Project 05 - FFT LCD Display

Introduction to Embedded Systems - University of Nebraska

Zach Swanson

Contents

1 Introduction

The purpose of this project was to introduce the fast Fourier transform (FFT) using the TI eZDSP5535 development board (eZDSP). A 256-point FFT was implemented for this project. The project required the use of BIOS to perform the operations of receiving audio, filtering audio, performing the FFT on the filtered audio signal, transmitting the filtered audio, and displaying the results of the FFT to the LCD display. Emphasis was placed on the quality of the project demonstration, i.e. sound and display quality. In order to achieve a quality display, a key challenge for this project was improving the frame rate for writing to the display.

2 Project Description

2.1 Filter Design

As in past projects, a low- and a high-pass filter were designed and implemented for filtering audio samples. The cutoff frequency of both filters were increased to provide a larger range of frequencies for demonstrating the display for low frequency tones. The filters' magnitude responses are shown in Figure 1. Both filters were 127th order.

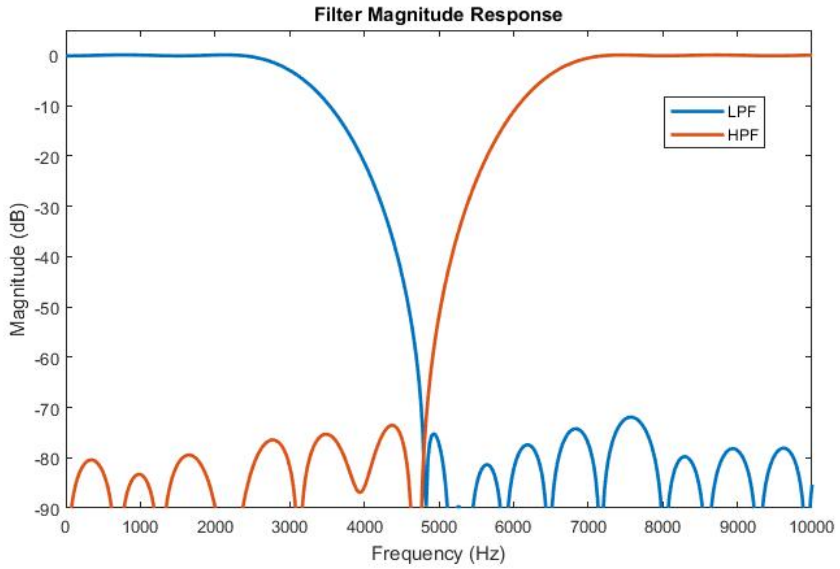

Figure 1: Magnitude response of low- and high-pass filters

2.2 Program Description

2.2.1 Audio Handling

The program contained two hardware interrupts, two tasks, two idle threads and two mailboxes. Figure 2 illustrates the overall flow of the program in a block diagram. Audio was sampled at a rate of 48 kHz by the stereo-in ADC. The I2S receive hardware interrupt was used to collect samples using a ping-pong array. Every 48 samples collected, the hardware interrupt posted the samples to a mailbox. The mailbox, MBXAudio, featured two messages, each containing 48 elements. The mailbox was read by the audio filtering task. The task execution was suspended

until a message was available; then, the samples were dumped into an array that was passed to a previously designed FIR filter function. The FIR function output the filtered audio samples to another ping-pong array. The ping-pong array was read by the I2S transmit hardware interrupt, which fed the filtered samples to the stereo-out DAC to play the audio to the listener. The transmit ping-pong array elements were also copied to an array to be passed to a second task that performed the FFT.

