Hae Chan Park

Chromatic Tuner

Introduction / Summary:

The goal of this project was to create a chromatic tuner. This implementation is able to identify music notes and deviation in cents of real-time audio signals and display information to the user with a simple UI. This involves sampling the audio signal and performing FFT analysis to extract the frequency components and highlight the dominant frequency of the note. We interface our on board microphone for capturing audio, LCD for user display, and our encoder and on-board buttons for user input to navigate the LCD UI and tune the frequency.

Hardware / Peripherals:

- microphone (on-board): capturing audio signals for processing
- LCD Display (ILI9341 display): display information about the note to the user.
- Rotary encoder: used to navigate to the tuning page where it would take user input to control/tune the frequency for the note
- buttons on the FPGA: used to navigate between the default window/page and the debug page,hosting the histogram for the FFT to visualize the data.

Design Overview:

1) ***Audio Sampling*** - stream_grabber.c, main.c
   Key functions:
    - stream_grabber_start()
    - stream_grabber_read_sample(unsigned which_sample)
    - stream_grabber_wait_enough_samples(unsigned required_samples)
    - stream_grabber_samples_sampled_captures()

```
void read_fsl_values(float* q, int n) {
          int i;
          unsigned int x;
          stream_grabber_wait_enough_samples(1);
          for(i = 0; i < n * REDUCER; i+=REDUCER) {
            int_buffer[i/REDUCER] = stream_grabber_read_sample(i);
            x = int_buffer[i/REDUCER];
            q[i/REDUCER] = 3.3*x/67108864.0;
          }
        }
```

We use the start method from our stream_grabber.c file and our read_fsl_values method defined in our main.c file to collect samples from the stream grabber and convert them to voltage and store in q for FFT processing.

```
stream_grabber_start();
read_fsl_values(q, SAMPLES);
```

2) ***Frequency Analysis*** - fft.c, main.c, complex.c
We used a Fast Fourier Transform (FFT) to convert time-domain signals into the frequency-domain. Our approach is optimized using lookup tables (LUTs) for sine and cosine values. These values are initialized in main.c prior to the first execution of our FFT method from fft.c.

```
k=0;
    for (j=0; j<m; j++){
        for(i=0; i<n; i+=2){
            if (i%(n_over_b[j])==0 && i!=0)
                k++;
            float sin_k_j = sinVals(k, j);
            float cos_k_j = cosVals(k, j);
            float q_iplus = q[i+1];
            float w_iplus = w[i+1];
            real=mult_real(q_iplus, w_iplus, cos_k_j, sin_k_j);
            imagine=mult_im(q_iplus, w_iplus, cos_k_j, sin_k_j);
            new_[i]=q[i]+real;
            new_im[i]=w[i]+imagine;
            new_[i+1]=q[i]-real;
            new_im[i+1]=w[i]-imagine;
        }
        for (i=0; i<n; i++){
            q[i]=new_[i];
            w[i]=new_im[i];
        }
```

By simply indexing sinVals and cosVals LUTs we speed up our computation by now having to calculate this each iteration. We also pulled out mult_real() and mult_imag() methods in the complex.c file to compute both parts separately. We select the highest magnitude component to be the dominant pitch of the audio sample. This file handles our method for plotting the FFT histogram as well.

3) ***Note Detection*** - note.c
Maps the dominant frequency computed by the FFT to the closest musical note and calculates the deviation from the actual frequency in cents.

Key functions:

- findNote(float freq)
- printNote(char* note)
- printFreq(char* freq)
- printCents(char* cents)
- printbar()
- printNewBase()
- clearPage()

The findNote method takes our dominant frequency and determines which octave it is in and computes the closest musical note.

```
c_note = a4 * pow(2, -0.75);
  if(freq >= c_note) {
    while(freq > c_note*2.0) {
      c_note=c_note*2.0;
      oct++;
    }
  }
  else {
    while(freq < c_note && oct-1 >= 0) {
      c_note=c_note/2.0;
      oct--;
    }
  }
  r=c_note*root2;
  while(freq > r) {
    c_note=c_note*root2;
    r=r*root2;
    note++;
  }
  if((freq-c_note) <= (r-freq)) { // left note
    deviation = log2(freq/c_note)*1200 ;
  }
  else { // right note
    note++;
    if(note >=12) note=0;
    deviation = log2(freq/r)*1200;
  }
```

This also handles methods for updating the LCD to update the page displayed to the user.

