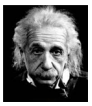
Infrared Remote Control for your Computer



Elmue

1 Dec 2017 CPOL

This article describes the easiest way to remote control any computer with any infrared remote control that you already have.The idea is to control your music or video player on the computer from your couch. I have seen serveral projects for this purpose in internet but none of them satisfied me.

Download sourcecode and documentation - 685 KB

Features

- This project is very easy to build. Perfect also for absolute beginners
- It works with any operating system: Windows, Linux, Macintosh,...
- Ultra simple hardware. You have to solder only 3 cables
- The hardware is extremely tiny (3,6 cm x 1,8 cm)
- Very cheap: the total hardware cost is US $21
- You don't have to install any additional software on the computer
- Works with ANY remote control no matter what type of signal it sends
- No need to program a special decoder for each type of remote signal
- Auto detection of the infrared data speed ("bitrate")
- Sends keystrokes to the computer by simulating a USB keyboard
- Sends debug output to the computer via virtual serial port (CDC) that shows the IR signals in real time
- Very sensitive receiver. Receives the IR signal from several meters distance even if it must be reflected twice over the walls
- In a few minutes you have adapted the source code to your needs (assigning keystrokes to remote control buttons)

Requirements

- You need a music player or video player that can be controled with keystrokes from the keyboard.
- The player program must be in the foreground (active window) otherwise it will not receive the keystrokes.

My Purpose

I want to control my music player on the computer. I use MusicMatch. The program is quite old already but much better than WinAmp or Windows Media Player. The most important functionality of the program can be controled via keyboard: Play, Pause, Stop, Next Track, Previous Track. This is all I need.

My amplifier has this remote control:

The buttons that I marked in red have no function when listening to music from the computer.
They are for the tuner and for other purposes.
When I press one of these buttons the amplifier does not do anything.
So my idea was to reuse these buttons to control the computer.

The Hardware

Only two parts are required: An infrared receiver chip and a Teensy 3.2.
The +5V power for the IR receiver comes from the USB cable.

The 3 wires are easy to solder even for a beginner.

The IR receiver TSOP38238 is very sensitive. It has a 38 kHz bandpass to filter the signal from the remote control.
This filter is required to eliminate noise because fluorescent lamps emit a very "dirty" light.
If you use a fluorescent lamp in your living room you may get a worse sensitivity for the remote control.
The majority of remote controls uses a 38 kHz carrier frequency. Some use 36 kHz which will also work. (See datasheet in the Zip file)
Only few companies (like Bang & Olufsen) use a completely different frequency.
In this case you need either another receiver chip or another remote control.

You can buy this IR receiver (or a similar model) in several online shops:

- Adafruit (US $1,95)
- Digikey (US $1,30)
- Newark (US $1,08)
- TME (US $0,55)

Also the Teensy (US $20) can be bought from Adafruit and from Digikey. Or you buy it directly from PJRC.

The **Teensy 3.2** is a very versatile and very fast board with a lot of memory.
In contrary to the **Arduino** boards the Teensy comes with a better software library (TeensyDuino) that easily allows to simulate USB devices (keyboard, mouse, joystick and more). Additionally the Teensy is much smaller (3,6 cm x 1,8 cm) and cheaper (US $20) than Arduino boards.

Remote Controls

Here you see some remote controls that I used for testing:

They are from a Yamaha amplifier, a Grundig amplifier (from the 1990's which still works!) and from a LG TV. In the middle below is a remote control from a RGB LED strip that allows to chose the color. Any of them can control the computer without the need to adapt my source code although they send completely different signals.

The IR signal

The infrared signals from Yamaha, LG, NEC and the LED control look like this:

This is measured at the output of the IR receiver chip. When there is no IR signal the output is high. When a carrier frequency of 38 kHz is deteced the output goes low. Or with other words: Each low phase in the above signal corresponds to a burst of 38 kHz modulated light. The shortest low phase in the signal is approx 550 µs for all the above remote controls. The entire signal consists of 67 high / low transitions. At the begining there is a longer start phase that gives the receiving microprocessor in the TV or Hifi device enough time to prepare for receiving the IR signal. The numbers below (in white) indicate the length of the low or high phase. A "1" means 550 µs. A "3" means 3 x 550 µs = 1650 µs. My source code detects automatically the "bitrate" of the IR signal.

