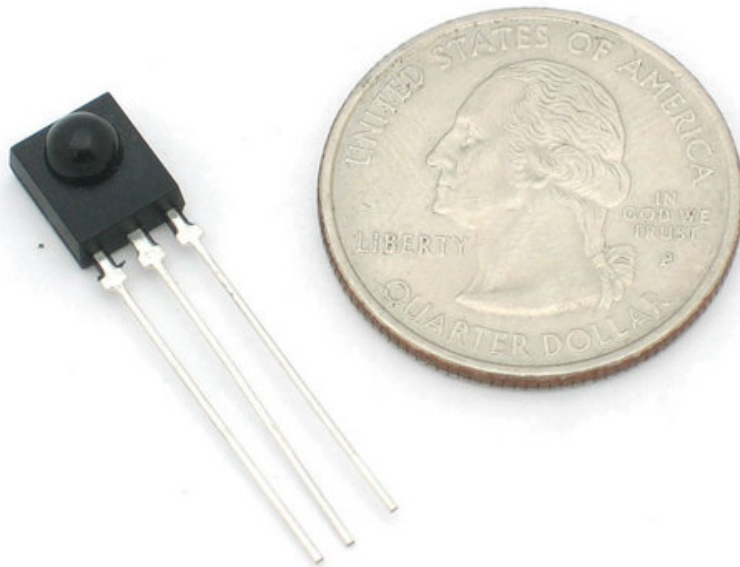




## IR Sensor

Created by Ladyada

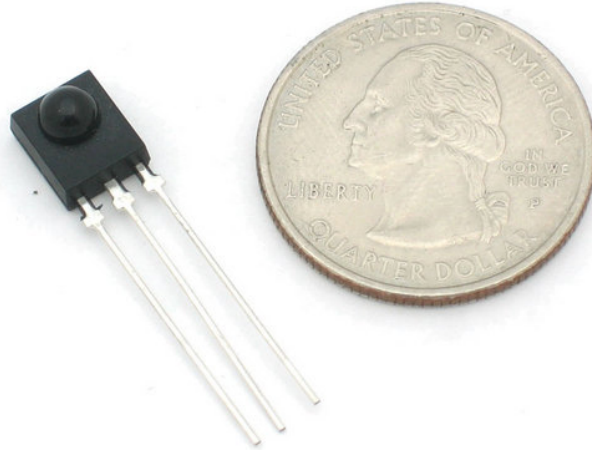


## Guide Contents

Guide Contents	2
Overview	3
Some Stats	4
What You Can Measure	4
Testing an IR Sensor	6
IR Remote Signals	8
Using an IR Sensor	12
Making an Intervalometer	16
Reading IR Commands	21
Buy an IR Sensor	29

## Overview

---



IR detectors are little microchips with a photocell that are tuned to listen to infrared light. They are almost always used for remote control detection - every TV and DVD player has one of these in the front to listen for the IR signal from the clicker. Inside the remote control is a matching IR LED, which emits IR pulses to tell the TV to turn on, off or change channels. IR light is not visible to the human eye, which means it takes a little more work to test a setup.

There are a few difference between these and say a [CdS Photocells \(http://adafru.it/aK7\)](http://adafru.it/aK7) :

- IR detectors are specially filtered for Infrared light, they are not good at detecting visible light. On the other hand, photocells are good at detecting yellow/green visible light, not good at IR light
- IR detectors have a **demodulator** inside that looks for modulated IR at 38 KHz. Just shining an IR LED wont be detected, it has to be PWM blinking at 38KHz. Photocells do not have any sort of demodulator and can detect any frequency (including DC) within the response speed of the photocell (which is about 1KHz)
- IR detectors are digital out - either they detect 38KHz IR signal and output low (0V) or they do not detect any and output high (5V). Photocells act like resistors, the resistance changes depending on how much light they are exposed to

In this tutorial we will show how to

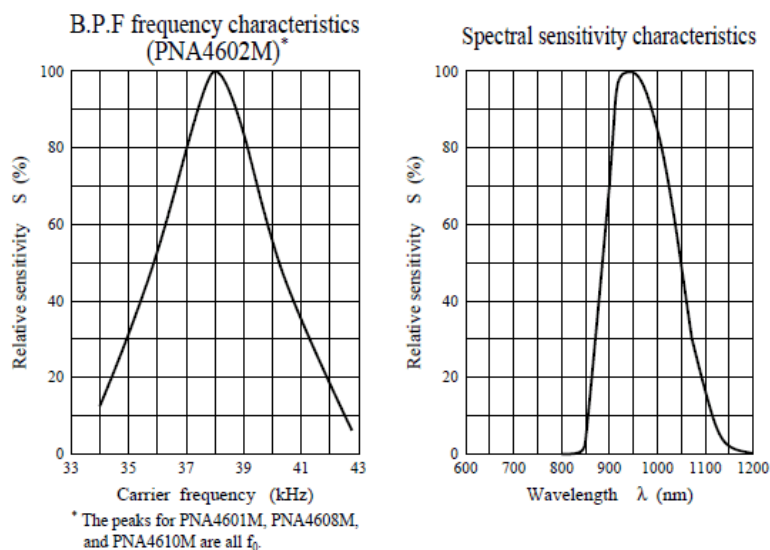
- [Test your IR sensor to make sure its working \(http://adafru.it/aK8\)](http://adafru.it/aK8)
- [Read raw IR codes into a microcontroller \(http://adafru.it/aK9\)](http://adafru.it/aK9)
- [Create a camera intervalometer \(http://adafru.it/aKa\)](http://adafru.it/aKa)
- [Listen for 'commands' from a remote control on your microcontroller \(http://adafru.it/aKa\)](http://adafru.it/aKa)

## Some Stats

These stats are for the [IR detector in the Adafruit shop \(http://adafru.it/aIH\)](http://adafru.it/aIH) also known as PNA4602. Nearly all photocells will have slightly different specifications, although they all pretty much work the same. If there's a datasheet, you'll want to refer to it

- **Size:** square, 7mm by 8mm detector area
- **Price:** \$2.00 at the [Adafruit shop \(http://adafru.it/aIH\)](http://adafru.it/aIH)
- **Output:** 0V (low) on detection of 38KHz carrier, 5V (high) otherwise
- **Sensitivity range:** 800nm to 1100nm with peak response at 940nm. Frequency range is 35KHz to 41KHz with peak detection at 38KHz
- **Power supply:** 3-5V DC 3mA
- **PNA4602 Datasheet (<http://adafru.it/aKb>)** (now discontinued)  
or **GP1UX311QS (<http://adafru.it/aKc>)** or **TSOP38238 (<http://adafru.it/aKd>)** (pin-compatible replacements)