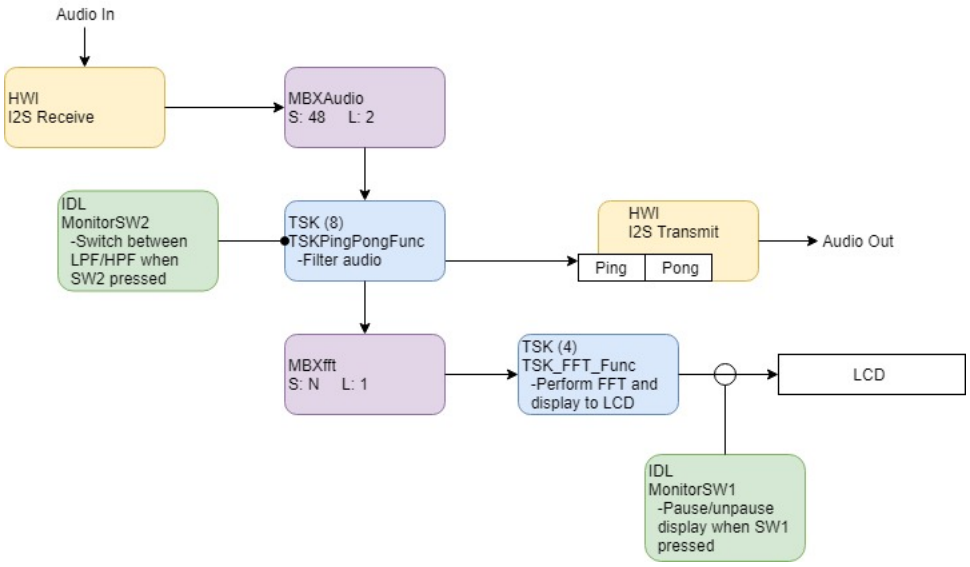

Figure 2: Block diagram of program flow.

2.2.2 FFT Calculation

The FFT is a transform that takes a discrete time signal and generates the signals frequency response. It is defined as being an "N-point" transform, where N is the number of frequency bins generated by the transform. The FFT output may also be defined by the window size, L, of the input data. For simplicity, N and L were kept equal to each other throughout the project, such that the computational and frequency resolution were always equal.

The rfft function from the TI-provided DSP library was used to implement the FFT in this program. To perform a N-point FFT, the rfft function required N data samples. Therefore, the audio filtering task had to copy N filtered samples before posting to a second mailbox, MBXfft, that consisted of one message compromised of N elements. Similar to the filtering task, the FFT task was suspended until a message was available. Once a message was received, the rfft function was called. The rfft function generated complex output in the input data array; hence, the function generated half the output of a N-point FFT. However, only half of the output was needed because the FFT is a symmetric function around the Nyquist freqency. Because the output was in complex format the magnitude of the frequency spectrum was calculated. Listing 1 shows the the rfft call and the calculation of the magnitude response.

For this project, a 256-point FFT was performed. The value 256 was selected because the rfft function generated 128 output data points. Therefore, at 128 points, the entire 96 pixel columns of the LCD display could be utilized to display the majority of the frequency spectrum.

```
1 MBX_pend(&MBXfft, &data1, SYS_FOREVER);
2 max = 0;
```

```
3  rfft(data1, N, SCALE);
4
5  fftMag[0] = (int16_t)(sqrt( pow((double)(data1[0]), 2) ) );
6  fftMag[NBY2] = (int16_t)(sqrt( pow((double)(data1[1]), 2) ) );
7
8  for(i = 1, j = 2; i < NBY2; i++, j += 2)
9  {
10         fftMag[i] = (int16_t)(sqrt( pow((double)(data1[j]), 2) + pow((double)(data1[(j +
               1)]), 2) ) );
11
12         if(fftMag[i] > max)
13         {
14                 max = fftMag[i];
15         }
16 }
17
18 IDL_run();
19 if(!displayFreeze)
20 {
21         fftDisplay(fftMag, NBY2, max);
22 }
```

Listing 1: FFT and magnitude calculation

2.2.3 FFT LCD Display

After the FFT and the magnitude response were calculated, the magnitude data was passed to a function that calculated pixel position and wrote the positions to the LCD display. As Listing 1 shows, a maximum value was determined and passed to the display function. This was done to ensure that the magnitude display could be easily seen regardless of the relative magnitudes of different sample sets. Listing 2 shows the code for the function that wrote the FFT magnitude data to the display and 3 shows the TI multi-send command for the LCD display. As shown, the fftDisplay function used the same I2C_write function as TI's multi-send function, but fftDisplay reduced the wait time by a factor of 100 which helped improve frame rates. Another frame rate improvement method was changing the addressing mode to vertical addressing mode, which wrote to the top and bottom page of the screen while moving horizontally across the screen.

```
1  void fftDisplay(int16_t * samples, int numSamps, int16_t maxVal)
2  {
3          int i, j;
4          Uint16 bot, top, cmd[193];
5
6          cmd[0] = 0x40 & 0x00FF;
7
8          for(i = 0, j = 1; i < 96; i++, j+=2)
9          {
10                 pixLoc(samples[i], &bot, &top, maxVal);
11
12                 cmd[j] = bot;
13                 cmd[(j + 1)] = top;
14         }
15
16         EZDSP5535_waitusec( 10 );
17         EZDSP5535_I2C_write(0x3C, &cmd[0], 193);
18 }
```

Listing 2: LCD display function

```
1  Int16 EZDSP5535_OSD9616_multiSend( Uint16* data, Uint16 len )
2  {
3      Uint16 x;
4      Uint16 cmd[10];
```

```
5       for(x=0;x<len;x++)                  // Command / Data
6       {
7           cmd[x] = data[x];
8       }
9       EZDSP5535_waitusec( 1000 );
10      return EZDSP5535_I2C_write( OSD9616_I2C_ADDR, cmd, len );
11  }
```

Listing 3: TI LCD multi-send function

2.2.4 Idle Threads

As in previous projects, two idle threads were used to monitor switch one and two strokes. Similar to past projects, whenever a switch two stroke occurred the audio filter was switched from low- to high-pass or vice-versa. When a switch one stroke was detected, a state variable, displayFreeze, was changed to pause or unpause the display. Listing 1 shows how the state variable was used to control the LCD display.

