

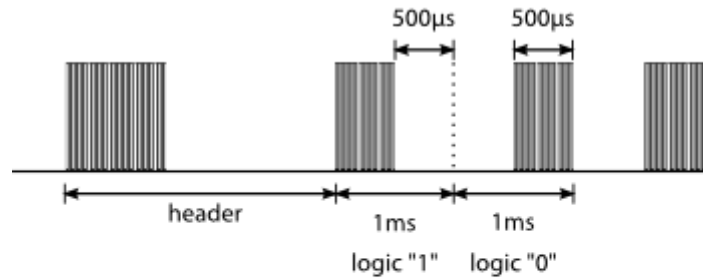
rell tan

ater
a Formats

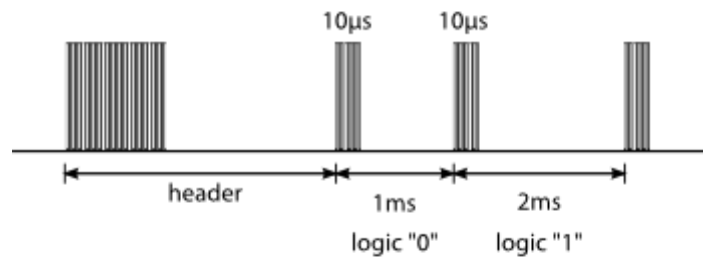
therefore the signal can only be “on” or “off”. A transmission typically starts with an a burst (“on” state) that is used for the Automatic Gain Control (AGC) circuitry in the receiver, followed by the “off” state and the actual data transmission.

There are 3 basic types of data transmission formats, which are illustrated in the following diagram. Protocols can be based on these transmission formats, but need not necessarily conform to them.

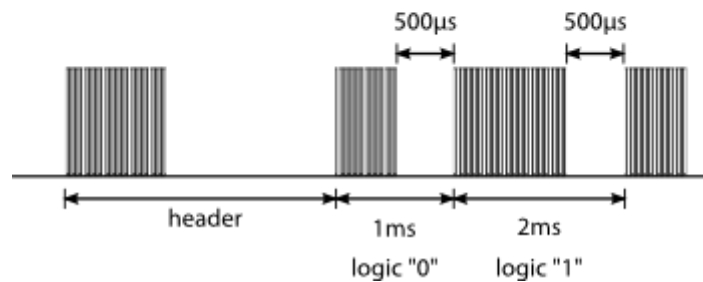
Manchester encoding



Pulse distance coding



Pulse length coding



So how do you know what your remote control uses? And how do you capture the sequence so that you can re-transmit it from an IR diode?

IR Signal Capture

To capture these infrared signals, we use an IR receiver module. This is typically a 3-pin device, which consists of an IR detector as well as built-in circuitry to demodulate the 36 – 38kHz signal, producing a digital output corresponding to the transmitted data. If we had used an IR photodiode or phototransistor instead, we would have to demodulate the signal ourselves.

If you have a logic analyzer or an oscilloscope, you could simply hook up the IR receiver module and see its output on your screen. However, I do not have either of these, so I am going to rely on my trusty Arduino. For instructions on hooking up your IR receiver, refer to either the [InfraredReceivers wiki page](http://arduino.cc/playground/Code/InfraredReceivers) (<http://arduino.cc/playground/Code/InfraredReceivers>) on Arduino Playground or [ladyada's IR tutorial](http://learn.adafruit.com/ir-sensor) (<http://learn.adafruit.com/ir-sensor>). You should also verify the pinout of your receiver by reading its datasheet.

I started off with [this sketch \(http://arduino.cc/playground/Code/InfraredReceivers\)](http://arduino.cc/playground/Code/InfraredReceivers) from the Arduino Playground, which uses pin #2 as its input. The sketch produces an output that you feed to gnuplot to get a waveform similar to what you see on an oscilloscope, so there's a lot of additional data being output to make the waveform appear square.

