A rambling tutorial on sending messages by visible light and IR

By JimboZA on the Arduino Forum

Revision history

Date	Version
26 December 2014	Draft: Part 1, 2 and 3 uploaded to forum
27 December 2014	Draft: part 4 added
28 December 2014	Draft complete: part 5 and 6 added

Who am I and what is this tutorial?

I'm an Arduino hobbyist, simple as that. I'm not an electronics guy- my undergrad was in civil engineering, although I did a post-grad diploma in electrical. But I'm by no means a guru on electronics, which leads me to the disclaimer: Your Mileage May Vary.

The Arduino Forum has a number of threads on the use of things like Light Dependent Resistors (LDRs) and TV IR remotes and so on, and I found it an interesting topic to look into. While messing around with some of that stuff I thought it would benefit the community if I collected some thoughts and wrote down some steps I followed in my quest to get some understanding. It's a bit of a ramble, rather than a properly structured "training course" or "learning intervention"; I write that stuff for a living, and today I'm off-duty. I'll correct any glaring errors as you and I find them, but this will (probably) never be anything more than a bit of a ramble.

Who are you and why should you read this?

You are likely someone who dabbles in Arduino, and might have a bit of confusion about terms like "modulation", wondering how on Earth an infrared message from your TV remote gets sent at 38kHz when, surely, infrared light has a frequency up in the GHz and THz. How can that be?

You might get a bit of understanding out of reading this.

What do you need?

Well on one level you need your eyes, is all: just read it and it should make some sense.

On another level you'll get more out if you have an assortment of components such as an IR LED and IR photodiode, an IR receiver like a TSOP34838, your TV remote and of course the normal Arduino related stuff like resistors and bits of wire. I use 2x 555 timers for creating signals at particular frequencies, and you might have a couple lying around. They're cheap, so maybe pick up a couple next time you're buying, but I'm sure you could investigate other ways of making signals. All part of the fun…

If you have, or can borrow, an oscilloscope that would be a real bonus. I've included screen shots from mine, so it's not a big deal if you don't have one. If your meter (and, surely, you have one of those?) has a frequency scale that would help. Oh, dust off your Arduino……

What's in here?

There are 6 parts to this:

- Visible light and the LDR
- Plain IR
- 555 to make an LED blink
- Modulated IR
- Sending IR carrier from Arduino
- How sensitive is a 38kHz TSOP detector to the actual IR frequency?

Please remember as you read this that my intention is not to win prizes for the world's best structured learning experience and I make no apologies if it seems a bit haphazard. I've included links to other sources where things might be elaborated or explained in different words.

Reading TV remotes
If you want to work with incoming signals from TV remotes, there's no better place to find out about that than Ken Shirriff's blog, where you can download his library. You'll be able to de-code your remote so you'll know what code is produced by each button. Then in your sketch it's easy to control your robot with an "if" looking for a particular incoming code, and if the "if" is true, do whatever you need to move forward or turn or beep or blink or whatever.

Lets' start then….

Part 1: Visible light and the LDR

When we were kids we all read stories about pirates and smugglers, so the use of visible light to signal "the coast is clear" (or not) is known to us. In an electronic world we can use the presence or absence of light to control things.

The light dependent resistor is probably the simplest device. As its name implies the resistance changes with the presence or absence of visible light, so it makes sense to use it in the same way as any variable resistor.

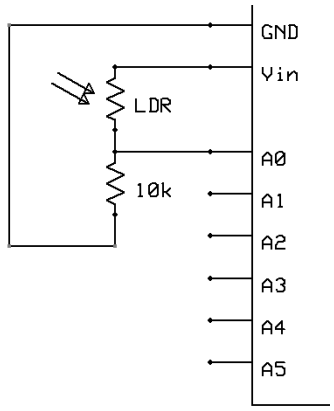
The photo below shows the resistance of an LDR in the ambient light at my desk on a dull day with my desk lamp on. Apart from that reading of 14kΩ, I got about 100kΩ with my hand over it, and under 1kΩ when I shone an LED torch onto it. The resistance decreases with the amount of light falling on it.

As adafruit say in their LDR tutorial, these are not precision devices and they exhibit variation between one LDR and the next.

A test circuit and code are on the next page. This is an analog setup, with a range of outputs from 0-1023. You can digitalise this of course by using some suitable thresholds, with (say) under 300 being on, over 700 being off and anything in between undefined.

The circuit below shows the junction between an LDR and a 10k resistor connected to pin A0 on a Uno. The code reads the analog value and displays that value in the serial monitor. I got analog reads of about 100 when covered, 400 in ambient, and 900 with an led torch.

```
// read an ldr and display value in serial monitor
int LDRpin =0;
void setup(){
  Serial.begin(9600);
  Serial.println("starting....");
}
void loop(){
  Serial.println(analogRead(LDRpin));
delay(1000);
}
```



Can we reconcile those readings with the resistances already mentioned?

Let's have look at the numbers for the ambient case, by looking at the maths associated with a potential divider, as explained here in Wikipedia.

Using the equation from Wikipedia, we have Z1 = 14k and Z2 =10, with Vin as 5V. Plug those numbers in and we get Vout = 5 (10/24) = 2V. For an Arduino analog pin which goes from 0-1023, we should get an analog reading 2/5 of 1023 = 410. I was getting around 400, close enough for me…

(Just for fun, I measured the actual voltage at the junction as 2.1, so all the numbers gel.)

I won't say anything more about LDRs: there use is limited and I just included them for completeness, and so we'll move on to plain infrared.

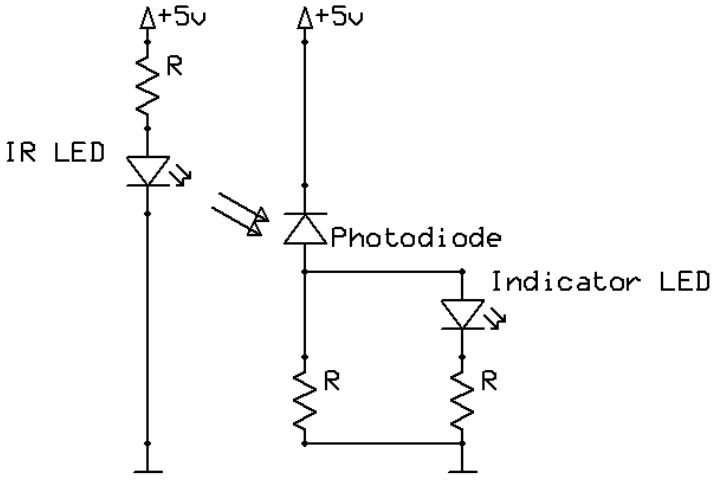
Part 2: Plain infrared

Infrared's easy to generate and read: you just need to get an IR LED to send and an IR photodiode to read and you're in business. The photo below shows one of each; the LED is the grey/blue one on the right. IR is not visible to the human eye so a trick to see if the LED is on is to view it through a digital camera, as shown below, right. Remember that an IR LED is an LED like any other and needs a series resistor to limit current. The cathode on both is denoted by the flat side on the plastic.

