Introduce to Infrared Remote Control

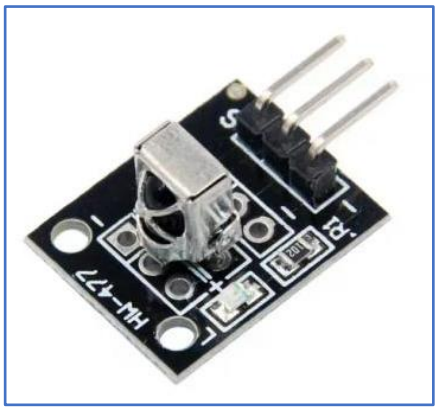
We have introduced infrared receiving tube in former passage, it is a kind of sensor which can identify the infrared. The integration of infrared sensor receives and modulates the 38kHZ infrared. In order to avoid in the process of wireless transmission from other infrared signal interference, the infrared remote control usually modulate the signal on a specific carrier frequency, and then launch out by the infrared emission diode. While the infrared receiving device needs to filter out other clutter, receives the specific frequency signal and restores it into binary pulse code, namely, demodulation.

Working Principle

The built-in receiving tube converts the light signal emitted by infrared transmitting tube to weak electrical signal, the signal is amplified through the internal IC, then restored to original code emitted by the infrared remote control through the automatic gain control, band-pass filtering, demodulation, waveform shaping, and input to the decoding circuit on electric appliance through the output pin of the receiving header

Infrared receiver header has three pins: connecting VOUT to analog interface, GND to the GND onboard and VCC to + 5 v.

Infrared receiver



Infrared receiver module physical map

Experiment Purpose

The aim is to ecode all the keys of the remote control by Ardunio and display them in a serial port. Due to the decoding process is based on the given protocol, and the code is massive, so we use open source libraries to decode and copy IRremote to arduino1.6.5-r5\libraries.

The link: https://github.com/shirriff/Arduino-IRremote to arduino1.6.5-r5\libraries

Experiment Principle

The encoding of a remote control is required to be learned first if we want to decode it. The control code used in this product is: NEC protocol.

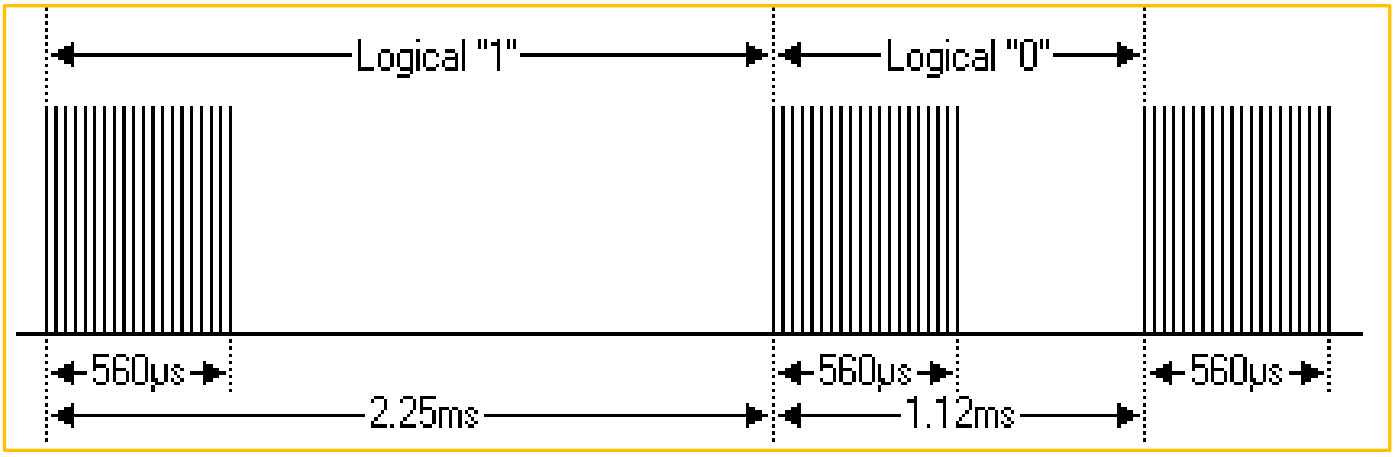
The following will introduce the NEC protocol:

