Adeept

RFID Learning kit for Raspberry Pi

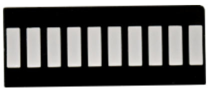
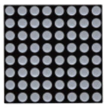
Sharing Perfects Innovation

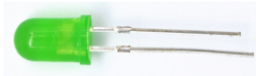
GitHub python™

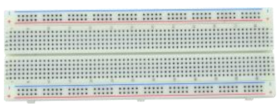
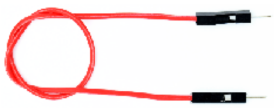
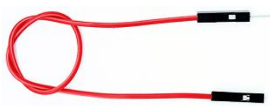
Components List

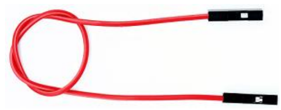
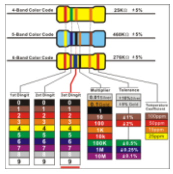
NO.	Name	Picture	Qty
1	RC522 RFID Module		1
2	ID Card		1
3	Special-shaped ID Card		1
4	LCD1602		1
5	PS2 Joystick Module		1
6	Ultrasonic Distance Sensor Module		1
7	ADXL345 (Triaxial Accelerometer Sensor Module)		1
8	DHT-11 (Digital Temperature & Humidity Sensor)		1
9	ADC0832		1

10	Stepper Motor		1
11	ULN2003-based Stepper Motor Driver		1
12	L9110 Motor Driver		1
13	DC Motor		1
14	4*4 Matrix Keyboard		1
15	Breadboard Power Supply Module		1
16	40 pin GPIO Extension Board		1
17	40 pin GPIO Cable		1
18	Light Sensor (Photoresistor)		2
19	Analog Temperature Sensor(Thermistor)		1

20	Relay		1
21	Active Buzzer		1
22	Passive Buzzer		1
23	7-segment Display		1
24	4-digit 7-segment Display		1
25	LED Bar Graph Display		1
26	Dot-matrix Display		1
27	74HC595		2
28	Tilt Switch		1
29	Switch		2
30	RGB LED		1

31	Red LED		8
32	Green LED		4
33	Yellow LED		4
34	Blue LED		4
35	Button (large)		4
36	Button (small)		5
37	Button cap (red)		1
38	Button cap (white)		1
39	Button cap (blue)		2
40	Resistor(220Ω)		16
41	Resistor(1kΩ)		10

42	Resistor(10kΩ)		5
43	Potentiometer (10KΩ)		2
44	Capacitor (104)		5
45	Capacitor (10uF)		2
46	1N4148 Diode		2
47	1N4001 Diode		2
48	NPN Transistor (8050)		4
49	PNP Transistor (8550)		4
50	Breadboard		1
51	Male to Male Jumper Wires		40
52	Male to Female Jumper Wires		20

53	Female to Female Jumper Wires		20
54	Header (40pin)		1
55	Band Resistor Card		1

Preface

About This Kit

This is an RFID learning kit for Raspberry Pi, which includes an RC522 RFID module, some common electronic components and sensors. Through the study, you will get a better understanding of RFID and Raspberry Pi, and be able to make fascinating works based on Raspberry Pi.

About Adeept

Adeept is a technical service team of open source software and hardware. Dedicated to applying the Internet and the latest industrial technology in open source area, we strive to provide best hardware support and software service for general makers and electronic enthusiasts around the world. We aim to create infinite possibilities with sharing. No matter what field you are in, we can lead you into the electronic world and bring your ideas into reality.

If you have any problems for learning, please contact us at support@adeept.com, or please ask questions in our forum www.adeept.com. We will do our best to help you solve the problem.

Content

About the Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything you'd expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games.

What's more, the Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. We want to see the Raspberry Pi being used by kids all over the world to learn to program and understand how computers work.

Learn more at:

https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

Raspberry Pi Pin Numbering Introduction

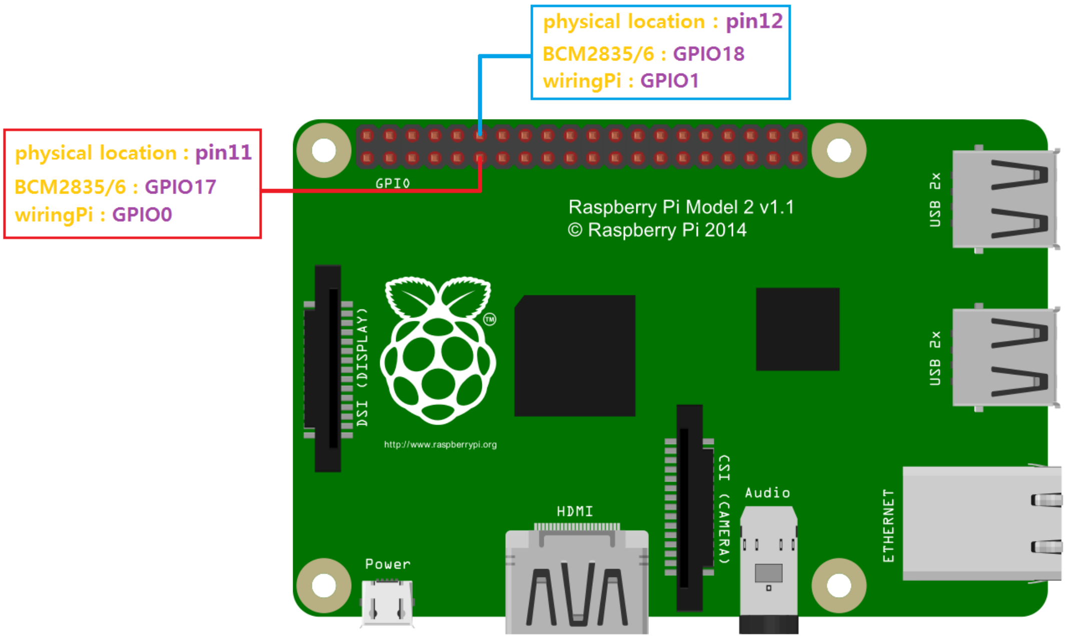
WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin
–	–	3.3v	1 I 2	5v	–	–
8	R1:0/R2:2	SDA1	3 I 4	5v	–	–
9	R1:1/R2:3	SCL1	5 I 6	0V	–	–
7	4	GPIO7	7 I 8	TXD	14	15
–	–	0V	9 I 10	RXD	15	16
0	17	GPIO0	11 I 12	GPIO1	18	1
2	R1:21/R2:27	GPIO2	13 I 14	0V	–	–
3	22	GPIO3	15 I 16	GPIO4	23	4
–	–	3.3v	17 I 18	GPIO5	24	5
12	10	MOSI	19 I 20	0V	–	–
13	9	MISO	21 I 22	GPIO6	25	6
14	11	SCLK	23 I 24	CE0	8	10
–	–	0V	25 I 26	CE1	7	11
30	0	SDA0	27 I 28	SCL0	1	31
21	5	GPIO21	29 I 30	0V	–	–
22	6	GPIO22	31 I 32	GPIO26	12	26
23	13	GPIO23	33 I 34	0V	–	–
24	19	GPIO24	35 I 36	GPIO27	16	27
25	26	GPIO25	37 I 38	GPIO28	20	28
		0V	39 I 40	GPIO29	21	29
WiringPi Pin	BCM GPIO	Name	Header	Name	BCM GPIO	WiringPi Pin

There are three methods for numbering Raspberry Pi's GPIO:

1. Numbering according to the physical location of the pins, from left to right, top to bottom – the left is odd and the right is even.

2. Numbering according the GPIO registers of BCM2835/2836/7 SOC.

3. Numbering according the GPIO library wiringPi.


physical location : pin12
BCM2835/6 : GPIO18
wiringPi : GPIO1
physical location : pin11
BCM2835/6 : GPIO17
wiringPi : GPIO0
GPIO
Raspberry Pi Model 2 v1.1
© Raspberry Pi 2014
USB 2x
USB 2x
DSI (DISPLAY)
http://www.raspberrypi.org
CSI (CAMERA)
Audio
HDMI
Power
ETHERNET

Raspberry Pi GPIO Library Introduction

Currently, there are two major GPIO libraries for Raspberry Pi: RPi.GPIO and wiringPi.

RPi.GPIO:

RPi.GPIO is a python module to control Raspberry Pi GPIO channels. For more information, please visit:

https://pypi.python.org/pypi/RPi.GPIO/

For examples and documentation:
http://sourceforge.net/p/raspberry-gpio-python/wiki/Home/

The RPi.GPIO module is pre-installed in the official Raspbian operating system, thus you can use it directly.

wiringPi:

The wiringPi is a GPIO access library written in C for the BCM2835/6/7 SOC used in the Raspberry Pi. It's released under the GNU LGPLv3 license and is usable from C and C++ and many other languages with suitable wrappers. It's designed to be familiar to people who have used the Arduino "wiring" system.

For more information about wiringPi, please visit: http://wiringpi.com/

Install wiringPi:

Step 1: Get the source code

```
$ sudo git clone git://git.drogon.net/wiringPi
```

Step 2: Compile and install

```
$ cd wiringPi
$ git pull origin
$ sudo ./build
```

Press Enter and the script *build* will automatically compile wiringPi source code and then install it to the Raspberry Pi.

Next, verify whether the wiringPi is installed successfully or not:

wiringPi includes a command-line utility gpio which can be used to program and set up the GPIO pins. You can use it to read and write the pins or even control them from shell scripts.

You can verify whether the wiringPi is installed successfully or not by the following commands:

$ sudo gpio -v

```
pi@raspberrypi ~ $ sudo gpio -v
gpio version: 2.26
Copyright (c) 2012-2015 Gordon Henderson
This is free software with ABSOLUTELY NO WARRANTY.
For details type: gpio -warranty

Raspberry Pi Details:
  Type: Model 2, Revision: 1.1, Memory: 1024MB, Maker: Sony
pi@raspberrypi ~ $
```

$ sudo gpio readall

```
pi@raspberrypi ~ $ sudo gpio readall
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 |     |     |    3.3v |      |   |  1 || 2  |   |      | 5v      |     |     |
 |   2 |   8 |   SDA.1 | ALT0 | 1 |  3 || 4  |   |      | 5V      |     |     |
 |   3 |   9 |   SCL.1 | ALT0 | 1 |  5 || 6  |   |      | 0v      |     |     |
 |   4 |   7 | GPIO. 7 |   IN | 1 |  7 || 8  | 1 | ALT0 | TxD     | 15  | 14  |
 |     |     |      0v |      |   |  9 || 10 | 1 | ALT0 | RxD     | 16  | 15  |
 |  17 |   0 | GPIO. 0 |   IN | 0 | 11 || 12 | 0 | IN   | GPIO. 1 | 1   | 18  |
 |  27 |   2 | GPIO. 2 |   IN | 0 | 13 || 14 |   |      | 0v      |     |     |
 |  22 |   3 | GPIO. 3 |   IN | 0 | 15 || 16 | 0 | IN   | GPIO. 4 | 4   | 23  |
 |     |     |    3.3v |      |   | 17 || 18 | 0 | IN   | GPIO. 5 | 5   | 24  |
 |  10 |  12 |    MOSI | ALT0 | 0 | 19 || 20 |   |      | 0v      |     |     |
 |   9 |  13 |    MISO | ALT0 | 0 | 21 || 22 | 0 | IN   | GPIO. 6 | 6   | 25  |
 |  11 |  14 |    SCLK | ALT0 | 0 | 23 || 24 | 1 | ALT0 | CE0     | 10  | 8   |
 |     |     |      0v |      |   | 25 || 26 | 1 | ALT0 | CE1     | 11  | 7   |
 |   0 |  30 |   SDA.0 |   IN | 1 | 27 || 28 | 1 | IN   | SCL.0   | 31  | 1   |
 |   5 |  21 | GPIO.21 |   IN | 1 | 29 || 30 |   |      | 0v      |     |     |
 |   6 |  22 | GPIO.22 |   IN | 1 | 31 || 32 | 0 | IN   | GPIO.26 | 26  | 12  |
 |  13 |  23 | GPIO.23 |   IN | 0 | 33 || 34 |   |      | 0v      |     |     |
 |  19 |  24 | GPIO.24 |   IN | 0 | 35 || 36 | 0 | IN   | GPIO.27 | 27  | 16  |
 |  26 |  25 | GPIO.25 |   IN | 0 | 37 || 38 | 0 | IN   | GPIO.28 | 28  | 20  |
 |     |     |      0v |      |   | 39 || 40 | 0 | IN   | GPIO.29 | 29  | 21  |
 +-----+-----+---------+------+---+----++----+---+------+---------+-----+-----+
 | BCM | wPi |   Name  | Mode | V | Physical | V | Mode | Name    | wPi | BCM |
 +-----+-----+---------+------+---+---Pi 2---+---+------+---------+-----+-----+
pi@raspberrypi ~ $
```

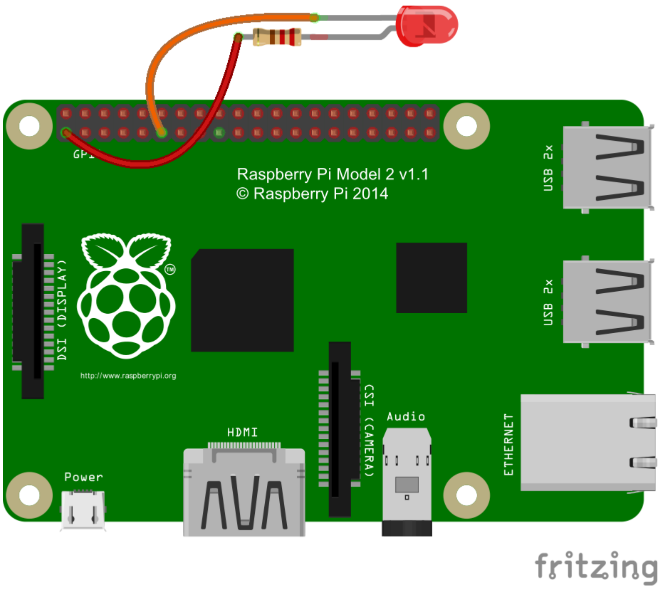
If the information above is shown, it indicates that the wiringPi has been installed successfully.

How to Use wiringPi and RPi.GPIO

For how to use the wiringPi C library and RPi.GPIO Python module, here we take blinking an LED for example.

Step 1: Build the circuit according to the following schematic diagram

Note: Resistance = 220Ω



For Python users:

Step 2: Create a file named *led.py*

$ sudo touch led.py

```
pi@raspberrypi /home $ ls
pi
pi@raspberrypi /home $ sudo touch led.py
pi@raspberrypi /home $ ls
led.py  pi
pi@raspberrypi /home $
```

Step 3: Open the file *led.py* with vim or nano

$ sudo vim led.py

Write the following source code, then save and exit.

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

Led = 11    # pin11

def setup():
    GPIO.setmode(GPIO.BOARD)    # Numbering according to the physical location
    GPIO.setup(Led, GPIO.OUT)   # Set pin mode as output
    GPIO.output(Led, GPIO.HIGH) # Output high level(+3.3V) to off the led

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

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

if __name__ == '__main__':     # Program start from here
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press 'Ctrl+C' to end the program
        destroy()
```

Step 4: Run

$ sudo python led.py

```
pi@raspberrypi /home $ ls
led.py  pi
pi@raspberrypi /home $ sudo python led.py
...led on
led off...
...led on
led off...
...led on
led off...
```

Now you should see the LED blinking. Press **Ctrl**+**C** and the program execution will be terminated.

For C language users:

Step 2: Create a file named *led.c*

$ sudo touch led.c

```
pi@raspberrypi /home $ ls
led.py  pi
pi@raspberrypi /home $ sudo touch led.c
pi@raspberrypi /home $
```

Step 3: Open the file *led.c* with vim or nano

$ sudo vim led.c

Write the following source code, then save and exit.

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

#define  Led    0  //wiringPi GPIO0, pin11

int main(void)
{
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !\n");
        return -1;
    }

    pinMode(Led, OUTPUT);

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

    return 0;
}
```

Step 4: Compile the code

$ sudo gcc led.c -lwiringPi

```
pi@raspberrypi /home $ ls
led.c  led.py  pi
pi@raspberrypi /home $ sudo gcc led.c -lwiringPi
pi@raspberrypi /home $
```

After the command is executed, you'll find a file named *a.out* appear in the current directory. It is an executable program.

```
pi@raspberrypi /home $ ls
a.out  led.c  led.py  pi
pi@raspberrypi /home $
```

Step 5: Run

$ sudo ./a.out

```
pi@raspberrypi /home $ sudo ./a.out
led on...
...led off
led on...
...led off
```

Now you should see that the LED is blinking. Press **Ctrl**+**C** and the program execution will be terminated.

Resources:

http://sourceforge.net/p/raspberry-gpio-python/wiki/Examples/

http://wiringpi.com/reference/

NOTE:

Before you continue learning, please copy the source code provided with the kit to your Raspberry Pi's */home/* directory, or download the source code directly from our github repository:

C Language Source Code:

$ git clone https://github.com/adeept/Adeept_RFID_Learning_Kit_C_Code_for_RPi.git

Python Source Code:

$ git clone https://github.com/adeept/Adeept_RFID_Learning_Kit_Python_Code_for_RPi.git

Lesson 1 Blinking LED

Overview

In this tutorial, we will start the journey of learning Raspberry Pi. To begin with simple experiments, we will first learn how to control an LED.

Components

- 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 level (+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. An LED has two electrodes: a positive electrode and a negative electrode. It lights up only when a forward current passes, and it can flash red, blue, green or yellow, etc. The color of the light depends on the materials it is 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.

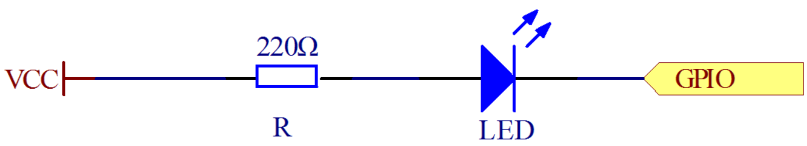
2. What is resistor?

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

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

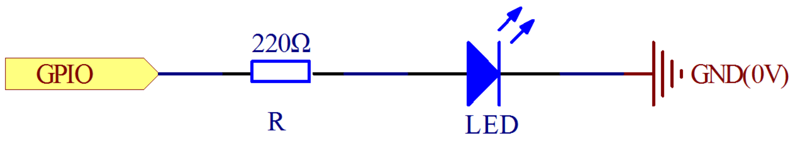
There are two methods for connecting an LED with Raspberry Pi GPIO:

①



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

②



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

The resistance of a 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:

$$R = U / I = 3.3V / (5\sim20mA) = 165\Omega\sim660\Omega$$

In this experiment, we use a 220ohm resistor.

The experiment is made based on method ① – use pin 11 of Raspberry Pi to control an LED. When pin 11 of Raspberry Pi is programmed to output low level, the LED will light up. Next, delay for some time. And then program pin 11 to high level to make the LED off. Repeat the above process and you can get a blinking LED then.

3. Key functions

For C language users:

- int wiringPiSetup (void)

The function must be called at the start of your program, or your program will fail to work.

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

- void pinMode (int pin, int mode)

This function 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.

The 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 starting your program.

- void digitalWrite (int pin, int value)

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

- void delay (unsigned int howLong)

This function 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.

For Python users:

- GPIO.setmode(GPIO.BOARD)

There are two ways of numbering the IO pins on a Raspberry Pi within RPi.GPIO. One is to use the **BOARD** numbering system. It refers to the pin numbers on the P1 header of the Raspberry Pi board. The advantage of 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 other numbering system is by 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 about 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)

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

- GPIO.output(channel, state)

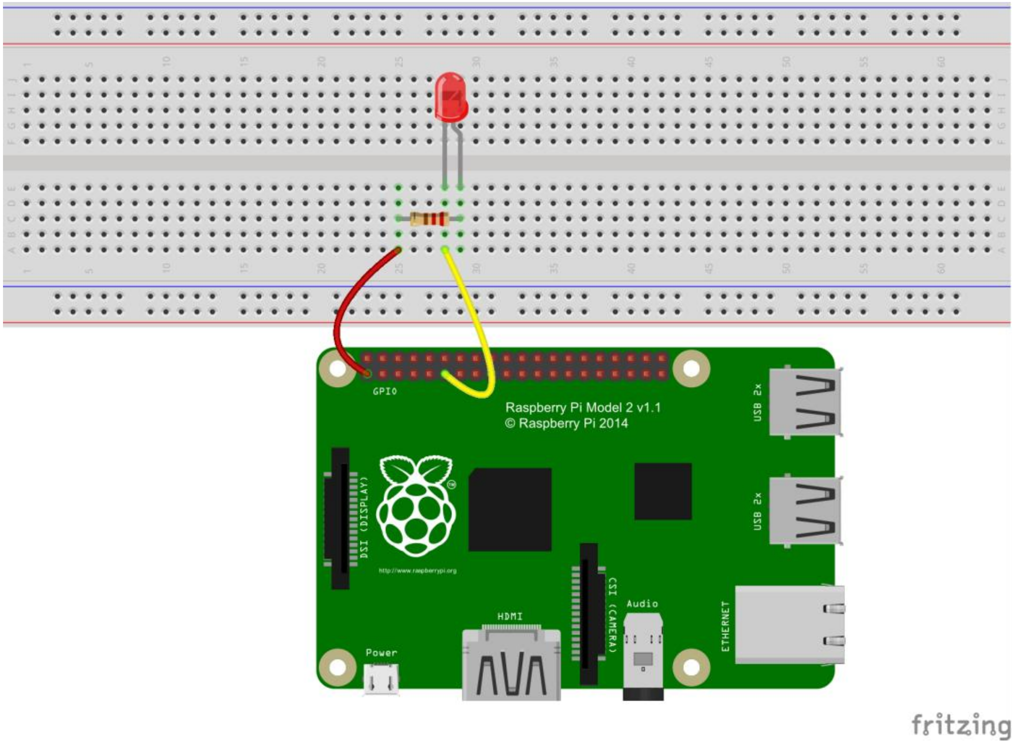
This function sets the output state of a GPIO pin. The argument **channel** is the channel number based on the numbering system you have specified (BOARD or BCM). **State** can be **0** / **GPIO.LOW** / **False** or **1** / **GPIO.HIGH** / **True**.

- GPIO.cleanup( )

At the end any program, it is a good habit to clean up all the resources you might have used. This is no different from RPi.GPIO. By returning all channels you have used back to input without pull up/down, you can avoid accidental damage to your RPi caused by pin short out. Note that this will only clean up GPIO channels that your script has ever used. And it also clears the pin numbering system in use.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_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;
}
```

Step 3: Compile

$ 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*.

Step 4: Run

$ sudo ./led

For Python users:

Step 2: Edit and save the code with vim or nano.

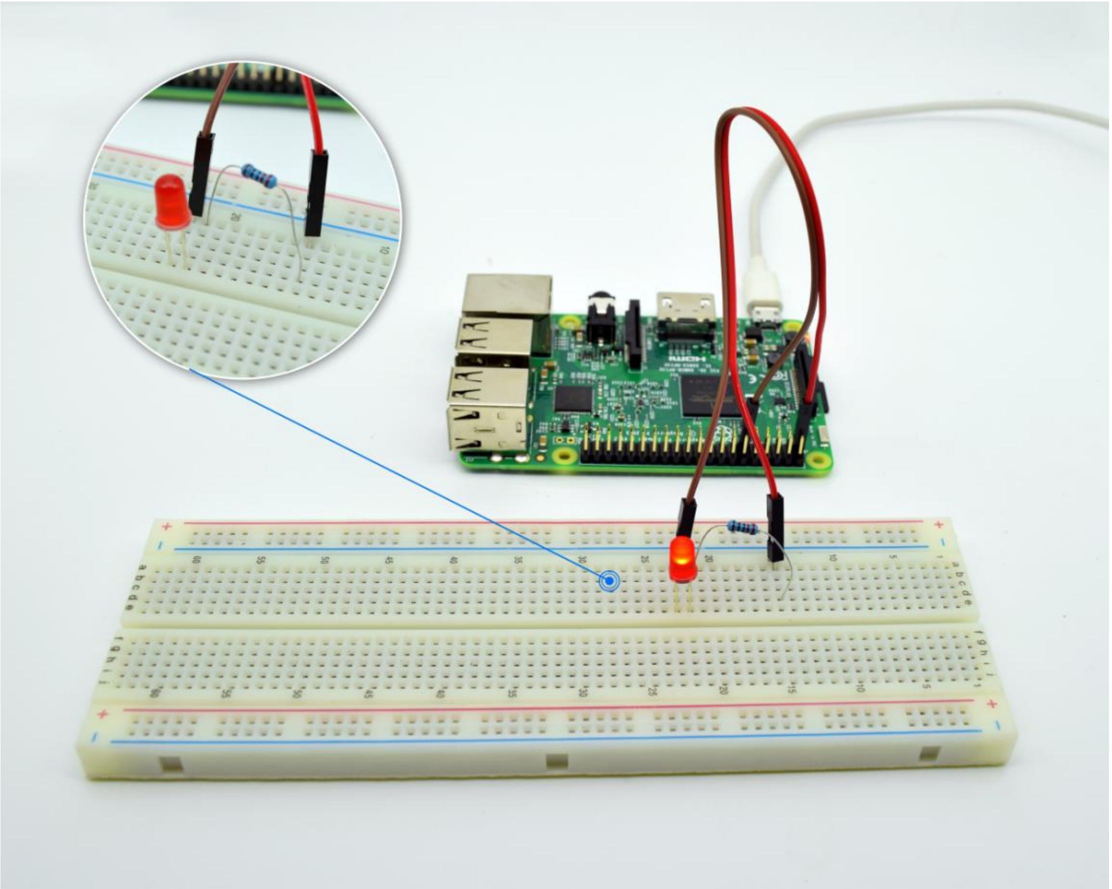
(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/01_blinkingLed_1.py)

```
#!/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()
```

Step 3: Run

$ sudo python 01_blinkingLed_1.py

Now you can see the LED blinking.



Lesson 2 Active Buzzer

Overview

In this lesson, we will learn how to program the Raspberry Pi to make an active buzzer beep.

Components

- 1* Raspberry Pi
- 1* Active buzzer
- 1* 1 kΩ Resistor
- 1* NPN Transistor (S8050)
- 1* Breadboard
- Several jumper wires

Principle

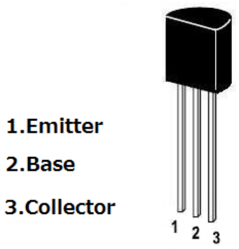
A buzzer or beeper is an audio signaling device. As a type of electronic buzzer with an integrated structure, which uses DC power supply, buzzers are widely used in computers, printers, photocopiers, alarms, electronic toys, automotive electronic equipments, telephones, timers and other electronic products for voice devices. Buzzers can be categorized as active and passive buzzers (See the following pictures).

Place the pins of the buzzer face up, and then you can see the two types of buzzer are different - the buzzer with a green circuit board onside is a passive one.

In this lesson, the buzzer we used is active buzzer. Active buzzers will beep as long as they are powered. We can program to make the Raspberry Pi output alternating high and low levels to make the buzzer beep.

A slightly larger current is needed to make a buzzer beep. However, the output current of Raspberry Pi GPIO is too low, so we need a transistor to help.

The main function of a transistor is to enlarge the voltage or current. It can also be used to control the circuit conduction or deadline. Transistors can be divided into two kinds: NPN, like the S8050 we provided; PNP, like the S8550 provided. The transistor we use is as shown below:



There are two kinds of driving circuit for the buzzer:

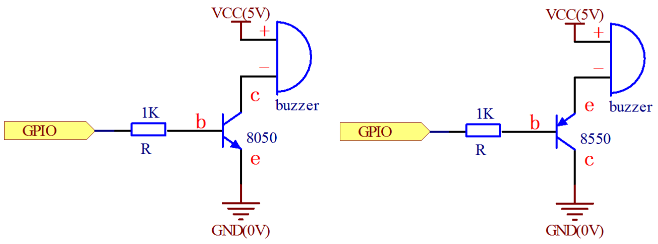


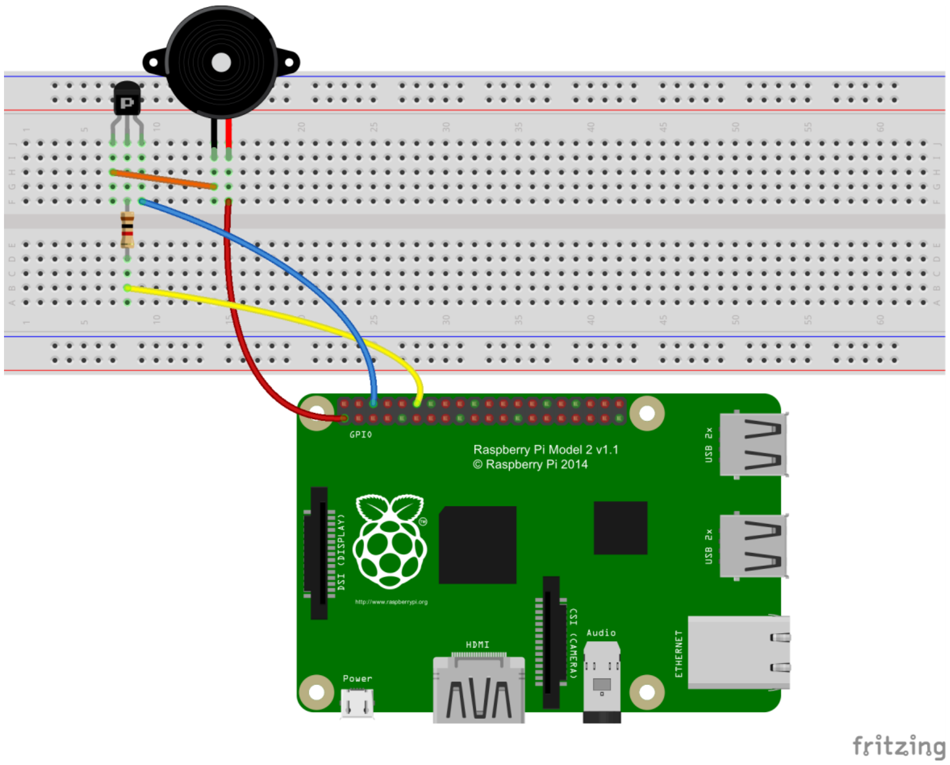
Figure 1 Figure 2

Figure 1: Set the Raspberry Pi GPIO as a high level. Then the transistor S8050 will conduct and the buzzer will make sounds. Set the GPIO as low. And the transistor S8050 will be de-energized and the buzzer stops beeping.

Figure 2: Set the Raspberry Pi GPIO as low level. The transistor S8550 will be energized and the buzzer will beep. Set the GPIO as a high. Then the transistor S8550 will be de-energized and the buzzer beeping will stop.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/02_ativeBuzzer/buzzer.c)

Step 3: Compile

