Two–Tone Pager Decoder Using Multimon

Overview

Alot of rural volunteer fire departments still rely on the Motorola two–tone sequential paging system and analog Motorola Minitor pagers for dispatching their crews to a fire scene. The standard "Motorola Quick Call 2" paging protocol consists of playing two separate audio tones, the "A" and "B" tone. The "A" tone is played first for one second, then the "B" tone for three seconds. Both of these tones are transmitted on the fire dispatch frequency (VHF usually) which the pager is tuned to. Inside the older Minitor pagers, a mechanical reed is used to filter and decode each of the proper tones. While this may sound primitive, it is actually very reliable. A modern tweak to this type of paging system would be for the fire dispatch page to also be sent to your computer or cellular phone via text or email message. That is what this project will attempt to cover, with the pager tone decoding being done in software instead of having to tie up an additional pager.

Actual Two–Tone Page at 398.1 and 912.0 Hz

For the tone decoding software, we will be using a slightly modified version of Thomas Sailer's `multimon` Linux radio transmission decoder, which is available at: www.baycom.org/~tom/ham/linux/multimon.html. This program uses a standard PC soundcard to acquire the signal from a radio scanner (or equivalent) and the decoding is done completely in software.

The modification will consist of replacing the DTMF tone values in `demod_dtmf.c` with tone values which are equal to that of the two–tone pager we wish to receive.

For example, in `demod_dtmf.c`, we see this matrix:

```
  DTMF Frequencies

       1209 1336 1477 1633
   697   1    2    3    A
   770   4    5    6    B
   852   7    8    9    C
   941   *    0    #    D
```

These are the standard 16–character DTMF tones we've known to love. The stock code array looks like this:

```
static const unsigned int dtmf_phinc[8] = {
        PHINC(1209), PHINC(1336), PHINC(1477), PHINC(1633),
        PHINC(697), PHINC(770), PHINC(852), PHINC(941)
};
```

As you can see, to decode a DTMF "1," the decoding routine looks for a simultaneous dual–tone value of 1209 Hz and 697 Hz in the incoming audio. Since two–tone pagers use *single* tones, and *not* DTMF tones, all we really need to do is replace both the "column" and "row" decoding array with the same tone value. Example:

```
static const unsigned int dtmf_phinc[8] = {
        PHINC(398), PHINC(912), PHINC(1477), PHINC(1633),
        PHINC(398), PHINC(912), PHINC(852), PHINC(941)
};
```

To decode a two–tone pager which uses a 398.1 Hz and 912.0 Hz tone, replace the "column" and "row" decoding structure with "398" and "912." This then maps a DTMF "1" to 398 Hz and the DTMF "5" to 912 Hz. Since the second tone is played for three seconds, you'll get three individual decodes of that tone.

Operation

Connect the audio output from a radio scanner or two–way radio tuned to the pager's dispatch frequency to the "line in" (`/dev/audio`) on the soundcard and wait for the proper two–tone page. An isolation transformer can be used to reduce any "hum" interference from ground loops. This is what you should see in `multimon`:

```
bash# ./multimon -a DTMF
multimod  (C) 1996/1997 by Tom Sailer HB9JNX/AE4WA
available demodulators: POCSAG512 POCSAG1200 POCSAG2400 AFSK1200 AFSK2400 AFSK2400_2 HAPN4800
 FSK9600 DTMF ZVEI SCOPE
Enabled demodulators: DTMF
DTMF: 1
DTMF: 5
DTMF: 5
DTMF: 5
```

When combined with an additional script or program which parses the output from `multimon`, you can then trigger a text or email message, or any other form of notification to be sent out.

A potential Perl script to trigger an external command would look something like the following code. This is just an example, and there are probably better methods (and better coders) than this.

```
-------------------------------------------CUT-------------------------------------------
#!/usr/bin/perl

# Path to multimon
$mm = "/usr/local/bin/multimon -q -a DTMF";

# Tone/DTMF string to trigger on
$on_str = "1555";

# Command or script to execute
$on_cmd = "echo Fire Fire Fire | mail -s Fire you@cellphone.com";

select STDOUT;
$| = 1;
$i = 0;

sub System {
if ((0xffff & system $args) != 0 ) {
  print STDERR "error: $!\n";
  exit 1;
 }
}

open M, "$mm |" || die "Can't open $mm: $!\n";

while (<M>) {

  ($a, $b) = split ':';
  $b =~ tr/0-9*#ABCD//csd; # Allow 0-9 * # A B C D
  $ans .= $b;
  $i++;

  if ($i == (length $on_str)) {

    if ($ans eq $on_str) {
      System($args = $on_cmd);
      undef $ans;
      $i = 0;
    }

    else {
      undef $ans;
      $i = 0;
    }
  }
}
-------------------------------------------CUT-------------------------------------------
```

Below is patch to `multimon` which adds a "quiet output" option to the DTMF decoding and also flushes `stdout` for better reliability when used in this application.

