Reinventing the Wheel

https://github.com/dnstandish/TPM_Reinventing_the_Wheel

Situation

- Can’t mount USB devices
- Can’t connect to devices on the local network
- Can’t share via bluetooth
- Can’t burn media

- But it’s difficult to lock something down completely without degrading its usefulness
- There are often side channels

- images on the display can be photographed
- information can be encoded via timing/volume of netowrk traffic
- a signal can be encoided as audio

Chosen approach

- encode data into a WAV file
- play the file via standard media player
- transmit to another computer via audio cable
- record the audio signal into a WAV file
- decode the data from the recording

Proof of concept

- 8 bit 8kHz sound file
- simple pulse encoding
- lots of space between pulses
- pulse derived from sine wave

WAVE File Format

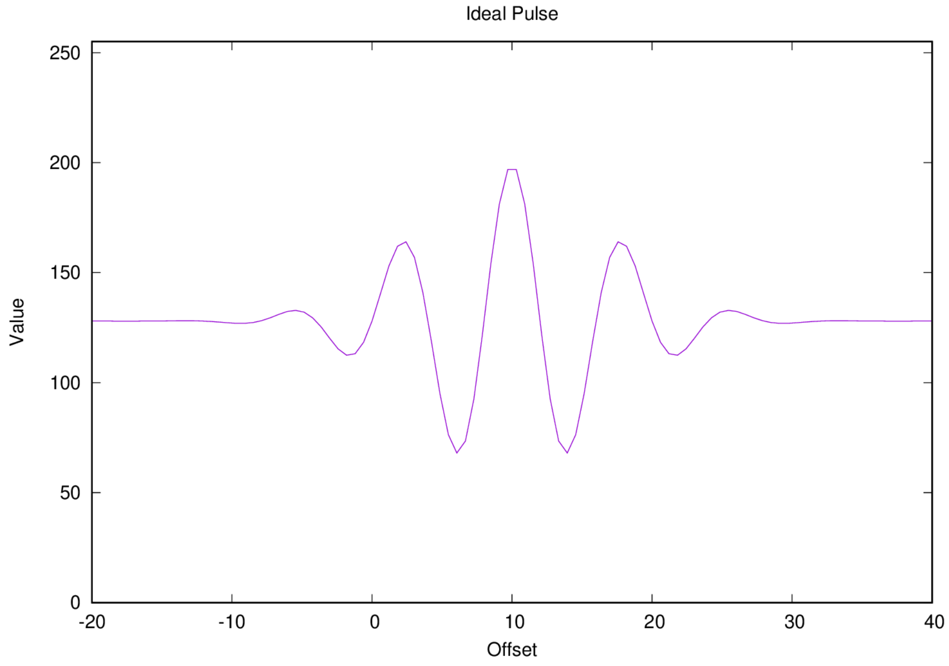
```
4    “RIFF”
4    length    length of following content unsigned 32 bit little endian4 ”WAVE”

  4    ”fmt “    format chunk

  4    length    length of following format chunk data unsigned 32 bit little endian(U32LE)

    2    wFormat   WAVE_FORMAT_PCM = 1

    2    nChannels unsigned 16 bit little endian (U16LE)

    4    nSamplesPerSec U32LE

    4    nAvgBytesPerSec    U32LE

    2    nBlockAlign    bytes for single sample U16LE

    2    bitsPerSample  U16LE

    (2)  cbSize    size of extra format info U16LE (must be 0 for WAVE_FMT_PCM. optional defaulting to 0)

  4    ”data”    data chunk

  4    length    length of following data U32LE
         <data>    8bit data is unsigned zero is 128; 16bit data is signed little endian
```

File header for 8bit 8kHz

```
sub wave_head {
    my $fd = shift;
    my $bytes_per_pulse = shift;
    my $nbytes = shift;
    print $fd
        'RIFF',
        pack('V', $bytes_per_pulse * $nbytes * 8 + 36 ),
            "WAVE",
                "fmt ",
                pack('V', 16 ), # there are 16 bytes in format chunk
                    pack('v', 1 ), # WAVE_FORMAT_PCM
                    pack('v', 1 ), # 1 channel
                    pack('V', 8000 ), # samples per second
                    pack('V', 8000 ), # avg bytes per second
                    pack('v', 1 ), # 1 channel 8bit is 1 byte for block of sample
                    pack('v', 8 ), # 8bit
                "data",
                pack('V', $bytes_per_pulse * $nbytes * 8 );
}
```

