Infrared

|  |  |
| --- | --- |
| Emitter | Receiver |

Overview  
 Infrared is light—electronic magnetic radiation—just beneath the limits of human vision. While we cannot see it, we can often feel it as heat, and many animals can see it or detect it through other sense organs. Because infrared light is invisible, it offers an excellent means for transmitting information wirelessly. An *infrared emitter* can send a pulse of infrared light; controlling logic can use the timing of such pulses as some sort of code to represent and “send” a secret message. Elsewhere, an *infrared receiver* is a device that translates the optical infrared light into an electric signal, and controlling logic on the receiving end can decode the timing of these received pulses into a reconstructed copy of the emitter’s transmitted message. Televisions, air conditions, toys and numerous other devices use this sort of “remote control” infrared transmission for short-range, inexpensive, low-power wireless communication between devices.

In this experiment, which is exactly the same for the emitter and receiver, you will use both an Infrared Emitter and the Infrared Receiver to set up a communication loop. If you had two Raspberry Pi devices, you could communicate from one device wirelessly to the other, but since you are more likely at this point to have only one, you will attach both sensors to the same Raspberry Pi and transmit “wirelessly” (by infrared) from one pin to another. Of course, the pins also happen to be wired together, so this is only a demonstration of the possibility of wireless communication rather than a compelling application of it!

Some cell phone cameras are also capable of “seeing” infrared light and making it visible in their captured images. You might try watching or filming the experiment through your cell phone camera to see if you can “see” the infrared being transmitted.

Experimental Materials

Raspberry Pi x1

Breadboard x1

Infrared Emitter x1

Infrared Receiver x1

LED (3 pin) x1

Resistor (330Ω) x1

Dupont jumper wires

Experimental Procedure

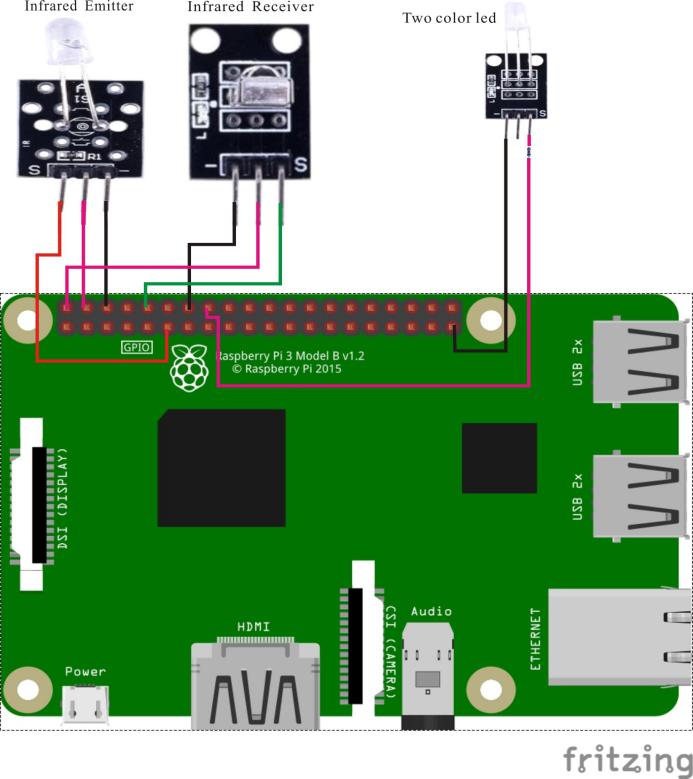
1. If you have not done so already, prepare your development system by installing the Python interpreter, RPi.GIO library, and wiringPi library as described in READ\_ME.TXT.
2. Install the infrared emitter and infrared receiver in your breadboard, and use the resistor and Dupont jumper wires as illustrated in the Wiring Diagram below. Note you will connect only two of the three pins on the LED.
3. Execute the sample stored in this experiment’s subfolder.

If using C, compile and execute the C code:  
cd Code/C  
gcc infrared.c -o infrared.out –lwiringPi  
./infrared.out

If using Python, launch the Python script:  
cd Code/Python  
python infrared.py

1. Make experimental observations. The sample code first instructs the Raspberry Pi to call a special routine—an interrupt handling callback—every time the receiver sees infrared light. This routine prints a statement to text output saying light was detected, and blinks an LED so you can see the activity. (Remember you cannot see infrared itself!) After preparing this routine, the code enters an endless loop, using the emitter to slowly blink the light, reporting to text output when it does so. When you run the code, the light emitted in the endless loop will be detected by the receiver, which will suspend the loop momentarily to invoke the interrupt handler. The resulting text output reports, in alternating sequence, the infrared light being emitted by the emitter (in the loop), and then immediately detected (wirelessly) by the receiver (in the interrupt service request callback).