To apply the patch;

1. `tar xvzf multimon.tar.gz`
2. `patch -p0 < multimon.patch`

```
-------------------------------------------CUT--------------------------------------------
diff -ur multimon.orig/demod_dtmf.c multimon/demod_dtmf.c
--- multimon.orig/demod_dtmf.c  Mon Dec  8 10:56:06 1997
+++ multimon/demod_dtmf.c       Fri Sep 10 03:27:30 1999
@@ -25,6 +25,7 @@
 #include "filter.h"
 #include <math.h>
 #include <string.h>
+#include <stdio.h>

 /* ---------------------------------------------------------------------- */

@@ -137,6 +138,7 @@
                         i = process_block(s);
                         if (i != s->l1.dtmf.lastch && i >= 0)
                                 verbprintf(0, "DTMF: %c\n", dtmf_transl[i]);
+                                fflush(stdout);
                         s->l1.dtmf.lastch = i;
                 }
         }
diff -ur multimon.orig/gen.c multimon/gen.c
--- multimon.orig/gen.c Mon Dec  8 10:56:06 1997
+++ multimon/gen.c      Fri Sep 10 03:38:12 1999
@@ -367,7 +367,7 @@
 "  -s <freq>  : encode sine\n"
 "  -p <text>  : encode hdlc packet\n";

-void main(int argc, char *argv[])
+int main(int argc, char *argv[])
 {
         int c;
         int errflg = 0;
diff -ur multimon.orig/unixinput.c multimon/unixinput.c
--- multimon.orig/unixinput.c   Mon Dec  8 10:56:06 1997
+++ multimon/unixinput.c        Fri Sep 10 03:41:43 1999
@@ -370,12 +370,14 @@
 "(C) 1996 by Thomas Sailer HB9JNX/AE4WA\n"
 "  -t <type>  : input file type (any other type than raw requires sox)\n"
 "  -a <demod> : add demodulator\n"
-"  -s <demod> : subtract demodulator\n";
+"  -s <demod> : subtract demodulator\n"
+"  -q         : quiet output messages\n";

 int main(int argc, char *argv[])
 {
         int c;
         int errflg = 0;
+        int quiet = 0;
         int i;
         char **itype;
         int mask_first = 1;
@@ -383,16 +385,15 @@
         unsigned int overlap = 0;
         char *input_type = "hw";
```

```
-       fprintf(stdout, "multimod  (C) 1996/1997 by Tom Sailer HB9JNX/AE4WA\n"
-               "available demodulators:");
-       for (i = 0; i < NUMDEMOD; i++)
-               fprintf(stdout, " %s", dem[i]->name);
-       fprintf(stdout, "\n");
-       while ((c = getopt(argc, argv, "t:a:s:v:")) != EOF) {
+       while ((c = getopt(argc, argv, "t:a:s:v:q")) != EOF) {
                switch (c) {
                case '?':
                        errflg++;
                        break;
+
+               case 'q':
+                       quiet++;
+                       break;

                case 'v':
                        verbose_level = strtoul(optarg, 0, 0);
@@ -445,17 +446,33 @@

                }
        }
+
+  if (!quiet)
+    {
+      fprintf (stdout, "multimod  (C) 1996/1997 by Tom Sailer HB9JNX/AE4WA\n"
+               "available demodulators:");
+      for (i = 0; i < NUMDEMOD; i++)
+        fprintf (stdout, " %s", dem[i]->name);
+        fprintf (stdout, "\n");
+    }
+
        if (errflg) {
                (void)fprintf(stderr, usage_str);
                exit(2);
        }
        if (mask_first)
                memset(dem_mask, 0xff, sizeof(dem_mask));
+  if (!quiet)
+    {
+      fprintf (stdout, "Enabled demodulators:");
+    }

-       fprintf(stdout, "Enabled demodulators:");
        for (i = 0; i < NUMDEMOD; i++)
                if (MASK_ISSET(i)) {
-                       fprintf(stdout, " %s", dem[i]->name);
+                        if (!quiet)
+                          {
+                            fprintf (stdout, " %s", dem[i]->name);
+                          }
                        memset(dem_st+i, 0, sizeof(dem_st[i]));
                        dem_st[i].dem_par = dem[i];
                        if (dem[i]->init)
@@ -463,16 +480,24 @@
                        if (sample_rate == -1)
                                sample_rate = dem[i]->samplerate;
                        else if (sample_rate != dem[i]->samplerate) {
-                               fprintf(stdout, "\n");
-                               fprintf(stderr, "Error: Current sampling rate %d, "
-                                       " demodulator \"%s\" requires %d\n",
-                                       sample_rate, dem[i]->name, dem[i]->samplerate);
-                               exit(3);
-                       }
```

```
+
+                   if (!quiet)
+                       {
+                         fprintf (stdout, "\n");
+                         fprintf (stderr, "Error: Current sampling rate %d, "
+                                  " demodulator \"%s\" requires %d\n",
+                                  sample_rate, dem[i]->name, dem[i]->samplerate);
+                       }
+
+               exit(3);
+               }
                        if (dem[i]->overlap > overlap)
                                overlap = dem[i]->overlap;
                }
-       fprintf(stdout, "\n");
+       if (!quiet)
+         {
+           fprintf (stdout, "\n");
+         }

        if (!strcmp(input_type, "hw")) {
                if ((argc - optind) >= 1)
```

----------CUT----------