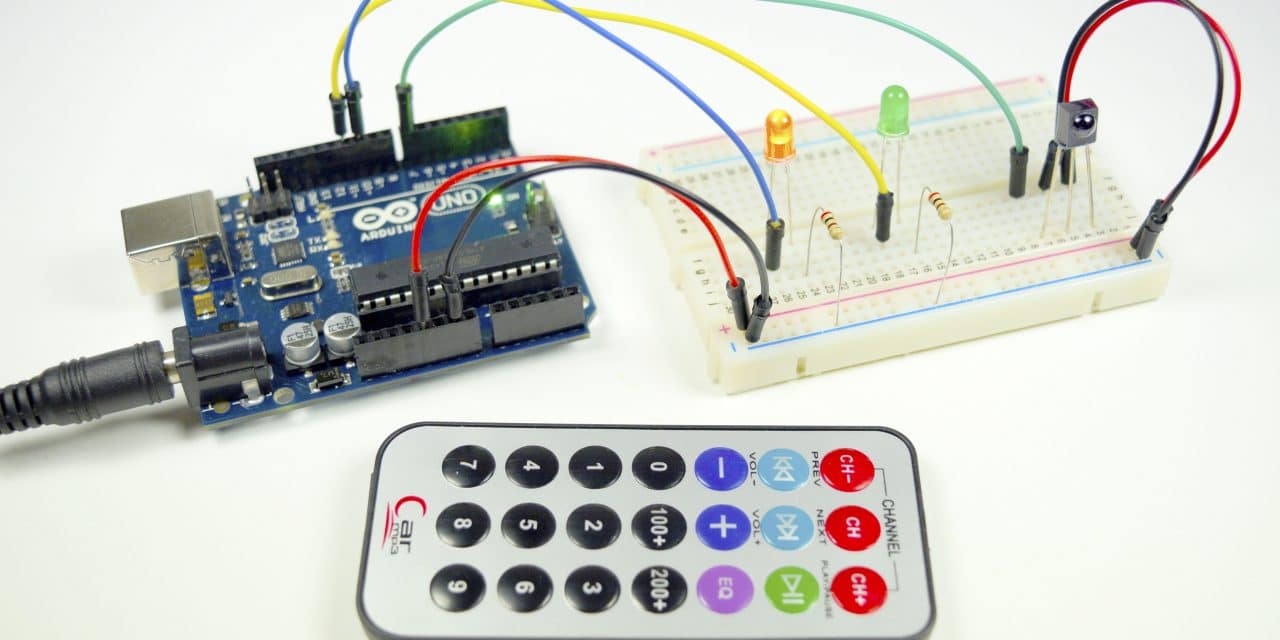
IR Remote and Receiver on an Arduino



**NOTE: To save battery life, each time you use the Remote Infrared module, re-insert the "hard polymer" plate to prevent contact between the battery and the circuit.**

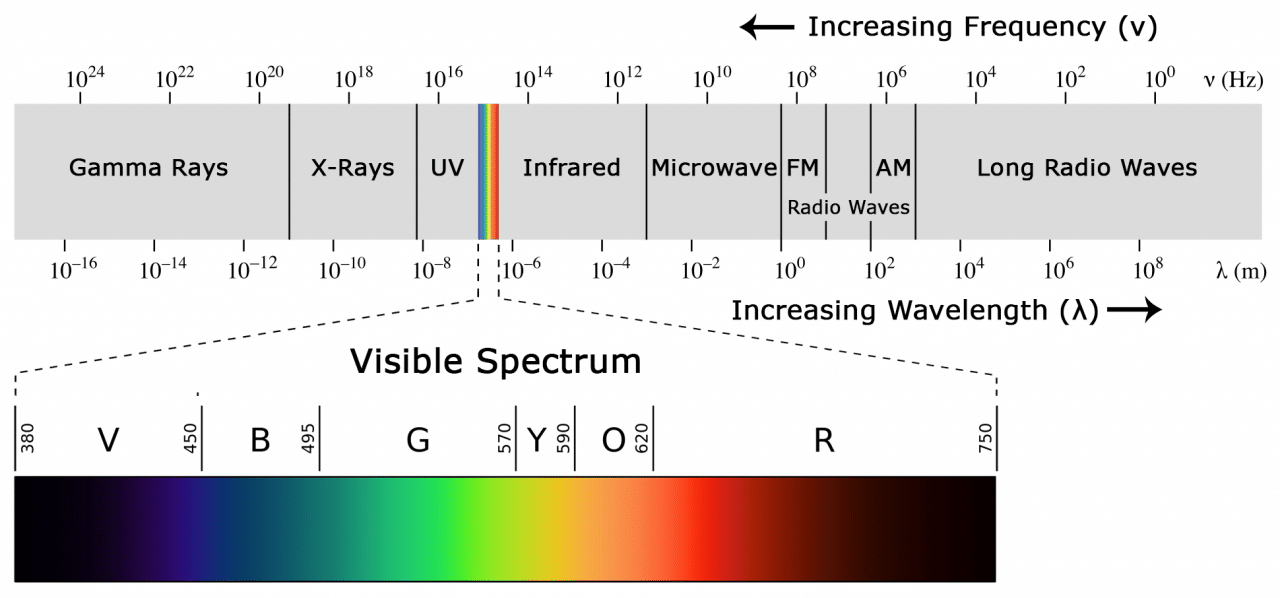
Infrared (IR) communication is a widely used and easy to implement wireless technology that has many useful applications. The most prominent examples in day to day life are TV/video remote controls, motion sensors, and infrared thermometers.

