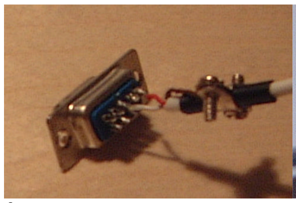
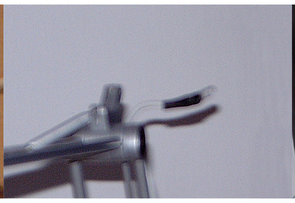
World's Simplest PC-IR Remote control

Oded Noam oded@no.am


first component in the circuit


Second and last component

That's it! Simple, huh?

Now I have a remote control on my desktop!

Free License

World's simplest – why, why not

There are other IR remote control projects on the Internet. Being very novice at electronics projects (but a very seasoned software engineer), I decided I should try a create one with simpler hardware (but more sophisticated software), more appropriate for my skills.

Studying other IR projects on the net, I saw that most of their technical complexity was in the circuit that creates a 40KHz carrier for modulating the IR signal. Some other projects also include IR sampling circuits useful for the encoding software, a problem I had to solve otherwise (see http://www.geocities.com/odednoam/AudioCardIRSampler.pdf). I figured all this complexity should be avoided in my project, as I wanted to avoid the more "complex" electronics; as it turned out, the modulation mechanism can be emulated by software.

If you have to choose between this project and a more complex project, I can only give you some points to consider:

- The more-complex modulation circuits usually require at least 4 different components (a resistor and capacitor for modulation, a transistor for interconnection and an IR-LED for transmitting). If you are not experienced with electronics, you might prefer something simpler to assemble (at which you will have a lesser chance of making a mistake).
- The more-complex modulation circuit also means more footprint: it will usually require a small box, some will also require an external power supply (which might be a problem).
- The more complex modulation circuit gives better modulation. That is because I could only create 38.4KHz modulation instead of 40KHz. This means better range for other projects (I don't know how well they work, but my LED needs to be 15-20cm from the IR sensor to work properly).
- Other projects include IR sampling circuits. Since I only give the codes for CMT-MD1 minidisc system you may need sampling (which is still possible – see http://www.geocities.com/odednoam/AudioCardIRSampler.pdf).

If you choose to implement my circuit, please let me know how well it works, or if you have any ideas of improving it.

What you need to buy

1 infrared LED
1 low-current electric cord (2-wires)
1 female DB9 connector (for serial port)

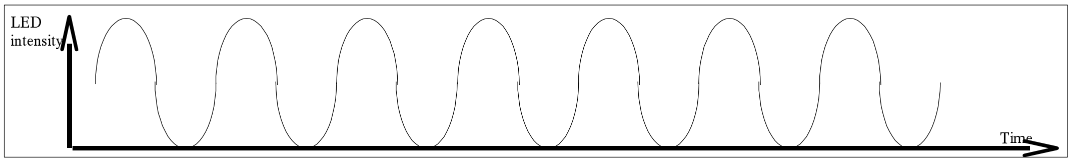
All components are common enough so you can find them in a regular electronics store (one that sells components to technicians). If you don't have one in your neighborhood, maybe a technician will sell the components to you (these guys usually have large stocks of everything), or buy them on the internet (www.radioshack.com, www.futurlec.com).

All in all, if you pay more than $1 to $2, you've been ripped-off.

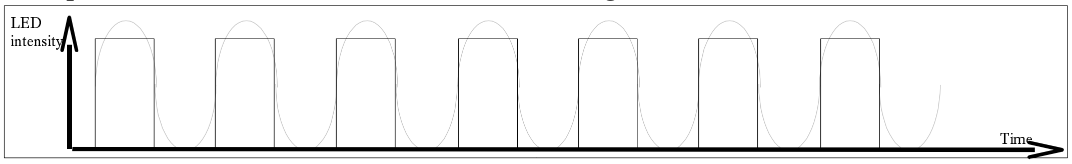
How and why it works

The infrared receiver expects a 40KHz modulated signal. “Modulated” means that the receiver has a frequency filter, and only sees light that “blinks” 40,000 times a second. To code the commands we will alternatively send “signal” (blink our LED) and quiet (send nothing).