```
$ gcc buzzer.c -o buzzer -lwiringPi
```

Step 4: Run

```
$ sudo ./buzzer
```

For Python users:

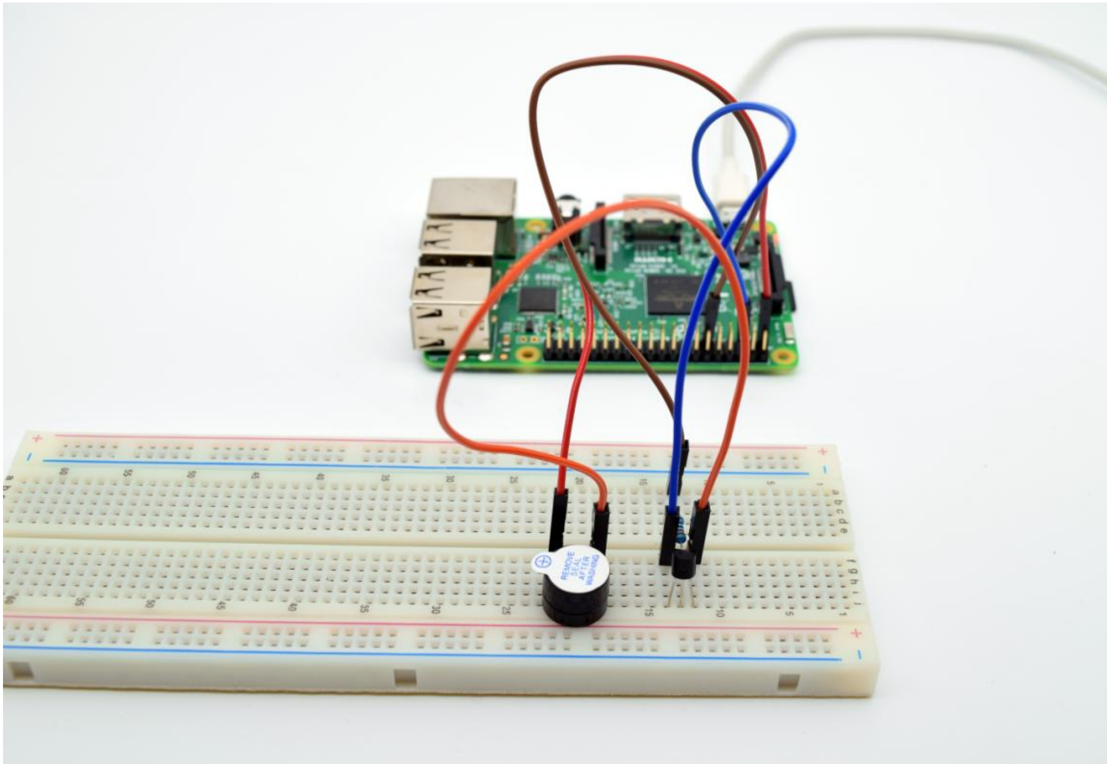
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/02_activeBuzzer.py)

Step 3: Run

```
$ sudo python 02_activeBuzzer.py
```

Now you can hear the buzzer beeping.



Summary

After learning this lesson, you can master the basic principle of the buzzer and transistor. Also you've learned how to program the Raspberry Pi and then control the buzzer. Now you can use what you've learned in this lesson to make some interesting things!

Lesson 3 Passive Buzzer

Overview

In this lesson, we will learn how to program the Raspberry Pi to make a passive buzzer beep with different frequencies.

Components

- 1* Raspberry Pi
- 1* Passive buzzer
- 1* 1 kΩ Resistor
- 1* NPN Transistor (S8050)
- 1* Breadboard
- Several jumper wires

Principle

As long as you send the square wave signals to a passive buzzer with different frequencies, the passive buzzer will make different sounds.

In this experiment, we continuously send different square wave signals to a passive buzzer so as to make it play a piece of music.

Key functions

For C language users:

- **int softToneCreate (int pin)**

This creates a software controlled tone pin. You can use any GPIO pin and the pin numbering will be that of the wiringPiSetup() function you used.

The return value is 0 for success. Anything else and you should check the global errno variable to see what went wrong.

- **void softToneWrite (int pin, int freq)**

This updates the tone frequency value on the given pin. The tone will be played until you set the frequency to 0.

For Python users:

- 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.

- p = GPIO.PWM(channel, frequency)

To create a PWM instance

- p.start(dc)

To start PWM.

- p.ChangeFrequency(freq)

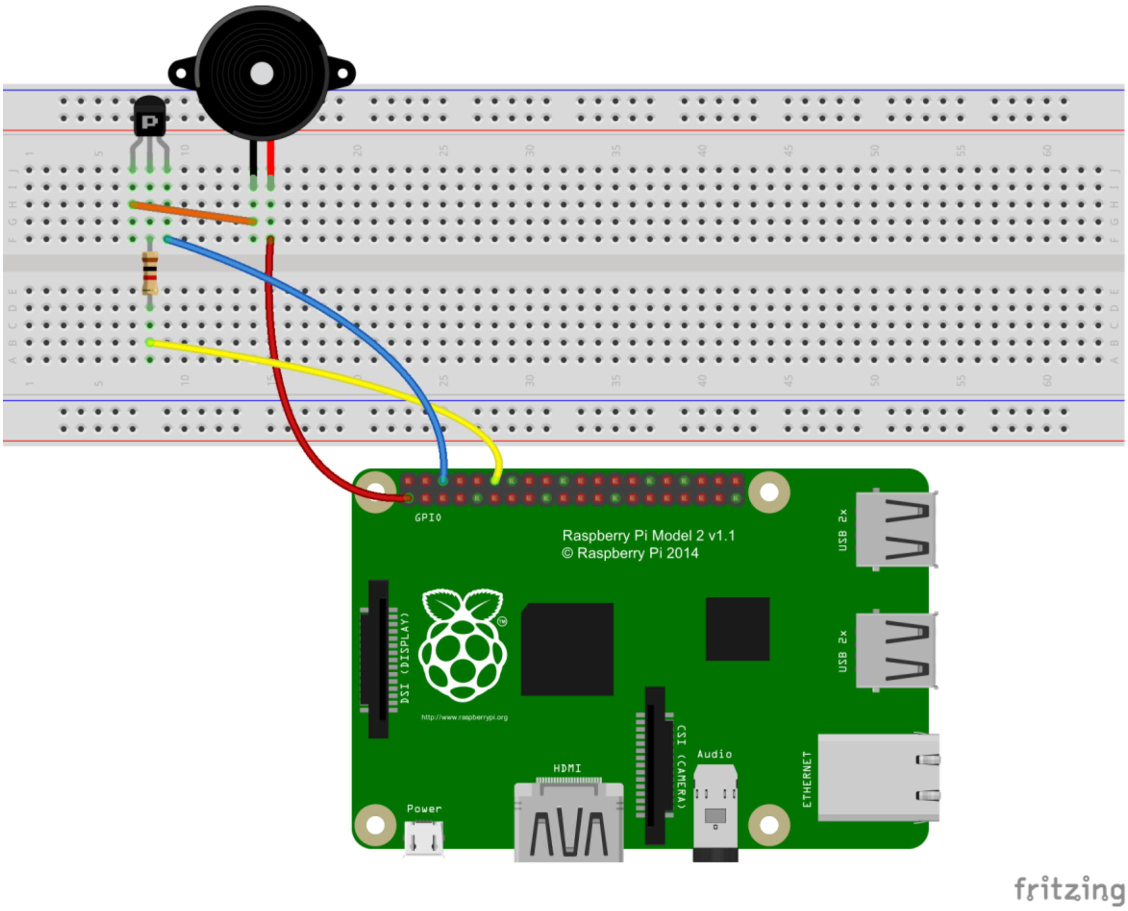
To change the frequency.

- p.stop()

To stop PWM.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/03_passiveBuzzer/passiveBuzzer.c)

Step 3: Compile

```
$ gcc passiveBuzzer.c -o passiveBuzzer –lwiringPi -lpthread
```

Step 4: Run

```
$ sudo ./passiveBuzzer
```

For Python users:

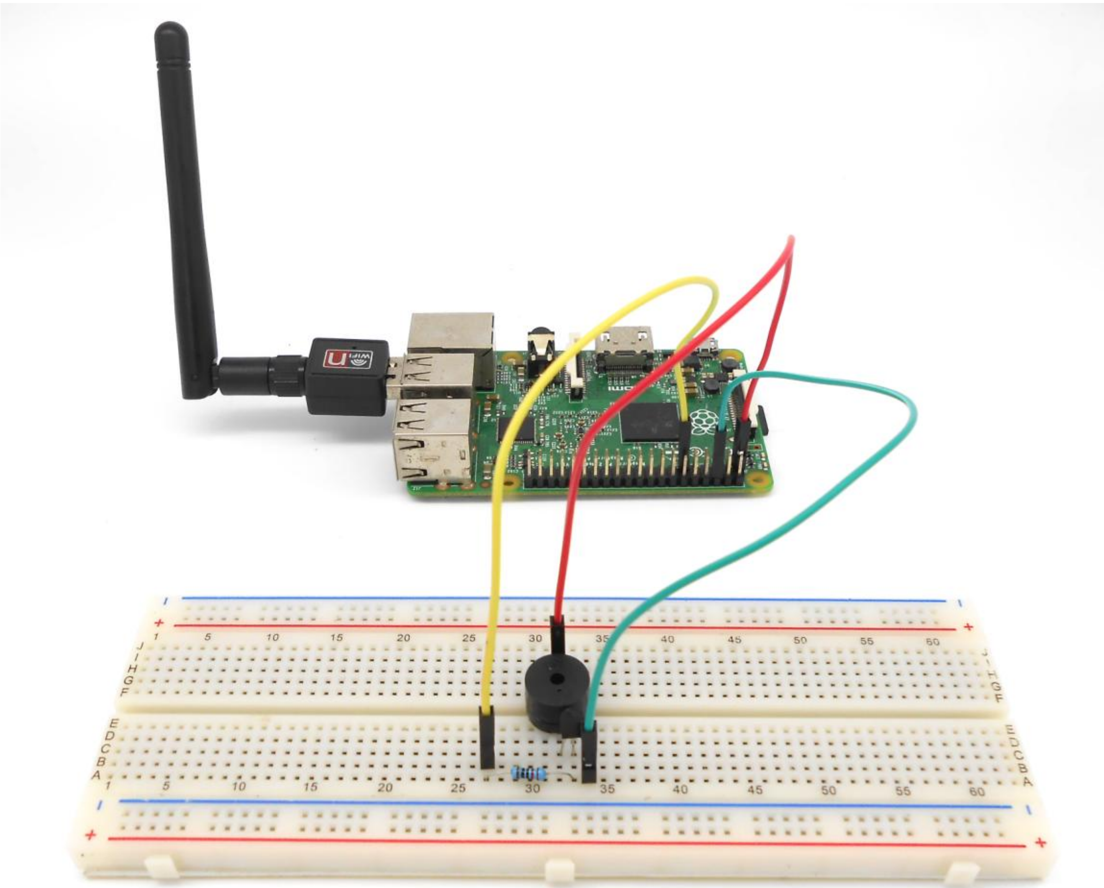
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/03_passiveBuzzer.py)

Step 3: Run

```
$ sudo python 03_passiveBuzzer.py
```

Now, you can hear the sounds of the buzzer – in varying "rhythms".



Lesson 4 Tilt Switch

Overview

In this lesson, we will learn how to use the tilt switch and change the status of an LED by changing the tilt angle of the tilt switch.

Components

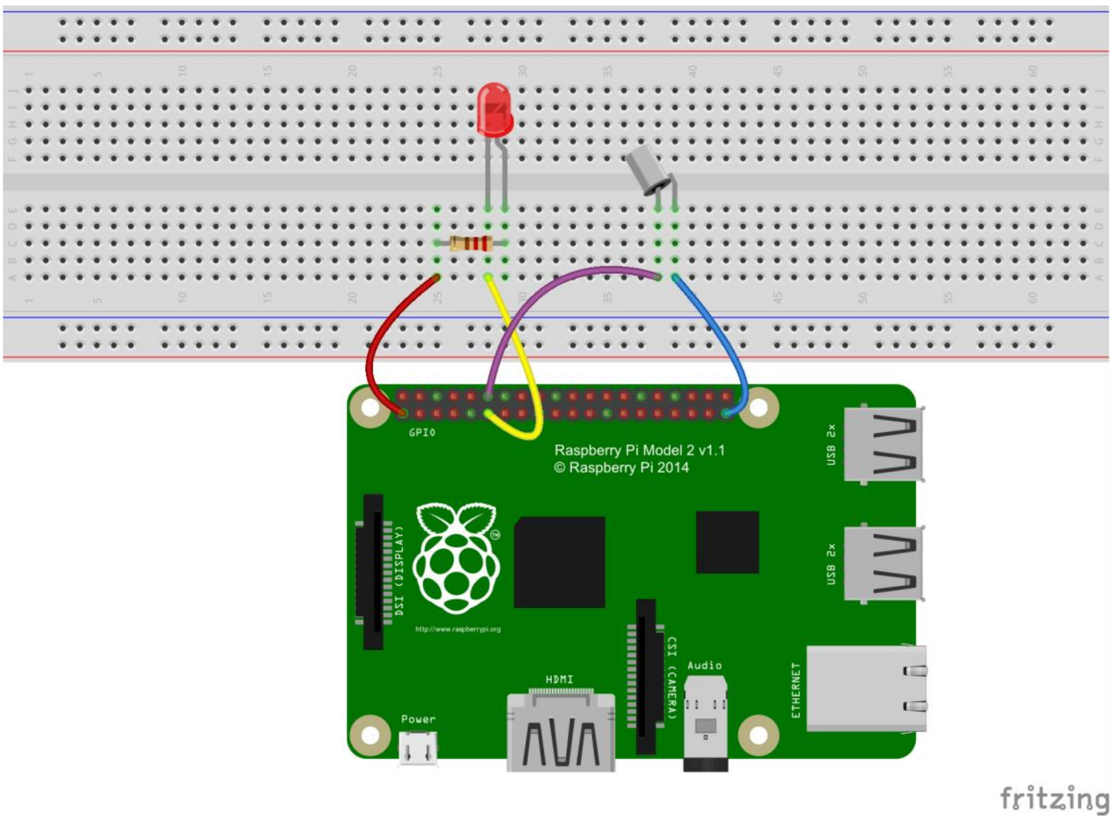
- 1* Raspberry Pi
- 1* Tilt switch
- 1* LED
- 1* 220Ω Resistor
- 1* Breadboard
- Several jumper wires

Principle

The tilt switch is also called ball switch. When the switch is tilted at a specific angle, the contacts will be connected, while tilting the switch back will cause the metallic ball to move away from that set of contacts, thus breaking the circuit.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/04_tiltSwitch/tiltSwitch.c)

Step 3: Compile

```
$ gcc tiltSwitch.c -o tiltSwitch -lwiringPi
```

Step 4: Run

```
$ sudo ./tiltSwitch
```

For Python users:

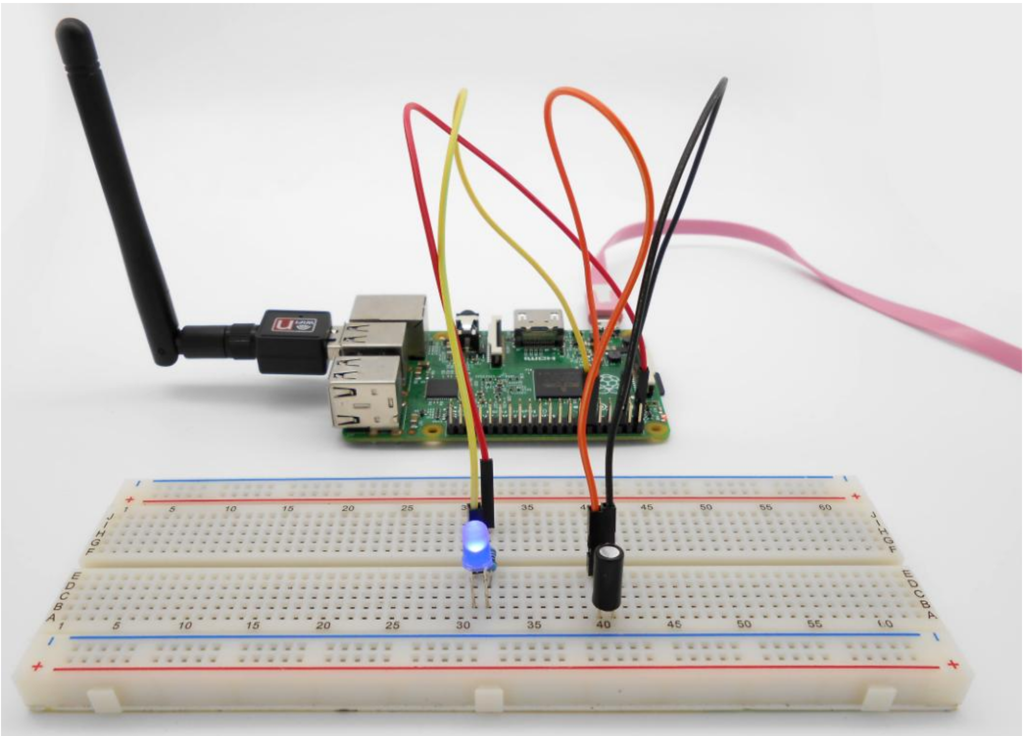
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/04_tiltSwitch.py)

Step 3: Run