There are plenty of interesting Arduino projects that use IR communication too. With a simple IR transmitter and receiver, you can make remote controlled robots, distance sensors, heart rate monitors, DSLR camera remote controls, TV remote controls, and lots more.

In this lab we will first explain what infrared is and how it works. Then we will show you how to set up an [IR receiver and remote](https://www.amazon.com/gp/product/B00RBHP3J8/ref=as_li_qf_asin_il_tl?ie=UTF8&tag=circbasi-20&creative=9325&linkCode=as2&creativeASIN=B00RBHP3J8&linkId=fe8a42e8938328b3de39d54fdb0dee99) on an [Arduino](https://www.amazon.com/gp/product/B008GRTSV6/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&tag=circbasi-20&camp=1789&creative=9325&linkCode=as2&creativeASIN=B008GRTSV6&linkId=d729825339a02795d7e175ec69120cbd). I’ll also show you how to use virtually any IR remote (like the one for your TV) to control things connected to the Arduino.

What is Infrared ?

Infrared radiation is a form of light similar to the light we see all around us. The only difference between IR light and visible light is the frequency and wavelength. Infrared radiation lies outside the range of visible light, so humans can’t see it:



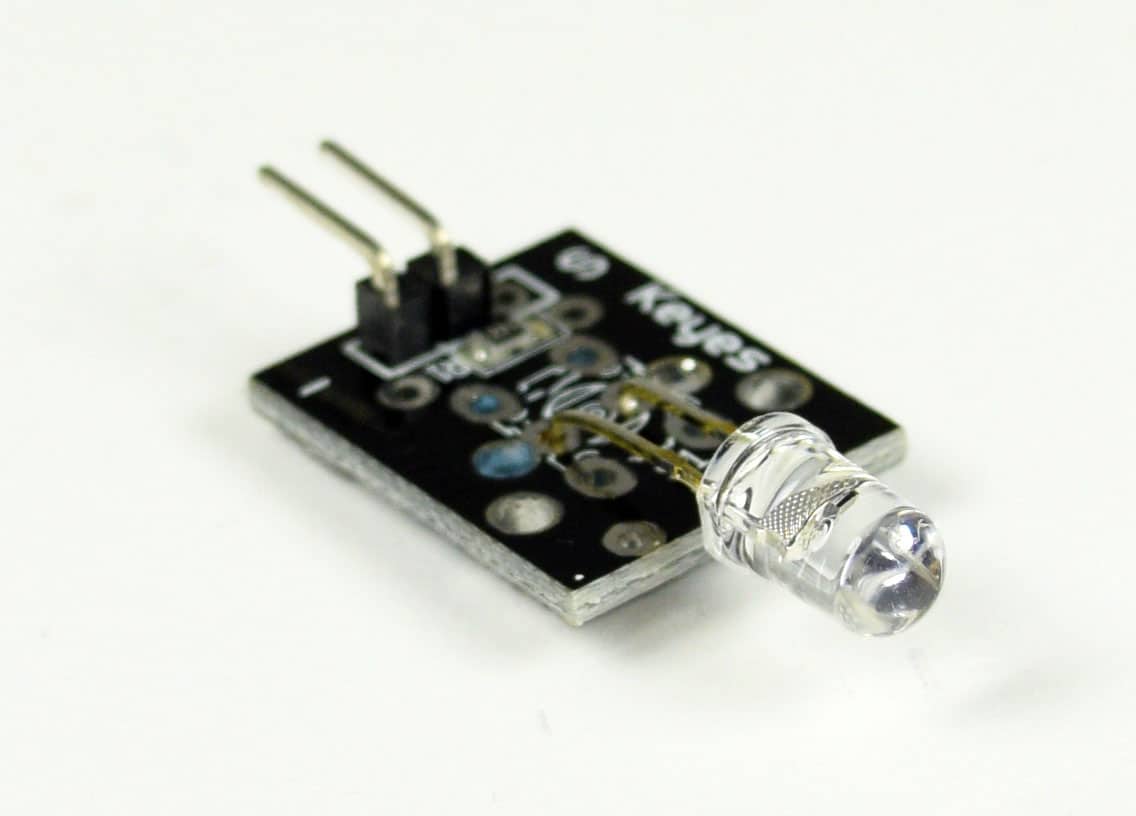
Because IR is a type of light, IR communication requires a direct line of sight from the receiver to the transmitter. It can’t transmit through walls or other materials like WiFi or Bluetooth.