The above signal encodes a button that has been pressed on the remote control. Each button generates another signal. But if you press a button for a longer time (e.g. to increase the volume) the remote control sends a "repeater sequence" which is always the same:

So if you press for example the "Volume Up" button, the signal in the first image is sent once and then the signal in the second image is repeated as long as you hold the button down.

A universal detection for any remote signal

I have seen on Github and in the Teensyduino library very long code to decode IR signals. For each and every type of remote signal an individual decoder has been written: for Aiwa, Denon, JVC, LG, Lego, Mitsubishi, NEC, Panasonic, RC5, RC6, Samsung, Sanyo, Sharp, Sony, Whynter. Huge libraries with thousand lines of code. And which of them corresponds to Yamaha? And what if my model is not implemented? Grundig is missing for example. This is surely not a universal solution.

I also found a Linux project that requires hundreds of config files. For each and every remote control an individual config file must be written. This is surely not my project.

There are also hardware projects in internet that use a much more complicated and more expensive hardware. I found a project that even uses a microprocessor that cannot be programmed via USB cable as the Teensy / Arduino boards. You have to buy an extra hardware programmer to upload the firmware. An absolute no-go for me!

I was searching for the simplest solution that does not depend on the type of remote signal. And I found one: I simply don't care about the **meaning** of each of the bits in the signal. In the first step I detect the average of the length of the short pulses (for the great majority of controls this is between 400µs and 700µs). Some remote controls use a different interval for Lo and Hi pulses. For example 400µs for Hi and 600µs for Low. Therefore my code calculates them separately.Then I calculate the multipliers. For the signal in the image above I get: 16,8,1,1,1,3,1,1,1,3,1,3,1,3,..... I don't have to understand what they mean. I just have to recognize them when they come in. Then I convert the signal into a character string and calculate a CRC 32 from that string. For each button on the remote control I get a different 8-digit hexadecimal value. My "decoder" needs less than 50 lines of C code for that.

When the signal is low I use lower-case characters and when it is high I use upper-case characters.
For example a "1" becomes "A" or "a". And a "3" becomes "C" or "c", etc...
But the two long pulses in the start phase (9 ms and 4.5 ms) are an exception. They are not an exact integer multiple of a short pulse.

Therfore the first long pulse may be detected one time as 16 and another time as 15 times the length of a short pulse. For that reason I replace long pulses with an "X" or "x" as their exact length is not interesting.