We are only interested in capturing the duration for each “on” and “off” period in the signal, so I've made changes to the sketch to output only the durations, starting with the “on” duration of the header. The output, formatted into rows of 4, looks something like this:

Analyze IR Remote

Waiting...

Bit stream detected!

```
9156      4548      616      584
592       1692      616     1716
592       1716      588      584
592       1716      592     1692
616       1716      588     1696
616       1712      592     1716
592       564       612      584
592       584       592      584
592      1724      592     1716
592       584       588     1716
592       584       592      584
588       588       592      584
588       588       588     1720
588       588       588      588
588       588       588     1720
588      1716      592     1716
588      1720      588
```

Bit stream end!

This output was captured from an Apple Remote, whose format is well-documented. Looking at the [Apple Remote page on Wikipedia \(http://en.wikipedia.org/wiki/Apple_Remote\)](http://en.wikipedia.org/wiki/Apple_Remote), the IR signal protocol can be summarized by the following table:

Signal Part	on (µs)	off (µs)
leader	9000	4500
0 bit	560	560
1 bit	560	1690
stop	560	—

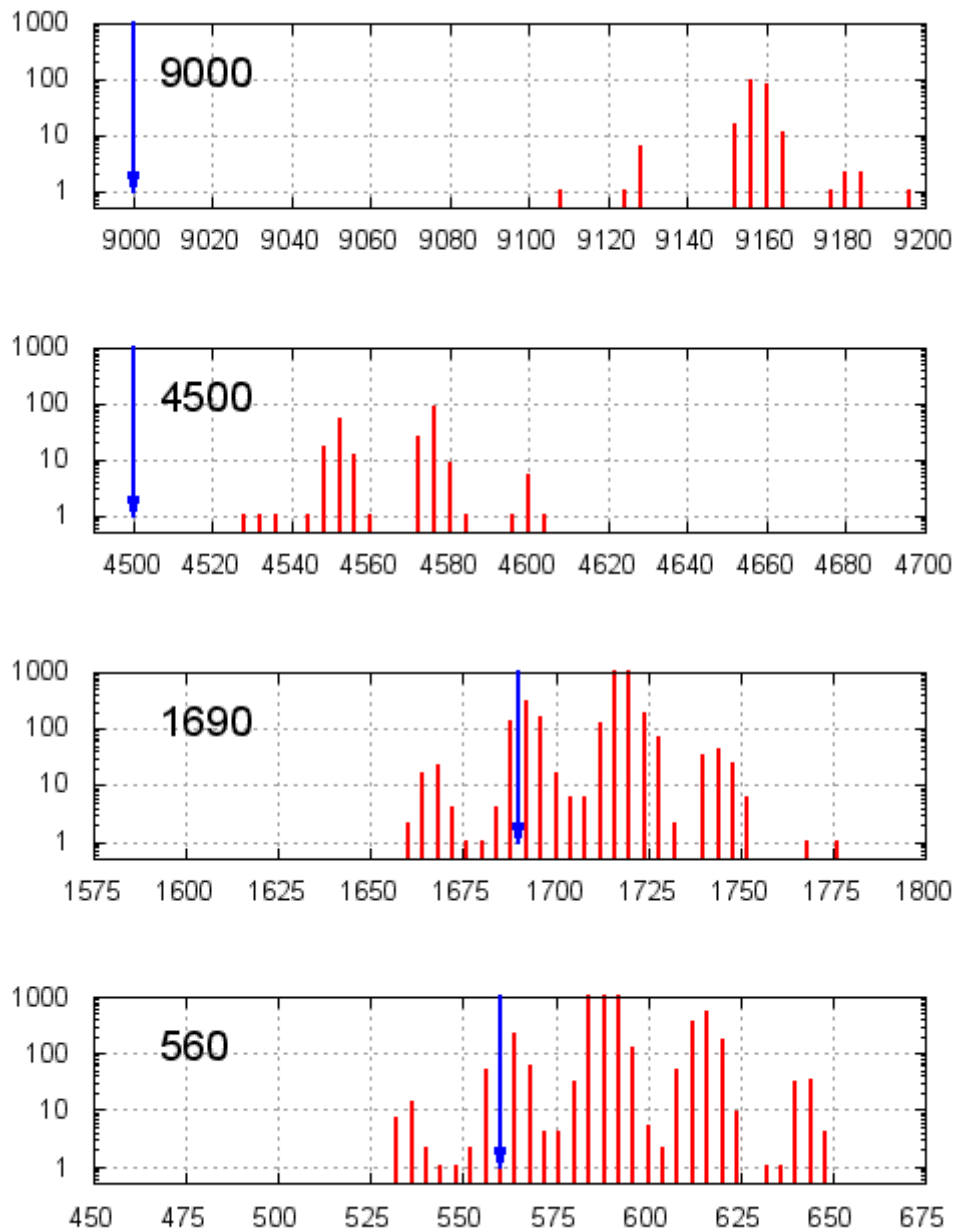
If you compare the first row of the output with the “leader” part of the signal, you can see that the values are quite close:

- 9156 vs 9000
- 4548 vs 4500

The reason why it's not spot-on is that remote controls are low-cost devices. The oscillator that provides the signal timing is probably an on-chip RC oscillator, which is not a very accurate time source. Another possible reason could be lag in the IR receiver module. Therefore when designing an IR receiver, this must be taken into account. A 20 – 25% tolerance should be sufficient to accurately decode the signal.

What should you do if the protocol that you captured is not documented? Well, you should capture it a couple of times to get a good idea of what values are used, as well as acceptable. To illustrate this, I pressed the Play/Pause button on my Apple Remote about 100 times and counted the number of times each duration value was seen.

When you plot the histogram for all the durations, you should see a few clusters, representing the regions of interest. For the Apple remote, there will be 4 such regions, each of which will correspond to a value in the above table. These regions are shown in detail below:



The blue arrows and the top left number indicate the documented pulse duration.

The 9000 and 4500 values appear less often because they only occur at the start of the IR signal, whereas the values 560 and 1690 are used to represent the data and therefore occur many times with each button press, depending on how many bits are transmitted.

If you have no idea what protocol this is, you could browse through the different IR protocols, looking for a match. However, if your aim is to simply to replay the IR signal, then you're done – just choose suitable values to simplify your code. The IR protocol is unlikely to use many different

durations, due to previously stated reasons. It is probably safe to take the middle of the seen values since any one of those are considered valid by the receiver.

An Improved IR Capture Sketch

The sketch has a hard-coded `SAMPLE_SIZE` of 64, which captures only 64 values. Contrary to what you might think, increasing `SAMPLE_SIZE` does not automatically allow you to capture more values. For signals that have less than `SAMPLE_SIZE` transitions, the sketch will wait indefinitely to fill all `SAMPLE_SIZE` values. You could workaround this problem by pressing the remote twice, but that would be an inelegant solution.

Instead, I've rewritten the sketch to timeout when a transition is not detected after some time (i.e. when `TIMER1` overflows). IR signals do not have long durations for practical reasons – you wouldn't want to have to aim your remote at the TV for 2 to 3 seconds. I have also cleaned up the code considerably.

My version of the sketch is below:

```

1  /*
2  * IRAnalyzer.pde
3  * based on http://arduino.cc/playground/Code/InfraredReceivers
4  *
5  * modified by Darell Tan
6  * see http://irq5.io/2012/07/27/infrared-remote-control-protocols-part-1
7  */
8
9  #include <avr/interrupt.h>
10 #include <avr/io.h>
11
12 #define READ_IR      ((PIND & _BV(2)) >> 2) // IR pin
13 #define TIMER_RESET  TCNT1 = 0
14 #define TIMER_OVF    (TIFR1 & _BV(TOV1))
15 #define TIMER_CLR_OVF TIFR1 |= _BV(TOV1)
16 #define SAMPLE_SIZE  128
17
18 unsigned int TimerValue[SAMPLE_SIZE];
19
20 void setup() {
21   Serial.begin(115200);
22   Serial.println("Analyze IR Remote");
23   TCCR1A = 0x00;
24   TCCR1B = 0x03;
25   TIMSK1 = 0x00;
26
27   pinMode(2, INPUT); // IR pin here
28 }
29
30 void loop() {
31   int i;
32
33   // reset
34   for (i = 0; i < SAMPLE_SIZE; i++)
35     TimerValue[i] = 0;
36