Idealized Pulse

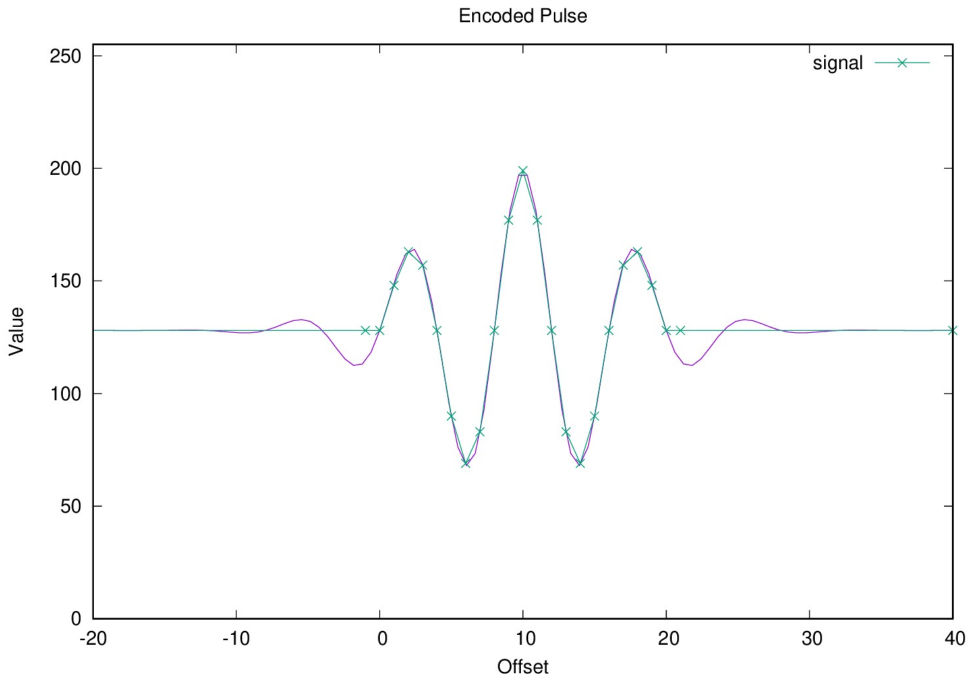
```
71 * sin(2*1000*pi*x/8000) * (0.5**(((10-x)/2)**2)) + 128
```



