Linux FRS Radio Control

Overview

This is a simple hardware computer interface and Linux program which can be used to control the most commonly used buttons on any consumer Family Radio Service (FRS) or General Mobile Radio Service (GMRS) radio.

There are 22 available FRS/GMRS frequencies in the 462–467 MHz band. Each channel is also capable of 38 "private" subchannels. The subchannels are the result of using different Continuous Tone Coded Squelch System (CTCSS) tones, also referred to as "PL" tones, for each channel. By remotely controlling the **Mode**, **Up Arrow**, and **Push–to–Talk** (PTT) buttons, you can control just about any feature or function of the radio. This may be useful for anyone trying to build a store–and–forward "cellular" network using cheap FRS/GMRS radios (hint, hint). It's possible to control other (or more) buttons with trivial hardware and software modifications.

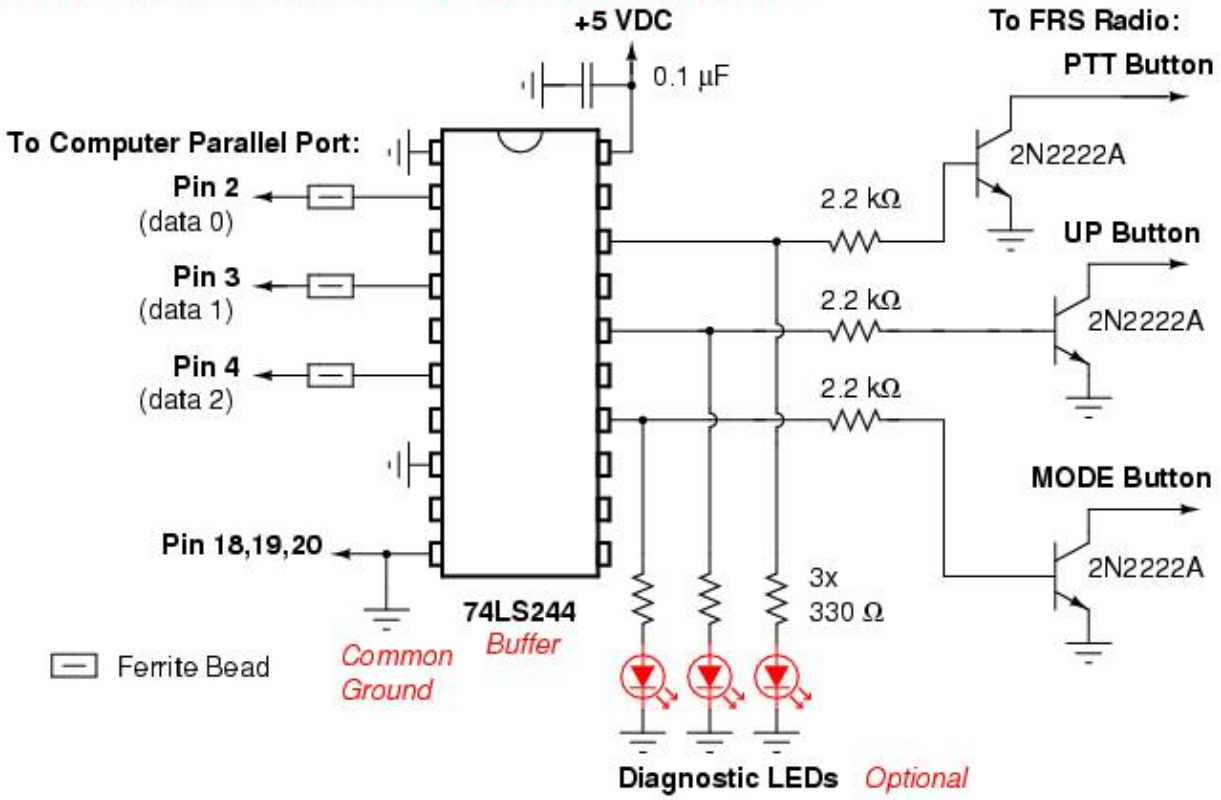
This particular example is built using a Cobra PR950DX FRS/GMRS radio (FCC ID: BBOPR950DXD) and a 74LS244 buffer/interface board. The radio's buttons are active–low (ground to enable) logic, so the hardware interface is quite simple. The computer software was written to run under the Linux operating system. It's not the best software in the world, but that's because I'm a fucking retard. I have no idea how to do this under Windows – or why you'd want to.

