Decoding ALERT with your StormLink IQ Receiver

White Paper

James Logan
OneRain, Inc.

Decoding ALERT with your StormLink IQ Receiver

Background:

ALERT (Automated Local Evaluation in Real-Time) is a radio protocol developed in the late 1970s that transmits four byte packets via RF. The transmissions are ALOHA, which means that a site can transmit at any time without knowing if the data was received. This protocol has possibility of loss of data do to data collisions between transmitters that have overlapping transmissions.

ALERT uses FSK (Frequency Shift Keying), and in the U.S. the Marker tone is 2133 Hz, and the Space tone is 1920 Hz. Each transmission is approximately 380 milliseconds long made up of 250 ms for carrier lock and bit synch, followed by 233 ms for transmitting 4 bytes of data with a transmission rate of 300 baud. The 4 byte payload (32 bits) is made up of 8 format bits, 13 sensor ID bits, and 11 data value bits.

stop	0	1	A5	A4	A3	A2	A1	A0	start
stop	0	1	A11	A10	A9	A8	A7	A6	start
stop	1	1	D4	D3	D2	D1	D0	A12	start
stop	1	1	D10	D9	D8	D7	D6	D5	start

Figure 1 ALERT transmission and bit decoding format

The sensor ID, made up of 11 bits, can have a range of values between 0 and 8191. The data value, made up of 11 bits has a range of values between 0 and 2047.

Let's get started decoding ALERT.

What you need:

1. StormLink IQ Receiver – USB Dongle
2. Download the following open source packages for installation
    a. FM Decoder: rtl_fm
        i. http://kmkeen.com/rtl-demod-guide/index.html
    b. Audio processing: sox

        i. http://sox.sourceforge.net/
    c. Mini modem: minimodem
        i. http://www.whence.com/minimodem/
        ii. http://macappstore.org/minimodem/
3. Perl command line environment

Steps to get it running:

1. Find out what frequency ALERT traffic is being transmitted
    a. For UDFCD, it is 169.5 MHz for the ALERT gauge frequency
2. Install rtl_fm
3. Install sox
4. Listen to ALERT traffic - Mac

    rtl_fm -f 169.5M -M fm -s 24000 -r 24000 -l 170 - | play -r 24000 -t s16 -L -c 1 –

    a. The rtl_fm program converts the narrow band fm signal to a 16 bit, 24,000 samples/second audio stream that can be fed into an audio player “play” so that it can be heard.
    b. Some of the options for rtl_fm
        i. –f 169.5M = 169.5 MHz
        ii. –M fm = FM narrowband modulation
        iii. –s 24000 = sample at 24K sample rate
        iv. –r 24000 = resample output at 24K sample rate
        v. –l 170 = squelch level set to 170
        vi. - = send the output to stdout to be read by next program
    c. Some of the options for play
        i. –r 24000 = input samples at 24K
        ii. –t s16 = input data types as signed 16 bit
        iii. –L = little endian, byte ordering
        iv. –c 1 = 1 channel, mono
        v. - = read input from stdin
5. Listen to ALERT traffic – PC

    rtl_fm -f 169.5M -M fm -s 24000 -r 24000 -l 170 - | aplay -r 24k -f S16_LE

    a. The rtl_fm program converts the narrow band fm signal to a 16 bit, 24,000 samples/second audio stream that can be fed into an audio player “aplay” so that it can be heard.
    b. Some of the options for aplay
        i. –r 24k = input samples at 24K
        ii. –f S16_LE = data coming in at 16 bits per sample, little endian
6. Adjust the squelch for ALERT. It is the –l option in the rtl_fm command. Reduce the squelch until it is quiet, just below the background noise.
7. Install minimodem
8. To listen to and decode ALERT, use the following command line:

rtl_fm -p 0 -M fm -f 169.5M -s24k -l 175 | sox -t raw -r 24k -es -b16 - -t wav -es -b16 - | minimodem --rx --mark 2133 --space 1920 --binary-output --quiet 300 -f - | ../alert_bin.pl