```
$ sudo python 04_tiltSwitch.py
```

Now, tilt the breadboard at a certain angle, and you will see the state of LED changed.



Summary

In this lesson, we have learned the principle and application of the tilt switch. It is a very simple electronic component, but simple devices can often make interesting things. Try to make your own works!

Lesson 5 Controlling an LED by Button

Overview

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

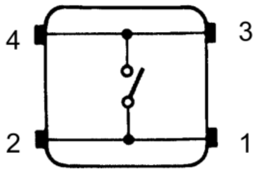
Components

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

Principle

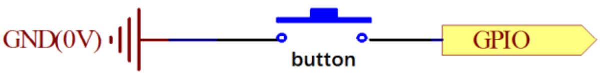
1. Button

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 below.

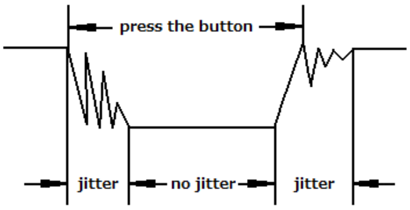


The button we used is a normally open type one. The two contacts of a button are in the off state under the normal conditions; only when the button is pressed they are closed.

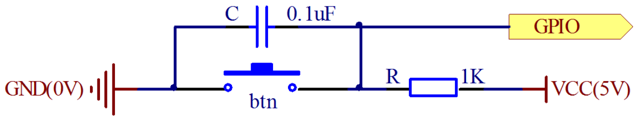
The schematic diagram is as follows:



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



Each time you press the button, the Raspberry Pi will regard you have pressed the button many times due to the jitter of the button. You should deal with the jitter of buttons before using. You can eliminate the jitter through software programming. Besides, you can use a capacitor to solve the issue. Take the software method for example. First, detect whether the level of button interface is low level or high level. If it is low level, 5~10 MS delay is needed. Then detect whether the level of button interface is low or high. If the signal is low, you can infer that the button is pressed once. You can also use a 0.1uF capacitor to avoid the jitter of buttons. The schematic diagram is as shown 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:

For C language users:

- **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.

For Python users:

- **GPIO.input(channel)**

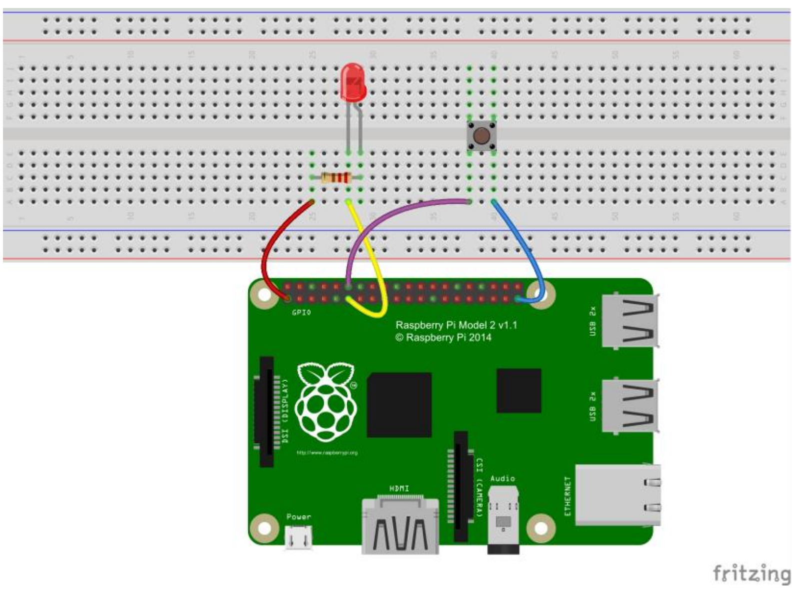
This is used for reading the value of a GPIO pin. Where channel is the channel number based on the numbering system you have specified (BOARD or BCM)). This will return either 0 / GPIO.LOW / False or 1 / GPIO.HIGH / True.

- **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

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/05_btnAndLed/btnAndLed_2.c)

Step 3: Compile

$ gcc btnAndLed_2.c -o btnAndLed -lwiringPi

Step 4: Run

$ sudo ./btnAndLed

For Python users:

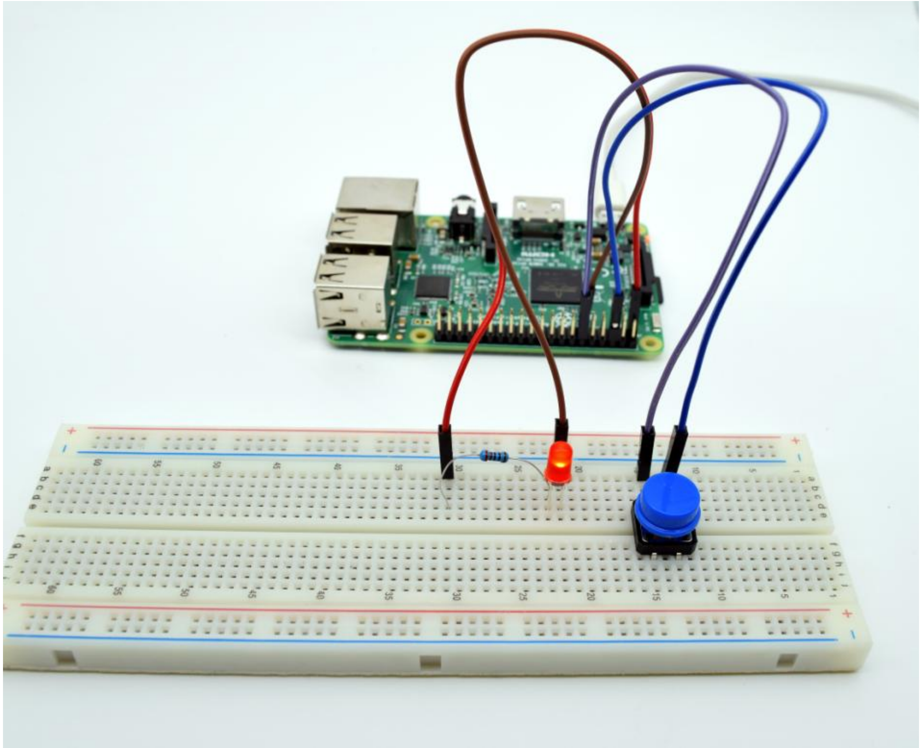
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/05_btnAndLed_1.py)

Step 3: Run

$ sudo python 05_btnAndLed_2.py

Now press the button, and you can see the state of the LED will be toggled between ON and OFF.



Summary

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

Lesson 6 Relay

Overview

In this lesson, we will learn how to control a relay to break or connect a circuit.

Components

- 1* Raspberry Pi
- 1* Relay
- 1* NPN Transistor (S8050)
- 1* Diode (1N4001)
- 1* 1KΩ Resistor
- 1* Breadboard
- Several jumper wires

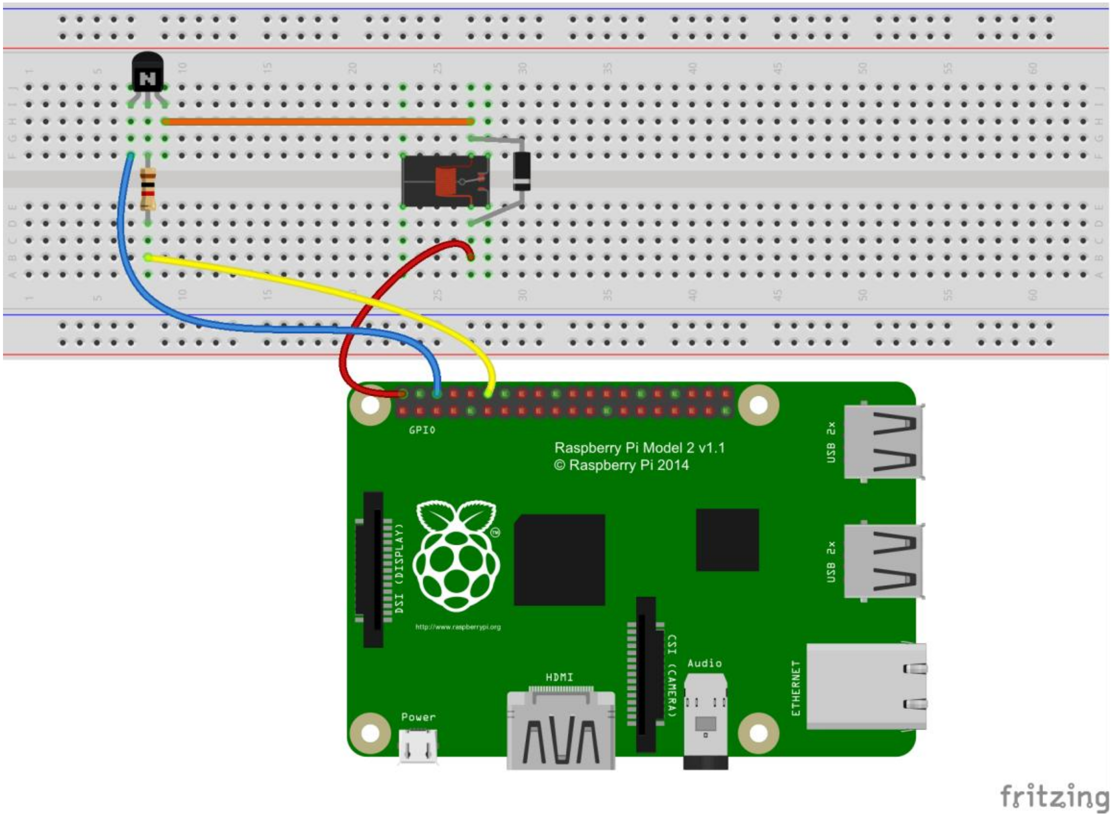
Principle

A relay is an electrically operated switch. It is generally used in automatic control circuit. Actually, it is an "automatic switch" which uses low current to control high current. It plays a role of automatic regulation, security protection and circuit switch. When an electric current is passed through the coil it generates a magnetic field that activates the armature, and the consequent movement of the movable contact (s) either makes or breaks (depending upon construction) a connection with a fixed contact. If the set of contacts was closed when the relay was de-energized, then the movement opens the contacts and breaks the connection, and vice versa if the contacts were open. When the current to the coil is switched off, the armature is returned by a force, approximately half as strong as the magnetic force, to its relaxed position. Usually this force is provided by a spring, but gravity is also used commonly in industrial motor starters. Most relays are manufactured to operate quickly. In a low-voltage application this reduces noise; in a high voltage or current application it reduces arcing.

When the coil is energized with direct current, a diode is often placed across the coil to dissipate the energy from the collapsing magnetic field at deactivation, which would otherwise generate a voltage spike dangerous to semiconductor circuit components.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/06_relay/relay.c)

Step 3: Compile

    $ gcc relay.c -o relay -lwiringPi

Step 4: Run

    $ sudo ./relay

For Python users:

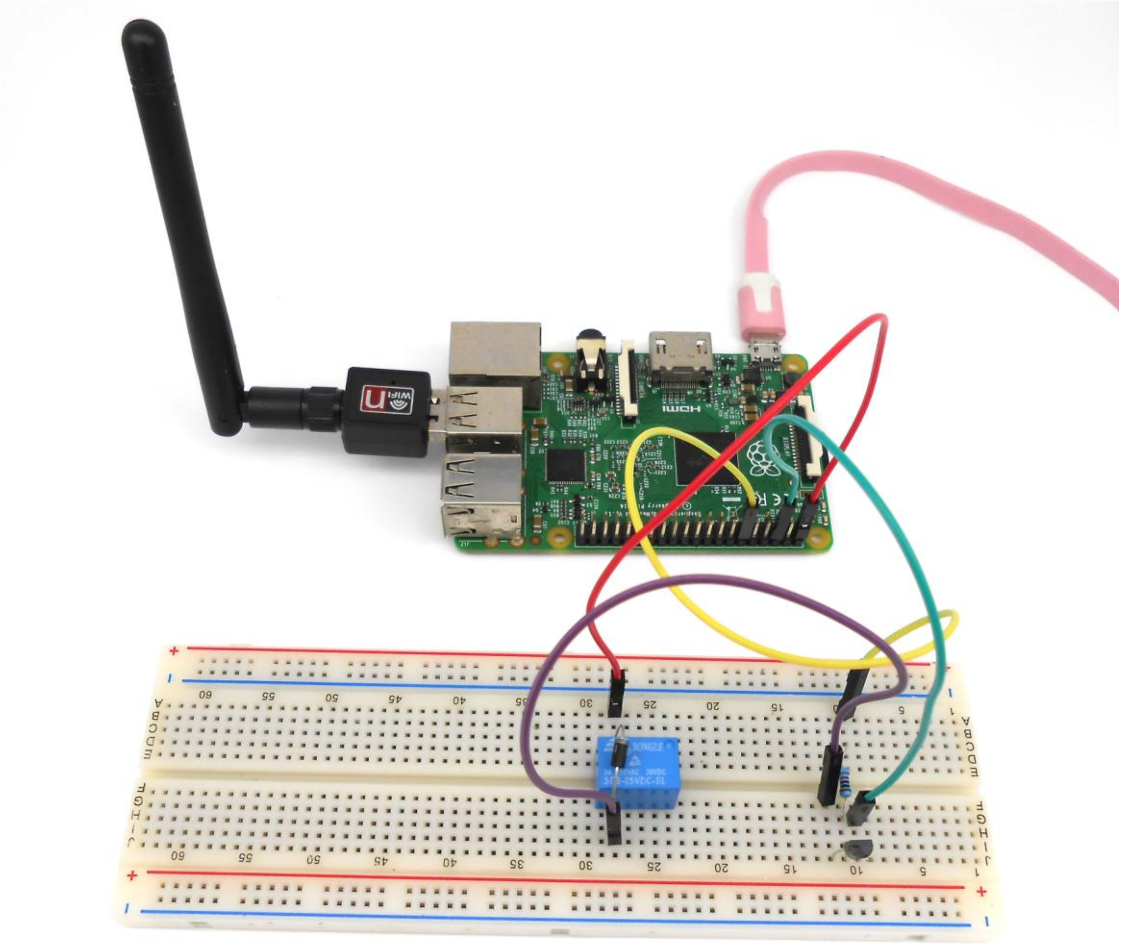
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/06_relay.py)

Step 3: Run

    $ sudo python 06_relay.py

Now you can hear tick-tocks, which are the sounds of relay toggling.



Lesson 7 LED Flowing Lights

Overview

In the first lesson, we have learned how to make an LED blink by programming the Raspberry Pi. Now we will use the Raspberry Pi to control 8 LEDs, to make 8 LEDs show the effects of flowing.

Components

- 1* Raspberry Pi
- 8* LED
- 8* 220Ω Resistor
- 1* Breadboard
- Several jumper wires

Principle

The principle of this experiment is very simple and quite similar with that in the first lesson.

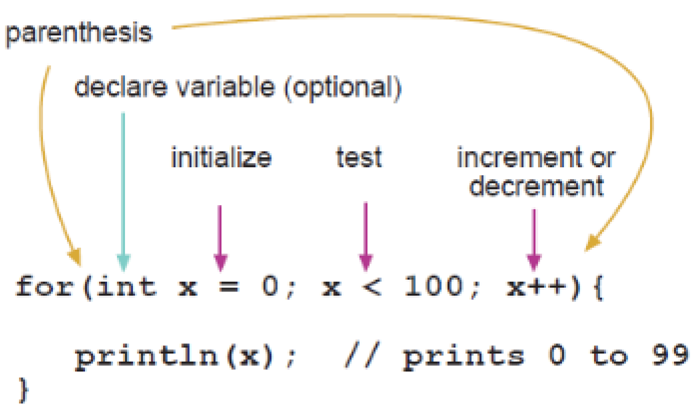
Key function:

- for statements

The for statement is used to repeat a block of statements enclosed in curly braces. An increment counter is usually used to increment and terminate the loop. The for statement is useful for any repetitive operation, and is often used in combination with arrays to operate on collections of data/pins.

There are three parts to the for loop header:

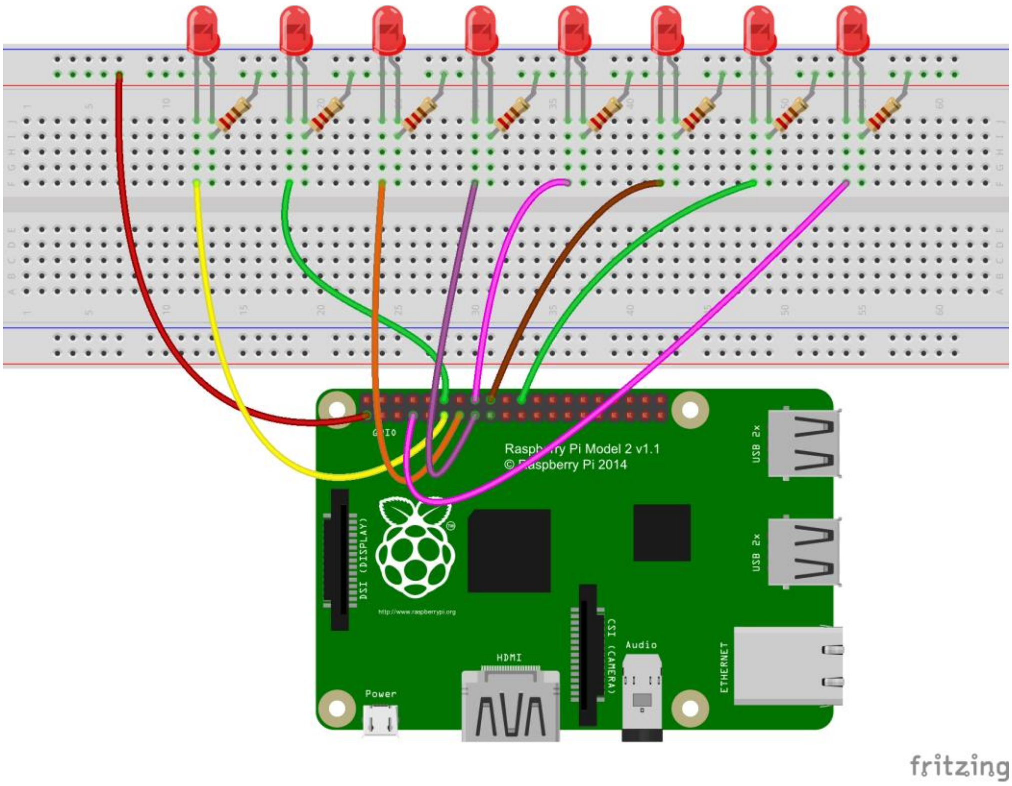
```
for (initialization; condition; increment) {
    //statement(s);
}
```



The initialization happens first and exactly once. Each time through the loop, the condition is tested; if it's true, the statement block, and the increment is executed, then the condition is tested again. When the condition becomes false, the loop ends.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/07_flowingLed/flowingLed.c)

Step 3: Compile

$ gcc flowingLed.c -o flowingLed -lwiringPi

Step 4: Run

$ sudo ./flowingLed

For Python users:

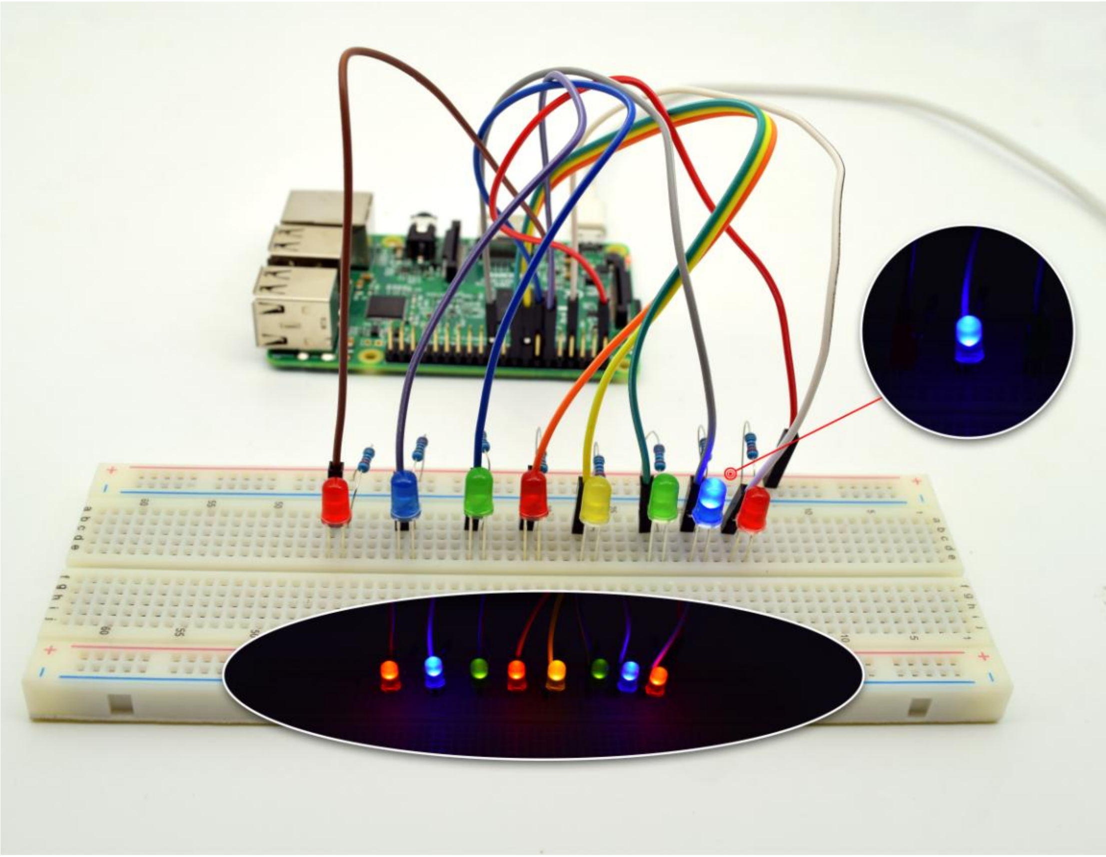
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/07_flowingLed.py)

Step 3: Run

$ sudo python 07_flowingLed.py

Now, you can see 8 LEDs light up in sequence from the red one on the right side to others on the left, and next from the left to the right. The LEDs flash like flowing water repeatedly in a circular way.



Summary

Through this simple but fun experiment, you should have learned more skills in programming on Raspberry Pi. In addition, you can also modify the circuit and code provided to achieve even more dazzling effects.

Lesson 8 Breathing LED

Overview

In this lesson, we will learn how to program the Raspberry Pi to generate PWM signals. And then we use the PWM square-wave signals to control an LED gradually getting brighter and then slowly dimmer, much like human breath.

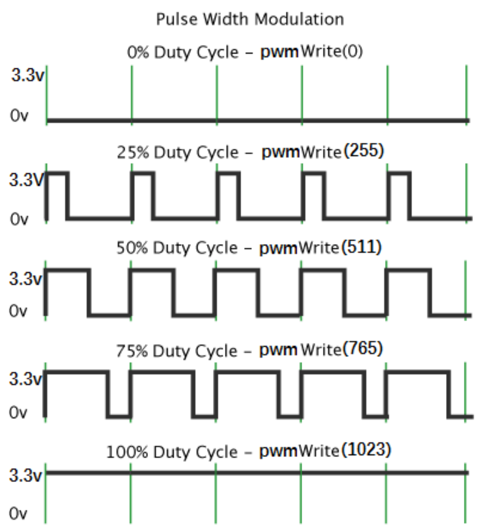
Components

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

Principle

Pulse Width Modulation, or PWM, is a technique for getting analog results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages in between full on (5 Volts) and off (0 Volts) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of "on time" is called the pulse width. To get varying analog values, you change, or modulate, that pulse width. If you repeat this on-off pattern fast enough with an LED for example, the result is as if the signal is a steady voltage between 0 and 5v controlling the brightness of the LED.

In the following figure, the green lines represent a regular time period. This duration or period is the inverse of the PWM frequency. In other words, with Raspberry Pi's PWM frequency at about 500Hz, the green lines would measure 2 milliseconds each. A call to pwmWrite() is on a scale of 0-1023, such that pwmWrite(1023) requests a 100% duty cycle (always on), and pwmWrite(511) is a 50% duty cycle (on half the time) for example.



Key functions:

For C language users:

- pwmWrite(int pin, int value)

Writes the value to the PWM register for the given pin. The Raspberry Pi has one on-board PWM pin, pin 1 (BMC_GPIO 18, Phys 12) and the range is 0-1024. Other PWM devices may have other PWM ranges.

This function is not able to control the Pi's on-board PWM when in Sys mode.

For Python users:

- p = GPIO.PWM(channel, frequency)

This is used for creating a PWM.

- p.start(dc)

Start the pwm you have created.

- p.ChangeFrequency(freq)

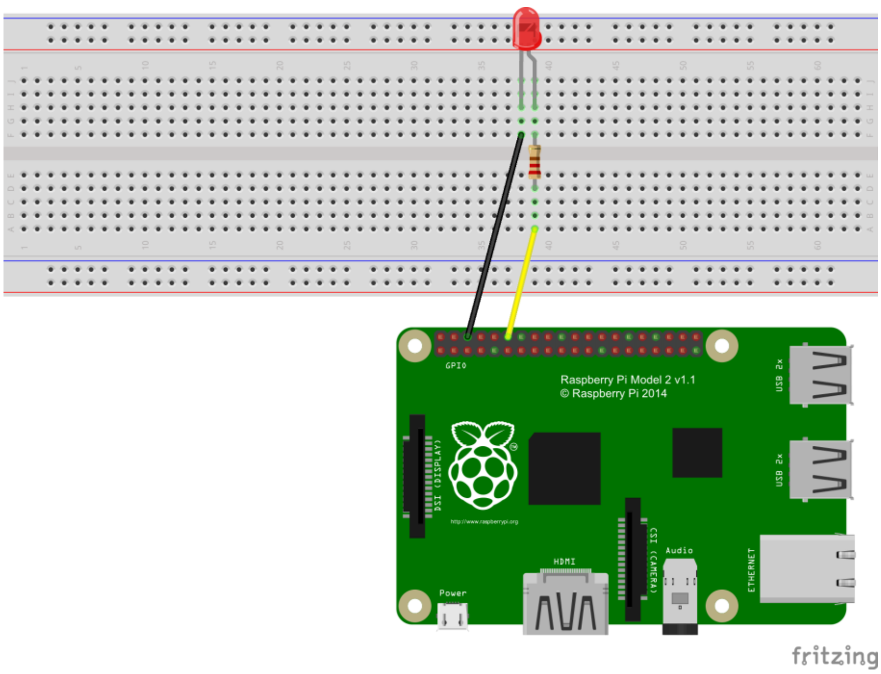
Change the frequency of pwm.

- p.stop( )

Stop the pwm.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/08_breathingLed/breathingLed.c)

Step 3: Compile

$ gcc breathingLed.c -o breathingLed -lwiringPi

Step 4: Run

$ sudo ./breathingLed

For Python users:

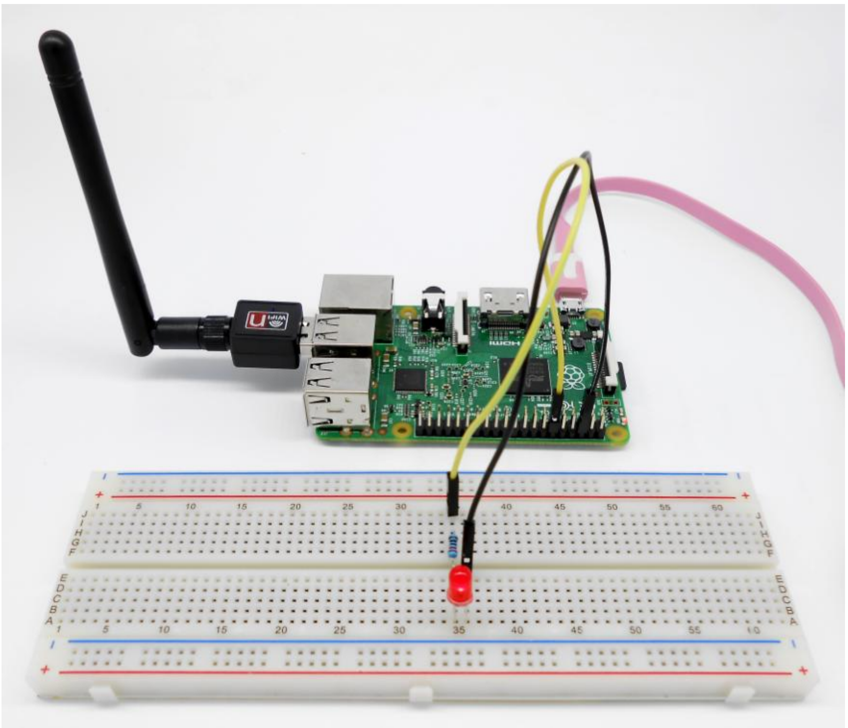
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi /08_breathingLed.py)

Step 3: Run

$ sudo python 08_breathingLed.py

Now, you should see the LED lights up and gets gradually brighter, and then slowly turns dimmer. The process repeats circularly, and with the particular rhythm it looks like animals' breath.



Summary

By learning this lesson, you should have mastered the basic principles of the PWM, and get skilled at the PWM programming on the Raspberry Pi platform.

Lesson 9 Controlling an RGB LED with PWM

Overview

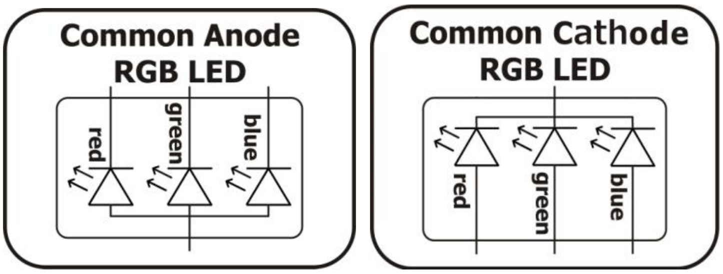
In this lesson, we will program the Raspberry Pi for RGB LED control, and make the RGB LED emit light of various colors.

Components

- 1* Raspberry Pi
- 1* RGB LED
- 3* 220Ω Resistor
- 1* Breadboard
- Several jumper wires

Principle

RGB LEDs consist of three LEDs in different colors: red, green and blue. These three colored LEDs are capable of producing any color. Tri-color LEDs with red, green, and blue emitters, in general use a four-wire connection with one common lead (anode or cathode).



What we use in this experiment is a common anode RGB LED. The longest pin is the common anode of the three LEDs. The pin is connected to the +3.3V pin of the Raspberry Pi, and the rest pins are connected to pin 11, pin 12, and pin 13 of Raspberry Pi with a current limiting resistor between.

In this way, we can control the color of the RGB LED by 3-channel PWM signal.

Key functions:

- int softPwmCreate (int pin, int initialValue, int pwmRange)

This creates a software controlled PWM pin. You can use any GPIO pin and the pin numbering will be that of the wiringPiSetup() function you used. Use 100 for the pwmRange, then the value can be anything from 0 (off) to 100 (fully on) for the given pin.

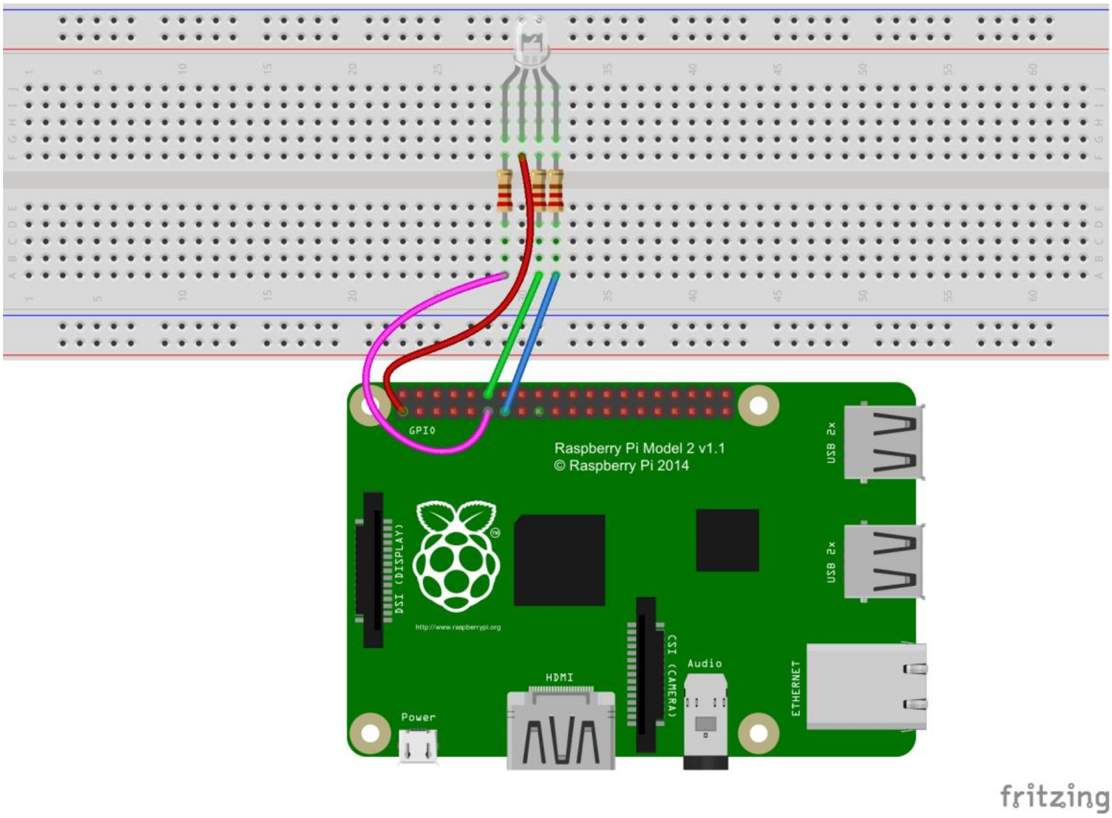
The return value is 0 for success. Anything else and you should check the global errno variable to see what went wrong.

- void softPwmWrite (int pin, int value)

This updates the PWM value on the given pin. The value is checked to be in-range and pins that haven't previously been initialised via softPwmCreate will be silently ignored.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/09_rgbLed/rgbLed.c)

Step 3: Compile

```
$ gcc rgbLed.c -o rgbLed -lwiringPi -lpthread
```

NOTE: The compiler option '-lpthread' is required because the implementation of softPwm is based on Linux multithreading.

Step 4: Run

```
$ sudo ./rgbLed
```

For Python users:

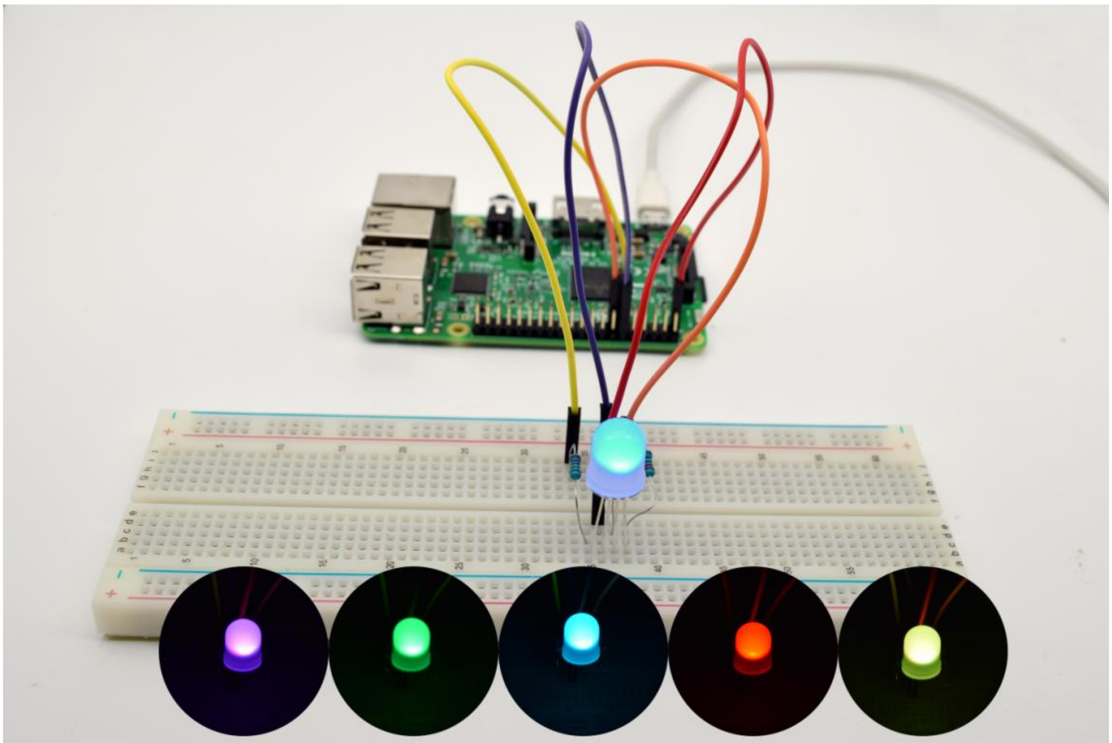
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/09_rgbLed.py)

Step 3: Run

$ sudo python 09_rgbLedLed.py

Now, you can see the RGB LED flashing red, green, blue, yellow, white and purple light, and then the RGB LED goes out. Each state lasts for 1s each time, and the LED flashes colors repeatedly in such sequence.



Summary

By learning this lesson, you should have already got the principle and the programming of RGB LED. Now you can use your imagination to achieve even more cool ideas based on what you learned in this lesson.

Lesson 10 7-Segment Display

Overview

In this lesson, we will program the Raspberry Pi to achieve the controlling of a segment display.

Components

- 1* Raspberry Pi
- 1* 220Ω Resistor
- 1* 7-Segment display
- 1* Breadboard
- Several jumper wires

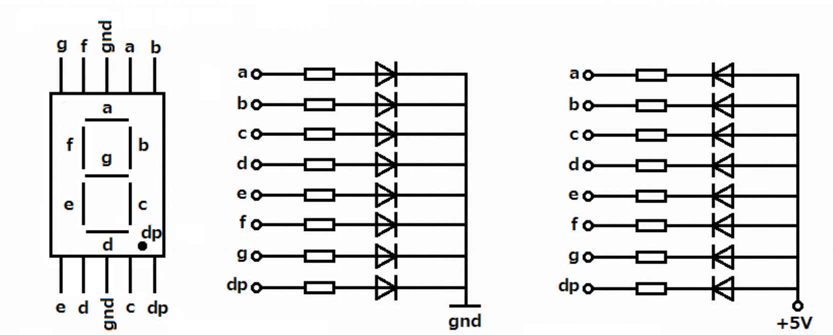
Principle

The seven-segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

Seven-segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

The seven-segment display is an 8-shaped LED display device composed of eight LEDs (including a decimal point). The segments are named respectively a, b, c, d, e, f, g, and dp.

The segment display can be divided into two types: common anode and common cathode segment displays, by internal connections.

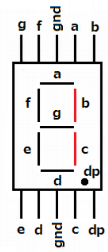


(a) 7-segment display (b) Common Cathode (c) Common Anode

For a common-anode LED, the common anode should be connected to the power supply (VCC); for a common-cathode LED, the common cathode should be connected to the ground (GND).

Each segment of a segment display is composed of an LED, so a resistor is needed for protecting the LED.

A 7-segment display has seven segments for displaying a figure and one more for displaying a decimal point. For example, if you want to display a number '1', you should only light the segment b and c, as shown below.



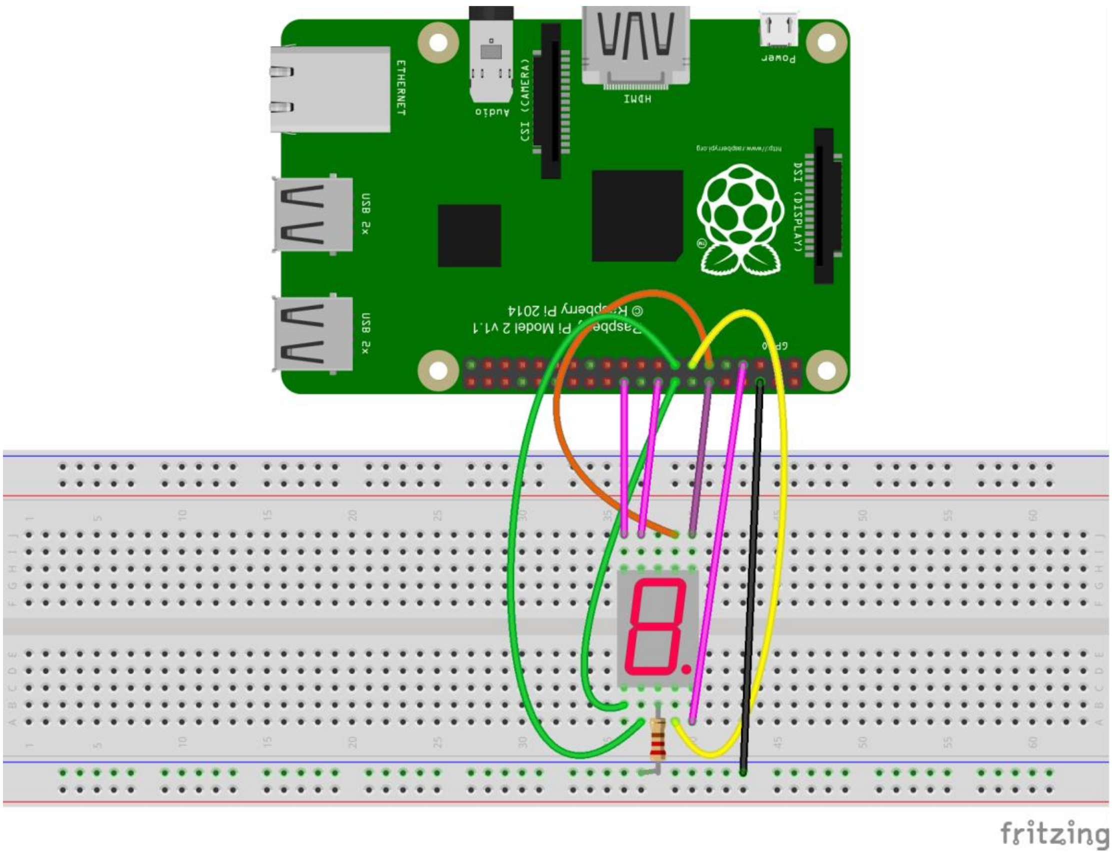
Key function:

- **void digitalWriteByte (int value)**

This writes the 8-bit byte supplied to the first 8 GPIO pins. It's the fastest way to set all 8 bits at once to a particular value, although it still takes two write operations to the Pi's GPIO hardware.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/10_segment/segment.c)

Step 3: Compile