```

```

37 Serial.println("Waiting...");
38 while(READ_IR == HIGH);
39
40 // reset timer, clear overflow flag
41 TIMER_RESET;
42 TIMER_CLR_OVF;
43
44 // first value
45 i = 0;
46 TimerValue[i++] = 0;
47
48 char prev_dir = 0;
49 while (i < SAMPLE_SIZE) {
50     while (READ_IR == prev_dir) {
51         // if the pin has not changed, check for overflow
52         if (TIMER_OVF)
53             break;
54     }
55     TimerValue[i++] = TCNT1;
56     prev_dir = !prev_dir;
57 }
58
59 Serial.println("Bit stream detected!");
60 for (i = 1; i < SAMPLE_SIZE; i++) {
61     long time = (long) (TimerValue[i] - TimerValue[i-1]) * 4;
62     if (time == 0 || TimerValue[i] == 0)
63         break;
64
65     Serial.print(time);
66     Serial.print("\t");
67
68     // newline every 4 values
69     if ((i % 4) == 0)
70         Serial.print("\n");
71 }
72 Serial.println("\nBit stream end!");
73 }

```

IR Capture using the iMON

I received a SoundGraph iMON device ([/2011/01/22/dissecting-the-soundgraph-imon/](http://2011/01/22/dissecting-the-soundgraph-imon/)) some time back and I thought that I could put it to good use here, so I downloaded LIRC (<http://www.lirc.org>) 0.9.0 and compiled it. You will need to compile two kernel modules as well: `lirc_dev` and `lirc_imon`. When loaded into the kernel, the `/dev/lirc0` device will be created (I only have one IR device).

I used the `mode2` utility that came with LIRC to print out the “raw data” received by the iMON, much like what the Arduino sketch does. By specifying the `-m` flag, values are displayed in a “wide” format.

```
[darell@localhost lirc-0.9.0]$ tools/mode2 -d /dev/lirc0 -m
  9250    4250    750    500    500    1750
    500    1750    500   11750    500    500
    750    1500    750    1500    750    1500
    750    1500    750    1500    750    1500
    750     500    750     500    500     500
    750     500    500    1750    750    1500
    750     500    500    1750    500     500
    750     500    500     500    750     500
    750     500    500    1750    500     500
    750     500    750     250    750    1750
    500    1750    500    1750    500    1750
    500    30250
```

One thing you might have noticed is, why are the numbers so “round”? If you take the greatest common divisor of 9250, 4250, 750 and 500, you will have your answer: **250**. All of these numbers have been rounded to 250µs, which is not terribly accurate.

Also, if you capture the same input signal several times, here’s what you will see (each capture is laid out vertically):

```
9000    9000    9000    9000    9000
4500    14500   4500    14250   4500
750     500     500     750     500
500     500     750     500     500
500     750     500     750     750
1750    1750    1750    1500    1500
500     500     500     750     750
1750    1750    1750    1500    1500
500     500     500     750     750
11750  1750    1750    1500    11500
500     500     500     750     750
```

How did 4500µs turn into 14500µs, and how did 1500µs become 11500µs? The values went from 1ms to 10ms – the error is quite large and it happens often.

The iMON or its LIRC drivers (whichever it is) is simply unreliable and offers very poor resolution. Sadly, I think this device will have to be scheduled for “organ harvesting” soon.