Wiring Diagram



Infrared Receiver pin position:

"S" ↔ Raspberry Pi pin 10

"+" ↔ Raspberry Pi +5V

"-" ↔ Raspberry Pi GND

Infrared Emitter pin position:

"S" ↔ Raspberry Pi 11

"+" ↔ Raspberry Pi +5V

"-" ↔ Raspberry Pi GND

LED pin position:

"S" ↔ Raspberry Pi 16(through resistor)

"-" ↔ Raspberry Pi GND

Sample Code

Python code

#!/usr/bin/env python

import RPi.GPIO as GPIO

import time

IrReceiverPin = 10

IrEmissionPin = 11

LedPin = 16

Led\_status = 1

def setup():

GPIO.setmode(GPIO.BOARD)       # Numbers GPIOs by physical location

GPIO.setup(LedPin, GPIO.OUT)   # Set LedPin's mode is output

GPIO.setup(IrReceiverPin, GPIO.IN, pull\_up\_down=GPIO.PUD\_UP)

GPIO.setup(IrEmissionPin, GPIO.OUT)   # Set IrPin's mode is output

GPIO.output(LedPin, GPIO.LOW) # Set LedPin low to off led

def irReceivedCallback(ev=None): # Called every time the irReceiver pin falls. Blink the LED!

global Led\_status

Led\_status = not Led\_status

GPIO.output(LedPin, Led\_status)  # switch led status(on-->off; off-->on)

print("received signal!\n")

time.sleep(0.1)

GPIO.output(LedPin, False)

def loop():

GPIO.add\_event\_detect(IrReceiverPin, GPIO.FALLING, callback=irReceivedCallback) # wait for falling

while True:

print '...IrPin high'

GPIO.output(IrEmissionPin, GPIO.HIGH)  # IrPin on

time.sleep(0.5)

print 'IrPin low...'

GPIO.output(IrEmissionPin, GPIO.LOW) # IrPin off

time.sleep(0.5)

def destroy():

GPIO.output(LedPin, GPIO.LOW)     # 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.

                print("KeyboardInterrupt.\n")

destroy()

C code

#include <wiringPi.h>

#include <stdio.h>

#define    IrEmissionPin    0

#define   IrReceiverPin 16

#define   LEDPin 4

int cnt = 0;

void infraredDetectionInterruptHandler(void)

// called at interrupt every time the IR receiver pin's signal falls to LOW

{

printf("Received infrared signal. cnt = %d\n", ++cnt);

    // blink the LED briefly

if(digitalRead(LEDPin) == HIGH)

{

digitalWrite(LEDPin, LOW);

}

    else

{

digitalWrite(LEDPin, HIGH);

}

delay(100);

digitalWrite(LEDPin, LOW);

}

int main(void)

{

if(wiringPiSetup() == -1)

{

printf("setup wiringPi failed !\n");

return -1;

}

pinMode(LEDPin, OUTPUT);

pinMode(IrEmissionPin, OUTPUT);

pinMode(IrReceiverPin, INPUT);

pullUpDnControl(IrReceiverPin, PUD\_UP)

    // register an interrupt service routine that will

    // be called every time the IR receiver pin goes LOW

if(wiringPiISR(IrReceiverPin, INT\_EDGE\_FALLING, &infraredDetectionInterruptHandler) == -1)

{

printf("setup ISR failed !");

return -1;

}

    // now loop forever, from time to time blinking the IR

    // emitter ON and OFF.  If the receiver receives the emitter's emission,

    // then the interrupt routine should report to us "I saw it!" shortly after

    // this emitter logic tells us "I sent it!"

while(1)

{

digitalWrite(IrEmissionPin, HIGH);

printf("IrEmissionPin is set High\n");

delay(500);

digitalWrite(IrEmissionPin, LOW);

printf("IrEmissionPin is set Low\n");

delay(500);

}

return 0;

}

Technical Background

◆Strong anti-interference: epoxy resin package plus shielding anti-jamming design  
◆ Wide operating voltage: 2.7-5.5V  
◆Low power consumption: Wide angle; Long distance reception  
◆ Output Match TTL: CMOS Level