So let's see how to do some sending and reading with these guys.

The LED's simple, just a series resistor and power. This page shows how to connect the receiver in a divider. Note the diode is reverse biased with the cathode to positive; I used a 1k pulldown. My circuit is shown below. No Arduino here yet, let's just get some volts out of the receiver.

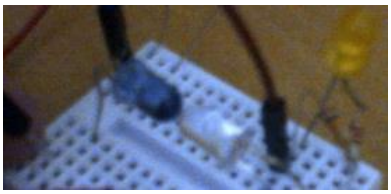
We want to get a voltage at the junction of the photodiode and the pull down. In the absence of IR, the diode will be closed and the junction will be pulled down to 0. In the presence of IR, the diode will conduct and the junction will pull up to 5V. So if we put a normal LED across the junction and ground, it should work in concert with the IR LED: on together when the junction is at 5V or off together when the junction is at 0V.

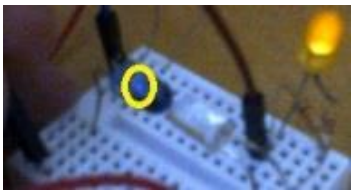


Note that the grounds for the transmitting and receiving sides need not be connected: the two sides need have no electrical knowledge of each other since it is merely the IR light which activates the photodiode.

Although they use phototransistors not photodiodes, the above is essentially the circuit inside an opto-isolator such as this or an interrupter like this.

These photos show the indicator LED is off when the IR LED is unplugged (left) and on when it's plugged in (right). You can vaguely see the IR LED is on in the right hand photo, circled. Note how I have the two components nose-to-nose so the IR beam finds its way to the diode.

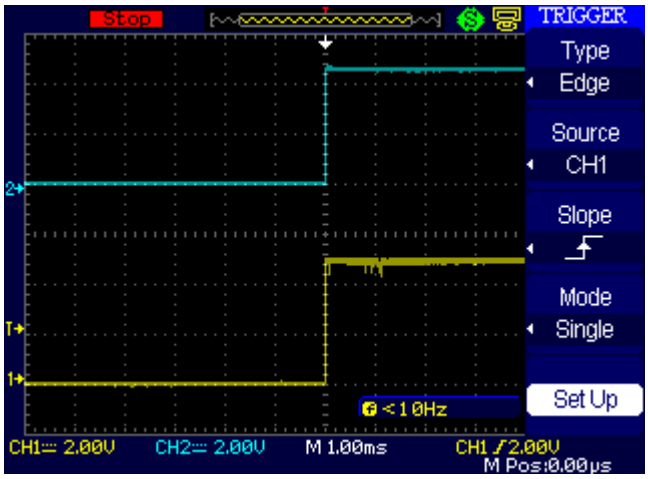




You could turn the indicator LED off by blocking the gap between the IR LED and the photodiode.

So, how to Arduinise this? Very simple: if we take the junction of the photodiode and its resistor to a digital pin, and our ground to the Arduino's ground, we should be able to read the state of the photoresistor, which in turn means we're reading the state of the IR LED which is totally isolated. (We'll be putting the Arduino in where our indicator led is.)

First let's double check the input and output with an oscilloscope, with Channel1 (yellow) on the input (the IR LED) and Channel2 (blue) on the output from the photodiode (where the indicator LED is).



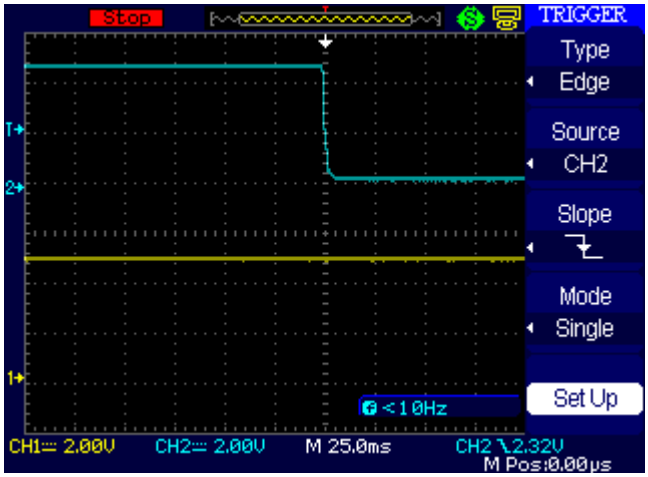
As you can see from the TRIGGER menu at the right hand side, this single shot was triggered on the rising edge of CH1, and it's clear that CH2 followed immediately when I connected the IR LED. (Both go from 0V to about 5V; the big blocks are 2V each as shown at the bottom of the screen.)

Just for kicks…. What do you think would happen if we set the 'scope to trigger on a falling edge on CH2 and then blocked the IR beam?

Pause for a thought before turning the page…..