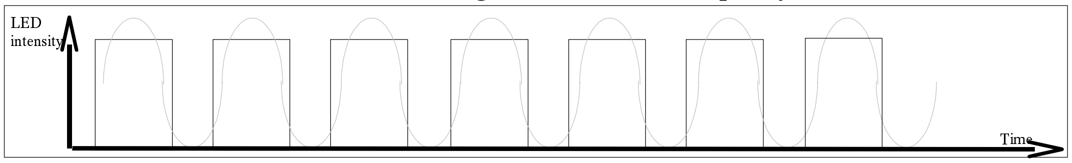
A real remote will send this wave-shaped signal:



The simple remote, however, does not implement the resonance circuit that creates the waveform. Instead, we use the PC serial port to send bits. Suppose we could sent 80,000 bits per second on the serial port, by simply sending a '1010101010....' bitstream down the serial port, we would create an almost-similar signal:

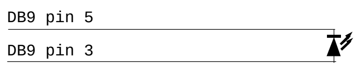


In that example, every 2 bits represent one phase of the waveform. Therefore, to create a 40KHz signal, we need 40,000 phases per second, hence the 80,000 bits per second bitrate we need.
Unfortunately this is not feasible: serial ports interfaces can be open at some common speeds (e.g., 9600, 19200, 33600, 57600 and 115200 bits per second), but not on 80000bps or anything near. But, if we can open a port at 115200, and send the signal '110110110...' or '100100100...', the signal would still be pretty much alike:



This signal is probably still good enough to fool the modulation circuit, and seem like a waveform. The problem is, the frequency of the signal we are sending is not exactly 40KHz – cause we are sending one phase every 3 bits, and at 115200bps it's (115200/3) =38400 phases per second, a 4% deviation. However, most modulation circuits will still receive this signal, but at lower intensity (which means shorter range, but hey, at least it works).

Building the circuit



As you can see in illustration 1, the circuit is fairly simple (did I say it is world's simplest?).
Just connect the cord to pins 3 (transmit signal) and 5 (signal ground) to the cord, and the other side of the cord to the LED.

In the suggested connection of the LED, a 0 bit sent to the port will turn on the LED, and a 1 bit will turn it off.

Should you connect it in reverse, it will still work: 1's will still have the correct waveform (only in reverse phase than we intended, but the modulation circuit doesn't know what we intended anyway), and 0's will be constant-on instead of constant-off (but if there's no waveform, the modulation circuit doesn't see it anyway, which makes it a 0).

Software

The software handles the outgoing signal in 3 layers:

1. **Command protocol**
   - Usually, specific to each device – for example, I implemented the coding for the CMT-MD1 shelf system.
2. **Signal bit-coding**
   - Usually, specific to the system's manufacture – I implemented Sony's SIRCS protocol.
3. **Modulation**
   - Most IR remote controls use the same modulation.

If you are not using my software implementation, I do suggest that you separate the 3 layers, to make it easier later to adapt your software to other devices.

Command protocol for CMT-MD1

There are many documents on the internet with Sony codes. Most are valid for the CMT-MD1; this is the list of codes I use, some of them might be specific to the MD1:

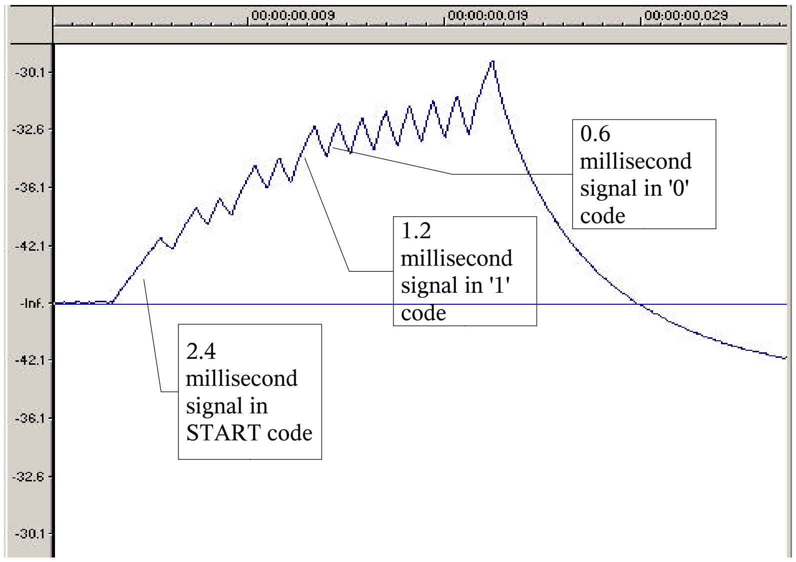
Command	*# bits in device ID*	*Device ID*	*Command code*	*Command*	*# bits in device ID*	*Device ID*	*Command code*
MD/Eject	5	15	22	System/Function menu	5	12	105
MD/Previous track	5	15	32	System/Power off	5	16	47
MD/Next track	5	15	33	System/Power on-off toggle	5	16	21
MD/Play	5	15	42	System/Set sleep timer	5	16	96
MD/Stop	5	15	40	“yes” (in menus)	13	1850	89
MD/Pause	5	15	41	“no” (in menus)	5	15	61
MD/Standby for record (rec+pause)	5	15	45	Volume up	5	16	18
MD/“Edit” menu	5	15	60	Volume down	5	16	19