## What You Can Measure



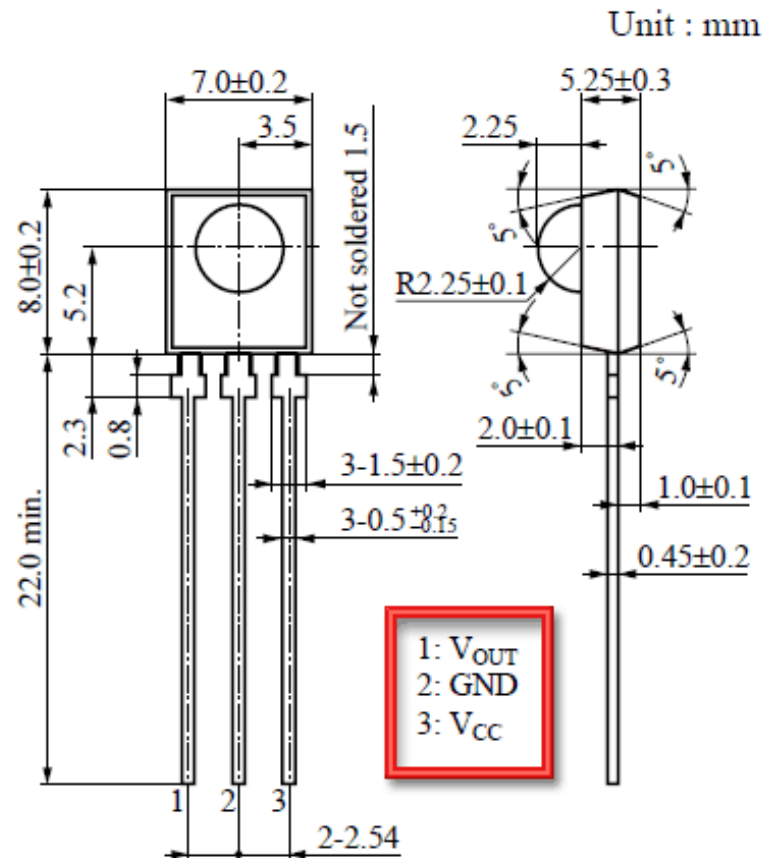
As you can see from these datasheet graphs, the peak frequency detection is at **38 KHz** and the peak LED color is **940 nm**. You can use from about 35 KHz to 41 KHz but the sensitivity will drop off so that it won't detect as well from afar. Likewise, you can use 850 to 1100 nm LEDs but they won't work as well as 900 to 1000nm so make sure to get matching LEDs! Check the datasheet for your IR LED to verify the wavelength.

Try to get a 940nm - remember that 940nm is not visible light (its Infra Red)!



## Testing an IR Sensor

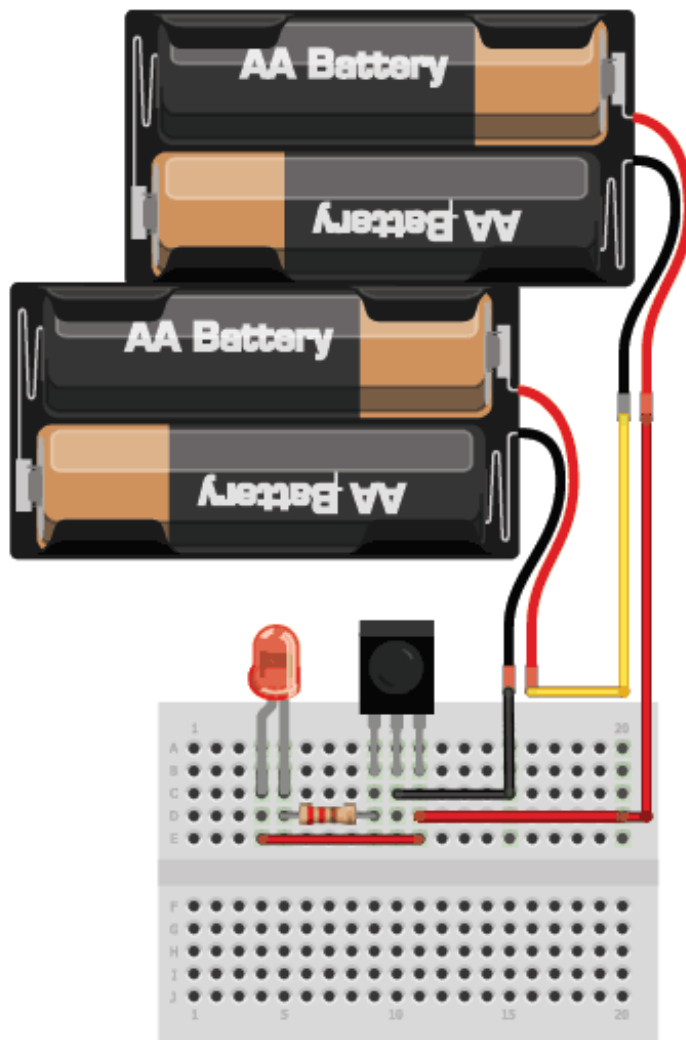
Because there is a semiconductor/chip inside the sensor, it must be powered with 5V to function. Contrast this to photocells and FSRs where they act like resistors and thus can be simply tested with a multimeter.



Here we will connect the detector as such:

- Pin 1 is the output so we wire this to a visible LED and resistor
- Pin 2 is ground
- Pin 3 is VCC, connect to 5V

When the detector sees IR signal, it will pull the output low, turning on the LED - since the LED is red its much easier for us to see than IR!



We will use 4xAA 1.5V batteries so that the voltage powering the sensor is about 6V. 2 batteries (3V) is too little. You can also get 5V from a microcontroller like an Arduino if you have one around. Ground goes to the middle pin.

The positive (longer) head of the Red LED connects to the +6V pin and the negative (shorter lead) connects through a 200 to 1000 ohm resistor to the first pin on the IR sensor.

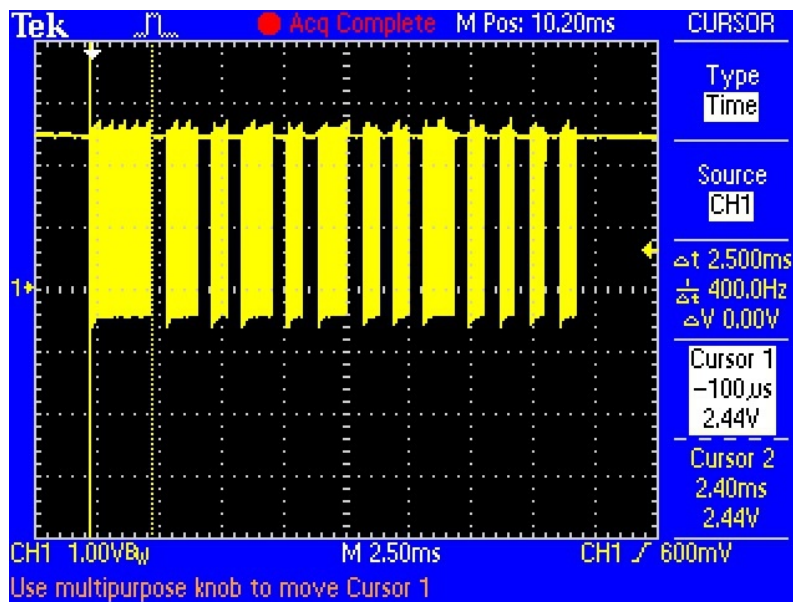
Now grab any remote control like for a TV, DVD, computer, etc. and point it at the detector while pressing some buttons, you should see the LED blink a couple times whenever the remote is pressed

## IR Remote Signals

Now we know that the sensor works, we want to figure out whats being sent right? But before we do that let's first examine exactly how data is being sent from the IR remote (in your hand) to the IR receiving sensor (on the breadboard)

For this example we will use the Sony power on/off IR code from a Sony TV remote. Its very simple and commonly documented!

