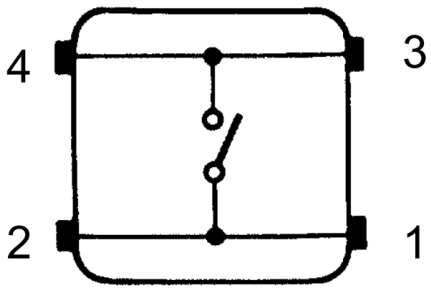
Requirement

- 1* Raspberry Pi
- 1* Button
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several Jumper wires

Principle

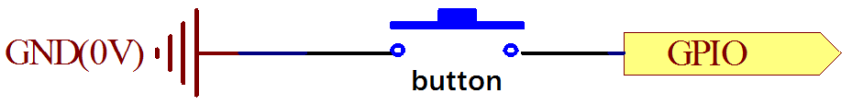
1. Button

Buttons are a common component used to control electronic devices. They are usually used as switches to connect or disconnect circuits. Although buttons come in a variety of sizes and shapes, the one used in this experiment will be a 12mm button as shown in the following pictures. Pins pointed out by the arrows of same color are meant to be connected.

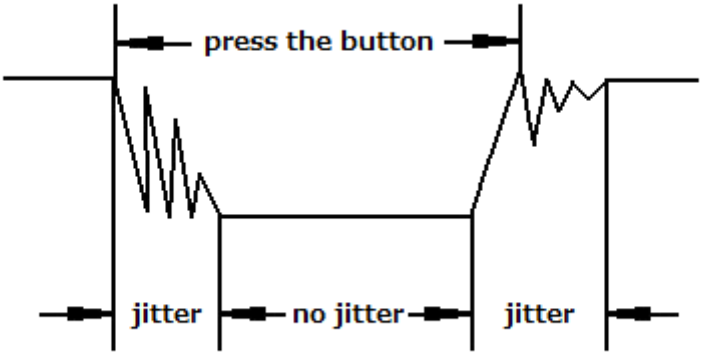


The button we used is a normally open type button. The two contacts of a button is in the off state under the normal conditions, only when the button is pressed they are closed.

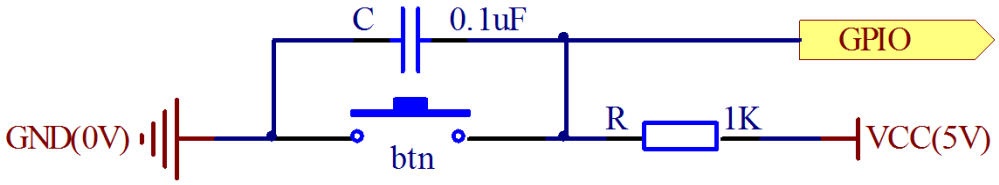
The schematic diagram we used is as follows:



The button jitter must be happen in the process of using. The jitter waveform is as the flowing:



Each time you press the button, the Raspberry Pi will think you have pressed the button many times due to the jitter of the button. We must to deal with the jitter of buttons before we use the button. We can remove the jitter of buttons through the software programming, besides, we can use a capacitance to remove the jitter of buttons. Here we introduce the software method. First, we detect whether the level of button interface is low level or high level. When the level we detected is low level, 5~10 MS delay is needed, and then detect whether the level of button interface is low or high. If the signal is low, we can confirm that the button is pressed once. You can also use a 0.1uF capacitance to clean up the jitter of buttons. The schematic diagram is shown in below:



2. interrupt

Hardware interrupts were introduced as a way to reduce wasting the processor's valuable time in polling loops, waiting for external events. They may be implemented in hardware as a distinct system with control lines, or they may be integrated into the memory subsystem.

3. Key functions:

C user:

- **void pullUpDnControl (int pin, int pud)**

This sets the pull-up or pull-down resistor mode on the given pin, which should be set as an input. Unlike the Arduino, the BCM2835 has both pull-up a down internal resistors. The parameter pud should be; PUD_OFF, (no pull up/down), PUD_DOWN (pull to ground) or PUD_UP (pull to 3.3v). The internal pull up/down resistors have a value of approximately 50KΩ on the Raspberry Pi.