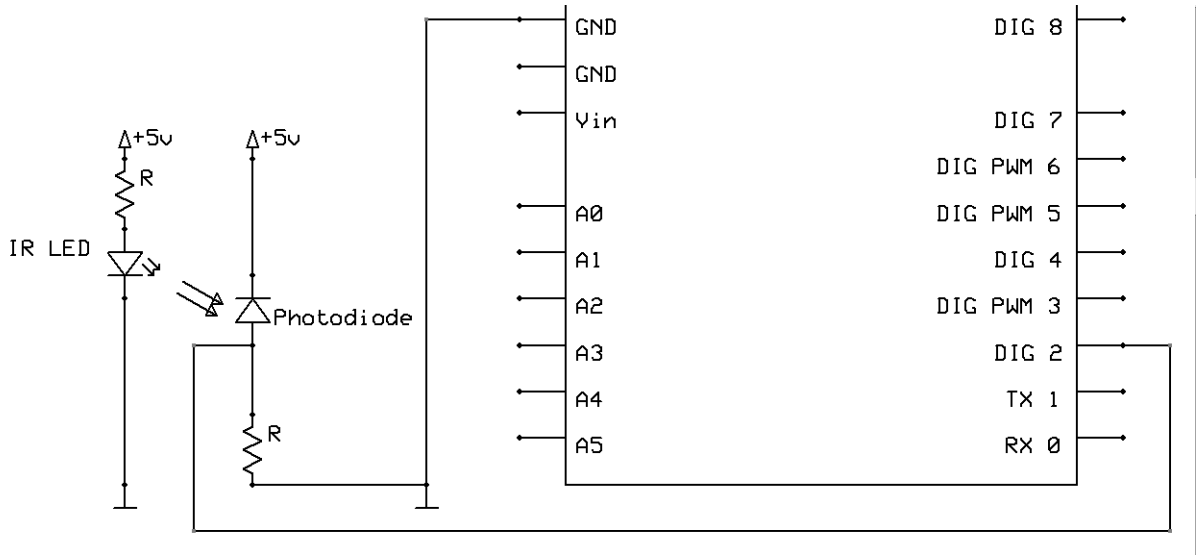
Since we're only blocking the beam, CH1 will obviously stay on, but since the photodiode's light source is being taken away by the blockage, CH2 should go low.

The 'scope trace shows that's exactly what happens: the TRIGGER menu shows we're doing a single shot on a fall in CH2. CH1 is horizontal at 5V, CH2 dies. (The arrows at the left of the screen labelled 1 and 2 are 0V for each channel; the blue T means the shot will trigger when the blue line passed through about 2V.)



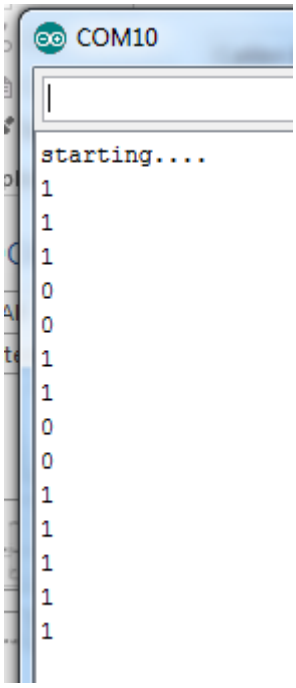
So let's take this to the Arduino and just simply read the state of the photodiode. I'm not going to do anything fancy here, let's just verify in the serial monitor that the Arduino sketch sees the pin going up and down as I unplug/plug the IR LED.

Here's the circuit: same as before but I took the indicator LED out and hooked that point to a digital pin; grounds are connected.



The serial monitor shows the pin started at 1. The first batch of 0's were when I unplugged the IR LED, and it went back to 1 as I plugged it back. The second batch of 0's are where I blocked the beam, and it went back to 1 when the beam was restored.

The code and the serial monitor output are on the next page.




```
// read a photodiode and display value in serial monitor
int phDiodePin =2;
void setup(){
  Serial.begin(9600);
  Serial.println("starting....");
}
void loop(){
  Serial.println(digitalRead(phDiodePin));
delay(5000);
}
```

Ok so far so good I hope?

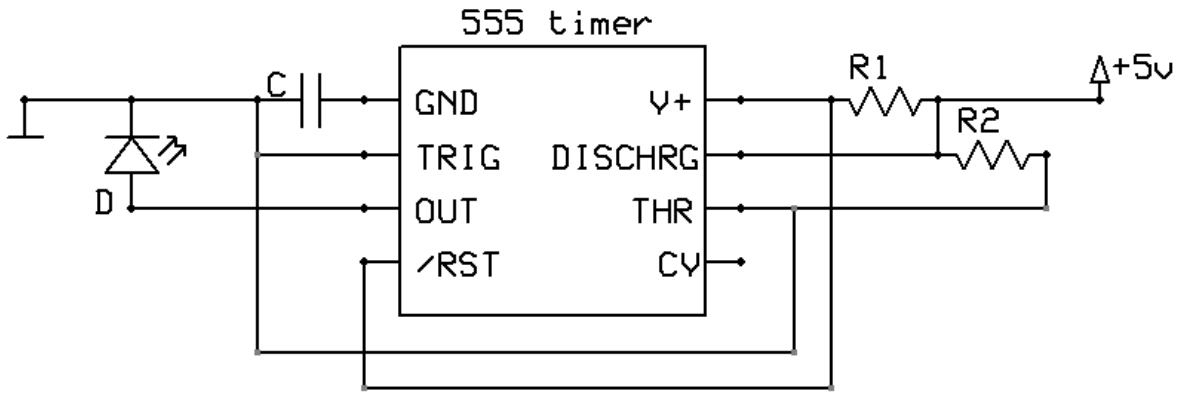
Just for a change of pace, and since I found it interesting, we'll pause to look at getting the IR LED to blink on and off. We can surely do this in software inside an Arduino (see Part 5, in fact), but just for the hell of it we'll do it with a 555 timer chip. I'm pretending my Arduino is the recipient of an IR signal from somewhere beyond my control.

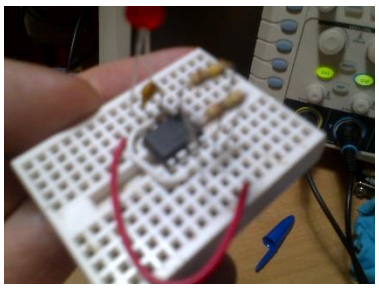
Part 3: Using a 555 to make an LED blink

I'm no expert on the 555 chip; ~~most~~ all of what I know comes from The Electronics Club. In this section we'll see it simply blink an LED off and on, pretending that the blink pattern is the message we want to send to someone. I started with a normal LED so I could see it, then swapped that for the IR one so the photodiode could see.

I'm not going to give any explanation of the 555. You may be intimate with it already, if not there are a zillion sources of info, not least being the link in the previous paragraph.

The circuit shown there does not show the 555 pins in the correct places, so here's a schematic from which it's dead simple to breadboard your 555 blinker. The photo shows my breadboard setup, power connections omitted, but it's ground to pin 1 top left, and V+ to pin 8 top right.