Lets pretend we have a Sony remote, and we can look at exactly what light is being blasted out of the IR LED. We'll hookup a basic light sensor (like a basic photocell!) and listen in. We won't use a decoder like a PNA4602 (just yet) because we want to see the undecoded signal. What we see is the following:

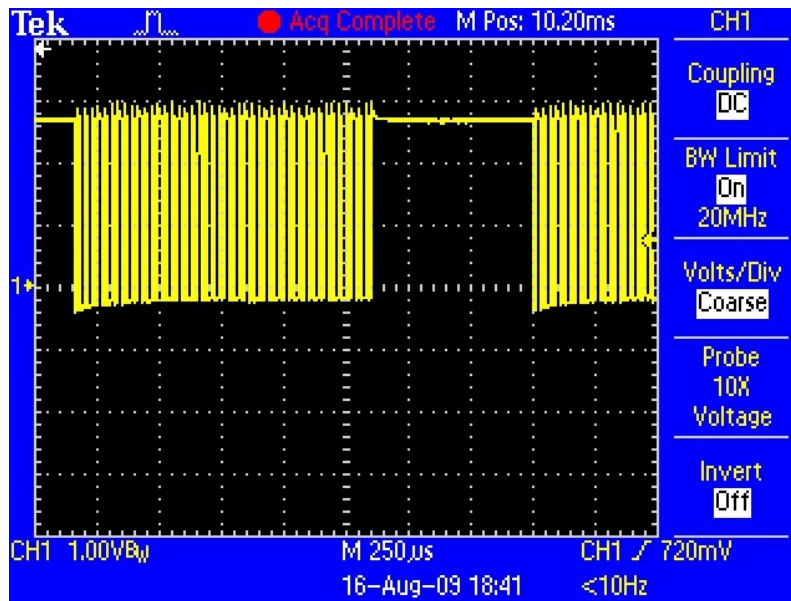


Basically we see pulses or IR signal. the yellow 'blocks' are when the IR LED is transmitting and when there is only a line, the IR LED is off. (Note that the voltage being at 3VDC is just because of the way I hooked up the sensor, if I had swapped the pullup for a pulldown it would be at ground.)

The first 'block' is about 2.5ms long (see the cursors and the measurement on the side)

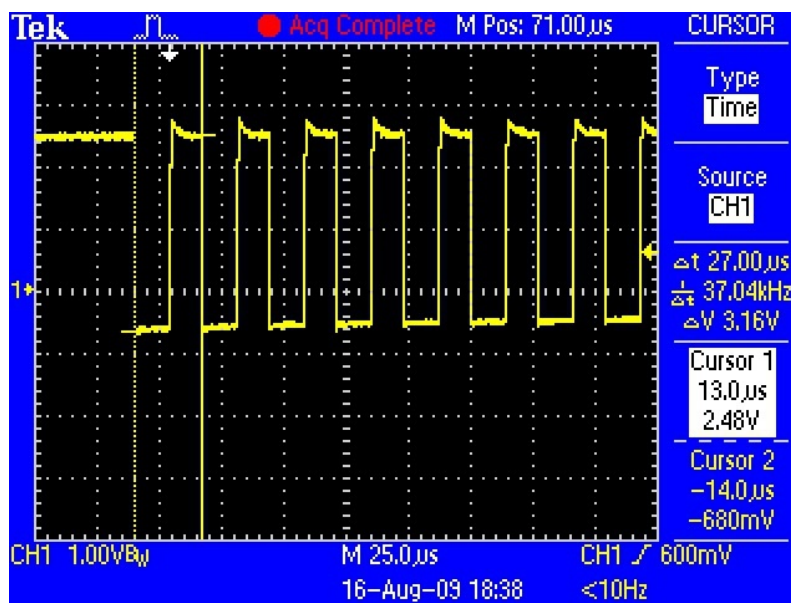
If you zoom into one of those blocks...





You see that they're not really 'blocks' but actually very fast pulses!

If you zoom in all the way...



You can measure the frequency of the IR pulses. As you can tell by the cursors and the measurements on the side, the frequency is about 37.04KHz

OK so now we can understand how IR codes are sent. The IR transmitter LED is quickly pulsed (PWM - pulse width modulated) at a high frequency of 38KHz and then that PWM is likewise pulsed on and off much slower, at times that are about 1-3 ms long.

Why not have the LED just on and off? Why have PWM 'carrier' pulsing? Many reasons!

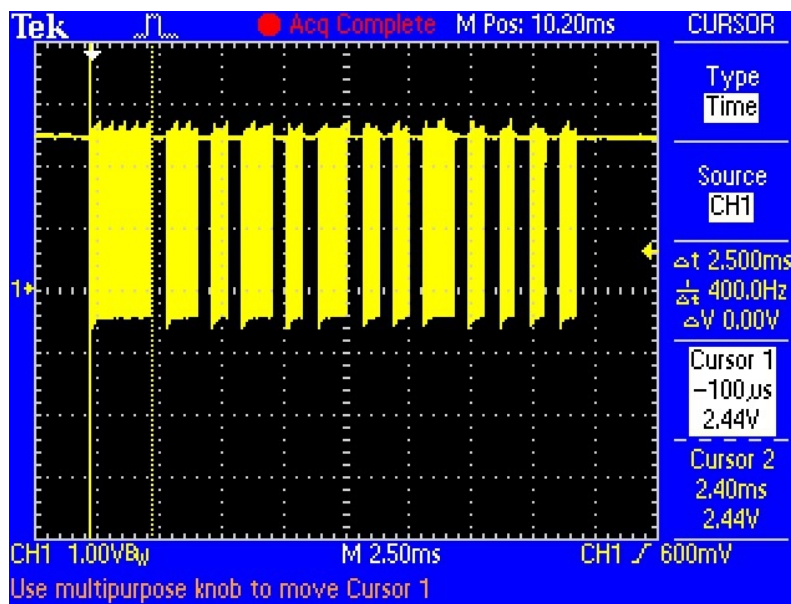
One reason is that this lets the LED cool off. IR LEDs can take up to 1 Amp (1000 milliamps!) of current. Most LEDs only take 20mA or so. This means IR LEDs are designed for high-power blasting BUT they can only take it for a few microseconds. By PWM'ing it, you let the LED cool off half the time

Another reason is that the TV will only listen to certain frequencies of PWM. So a Sony remote at 37KHz wont be able to work with a JVC DVD player that only wants say 50KHz.

Finally, the most important reason is that by pulsing a carrier wave, you reduce the affects of ambient lighting. The TV only looks for changes in light levels that clock in around 37KHz. Just like its easier for us to tell differences between audio tones than to pin down the precsise pitch of a tone (well, for most people at least)

OK so now we know the carrier frequency. Its 37KHz. Next lets find the pulse widths!

Looking back at the first scope picture



The first pulse is 2.5ms. We can use the cursors to measure the remaining pulses. I'll spare you the 12 images and let you know that the pulses are:

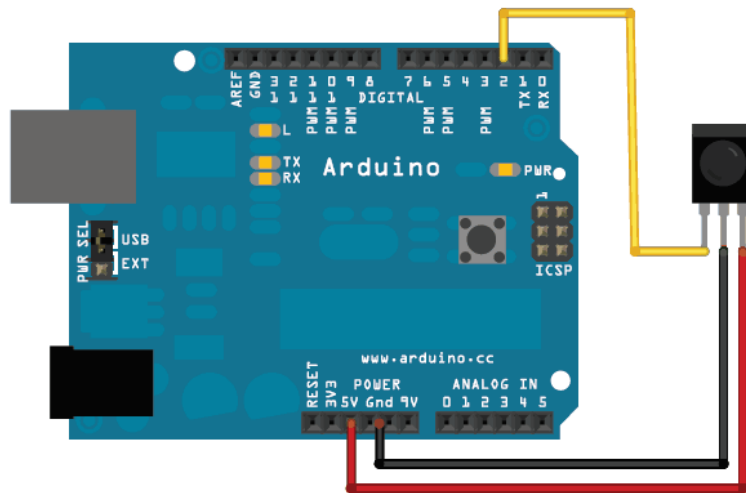
PWM ON	OFF
2.4 ms	0.6 ms
1.2 ms	0.6 ms
0.6 ms	0.6 ms
1.2 ms	0.6 ms
0.6 ms	0.6 ms
1.2 ms	0.6 ms

0.6 ms	0.6 ms
0.6 ms	0.6 ms
1.2 ms	0.6 ms
0.6 ms	0.6 ms
0.6 ms	0.6 ms
0.6 ms	0.6 ms
0.6 ms	270 ms

So lets say you don't have a \$1000 oscilloscope, how else can you read these signals? Well the IR decoder such as the PNA4602 does us one favor, it 'filters out' the 38KHz signal so that we only get the big chunks of signal in the millisecond range. This is much easier for a microcontroller to handle. Thats what we'll do in the next section!

