

# INFRARED PROTOCOL ANALYSIS WITH PC SOUND CARD



Infrared communication is widespread among electronic devices that use remote controls. Because of this, there are more protocols in use and the data (bytes) sent to the device depend on manufacturer. This post will show you how to view the waveform of an infrared signal emitted by a remote control, how to decode it and how you can reproduce it. In this way you can program your universal remote control or your mobile device (smartphone with IR transmitter) with the right code for the best results. Instead of using an oscilloscope for signal analysis, due to the low frequency of the IR carrier an ordinary soundcard will be used.

## **The IR receiver**

### **Modify a receiver**

The simplest receiver you can build will connect straight to the microphone plug.

This supplies the IC with 5 V too so no external power supply is needed.

If you already have the IR receiver from a TV tuner card or a SCART receiver don't rush to plug it in the sound card because pinouts are different and you may destroy the IR receiver IC.



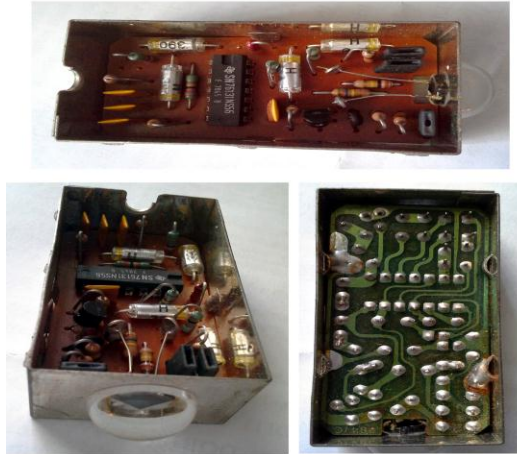
IR receiver of a SCART DVB-T tuner.  
Note that it has four pins instead of only two needed by a microphone.

The above receiver contains an IR IC and a bicolor LED. Nowadays, in all IR controlled devices you'll find an integrated circuit that receives the IR, amplifies and filters it. Here are such ICs: TFMS5300, TSOP1738, SFH506. Go to the DIY section to see how to re-wire it.

### **DIY - with integrated circuit or module**

Older devices contained an entire module responsible for IR reception.

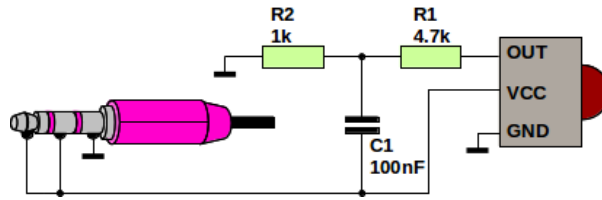
The entire module from above can be used instead of an IR IC, the only difference being the supply voltage (5 V for most ICs and 18 V for that thing). All these ICs and devices have only three pins: power supply, ground and signal output. Notice the similarities with electret microphones.



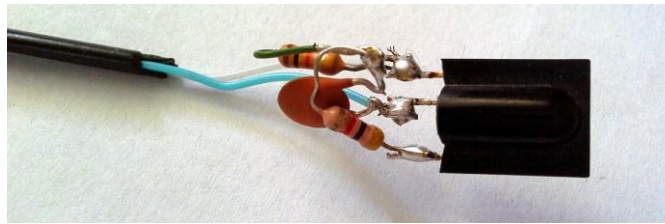
Old IR receiver made of a photodiode and an opamp.

The only difference is that a microphone's signal output is capacitor coupled to power supply, thus the microphone has only two pins. That's what our adapter will do. It will connect the signal output to power supply via a capacitor and it will use a current limiting resistor to power the IC. Note that if you own and plan to use a device similar to the old receiver shown above, things change. You'll need external power supply and most important you'll need to chop or shift level the output signal in order not to damage the soundcard.

Here is the schematic and a photo of the simplest IR receiver with microphone plug. You can modify something similar to the above receiver (for a SCART DVB-T tuner) according to this schematic. If you want to play with an old module similar to the above keep in mind that supply voltage is different - you'll need separate power because microphone jack power cannot be used anymore.