a. The rtl_fm program converts the narrow band fm signal to a 16 bit, 24,000 samples/second audio stream that can be fed into sox which formats the audio as a wav file data stream. The wave file data stream is fed into minimodem which detects the ALERT RF transmission and outputs the raw binary alert data. The alert_bin.pl program takes the raw binary alert is converted to distinct ALERT ID, Value pairs.
b. Some of the new options for rtl_fm
    i. –p 0 = set the ppm frequency error to 0
c. Some of the options for sox to translate
    i. –t raw = raw input
    ii. –r 24k = 24k input sample rate
    iii. –es = signed data on input
    iv. –b16 = 16 bit data on input
    v. - = read from stdin
    vi. –t wav = output wav file
    vii. -es = output signed data
    viii. –b16 = 16 bit data on output
    ix. - = output to stdout
d. Some options for minimodem
    i. –rx = receive mode
    ii. –mark 2133 = mark frequency at 2133 hz
    iii. –space 1920 = space frequency at 1920 hz
    iv. –binary-output = output binary translated mark and space bits
    v. –quiet = don't report CARRIER/NOCARRIER or signal analysis metrics
    vi. 300 = BAUD mode 300 bps rate
    vii. –f - = read input from stdin

9. Example output from command line above

```
$ rtl_fm -p 0 -M fm -f 169.5M -s24k -l 175 | sox -t raw -r 24k  -es -b16 -
-t wav -es -b16 - | minimodem --rx --mark 2133 --space 1920 --binary-
output --quiet 300 -f - | ../alert_bin.pl
sox WARN wav: Length in output .wav header will be wrong since
can't seek to fix it
Found 1 device(s):
  0:  Realtek, RTL2838UHIDIR, SN: 00000001

Using device 0: Generic RTL2832U OEM
Found Rafael Micro R820T tuner
Tuner gain set to automatic.
Tuned to 169752000 Hz.
Oversampling input by: 42x.
Oversampling output by: 1x.
Buffer size: 8.13ms
Exact sample rate is: 1008000.009613 Hz
Sampling at 1008000 S/s.
Output at 24000 Hz.
1247   1023
3997   2020
7534   1599
5474   1893
1432   546
930   1418
4845   318
1826   454
203   900
5415   1652
7199   521
2119   133
1138   717
7341   1714
1002   430
3281   1899
588   1412
```

For the examples above, the first report ID is 1247 and the value is 1023. The second report ID is 3997 with a value of 2020 and so on.

Source Code:

The following is the Perl source code to decode ALERT

```
#!/usr/bin/perl

# A Simple ALERT Decoder that reads raw bytes from a stream
# James Logan
#

use strict;

my $fileName = shift @ARGV;
my $ifh;
my $is_stdin = 0;
my $oneByte = 0;
my $twoByte = 0;
my $threeByte = 0;
my $fourByte = 0;
my $id = 0;
my $value = 0;
my $ofh = *STDOUT;
my $unused_bytes = 0;

if( defined $fileName ){
  open $ifh, "<:raw", $fileName or die "Couldn't open $!\n";
} else {
  $ifh = *STDIN;
#   binmode( $ifh ) || die "cannot binmode STDIN";
  $is_stdin++;
}

while(<$ifh>)
{
 my $binLine = $_;  # read a line in at a time
 print $ofh "$binLine ";

 # Convert ascii to binary
 # Reverse the order of the bits
 $fourByte = 0;
 for( my $index = 7; $index >= 0; $index++ ) {
  if( substr($binLine, $index, 1 ) == "1" ) {
   $fourByte |= (0x01 << $index);
  }
 }

 print $ofh " $fourByte\n";
```

```
  # Check for the marker bits that confirm its an ALERT format message
  if( (($oneByte & 0xC0) == 0x40) &&   # 01000000
     (($twoByte & 0xC0) == 0x40) &&   # 01000000
     (($threeByte & 0xC0) == 0xC0) && # 11000000
     (($fourByte & 0xC0) == 0xC0) )   # 11000000
  {
   use integer;
   # Extract the ID using first thirteen bits 0 - 8191
   $id = ($oneByte & 63) + 64 * ($twoByte & 63) + 4096 * ($threeByte & 1);
   # Extract the value using remaining 11 bits
   $value = ($fourByte & 63) * 32 + (($threeByte & 62) >> 1);
   print $ofh "$id   $value \n";
   printf ("%8.8b %8.8b %8.8b %8.8b\n", $oneByte, $twoByte, $threeByte,
$fourByte );
   $unused_bytes = 0;
  } else {
   print $ofh " %8.8b \n", $oneByte;
  }
  $oneByte = $twoByte;
  $twoByte = $threeByte;
  $threeByte = $fourByte;
  $unused_bytes++;
}

close $ifh unless $is_stdin;
```