Pulse Data 8bit 8kHz

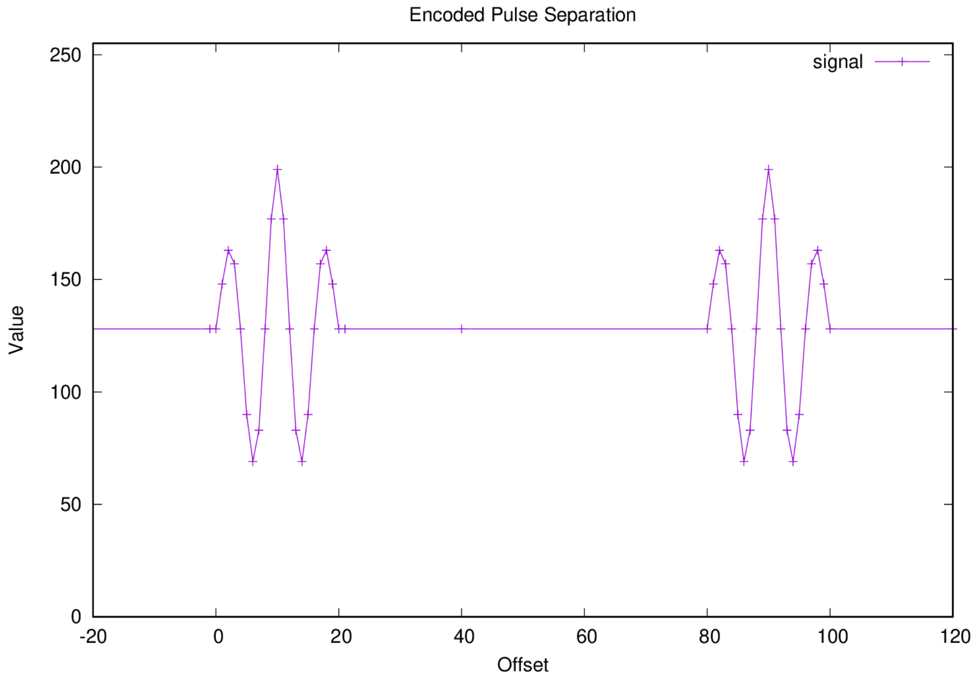
```
sub func
{
    my $i = shift;
    return 128 if $i > 20;
    my $x = sin( 2 * 1000 * pi * $i / 8000 ) * 71;
    my $f = 0.5 ** ( ( (10.0 - $i) / 8.0 ) ** 2 );
    return int($x * $f) + 128;
}

my $N_SAMPLE = 80;
my @a = map { func($_) } 0..($N_SAMPLE-1);

my $bytes = pack('C*', @a); # bytes for bit=1
my $ebytes = pack('C*', (128) x $N_SAMPLE); # bytes for bit=0
```

Encoded pulse

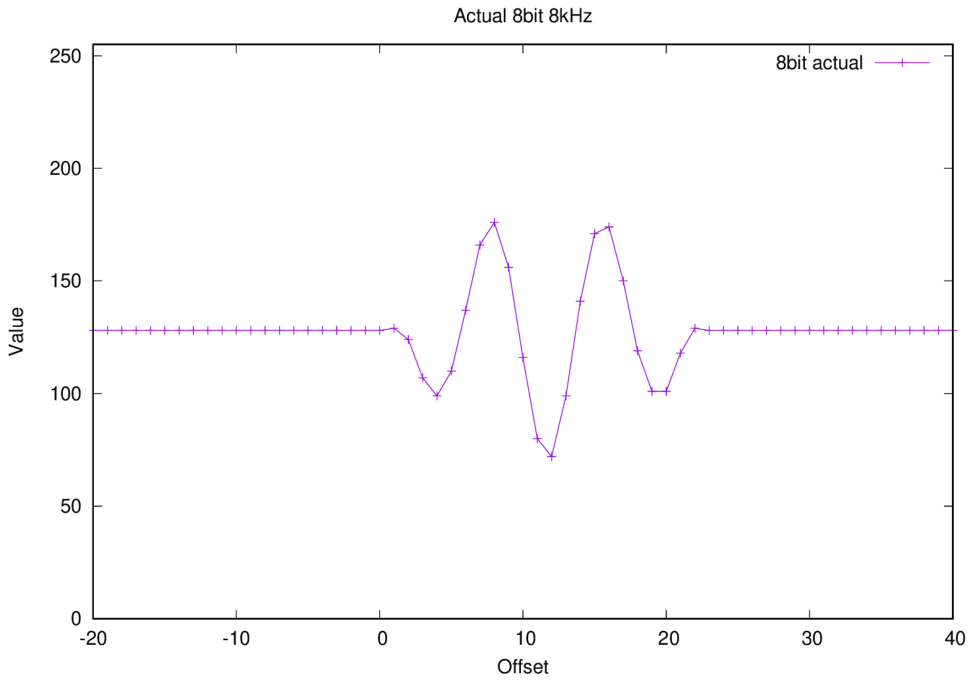


Generous Pulse Separation

20 byte pulse separated by 60 bytes silence



Pulse as Recorded



Envelope of data within recording

- Start of signal indicated by marker 0xff
- followed by length of encoded data unsigned 32 bit little endian
- data bits 1 = pulse; 0 = zeroes
- end padding 0x00 4 times