The Yamaha code

Hide Shrink Copy Code

```
Lo:  9000 us       --> Len: 15 --> Char: 'x'
Hi:  4473 us       --> Len:  8 --> Char: 'X'
Lo:   585 us short --> Len:  1 --> Char: 'a'
Hi:   537 us short --> Len:  1 --> Char: 'A'
Lo:   613 us short --> Len:  1 --> Char: 'a'
Hi:  1633 us       --> Len:  3 --> Char: 'C'
Lo:   579 us short --> Len:  1 --> Char: 'a'
Hi:   541 us short --> Len:  1 --> Char: 'A'
Lo:   581 us short --> Len:  1 --> Char: 'a'
Hi:  1657 us       --> Len:  3 --> Char: 'C'
Lo:   585 us short --> Len:  1 --> Char: 'a'
Hi:  1662 us       --> Len:  3 --> Char: 'C'
Lo:   581 us short --> Len:  1 --> Char: 'a'
Hi:  1659 us       --> Len:  3 --> Char: 'C'
Lo:   584 us short --> Len:  1 --> Char: 'a'
Hi:  1659 us       --> Len:  3 --> Char: 'C'
Lo:   583 us short --> Len:  1 --> Char: 'a'
Hi:   537 us short --> Len:  1 --> Char: 'A'
Lo:   581 us short --> Len:  1 --> Char: 'a'
Hi:  1662 us       --> Len:  3 --> Char: 'C'
Lo:   581 us short --> Len:  1 --> Char: 'a'
Hi:   539 us short --> Len:  1 --> Char: 'A'
Lo:   585 us short --> Len:  1 --> Char: 'a'
Hi:  1660 us       --> Len:  3 --> Char: 'C'
Lo:   579 us short --> Len:  1 --> Char: 'a'
Hi:   541 us short --> Len:  1 --> Char: 'A'
Lo:   584 us short --> Len:  1 --> Char: 'a'
Hi:   536 us short --> Len:  1 --> Char: 'A'
Lo:   583 us short --> Len:  1 --> Char: 'a'
Hi:   537 us short --> Len:  1 --> Char: 'A'
Lo:   585 us short --> Len:  1 --> Char: 'a'
Hi:   535 us short --> Len:  1 --> Char: 'A'
Lo:   585 us short --> Len:  1 --> Char: 'a'
Hi:  1658 us       --> Len:  3 --> Char: 'C'
Lo:   585 us short --> Len:  1 --> Char: 'a'
Hi:   535 us short --> Len:  1 --> Char: 'A'
Lo:   586 us short --> Len:  1 --> Char: 'a'
Hi:  1659 us       --> Len:  3 --> Char: 'C'
Lo:   609 us short --> Len:  1 --> Char: 'a'
Hi:   510 us short --> Len:  1 --> Char: 'A'
Lo:   584 us short --> Len:  1 --> Char: 'a'
Hi:  1659 us       --> Len:  3 --> Char: 'C'
Lo:   581 us short --> Len:  1 --> Char: 'a'
Hi:  1636 us       --> Len:  3 --> Char: 'C'
Lo:   606 us short --> Len:  1 --> Char: 'a'
Hi:   539 us short --> Len:  1 --> Char: 'A'
Lo:   586 us short --> Len:  1 --> Char: 'a'
Hi:   532 us short --> Len:  1 --> Char: 'A'
Lo:   588 us short --> Len:  1 --> Char: 'a'
Hi:   537 us short --> Len:  1 --> Char: 'A'
Lo:   583 us short --> Len:  1 --> Char: 'a'
Hi:  1637 us       --> Len:  3 --> Char: 'C'
Lo:   632 us short --> Len:  1 --> Char: 'a'
Hi:   486 us short --> Len:  1 --> Char: 'A'
Lo:   607 us short --> Len:  1 --> Char: 'a'
Hi:  1662 us       --> Len:  3 --> Char: 'C'
Lo:   581 us short --> Len:  1 --> Char: 'a'
Hi:   539 us short --> Len:  1 --> Char: 'A'
Lo:   581 us short --> Len:  1 --> Char: 'a'
Hi:   541 us short --> Len:  1 --> Char: 'A'
```

```
Lo:   583 us short --> Len:  1 --> Char: 'a'
Hi:  1655 us       --> Len:  3 --> Char: 'C'
Lo:   584 us short --> Len:  1 --> Char: 'a'
Hi:  1659 us       --> Len:  3 --> Char: 'C'
Lo:   585 us short --> Len:  1 --> Char: 'a'
Hi:  1657 us       --> Len:  3 --> Char: 'C'
Lo:   585 us short --> Len:  1 --> Char: 'a'
Lo: Shortest:  579 us, Limit:  868 us, Average over 33 short intervals:  587 us
Hi: Shortest:  486 us, Limit:  729 us, Average over 16 short intervals:  532 us
Decoded: xXaAaCaAaCaCaCaCaAaCaAaCaAaAaAaAaCaAaCaAaCaCaAaAaAaCaAaCaAaAaCaCaCa
CRC:     0x1856440C
Button:  Volume Up
--------------------------

Lo:  9025 us       --> Len: 16 --> Char: 'x'
Hi:  2210 us short --> Len:  4 --> Char: 'D'
Lo:   581 us short --> Len:  1 --> Char: 'a'
Lo: Shortest:  581 us, Limit:  871 us, Average over  1 short intervals:  581 us
Decoded: xDa
CRC:     0xF4FCC4CA
Button:  Volume Up
--------------------------
```

This is the debug output that is sent to the virtual serial COM port on the computer. You see this output for example in the "Serial Monitor" of the Arduino compiler. You see exactly what has been received and how it has been processed by the Teensy. At the end the CRC is printed which identifies the button on the remote control or the repeater sequence.

My code also takes care of the "repeater sequence". The repeater is always sent while you hold any button down on the remote control. It makes sense for buttons like Volume Up/Down to repeat a keystroke to the computer while the button is pressed. But it does not make sense for a button like "Pause". If you would repeat the "Pause" command, the audio/video player would permanently switch between Play and Pause while you press the button. Some remote controls do not use a repeater sequence at all. Therefore my code has a flag that you must only set for commands to be repeated.