Other model radios should be quite similar. Check the FCC ID database to see if a schematic is available for the radio.

Computer Interface

The computer–to–radio interface hardware consists of parts (except the 74LS244) which are available at your local Radio Shack. The 74LS244 buffer IC is actually optional, as most computer parallel ports are limited to the amount of current they can source (output). This buffer will prevent any damage to your computer hardware should something short out, or if you need to control any additional hardware. The diagnostic LEDs are also optional, but are helpful to verify what the computer is outputting, data wise, at a quick glance.

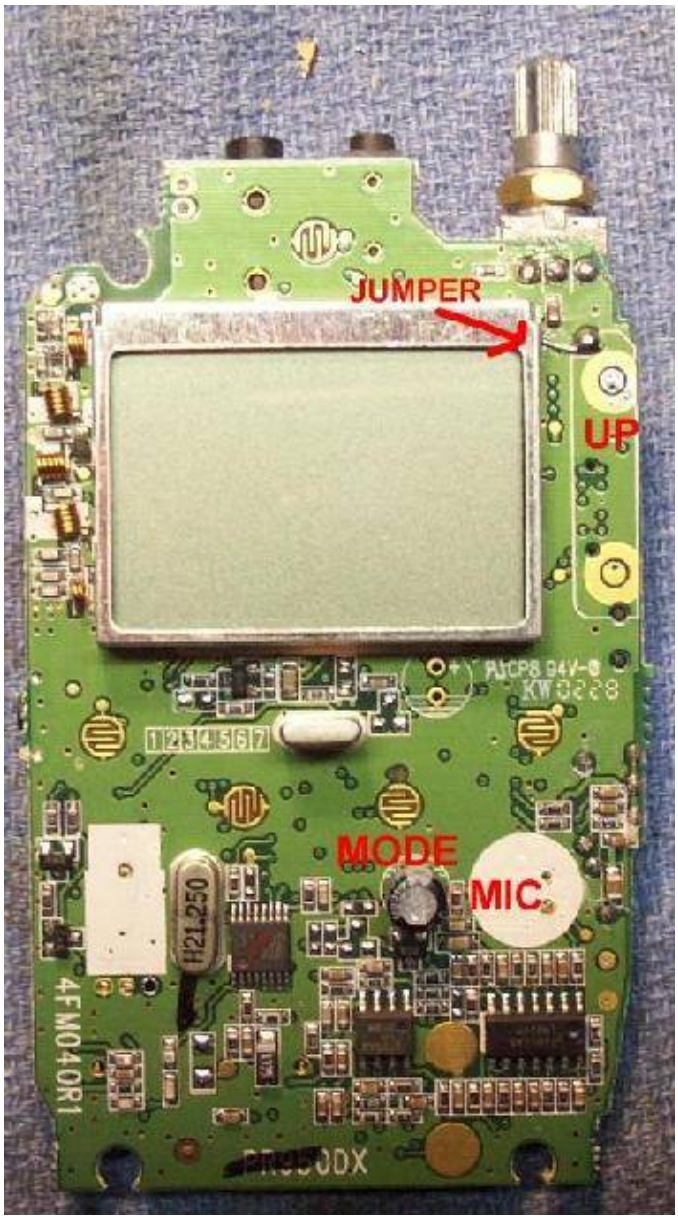
FRS Radio Control - Computer Interface



Pictures

Stock Cobra PR950DX overview.