How IR remote and Receivers work

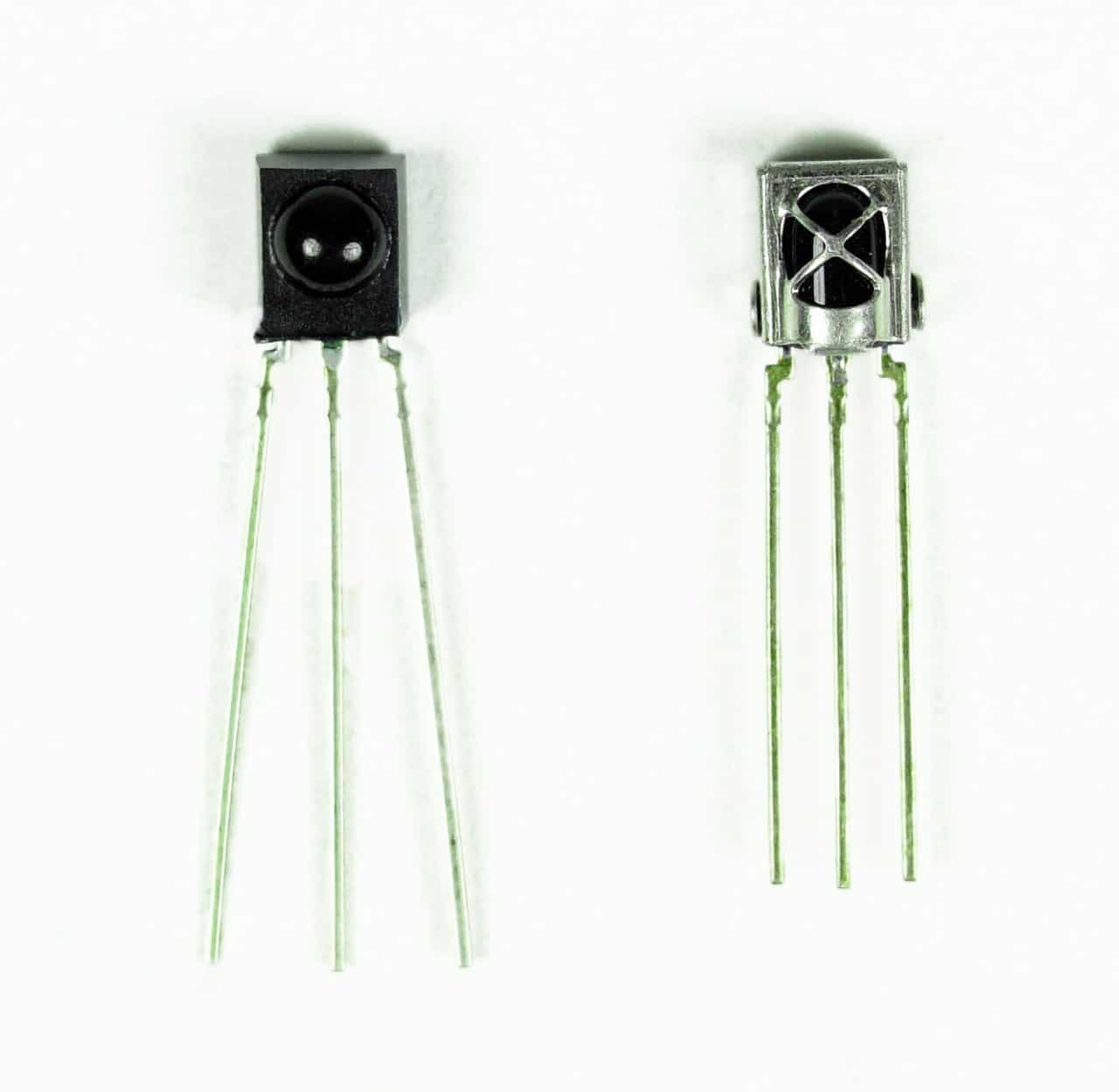
A typical infrared communication system requires an IR transmitter and an IR receiver. The transmitter looks just like a standard LED, except it produces light in the IR spectrum instead of the visible spectrum. If you have a look at the front of a TV remote, you’ll see the IR transmitter LED:



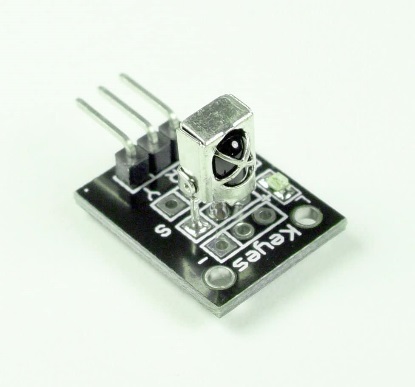
The same type of LED is used in IR transmitter breakout boards for the Arduino. You can see it at the front of this Keyes IR transmitter:



The IR receiver is a [photodiode](https://en.wikipedia.org/wiki/Photodiode) and pre-amplifier that converts the IR light into an electrical signal. [IR receiver diodes](https://www.amazon.com/gp/product/B00X77KXV4/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&tag=circbasi-20&camp=1789&creative=9325&linkCode=as2&creativeASIN=B00X77KXV4&linkId=ecbb73af4aa76fdfa77015c76c87fa3f) typically look like this:



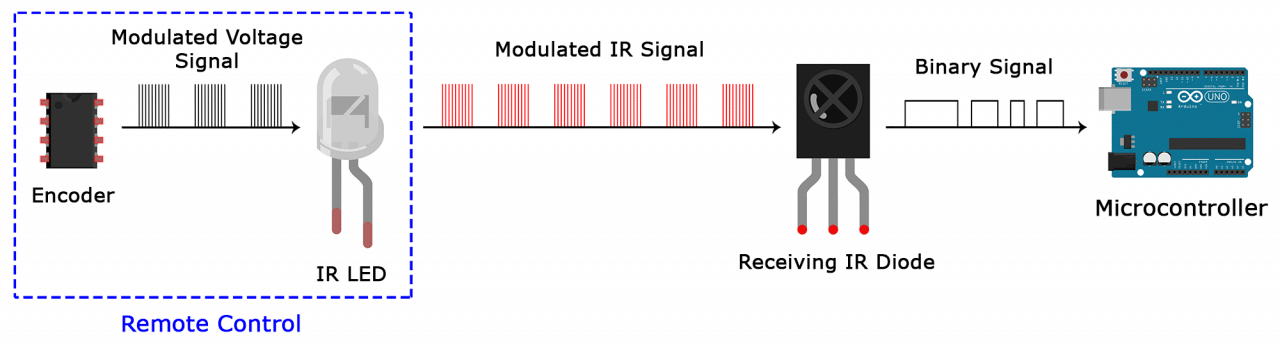
Some may come on a breakout board like this:



IR signal modulation

IR light is emitted by the sun, light bulbs, and anything else that produces heat. That means there is a lot of IR light noise all around us. To prevent this noise from interfering with the IR signal, a signal modulation technique is used.

In IR signal modulation, an encoder on the IR remote converts a binary signal into a modulated electrical signal. This electrical signal is sent to the transmitting LED. The transmitting LED converts the modulated electrical signal into a modulated IR light signal. The IR receiver then demodulates the IR light signal and converts it back to binary before passing on the information to a microcontroller:



The modulated IR signal is a series of IR light pulses switched on and off at a high frequency known as the carrier frequency. The carrier frequency used by most transmitters is 38 kHz, because it is rare in nature and thus can be distinguished from ambient noise. This way the IR receiver will know that the 38 kHz signal was sent from the transmitter and not picked up from the surrounding environment.