Soundcard IR adapter (based on Matthias Ringwald's schematic)

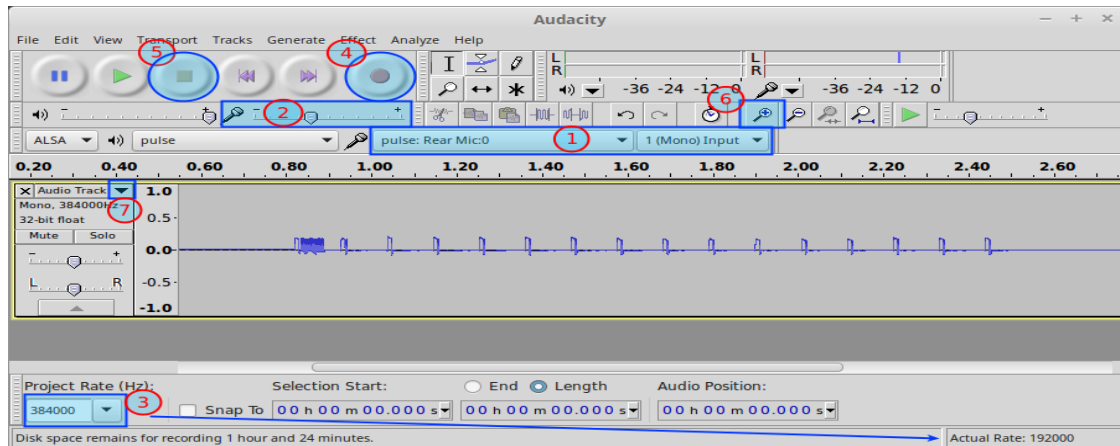


Built IR receiver. There's no need for a PCB.

R1 and R2 form a voltage divider that lowers output signal maximum level to less than 1 V. C1 couples the AC waveform to the VCC line (replacing it with a 4.7 - 10 uF electrolytic with positive pole to R1/R2 and negative to GND would produce better results). Inside the soundcard another capacitor will feed this signal into the ADC. Plug this adapter into the microphone input and go to the next section.

## View the waveform

The simplest to use is Audacity. It is a cross-platform software that can record and display the signal. First you must select the right device (rear microphone in my case) and choose mono input (1). Do a few test records (4 and 5) and if the level is too low adjust it (2) - if there's no signal at all check input and adapter. Set the project sample rate to the maximum value (3) - note that after the record is done, the real sample rate will be displayed in the bottom right corner.

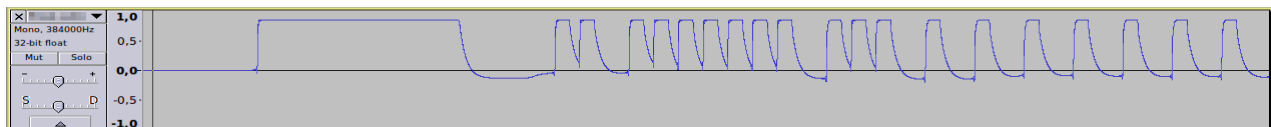


Audacity main window.

As you can see the level is a bit low in my example. But if you click on the drop arrow next to **Audio Track** (7) and select Waveform (dB) things get a lot better.

Now start zooming in (6) to see the actual protocol.

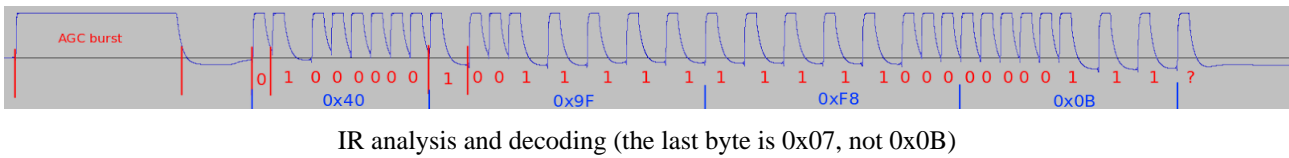
Looking at my example, this remote control sends the actual command bytes once and then if I keep pressing the button it sends a repeat pulse. So I will zoom in the blueish part of the signal - you can use Ctrl++. This is similar to what you'll get:



IR waveform in Audacity

Of course I didn't have any idea how to decode this protocol. But, using San Bergmans' IR Remote Control page I was able to tell that this is some kind of NEC protocol.

