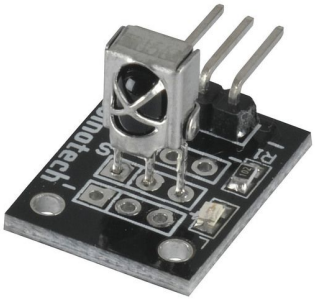
IR Receiver Module

An Arduino guide by the one and only Spudnit

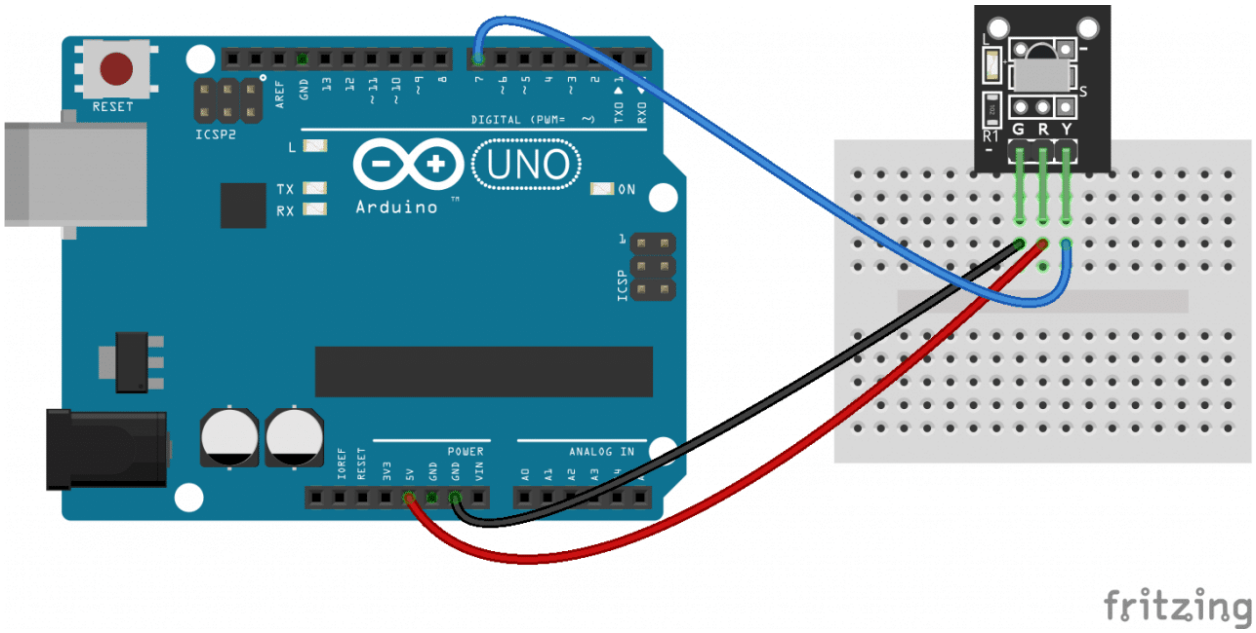


What is an IR Receiver Module?

An IR receiver module is, effectively, the “brain” that remote controls direct their signals to. The “IR” stands for infrared, which is the type of light that remote controls employ.

By connecting an IR receiver module to a breadboard (and, from there, a microcontroller), you can set up your board to be remote controllable.

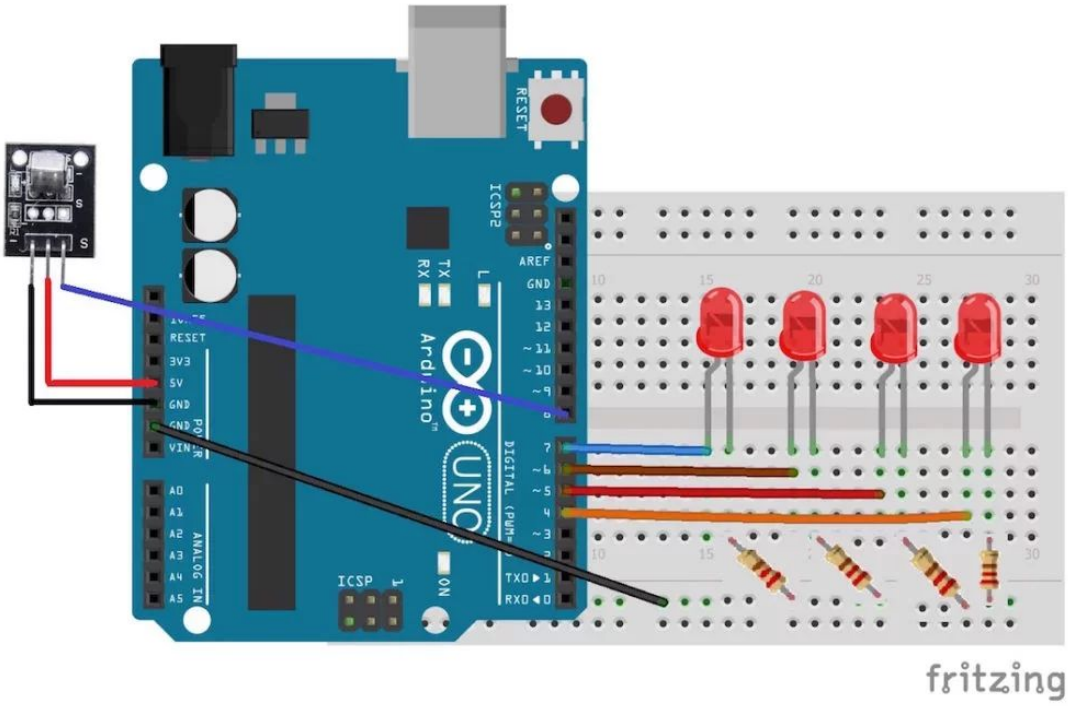
Setup with Arduino Board (or comparable microcontroller)