## Using an IR Sensor

The good news is that it is very easy to hook up this sensor. Just connect the output to a digital pin. The bad news is that the Arduino's friendly **digitalRead()** procedure is a tad too slow to reliably read the fast signal as its coming in. Thus we use the hardware pin reading function directly from pin D2, thats what the line "IRpin\_PIN & BV(IRpin))" does.



You can also get the latest version of this code on github (<http://adafru.it/aKe>).

```

/* Raw IR decoder sketch!
This sketch/program uses the Arduno and a PNA4602 to
decode IR received. This can be used to make a IR receiver
(by looking for a particular code)
or transmitter (by pulsing an IR LED at ~38KHz for the
durations detected
Code is public domain, check out www.ladyada.net and adafruit.com
for more tutorials!
*/

// We need to use the 'raw' pin reading methods
// because timing is very important here and the digitalRead()
// procedure is slower!
//uint8_t IRpin = 2;
// Digital pin #2 is the same as Pin D2 see
// http://arduino.cc/en/Hacking/PinMapping168 for the 'raw' pin mapping
#define IRpin_PIN PIND
#define IRpin_2
// for MEGA use these!
// #define IRpin_PIN PINE
// #define IRpin_4

// the maximum pulse we'll listen for - 65 milliseconds is a long time
#define MAXPULSE 65000

```

```

// what our timing resolution should be, larger is better
// as its more 'precise' - but too large and you wont get
// accurate timing
#define RESOLUTION 20

// we will store up to 100 pulse pairs (this is -a lot-)
uint16_t pulses[100][2]; // pair is high and low pulse
uint8_t currentpulse = 0; // index for pulses we're storing

void setup(void) {
  Serial.begin(9600);
  Serial.println("Ready to decode IR!");
}

void loop(void) {
  uint16_t highpulse, lowpulse; // temporary storage timing
  highpulse = lowpulse = 0; // start out with no pulse length

  // while (digitalRead(IRpin)) { // this is too slow!
  while (IRpin_PIN & (1 << IRpin)) {
    // pin is still HIGH

    // count off another few microseconds
    highpulse++;
    delayMicroseconds(RESOLUTION);

    // If the pulse is too long, we 'timed out' - either nothing
    // was received or the code is finished, so print what
    // we've grabbed so far, and then reset
    if ((highpulse >= MAXPULSE) && (currentpulse != 0)) {
      printpulses();
      currentpulse=0;
      return;
    }
  }
  // we didn't time out so lets stash the reading
  pulses[currentpulse][0] = highpulse;

  // same as above
  while (! (IRpin_PIN & _BV(IRpin))) {
    // pin is still LOW
    lowpulse++;
    delayMicroseconds(RESOLUTION);
    if ((lowpulse >= MAXPULSE) && (currentpulse != 0)) {
      printpulses();
      currentpulse=0;
      return;
    }
  }
  pulses[currentpulse][1] = lowpulse;

  // we read one high-low pulse successfully, continue!
  currentpulse++;
}

void printpulses(void) {

```

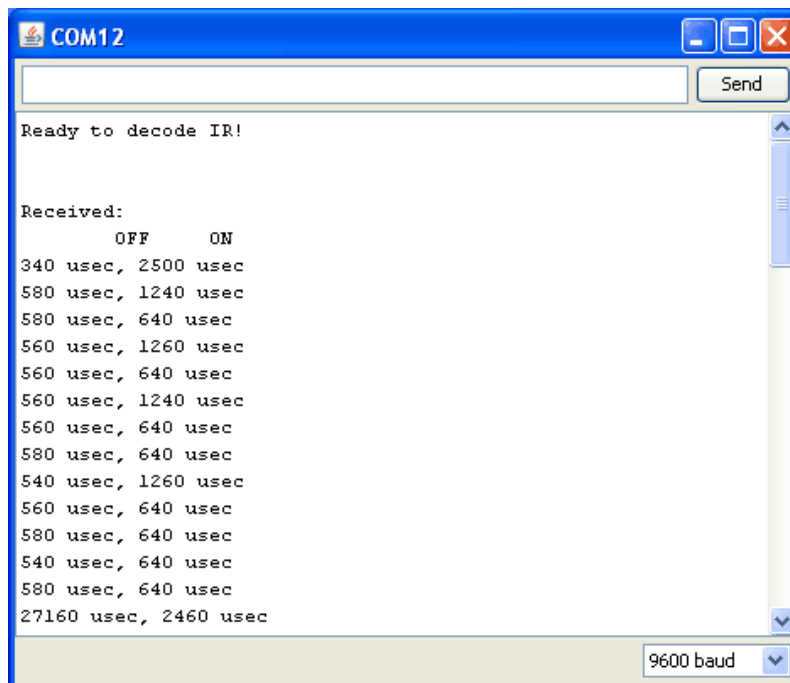
```

Serial.println("\n\nReceived: \n\nOFF \tON");
for (uint8_t i = 0; i < currentpulse; i++) {
  Serial.print(pulses[i][0] * RESOLUTION, DEC);
  Serial.print(" usec, ");
  Serial.print(pulses[i][1] * RESOLUTION, DEC);
  Serial.println(" usec");
}

// print it in a 'array' format
Serial.println("int IRsignal[] = {");
Serial.println("// ON, OFF (in 10's of microseconds)");
for (uint8_t i = 0; i < currentpulse-1; i++) {
  Serial.print("\t"); // tab
  Serial.print(pulses[i][1] * RESOLUTION / 10, DEC);
  Serial.print(", ");
  Serial.print(pulses[i+1][0] * RESOLUTION / 10, DEC);
  Serial.println(";");
}
Serial.print("\t"); // tab
Serial.print(pulses[currentpulse-1][1] * RESOLUTION / 10, DEC);
Serial.print(", 0};");
}

```

If you run this while pointing a Sony IR remote and pressing the ON button you will get the following...



If you ignore the first OFF pulse (its just the time from when the Arduino turned on to the first IR signal received) and the last ON pulse (it the beginning of the next code) you'll find the Sony power code:

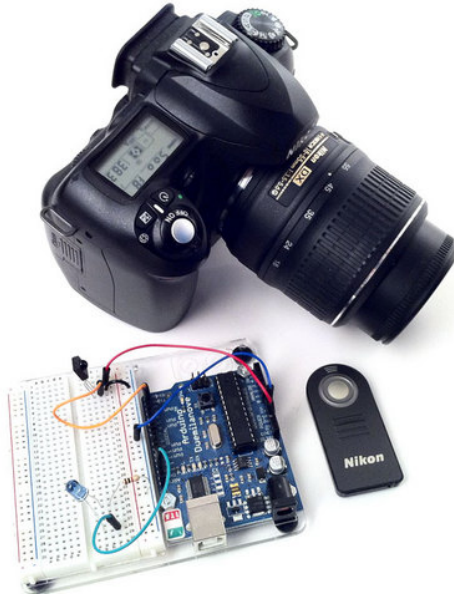
PWM ON	OFF
2.5 ms	0.6 ms

1.2 ms	0.6 ms
0.6 ms	0.6 ms
1.2 ms	0.6 ms
0.6 ms	0.6 ms
1.2 ms	0.6 ms
0.6 ms	0.6 ms
0.6 ms	0.6 ms
1.2 ms	0.6 ms
0.6 ms	0.6 ms
0.6 ms	0.6 ms
0.6 ms	0.6 ms
0.6 ms	270 ms

## Making an Intervalometer

---

OK now that we can read IR codes, lets make a basic project. The first one we will do is to make an intervalometer. An intervalometer is basically a electronic thingy that makes a camera go off every few minutes or so. This can be used for timelapse projects or kite arial photography or other photo projects.



The camera we'll be using has an IR remote you can use to set it off (most higher-end cameras have these).