Introduction of NEC

Characteristics

- 8 address bits, 8 command bits
- Address bits and command bits are transmitted twice in order to ensure reliability
- Pulse-position modulation
- Carrier frequency 38kHz
- Every bit lasts 1.125ms or 2.25ms

The definitions of logic 0 and 1 are as below

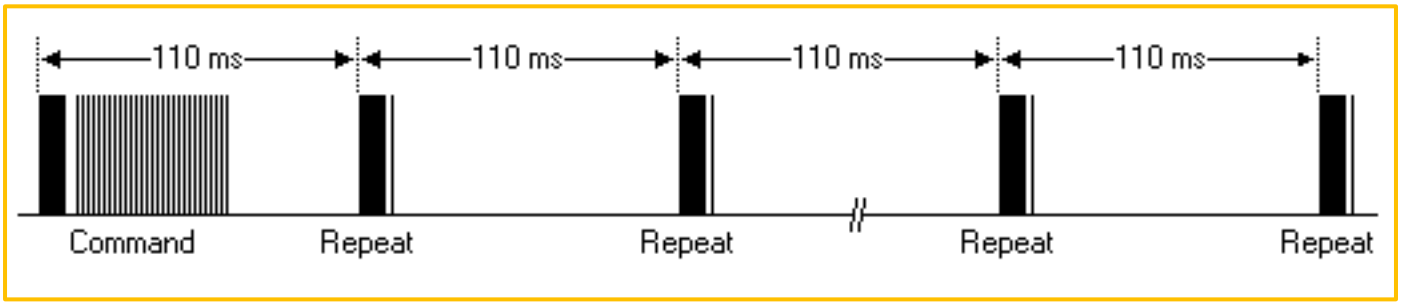


Transmitted pulse which the pressed button released immediately

The picture above shows a typical pulse sequence of the NEC protocol. Notice: The protocol of LSB (least significant) is firstly transmitted. In the above, pulse transmission

address is 0x16 and the command is 0x59. A message starts from a 9ms high level, then followed by a 4.5ms low level, and by the address code and command code. The address and command are transmitted twice. All bits flip in the second transmission, this can be used in the confirmation of the received message. The total transmission time is constant, because every bit repeats the flip length. If you are not interested, you can ignore this reliable inversion and expand the address and command every 16 bit as well !

Transmitted pulse which the pressed button last for a while



Even a button on the remote control is pressed again, the command is transmitted only once. When the button has been pressing, the first 110ms pulse is same as above, then the same code is transmitted every 110ms. The next repeated code consists of a 9ms high level pulse, a 2.25ms low level pulse and a 560μs high level pulse.

Notice: When the pulse received by the integrative header, the header needs to decode,amplify and shape the signal. So we should notice that the output is high level when the infrared signal is absent, otherwise, the output is low level, so the output signal level is reversed in transmitter. We can see the pulse of receiver through the oscilloscope and understand the program by waveform.