Encoding

```
sub encode_bytes {
  my $fd = shift;
  my $pulse1 = shift;
  my $pulse0 = shift;
  my $data = shift;

  for my $c ( split '', $data )
  {
    print $fd  map { $_ ? $pulse1 : $pulse0 }  split '', unpack('B8', $c);
  }
}
open my $fd_out, '>:raw', $out;
wave_head( $fd_out, $pulse_len, 4 + 1 + $file_len + $n_zero_pad );
for my $s ( "\x{ff}", pack('V',  $file_len ), $content, "\x00" x $n_zero_pad ) {
  encode_bytes( $fd_out, $bytes, $ebytes, $s);
}
close( $fd_out );
```

Decode

- verify WAV header for 8kHz 8bit
- find start marker
- decode length
- decode data

How to detect pulse?

Possibilities:

- look at 1kHz component of Fourier series?
- convolute bytes with original pulse?
- sum of absolute magnitude of bytes?
- once starting marker found data should be a fixed series of 80 byte chunks

Finding signal start

- convert unsigned data bytes to signed int relative to 128
- look for 20 byte running sum over a threshold followed by decrease of of running sum below threshold
- ignore bytes below arbitrary noise floor
- find index of maximum running sum starting 40 bytes before drop below threshhold
- 1 if 20 byte running sum > 70% max running sum
- 0 if 20 byte running sum < 40% max running sum
- if signal start should see 10001000100010001000100010001000

  (i.e. 0xFF)
- if doesn’t match continue looking

Decode data

- decode one bit at a time
- first 20 bytes > 70% threshold is a 1
- first 20 bytes < 40% threshold is a 0
- verify next three 20 byte chunks are under 40% threshold
- give up if a 20 byte chunk is between 40% and 70%
- combine 8 bits into a byte
- continue until end of specified data length

Does it work?

Yes !

Does it work?

Yes !

If the encoded file is small

For larger file the signal slowly shifts earlier in file

Effect looks to regular to be dropped bytes

Suspect this is an artifact of audio play/record not supporting 8kHz at hardware level

Add periodic resyncronization

after 20 bytes decoded data (1.6 sec of audio)

on next 1 bit find position of max running sum in range [-30,+40]

if position differs more than 2 samples adjust signal offset

For a 3MB audio file the total correction is around -127

Live Demonstration?

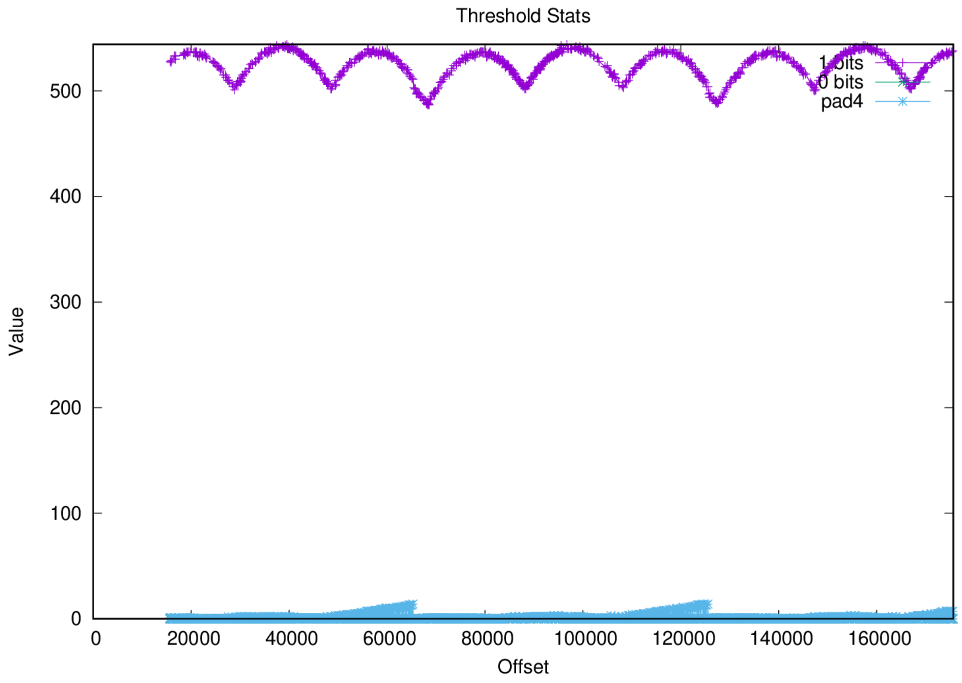
Live Demonstration?