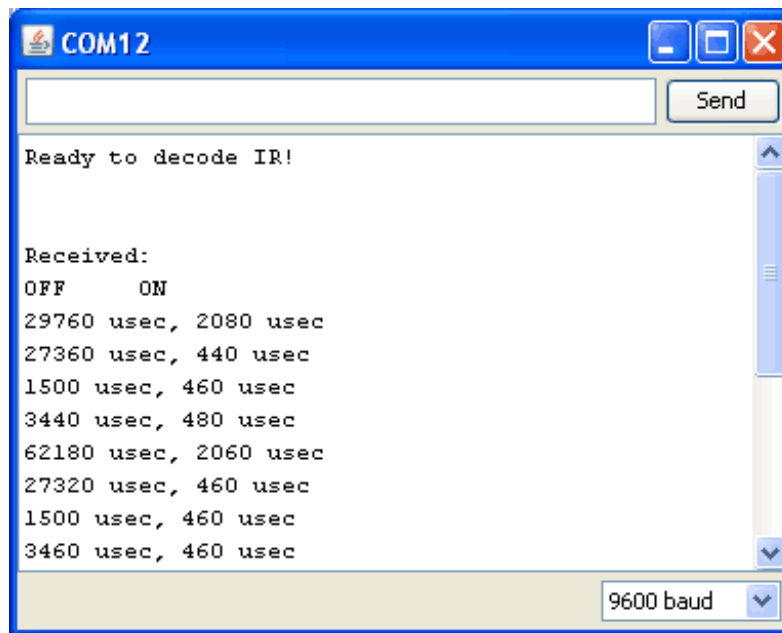


First we will figure out the codes by reading the signal sent when the button is pressed. Then



we'll take that data and make the Arduino spit out that code into an IR LED once a minute

OK step one is easy, point the remote control at the IR sensor and press the button, we got the following for our ML-L3 Nikon remote.



Looks like the data sent is:

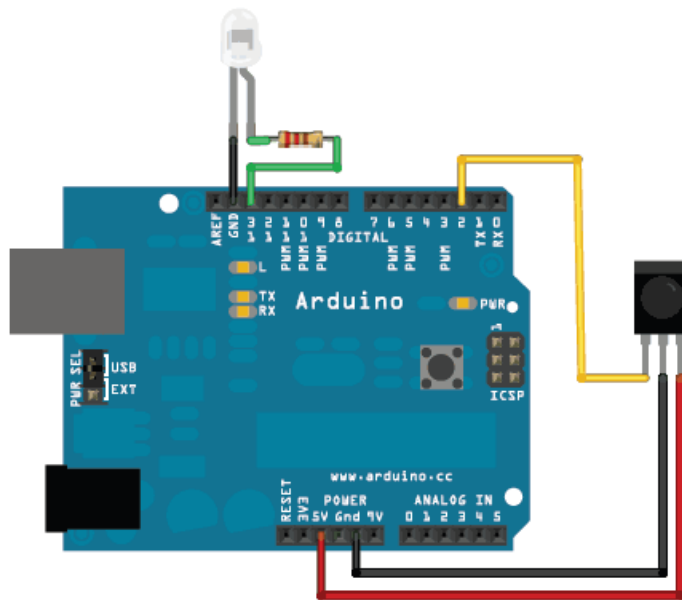
PWM ON	OFF
2.0 ms	27 ms
0.4 ms	1.5 ms
0.5 ms	3.5 ms
0.5 ms	62.2 ms
2.0 ms	27 ms
0.5 ms	1.5 ms
0.5 ms	3.5 ms
0.5 ms	

If you look closely you'll see its actually just

PWM ON	OFF
2.0 ms	27 ms
0.4 ms	1.5 ms
0.5 ms	3.5 ms
0.5 ms	62.2 ms

sent twice. Sending the same signal twice is very common - doubling up to make sure it gets received

Next up we'll need to connect an IR 940nm LED to the output of the Arduino



Then we'll write a sketch which will pulse pin #13 on and off very fast in the proper code sequence.

```
// This sketch will send out a Nikon D50 trigger signal (probably works with most Nikons)
// See the full tutorial at http://www.ladyada.net/learn/sensors/ir.html
// this code is public domain, please enjoy!
```

```
int IRledPin = 13; // LED connected to digital pin 13
```

```
// The setup() method runs once, when the sketch starts
```

```
void setup() {
  // initialize the IR digital pin as an output:
  pinMode(IRledPin, OUTPUT);

  Serial.begin(9600);
}
```

```
void loop()
{
  Serial.println("Sending IR signal");

  SendNikonCode();

  delay(60*1000); // wait one minute (60 seconds * 1000 milliseconds)
}
```

```
// This procedure sends a 38KHz pulse to the IRledPin
// for a certain # of microseconds. We'll use this whenever we need to send codes
```

```
void pulseIR(long microsecs) {
  // we'll count down from the number of microseconds we are told to wait
```

```

cli(); // this turns off any background interrupts

while (microsecs > 0) {
  // 38 kHz is about 13 microseconds high and 13 microseconds low
  digitalWrite(IRledPin, HIGH); // this takes about 3 microseconds to happen
  delayMicroseconds(10);        // hang out for 10 microseconds, you can also change this to 9 if it
  digitalWrite(IRledPin, LOW);  // this also takes about 3 microseconds
  delayMicroseconds(10);        // hang out for 10 microseconds, you can also change this to 9 if it

  // so 26 microseconds altogether
  microsecs -= 26;
}

sei(); // this turns them back on
}

void SendNikonCode() {
  // This is the code for my particular Nikon, for others use the tutorial
  // to 'grab' the proper code from the remote

  pulseIR(2080);
  delay(27);
  pulseIR(440);
  delayMicroseconds(1500);
  pulseIR(460);
  delayMicroseconds(3440);
  pulseIR(480);

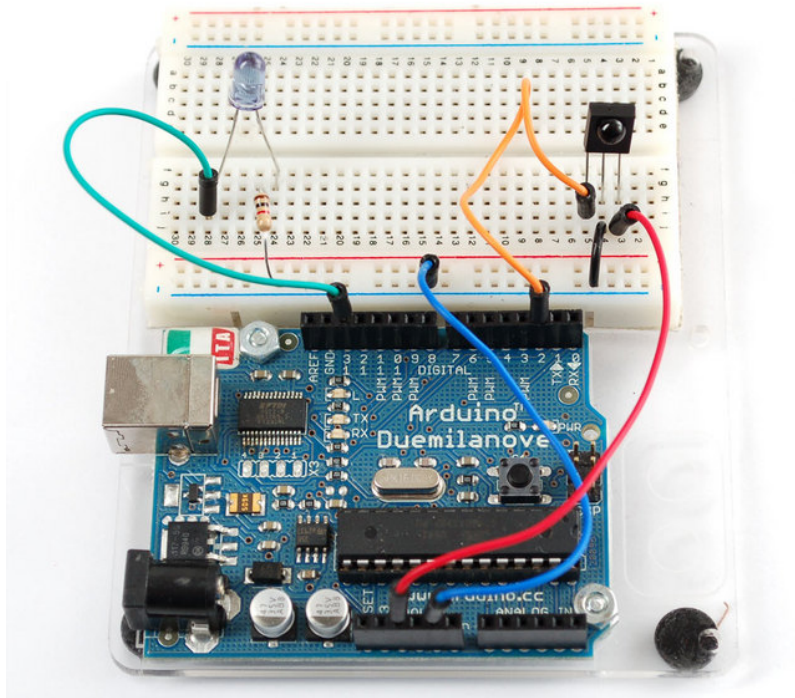
  delay(65); // wait 65 milliseconds before sending it again

  pulseIR(2000);
  delay(27);
  pulseIR(440);
  delayMicroseconds(1500);
  pulseIR(460);
  delayMicroseconds(3440);
  pulseIR(480);
}

```

void pulseIR(long microsecs) is our helper procedure, it will create the PWM IR signal like we saw before. I used my scope to fine-tune it so that the delays added up right. We use the not-often-discussed **cli()** and **sei()** procedures to turn off interrupts. The arduino does a couple things in the background like looking for serial data to read or write, keeping track of time, etc. Most of the time we can just ignore it but for delicate high speed signals like this we want to keep quiet so that we get a nice clean signal

If you look at SendNikonCode() you will see the IR command code that we deduced in the previous project by timing the pulses from the IR sensor.