```
$ gcc segment.c -o segment -lwiringPi
```

Step 4: Run

$ sudo ./segment

For Python users:

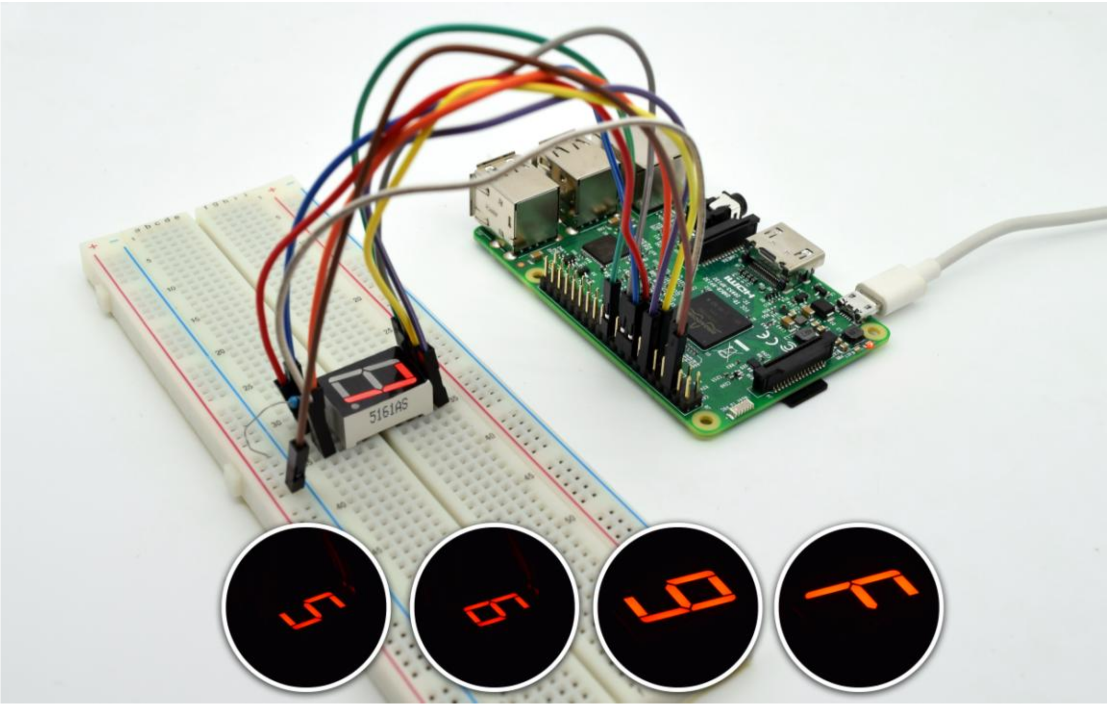
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/10_segment.py)

Step 3: Run

$ sudo python 10_segment.py

Now, you should see the number 0~9 and character A~F are displayed in turn on the segment display.



Summary

Through this lesson, you should have learned the principle and programming of the segment display. You can use what you've learned in the previous lessons to modify the code provided in this lesson to make cooler works.

Lesson 11 4-Digit 7-Segment Display

Overview

In this lesson, we will program the Raspberry Pi to control a 4-digit 7-segment display.

Components

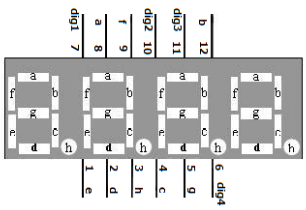
- 1* Raspberry Pi
- 4* 220Ω Resistor
- 1* 4-digit 7-segment display
- 1* Breadboard
- Several jumper wires

Principle

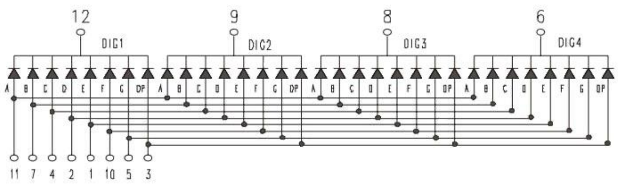
The four-digit segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays.

Four-digit segment displays are widely used in digital clocks, electronic meters, basic calculators, and other electronic devices that display numerical information.

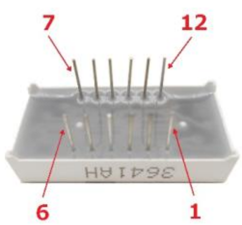
The four-digit segment display is a 4*8-shaped LED display device composed of 32 LEDs (including four decimal points). The segments are named respectively a, b, c, d, e, f, g, h, dig1, dig2, dig3, and dig4.



What we use in this experiment is a common-cathode 4-digit 7-segment display. Its internal structure is as shown below:

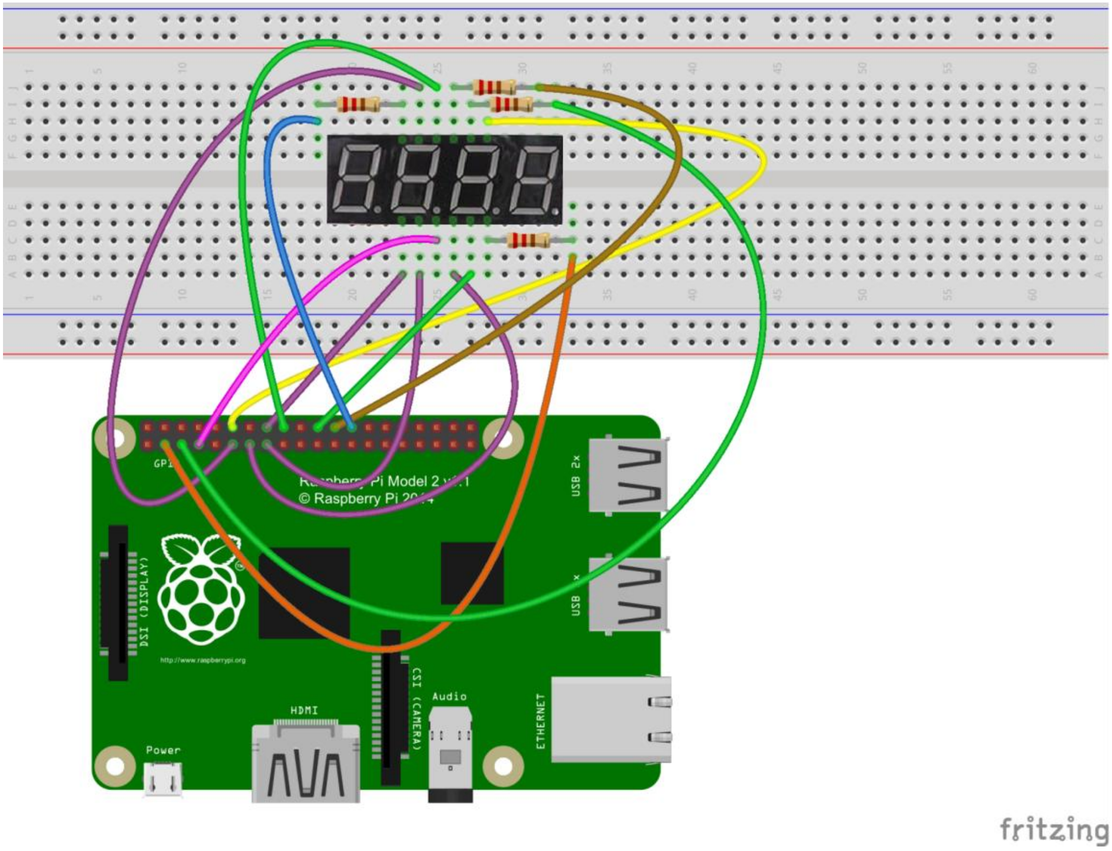


The pin number is as follows:



Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/11_4bitSegment/fourBitSegment.c)

Step 3: Compile

$ gcc fourBitSegment.c -o fourBitSegment -lwiringPi

Step 4: Run

$ sudo ./fourBitSegment

For Python users:

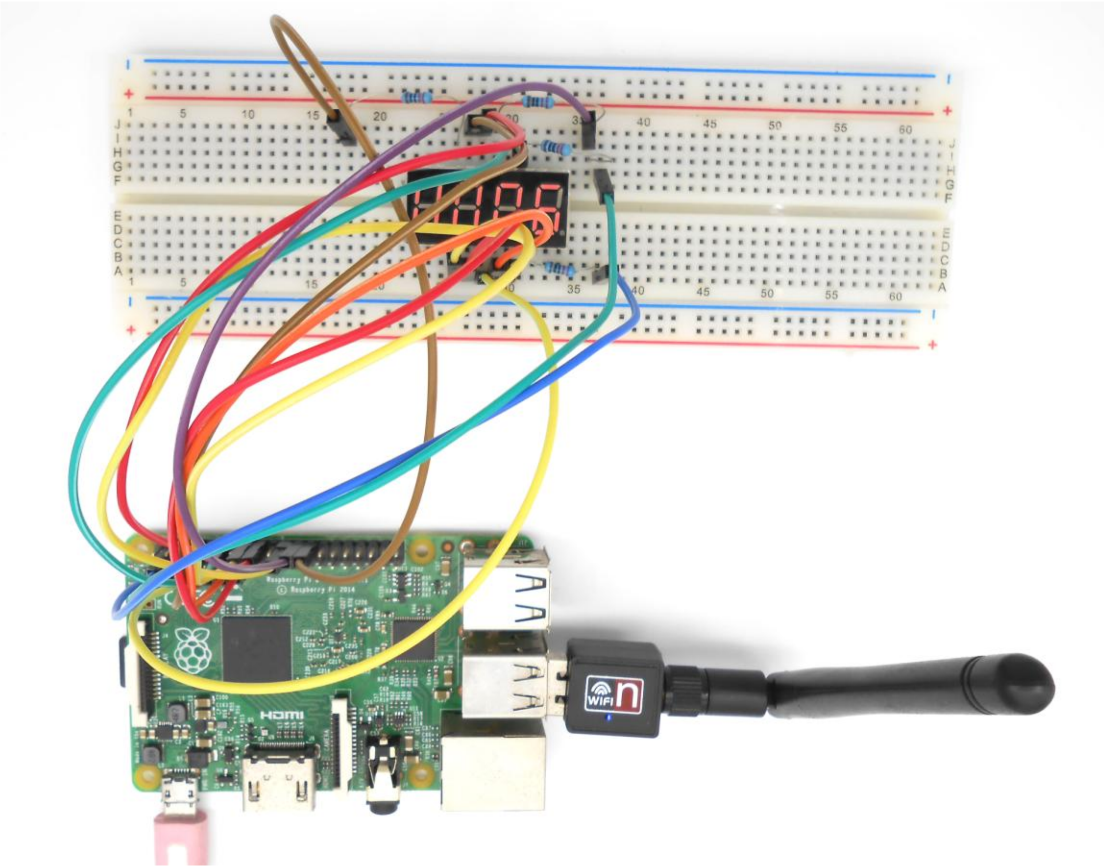
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/11_fourBitSegment.py)

Step 3: Run

$ sudo python 11_fourBitSegment.py

Now, you should see the number displayed on the segment display.



Lesson 12 LCD1602

Overview

In this lesson, we will learn how to use a character display device - LCD1602 on the Raspberry Pi platform. We first make the LCD1602 display a string "Hello Geeks!" scrolling, and then display “Adeept” and “www.adeept.com” statically.

Components

- 1* Raspberry Pi
- 1* LCD1602
- 1* 10KΩ Potentiometer
- 1* Breadboard
- Several jumper wires

Principle

LCD1602 is a kind of character LCD display. The LCD has a parallel interface, meaning that the microcontroller has to manipulate several interface pins at once to control the display. The interface consists of the following pins:

● A register select (RS) pin that controls where in the LCD's memory you're writing data to. You can select either the data register, which holds what goes on the screen, or an instruction register, which is where the LCD's controller looks for instructions on what to do next.

● A Read/Write (R/W) pin that selects reading mode or writing mode

● An Enable pin that enables writing to the registers

● 8 data pins (D0-D7). The status of these pins (high or low) is the bits that you're writing to a register when you write, or the values when you read.

● There are also a display contrast pin (Vo), power supply pins (+5V and Gnd) and LED Backlight (Bklt+ and BKlt-) pins that you can use to power the LCD, control the display contrast, and turn on or off the LED backlight respectively.

The process of controlling the display involves putting the data that form the image of what you want to display into the data registers, then putting instructions in the instruction register. The wiringPiDev Library simplifies this for you, so you don't need to know the low-level instructions.

The Hitachi-compatible LCDs can be controlled in two modes: 4-bit or 8-bit. The 4-bit mode requires six I/O pins from the Raspberry Pi, while the 8-bit mode requires 10 pins. For displaying text on the screen, you can do most everything in 4-bit mode, so example shows how to control a 2x16 LCD in 4-bit mode.

A potentiometer, informally a pot, is a three-terminal resistor with a sliding or rotating contact that forms an adjustable voltage divider. If only two terminals are used, one end and the wiper, it acts as a variable resistor or rheostat.

Key functions:

- int lcdInit (int rows, int cols, int bits, int rs, int strb, int d0, int d1, int d2, int d3, int d4, int d5, int d6, int d7)

This is the main initialisation function and must be called before you use any other LCD functions.

Rows and cols are the rows and columns on the display (e.g. 2, 16 or 4,20). Bits is the number of bits wide on the interface (4 or 8). The rs and strb represent the pin numbers of the displays RS pin and Strobe (E) pin. The parameters d0 through d7 are the pin numbers of the 8 data pins connected from the Pi to the display. Only the first 4 are used if you are running the display in 4-bit mode.

The return value is the 'handle' to be used for all subsequent calls to the lcd library when dealing with that LCD, or -1 to indicate a fault. (Usually incorrect parameters)

- lcdPosition (int handle, int x, int y)

Set the position of the cursor for subsequent text entry. x is the column and 0 is the left-most edge. y is the line and 0 is the top line.

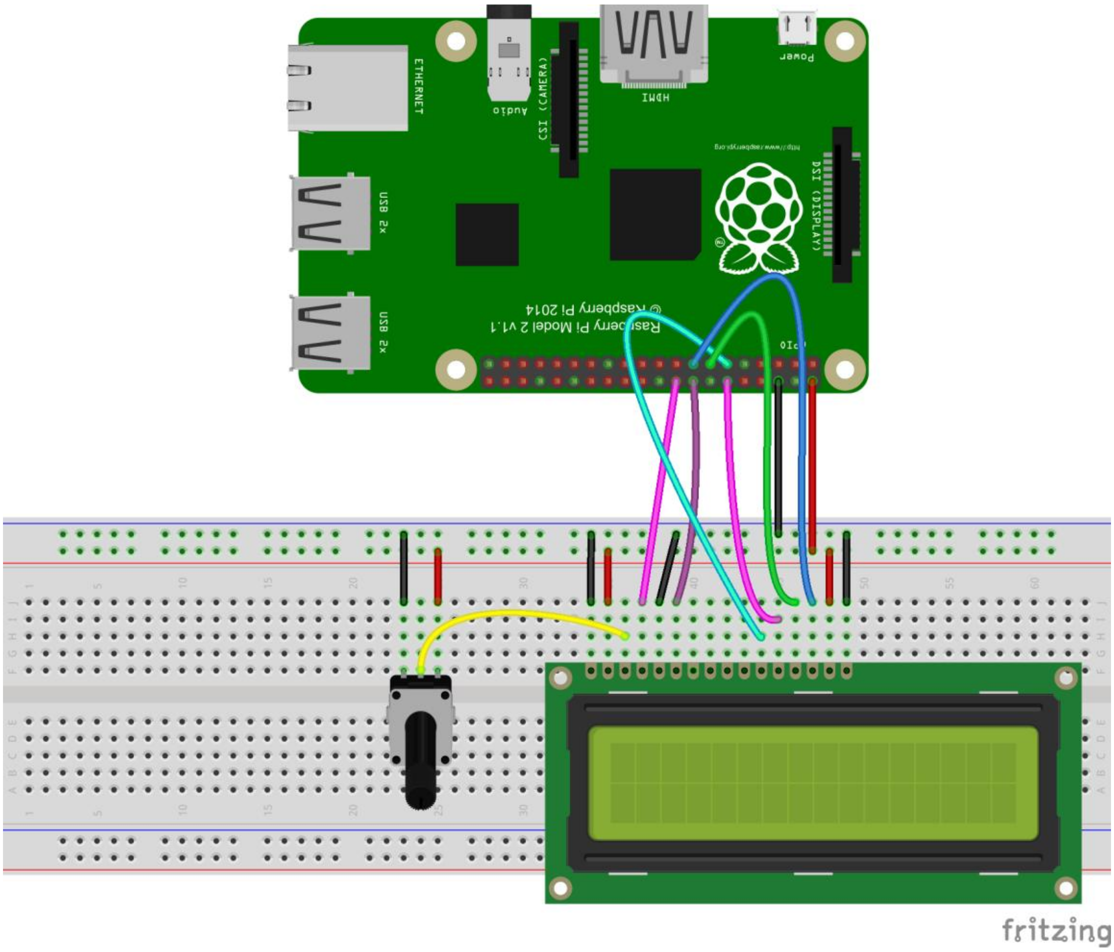
- lcdPuts (int handle, const char *string)
- lcdPrintf (int handle, const char *message, …)
- lcdPutchar (int handle, unsigned char data)

These output a single ASCII character, a string or a formatted string using the usual printf formatting commands.

At the moment, there is no clever scrolling of the screen, but long lines will wrap to the next line, if necessary.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/12_lcd1602/lcd1602_2.c)

Step 3: Compile

$ gcc lcd1602_2.c -o lcd1602_2 -lwiringPi -lwiringPiDev

Step 4: Run

$ sudo ./lcd1602_2

Python users:

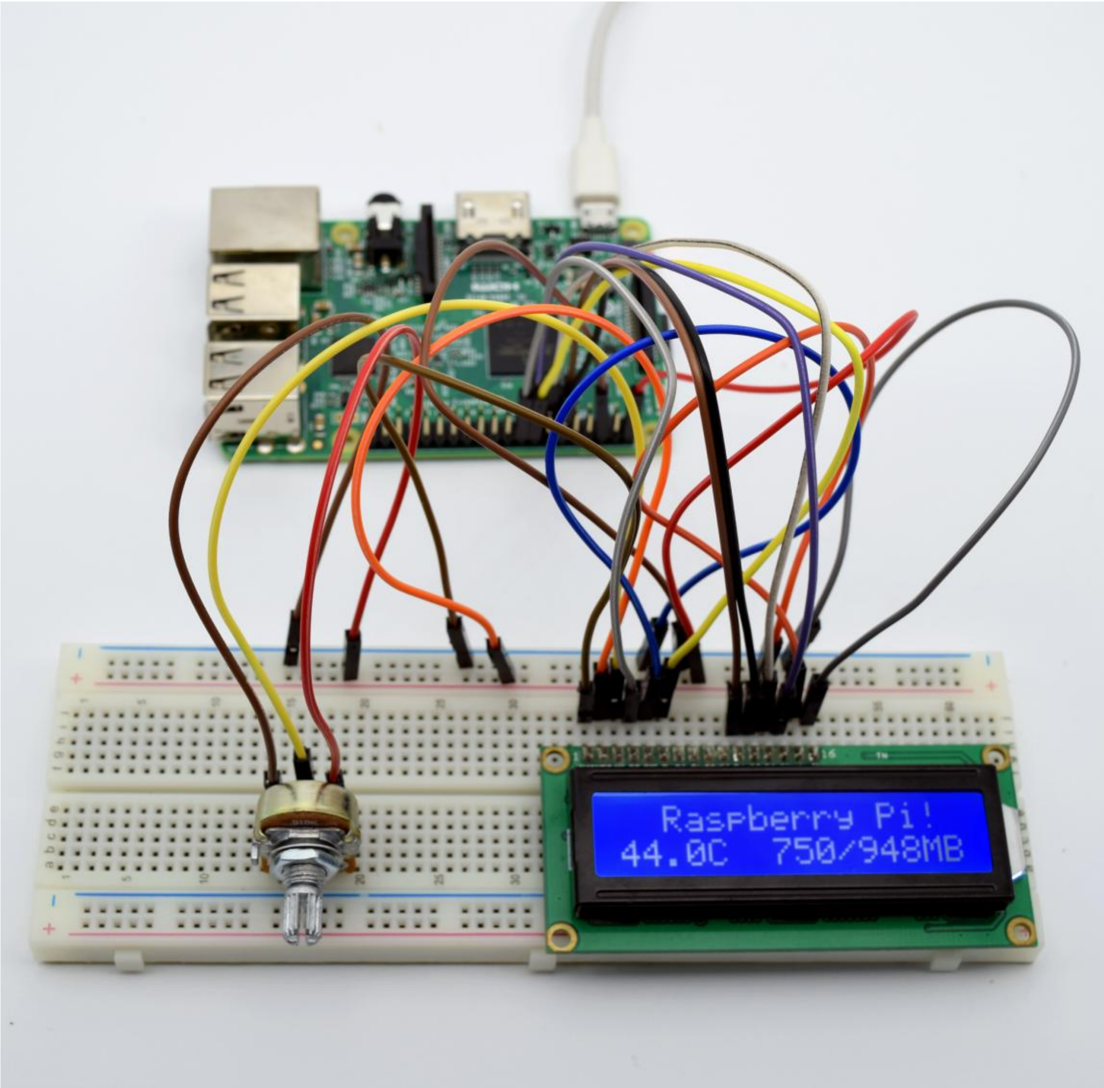
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/12_lcd1602.py)

Step 3: Run

$ sudo python 12_lcd1602.py

Now, you can see the string "Hello Geeks!" shown on the LCD1602 scrolling, and then the string "Adeept" and "www.adeept.com" displayed statically.



Summary

After learning the experiment, you should have already mastered the driver of the LCD1602. Now you can make something more interesting based on this lesson and the previous lessons learned.

Lesson 13 Matrix Keyboard

Overview

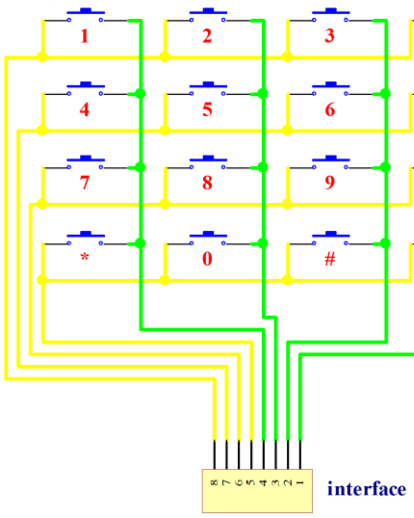
In this lesson, we will learn how to use a matrix keyboard.

Components

- 1* Raspberry Pi
- 1* 4x4 Matrix Keyboard
- Several jumper wires

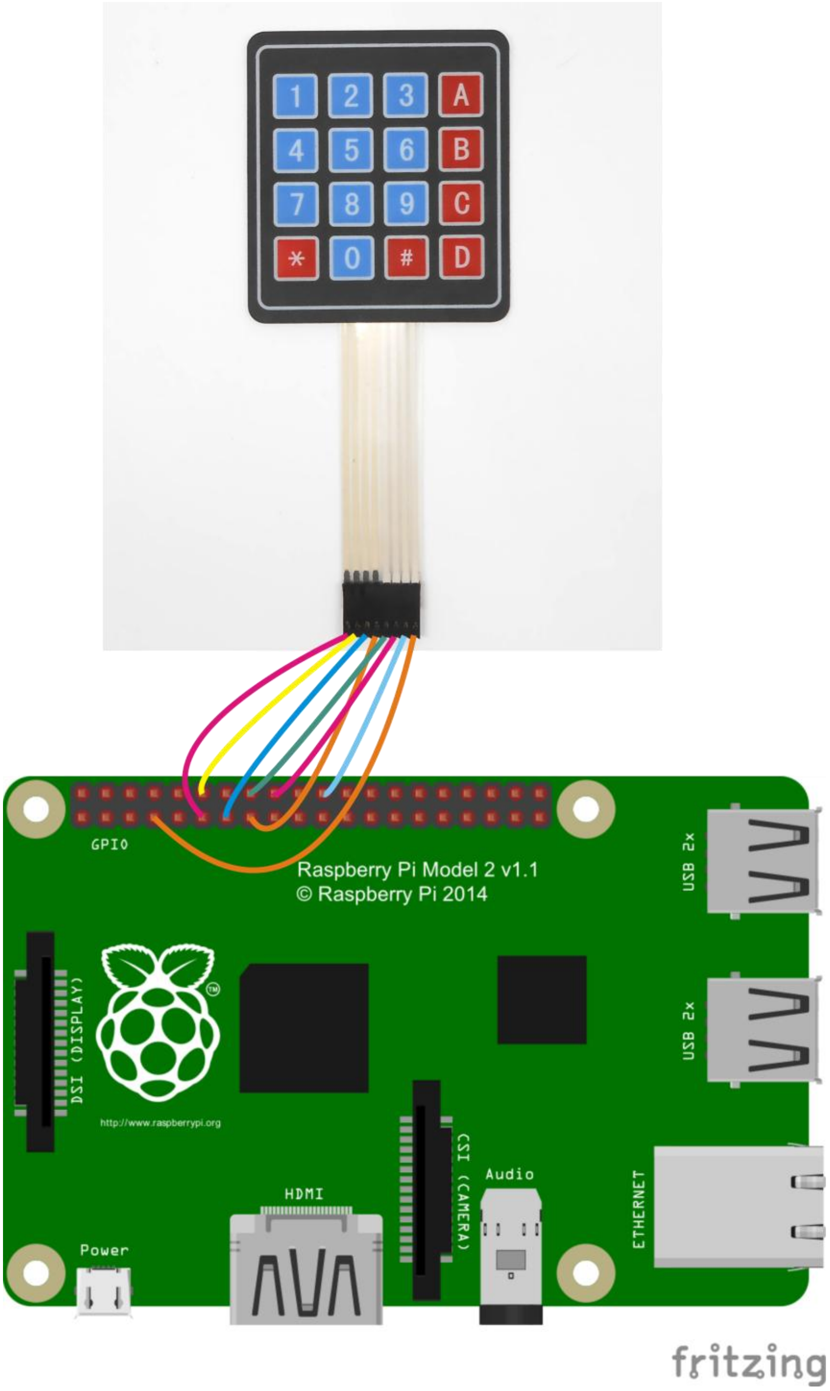
Principle

In order to save the resources of the microcontroller ports, we usually connect the buttons of a matrix in practical projects.
See the following figure for the schematics of a 4x4 matrix keyboard:



Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/13_matrixKeyboard/matrixKeyboard.c)

Step 3: Compile

```
$ gcc matrixKeyboard.c -o matrixKeyboard -lwiringPi
```

Step 4: Run

```
$ sudo ./matrixKeyboard
```

For Python users:

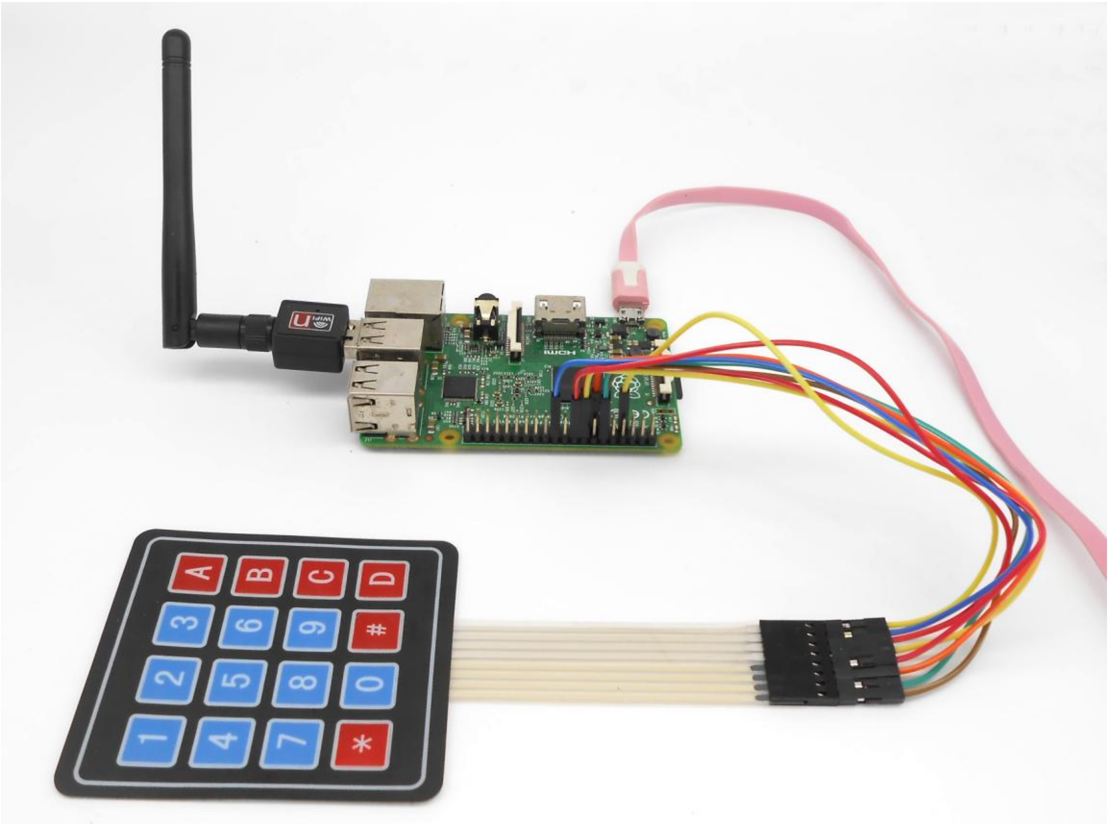
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/13_matrixKeyboard.py)

Step 3: Run

$ sudo python 13_ matrixKeyboard.py

Now, press one of the buttons on the 4x4 matrix keyboard, and you will see the corresponding key value displayed on the terminal.



Lesson 14 Measuring the Distance

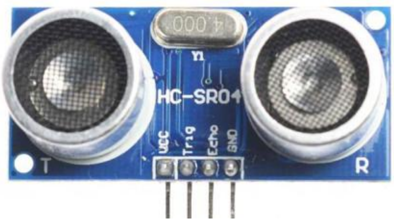
Overview

In this lesson, we will learn how to measure the distance by an ultrasonic distance sensor.

Components

- 1* Raspberry Pi
- 1* Ultrasonic distance sensor
- 1* Breadboard
- Several jumper wires

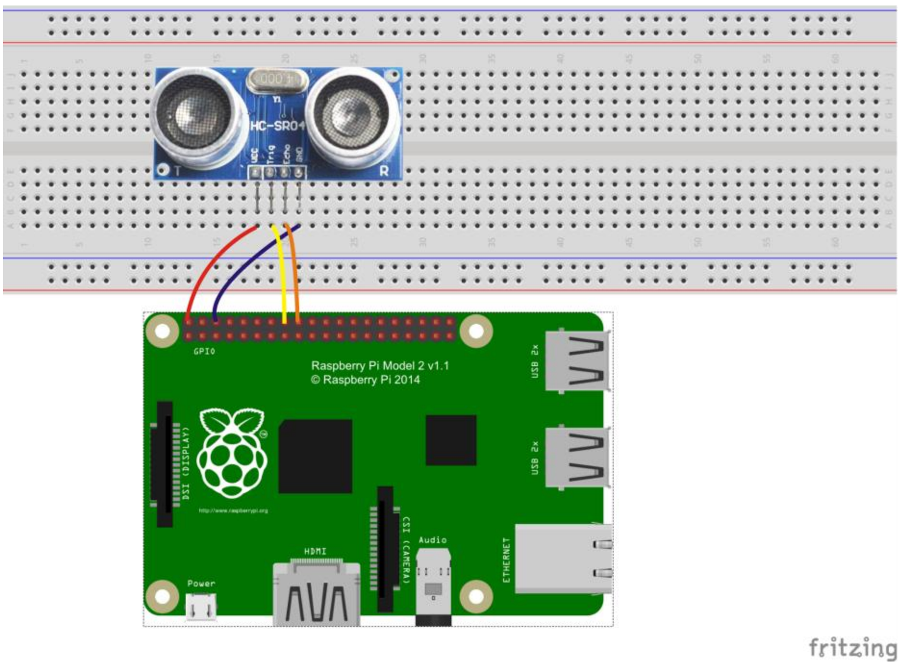
Principle



This recipe uses the popular Parallax PING ultrasonic distance sensor to measure the distance to an object ranging from 2cm to around 3m. This sensor works by sending a sound wave out and calculating the time it takes to get back. By doing this, it can tell us how far away an obstacle is to the ultrasonic sensor.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/14_ultrasonicSensor/distance.c)

Step 3: Compile