NOTE: It is an error to regulate the volume on the computer. You should always let the volume on the computer at 100% and regulate the volume only at your amplifier.

The Grundig code

Hide Copy Code

```
Decoded: aXaAaAaAaAaAaAaAaAa
Decoded: aXaAaBbAaAaAaAaAa
Decoded: aXaAaBbAaAaAaAaAa
Decoded: aXaAaBbAaAaAaAaAa
Decoded: aXaAaAaAaAaAaAaAaAa
```

It is completely different:

- It uses a relation of 1:2 between short an long pulses rather than 1:3 as the Yamaha. ("b" and "B" instead of "c" and "C")
- The packages are much shorter.
- Instead of sending a repeater sequence the command itself is repeated while holding a button down.
- Before and after the command a sequece is sent which is always the same: 0101010101010101010

But all these differences between the remote controls do not matter. My code works with any remote control.

Controling the computer

For the MusicMatch player the following keystrokes have to be sent:

- Play button on remote control sends CTRL + P to the computer
- Stop button on remote control sends CTRL + S to the computer
- Pause button on remote control sends key Pause to the computer
- Tuning down button on remote control sends ALT + Left Arrow to the computer (previous track)

- Tuning up button on remote control sends ALT + Right Arrow to the computer (next track)

You must only adapt the function `ExecuteAction()` to your needs. You get the CRC code for each button from the debug output and just copy and paste it into the source code. The function `PressKey()` then sends a keystroke to the computer via USB keyboard which is simulated by the Teensy.

Hide Shrink ▲ Copy Code

```
void ExecuteAction(uint32_t u32_CRC)
{
    .....

    switch (u32_CRC)
    {
       case 0xA9EBE518:
          Serial.println("Button:  Tuning Up");
          PressKey(MODIFIERKEY_ALT,  KEY_RIGHT, 0);  // Next Track
          break;
       case 0x168ECD6E:
          Serial.println("Button:  Tuning Down");
          PressKey(MODIFIERKEY_ALT,  KEY_LEFT,  0);  // Previous Track
          break;
       case 0x8F21CFD3:
          Serial.println("Button:  Play");
          PressKey(MODIFIERKEY_CTRL, KEY_P, 0);      // Play
          break;
       case 0x3044E7A5:
          Serial.println("Button:  Stop");
          PressKey(MODIFIERKEY_CTRL, KEY_S, 0);      // Stop
          break;
       case 0x57041AEF:
          Serial.println("Button:  Pause");
          PressKey(0, KEY_PAUSE, 0);                 // Pause
          break;

       .....
    }
    ......
}
```

Special Case: RC5

There are remote controls that send a toggle bit. A typical example is the RC5 code which has been developed by Philips:

This means that each time you press the same button on the remote control this bit is toggled. So you get **two** CRC codes for one button: One with the toggle bit = high and one with the toggle bit = low. In this case your code must look like this:

Hide Copy Code

```
switch (u32_CRC)
{
   case 0xA9EBE518: // Toggle bit = 1
   case 0x4F78A03C: // Toggle bit = 0
      Serial.println("Button:  Tuning Up");
      PressKey(MODIFIERKEY_ALT,  KEY_RIGHT, 0);  // Next Track
      break;
   .....
}
```

Programming

To program the Teensy's processor you need a micro USB cable and you have to install:

1. The Arduino compiler. (Download)

2. The TeensyDuino library (Download)

You have to configure the compiler like the red settings in this screenshot:

With the option "**Serial + Keyboard + Mouse + Joystick**" the Teensy simulates a USB HID keyboard, a HID mouse and a HID joystick. Only the keyboard is used here. And "Serial" is the virtual COM port that will appear on your computer where you receive the debug output.

You can read the debug output with the freeware **TeraTerm** or with the "Serial Monitor" in the Arduino compiler.

ATTENTION: Windows may be extremely slow when you connect the Teensy's USB cable. There are 4 USB devices simulated at once by the Teensy and Windows may take up to 10 seconds until you can access the serial port. Additionally there is a bug in the Serial Monitor of the Arduino compiler which may result in the COM port never appearing. To avoid this problem you must **close** the Serial Monitor each time before you upload firmware to the Teensy.