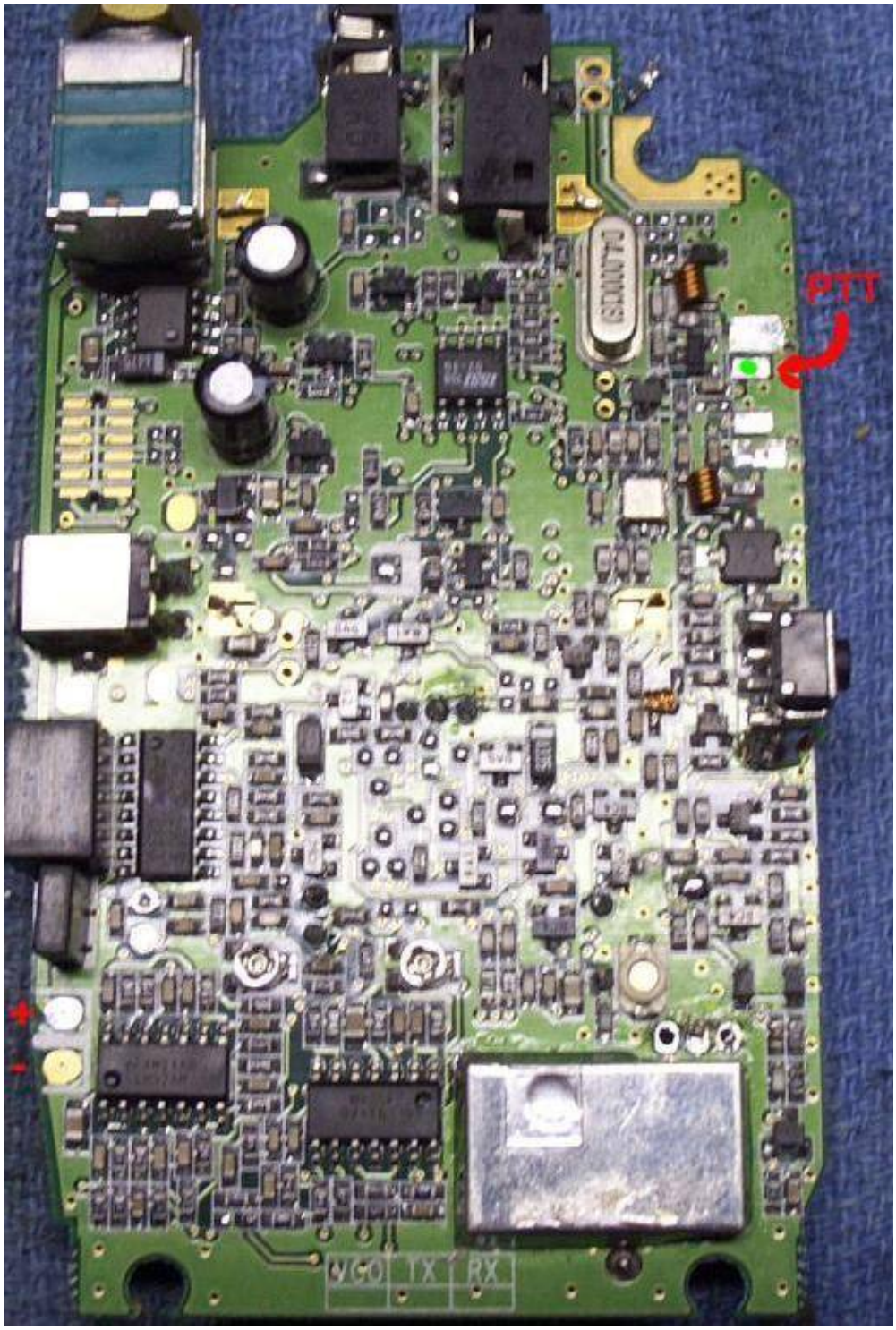
Buttons are: **Talk** / **PTT**, **Monitor** / **Light** (left side), **Call**, **Hi** / **Lo** / **Lock**, **Mode**, **Enter** (under LCD screen), **Up & Down** (right side).



Cobra PR950DX's internal front view. The motor/vibrator and electret microphone have been removed. Inject your own audio (say from another radio) into the microphone jack on the top of the radio. The **On**/**Off** switch on the volume potentiometer is locked in the **On** position by soldering a jumper wire across its terminals.