```
$ gcc distance.c -o distance -lwiringPi
```

Step 4: Run

```
$ sudo ./distance
```

For Python users:

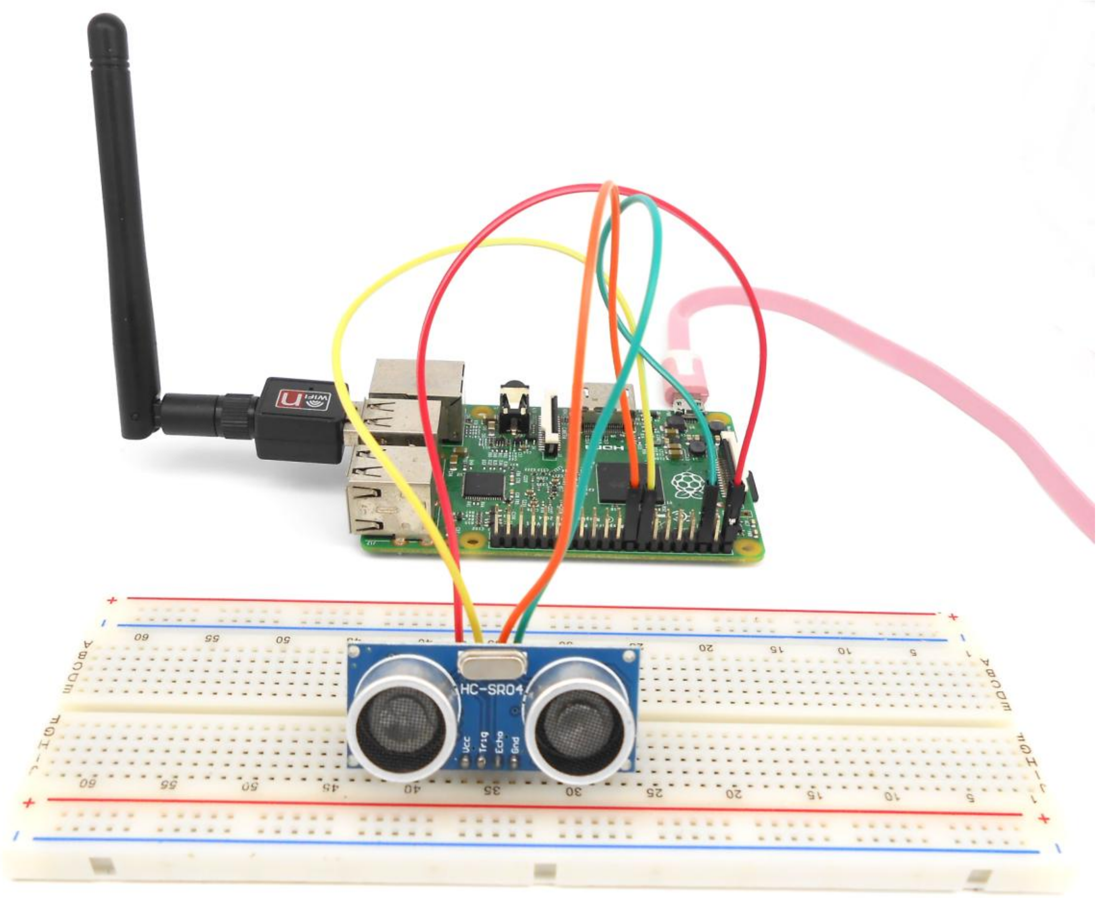
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/14_distance.py)

Step 3: Run

```
$ sudo python 14_distance.py
```

Now, you will see the distance between the obstacle and the ultrasonic sensor displayed on the screen.



Lesson 15 Temperature & Humidity Sensor – DHT-11

Overview

In this lesson, we will learn how to use DHT-11 to collect temperature and humidity data.

Components

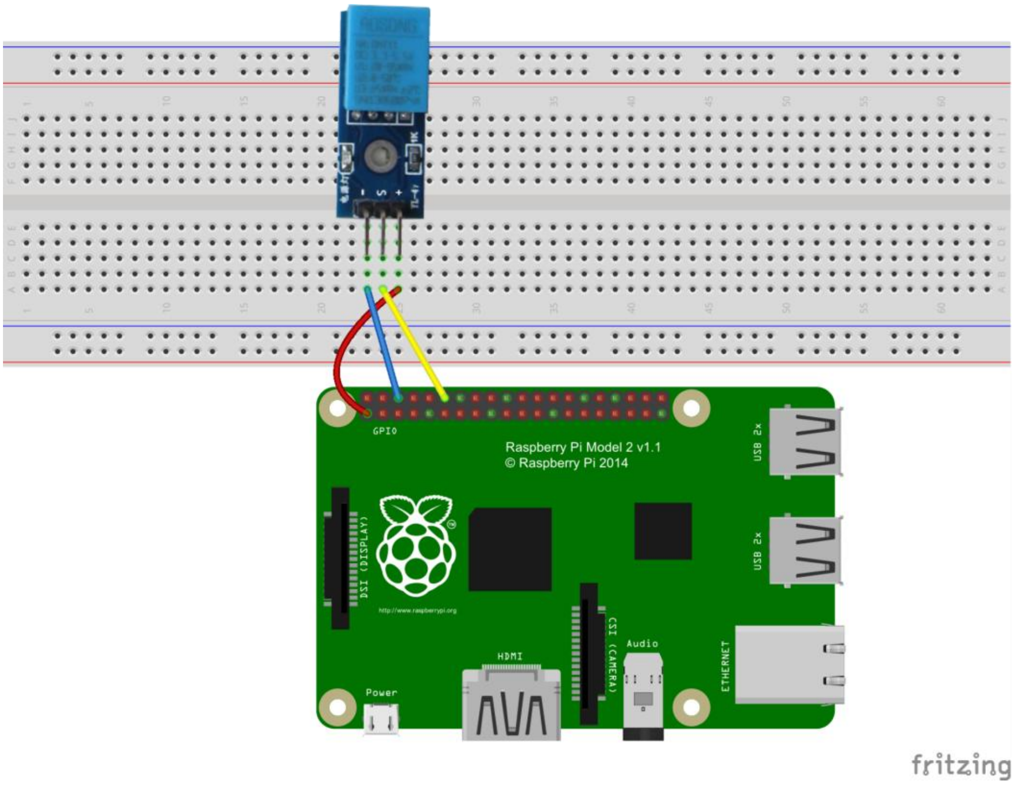
- 1* Raspberry Pi
- 1* DHT-11
- 1* Breadboard
- Several jumper wires

Principle

This DHT-11 temperature & humidity sensor features a temperature & humidity sensor complex with a calibrated digital signal output. By using the exclusive digital signal acquisition technique and temperature & humidity sensing technology, it ensures high reliability and excellent long-term stability. This sensor includes a resistive-type humidity measurement component and an NTC temperature measurement component, and connects to a high-performance 8-bit microcontroller, offering excellent quality, fast response, anti-interference ability and cost-effectiveness.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/15_DHT11/dht11.c)

Step 3: Compile

```
$ gcc dht11.c -o dht11 -lwiringPi
```

Step 4: Run

```
$ sudo ./dht11
```

For Python users:

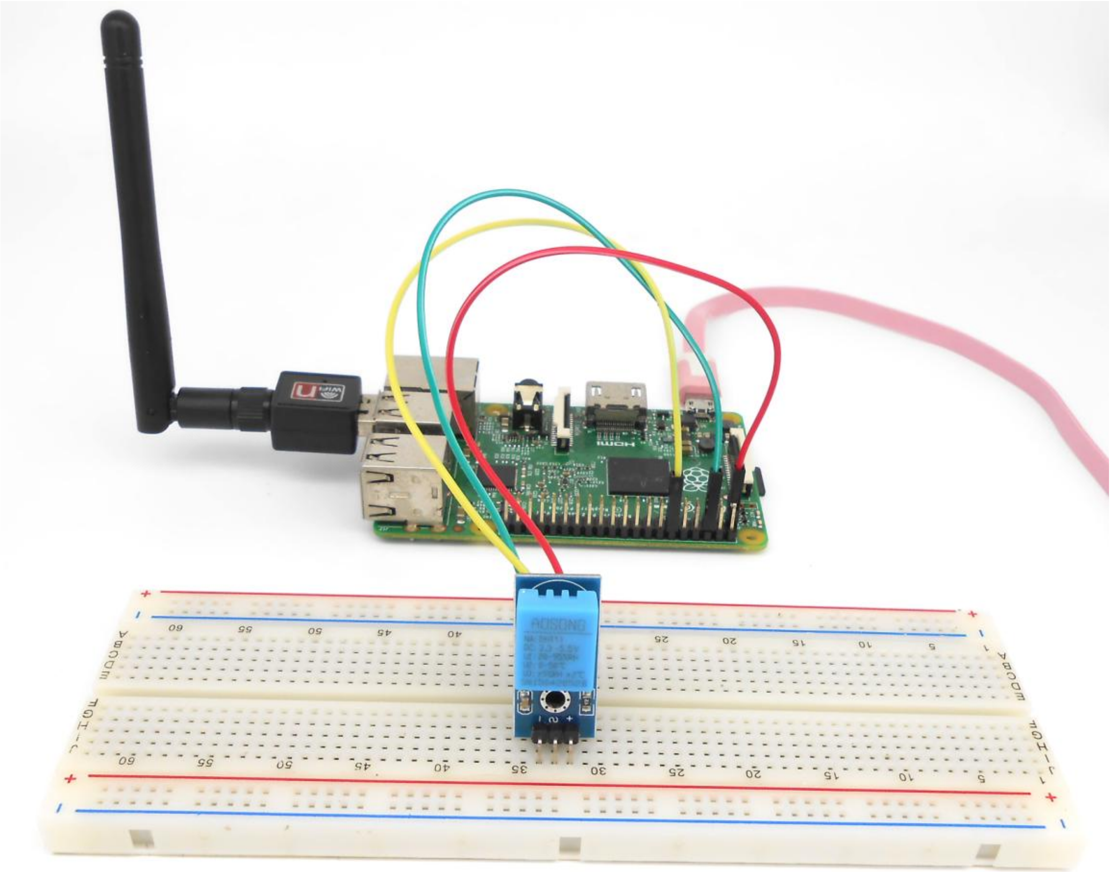
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/15_dht11.py)

Step 3: Run

```
$ sudo python 15_dht11.py
```

Now, you can see the temperature and humidity data displayed on the terminal.



Lesson 16 Dot-matrix Display

Overview

In this lesson, we will program to control an 8*8 dot-matrix display to display the graphs and numbers as we want to.

Components

- 1* Raspberry Pi
- 1* 8*8 Dot-matrix display
- 2* 74HC595
- 1* Breadboard
- Several jumper wires

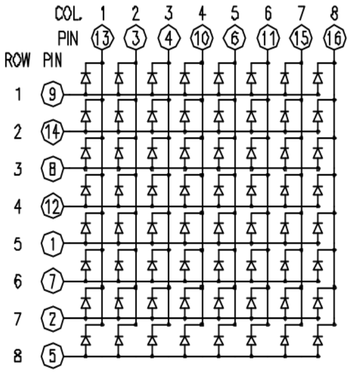
Principle

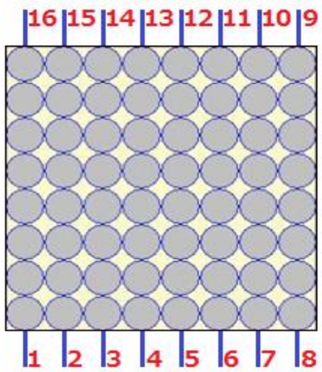
1. Dot-matrix display

A dot-matrix display is a display device used to display information on machines, clocks, railway departure indicators and many other devices requiring a simple display device of limited resolution.

The display consists of a dot-matrix of lights or mechanical indicators arranged in a rectangular configuration (other shapes are also possible, although not common) such that by switching on or off selected lights, text or graphics can be displayed. A dot-matrix controller converts instructions from a processor into signals which turns on or off lights in the matrix so that the required display is produced.

The internal structure and appearance of the dot-matrix display is as shown below:





dot matrix display

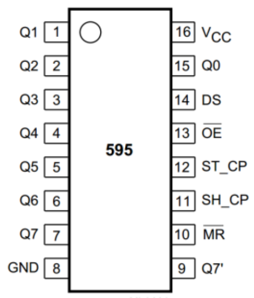
An 8*8 dot-matrix display consists of 64 LEDs, and each LED is placed at the intersection of the lines and columns. When the corresponding row is set as high level and the column as low level, the LED will be lit.

A certain drive current is required for the dot-matrix display. In addition, more pins are needed for connecting the dot-matrix display with a controller. Thus, to save the Raspberry Pi’s GPIOs, the driver IC 74HC595 is used in this experiment.

2. 74HC595

The 74HC595 is an 8-stage serial shift register with a storage register and 3-state outputs. The shift register and storage register have separate clocks. Data is shifted on the positive-going transitions of the SH_CP input. The data in each register is transferred to the storage register on a positive-going transition of the ST_CP input. The shift register has a serial input (DS) and a serial standard output (Q7') for cascading. It is also provided with asynchronous reset (active LOW) for all 8 shift register stages. The storage register has 8 parallel 3-state bus driver outputs. Data in the storage register appears at the output whenever the output enable input (OE) is LOW.

In this experiment, only 3 pins of the Raspberry Pi are used for controlling the dot-matrix display thanks to the 74HC595.



The function of each pin:

DS: Serial data input

Q0-Q7: 8-bit parallel data output

Q7': Series data output pin, always connected to DS pin of the next 74HC595

OE: Output enable pin, effective at low level, connected to the ground directly

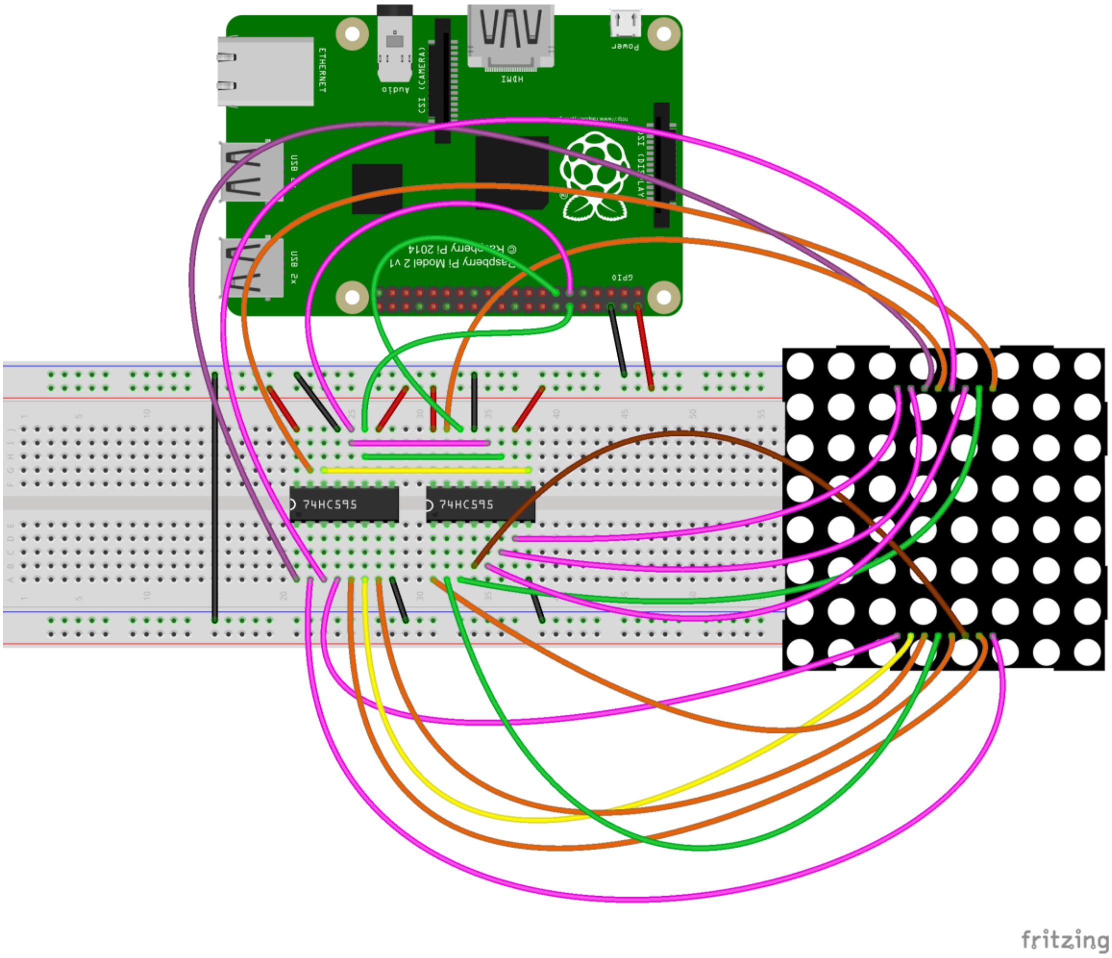
MR: Reset pin, effective at low level, directly connected to 5V high level in practical applications

SH_CP: Shift register clock input

ST_CP: storage register clock input

Procedures

Step 1: Build the circuit (Make sure that the circuit connection is correct and then power on, otherwise it may cause the chips burned.)



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/16_ledMatrix/ledMatrix.c)

Step 3: Compile

$ gcc ledMatrix.c -o ledMatrix -lwiringPi

Step 4: Run

$ sudo ./ledMatrix

For Python users:

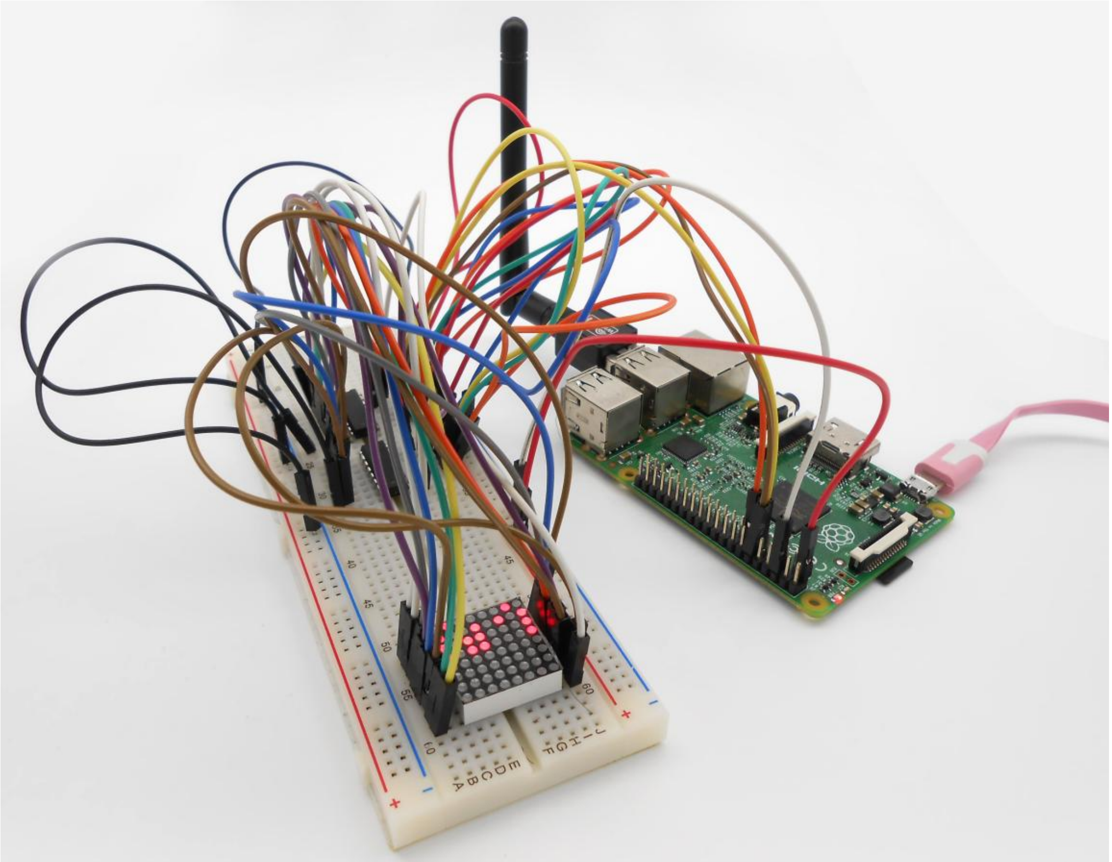
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/16_ledMatrix.py)

Step 3: Run

$ sudo python 16_ledMatrix.py

Now, you can see a rolling character "Adeept" displayed on the dot-matrix display.



Summary

In this experiment, we have not only learned how to use a dot-matrix display to display numbers and letters, but also learned the basic usage of 74HC595. Next you can try utilizing the dot-matrix display to show more effects.

Lesson 17 Photoresistor

Overview

In this lesson, we will learn how to measure the light intensity by photoresistor and display the measurement result on the screen.

Components

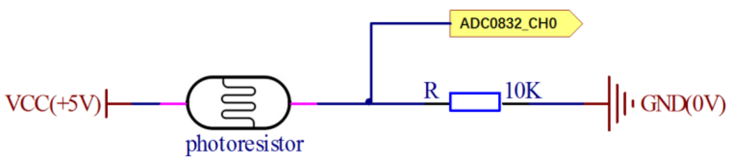
- 1* Raspberry Pi
- 1* ADC0832
- 1* Photoresistor
- 1* 10KΩ Resistor
- 1* Breadboard
- Several jumper wires

Principle

A photoresistor is a light-controlled variable resistor. The resistance of a photoresistor decreases with the increasing incident light intensity; in other words, it exhibits photoconductivity. A photoresistor can be applied in light-sensitive detector circuits.

A photoresistor is made of a high resistance semiconductor. In the dark, it can show a resistance as high as a few megohms (MΩ), while in the light, its resistance can be as low as a few hundred ohms. If the incident light on a photoresistor exceeds a certain frequency, photons absorbed by the semiconductor will give bound electrons enough energy to jump into the conduction band. The resulting free electrons (and their hole partners) conduct electricity, thereby lowering the resistance. The resistance range and sensitivity of a photoresistor can substantially differ among dissimilar devices. Moreover, unique photoresistors may react substantially differently to photons within certain wavelength bands.

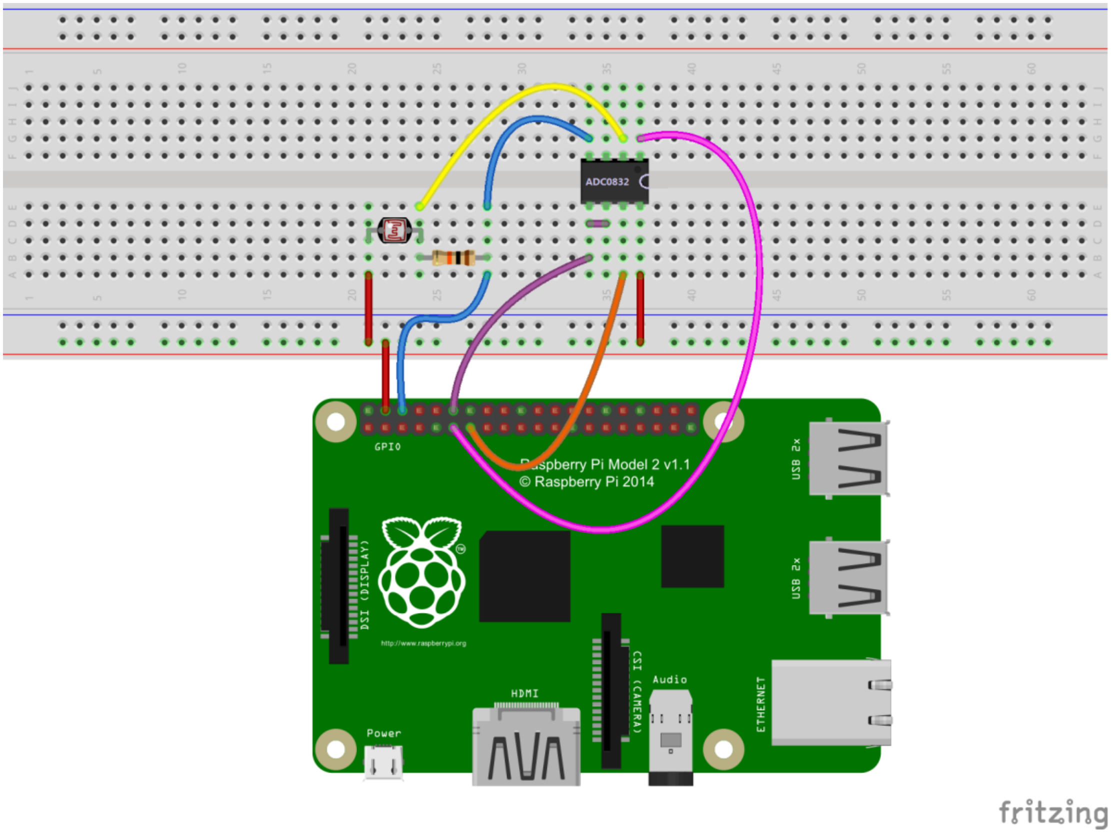
The schematic diagram of this experiment is as shown below:



With the increase of the light intensity, the resistance of the photoresistor will decrease. The voltage of the GPIO port in the above figure will become high.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/17_photoresistor/photoresistor.c)

Step 3: Compile

```
$ gcc photoresistor.c -o photoresistor -lwiringPi
```

Step 4: Run

```
$ sudo ./photoresistor
```

For Python users:

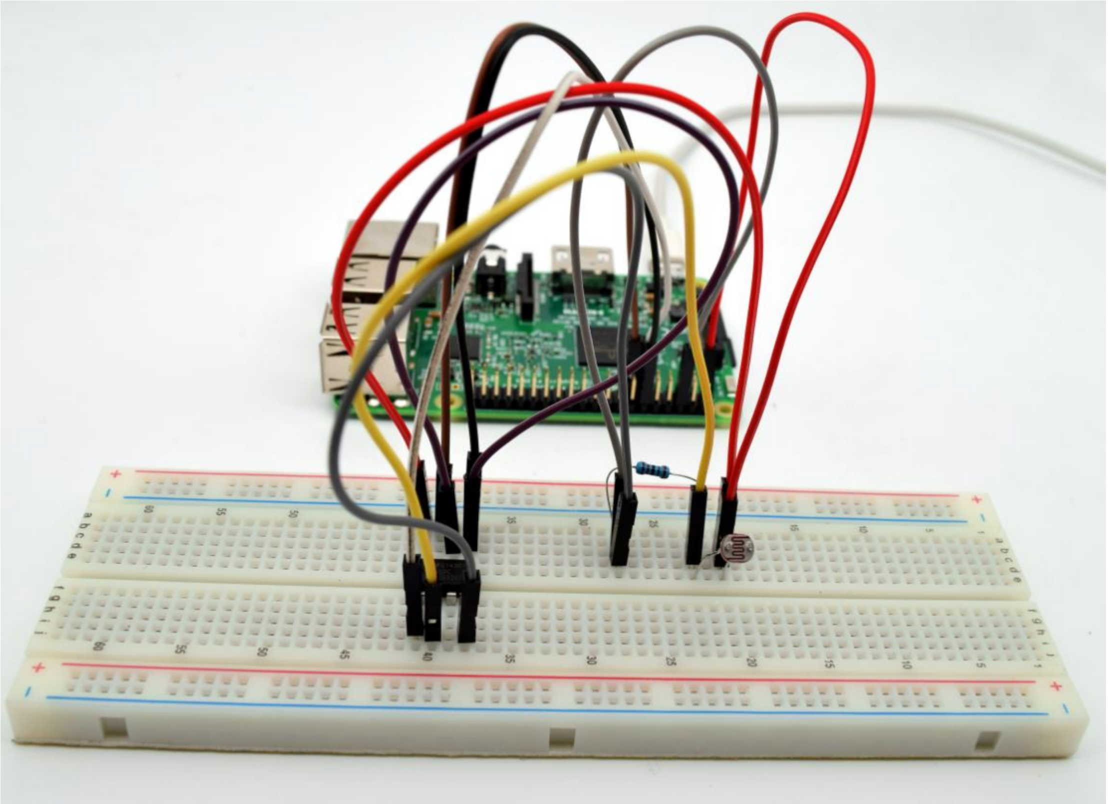
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/17_photoresistor.py)

Step 3: Run