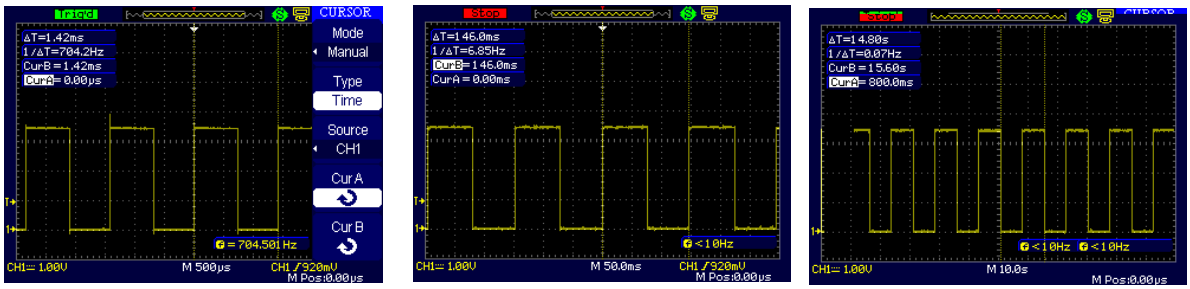




The essence of the 555's blinkability is the combination of the resistors and capacitor to influence the timing. Below I have extracted the table from that linked page, showing the timings for various combinations of C, R1 and R2.

555 astable frequencies			
C1	R2 = 10kΩ R1 = 1kΩ	R2 = 100kΩ R1 = 10kΩ	R2 = 1MΩ R1 = 100kΩ
0.001µF	68kHz	6.8kHz	680Hz
0.01µF	6.8kHz	680Hz	68Hz
0.1µF	680Hz	68Hz	6.8Hz
1µF	68Hz	6.8Hz	0.68Hz
10µF	6.8Hz	0.68Hz (41 per min.)	0.068Hz (4 per min.)

I decided to use the R2= 1MΩ and R1= 100kΩ column, and the 'scope output shows the results with the 3 capacitors highlighted, 0.001uF (aka 102), 0.1uF (aka 104) and 10uF. The actual frequencies are shown in the blue boxes at the top left of each shot as 704Hz, 6.85Hz and 0.07Hz.

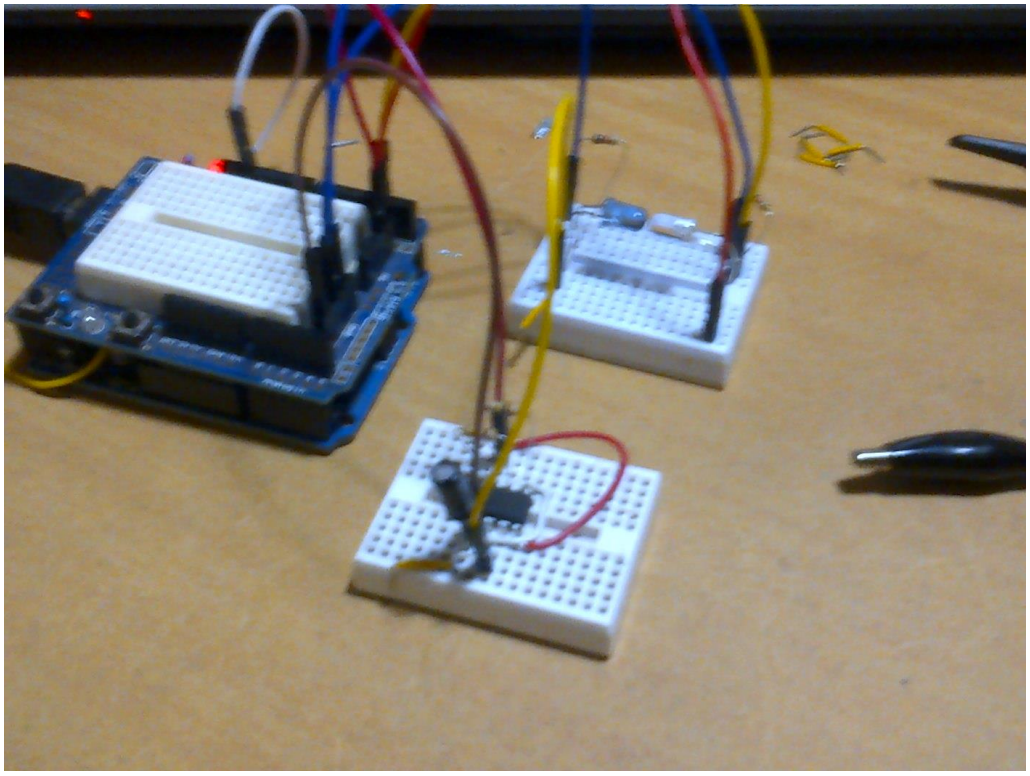


How can we use these blinks in Arduino. Well that depends what you want to do I guess. Let's revisit Part 2.

Part 2 Revisited: More plain infrared

If the blinking IR LED is a signal or message of some kind, we would obviously want to do something with that message in the Arduino. It struck me that a blinking LED raises and lowers a pin just as a mechanical button does, so I thought to try the standard State Change Detection example which you can find in the Arduino IDE in File > Examples > Digital. The sketch reads a digital pin, and every 4th time the pin goes high, it turns an LED on. I won't reproduce it here, but I ran it with the photodiode's output to Arduino pin 2 as before, replacing the sketch's actual button. This time though, instead of manually plugging / unplugging the IR LED, I fed the output from the 555 to the IR LED so the 555 was electronically blinking the IR LED. Thus, the photodiode is essentially responding to the 555 through the medium of IR. An LED on my Arduino protoshield is connected to pin 13; it's under the control of the sketch, and therefore controlled by the 555's output. I think that's ~~quite~~ very cool.

I won't reproduce the circuit here- you've seen the pieces already, but here's a photo instead. I made a simple change to that standard program, just to print millis() each time the pin changed state. We'll use that in a moment to verify the speed matches that of the 555 so you know I'm not making this all up.