4) ***Display*** - lcd.c, fonts.c, note.c, main.c

   Display changes can be triggered by the findNote method in note.c as well as user interaction to switch pages. The lcd.c file manages the graphical rendering for our user interface. Scalable fonts are predefined. This handles displaying the note (with octave), frequency, cents, and a bar graph to visualize accuracy. In the main.c file we handle our state machine for the pages / modes.

   We have three pages / modes:

   - Default - display note information computed by findNote method
   - Encoder - adjust base frequency with rotary encoder
   - Debug - displays fftHistogram

5) ***User interaction*** - bsp.c, extra.c

   Manages our user inputs for the rotary encoder and on-board buttons to navigate pages and adjust base frequency via interrupt-based controls. TwistHandler(void *CallbackRef) method for rotary encoder functionality to adjust the base frequency in 1Hz increments and shifts note mapping to adjust. This method also handles our encoder state machine. These two files ensure our hardware features such as timer, interrupts, and GPIO behave as we designed.

main.c Workflow

- First initialize hardware and LUTs for FFT processing
- Get initial audio sample from stream_grabber
- Enter main execution loop (while(1) { … })
- Repeated steps:
  1) freq = FFT(sample q)
  2) Get next sample q ← q_next
  3) Case (page)
     Default: findNote(freq)
     Encoder: printNewbase()
     Debug: fftHistogram()

Testing and Validation

Testing for this project was aimed at ensuring the chromatic tuner functioned reliably and accurately within the desired frequency range. Key tests included:

- ***FFT Visualization:*** This feature was tested by verifying the frequency spectrum display, ensuring that it correctly represented the frequency components of the input signal. The FFT bins were checked for accuracy and appropriate spacing, particularly at higher frequencies.

- ***Diagnostic Data:*** The system outputs various diagnostic data, such as the sampled values, FFT bin spacing, and potential aliasing effects, to verify the integrity of the signal processing stages.
- ***Log History:*** A logging mechanism was implemented to track the state of the system during signal processing. This history helped to pinpoint any discrepancies or errors in the computations and verify that all values were updated as expected.

Testing was done via phone and portable speakers. Tones were generated off of a website, not the one given in the handout but a similar generator. The speaker was placed near the on-board microphone. Our implementation was to be able to reliably read upwards of 6kHz and cover the required range from E2 to C7. Testing as such was mostly using signals in this required range {82- 4200} Hz
Procedure:

- Play tones at various frequencies
- Observe detected information displayed on LCD
- Tune adjustment using feedback from our accuracy bar on the display

Design Challenges:

1) **Efficient Real-time Signal Processing:** The most critical challenge in this project was processing audio signals in real-time. The Fast Fourier Transform (FFT) was optimized using Look-Up Tables (LUTs) to speed up the computation. Signal decimation was also employed to reduce the number of samples processed, ensuring the system could handle incoming audio data. Achieving real-time performance was essential to provide accurate and immediate feedback to users.
2) **Low-frequency Detection:** Accurately detecting low-frequency tones posed a challenge due to the limited resolution of the FFT. Zero-padding was applied to enhance the frequency resolution, ensuring that even subtle variations in low-frequency notes could be detected and displayed with precision. This technique allowed the system to distinguish between nearby pitches, ensuring high precision.
3) **Handling Button Mashing and User Interface Stability:** A unique challenge was ensuring the system remained stable during intense user interaction, particularly with the rotary encoder and button presses. To mitigate potential issues like LCD freezes caused by rapid page switching or excessive button mashing, a ClearPage() method was implemented. This method effectively provided a "soft reset" by clearing the screen and resetting the page state every time a new page was loaded. This approach handled most instability cases and improved the overall user experience by ensuring the system remained responsive.

Conclusion

The chromatic tuner implements audio-signal processing for note detection, feedback, and tuning in real-time. By combining the FFT, a simple UI, and intuitive controls we realize an embedded system that meets design objectives and is tested to ensure robust and precise operation.