```
$ sudo python 17_photoresistor.py
```

Now, when you try to cover the photoresistor, you will find that the value displayed on the screen decreasing. On the contrary, when you shine the photoresistor with strong light, the value displayed will increase.



Summary

By learning this lesson, you should have learned how to detect the ambient light intensity with the photoresistor. You can take full advantage of your own wisdom and make more original works based on your gains in this and previous experiments.

Lesson 18 Thermistor

Overview

In this lesson, we will learn how to use a thermistor to collect the temperature data by programming the Raspberry Pi and ADC0832.

Components

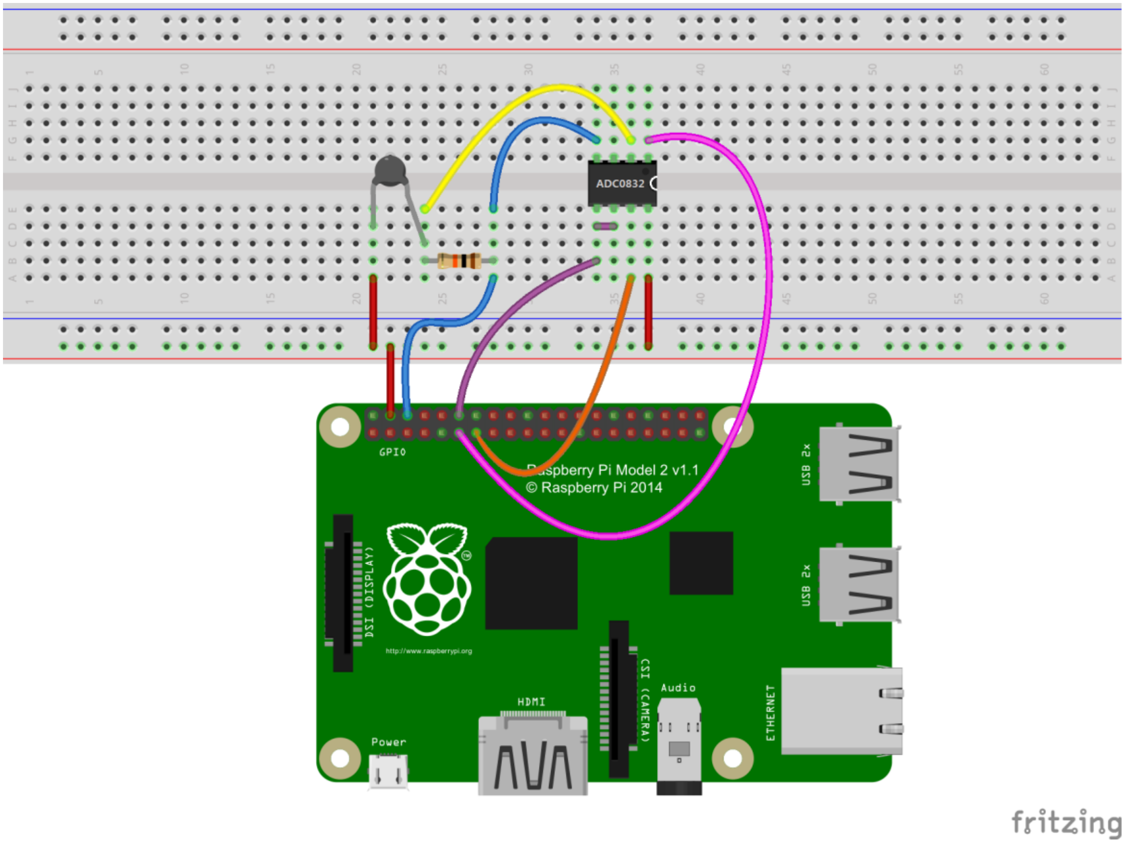
- 1* Raspberry Pi
- 1* ADC0832
- 1* Thermistor
- 1* 10KΩ Resistor
- 1* Breadboard
- Several jumper wires

Principle

A thermistor is a type of resistor whose resistance varies significantly with temperature, more so than in standard resistors. When the temperature increases, the thermistor resistance decreases; when the temperature decreases, the thermistor resistance increases. It can detect the ambient temperature changes in real time. In the experiment, we need an analog-digital converter ADC0832 to convert analog signal into digital signal.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/18_thermistor/thermistor.c)

Step 3: Compile

$ gcc thermistor.c -o thermistor -lwiringPi

Step 4: Run

$ sudo ./thermistor

For Python users:

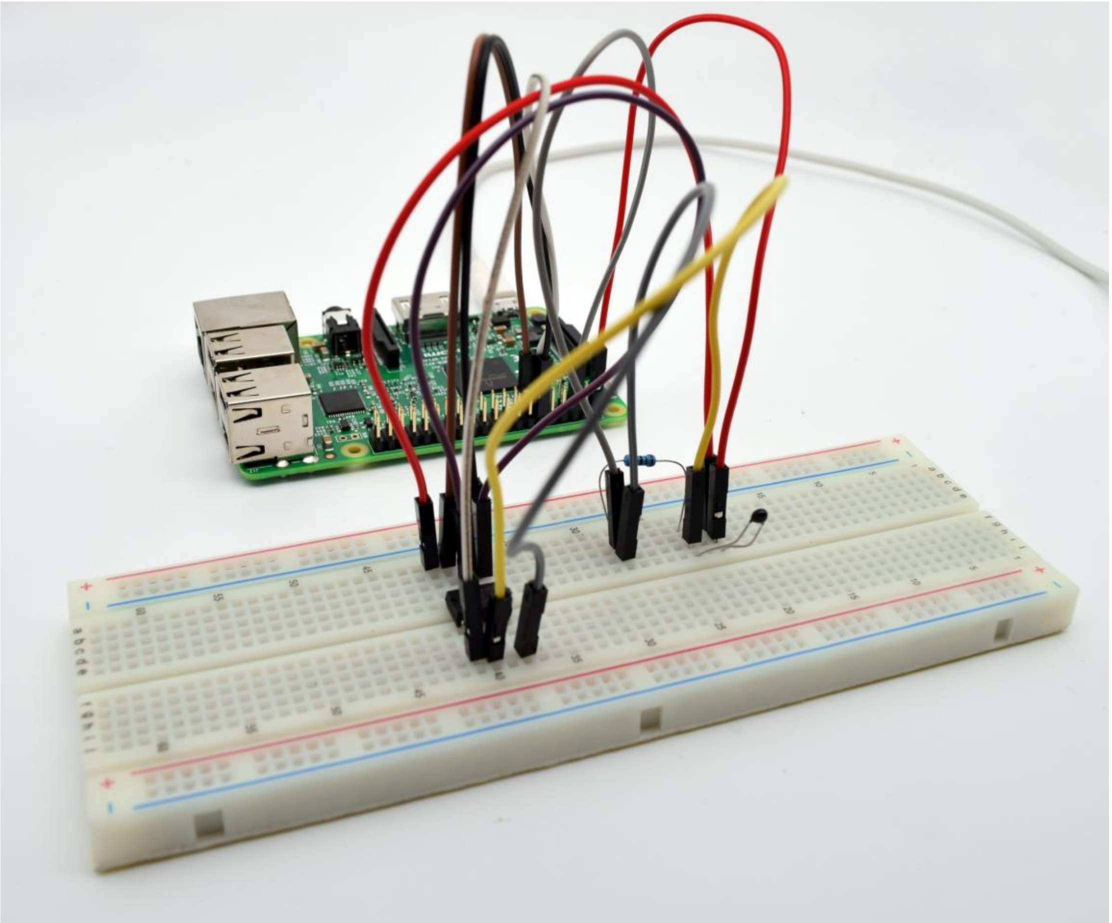
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/18_thermistor.py)

Step 3: Run

$ sudo python 18_thermistor.py

Now, touch the thermistor and you can see the current temperature value displayed on the screen, which changes accordingly.



Lesson 19 RFID

Overview

In this lesson, we will learn how to use an RFID Module. We program the Raspberry Pi to read the data acquired by the RFID module, and then display the ID data on the terminal.

Components

- 1* Raspberry Pi
- 1* RFID module
- 1* ID Card
- 1* Special-shaped ID Card
- Several jumper wires

Principle

The RFID technology is used for a wide variety of applications including access control, package identification, warehouse stock control, point-of-sale scanning, retail antitheft systems, toll-road passes, surgical instrument inventory, and even for identifying individual sheets of paper placed on a desk. RFID tags are embedded in name badges, shipping labels, library books, product tags and boxes; installed in aircraft; hidden inside car keys; and implanted under the skin of animals or even people. RFID systems work on a wide range of frequencies, have a variety of modulation and encoding schemes, and vary from low-power passive devices with range of only a few millimeters to active systems that work for hundreds of kilometers.

However, all RFID systems have the same basic two-part architecture: a reader and a transponder. The reader is an active device that sends out a signal and listens for responses, and the transponder (the part generally called the "tag") detects the signal from a reader and automatically sends back a response containing its identity code.

A reader can be like this:

A transponder:

Different types of RFID tags fall into one of the three broad categories: active, passive, and battery-assisted passive.

Active tags are physically large because they require their own power supply such as a battery. They can also have a very long range because the availability of local power allows them to send high-powered responses that can travel from tens of meters to hundreds of kilometers. An active tag is essentially a combination of a radio receiver to detect the challenge, some logic to formulate a response, and a radio transmitter to send back the response. They can even have the challenge and response signals operate on totally different frequencies. The downsides are the size of the tag, a high manufacturing cost due to the number of parts required, and the reliance on a battery that will go flat eventually.

Passive tags can be much smaller and cheaper than active tags because they don't require a local power supply and have much simpler circuitry. Instead of supplying their own power, they leach all the power they need from the signal sent by the reader. Early passive tags operated on the "Wiegand effect," which uses a specially formed wire to convert received electromagnetic energy into radio-wave pulses. Some early passive RFID tags actually consisted of nothing more than a number of very carefully formed wires made from a combination of cobalt, iron, and vanadium, with no other parts at all.

Modern passive tags use a clever technique that uses current induced in their antenna coil to power the electronics required to generate the response. The response is then sent by modulating the reader's own field, and the reader detects the modulation as a tiny fluctuation in the voltage across the transmitter coil. The result is that passive tags can be incredibly small and extremely inexpensive: the antenna can be a simple piece of metal foil, and the microchips are produced in such large quantities that a complete RFID-enabled product label could cost only a few cents and be no thicker than a normal paper label. Passive tags can theoretically last indefinitely because they don't contain a battery to go flat, but their disadvantage is a very short operational range due to the requirement to leach power from the reader's signal, and lack of an actively powered transmitter to send back the response.

Passive tags typically operate over a range of a few millimeters up to a few meters.

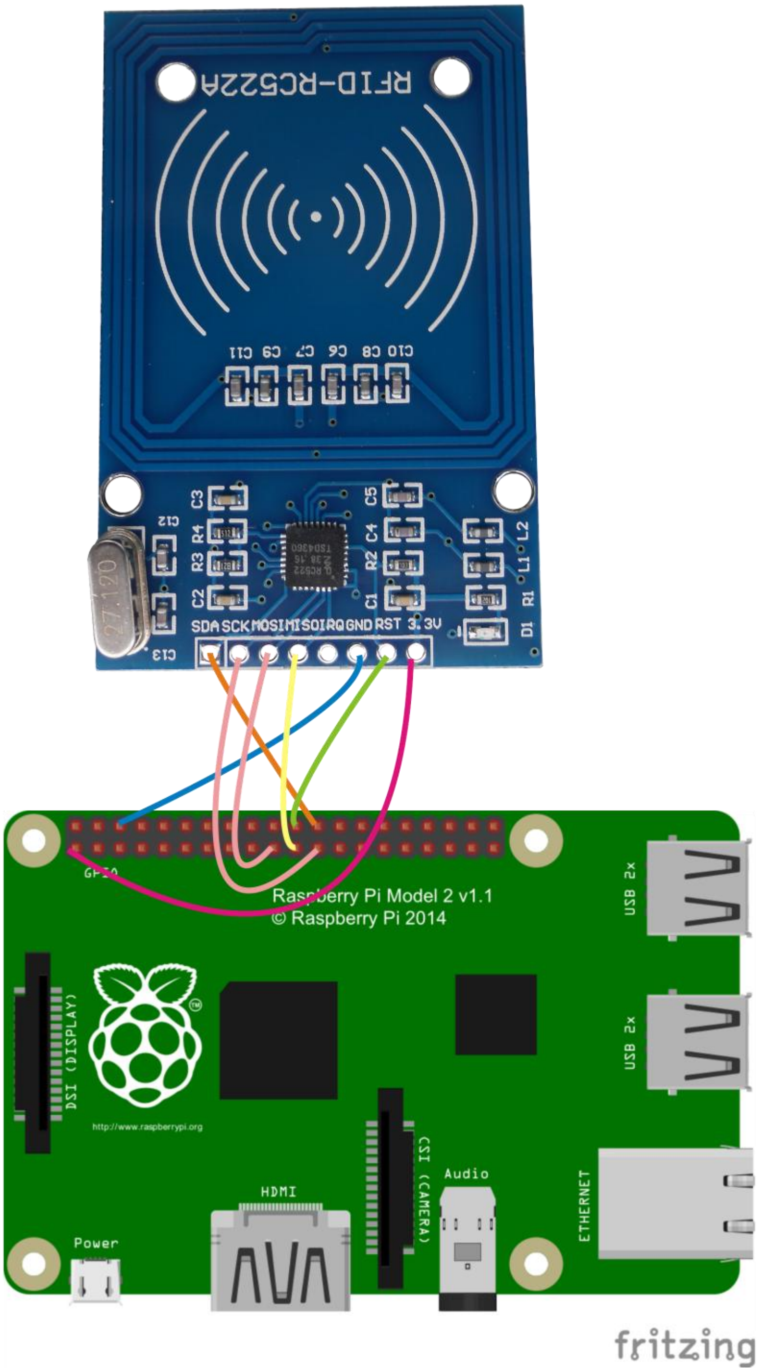
Tags can also have a variety of different modulation schemes, including AM, PSK, and ASK, and different encoding systems. With so many incompatible variations, it's sometimes hard to know if specific tags and readers are compatible. Generally speaking, each type of tag will only function on one specific frequency, modulation scheme, and communications protocol. Readers, on the other hand, are far more flexible and will often support a range of modulation schemes and comms protocols, but are usually still limited to just one frequency due to the tuning requirements of the coil.

Apart from the specific requirements for communicating with them, tags can also have a number of different features. The most common passive tags simply contain a hard-coded unique serial number and when interrogated by a reader they automatically respond with their ID code. Most tags are read-only so you can't change the value they return, but some types of tags are read/write and contain a tiny amount of rewritable storage so you can insert data into them using a reader and retrieve it later. However, most uses of RFID don't rely on any storage within the tag, and merely use the ID code of the tag as a reference number to look up information about it in an external database or other system.

RFID tags are produced in a wide variety of physical form factors to suit different deployment requirements. The most commonly seen form factor is a flat plastic card the same size as a credit card, often used as an access control pass to gain access to office buildings or other secure areas. The most common form by sheer number produced, even though you might not notice them, is RFID-enabled stickers that are commonly placed on boxes, packages, and products. Key fob tags are also quite common, designed to be attached to a keyring so they're always handy for operating access control systems.

Procedures

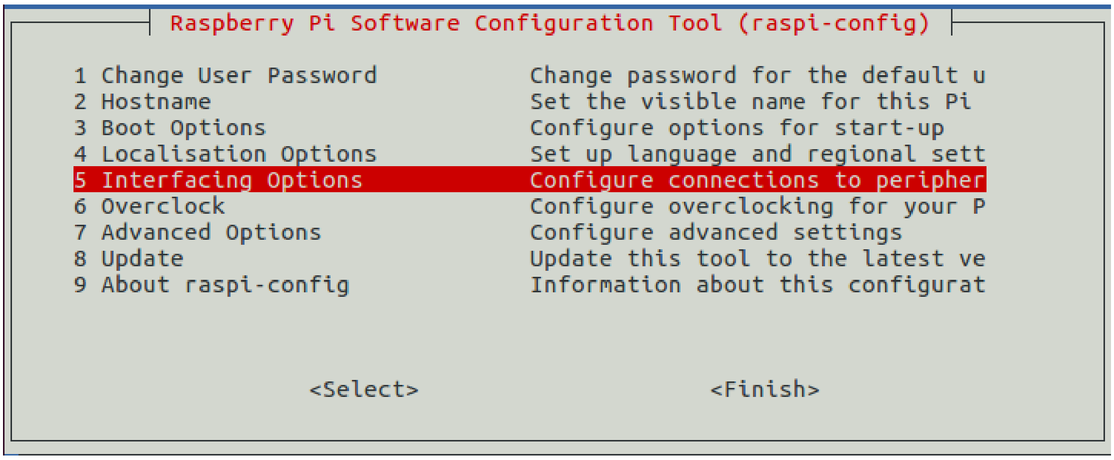
Step 1: Build the circuit



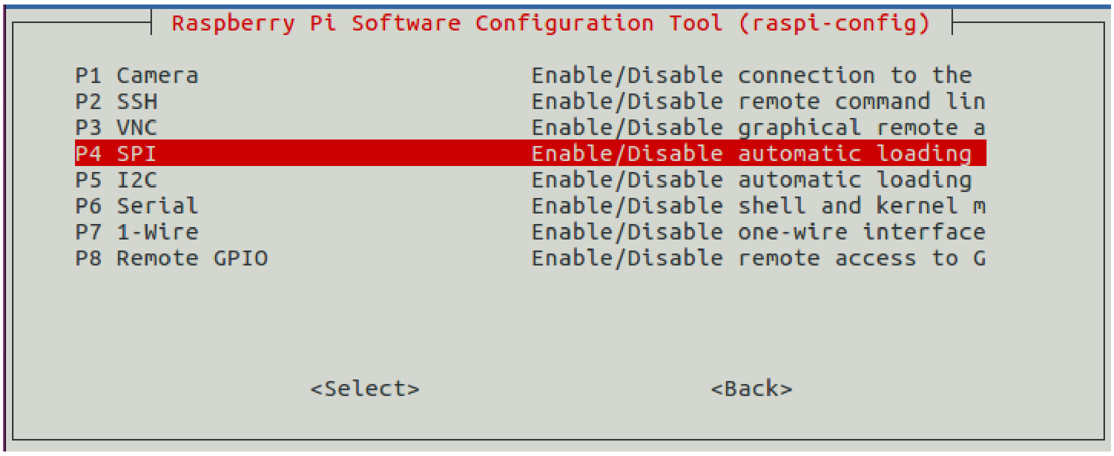
Step 2: The Raspberry communicates with the RC522 RFID module via the SPI interface. The interface is not enabled by default, and need some extra configuration before you can use it. You can use raspi-config to enable it.

$ sudo raspi-config

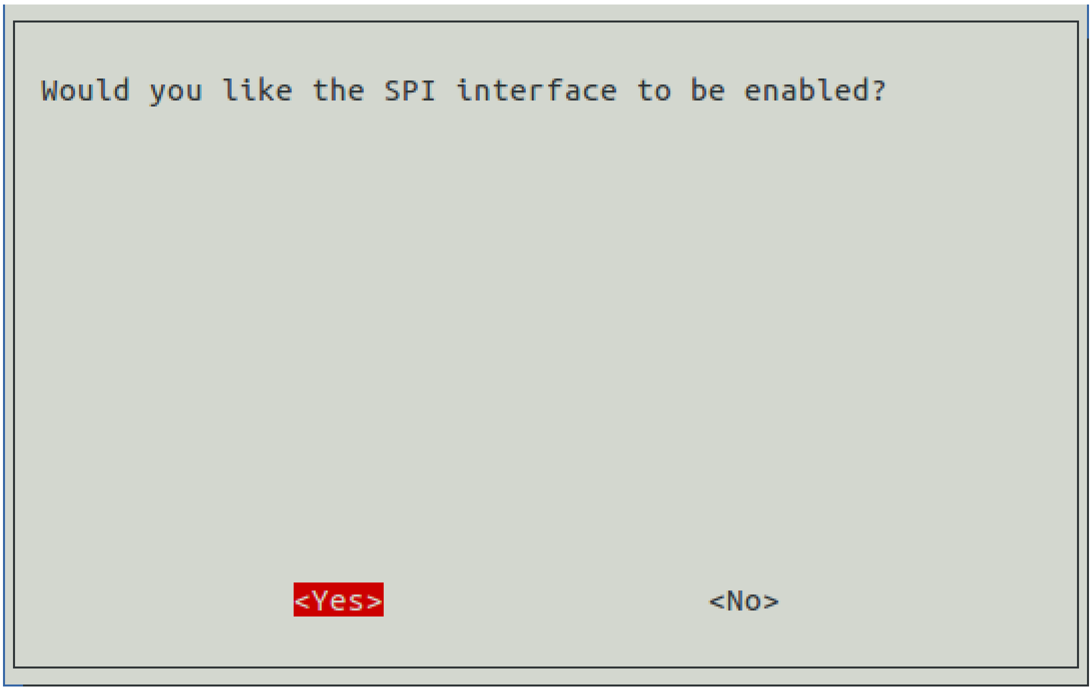
Use the down arrow to select "5 Interfacing Options"



Arrow down to "P4 SPI"



Select "Yes" when it asks you to enable SPI.



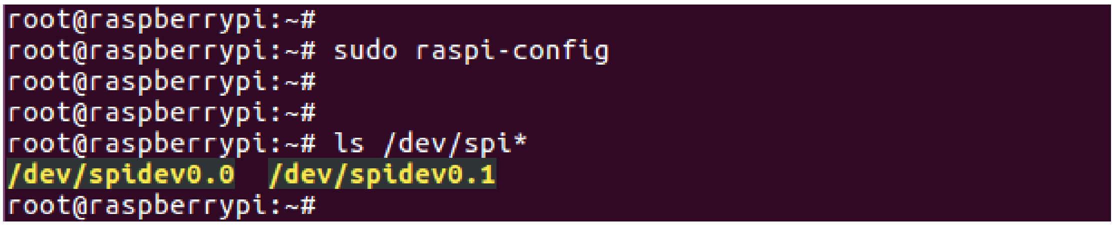
Also select “Yes” when it tasks about automatically loading the kernel module.

Use the right arrow to select the “Finish” button.

Select “Yes” when it asks to reboot.

The system will reboot. When it comes back up, log in and enter the following command:

$ ls /dev/spi*



```
root@raspberrypi:~#
root@raspberrypi:~# sudo raspi-config
root@raspberrypi:~#
root@raspberrypi:~#
root@raspberrypi:~# ls /dev/spi*
/dev/spidev0.0  /dev/spidev0.1
root@raspberrypi:~#
```

The Pi should respond with

/dev/spidev0.0 /dev/spidev0.1

These represent SPI devices on chip enable pins 0 and 1, respectively. These pins are hardwired within the Pi. Ordinarily, this means the interface supports at most two peripherals, but there are cases where multiple devices can be daisy-chained, sharing a single chip enable signal.

For C language users:

Step 3: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/19_RFID)

Step 4: Compile

sudo ./compile.sh

Step 5: Run

sudo ./rfid_test

RC522>scan

```
root@raspberrypi:/home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/19_RFID# ls
compile.sh  main.c          mfrc522.h           rfid_test
dump.c      mfrc522.c       mfrc522_hal_linux.c TI_aes_128.c
dump.h      mfrc522_debug.c README.md           TI_aes_128.h
root@raspberrypi:/home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/19_RFID# ./rfid_t
est
Try to open device /dev/spidev0.0
Device opened
Device Number:3
SPI mode [OK]
SPI word bits[OK]
SPI max speed[OK]
User Space RC522 Application
RC522>scan
```

Now, put the ID card close to the RFID reader and you can see the ID number sent to the terminal.

```
RC522>scan


Scanning
................................................................................
................................................................................
......................................................................Card detec
ted      0x96 0x6E 0x01 0xA4, Check Sum = 0x5D
Card Selected, Type:PICC_TYPE_MIFARE_1K
RC522>966E01A4>scan
scan
Usage:
        read <blockstart>
        dump
        writestr <blockaddr>
................................................................................
................................................................................
................................................................Card detected    0
x85 0x5E 0xE9 0xAB, Check Sum = 0x99
Card Selected, Type:PICC_TYPE_MIFARE_1K
RC522>855EE9AB>
```

For Python users:

Step 3: Install the header files and static libraries for python dev

$ sudo apt-get install python-dev   python3-dev

```
root@raspberrypi:/home# apt-get install python-dev python3-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
python3-dev is already the newest version.
python3-dev set to manually installed.
The following extra packages will be installed:
  libpython-dev libpython2.7-dev python2.7-dev
The following NEW packages will be installed:
  libpython-dev libpython2.7-dev python-dev python2.7-dev
0 upgraded, 4 newly installed, 0 to remove and 0 not upgraded.
Need to get 18.2 MB of archives.
After this operation, 25.7 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Step 4: Install SPI Python Module

$ sudo git clone https://github.com/lthiery/SPI-Py.git

```
root@raspberrypi:/home#
root@raspberrypi:/home# git clone https://github.com/lthiery/SPI-Py.git
Cloning into 'SPI-Py'...
remote: Counting objects: 81, done.
remote: Total 81 (delta 0), reused 0 (delta 0), pack-reused 81
Unpacking objects: 100% (81/81), done.
Checking connectivity... done.
root@raspberrypi:/home#
```

```
root@raspberrypi:/home#
root@raspberrypi:/home# ls
Adeept_RFID_Learning_Kit_C_Code_for_RPi
Adeept_RFID_Learning_Kit_Python_Code_for_RPi
Adeept_RFID_RC522_C_Code
Adeept_Sensor_Kit_for_RPi_Python_Code
Adeept_Ultimate_Starter_Kit_Python_Code_for_RPi
pi
SPI-Py
root@raspberrypi:/home#
```

$ cd SPI-Py

```
root@raspberrypi:/home#
root@raspberrypi:/home# cd SPI-Py/
root@raspberrypi:/home/SPI-Py#
root@raspberrypi:/home/SPI-Py# ls
README.md  setup.py  spi.c  SPIGames  spi_multi.c  test_multi.py  test-nRF.py
root@raspberrypi:/home/SPI-Py#
```

$ sudo python setup.py install

```
root@raspberrypi:/home/SPI-Py#
root@raspberrypi:/home/SPI-Py# python setup.py install
```

```
root@raspberrypi:/home/SPI-Py# python setup.py install
running install
running build
running build_ext
building 'spi' extension
creating build
creating build/temp.linux-armv7l-2.7
arm-linux-gnueabihf-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototyp
es -fno-strict-aliasing -D_FORTIFY_SOURCE=2 -g -fstack-protector-strong -Wformat
 -Werror=format-security -fPIC -I/usr/include/python2.7 -c spi.c -o build/temp.l
inux-armv7l-2.7/spi.o
creating build/lib.linux-armv7l-2.7
arm-linux-gnueabihf-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-z
,relro -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -D
_FORTIFY_SOURCE=2 -g -fstack-protector-strong -Wformat -Werror=format-security -
Wl,-z,relro -D_FORTIFY_SOURCE=2 -g -fstack-protector-strong -Wformat -Werror=for
mat-security build/temp.linux-armv7l-2.7/spi.o -o build/lib.linux-armv7l-2.7/spi
.so
running install_lib
copying build/lib.linux-armv7l-2.7/spi.so -> /usr/local/lib/python2.7/dist-packa
ges
running install_egg_info
Removing /usr/local/lib/python2.7/dist-packages/SPI_Py-1.0.egg-info
Writing /usr/local/lib/python2.7/dist-packages/SPI_Py-1.0.egg-info
```

Step 5: Edit and save the code with vim or nano.

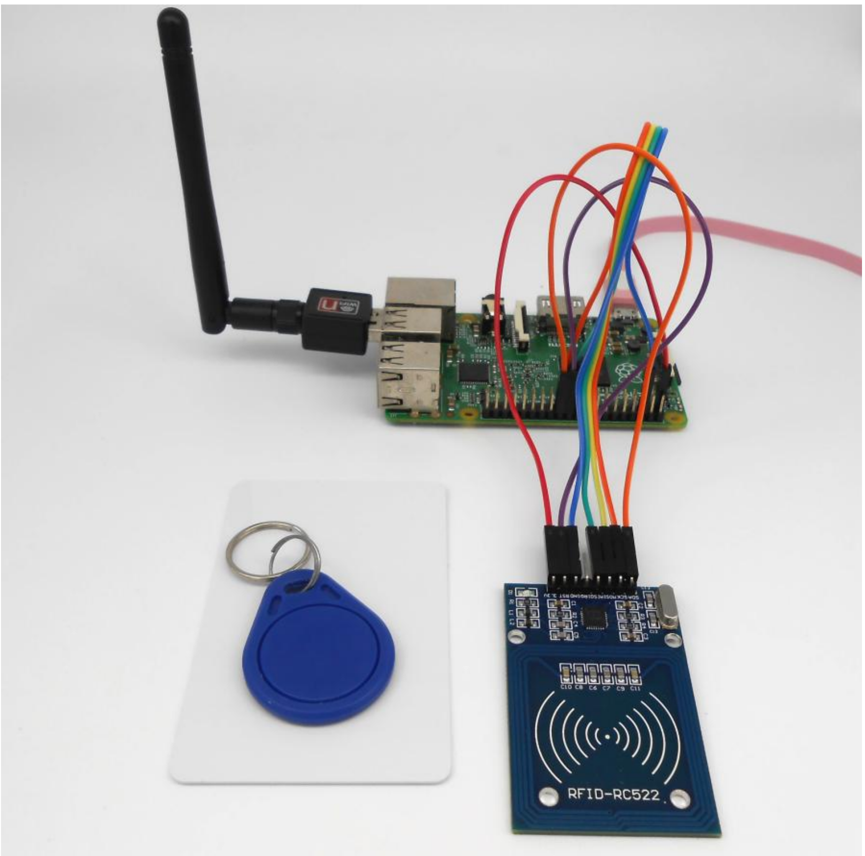
(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/19_RFID/MFRC522-python/)

Step 6: Run

$ sudo python Read.py

Now, put the ID card close to the RFID reader and you can see the ID number sent to the terminal.

```
root@raspberrypi:/home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/19_RFID/MFRC
522-python# ls
Dump.py  MFRC522.py  MFRC522.pyc  README.md  Read.py  Write.py
root@raspberrypi:/home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/19_RFID/MFRC
522-python# python Read.py
Welcome to the MFRC522 data read example
Press Ctrl-C to stop.
Card detected
Card read UID: 150,110,1,164
Size: 8
Sector 8 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Card detected
Card read UID: 150,110,1,164
Size: 8
Sector 8 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
Card detected
Card read UID: 133,94,233,171
Size: 8
Sector 8 [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```


RFID-RC522

Lesson 20 LED Bar Graph

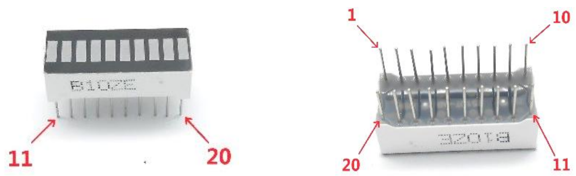
Overview

In this lesson, we will learn how to control an LED bar graph by programming the Raspberry Pi.

Components

- 1* Raspberry Pi
- 1* ADC0832
- 1* LED bar graph
- 10* 220Ω Resistor
- 1* 10KΩ Potentiometer
- 1* Breadboard
- Several jumper wires

Principle



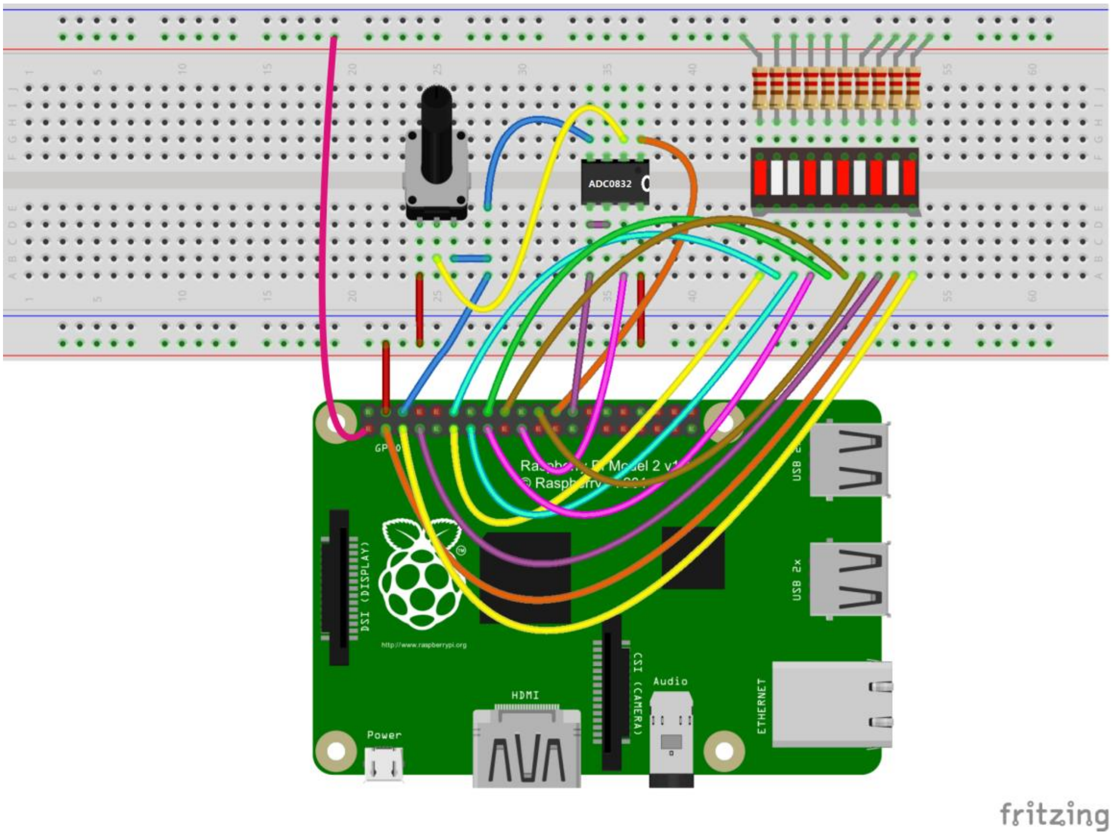
The bar graph - a series of LEDs in a line, such as you see on an audio display - is a common hardware display for analog sensors. It's made up of a series of LEDs in a row, an analog input like a potentiometer, and a little code in between. You can buy multi-LED bar graph displays fairly cheaply. This tutorial demonstrates how to control a series of LEDs in a row, but can be applied to any series of digital outputs.

This tutorial borrows from the For Loop and Arrays tutorial as well as the Analog Input tutorial.

The sketch works like this: first read the analog value. Map the value to the output range which is 0-10 in this case since ten LEDs are used. As the analog value changes, LEDs in a corresponding number will light up on the bar – the bigger the value is, the more LEDs will be turned on.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/20_ledBar/ledBar.c)

Step 3: Compile