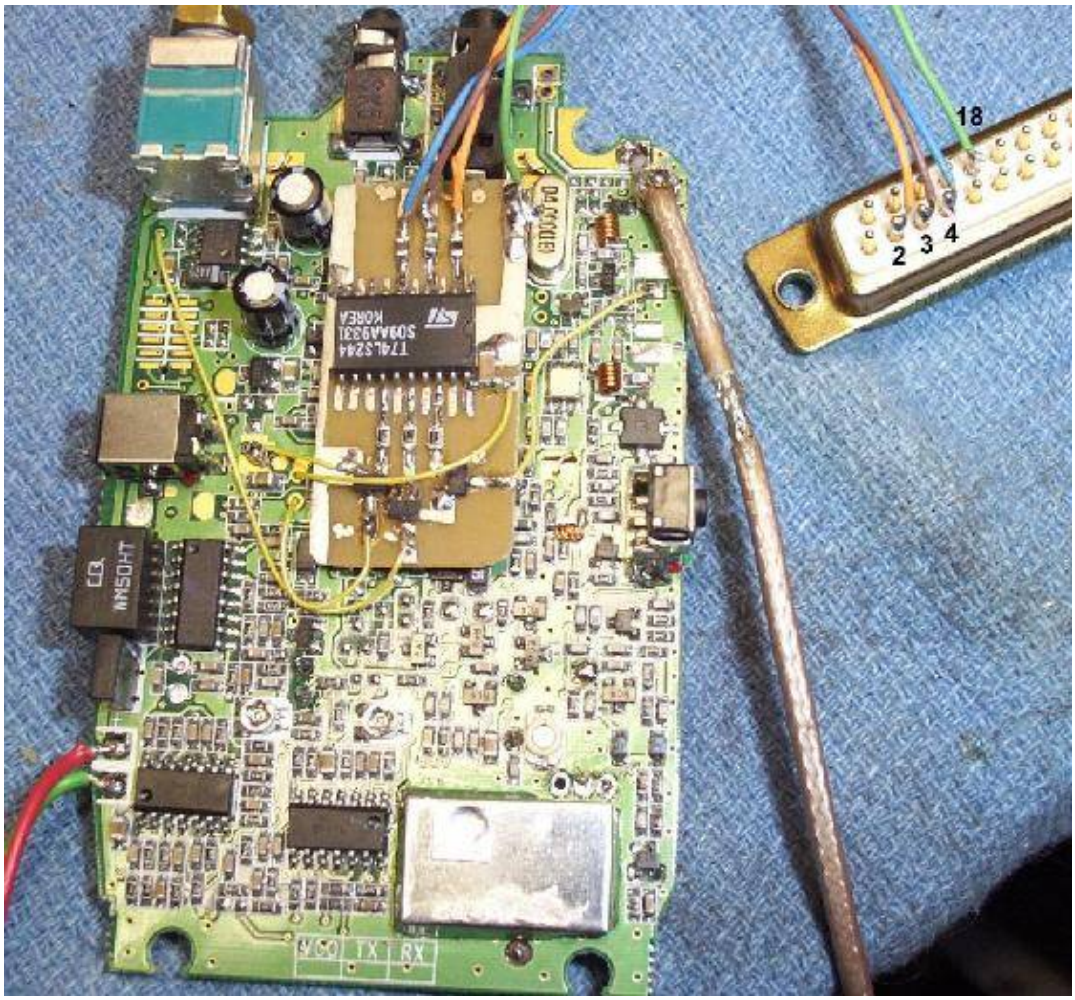
Alternate view. The **RED** dot shows where to solder the control wire from the computer interface board to control the **Mode** button.

The **BLUE** dot shows where to solder the control wire from the computer interface board to control the **Up Arrow** button.