This function has no effect on the Raspberry Pi's GPIO pins when in Sys mode. If you

need to activate a pull-up/pull-down, then you can do it with the gpio program in a script before you start your program.

- int digitalRead (int pin)

This function returns the value read at the given pin. It will be HIGH or LOW (1 or 0) depending on the logic level at the pin.

- int wiringPiISR (int pin, int edgeType, void (*function)(void))

This function registers a function to received interrupts on the specified pin. The edgeType parameter is either **INT_EDGE_FALLING**, **INT_EDGE_RISING**, **INT_EDGE_BOTH** or **INT_EDGE_SETUP**. If it is **INT_EDGE_SETUP** then no initialisation of the pin will happen – it's assumed that you have already setup the pin elsewhere (e.g. with the gpio program), but if you specify one of the other types, then the pin will be exported and initialised as specified. This is accomplished via a suitable call to the gpio utility program, so it need to be available.

The pin number is supplied in the current mode – native wiringPi, BCM_GPIO, physical or Sys modes.

This function will work in any mode, and does not need root privileges to work.

The function will be called when the interrupt triggers. When it is triggered, it's cleared in the dispatcher before calling your function, so if a subsequent interrupt fires before you finish your handler, then it won't be missed. (However it can only track one more interrupt, if more than one interrupt fires while one is being handled then they will be ignored)

This function is run at a high priority (if the program is run using sudo, or as root) and executes concurrently with the main program. It has full access to all the global variables, open file handles and so on.

Python user:

- GPIO.input(channel)

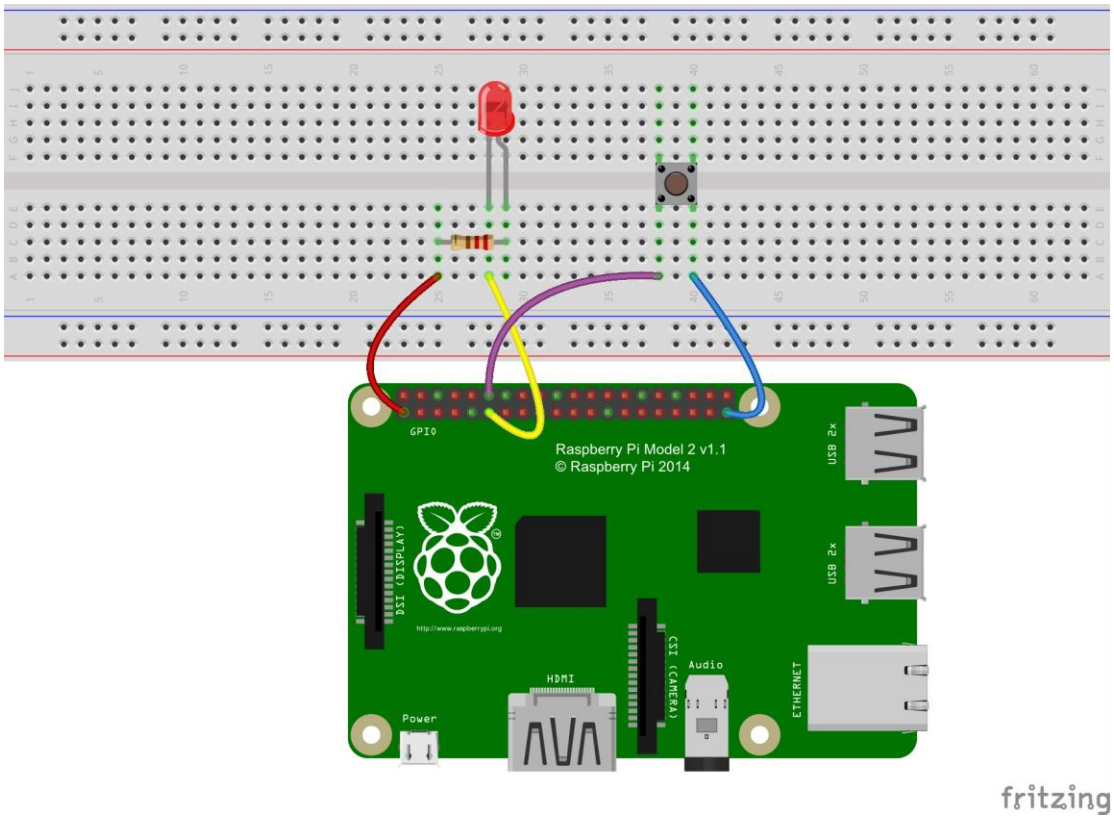
This is used for reading the value of a GPIO pin. Where channel is the channel number based on the numbering system you have specified (BOARD or BCM)). This will return either 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.