Troubleshooting

If it does not work check the following steps:

1. When the USB cable is plugged into the Teensy the LED on the Teensy board must flash **once**. If it does not, you probably did not upload the firmware correctly. Unpack the Zip file and double click the file "IR_Receiver.ino". Configure the Arduino compiler as in the image above. In the compiler toolbar click the second button "Upload" (with the right arrow). Another window must open: the Teensy Loader. Now press the tiny button on the Teensy board. Wait until the Teensy Loader says "Reboot OK".

2. Whenever the Teensy receives any IR signal the LED on the Teensy must flash fast in the **rythm of the IR signal**. If you press a button on the remote control and the LED stays off, check that the remote control is working and check the correct connection between IR receiver and Teensy.

3. When the Teensy is connected to the computer a COM port must appear. This may take up to 10 seconds. The first time you connect the Teensy this will even take longer and you must install the **CDC driver** (which is only one INF file). You find the driver in the ZIP file. You see this port in control panel under "Ports" displayed as "Teensy USB serial". Additionally you will find under "Human Interface Devices" three "HID Input Device". They are the keyboard, mouse and joystick which you find additionally under "Keyboards" and "Mice and other pointing devices". If you are not sure disconnect the Teensy and these

devices must disappear in control panel after a few seconds.

4. In menu "Tools" of the compiler select the Teensy COM port and open the "Serial Monitor". Then press a button on the remote control. You must see the debug output that the Teensy sends. In the source code you can set PRINT_RAW_SAMPLES to true or false depending on how much debug output you want to see.

Possible Extensions

You can do much more with this project. Additionally you could connect **relays** to the Teensy to switch devices on and off or you could dim your lights with the remote control. If you connect 12V **LED strips** or LED lamps you can use a PWM output of the Teensy at 500 Hz and amplify the signal with a MOSFET transistor like BUZ71A.

You can also write a program for the computer that listens on the serial COM port for the CRC codes that are sent from the Teensy and executes more complex actions (e.g. shut down the operating system, start programs, etc..) In C# this can be done with a few lines of code using System.IO.Ports.SerialPort.

A simple light switch

You can extend the main light switch of your room with a bistable relay. So you can switch the light on and off either via remote control or via light switch.

A **bistable relay** has two coils and needs only a short pulse to switch into the other position and stays forever in this position until the other coil gets powered. You find bistable relays for example at Digikey. The PB1695-ND needs 80mA at 5V. It costs US $4,40 and

switches 16A. The pin **Vin** of the Teensy is connected directly to the +5V of the USB cable which delivers enough current (500mA) for the relay so you don't need an extra 5V supply if the computer is running.

But if you want to switch the light also if the computer is off then you need an additional 5V supply and you must cut the jumper at the bottom side of the Teensy to disconnect this 5V supply from the 5V of the USB cable.

More projects with Teensy

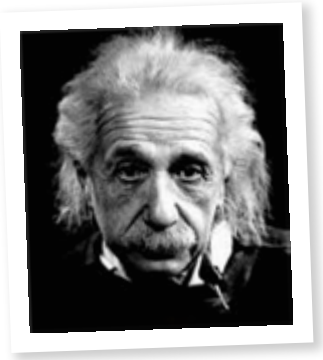
This is my third electronic project with a Teensy published on Codeproject. You may also be interested in:

- DIY electronic RFID Door Lock with Battery Backup
  It uses Desfire RFID cards with DES / AES encryption for an electronic door lock.
- A USB HID Keyboard, Mouse, Touchscreen emulator with Teensy
  It simulates 3 types of touchscreens via USB.

License

Share

About the Author



Elmue

Software Developer (Senior) ElmüSoft
Chile

Watch this Member

Software Engineer since 27 years.

Comments and Discussions

Add a Comment or Question

Email Alerts | Search Comments

First Prev Next

Thank you Elmue!
Member 14740807 1-Mar-20 21:34

Re: Thank you Elmue!
Elmue 2-Mar-20 2:42

My vote of 5
raddevus 19-Feb-18 19:39

My vote of 5
tobias1959 5-Dec-17 13:42

TV Remote control
Member 13554577 3-Dec-17 15:58

Re: TV Remote control
Elmue 6-Dec-17 18:26

Refresh 1

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.