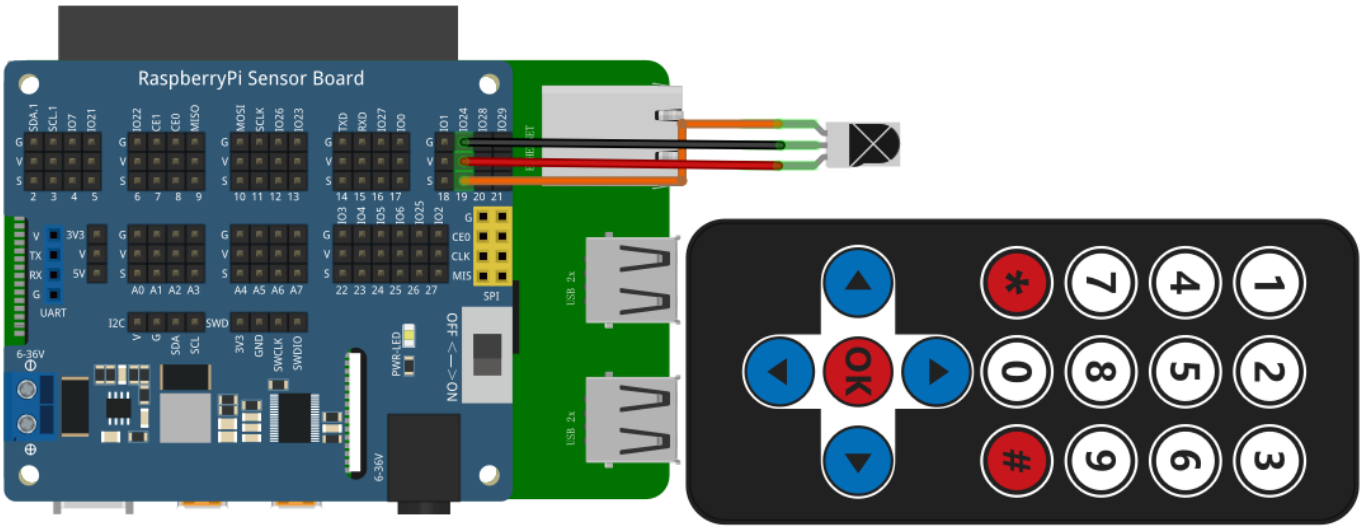
Component List

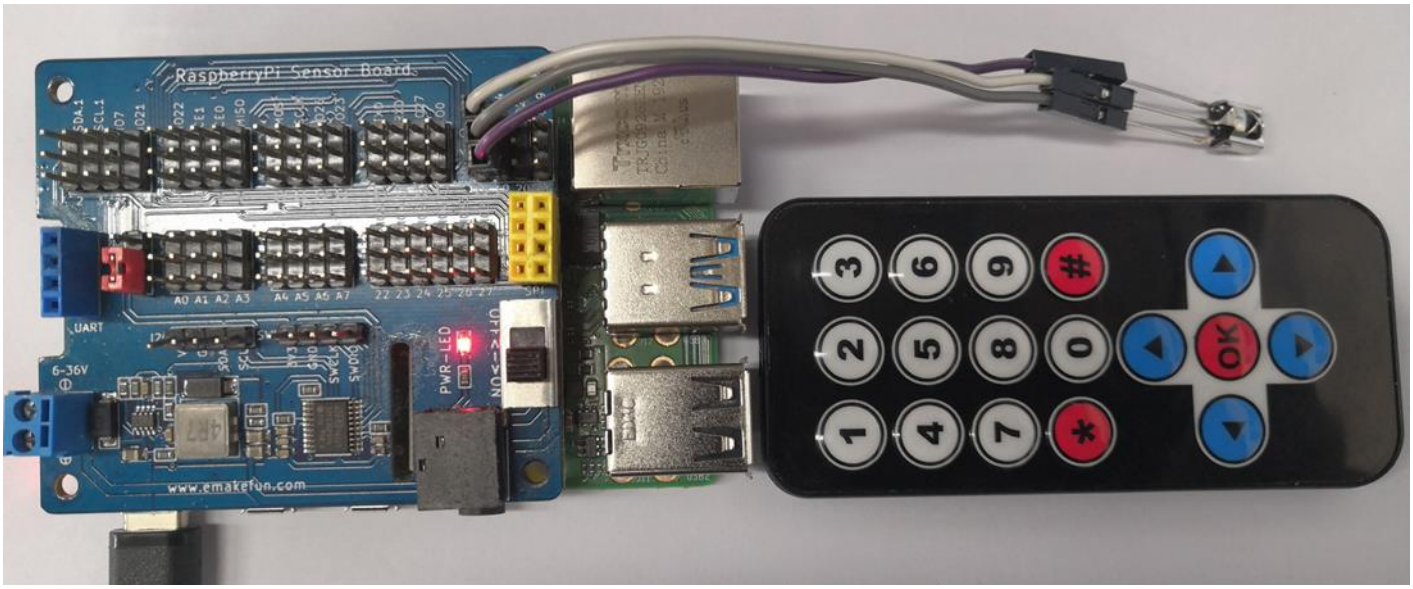
- RaspberryPi mainboard
- Breadboard
- Data line
- Infrared remote control *1
- Integrated infrared receiver * 1
- Modular infrared receiver * 1
- Several jumpers