Since the receiver accepts incoming signals from a remote control, the one provided in the start kit should also be on hand when constructing anything with this device (obviously [at least I hope that’d be obvious]).

Using the IR Receiver Module: Setup

After setting up the receiver module, you'll want to connect something to the board to control using the remote. In this case, we'll use LEDs, like shown:



Using the IR Receiver Module: Code

Open Arduino and copy/paste the following code in. Each segment will be explained.

```
#include
#define first_key  48703
#define second_key  58359
#define third_key  539
#define fourth_key  25979
int receiver_pin = 8;
```

```
int first_led_pin = 7;
int second_led_pin = 6;
int third_led_pin = 5;
int fourth_led_pin = 4;
int led[] = {0,0,0,0};
IRrecv receiver(receiver_pin);
decode_results output;
```

Left: The numbers listed next to "first_key", etc. are all changeable by you. They reflect the unique code that is associated with each of the keys on the remote. If you want to customize the keys shown here, you can upload the entire block of code (copy in everything from start to finish) and then press keys; the code for each key will show up as serial output, and you can adjust the code according to what key you want to correspond to each variable.
Right: Define your LED pins based on where you've connected each LED from breadboard to microcontroller. The "led" array creates an array of four zeroes, indicating that nothing has been

turned on. The IRrecv line sets up the receiver using whatever pin you've connected it to on your microcontroller.

```
void setup()
{
Serial.begin(9600);
receiver.enableIRIn();
pinMode(first_led_pin, OUTPUT);
pinMode(second_led_pin, OUTPUT);
pinMode(third_led_pin, OUTPUT);
pinMode(fourth_led_pin, OUTPUT);
}
```

```
void loop() {
if (receiver.decode(&output)) {
unsigned int value = output.value;
switch(value) {
case first_key:
if(led[1] == 1) {
digitalWrite(first_led_pin, LOW);
led[1] = 0;
} else {
digitalWrite(first_led_pin, HIGH);
led[1] = 1;
}
break;
case second_key:
```

Left: Set up your board by assigning each of the LEDs as an output. Remember that the LEDs were defined as numbers in the last step.
Right: Check to see if a key has been pressed; if a key has been pressed, see if the code for that key has been indicated in our definitions for each key. The switch/case portion of this just means that the code will check and see if the first key contains the same number as the key code of the button that was pressed; if it does, and the first_key is currently set to 1 in the array (indicating that it's turned on), then the key is set to 0 instead, turning it off. Otherwise, the key is set to 1, lighting up the LED. "break;" just means moving on.

```
if(led[2] == 1) {
digitalWrite(second_led_pin, LOW);
led[2] = 0;
} else {
digitalWrite(second_led_pin, HIGH);
led[2] = 1;
}
break;
case third_key:
```

```
if(led[3] == 1) {
digitalWrite(third_led_pin, LOW);
led[3] = 0;
} else {
digitalWrite(third_led_pin, HIGH);
led[3] = 1;
}
break;
case fourth_key:
```

Copy these in; they are the same as the last brick, just with different LED numbers. The numbers represent the position of the key within the array.

```
if(led[4] == 1) {
digitalWrite(fourth_led_pin, LOW);
led[4] = 0;
} else {
digitalWrite(fourth_led_pin, HIGH);
led[4] = 1;
}
break;
}
Serial.println(value);
receiver.resume();
}
}
```

This last block is also the same, but includes a serial print line so that you can see what value corresponds to each key of the remote. The receiver.resume() function allows the receiver to continually input information.

Njoi