Command	*# bits in device ID*	*Device ID*	*Command code*
MD/Play track #0..9	5	15	0-9
MD/Track #10..99	5	15	10
MD/Scroll display	5	15	25
Tuner/Scroll frequency up	5	13	18
Tuner/Scroll frequency down	5	13	19
Tuner/Scroll preset up	5	13	16
Tuner/Scroll preset down	5	13	17
Tuner/Switch band	5	13	15
Tuner/Stereo-mono toggle	5	13	33
CD/Play	5	17	50
CD/Pause	5	17	57
CD/Stop	5	17	56
CD/Eject	5	17	22
CD/Previous track	5	16	48
CD/Next track	5	16	49
"Edit" menu	5	1850	88
"Display" menu	5	16	75
Toggle DBFB	5	12	49
Timer select	5	16	98
Timer set	5	16	101
"Music" menu	5	12	98
CD+MD/Repeat button	13	1850	45
Tuner+tape/Stereo-mono toggle			
Play mode (suffle/pgm)	13	1850	40
"+" button (in menus)	13	1850	120
"-" button (in menus)	13	1850	121
Cursor forward (in editing)	13	1850	122
Cursor backward (in editing)	13	1850	123
Send character (in editing)	13	3130	ASCII character

Bit coding for Sony SIRCS

As can be found in many other documents on the internet, the bit coding of SIRCS is simple: a constant code before the signal (probably used to calibrate the AGC to the signal intensity), then bits are sent in reverse order (LSB first), with a given code for 1 and another for 0. However, I found that the coding mentioned in some of these other documents was inaccurate or incompatible with the SMD1 remote (used by the CMT-MD1). I built the remote sampler (http://www.geocities.com/odednoam/AudioCardIRSampler.pdf) to find out what is the correct coding to use, and got these results:

START code: 2.4 millisecond signal, 0.6 millisecond quiet
0 code: 0.6 millisecond signal, 0.6 millisecond quiet
1 code: 1.2 millisecond signal, 0.6 millisecond quiet



Other documents specify the codes to be based on 0.4 millisecond intervals; to my great disappointment at the time these signals did not work on my CMT-MD1. I would very much like to know if it works for everyone, so please e-mail me if you have any trouble with it.

Modulating for 40KHz remote controls

As explained in the ***“How and why it works”*** section, the goal is to encode the data in a way it will send the '001001001...' or '110110110...' bits at 115200bps. The limitation are:

- Byte size can only be 7 or 8 bits
- A start bit of '1' is sent by the serial port before each byte
- The port can be configured to send a '0' bit can as a stop bits after each byte.

I chose to implement the modulation by configuring the port to use 7-bits per byte and one stop bit, creating a 9-bit word. If the sent byte is 0x5b (91 decimal, 101101 binary), the signal is:

1	**1**	**0**	**1**	**1**	**0**	**1**	**1**	0
(start)								(stop)

The same effect can be achieved by using 8-bits per byte an no (0) stop bits.

A good technique of debugging the signal if it doesn't work is to connect a regular (visible red) LED instead of the IR led, and send the signal at 1 or 2 bps (the serial port should be able to open at this speed).

My CMT-MD1 remote/Minidisc titler

At my website (http://www.geocities.com/odednoam) you may also download the full binary and source code of the remote control software I wrote for my CMT-MD1. The source code is written in Visual Basic, and since Microsoft won't give their serial-port communications module with the $100 “Learning edition” (don't students use serial ports? Must you be professional and pay $500 for it?), I had to write a serial communications library myself. I think the code is relatively simple to understand; I apologize for it not being as well-documented as it should be at some points.

The source code is released under the GPL license. The main idea is the code is free for use and for modification, however, it must remain open and free in any further modification that is based on my source code. Just as I want to share my knowledge with all people interested in this project, I expect those people to do the same.

I have lots of plans for further development of this program, mainly in making it more versatile, up to the state in which it could replace any other remote control by simply replacing configuration files. However, I have been busy doing other projects lately, so if anyone wants to continue the development, I would gladly hand this over – people interested are welcome to contact me by e-mail.