```
/*
 *  derived from 441k512fr_latestc.
 *
 */


/*********tools installation:
        sudo apt-get install fftw3 fftw3-dev pkg-config
********************/

/********************
   to list record devices:
   arecord -l
   output display:
     **** List of CAPTURE Hardware Devices ****
     card 1: Direct [Samson Go Mic Direct], device 0: USB Audio
     Subdevices: 1/1
     Subdevice #0: subdevice #0

   "plughw:1,0" is for card 1 subdevice 0
********************/
/********************
   ALSA libraries, use compile flag: -lasound
   use compile flag: -lm   (for math library, math.h)
    gcc -lasound -lfftw3 capture1.c -o capture1.exe
********************/
/********************
   to restart alsa:
      alsactl restore
      sudo /etc/init.d/alsa-utils start
********************/
/********************
   adjust the playback volume:
   amixer -c 0 set PCM 4dB+
   adjust the capture volume:
   amixer -c 1 set Mic 100
   to play an S16 44100 raw file, type the following command line:
   aplay -D hw:0,0 -f cd record1.raw
********************/
```

```
/* ----------------------------------------
http://www.linuxjournal.com/node/6735
This example reads from the default PCM device
and writes to standard output for x seconds of data.
----------------------------------------*/


#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <sched.h>
#include <errno.h>
#include <getopt.h>
#include <pthread.h>
#include "/usr/include/alsa/asoundlib.h"
#include "/usr/include/alsa/control.h"
#include "/usr/include/alsa/pcm.h"
#include "/usr/include/fftw3.h"
//#include "/usr/include/iolib.h"

#include <sys/time.h>
#include <math.h>
#include <locale.h>


unsigned long SAMPLERATE = 44100L;  // 16k works
unsigned int g_samplerate;
int repeats = 45; // 60
int runtime;
int size;
int dir = 0;
float *buffer;
snd_pcm_t *handle = 0;
snd_pcm_hw_params_t *params = 0;

fftw_complex *fftw_in;
fftw_complex *fftw_out;
fftw_plan plan;
int captures = 0;

#define BINSIZE         512     // 16000/512 = 31.25
// frequency resolution: SR / binsize = 8000 / 512 = 15.625 Hz in each bin
// siren frequency 3125 Hz can be found in bin #200

snd_pcm_uframes_t frames;
int prepareDevice(void);

void prepare_fftw(void);
void destroy_fftw(void);
int compute_fftw(float* buffer);

void convert2complex(float* buf);
void ComplexFFT(void);
int pcm_detector(void);
```

```
////////////////////////////////////////////////////////////////////
int prepareDevice(void)
{
   int rc = -1;

   /* Open PCM device for recording (capture). */
   rc = snd_pcm_open(&handle, "plughw:1,0", SND_PCM_STREAM_CAPTURE, 0);

    if (rc < 0) {
        fprintf(stderr, "unable to open pcm device: %s\n",
            snd_strerror(rc));
        return rc;
     }

   /* Allocate a hardware parameters object. */
   snd_pcm_hw_params_alloca(&params);
   /* Fill it in with default values. */
   snd_pcm_hw_params_any(handle, params);

   /***** Set the desired hardware parameters. *****/

   snd_pcm_hw_params_set_access(handle, params,
                       SND_PCM_ACCESS_RW_INTERLEAVED);

   snd_pcm_hw_params_set_format(handle, params, SND_PCM_FORMAT_FLOAT);

   /* mono */
   snd_pcm_hw_params_set_channels(handle, params, 1);
   g_samplerate= SAMPLERATE;

   snd_pcm_hw_params_set_rate_near(handle, params, &g_samplerate, &dir);

   /* Set period size, 64 or 128 frames works.
        bits per sample for float type: 32 bits
        1 frame time: 1/SR * bits per sample
               = 1/8000 * 32 = 125 usec (or sample time).
    */
  frames = 512; // 1 period time = frames * frame time = 8 msec.