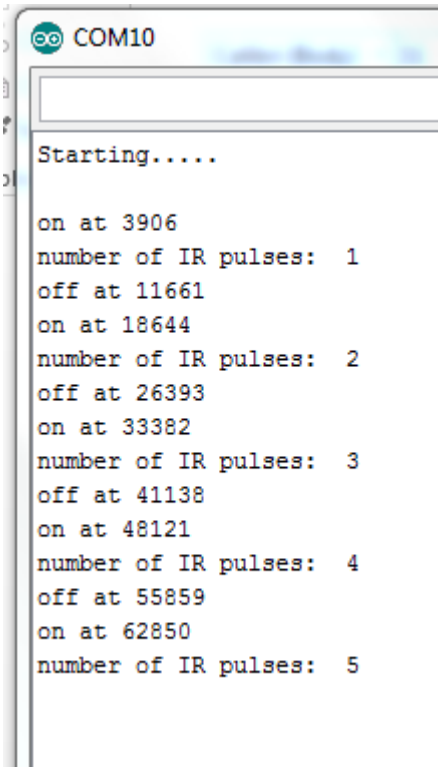
The mini-breadboard at the front is the 555 with the 10uF cap and there's a yellow wire from its output to the rear mini-breadboard where it feeds the IR LED. The IR LED is nose-to-nose with the photodiode as always, and the other yellow wire back right, feeds pin 2 on the Uno. The white wire back left on the Uno is from pin 13 to the red LED.

So to recap:

- 555 produces a 0.068Hz (14 second, 7 second each on / off) signal on its output
- That feeds the IR LED which blinks at that rate
- The photodiode responds to the IR and it in turn feeds Uno pin 2
- Uno sketch reads pin 2
- Every 4th on of the pin, pin13 turns its LED on.

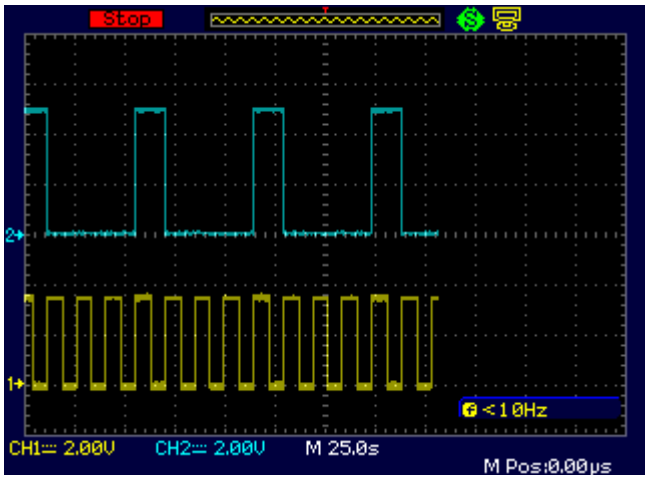
Probably the most complicated way to turn on a Uno's pin 13 LED?

Here's some serial print output:



Lets' check the numbers. Ignore the first pulse since we don't know where the 555 was in its cycle when we powered up the sketch. But after that we see an on at 18644 ms and again at 33382. That means the cycle was 33382-18644 = 14738ms or 14.738s. The inverse of that is 0.068 which is the frequency of the 555 with the 10uF cap so all is well.

All that remains is to double check that with the ‘scope. With Channel 1 (yellow) to the 555’s output and Channel 2 (blue) to Uno pin 13, we indeed see that pin 13 goes high every 4 cycles.



Remember, I just decided to use that standard sketch for simplicity since it was there. With the photodiode driving Uno pin2, you can do whatever is necessary to interpret your IR LED’s messages.

That’s the end of this part about simple plain old IR transmission and collection at the Arduino. In the next section we’ll look at the more complicated case of 38kHz modulation.

Part 4: Modulated infrared.

We've seen that it's possible to use an IR LED to switch an IR photodiode: the diode's output followed the LED's input as high for high and low for low. I used a 555 timer to control the LED, but before that I simply plugged the LED in and out to turn it on and off. However it gets done, the point is that we sent our message (the ons and offs of the IR LED) and interpreted that message on receipt at the Arduino. In the example we used existing button code to read the pin and turn a normal LED on every 4th high.

If we are actually successful- and so far we are, what's all this about modulation then?

First, what is modulation? When you listen to the radio, you hear voices and music. Yet those voices and music don't come at you from the air, they come out of the radio. What you hear coming out (somewhere in the 20Hz to 20kHz range) did not get sent through the air at that frequency. If it did we'd all hear all the stations all the time, assuming it were possible.

When you tune in your radio, you don't tune to human frequencies. My favourite station ClassicFM is on the dial at 102.7MHz, way above the human ear down in the low Hz range. The presenter speaks in the studio at 20-20kHz, it gets broadcast at 102.7MHz, and my radio plays his voice to me at 20-20k in the car. That's modulation in action: the 102.7MHz signal is known as the carrier, presumably since it carries the message, and the message is said to modulate the carrier. The message is "captured" in the carrier wave at the transmitter, and "released" by the radio.

That capturing and releasing is the modulation and demodulation. It's where the word modem comes from in data transmission where digital data is carried on an analog voice line. The sending device MO-dulates the voice line carrier with the data, and the other end DEM-odulates the signal and gives you the data.

So, do we *have* to use modulation in sending IR? We just saw that we don't. We were successful in sending a message from our 555 to the Arduino by flashing the IR LED according to our message. Problem is though: what happens if there's lots of other IR around? There's ambient IR around all the time, and there may be other experimenters flicking their IR LEDs off and on too. There's too much risk of some stray signal being read by your TV, which might change channels just because the sun came out and caused a spurious signal.

So secondly, what's modulation in the IR world?

There's a specific variety of modulation called Amplitude-shift Keying (ASK), where the modulation is simply to turn the carrier on or off. So it's a bit like what we already did, by turning the LED on or off, but instead of being "solid on", what looks like on is in fact going off and on at a frequency much much higher than that at which we actually require for our message. In the IR world it's common for the carrier to be at 38kHz, and an "on" as far as we're concerned is actually a 38kHz off-on repeating burst sequence of the carrier wave. An "off" is a genuine off though, ie the absence of carrier.

So our TV is less likely to pick up spurious signals, since it's unlikely that opening the curtains will cause the sun to send a 38kHz carrier burst to the TV. It means, of course, that some gubbins inside the TV has to be on the look out for that 38kHz carrier.