Wiring of Circuit

Infrared receiver module connection:

Infrared receiver module	RaspberryPi
G	GND
R	5V
Y	IO24(wiringPi)/19(BCM)





CODE

C++ main program

```
#include "IR_REC.h"

int main()
{
```

```
    int key;
    if(wiringPiSetup() == -1){
        printf("setup wiringPi failed !");
        return 1;
    }
    while(1){
        if (GetKey() != ERROR) {
            printf("Change_Map %s \n",Change_Map().c_str());
            //cout << VAL.keyname << endl;
        }
    }
}
```

Python program

```
#!/usr/bin/python
# -*- coding:utf-8 -*-
import RPi.GPIO as GPIO
import time
ERROR = 0xFE
PIN = 19
    # Define the infrared receiver pin
keymap = {
   0x45:"1",
   0x46:"2",
   0x47:"3",
   0x44:"4",
   0x40:"5",
   0x43:"6",
   0x07:"7",
   0x15:"8",
   0x09:"9",
   0x19:"0",
   0x16:"*",
   0x0D:"#",
   0x18:"up",
   0x52:"down",
   0x1C:"ok",
   0x08:"left",
   0x5A:"right",
   0xfe:0xfe
```

```
}

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(PIN, GPIO.IN, GPIO.PUD_UP)    # Set the infrared receiving pin to pull-up mode

def getKey():
    byte = [0, 0, 0, 0]
    if IRStart() == False:      # Judge the infrared guidance pulse
        time.sleep(0.11)        # One message frame lasts 108 ms.
        return ERROR
    else:
        for i in range(0, 4):
                byte[i] = getByte()  # Receive 32-bit infrared data (address, address
inversion, data, data inversion)
        if byte[0] + byte[1] == 0xff and byte[2] + byte[3] == 0xff:
#print("right")
            return byte[2]
        else:
        #print("error")
            return ERROR
    #return byte[2]
def IRStart():     # Judge the infrared guidance pulse
    timeFallingEdge = [0, 0]
    timeRisingEdge = 0
    timeSpan = [0, 0]
    # Read pulse time by pulse rising and falling edge
    GPIO.wait_for_edge(PIN, GPIO.FALLING)
    timeFallingEdge[0] = time.time()
    GPIO.wait_for_edge(PIN, GPIO.RISING)
    timeRisingEdge = time.time()
    GPIO.wait_for_edge(PIN, GPIO.FALLING)
    timeFallingEdge[1] = time.time()

    timeSpan[0] = timeRisingEdge - timeFallingEdge[0]  # First pulse time
    timeSpan[1] = timeFallingEdge[1] - timeRisingEdge  # Second pulse time
    #print(timeSpan[0],timeSpan[1])
    if timeSpan[0] > 0.0085 and timeSpan[0] < 0.0095 and timeSpan[1] > 0.004 and timeSpan[1]
< 0.005:
    #print("1")
        return True
```

```
    else:
    #print("0")
        return False
def getByte():   # Receive 32-bit infrared data (address, address inversion, data, data
inversion)
    byte = 0
    timeRisingEdge = 0
    timeFallingEdge = 0
    timeSpan = 0
    for i in range(0, 8):
        # Read pulse time by pulse rising and falling edge
        GPIO.wait_for_edge(PIN, GPIO.RISING)
        timeRisingEdge = time.time()
        GPIO.wait_for_edge(PIN, GPIO.FALLING)
        timeFallingEdge = time.time()

        timeSpan = timeFallingEdge - timeRisingEdge   # Read pulse time
        if timeSpan > 0.0016 and timeSpan < 0.0018:   # Determine whether the pulse is
representative 1
            byte |= 1 << i
    return byte

def change_map(key_val):
    for index in keymap.keys():
        if index == key_val :
            return keymap[index]

print('IRM Test Start ...')
try:
    while True:
        #key = getKey()     # Read infrared pulse
        key = change_map(getKey())
        if(key != ERROR):  # Print infrared pulse value
            print("Get the key:" + key)
except KeyboardInterrupt:
    GPIO.cleanup()
```