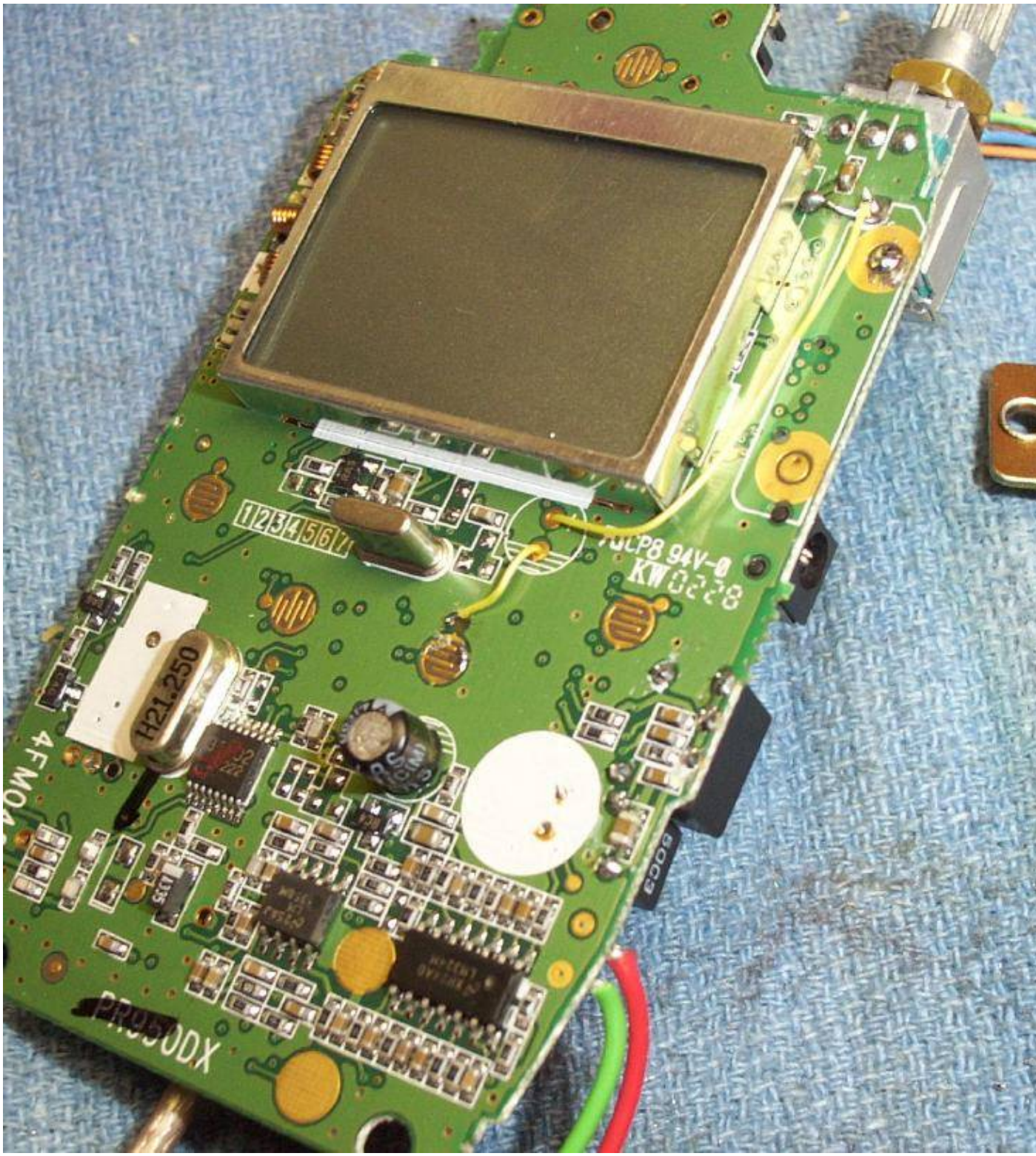


Internal rear view. The **PTT** button has been removed. The solder pad with the arrow is the new **PTT** switch. The + and – on the lower left is for the +6 VDC power supply input.

The **GREEN** dot shows where to solder the control wire from the computer interface board to control the **PTT** button.



Test setup showing the computer interface board mounted internally. The RF output (via a short BNC jumper) is taken from the stock antenna connection. The **BLACK** numbers are the parallel port's DB–25 pins. There should be a common ground between the radio and the computer. There are (optional) ferrite beads on the computer lines going into the 74LS244 buffer.



Front panel close–up.

Software

FRS/GMRS radio control software for Linux. Needs to run as `root` (userid = 0) for hardware port access.