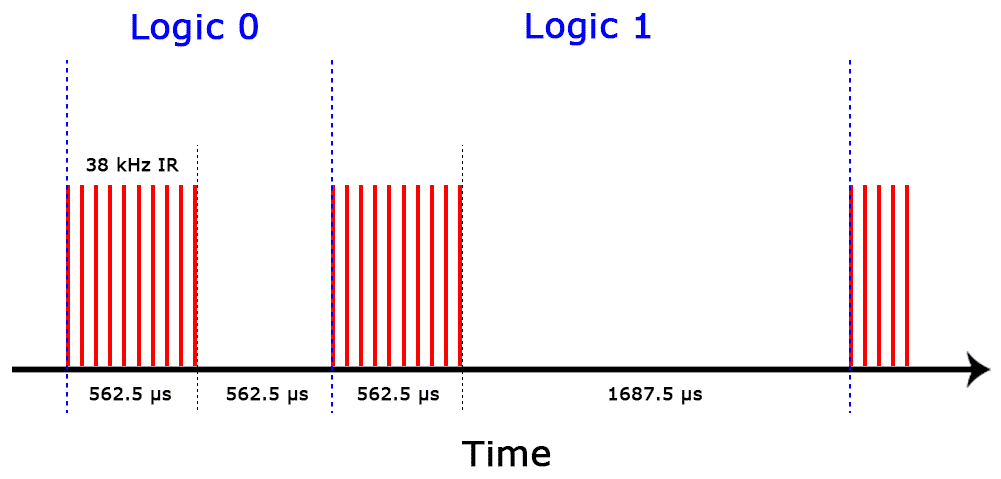
The receiver diode detects all frequencies of IR light, but it has a band-pass filter and only lets through IR at 38 kHz. It then amplifies the modulated signal with a pre-amplifier and converts it to a binary signal before sending it to a microcontroller.

IR transmission protocols

The pattern in which the modulated IR signal is converted to binary is defined by a transmission protocol. There are many IR transmission protocols. Sony, Matsushita, NEC, and RC5 are some of the more common protocols.

The NEC protocol is also the most common type in Arduino projects, so I’ll use it as an example to show you how the receiver converts the modulated IR signal to a binary one.

Logical ‘1’ starts with a 562.5 µs long HIGH pulse of 38 kHz IR followed by a 1,687.5 µs long LOW pulse. Logical ‘0’ is transmitted with a 562.5 µs long HIGH pulse followed by a 562.5 µs long LOW pulse:



This is how the NEC protocol encodes and decodes the binary data into a modulated signal. Other protocols differ only in the duration of the individual HIGH and LOW pulses.

IR codes

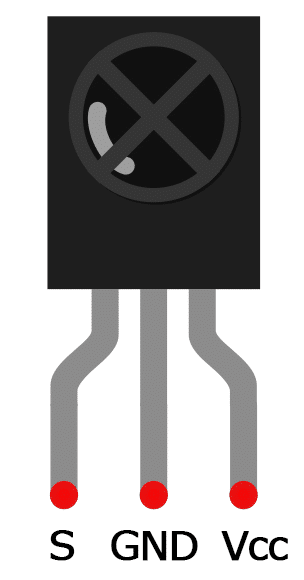
Each time you press a button on the remote control, a unique hexadecimal code is generated. This is the information that is modulated and sent over IR to the receiver. In order to decipher which key is pressed, the receiving microcontroller needs to know which code corresponds to each key on the remote.

Different remotes send different codes for the keypresses, so you’ll need to determine the code generated for each key on your particular remote. If you can find the datasheet, the IR key codes should be listed. If not though, there is a simple Arduino sketch that will read most of the popular remote controls and print the hexadecimal codes to the serial monitor when you press a key.