Protocol Format

Since this protocol has been documented, understanding it will be much easier. The protocol starts off with two values for the header: 9000 and 4500. Following that, it is just a stream of a pair of (560, 560) to represent a “0” or a pair of (560, 1690) to represent a “1”. This protocol uses the “pulse distance coding”, where the pulses are of the same length (560µs) but the distance between this pulse and the next decides whether the data bit is a “1” or “0”.

Let’s start by decoding the captured values:

9156	4548	616	584		header, "0"
592	1692	616	1716		"1", "1"
592	1716	588	584		"1", "0"
592	1716	592	1692		"1", "1"
616	1716	588	1696		"1", "1"
616	1712	592	1716		"1", "1"
...					
588	1716	592	1716		"1", "1"
588	1720	588			"1", STOP

Note what the wiki states about the protocol format:

The first two bytes sent are the Apple custom code ID (0xEE followed by 0x87), which are followed by ... making a total of 32 bits of data. All bytes are sent least significant bit first.

The first 8 decoded bits are 0111 0111 but since the LSB is sent first, the bits should read 1110 1110 instead, which is 0xEE in hexadecimal. The subsequent decoded byte should therefore be 0x87.

There will be a total of 32 bits and it ends with a single 560µs pulse.

What's Next?

In the [next post \(/2012/08/11/infrared-remote-control-protocols-part-2/\)](/2012/08/11/infrared-remote-control-protocols-part-2/), a case study of an unknown remote control protocol will be presented, as well as how we can use this information to make our own remote control that emulates the protocol.

ADVERTISEMENT

Skip

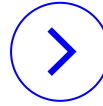
Advertisements



Instant Zero Balance A/c

Why Earn 3.5% when you
can Earn 6% p.a*. Open
Savings Acc Instantly

kotak.com

[Report this ad](#)

Instant Zero Balance A/c

Why Earn 3.5% when you
can Earn 6% p.a*. Open
Savings Acc Instantly

kotak.com

[Report this ad](#)

This entry was posted in [electronics](#) and tagged [arduino](#), [infrared](#).

14 comments on “[Infrared Remote Control Protocols: Part 1](#)”

Communication protocol for an indoor helicopter says:

22 Jan 2013 at 5:02 am

[...] It turns out the packets are using pulse-length coding (we were unfamiliar with this protocol but you can read a bit more about it here). The last piece of the puzzle was to capture packets produced by each unique change of the control [...]

Reply.

Communication protocol for an indoor helicopter | Daily IT News on it news..it news.. says:

22 Jan 2013 at 5:20 am

[...] It turns out the packets are using pulse-length coding (we were unfamiliar with this protocol but you can read a bit more about it here). The last piece of the puzzle was to capture packets produced by each unique change of the control [...]

Reply.

Valchanov Marin says:

11 Feb 2013 at 5:15 pm

Very good explanation

Reply.

Muting DSTV using Arduino | House 4 Hack says:

11 Feb 2013 at 6:05 pm

[...] background on how IR remote controls works check out this article on the irq5 blog. In short the signal is encoded in bursts of “on” and [...]

Reply.

Harry Lee says:

21 Jul 2013 at 1:58 am

Dear.

I got this decoded from Sharp projector XR-55X remote.
push button On

Bit stream detected!

```
288 1804 292 780 1 0
268 1808 288 1808 1 1
288 784 264 784 0 0
264 1808 292 1808 1 1
288 780 268 1808 0 1
288 780 268 1808 0 1
288 784 264 1808 0 1
292 780 264 42728 0 Header ?
268 1796 296 780 1 0
248 1824 296 1804 1 1
292 780 244 1852 0 1
244 804 244 804 0 0
244 1828 292 780 1 0
244 1828 296 776 1 0
248 1824 296 780 1 0
244 1824 296 42700 1 Header ?
272 1800 296 752 1 0
272 1828 268 1828 1 1
292 780 244 804 0 0
244 1828 292 1804 1 1
280 792 248 1824 0 1
276 796 248 1828 0 1
268 804 244 1828 0 1
292 780 244 42720 0 Header?
292 1800 296 776 1 0
248 1828 268 1852 1 1
268 780 244 1828 0 1
292 780 244 804 0 0
268 1828 268 780 1 0
268 1804 292 784 1 0
264 1832 264 784 1 0
268 1828 268 1 Stop
```

I tried the library of Ken Shriff but no use, cant understand the protocol. Can you help me for this ?