  snd_pcm_hw_params_set_period_size_near(handle,
                              params, &frames, &dir);

  /* Write the parameters to the driver */
  rc = snd_pcm_hw_params(handle, params);
  if (rc < 0) {
    fprintf(stderr, "unable to set hw parameters: %s\n",
            snd_strerror(rc));
    return rc;
  }

   /* Use a buffer large enough to hold one period */
  snd_pcm_hw_params_get_period_size(params, &frames, &dir);
  size = frames * 4; /* 2 bytes/sample, 2 channels */
  buffer = malloc(size);
  if (buffer == 0)
     printf("malloc failed\n");

}
```

```
////////////////////////////////////////////////////////////
int pcm_detector(void)
{
   unsigned long loops = 125;
   int rc;

  /* We want to loop for few seconds */
  snd_pcm_hw_params_get_period_time(params, &g_samplerate, &dir);
                // also same as the designed 16000
  printf("buffer size: %d, actual SR: %d\n", size, g_samplerate);
  // sampling bits per seconds: 1/16000 = 0.0000625
  runtime = (int)(0.000022675 * frames * repeats * loops);
  printf(" loops: %u, runtime: %u seconds\n", loops, runtime);

  while(repeats-- > 0)
  {
        loops = 50;  // 125, must sample it frequent enough
        while (loops > 0) {

        loops--;
        rc = snd_pcm_readi(handle, buffer, frames);
        /* EPIPE means overrun */
        if (rc == -EPIPE) {
                fprintf(stderr, "overrun occurred\n");
        snd_pcm_prepare(handle);
        } else if (rc < 0) {
        fprintf(stderr, "error from read: %s\n",
                snd_strerror(rc));
    } else if (rc != (int)frames) {
        fprintf(stderr, "short read, read %d frames\n", rc);
        }

        if (filedesc > 0)
        {
                rc = write(filedesc, buffer, size);
        }
        if (rc != size)
                fprintf(stderr, "short write: wrote %d bytes\n", rc);
  }

  compute_fftw(buffer);

 }
   repeats = 45;        // reset for the next round

   printf("buffer size: %d, actual SR: %u\n", size, g_samplerate);

   snd_pcm_drain(handle);
   snd_pcm_close(handle);

   return rc;

}


////////////////////////////////////////////////////////////
```

```
void prepare_fftw(void)
{
    int i = repeats;
    fftw_in = 0;

    fftw_in = fftw_malloc (sizeof(fftw_complex)* 4096 ); // 16000
    fftw_out = fftw_malloc (sizeof(fftw_complex) * 4096 );// 8000

    // use FFT size (bins) of 512
    plan = fftw_plan_dft_1d ( BINSIZE, fftw_in, fftw_out,
                              FFTW_FORWARD, FFTW_ESTIMATE );
}

////////////////////////////////////////////////////////////
void destroy_fftw(void)
{
  fftw_destroy_plan(plan);
  fftw_free ( fftw_out );
  fftw_free ( fftw_in );
}

////////////////////////////////////////////////////////////
int compute_fftw(float* buffer)
{
  unsigned long   fundamental_freq = 0, bin = 0;
  int i;

  float amplitude = 0.0f, magnitude_dB = 0.0f;

  for (i = 0; i < 4096; i++)     // 1024, 8000
  {
    fftw_in[i][0] = buffer[i];
    fftw_in[i][1] = 0;
  }
  for (i = 0; i < 4096; i++)     // 2048, 4000
  {
    fftw_out[i][0] = 0;
    fftw_out[i][1] = 0;
  }

  fftw_execute (plan);

  for(i=2; i<= 512; i+=2) // the actual sample rate used by the device
  {
    if(sqrt(pow(fftw_out[i][0],2)+pow(fftw_out[i][1],2)) >
       sqrt(pow(fftw_out[bin][0],2)+pow(fftw_out[bin][1],2)) )
       {
          bin=i;
          amplitude = sqrt(pow(fftw_out[i][0],2)+pow(fftw_out[i][1],2));
       }
  }

  //printf ( "\nfftw_out:  %5f  %5f\n", fftw_out[4][0], fftw_out[100][0] );
  // bin index points to the fundamental_freq:
  // find the bin index for fo of 3kHz:
  // fo * bin size / SR = 3000 * 512 / SampleRate
  // bin index = 96
```

```
  // or fo = SampleRate * bin / 512
  i = (int)(bin/2);

  //bin = (unsigned long)(floor((float)bin/2));
  //printf("\nfrequency in bin: %lu\n", bin);
  fundamental_freq = (unsigned long)(SAMPLERATE * bin / BINSIZE);
  magnitude_dB = 20*log10(amplitude);
  printf("frequency: %lu, amplitude: %2.5f, %2.5f dB\n",
          fundamental_freq, amplitude, magnitude_dB);

  if (amplitude > 2)
  {
     if( (fundamental_freq > 3050 && fundamental_freq < 3200) ||
         (fundamental_freq > 6050 && fundamental_freq < 6300) )
     {
        captures++;
     }
  }
  return;
}       // compute_fftw



////////////////////////////////////////////////////////////
void main(void)
{
  int i = repeats, err = -1;

  prepareDevice();
  prepare_fftw();
  pcm_detector();
  destroy_fftw();
  free(buffer);
  printf("run %d times. captures: %d in %u seconds\n",
          i, captures, runtime);

  if (captures > 9)
        system("echo 1 > /sys/class/gpio/gpio60/value");
   else
     system("echo 0 > /sys/class/gpio/gpio60/value");


  return;
}
```