Not yet

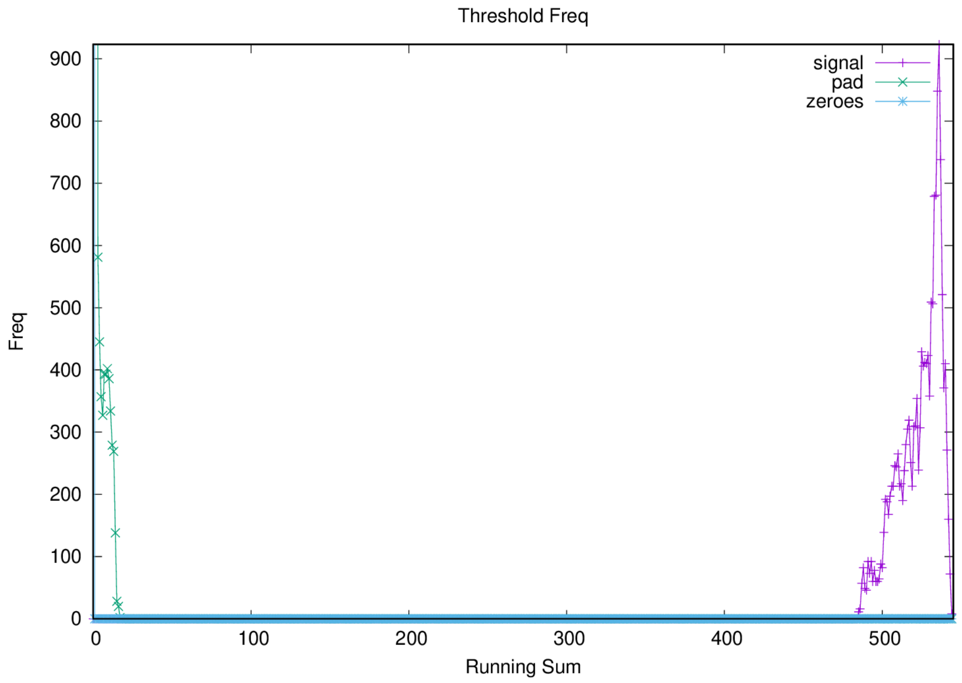
Thoughts

- current transfer rate 100 bits/s
- could double by adding 2nd channel
- might be able to scale pulse to pack in more than one bit
- could reduce width of pulse and padding
- could increase frequency for pulse and sampling
- code needs refactoring
- error detection via gzip of file before encoding