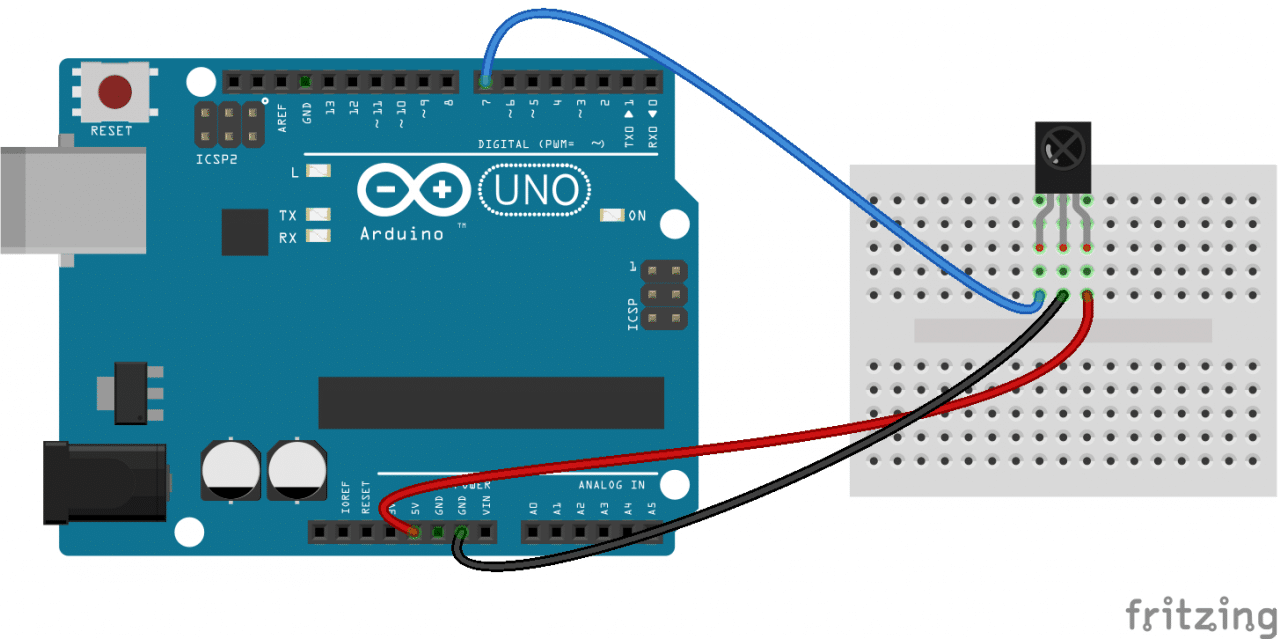
How to connect an IR receiver to the Arduino

There are several different types of IR receivers, some are stand-alone, and some are mounted on a breakout board. All IR receivers will have three pins: signal, ground, and Vcc.

Lets get started with the hardware connections. The pinout of most stand-alone diodes is like this:



To connect a stand-alone receiver diode, wire it like this:



Programming the IR Receiver

Once you have the receiver connected, we can install the Arduino library and start programming. In the examples below, I’ll show you how to find the codes sent by your remote, how to find the IR protocol used by your remote, how to print key presses to the serial monitor or an LCD, and finally, how to control the Arduino’s output pins with a remote.

Install the IR remote library

This lab will be using the IRremote library for all of the code examples below. You can download a ZIP file of the library from <http://z3t0.github.io/Arduino-IRremote/> .

To install the library from the ZIP file, open up the Arduino IDE, then go to Sketch > Include Library > Add .ZIP Library, then select the IRremote ZIP file that you downloaded from the link above.

Find the codes for the remote

To find the key codes for your remote control, upload this code to your Arduino and open the serial monitor:

*#include <IRremote.h>*

*const int RECV\_PIN = 7;*

*IRrecv irrecv(RECV\_PIN);*

*decode\_results results;*

*void setup(){*

*Serial.begin(9600);*

*irrecv.enableIRIn();*

*irrecv.blink13(true);*

*}*

*void loop(){*

*if (irrecv.decode(&results)){*

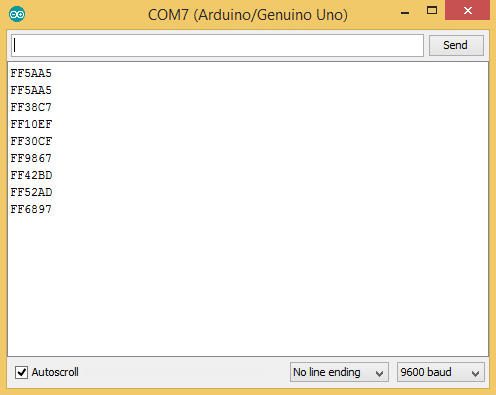
*Serial.println(results.value, HEX);*

*irrecv.resume();*

*}*

*}*

Now press each key on your remote and record the hexadecimal code printed for each key press.