```
$ gcc ledBar.c -o ledBar -lwiringPi
```

Step 4: Run

```
$ sudo ./ledBar
```

For Python users:

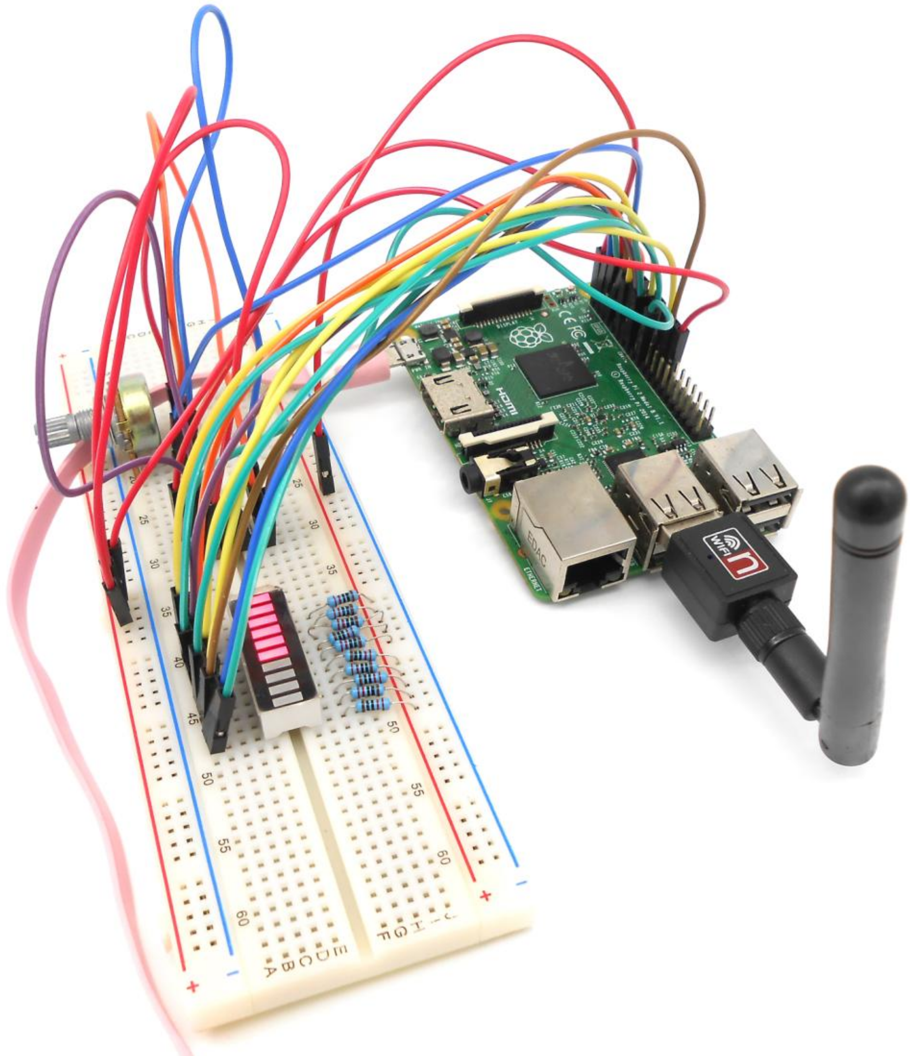
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/20_ledBar/ledBar.py)

Step 3: Run

```
$ sudo python ledBar.py
```

Now, when you turn the shaft of the potentiometer, you will see that the number of LEDs lit in the LED bar graph changing.



Lesson 21 Controlling an LED Through LAN

Overview

In this lesson, we will introduce TCP and socket, and then how to program the Raspberry Pi to control an LED through the local area network (LAN).

Components

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

Principle

1. TCP

The Transmission Control Protocol (TCP) is a core protocol of the Internet Protocol Suite. It originated in the initial network implementation in which it complemented the Internet Protocol (IP). Therefore, the entire suite is commonly referred to as TCP/IP. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network. TCP is the protocol that major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on. Applications that do not require reliable data stream service may use the User Datagram Protocol (UDP), which provides a connectionless datagram service that emphasizes reduced latency over reliability.

2. Socket

A network socket is an endpoint of an inter-process communication across a computer network. Today, most communication between computers is based on the Internet Protocol; therefore most network sockets are Internet sockets.

A socket API is an application programming interface (API), usually provided by the operating system, that allows application programs to control and use network sockets. Internet socket APIs are usually based on the Berkeley sockets standard.

A socket address is the combination of an IP address and a port number, much like one end of a telephone connection is the combination of a phone number and a particular extension. Based on this address, internet sockets deliver incoming data packets to the appropriate application process or thread.

Several Internet socket types are available:

1. Datagram sockets, also known as connectionless sockets, which use User Datagram Protocol (UDP).

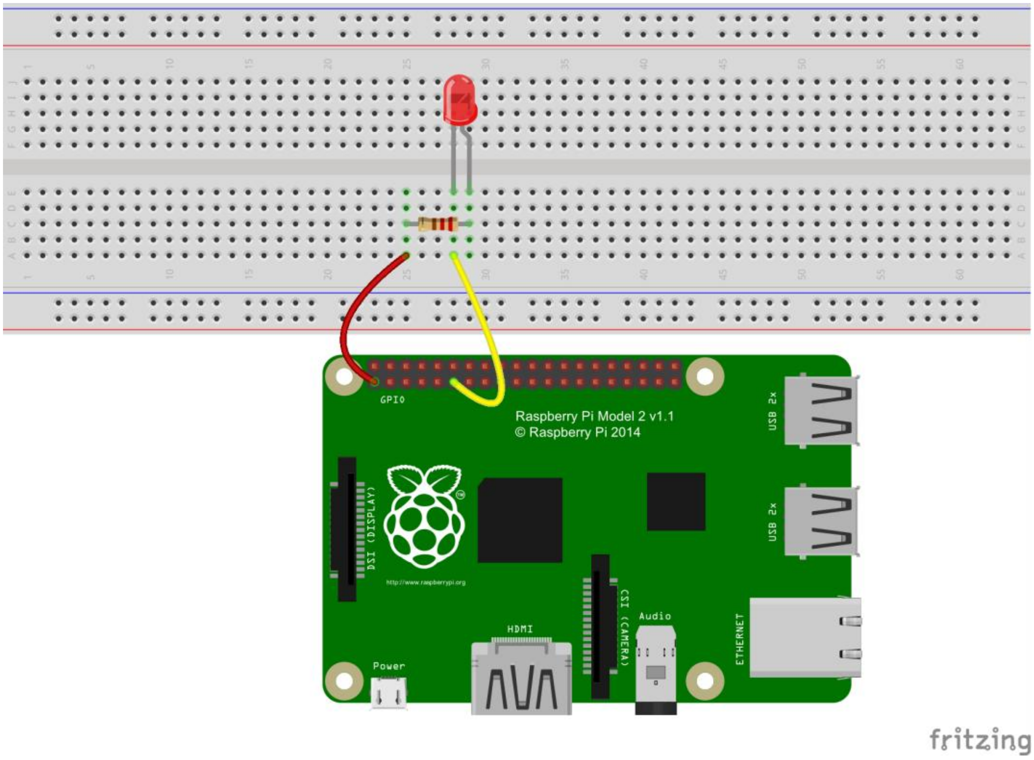
2. Stream sockets, also known as connection-oriented sockets, which use Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP).

3. Raw sockets (or Raw IP sockets), typically available in routers and other network equipment. Here the transport layer is bypassed, and the packet headers are made accessible to the application.

In this experiment, our program is based on stream socket, and the program is divided into two parts, the client and the server. The server routine is run on the Raspberry Pi, and the client routine is run on the PC. So you can send command to the server through the client, and then control the LED connected to the Raspberry Pi.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the server code with vim or nano on the Raspberry Pi.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/21_TCPCtrlLed/ledServer.c)

Step 3: Compile(On Raspberry Pi)

$ gcc ledServer.c -o ledServer -lwiringPi

Step 4: Edit and save the client code with vim or nano on the PC.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/21_TCPCtrlLed/client.c)

Step 5: Compile (On Linux PC)

$ gcc ledClient.c -o ledClient

Step 6: Run

$ sudo ./ledServer (On Raspberry Pi)

$ ./ledClient 192.168.1.188 (On PC, modify the IP Address to your Raspberry Pi's IP Address)

Now, input "ON" in the terminal and then press Enter. The LED connected to the Raspberry Pi will light up; input "OFF" and the LED goes out.

For Python users:

Step 2: Edit and save the server code with vim or nano on the Raspberry Pi.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/21_TCPCtrlLed/ledServer.py)

Step 3: Edit and save the client code with vim or nano on the PC. Modify the IP Address to your Raspberry Pi's IP Address.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/12_TCPCtrlLed/ledClient.py)

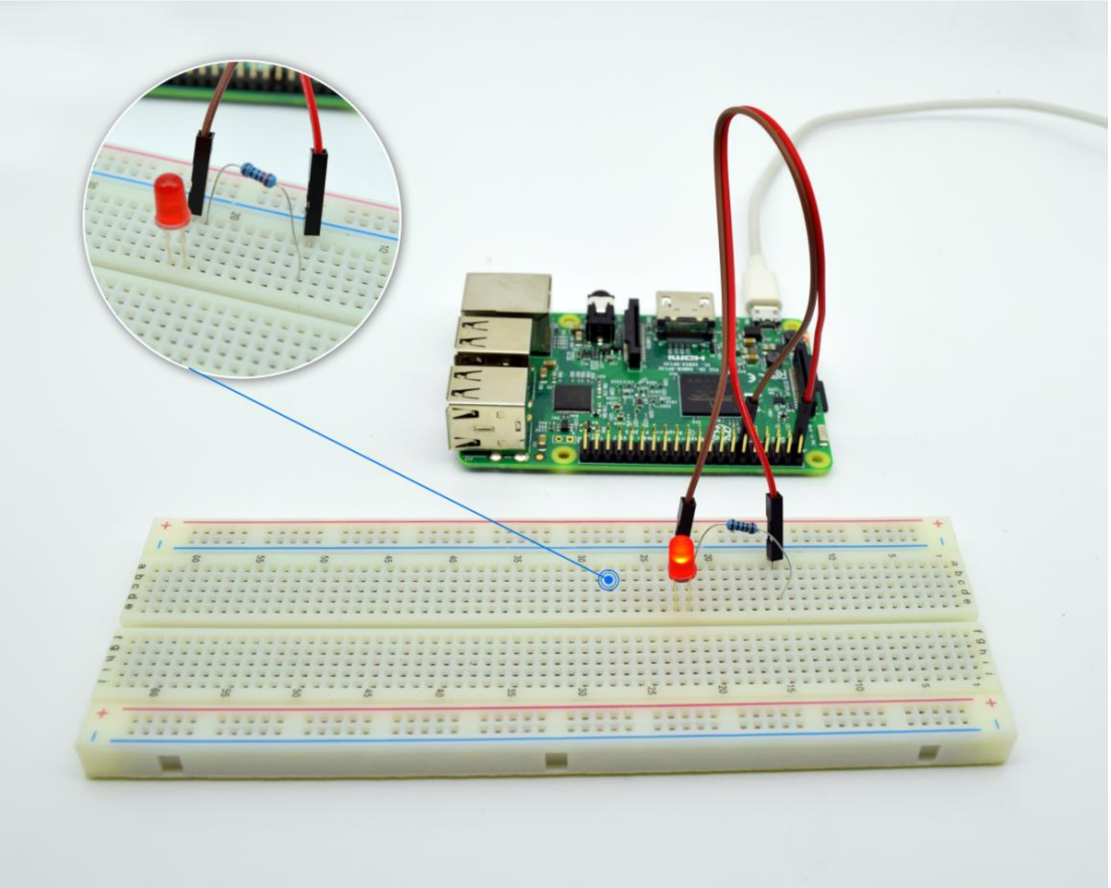
```
#!/usr/bin/env python
from socket import *

HOST = '192.168.0.114'    # Server(Raspberry Pi) IP address
PORT = 8080
BUFSIZ = 1024            # buffer size
ADDR = (HOST, PORT)

tcpCliSock = socket(AF_INET, SOCK_STREAM)   # Create a socket
tcpCliSock.connect(ADDR)                    # Connect with the server

def loop():
        while True:
                cmd = raw_input('input cmd : ')
                tcpCliSock.send(cmd)

if __name__ == '__main__':
        try:
                loop()
        except KeyboardInterrupt:
                tcpCliSock.close()
```

Step 4: Run

$ sudo python ledServer.py (On Raspberry Pi)

$ python ledClient.py (On PC)

Now, input "ON" in the terminal and then press Enter. The LED connected to the Raspberry Pi will light up; input "OFF" and the LED goes out.



Summary

By learning this lesson, you should have mastered the basic principles of inter-computer communication. This lesson can help you open the door to learn the Internet of Things (IoT).

Lesson 22 DC Motor

Overview

In this comprehensive experiment, we will learn how to control the state of a DC motor with Raspberry Pi. The state of DC motors includes its forward, reverse, acceleration, deceleration and stop.

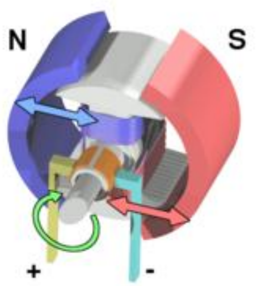
Components

- 1* Raspberry Pi
- 1* L9110 DC Motor Driver
- 1* DC motor
- 4* Button
- 1* LED
- 1* 220Ω Resistor
- 1* Capacitor (104, 0.1uF)
- 1* Breadboard
- Several jumper wires

Principle

1. DC motor

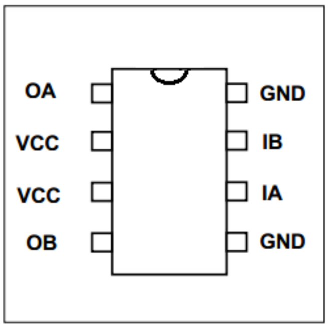
A DC motor is any of a class of electrical machines that converts direct current electrical power into mechanical power. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor. Most types produce rotary motion; a linear motor directly produces force and motion in a straight line.



DC motors were the first type widely used, since they could be powered from existing direct-current lighting power distribution systems. A DC motor's speed can be controlled over a wide range, using either a variable supply voltage or by changing the strength of current in its field windings. Small DC motors are used in tools, toys, and appliances. The universal motor can operate on direct current but is a lightweight motor used for portable power tools and appliances.

2. L9110

L9110 is a driver chip which is used to control and drive motor. The chip has two TTL/CMOS compatible input terminals, and possesses the property of anti-interference: it has high current driving capability, two output terminals that can directly drive DC motor, each output port can provide 750~800mA dynamic current, and its peak current can reach 1.5~2.0A; L9110 is widely applied to various motor drives, such as toy cars, stepper motor, power switches and other electric circuits.



OA, OB: These are used to connect the DC motor.

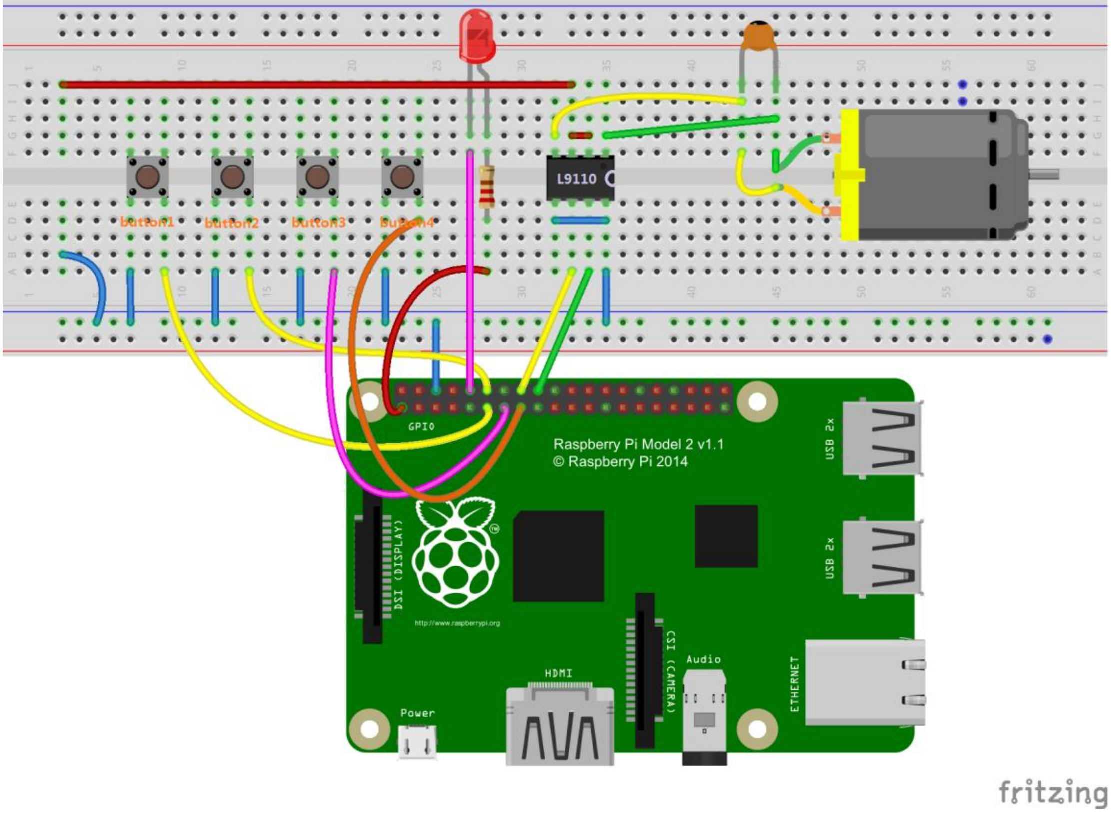
VCC: Power supply (+5V)

GND: The cathode of the power supply (Ground).

IA, IB: The input terminal of drive signal.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/22_motor/motor.c)

Step 3: Compile

$ gcc motor.c -o motor –lwiringPi -lpthread

Step 4: Run

$ sudo ./motor

For Python users:

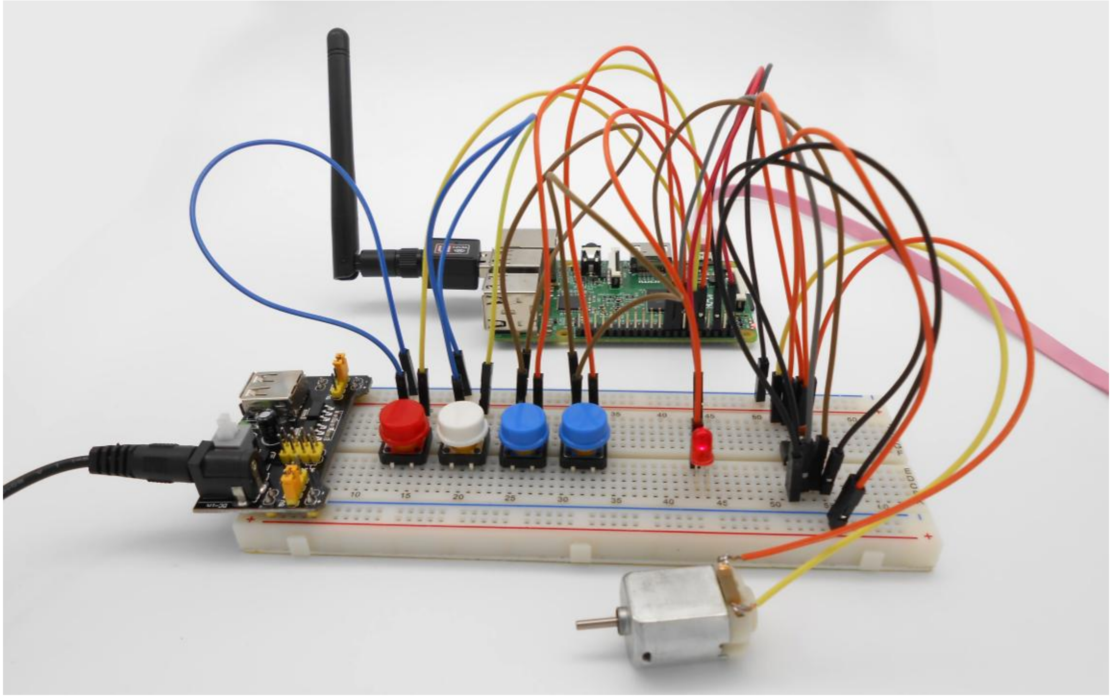
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/22_motor.py)

Step 3: Run

$ sudo python 22_motor.py

Press button 1 to stop or run the DC motor; press button 2 to make the DC motor move forward or reverse; press button 3 to accelerate the DC motor; press button 4 to decelerate the DC motor. When the motor is running, the LED will light up. Otherwise, the LED will stay off.



Summary

After learning, you must have grasped the basic theory and programming of the DC motor. You can not only make it move forward and reverse, but also regulate its speed. Besides, you can do some interesting applications with what you've got in this lesson and the knowledge acquired previously.

Lesson 23 Controlling a Stepper Motor

Overview

In this lesson, we will introduce a new electronic device – stepper motor, and you can also learn how to control it with Raspberry Pi.

Components

- 1* Raspberry Pi
- 1* Stepper motor
- 1* ULN2003 stepper motor driver module
- Several jumper wires

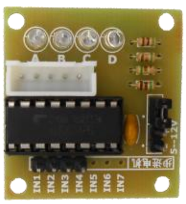
Principle

1. Stepper motor

Stepper motors, due to their unique design, can be controlled to a high degree of accuracy without any feedback mechanisms. The shaft of a stepper, mounted with a series of magnets, is controlled by a series of electromagnetic coils that are charged positively and negatively in a specific sequence, precisely moving it forward or backward in small "steps".

There are two types of steppers, Unipolars and Bipolars, and it is very important to know which type you are working with. In this experiment, we will use a Unipolar stepper.

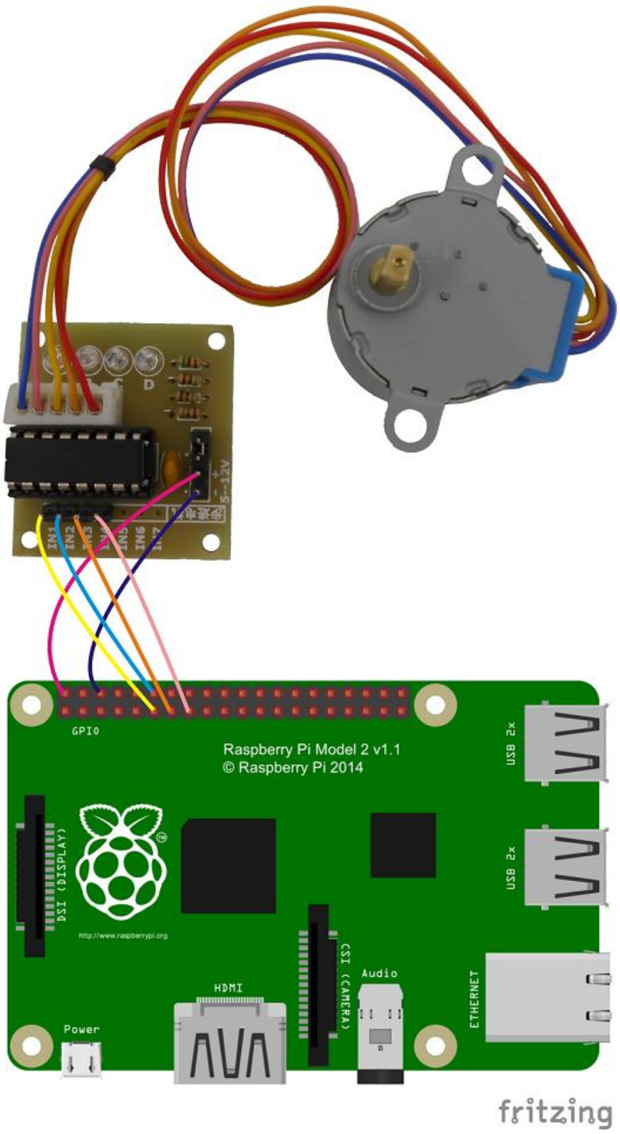
2. ULN2003 driver module



The Raspberry Pi's GPIO cannot directly drive a stepper motor due to the weak current. Therefore, a driver circuit is necessary for controlling a stepper motor. What we used in this experiment is a ULN2003-based driver module. There are four LEDs on the module. The white socket in the middle is to connect a stepper motor. IN1, IN2, IN3, IN4 are to connect with the Raspberry Pi.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/23_stepperMotor/stepperMotor.c)

Step 3: Compile

$ gcc stepperMotor.c -o stepperMotor -lwiringPi

Step 4: Run

$ sudo ./stepperMotor

For Python users:

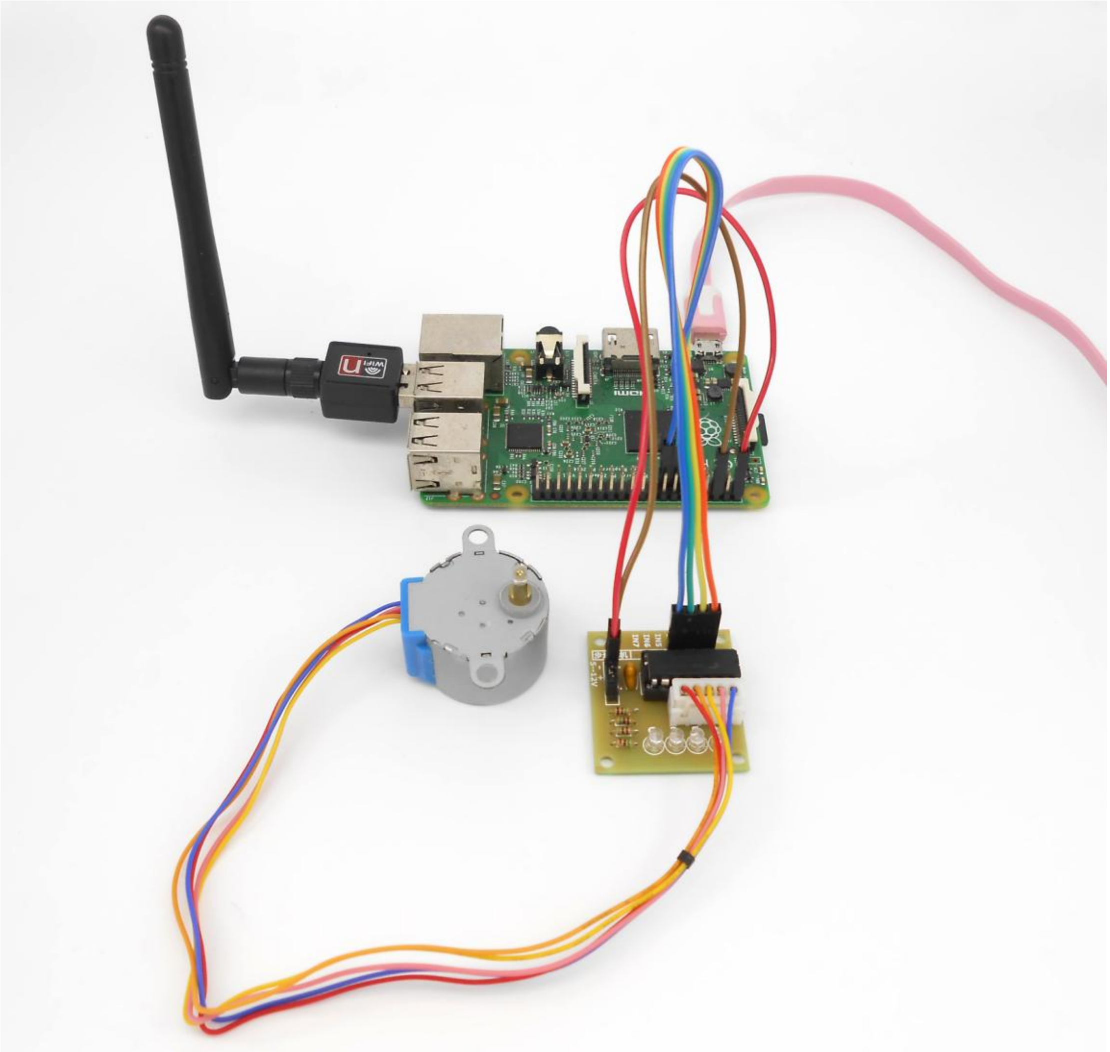
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/23_stepperMotor.py)

Step 3: Run

$ sudo python 23_stepperMotor.py

Now you should see that the stepper motor spinning.



Lesson 24 Acceleration Sensor ADXL345

Overview

In this lesson, we will learn how to use an acceleration sensor ADXL345 to get the acceleration data.

Components

- 1* Raspberry Pi
- 1* ADXL345 module
- Several jumper wires

Principle

1. ADXL345

The ADXL345 is a small, thin, ultralow power, 3-axis accelerometer with high resolution (13-bit) measurement at up to ±16g. Digital output data is formatted as 16-bit twos complement and is accessible through either a SPI (3-wire or 4-wire) or I2C digital interface. The ADXL345 is well suited for mobile device applications. It measures the static acceleration of gravity in tilt-sensing applications, as well as dynamic acceleration resulting from motion or shock. Its high resolution (3.9 mg/LSB) enables measurement of inclination changes less than 1.0°.

Low power modes enable intelligent motion-based power management with threshold sensing and active acceleration measurement at extremely low power dissipation.

2. Key functions

- int wiringPiI2CSetup (int devId)

This initialises the I2C system with your given device identifier. The ID is the I2C number of the device and you can use the **i2cdetect** program to find this out. wiringPiI2CSetup() will work out which revision Raspberry Pi you have and open the appropriate device in /dev.

The return value is the standard Linux filehandle, or -1 if any error – in which case, you can consult errno as usual.

- int wiringPiI2CRead (int fd)

Simple device read. Some devices present data when you read them without having to do any register transactions.

- int wiringPiI2CWriteReg8 (int fd, int reg, int data)
- int wiringPiI2CWriteReg16 (int fd, int reg, int data)

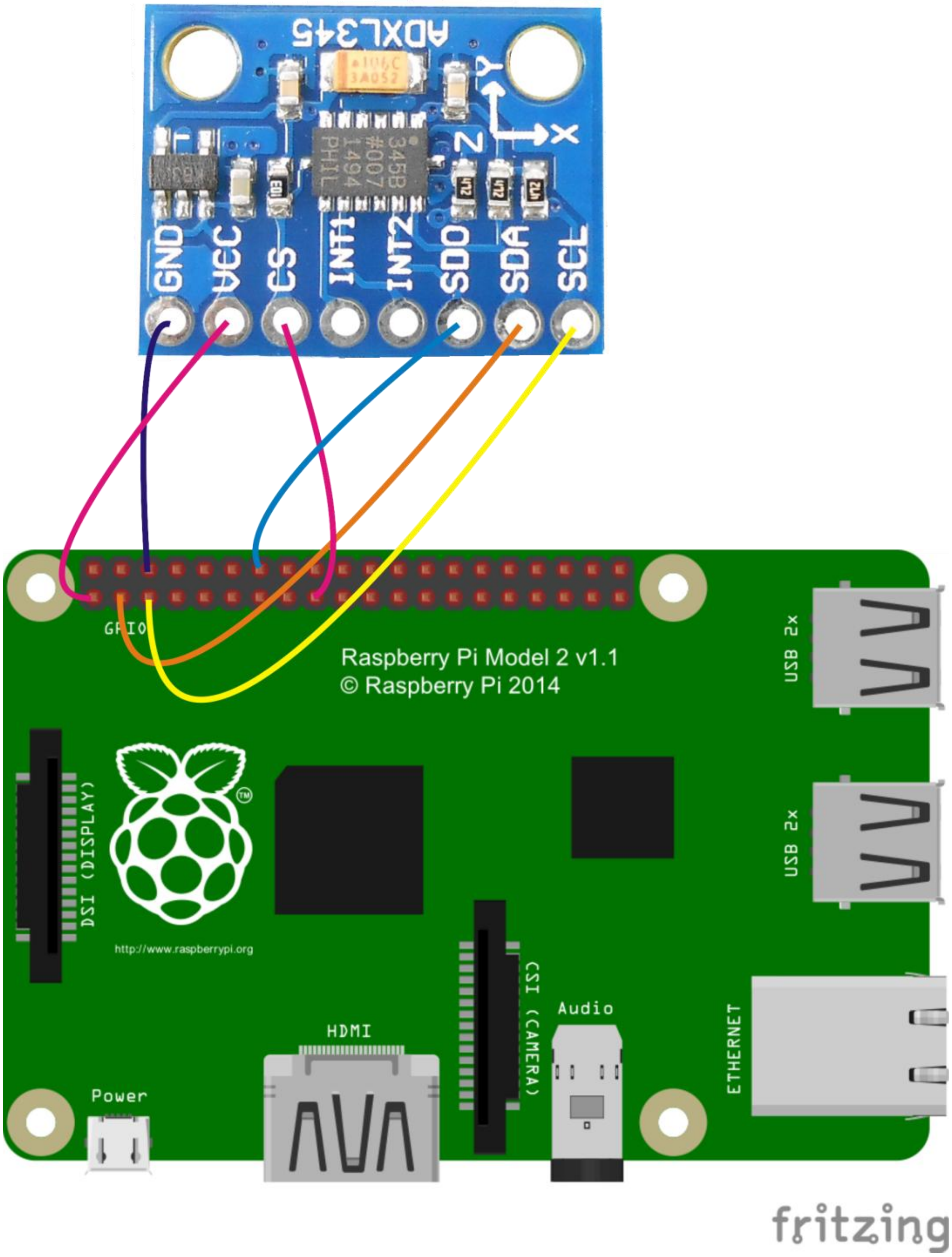
These write an 8 or 16-bit data value into the device register indicated.

- int wiringPiI2CReadReg8 (int fd, int reg)
- int wiringPiI2CReadReg16 (int fd, int reg)

These read an 8 or 16-bit value from the device register indicated.

Procedures

Step 1: Build the circuit

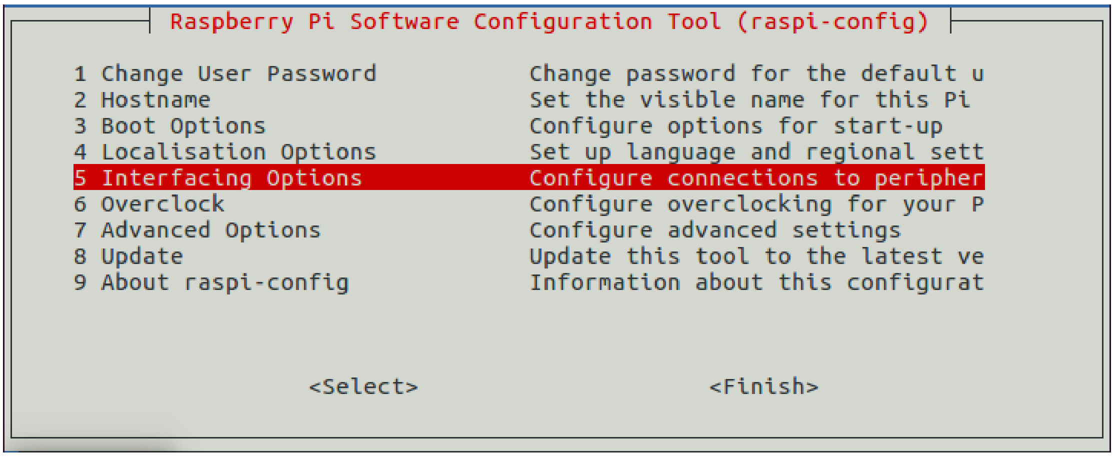


Step 2: Like the SPI peripheral, I2C is not turned on by default. You can use raspi-config to enable it.

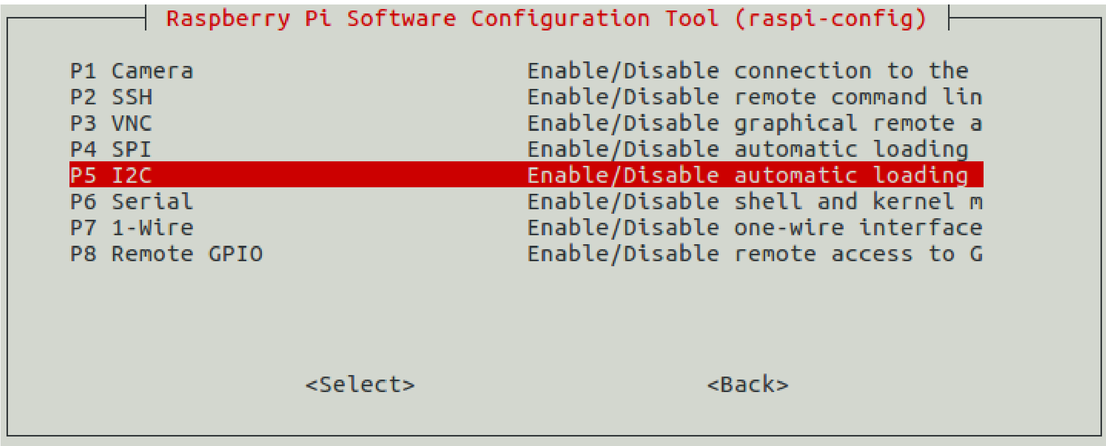
$ sudo raspi-config