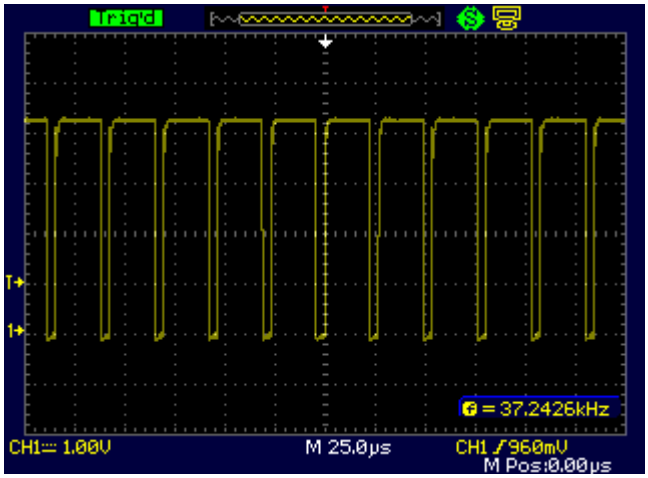
And indeed, such gubbins exists in the form of, for instance, the TSOP348XX series of demodulators which are available to demodulate various carrier frequencies including 38kHz.

So, let's make a carrier wave and see if we can recognise it with a TSOP….

I used a 555 to make a carrier. It is, after all, no different from the method used to make our actual signal earlier: it's just a matter of choosing C, R1 and R2 to generate a 38kHz signal rather than the ones we made before.

Model railway enthusiast Dave Bodnar uses 38kHz IR in his outdoor rail setup, saying on his web page that it is "much less prone to interference from sunlight and other sources of infrared radiation". He also used a 555, with one of the resistors being variable to set the frequency correctly. I homed in on a value of 4k7 for that resistor, and used his values of 22k for the other and 0.001uF for the cap.

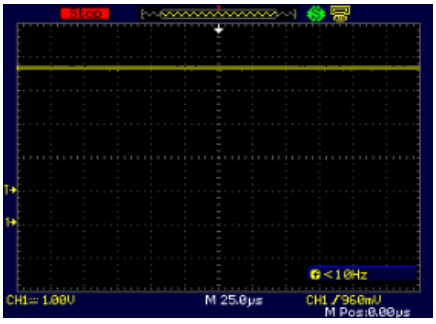
Here's a 'scope trace of the output from the 555, which creates a wave of 37.2kHz (bottom right of shot). It's very asymmetric with those values, on-time is about 85%.



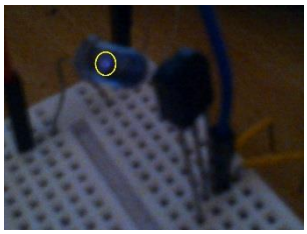
Introducing the TSOP: let's see what the TSOP makes of my (almost) 38kHz very asymmetric carrier.

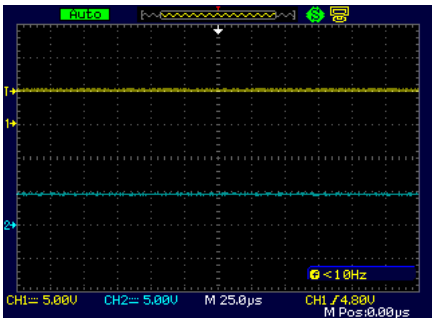
Have a look at the datasheet linked earlier, and you'll see the TSOP simply has 3 connections: power, ground and output. The output varies according to the presence or absence of the 38kHz wave, being a digital 0 or 1. The datasheet points out it's "active low" which means it outputs a 1 in the absence of carrier, and when activated by the carrier it goes low, to 0. You need to remember that and handle that (possibly counter-intuitive) logic in your sketch.

First, let's apply power to the TSOP and see what its output is in ambient light. The photo below shows 5V applied to the right pin, with ground in the centre. The scope trace shows that, as expected, the TSOP outputs 5V under ambient light, since it is active low.



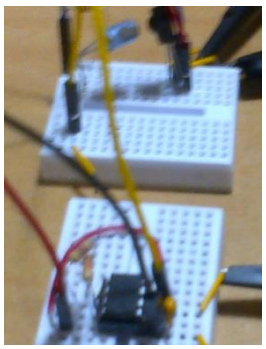
Next let's shine our IR LED on it and see if it reacts. First we'll just apply 5V to the LED so it's solid on. The photo shows the LED is glowing (circled yellow). The 'scope traces show that although the LED is solid on (Channel 2 (blue) at 5V), that's having no effect on the output from the TSOP (Channel 1 (yellow) inactive at 5V)

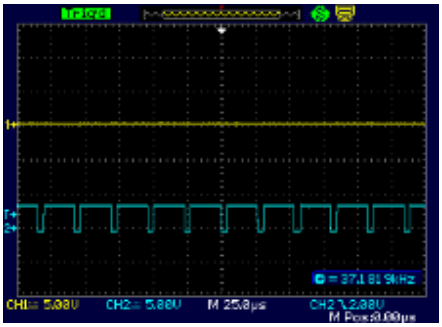




So far that's what we expected: no reaction from the TSOP in either ambient light or under a solid IR beam.

Let's see if it reacts to our 38kHz output from the 555 to the IR LED shown left. The trace shows that under the influence of the 38kHz carrier (blue) the TSOP indeed goes active to 0V (yellow). **YAY!**





The huge skew in our carrier (85% on) doesn't seem to worry the TSOP, nor does the fact that the frequency is closer to 37kHz than to 38. It's working as advertised.

So what next? Well first a quick recap:

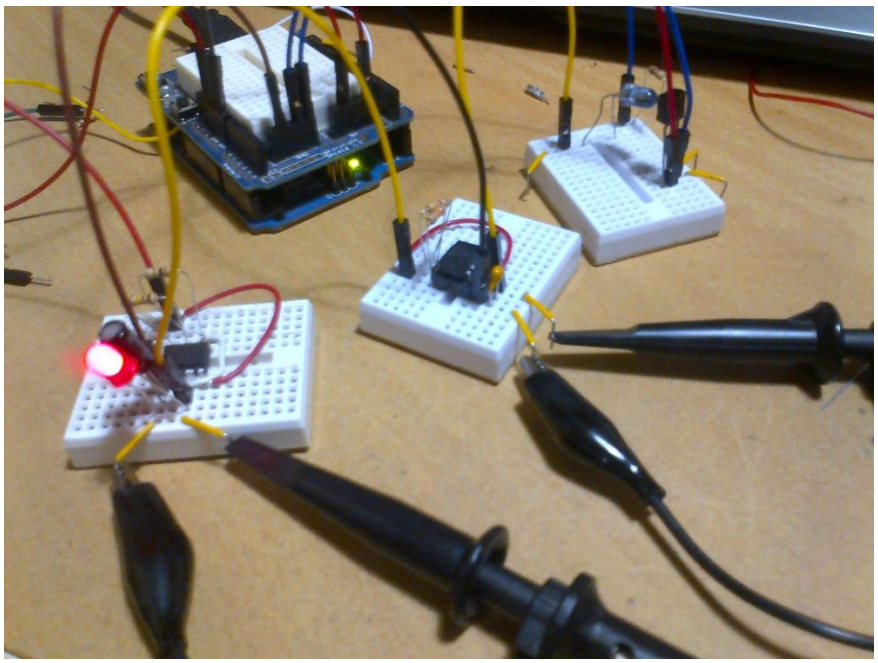
- The TSOP is inactive in ambient light: outputs 5V
- It's inactive under a solid IR beam: still outputs 5V
- It's active under a 38kHz carrier: drops to 0V

But we still haven't modulated the carrier. Remember that modulation is the superimposition of our actual message on the carrier. In the case of ASK, the carrier (which itself will always be at 38kHz) is present or absent. So let's dig out the 555 we used earlier, the one that outputs the solid on and off, and use its output in turn, to switch the other 555 off and on.

The 555 has a reset pin, so I'm thinking to wire the output of the first 555 (ie the solid on or off) to the reset of the second one (the one that produces the carrier). If that works, the solid on from the first 555 will switch on the second 555, thus producing a 38kHz signal to the LED and the TSOP; an off from the first one will produce an off from the second one. We will thus have modulated the carrier with our original signal.

Let's try that…..

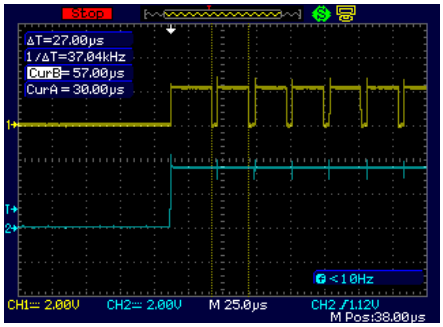
The photo below shows the setup. The left side breadboard is the 555 creating the message: a solid on/off sequence at 0.068Hz (T=14s) as before. Its output goes to the reset of the 555 on the middle breadboard. In theory the output from there should be modulated by the original message (since the reset will be switching the second 555 off and on), and pulse the IR LED on the third breadboard accordingly. The TSOP should then output solid on and off once it demodulates that signal. Arduino for breadboard power only, at this stage.

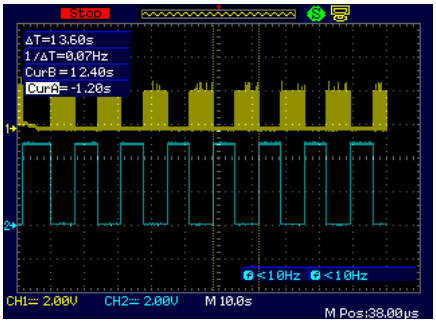


Unfortunately I have a 'scope with only 2 channels so we'll have to test this in 2 steps: first to show the raw signal being modulated and then that modulated signal being demodulated.

The left hand screen shot below shows that when the first 555 turns its LED on, the blue trace goes high. Because the first 555 output switches the second one, the 38kHz (actually 37.04) fires up as shown by the yellow trace. That signal is, remember, feeding the IR LED.

In the right hand screen I've left the yellow trace connected to the modulated output, and now the blue trace shows the output from the TSOP. I've changed the horizontal time scale by some orders of magnitude. The upper (yellow) scale is still the modulated output from the second 555, and each yellow "block" is a 7 second on of the first 555's output. Since the TSOP is active low, the blue trace goes low for each 7 second yellow "burst".



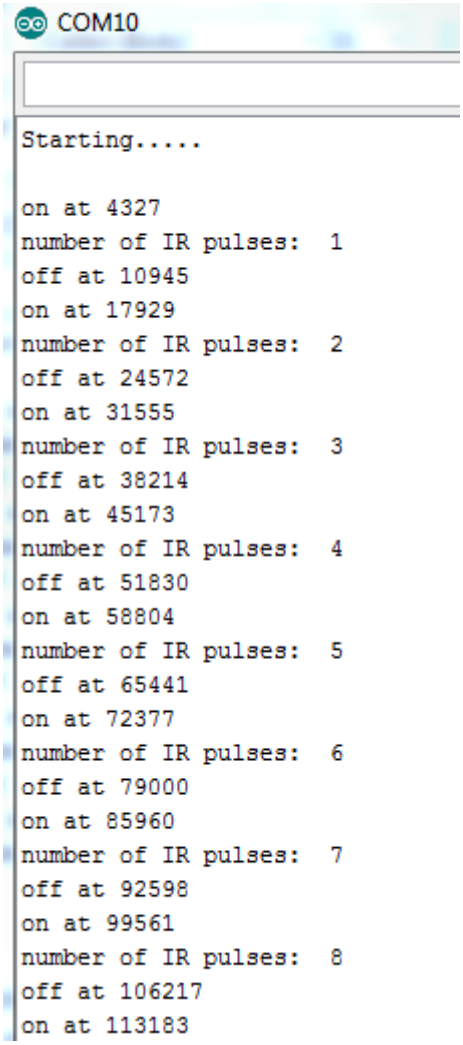


So all is working to plan. Final recap:

- First 555 creates our "message", a 7 second on 7 second off flash.
- Second 555 creates our 38kHz carrier
- The first 555's output switches the second by feeding its reset
- Thus the second 555 produces a 38kHz modulated version of the first 555's 14 second on/off cycle
- That modulated signal feeds the IR LED
- The TSOP demodulates the IR LED's signal and produces an inverted version of the first 555's on / off as off / on.

There's one more thing to do….. let's feed the TSOP's output into the Arduino sketch that we used earlier, and see what happens with the LED that it switched on every 4 cycles.

Here's the serial print out, exactly as it was previously. As I watched the serial print though, I noticed the ons and offs where out of phase with the LED on the first 555: that makes sense, since the active low TSOP inverted the signal. That could be fixed either by inverting the TSOP output in hardware, or inverting the logic in the sketch.