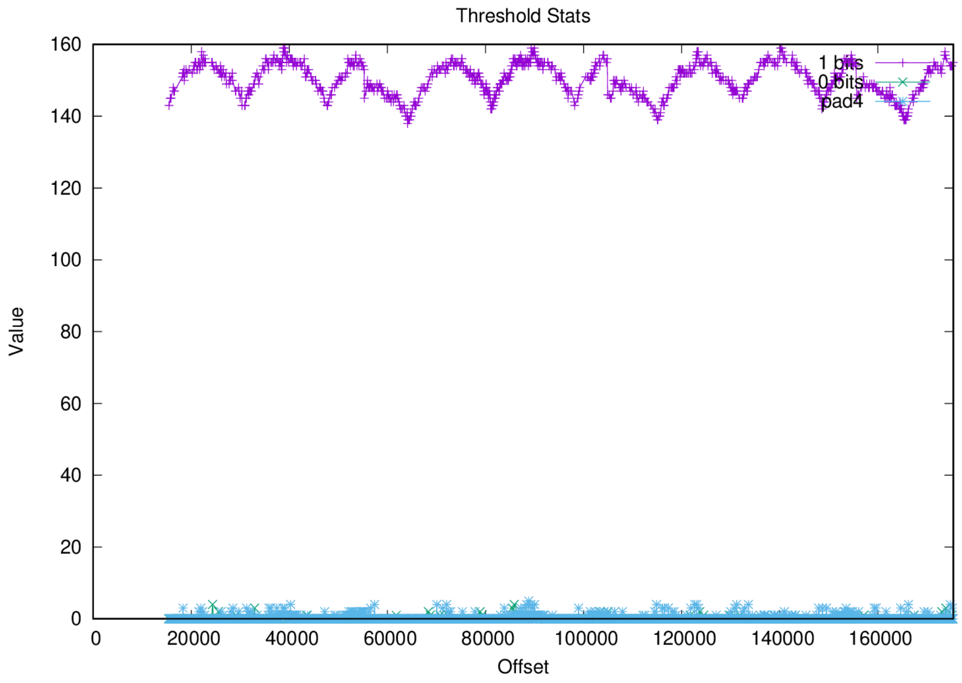
Variation in signal sum



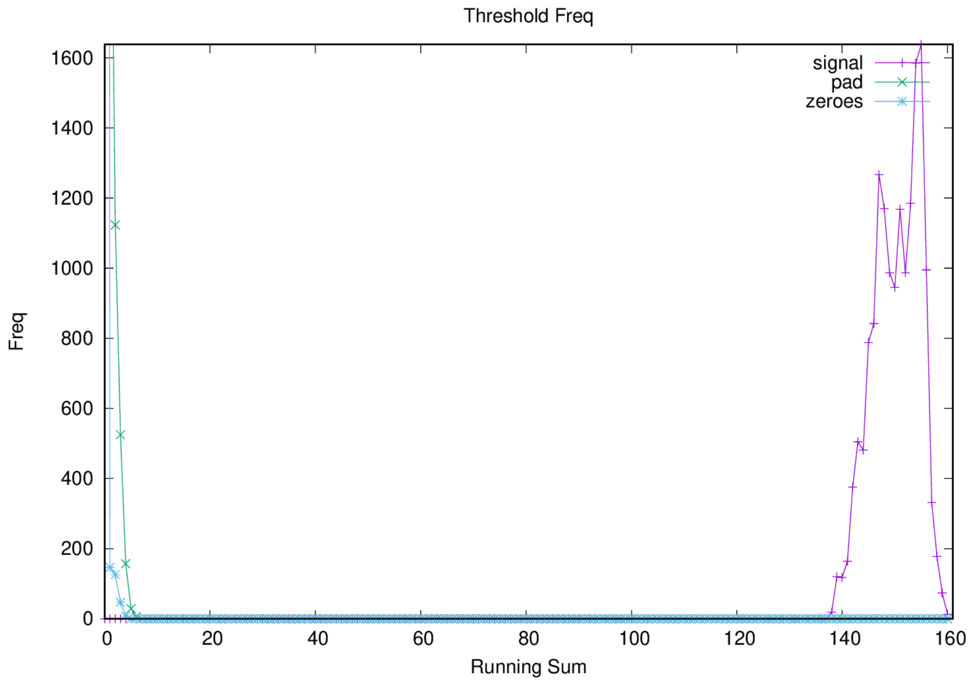
Variation of signal sum - histogram



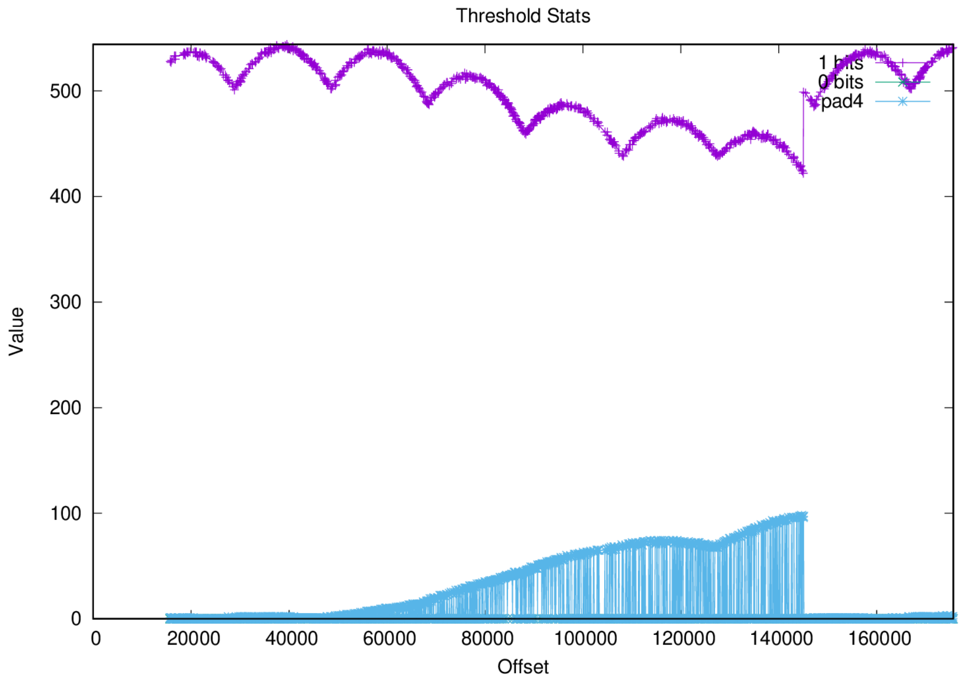
Lower volume signal



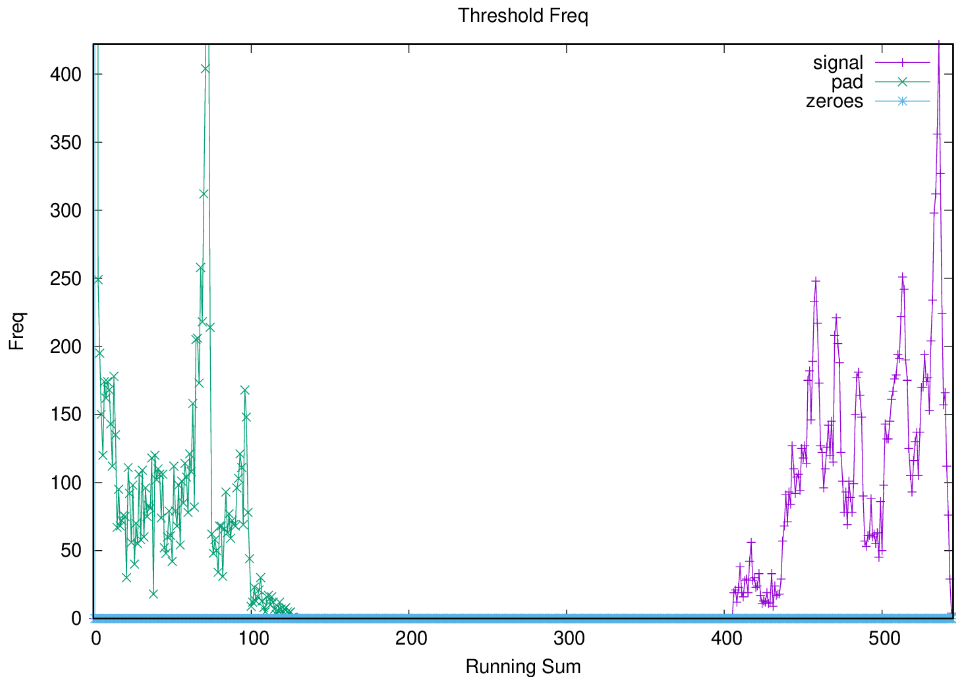
Lower volume - histogram



Loosened resync



Loosened resync - histogram



Reinvented soft modem but without 2 way communication and much poorer use of available bandwidth.

References

RIFF and WAV format

http://www-mmsp.ece.mcgill.ca/Documents/AudioFormats/WAVE/WAVE.html

https://docs.microsoft.com/en-us/previous-versions/dd757713(v%3dvs.85)?redirectedfrom=MSDN

https://github.com/HertzDevil/raw2wav

https://wiki.multimedia.cx/index.php/WAVEFORMATEX

https://wiki.multimedia.cx/index.php/Microsoft_Wave

https://wiki.multimedia.cx/index.php/RIFF