Now, if I analyze this it has the following meaning:

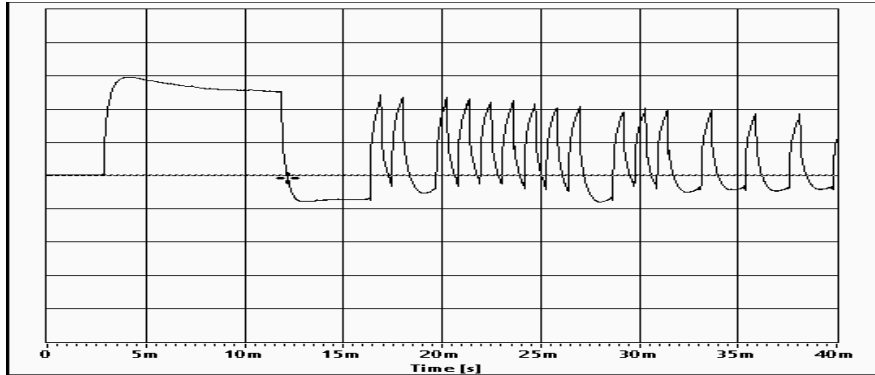


The waveform starts with a long burst needed by the receiver IC to calibrate its gain. Then the remote sends four bytes in total but only two are useful. First is the address byte. Then it should be the inverted address byte for error control. With some manufacturers, no matter what button you press on the same remote control, the address byte stays the same. I don't know why this remote control sends the address wrong ( $0x40 + 0x9F$  is not  $0xFF$ ). The third byte is the actual command and the fourth is the inverted command byte ( $0xF8 + 0x07 = 0xFF$ ). Whether the bytes are sent with MSB or LSB first I'm not sure.

Here are some other software solutions for viewing the signal waveform. Note that with all you must first choose the audio input device before using. Always choose the highest sample rate possible and mono input. Note that some Windows versions disable audio devices when nothing is plugged in.

### Soundcard Oscilloscope - Windows only

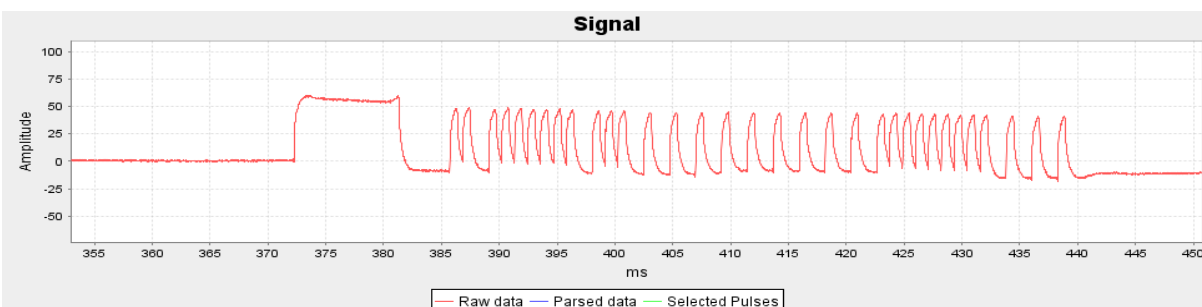
It's quite difficult to "catch" the signal on this software. Go to **Settings** and choose correct **Input** device.



The waveform displayed on *Scope*

## Protocol Analyzer

This is a Java based cross-platform software that can record and analyze various protocols. The analysis doesn't always work as it should so it's best to sample the raw signal. Go to **View - Settings - Audio Settings** tab and select the correct **Source**. Go to **Action - Sample x seconds** and quickly press a button on the remote control. A new line should appear in the main window. If not tweak the input volume from sound settings. Double click the line with raw recorded signal and you should see a graph. Click and drag to select a rectangle that will be zoomed in. Also right-click and zoom each axis as you like. With the right zooming my protocol looked like this:

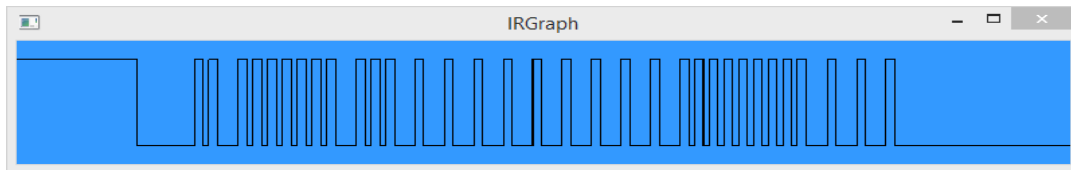


The waveform in *Protocol Analyzer*

## WinLIRC

WinLIRC contains the IRGraph utility which displays a windows with a processed square wave. It starts recording after the first rising edge which is not displayed.

Continue reading to see how to set up LIRC.



The square wave signal in *IRGraph*

## Record the signal

There are multiple ways of recording that waveform but the point is to be able to reproduce it. The waveform as it comes out of that IR receiver IC is a square wave that is highly distorted especially by that coupling capacitor. You can try to improve the hardware by using a separate power supply (can be derived from USB) and feed the output of the IC straight into sound card.

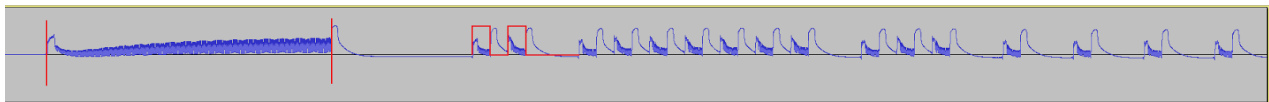
## Audio recording

This is almost pointless. You'll have the waveform recorded as-is but even if you build a simple amplifier with an IR transmitter LED and plug it into the output of the sound card, then play the record through it you'll get nothing. That's because the signal is more complex than that. The remote sends a 38 kHz pulse that is modulated by the waveform you recorded.



The high pulses you see are actually 38 kHz signals that are interpreted by the soundcard as high. If you can set up the sample rate to a higher value, multiple of 38 kHz you could see the carrier of 38 kHz.

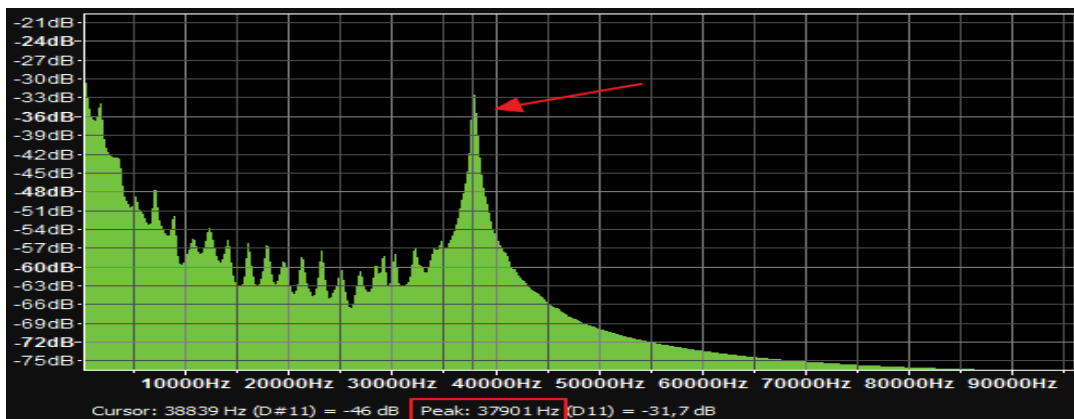
This is how a IR waveform looks when sampled at 192 kHz. The signal is distorted a lot by the adapter but it can be seen clearly the carrier with a 38 kHz frequency.



Waveform sampled at 192 kHz. The high peaks after each 38 kHz carrier shouldn't be there.

If I plot the spectrum of this waveform I can see that the carrier really is at 38 kHz.

Totally irrelevant results are displayed at low sample rate (44 kHz).