We wired this up and it worked great, make sure to point the IR LED at the camera properly

You can also get the latest code at github (<http://adafru.it/aKf>)

## Reading IR Commands

For our final project, we will use a remote control to send messages to a microcontroller. For example, this might be useful for a robot that can be directed with an IR remote. It can also be good for projects that you want to control from far away, without wires.

For a remote in this example we'll be using an Apple clicker remote. You can use any kind of remote you wish, or you can steal one of these from an unsuspecting hipster.



We'll use the code from our previous sketch for raw IR reading but this time we'll edit our printer-outter to have it give us the pulses in a C array, this will make it easier for us to use for pattern matching.

```
void printpulses(void) {
  Serial.println("\n\nReceived: \n\nOFF \tON");
  for (uint8_t i = 0; i < currentpulse; i++) {
    Serial.print(pulses[i][0] * RESOLUTION, DEC);
    Serial.print(" usec, ");
    Serial.print(pulses[i][1] * RESOLUTION, DEC);
    Serial.println(" usec");
  }

  // print it in a 'array' format
  Serial.println("int IRsignal[] = {");
  Serial.println("// ON, OFF (in 10's of microseconds)");
  for (uint8_t i = 0; i < currentpulse-1; i++) {
    Serial.print("\t"); // tab
    Serial.print(pulses[i][1] * RESOLUTION / 10, DEC);
    Serial.print(", ");
    Serial.print(pulses[i+1][0] * RESOLUTION / 10, DEC);
    Serial.println(",");
  }
  Serial.print("\t"); // tab
  Serial.print(pulses[currentpulse-1][1] * RESOLUTION / 10, DEC);
  Serial.print(", 0};");
}
```

I uploaded the new sketch and pressed the **Play** button on the Apple remote and got the following:

```
int IRsignal[] = { // ON, OFF (in 10's of microseconds)
912, 438,
68, 48,
68, 158,
68, 158,
68, 158,
68, 48,
68, 158,
68, 158,
68, 158,
70, 156,
70, 158,
68, 158,
68, 48,
68, 46,
70, 46,
68, 46,
68, 160,
68, 158,
70, 46,
68, 158,
68, 46,
70, 46,
68, 48,
68, 46,
68, 48,
66, 48,
68, 48,
66, 160,
66, 50,
66, 160,
66, 52,
64, 160,
66, 48,
66, 3950,
908, 214,
66, 3012,
908, 212,
68, 0};
```

We'll try to detect that code. Lets start a new sketch called **IR Commander** ([you can download the final code from github](http://adafru.it/aKg)) (<http://adafru.it/aKg>) this will use parts of our previous sketch. The first part we'll do is to create a function that just listens for an IR code and puts the pulse timings into the **pulses[]** array. It will return the number of pulses it heard as a return-value.

```
int listenForIR(void) {
    currentpulse = 0;

    while (1) {
        uint16_t highpulse, lowpulse; // temporary storage timing
        highpulse = lowpulse = 0; // start out with no pulse length
```

```

// while (digitalRead(IRpin)) { // this is too slow!
while (IRpin_PIN & (1 << IRpin)) {
    // pin is still HIGH

    // count off another few microseconds
    highpulse++;
    delayMicroseconds(RESOLUTION);

    // If the pulse is too long, we 'timed out' - either nothing
    // was received or the code is finished, so print what
    // we've grabbed so far, and then reset
    if ((highpulse >= MAXPULSE) && (currentpulse != 0)) {
        return currentpulse;
    }
}
// we didn't time out so lets stash the reading
pulses[currentpulse][0] = highpulse;

// same as above
while (!(IRpin_PIN & _BV(IRpin))) {
    // pin is still LOW
    lowpulse++;
    delayMicroseconds(RESOLUTION);
    if ((lowpulse >= MAXPULSE) && (currentpulse != 0)) {
        return currentpulse;
    }
}
pulses[currentpulse][1] = lowpulse;

// we read one high-low pulse successfully, continue!
currentpulse++;
}
}

```

Our new **loop()** will start out just listening for pulses

```

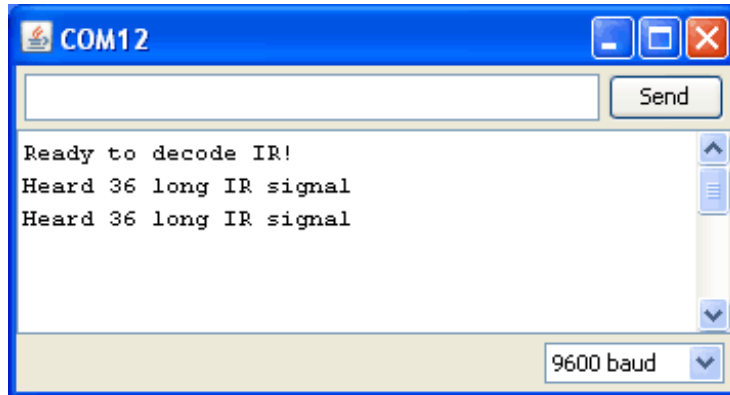
void loop(void) {
    int numberpulses;

    numberpulses = listenForIR();

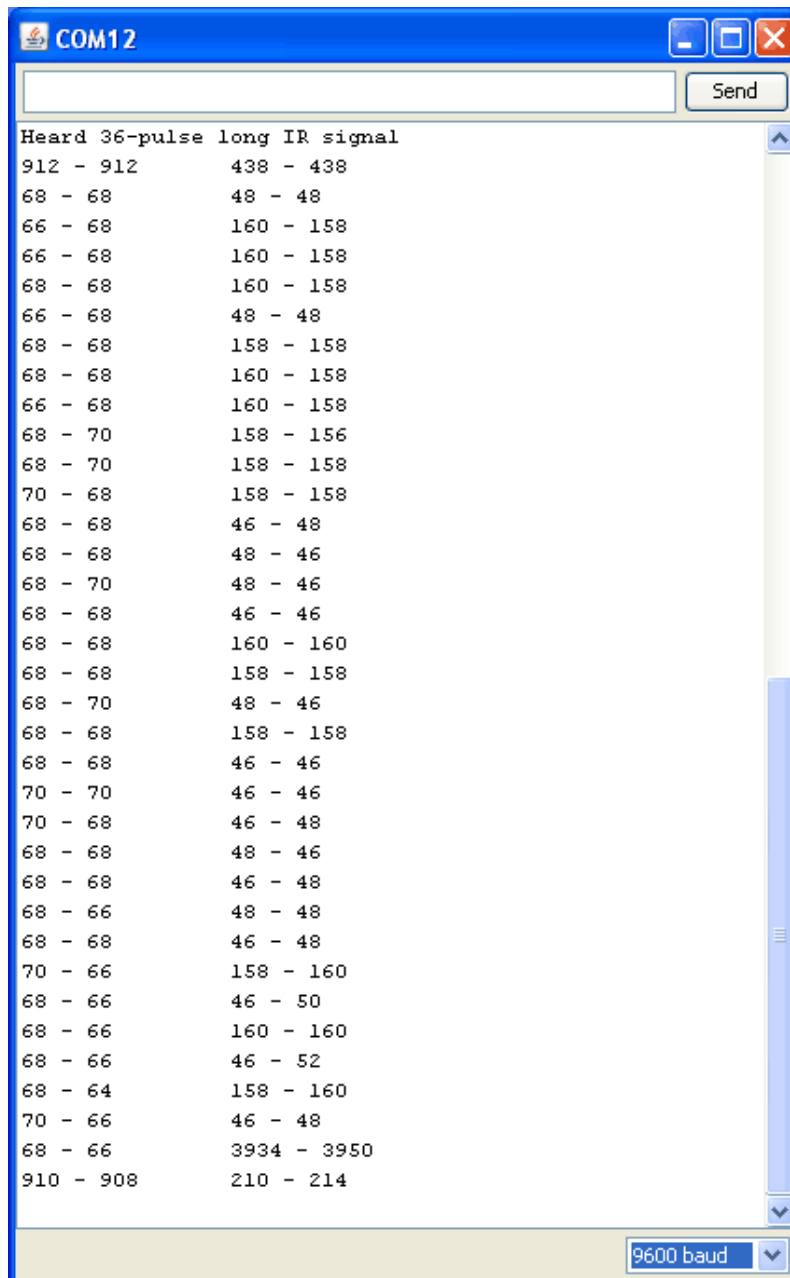
    Serial.print("Heard ");
    Serial.print(numberpulses);
    Serial.println("-pulse long IR signal");
}