Edit `frs_control.c` for your parallel port address. Default is `0x378`, which is `LPT1` on most computers.

```
---------- CUT & SAVE AS frs_control.c ----------

/*  Linux FRS Radio Control                                          */
/*  GBPPR 'Zine - Issue #13 - April 2005 - http://zine.gbppr.org     */
/*                                                                   */
/*  Compile using: gcc -O2 frs_control.c -o frs_control              */

#include <stdio.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <asm/io.h>

#define BASE_ADDRESS    0x378   /* 0x378 is 1st parallel port */
                                /* 0x278 is 2nd parallel port */

int c;
extern int errno;
                           /*       pin 432  on the parallel port */
unsigned char data_0 = 0;  /* Send 00000000b */
unsigned char data_1 = 1;  /* Send 00000001b */
unsigned char data_2 = 2;  /* Send 00000010b */
unsigned char data_4 = 4;  /* Send 00000100b */

void usage (char *);
int txtime, verbose;

int
main (int argc, char **argv)
{

  if (geteuid () && getuid ())
    {
      fprintf (stderr, "\n\n%s must be run as root to access hardware ports.\n", argv[0]);
      fprintf (stderr, "Leaving...\n\n");
      exit (1);
    }

  if (argc < 2)
    {
      usage (argv[0]);
    }

  while ((c = getopt (argc, argv, "t:rum")) != EOF)
    {
      switch (c)
        {
          case 't':
            txtime = atoi(optarg);
            TX ();
            exit (0);
          case 'r':
            RESET ();
            exit (0);
          case 'u':
```

```
            UP ();
            exit (0);
          case 'm':
            MODE ();
            exit ();
          default:
            usage (argv[0]);
        }
    }

  return (0);
}

int
TX (void)
{

  porttest ();

  outb (data_1, BASE_ADDRESS);
  sleep (txtime);
  outb (data_0, BASE_ADDRESS);
  usleep (200000);
  ioperm (BASE_ADDRESS, 3, 0);

  return (0);
}

int
RESET (void)
{

  porttest ();

  outb (data_0, BASE_ADDRESS);
  printf ("Parallel Port Reset: 0x%x hex\t(%d decimal)\n", BASE_ADDRESS, BASE_ADDRESS);
  printf ("Sending '00000000' ...\n");
  usleep (200000);
  ioperm (BASE_ADDRESS, 3, 0);

  return (0);
}

int
UP (void)
{

  porttest ();

  outb (data_2, BASE_ADDRESS);
  usleep (500000);
  outb (data_0, BASE_ADDRESS);
  usleep (200000);
  ioperm (BASE_ADDRESS, 3, 0);

  return (0);
}

int
MODE (void)
{

  porttest ();
```

```
  outb (data_4, BASE_ADDRESS);
  usleep (500000);
  outb (data_0, BASE_ADDRESS);
  usleep (200000);
  ioperm (BASE_ADDRESS, 3, 0);

  return (0);
}

int
porttest (void)
{

  if (ioperm (BASE_ADDRESS, 3, 1) < 0)
    {
      fprintf (stderr, "\nERROR: Can not open port 0x%x\n\n", BASE_ADDRESS);
      perror ("port");
      exit (errno);
    }

  usleep (200000);

  return (0);
}

void
usage (char *pname)
{
  fprintf (stderr, "\n\nUsage: %s\n\n", pname);
  fprintf (stderr, "\t-t x\tEnable Transmit for 'x' seconds\n");
  fprintf (stderr, "\t-r\tParallel port reset\n");
  fprintf (stderr, "\t-u\tEnable Up Arrow\n");
  fprintf (stderr, "\t-m\tEnable Mode Function\n");
  exit (1);
}

---------- CUT ----------
```

The PR950DX (or whatever radio you use) should be in a "clean" state when the software first starts. That is, all subchannels should be at "`00`" and the starting channel should be `1`. Otherwise, you'll get all confused. Have fun!

Use `frs_control -t x` to enable the **PTT** button (transmit) for "`x`" seconds.

Use `frs_control -r` to reset the parallel port to "all zeros" if the program crashes or doesn't exit right.

Use `frs_control -u` to enable the **Up Arrow** button.

Use `frs_control -m` to enable the **Mode** button.

Call the `frs_control` program from a Perl or shell script to interface it with audio recording/playing software or other programs.