Reply

darell tan says:

22 Jul 2013 at 9:45 am

Hi Harry, the protocol looks fairly similar to the Apple Remote, except that there are 42700 length pulses in between. Is this a single capture or multiple captures? Also, I would recommend you press and release the button briefly instead of pressing & holding the button down.

The output waveform should look similar to the first diagram on this page, meaning the 42700 pulses *should* appear once in front only.

Hope that helps.

Reply

Emmanuel says:

1 May 2014 at 2:42 am

Nice explanation. I have read several notes on this protocol, but this one made me understand. Keep it up

Reply

Nick says:

29 Sep 2014 at 1:18 am

Hi there

Very nice explanation, easy to follow and understand. Which graphing tool are you using to make your IR plots?

Reply

darell tan says:

30 Dec 2014 at 6:16 pm

Hi Nick, I'm actually using gnuplot for this, if I remember correctly.

Reply

jiali says:

29 Dec 2014 at 5:54 pm

Hi,i want to know why a stream of a pair of (560, 560) to represent a "0" or a pair of (560, 1690) to represent a "1".

Reply

darell tan says:

30 Dec 2014 at 6:14 pm

Hi Jiali, within the IR stream for this remote there are only those few periods – 9000, 4500, 560 and 1690. The typical ways of encoding data with IR is shown in the first diagram. Since 560 and 1690 occurs frequently in the stream, it fits the "pulse distance" scheme and that's how we get the 1's and 0's.

Whether (560, 560) is actually a 1 or 0 is debatable if you don't know much about the signal. In this case someone has already done the work, so that's how the 1 and 0 has been assigned.

Reply

marco_c says:

16 Nov 2015 at 2:15 pm

Interesting article. If you are going to do more work with IR protocols you may be interested in this Analysir (<http://www.analysir.com>). Inexpensive and does a good job of capturing, analysing and display the IR stream.

Reply.

georg says:

16 Jan 2018 at 11:07 pm

Hi, I have now for some time fiddled with an IR sending device to control an LG air cond/heat pump.

Strange enough I get some of the comands to work, but not those Im after. And trying to figure out what might go wrong.

One question is; the 38kHz base frequency, what are the requirements for that fitting into the generated pulses. I mean (sloppy counting) there is some 20 38kHz pulses going into a pulse;

1. the ends of the pulse (turning on and of the "envelope" of say 590ms) may that "slice" a 38kHz basic pulse, or must the 38kHz pulses be "whole" ?
2. the phase of the 38kHz inbetween the modulation pulses (again say 590ms pulse) does it matter if the phase shifts between pulse or should it b the same ? (like some pll locking into the carrier)

In my setup i generate a 38kHz using a microprocessor counter for fast fast freq. generation, and just turn the pin output on/of, which means that the phase will be the same, while bits in ends will be sliced since I dont synchronize the on/off of carrier with the 38k.

Reply.

darell tan says:

16 Jan 2018 at 11:57 pm

Hi Georg, the timing requirements for IR signals are usually not too strict, because I suspect the remotes are meant to be cheap to produce and therefore aren't very precise. In part 2 of this IR post, I'm doing exactly the same thing as you are describing, i.e. turning the output on & off without regard to the signal phase. You can refer to the datasheet of typical IR receivers like the Vishay ones to see their test waveforms and output.

So far I have not encountered any IR signals which don't work when I re-transmit them using a microcontroller.

Hope that helps.

Reply.

This site uses Akismet to reduce spam. [Learn how your comment data is processed.](#)

[Create a free website or blog at WordPress.com.](#)