Using the program above, I derived a table of keys and their corresponding codes from the remote that came with my [HX1838 IR receiver and remote set](https://www.amazon.com/gp/product/B015SMS5L0/ref=as_li_qf_sp_asin_il_tl?ie=UTF8&tag=circbasi-20&camp=1789&creative=9325&linkCode=as2&creativeASIN=B015SMS5L0&linkId=8e7a6e8ea0bbbedcaa61a972fd76165b). Note that you will receive a 0XFFFFFFFF code when you press a key continuously.

|  |  |
| --- | --- |
| **Key** | **Code** |
| CH- | 0xFFA25D |
| CH | 0xFF629D |
| CH+ | 0xFFE21D |
| << | 0xFF22DD |
| >> | 0xFF02FD |
| >|| | 0xFFC23D |
| – | 0xFFE01F |
| + | 0xFFA857 |
| EQ | 0xFF906F |
| 100+ | 0xFF9867 |
| 200+ | 0xFFB04F |
| 0 | 0XFF6897 |
| 1 | 0xFF30CF |
| 2 | 0xFF18E7 |
| 3 | 0xFF7A85 |
| 4 | 0xFF10EF |
| 5 | 0xFF38C7 |
| 6 | 0xFF5AA5 |
| 7 | 0xFF42BD |
| 8 | 0xFF4AB5 |
| 9 | 0xFF52AD |

Find the protocol used by your remote

Knowing which protocol your remote uses can be useful if you want to work on some more advanced projects. Or you might just be curious. The program below will identify the protocol used by your remote. It even works on most remote controls around your house.

*#include <IRremote.h>*

*const int RECV\_PIN = 7;*

*IRrecv irrecv(RECV\_PIN);*

*decode\_results results;*

*void setup(){*

*Serial.begin(9600);*

*irrecv.enableIRIn();*

*irrecv.blink13(true);*

*}*

*void loop(){*

*if (irrecv.decode(&results)){*

*Serial.println(results.value, HEX);*

*switch (results.decode\_type){*

*case NEC: Serial.println("NEC"); break ;*

*case SONY: Serial.println("SONY"); break ;*

*case RC5: Serial.println("RC5"); break ;*

*case RC6: Serial.println("RC6"); break ;*

*case DISH: Serial.println("DISH"); break ;*

*case SHARP: Serial.println("SHARP"); break ;*

*case JVC: Serial.println("JVC"); break ;*

*case SANYO: Serial.println("SANYO"); break ;*

*case MITSUBISHI: Serial.println("MITSUBISHI"); break ;*

*case SAMSUNG: Serial.println("SAMSUNG"); break ;*

*case LG: Serial.println("LG"); break ;*

*case WHYNTER: Serial.println("WHYNTER"); break ;*

*case AIWA\_RC\_T501: Serial.println("AIWA\_RC\_T501"); break ;*

*case PANASONIC: Serial.println("PANASONIC"); break ;*

*case DENON: Serial.println("DENON"); break ;*

*default:*

*case UNKNOWN: Serial.println("UNKNOWN"); break ;*

*}*

*irrecv.resume();*

*}*

*}*

Print keys to the serial monitor

This example extended the code above to print the key value instead of the hexadecimal code:

*#include <IRremote.h>*

*const int RECV\_PIN = 7;*

*IRrecv irrecv(RECV\_PIN);*

*decode\_results results;*

*unsigned long key\_value = 0;*

*void setup(){*

*Serial.begin(9600);*

*irrecv.enableIRIn();*

*irrecv.blink13(true);*

*}*

*void loop(){*

*if (irrecv.decode(&results)){*

*if (results.value == 0XFFFFFFFF)*

*results.value = key\_value;*

*switch(results.value){*

*case 0xFFA25D:*

*Serial.println("CH-");*

*break;*

*case 0xFF629D:*

*Serial.println("CH");*

*break;*

*case 0xFFE21D:*

*Serial.println("CH+");*

*break;*

*case 0xFF22DD:*

*Serial.println("|<<");*

*break;*

*case 0xFF02FD:*

*Serial.println(">>|");*

*break ;*

*case 0xFFC23D:*

*Serial.println(">|");*

*break ;*

*case 0xFFE01F:*

*Serial.println("-");*

*break ;*

*case 0xFFA857:*

*Serial.println("+");*

*break ;*

*case 0xFF906F:*

*Serial.println("EQ");*

*break ;*

*case 0xFF6897:*

*Serial.println("0");*

*break ;*

*case 0xFF9867:*

*Serial.println("100+");*

*break ;*

*case 0xFFB04F:*

*Serial.println("200+");*

*break ;*

*case 0xFF30CF:*

*Serial.println("1");*

*break ;*

*case 0xFF18E7:*

*Serial.println("2");*

*break ;*

*case 0xFF7A85:*

*Serial.println("3");*

*break ;*

*case 0xFF10EF:*

*Serial.println("4");*

*break ;*

*case 0xFF38C7:*

*Serial.println("5");*

*break ;*

*case 0xFF5AA5:*

*Serial.println("6");*

*break ;*

*case 0xFF42BD:*

*Serial.println("7");*

*break ;*

*case 0xFF4AB5:*

*Serial.println("8");*

*break ;*

*case 0xFF52AD:*

*Serial.println("9");*

*break ;*

*}*

*key\_value = results.value;*

*irrecv.resume();*

*}*

*}*

If your remote sends different codes than the ones in the table above, just replace the hex code in each line where it says:

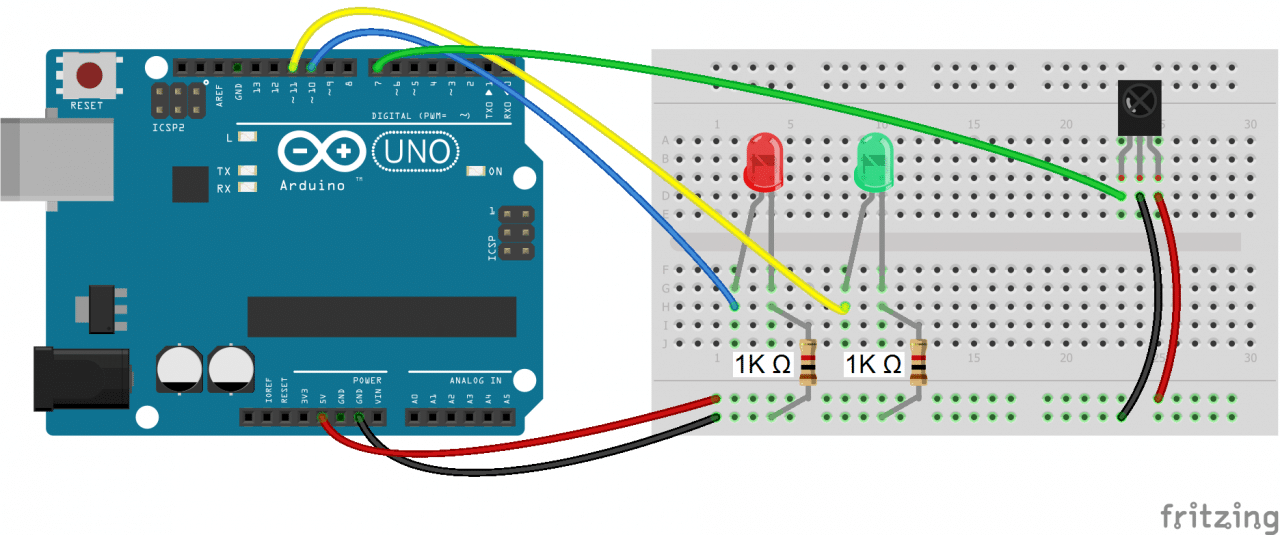
Case 0xFFA25D:  
Serial.println(“CH-“);

In these lines, when the hex code 0xFFA25D is received, the Arduino prints “CH-“.

Using the IR remote to control things

This example will show you a simple demonstration of how you can use the IR remote to control the Arduino’s output pins. In this example, we will light up an LED when a particular button is pressed. You can easily modify the code to do things like control servo motors, or activate relays with any button press from the remote.

The example circuit has the IR receiver connected to the Arduino, with a red LED connected to pin 10 and a green LED connected to pin 11:



The code below will write digital pin 10 HIGH for 2 seco nds when the “5” button is pressed, and write digital pin 11 HIGH for 2 seconds when the”2″ button is pressed:

*#include <IRremote.h>*

*const int RECV\_PIN = 7;*

*IRrecv irrecv(RECV\_PIN);*

*decode\_results results;*

*const int redPin = 10;*

*const int greenPin = 11;*

*void setup(){*

*irrecv.enableIRIn();*

*irrecv.blink13(true);*

*pinMode(redPin, OUTPUT);*

*pinMode(greenPin, OUTPUT);*

*}*

*void loop(){*

*if (irrecv.decode(&results)){*

*switch(results.value){*

*case 0xFF38C7: //Keypad button "5"*

*digitalWrite(redPin, HIGH);*

*delay(2000);*

*digitalWrite(redPin, LOW);*

*}*

*switch(results.value){*

*case 0xFF18E7: //Keypad button "2"*

*digitalWrite(greenPin, HIGH);*

*delay(2000);*

*digitalWrite(greenPin, LOW);*

*}*

*irrecv.resume();*

*}*

*}*

So far we have covered the properties of infrared radiation and how communication happens between the transmitter and receiver. We saw how to identify the IR key codes for a given remote control. Finally I showed you how to control the Arduino’s output with the remote.