Java program

```
import com.pi4j.wiringpi.Gpio;
import java.util.HashMap;
```

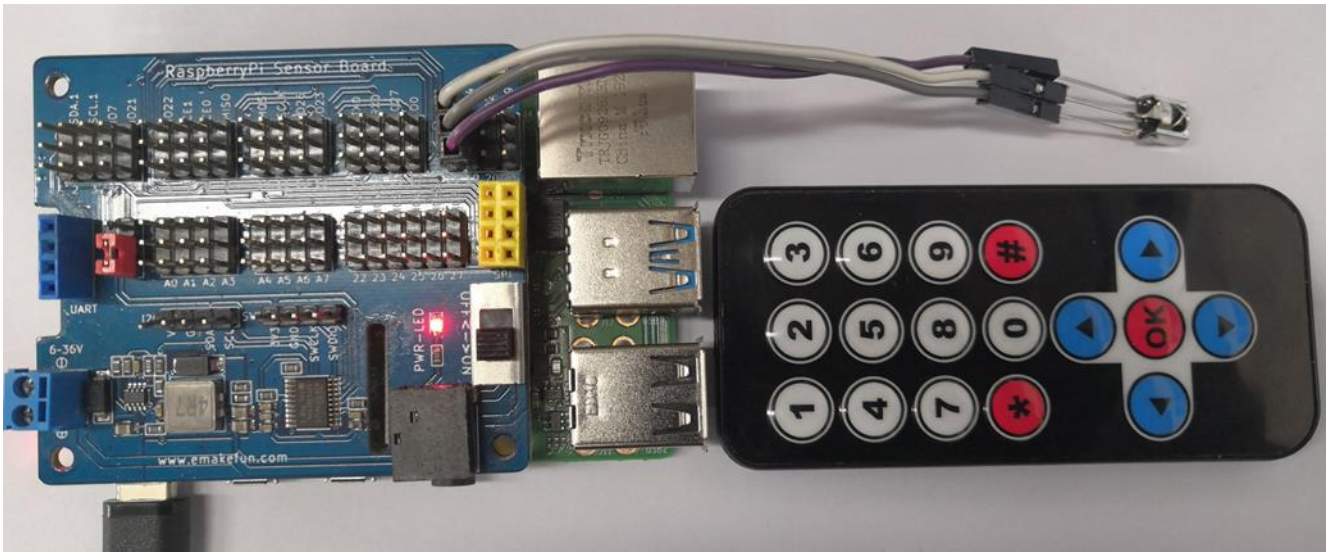
```
import java.util.Iterator;
import java.util.Map;
import java.util.Map.Entry;
import java.util.Set;


public class IR_NEC {
	static int PIN = 24;
	static int ERROR = 0xfe, key;
	static long timeRisingEdge, timeFallingEdge, timeRising, timeFalling_0,
timeFalling_1;
	static long timeSpan_val = 0;
	static long[] time_span = new long[2];
	static Map<Integer, String> map = new HashMap<>();


    static {
	    if (Gpio.wiringPiSetup() == -1) {
	        System.out.println(" ==>> GPIO SETUP FAILED");
	    }

	    // store data in map collection
	     map.put(0x45, "1" );
	    map.put(0x46, "2" );
	    map.put(0x47, "3" );
	    map.put(0x44, "4" );
	    map.put(0x40, "5" );
	    map.put(0x43, "6" );
	    map.put(0x07, "7" );
	    map.put(0x15, "8" );
	    map.put(0x09, "9" );
	    map.put(0x19, "0" );
	    map.put(0x16, "*" );
	    map.put(0x0D, "#" );
	    map.put(0x18, "up");
	    map.put(0x52, "down");
	    map.put(0x1C, "ok");
	    map.put(0x08, "left");
	    map.put(0x5A, "right");
	    map.put(0xfe, "error");
```

```
        Gpio.pinMode(PIN, Gpio.INPUT);
    }

    public static boolean  IRStart() {
        while(!(Gpio.digitalRead(PIN) == 0));
        timeFalling_0 = gettimeofday();
        while(!(Gpio.digitalRead(PIN) == 1));
        timeRising = gettimeofday();
        while(!(Gpio.digitalRead(PIN) == 0));
        timeFalling_1 = gettimeofday();
        time_span[0] = timeRising - timeFalling_0;
        time_span[1] = timeFalling_1 - timeRising;

//      System.out.println("start_time " + time_span[0] + "," +time_span[1]);

        if (time_span[0] > 8500 && time_span[0] < 9500 && time_span[1] >= 4000 &&
time_span[1] <= 5000)
        {
//          System.out.println("start singe*************");
           return true;
        }
        else   {
            return false;
        }
    }

    public static long gettimeofday() {
//      return System.currentTimeMillis() ;// +System.nanoTime() / 1000;
        return System.nanoTime() / 1000;

    }

    public static int GetByte() {
        int byte_val = 0;
        for (int i = 0; i < 8; i++) {
            while(!(Gpio.digitalRead(PIN) == 1));