```
root@raspberrypi:/home# raspi-config
```

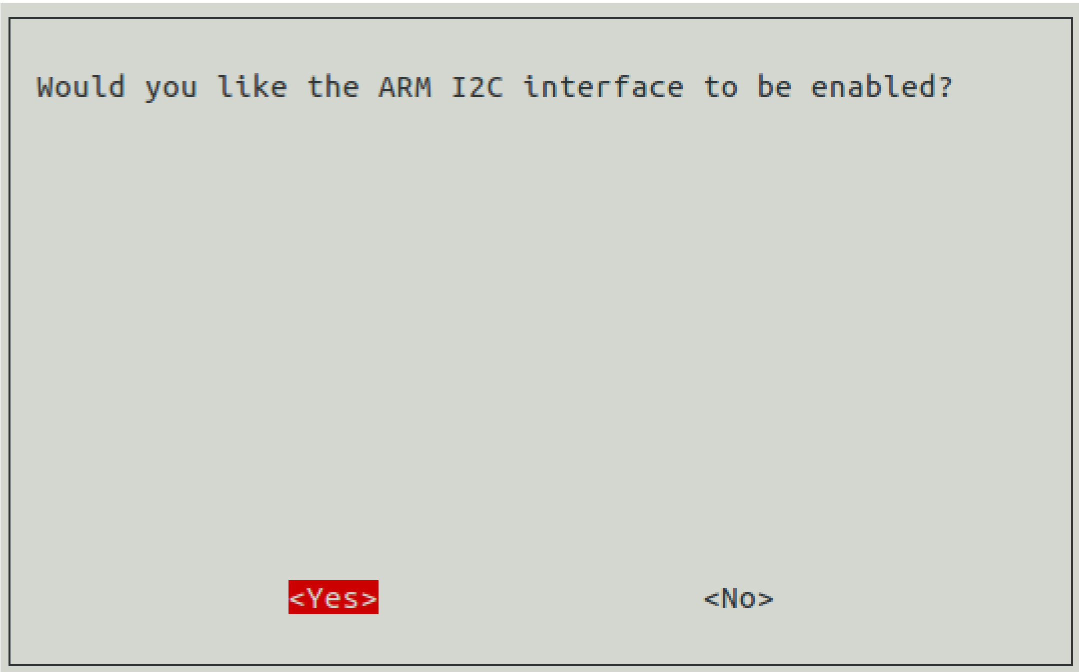
Use the down arrow to select “5 Interfacing Options”

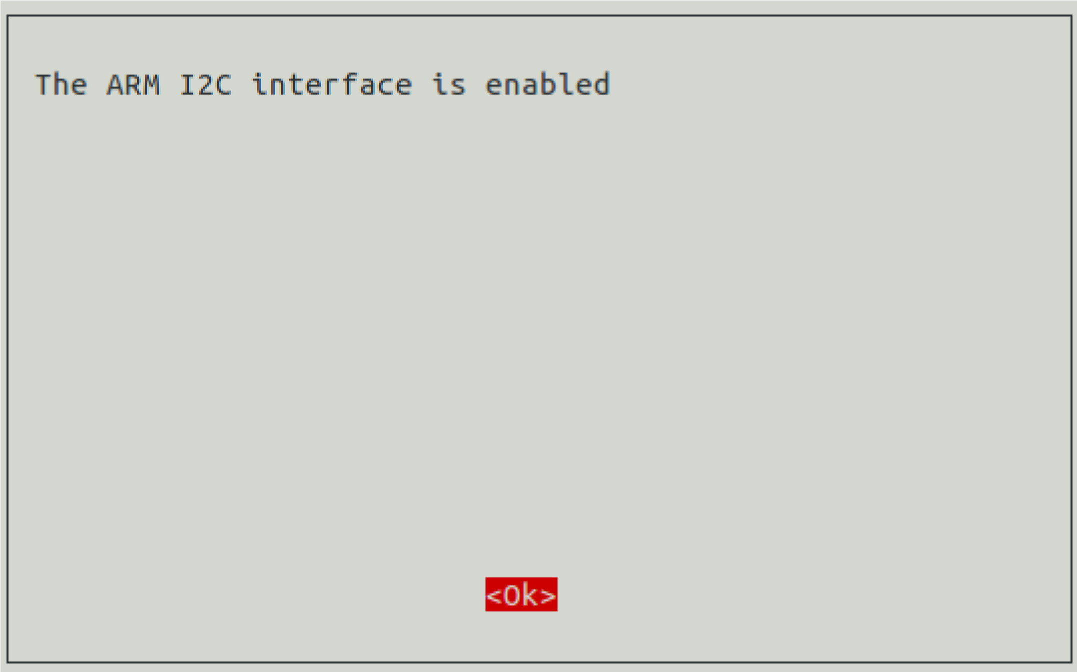


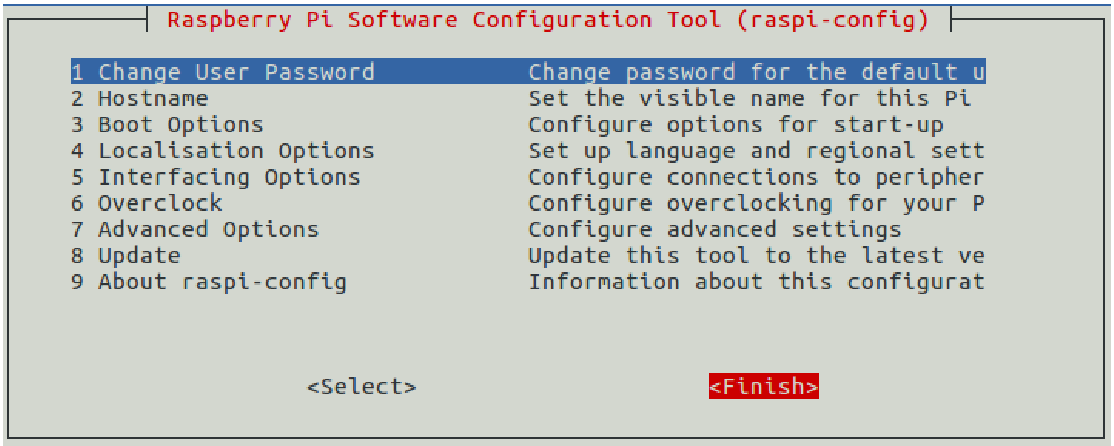
Arrow down to “P5 I2C”



Select “Yes” when it asks you to enable I2C.

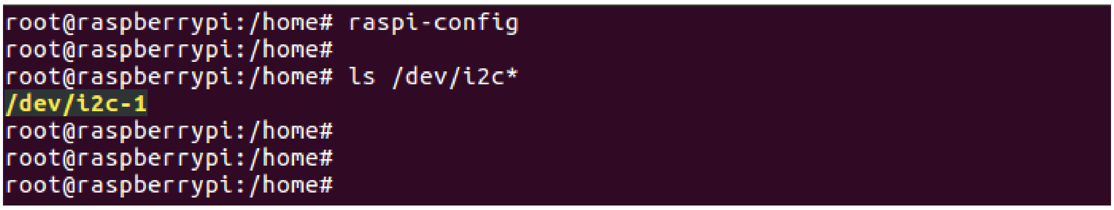






Also select “Yes” when it tasks about automatically loading the kernel module.
Use the right arrow to select the “Finish” button.
Select “Yes” when it asks to reboot.
The system will reboot. When it comes back up, log in and enter the following command:

$ ls /dev/*i2c*



The Pi should respond with:

/dev/i2c-1

Which represents the user-mode I2C interface.

For C language users:

Step 3: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/24_ADXL345/adxl345.c)

Step 4: Compile

$ gcc adxl345.c -o adxl345 -lwiringPi

Step 5: Run

$ sudo ./adxl345

```
root@raspberrypi:/home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/24_ADXL345# ./adx
l345
x:-0.023438
y:-0.015625
z:2.000000

x:-0.015625
y:-0.015625
z:2.000000

x:-0.015625
y:-0.015625
z:1.984375
```

For Python users:

Step 3: Install the required tools and libraries

$ sudo apt-get install build-essential libi2c-dev i2c-tools python-dev libffi-dev

$ sudo apt-get install python-smbus

Step 4: Install ADXL345 library

$ cd /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/24_ADXL345/

$ sudo python setup.py install

Step 5: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/24_ADXL345/examples/)

$ cd examples

Step 6: Run

$ sudo python simpletest.py

Now you should see the acceleration data displayed on the terminal.


SCL
SDA
SDO
INT2
INT1
CS
VCC
GND
ADXL345

Lesson 25 PS2 Joystick

Overview

In this lesson, we will learn the usage of joystick. We program the Raspberry Pi to detect the state of the joystick.

Components

- 1* Raspberry Pi
- 1* ADC0832
- 1* PS2 Joystick
- 1* Breadboard
- Several jumper wires

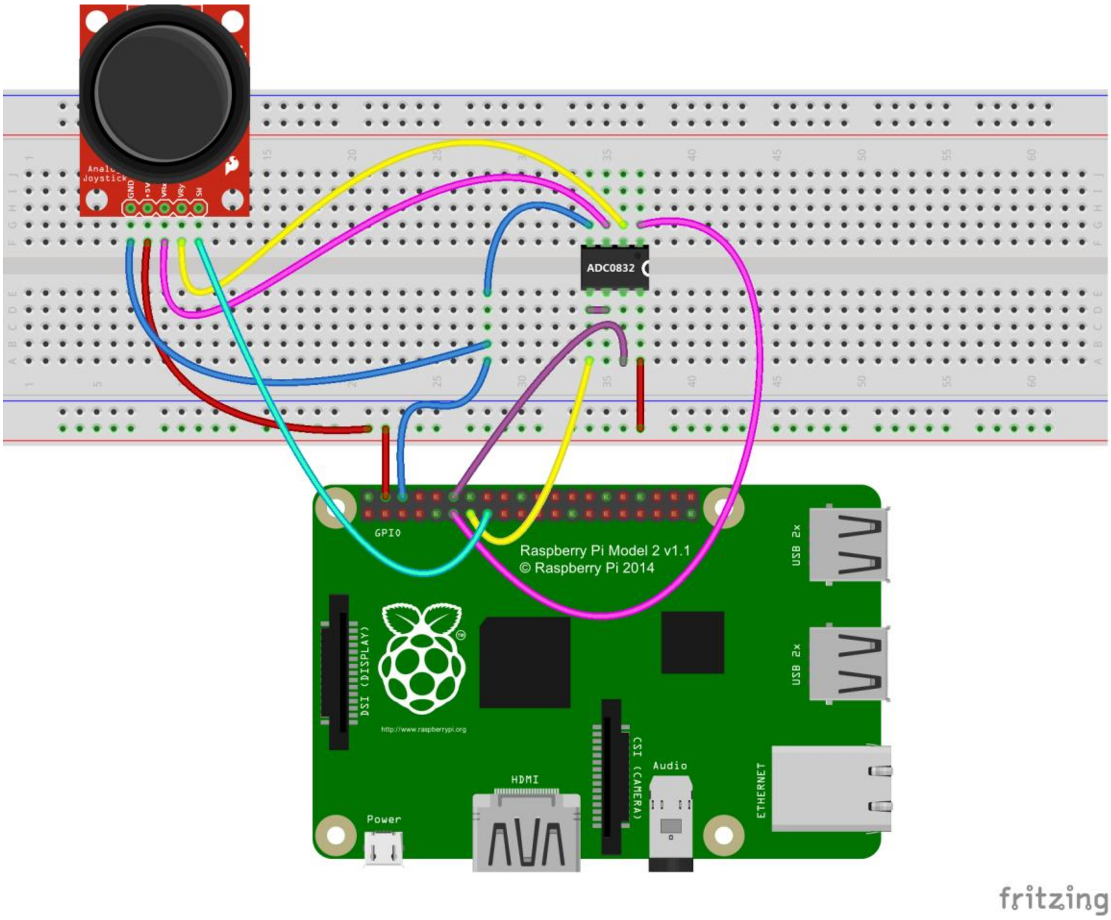
Principle

A joystick is an input device consisting of a stick that pivots on a base and reports its angle or direction to the device it is controlling. A joystick, also known as the control column, is the principal control device in the cockpit of many civilian and military aircraft, either as a center stick or side-stick. It often has supplementary switches to control various aspects of the aircraft's flight.

Joysticks are often used to control video games, and usually have one or more push-buttons whose state can also be read by the computer. A popular variation of the joystick used on modern video game consoles is the analog stick. Joysticks are also used for controlling machines such as cranes, trucks, underwater unmanned vehicles, wheelchairs, surveillance cameras, and zero turning radius lawn mowers. Miniature finger-operated joysticks have been adopted as input devices for smaller electronic equipment such as mobile phones.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/25_ps2Joystick/joystick.c)

Step 3: Compile

$ gcc joystick.c -o joystick -lwiringPi

Step 4: Run

$ sudo ./joystick

For Python users:

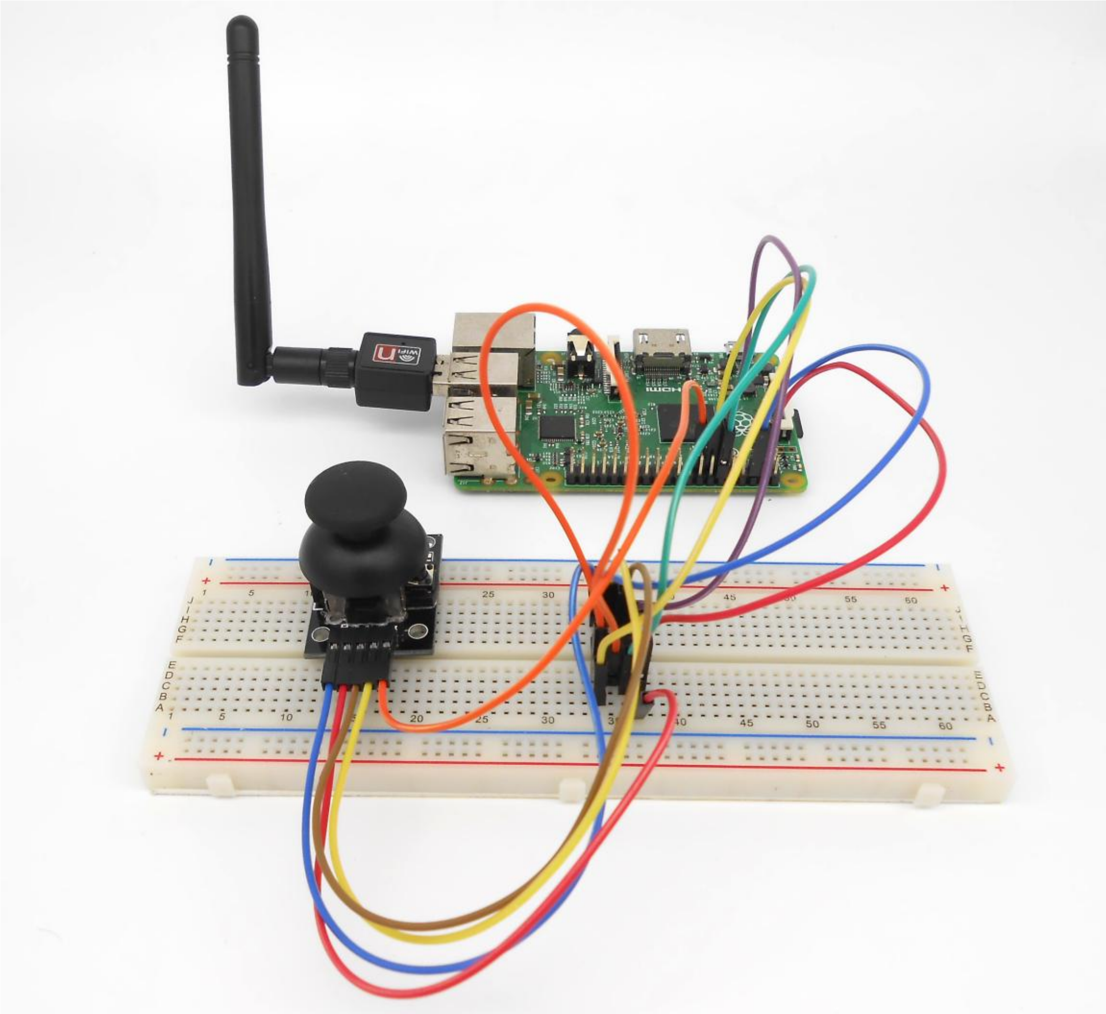
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/25_joystick.py)

Step 3: Run

$ sudo python 25_joystick.py

Now you should see the joystick state information displayed on the terminal.



Lesson 26 A Simple Access Control System

Overview

In this lesson, we will learn how to make a simple access control system based on the Raspberry Pi and the RC522-based RFID module.

Components

- 1* Raspberry Pi
- 1* RFID module
- 1* RFID ID Card
- 1* Special-shaped RFID ID Card
- 1* Active buzzer
- 1* LED
- 1* 220Ω Resistor
- 1* 1KΩ Resistor
- 1* NPN Transistor (S8050)
- 1* Breadboard
- Several jumper wires

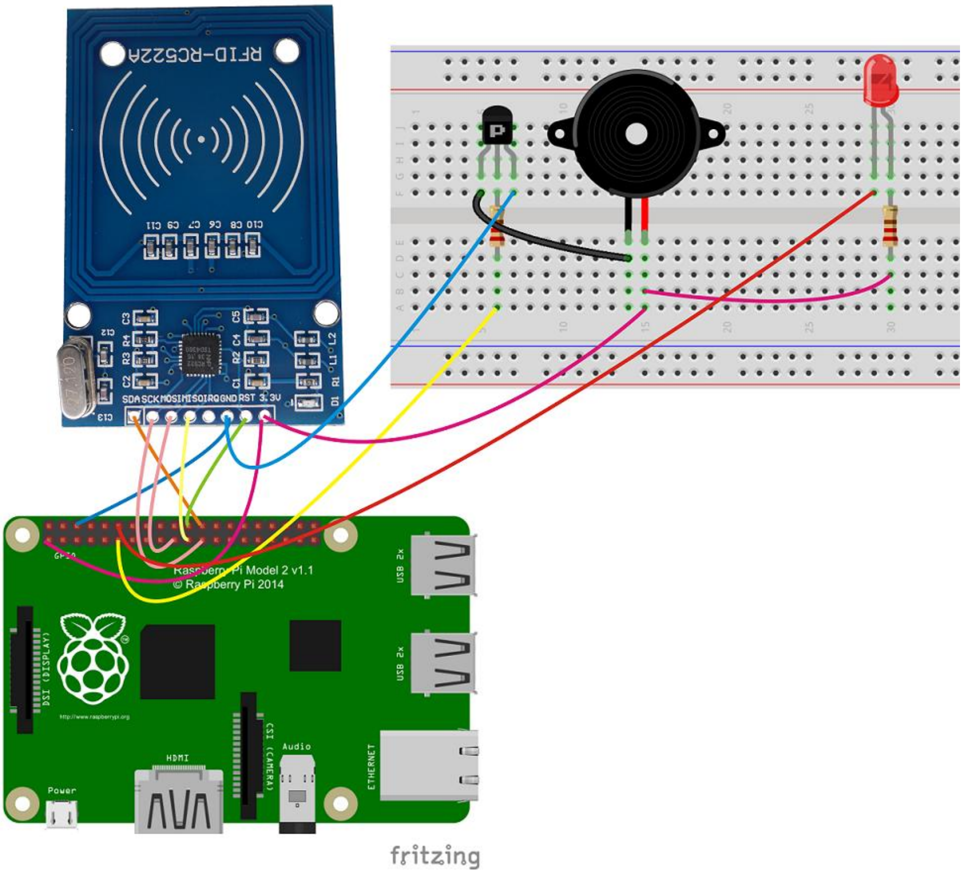
Principle

It is a comprehensive experiment which contains many devices. For more information about RFID and RC522, please refer to lesson 19 in this kit.

In this experiment, we program the Raspberry Pi to read the RFID ID card through the RC522 RFID module. If you get the same ID number as previously input, the LED will light up. In addition, when the RFID ID card approaches the reader, the buzzer will make sounds.

Procedures

Step 1: Build the circuit



For C language users:

Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_C_Code_for_RPi/26_AccessCtrlSystem)

Step 3: Compile

    $ sudo ./compile.sh

Step 4: Run

    $ sudo ./test

For Python users:

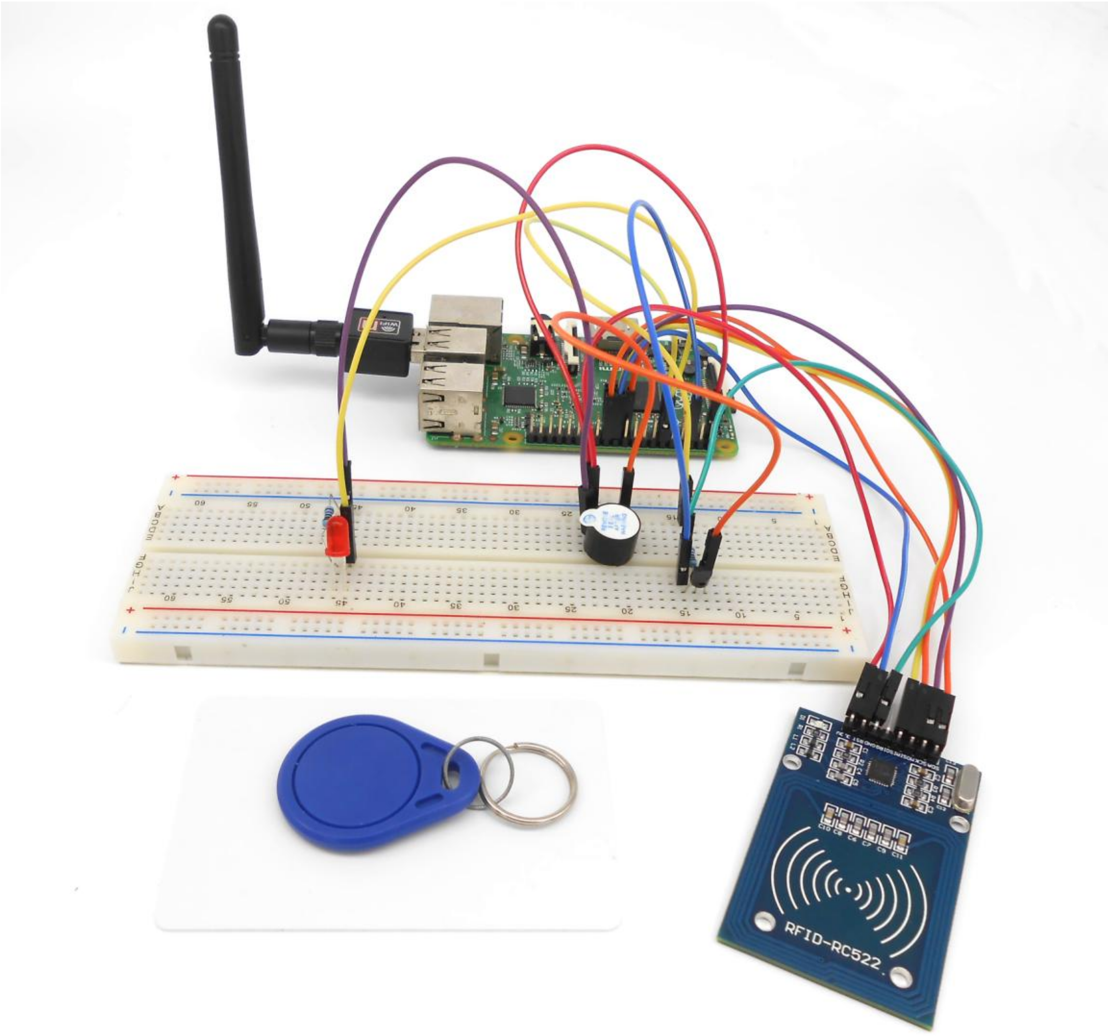
Step 2: Edit and save the code with vim or nano.

(Code path: /home/Adeept_RFID_Learning_Kit_Python_Code_for_RPi/26_AccessCtrlSystem/MFRC522-python/Read_test.py)

Step 3: Run

    $ sudo python Read_test.py

Now put the RFID ID card close to the reader and the buzzer will beep. At the same time, the LED lights up.


RFID-RC522


Adeept
Sharing Perfects Innovation
E-mail: support@adeept.com
website: www.adeept.com
2017.08