```

When we run this it will print out something like...



OK time to make the sketch compare what we received to what we have in our stored array:





As you can see, there is some variation. So when we do our comparison we can't look for precisely the same values, we have to be a little 'fuzzy'. We'll say that the values can vary by 20% - that should be good enough.

```
// What percent we will allow in variation to match the same code \\ #define FUZZINESS 20

void loop(void) {
  int numberpulses;

  numberpulses = listenForIR();

  Serial.print("Heard ");
  Serial.print(numberpulses);
  Serial.println("-pulse long IR signal");

  for (int i=0; i< numberpulses-1; i++) {
    int oncode = pulses[i][1] * RESOLUTION / 10;
    int offcode = pulses[i+1][0] * RESOLUTION / 10;

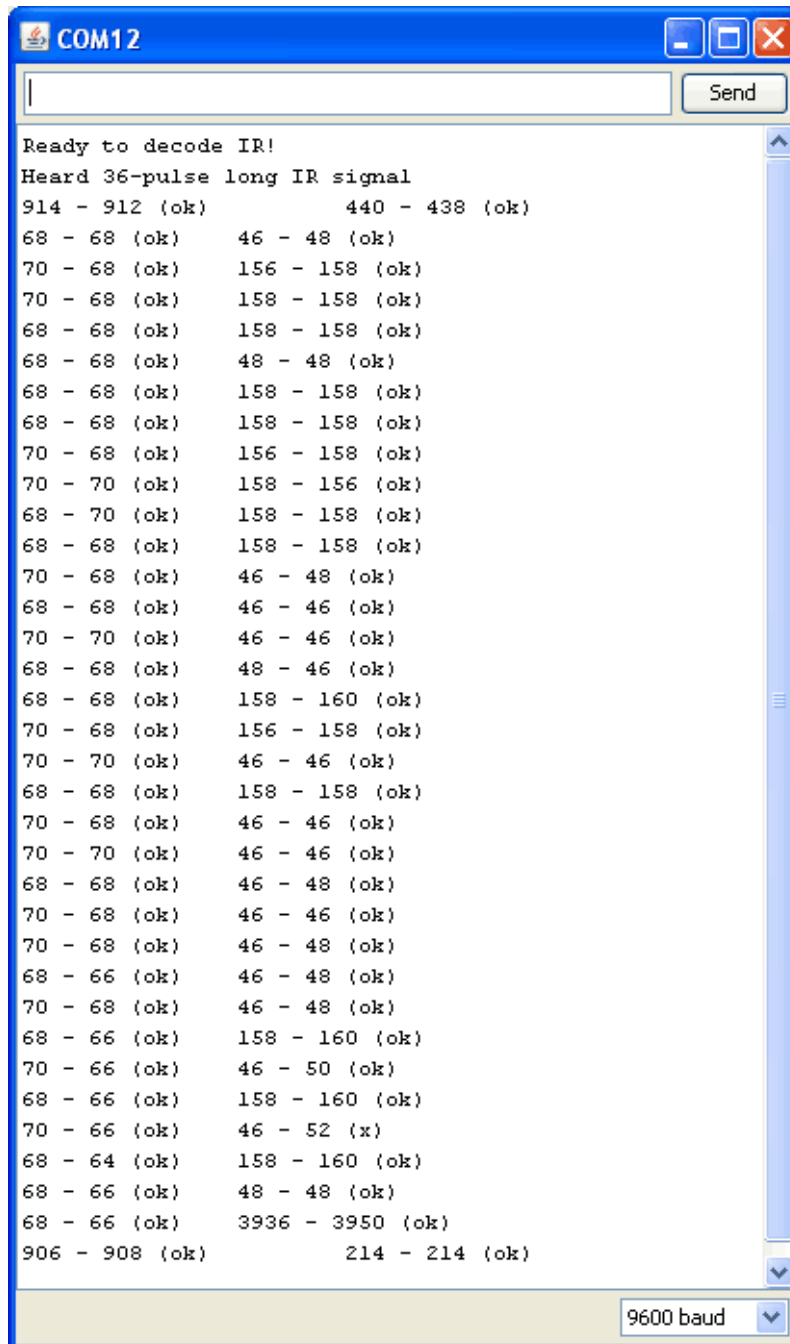
    Serial.print(oncode); // the ON signal we heard
    Serial.print(" - ");
    Serial.print(ApplePlaySignal[i*2 + 0]); // the ON signal we want

    // check to make sure the error is less than FUZZINESS percent
    if ( abs(oncode - ApplePlaySignal[i*2 + 0]) <= (oncode * FUZZINESS / 100)) {
      Serial.print(" (ok)");
    } else {
      Serial.print(" (x)");
    }
    Serial.print(" \t"); // tab

    Serial.print(offcode); // the OFF signal we heard
    Serial.print(" - ");
    Serial.print(ApplePlaySignal[i*2 + 1]); // the OFF signal we want

    if ( abs(offcode - ApplePlaySignal[i*2 + 1]) <= (offcode * FUZZINESS / 100)) {
      Serial.print(" (ok)");
    } else {
      Serial.print(" (x)");
    }

    Serial.println();
  }
}
```