            timeRisingEdge = gettimeofday();
            while(!(Gpio.digitalRead(PIN) == 0));
            timeFallingEdge = gettimeofday();
            timeSpan_val = timeFallingEdge - timeRisingEdge;
//          System.out.print("start byte  ");
```

```
//          System.out.println(timeSpan_val);

            if (timeSpan_val > 1500 && timeSpan_val < 1800)
                byte_val |= 1 << i;

        }
//  System.out.printf("byte_val: %x  \n", byte_val);

        return byte_val;
    }

    public static int GetKey() {
        int[] byte_val = new int[4];
        if (IRStart() == false) {
            Gpio.delay(108);
            return ERROR;
        } else {
            for (int i = 0; i < 4; i++) {
                byte_val[i] = GetByte();
//              System.out.printf("byte_val[%d]: %x \n",i, byte_val[i]);

            }
            if ((byte_val[0] + byte_val[1] == 0xff) && (byte_val[2] + byte_val[3] == 0xff))
{
                return byte_val[2];
            } else {
                    return ERROR;
              }
          }
     }

    public static String change_map(int data) {
        Set<Integer> keys = map.keySet();
        for(Integer key:keys){
            //System.out.println("key值: "+key+" value值: "+map.get(key));
            if(data == key)
                return map.get(key);
        }
        return "ERROR";
    }
```

```
    public static void main(String args[]) {
        String rec_val;
        IR_NEC ir_nec = new IR_NEC();
        for ( ; ;) {
            rec_val = ir_nec.change_map(ir_nec.GetKey());
            if (rec_val != "error") {
                System.out.println("key: " + rec_val);
            }
        }
    }
}
```

Experiment Result




```
pi@raspberrypi:~/Desktop/test_send/红外遥控实验/Python $ python IR_REC.py
IRM Test Start ...
Get the key:up
Get the key:3
Get the key:5
Get the key:6
Get the key:7
Get the key:8
Get the key:9
Get the key:0
Get the key:ok
Get the key:0
Get the key:left
```

Encode all the buttons of the remote control through RaspberryPi, press the button you want on the infrared remote control, and the button you pressed will be displayed on the serial monitor. The infrared remote control requires not to be far away. Quasi-receiving head。