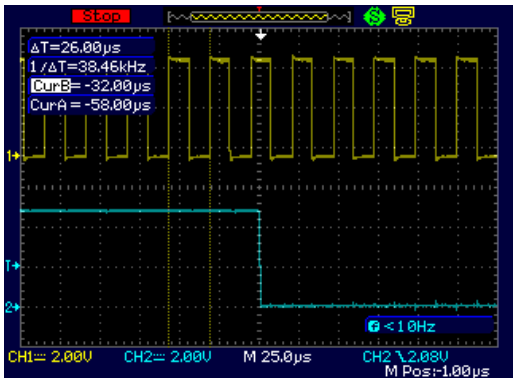
Part 5: Creating a 38kHz carrier in Arduino

Lastly let's see if it's feasible to make a 38kHz carrier in an Arduino sketch. Of course it's possible: every Arduino sketch on the planet turns pins on and off, so it's just a matter of getting the timing right, right?

First let's do it the long way, using the sketch below which is based on the pulseIR() function in the sketch here at adafruit.

```
// to get a 38kHz square wave

byte wavePin = 2;

void setup()

{

  pinMode(wavePin, OUTPUT);

}

void loop()

{

// 38 kHz is 26u

// total 4+9+4+9 = 26

   digitalWrite(wavePin, HIGH);  // this takes about 4us

   delayMicroseconds(9);

   digitalWrite(wavePin, LOW);

   delayMicroseconds(9);

}
```

With the IR LED hooked up to pin 2, and pointing at the TSOP, I get the following 'scope trace with the yellow being the IR LED and the blue from the TSOP. Seems the TSOP takes a few cycles of the carrier to latch on to the fact that there's work to be done before it drops to 0V. The frequency is 38.46kHz by the way.



A simpler way is to use tone() as shown below

```
// to get a square wave, now using tone

byte wavePin = 2;

void setup()

{

tone(wavePin, 38000);

}

void loop()

{

}
```

That gives similar results as above.

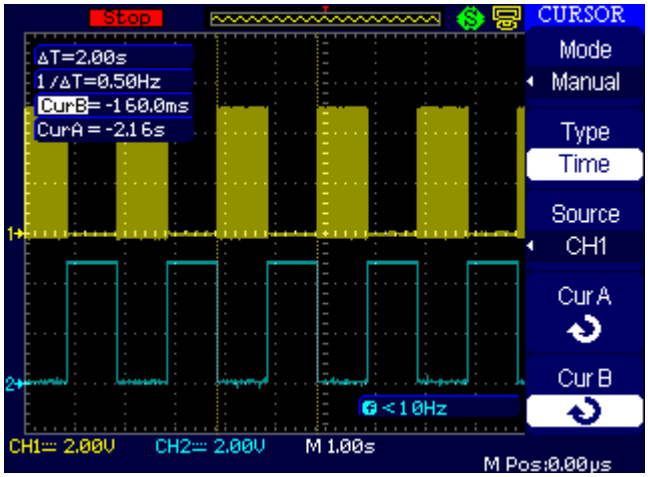
To see how to use the Arduino to modulate a carrier for a specific purpose, in their case a Nikon camera, have a look at the adafruit code linked earlier.

But let's do a simple modulation. Using tone() we should be able to provide a modulated signal using the well known (and hated, but simple) Blink WITH Delay sketch, in the IDE at File > Example > Basics. Instead of turning the LED on, we'll turn a 38kHz tone off and on. If we put an indicator LED across the TSOP output and ground, we should then be able to blink that by IR.

Here's the code:

```
int IRled = 2;

void setup() {

pinMode(IRled, OUTPUT);

}

void loop() {

  tone(IRled, 38000);

  delay(1000);

  noTone(IRled);

  delay(1000);

}
```

Here's the 'scope trace of that, showing the 38kHz on for a second (the yellow blocks) and off for a second. The TSOP being active low, goes low when the 38kHz is transmitting, and high when it's not. The on for a second, off for a second "blink" is a frequency of 0.5Hz as shown in the blue box at top left.

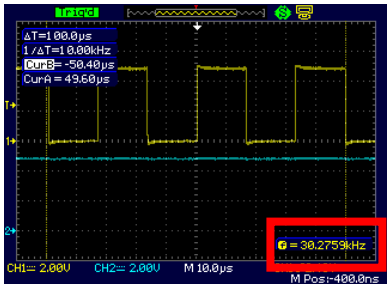


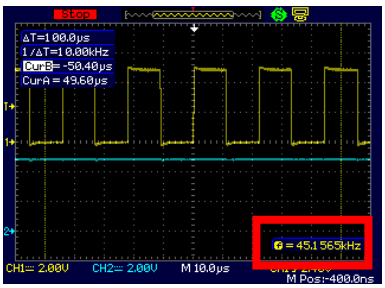
Part 6: Sensitivity of the TSOP to the input frequency

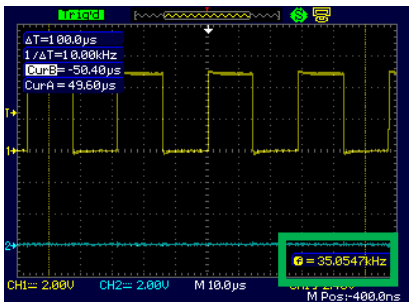
While doing that experiment with tone(), it struck me that it's really easy to change the carrier frequency and see what frequencies my 38kHz TSOP actually responded to.

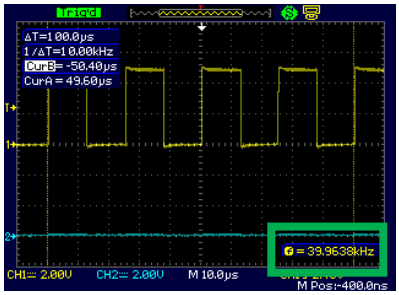
Using the code from earlier, I just captured 'scope traces with various frequencies of tone().

Here are some examples. Top left is a low frequency of 30kHz (yellow) and the TSOP remained inactive at 5V (blue). Top right is a high frequency of 45kHz, which also didn't work. The TSOP went low (ie activated) with a 35kHz carrier (bottom left) and 40kHz (bottom right) also worked.









So there's a lower threshold between 30 and 35kHz; upper is between 40 and 45kHz.