The spectrum shows a high level of the 38 kHz carrier. Generated in Audacity (*Analyze - Plot Spectrum*)

Probably the waveform looks much better with another adapter but Audacity can only display audio waveforms.

Note that 38 kHz is the most used carrier frequency but there are remotes which use 36 kHz or 40 kHz.

## Data recording

LIRC (and its Windows equivalent WinLIRC) will be used here.

**On Linux** it's easy to install LIRC. Just run `sudo apt-get install lirc` in a Terminal on Ubuntu distros. You'll be prompted to choose your hardware. I couldn't get the receiver working in Ubuntu with the ALSA audio input source so I built a serial port transceiver like this. To record the signals open a Terminal and run (the configuration file will be created in the directory you are running `irrecord` from):

```
irrecord -f -d /dev/lircd test.cfg
```

**On Windows** just extract the downloaded archive and start `winlirc.exe`. Click OK at the error message and when the *Setup* dialog appears choose *AudioCapture.dll* plugin. Configure it with the right input and the maximum sample rate. Click the *Create Config* button and close the CMD windows that appears. Click OK at the error message and when prompted to open the *Setup* dialog again select *Cancel*. Now go to plugins folder and open `WinLIRC.ini`. You should see the audio configuration in this file. If it doesn't exist try to configure WinLIRC again by launching it.

Here's my audio configuration:

```
[AudioInputPlugin]
AudioDeviceName=Microphone (Realtek High Defini
AudioFormat=65536
LeftChannel=1
Polarity=2
NoiseValue=16
```

Now open a command prompt in the folder where winlirc.exe and irrecord.exe are and execute the command:

```
irrecord -f -d AudioCapture.dll ../test.cfg
```

## All OSes

Follow on-screen instructions (press Enter a few times, then press and hold remote buttons and name them). Note that I always used the -f argument which disables any processing of the recorded waveform, saving it as raw timeframes. After you finish open test.cfg with a text editor. It will look like similar to this (just the codes for a button showed):

```
name OK
9395 4072 781 343 770 1458
770 354 760 354 760 354
760 364 760 354 750 364
```

```

750 1489 750 364 750 375
739 1500 10 10 708 1500
739 1500 10 10 708 1510
10 10 708 1510 10 10
708 1500 10 10 729 1489
10 10 718 1500 10 10
718 1489 10 10 718 1500
10 10 708 385 739 375
729 395 718 395 718 395
718 406 708 406 708 416
708 1531 10 10 687 1520
10 20 10 10 666 1520
10 10 687

```

What are those numbers? It's very simple. They represent time in microseconds. So 9.395 ms is high followed by 4.072 ms low (doesn't it look like the AGC burst and the space after it?) and so on. Let's rewrite it by grouping data bits timeframes:

```
(9395,4072)=header
```

```
( 781, 343)( 770,1458)( 770, 354)( 760, 354)( 760, 354)( 760, 364)( 760 ,354)( 750,
364)=01000000
```

```
( 750,1489)( 750, 364)( 750, 375)( 739,1500)( 708,1500)( 739,1500)( 708,1510)(
708,1510)=10011111
```

```
( 708,1500)( 729,1489)( 718,1500)( 718,1489)( 718,1500)( 708, 385)( 739, 375)( 729,
395)=11111000
```



The syntax is the following: frequency, pulses, space. All timings are represented as microseconds divided by 26. So frequency is 38000 and space is 110 ms = 110000 / 26 = 4230. I'll convert my code to this format. There are two ways of doing this:

- divide each time frame by 26 or
- The example protocol I showed here is NEC and if I were to follow the specs: bit 1 is 22, 65, and bit 0 is 22, 22,.

I'll use the second method although it doesn't produce exactly what I recorded and it sends a standard NEC protocol. So my code for the Android app will look like:

```
38000,346,173,22,22,22,65,22,22,22,22,22,22,22,22,22,22,22,22,22,65,22,22,22,22,22,65,22,65,22,65,22,65,2,2,65,22,65,22,65,22,65,22,22,22,22,22,22,22,22,22,22,22,22,22,22,22,22,65,22,65,22,65,4231
```

Source: <http://onetransistor.blogspot.in/2014/12/infrared-protocol-analysis-with-pc.html>