- GPIO.add_event_detect(channel, mode)

The event_detected( ) function is designed to be used in a loop with other things, but unlike polling it is not going to miss the change in state of an input while the CPU is busy working on other things. This could be useful when using something like Pygame or PyQt where there is a main loop listening and responding to GUI events in a timely basis.

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/04_btnAndLed/btnAndLed_2.c)

2.2 Compile the program

```
$ gcc btnAndLed_2.c -o btnAndLed -lwiringPi
```

2.3 Run the program

```
$ sudo ./btnAndLed
```

Python user:

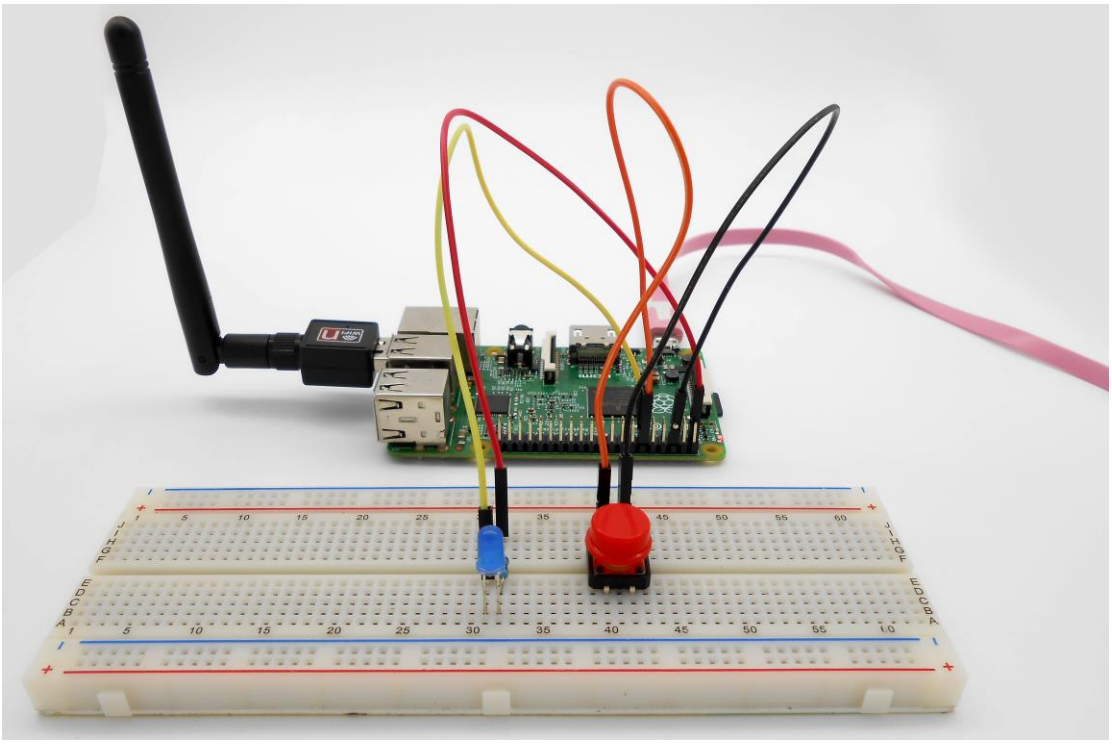
2.1 Edit and save the code with vim or nano.

(Code path: /home/Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi/04_btnAndLed_1.py)

2.2 Run the program

```
$ sudo python 04_btnAndLed_1.py
```

Now, when you press the button, you can see the state of the LED will be toggled. (ON->OFF, OFF->ON).



Summary

Through this lesson, you should have learned how to use the Raspberry Pi detect an external button status, and then toggle the status of LED relying on the status of the button detected before.