A screenshot of a Windows-style terminal window titled 'COM12'. The window has a blue title bar with standard minimize, maximize, and close buttons. Below the title bar is a text input field and a 'Send' button. The main area of the window displays a list of decoded IR signal data. The text is as follows:  
Ready to decode IR!  
Heard 36-pulse long IR signal  
914 - 912 (ok)                      440 - 438 (ok)  
68 - 68 (ok)                      46 - 48 (ok)  
70 - 68 (ok)                      156 - 158 (ok)  
70 - 68 (ok)                      158 - 158 (ok)  
68 - 68 (ok)                      158 - 158 (ok)  
68 - 68 (ok)                      48 - 48 (ok)  
68 - 68 (ok)                      158 - 158 (ok)  
68 - 68 (ok)                      158 - 158 (ok)  
70 - 68 (ok)                      156 - 158 (ok)  
70 - 70 (ok)                      158 - 156 (ok)  
68 - 70 (ok)                      158 - 158 (ok)  
68 - 68 (ok)                      158 - 158 (ok)  
70 - 68 (ok)                      46 - 48 (ok)  
68 - 68 (ok)                      46 - 46 (ok)  
70 - 70 (ok)                      46 - 46 (ok)  
68 - 68 (ok)                      48 - 46 (ok)  
68 - 68 (ok)                      158 - 160 (ok)  
70 - 68 (ok)                      156 - 158 (ok)  
70 - 70 (ok)                      46 - 46 (ok)  
68 - 68 (ok)                      158 - 158 (ok)  
70 - 68 (ok)                      46 - 46 (ok)  
70 - 70 (ok)                      46 - 46 (ok)  
68 - 68 (ok)                      46 - 48 (ok)  
70 - 68 (ok)                      46 - 46 (ok)  
70 - 68 (ok)                      46 - 48 (ok)  
68 - 66 (ok)                      46 - 48 (ok)  
70 - 68 (ok)                      46 - 48 (ok)  
68 - 66 (ok)                      158 - 160 (ok)  
70 - 66 (ok)                      46 - 50 (ok)  
68 - 66 (ok)                      158 - 160 (ok)  
70 - 66 (ok)                      46 - 52 (x)  
68 - 64 (ok)                      158 - 160 (ok)  
68 - 66 (ok)                      48 - 48 (ok)  
68 - 66 (ok)                      3936 - 3950 (ok)  
906 - 908 (ok)                      214 - 214 (ok)  
At the bottom right of the window, there is a dropdown menu showing '9600 baud'.

```
COM12
Ready to decode IR!
Heard 36-pulse long IR signal
914 - 912 (ok)          440 - 438 (ok)
68 - 68 (ok)           46 - 48 (ok)
70 - 68 (ok)           156 - 158 (ok)
70 - 68 (ok)           158 - 158 (ok)
68 - 68 (ok)           158 - 158 (ok)
68 - 68 (ok)           48 - 48 (ok)
68 - 68 (ok)           158 - 158 (ok)
68 - 68 (ok)           158 - 158 (ok)
70 - 68 (ok)           156 - 158 (ok)
70 - 70 (ok)           158 - 156 (ok)
68 - 70 (ok)           158 - 158 (ok)
68 - 68 (ok)           158 - 158 (ok)
70 - 68 (ok)           46 - 48 (ok)
68 - 68 (ok)           46 - 46 (ok)
70 - 70 (ok)           46 - 46 (ok)
68 - 68 (ok)           48 - 46 (ok)
68 - 68 (ok)           158 - 160 (ok)
70 - 68 (ok)           156 - 158 (ok)
70 - 70 (ok)           46 - 46 (ok)
68 - 68 (ok)           158 - 158 (ok)
70 - 68 (ok)           46 - 46 (ok)
70 - 70 (ok)           46 - 46 (ok)
68 - 68 (ok)           46 - 48 (ok)
70 - 68 (ok)           46 - 46 (ok)
70 - 68 (ok)           46 - 48 (ok)
68 - 66 (ok)           46 - 48 (ok)
70 - 68 (ok)           46 - 48 (ok)
68 - 66 (ok)           158 - 160 (ok)
70 - 66 (ok)           46 - 50 (ok)
68 - 66 (ok)           158 - 160 (ok)
70 - 66 (ok)           46 - 52 (x)
68 - 64 (ok)           158 - 160 (ok)
68 - 66 (ok)           48 - 48 (ok)
68 - 66 (ok)           3936 - 3950 (ok)
906 - 908 (ok)         214 - 214 (ok)
9600 baud
```

This loop, as it goes through each pulse, does a little math. It compares the absolute (**abs()**) difference between the code we heard and the code we're trying to match `abs(ocode - ApplePlaySignal[i*2 + 0])` and then makes sure that the error is less than FUZZINESS percent of the code length (`ocode * FUZZINESS / 100`)

We found we had to tweak the stored values a little to make them match up 100% each time. IR is not a precision-timed protocol so having to make the FUZZINESS 20% or more is not a bad thing

Finally, we can turn the **loop()** into its own function which will return **true** or **false** depending on whether it matched the code we ask it to. We also commented out the printing functions

```

boolean IRcompare(int numpulses, int Signal[]) {

  for (int i=0; i< numpulses-1; i++) {
    int oncode = pulses[i][1] * RESOLUTION / 10;
    int offcode = pulses[i+1][0] * RESOLUTION / 10;

    /*
    Serial.print(oncode); // the ON signal we heard
    Serial.print(" - ");
    Serial.print(Signal[i*2 + 0]); // the ON signal we want
    */

    // check to make sure the error is less than FUZZINESS percent
    if ( abs(oncode - Signal[i*2 + 0]) <= (Signal[i*2 + 0] * FUZZINESS / 100)) {
      //Serial.print(" (ok)");
    } else {
      //Serial.print(" (x)");
      // we didn't match perfectly, return a false match
      return false;
    }

    /*
    Serial.print(" \t"); // tab
    Serial.print(offcode); // the OFF signal we heard
    Serial.print(" - ");
    Serial.print(Signal[i*2 + 1]); // the OFF signal we want
    */

    if ( abs(offcode - Signal[i*2 + 1]) <= (Signal[i*2 + 1] * FUZZINESS / 100)) {
      //Serial.print(" (ok)");
    } else {
      //Serial.print(" (x)");
      // we didn't match perfectly, return a false match
      return false;
    }

    //Serial.println();
  }
  // Everything matched!
  return true;
}

```

We then took more IR command data for the 'rewind' and 'fastforward' buttons and put all the code array data into **ircodes.h** to keep the main sketch from being too long and unreadable ([you can get all the code from github](http://adafru.it/aKg)) (<http://adafru.it/aKg>)

Finally, the main loop looks like this:

```

void loop(void) {
  int numberpulses;

  numberpulses = listenForIR();

```

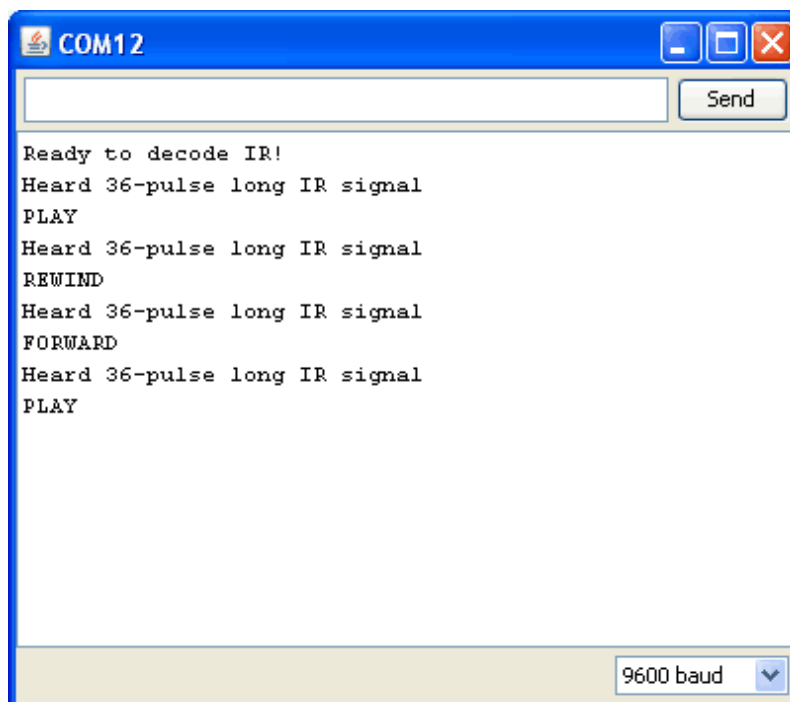
```

Serial.print("Heard ");
Serial.print(numberpulses);
Serial.println("-pulse long IR signal");
if (IRcompare(numberpulses, ApplePlaySignal)) {
  Serial.println("PLAY");
}
if (IRcompare(numberpulses, AppleRewindSignal)) {
  Serial.println("REWIND");
}
if (IRcompare(numberpulses, AppleForwardSignal)) {
  Serial.println("FORWARD");
}
}
}

```

We check against all the codes we know about and print out whenever we get a match. You could now take this code and turn it into something else, like a robot that moves depending on what button is pressed.

After testing, success!



## Buy an IR Sensor

---

Buy an IR Sensor (<http://adafru.it/157>)