3 Results

3.1 Display

The best frame rate achieved was approximately 53 frames per second. As stated previously, the frame rates were improved by drastically reducing the time waiting before the I2C_write call and by switching to vertical addressing mode. Additionally, to enhance the demo the pixels from the axis of the LCD graph to the actual data value pixel were illuminated to provided a better magnitude profile for the viewer. The only drawback of the screen was the screen provided 96 pixels along the x-axis. However, by performing a 256-point FFT, 128 pixels were needed to represent the positive frequency spectrum from DC to Nyquist. Hence, the screen in the demo displayed frequencies from zero to 18 kHz. Since the audio used for the demo would not likely exceed the upper bound, it was deemed acceptable.

3.2 FFT Results

Figures 3 and 4 show a comparison of the MATLAB and TI FFT results for the same input at 1,125 Hz and 5,000 Hz, respectively. As illustrated, the TI rfft results were similar to the MATLAB fft results, especially for the 1,125 Hz tone. The FFT on the 1,125 Hz tone was better for both FFT implementations because of the frequency resolution, which was equal to 187.5 Hz. Because 1,125 is a multiple of 187.5, it lines up with one of the transform output bins. On the other hand, the 5,000 Hz tone was not a multiple of 187.5 and spreading of frequency near 5,000 Hz was observed as a result. It was also observed that the peak wasn't exactly at 5,000; rather, it was shifted to the next nearest multiple of 187.5 Hz, 5,062.5 Hz.

3.3 CPU Usage

To accurately measure the CPU usage of the tasks the HWI_disable/HWI_restore and TSK_disable/TSK_enable commands were used to suspend unwanted hardware interrupts and tasks. For the audio sampling task, the cycles were counted from one mailbox pend to the next. The audio sampling task takes 12,032 cycles when using a 128 coefficient filter. Filtering 48-sample frames, the CPU usage for the task was 12%. For the FFT task, the cycles were counted from the mailbox pend to a dummy line of code at the end of the task. The FFT task took 3,019,811 cycles to complete.

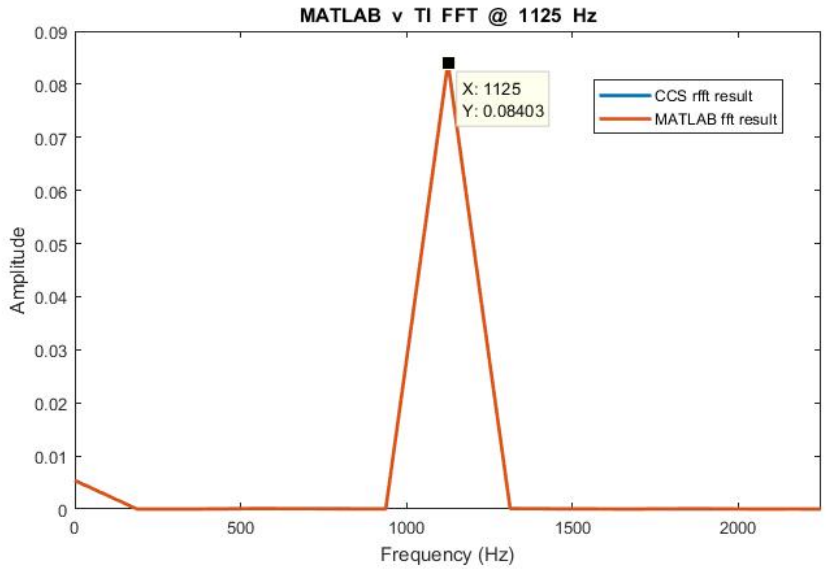

Figure 3: MATLAB and CCS comparison of FFT results at 1125 Hz Tone

Hence, when the task was running, it was consuming 100% of the CPU time. Therefore, the task was given lowest priority, no other tasks or hardware interrupts depended on the FFT task and the IDL_run command was called within the task to ensure the idle threads were called.

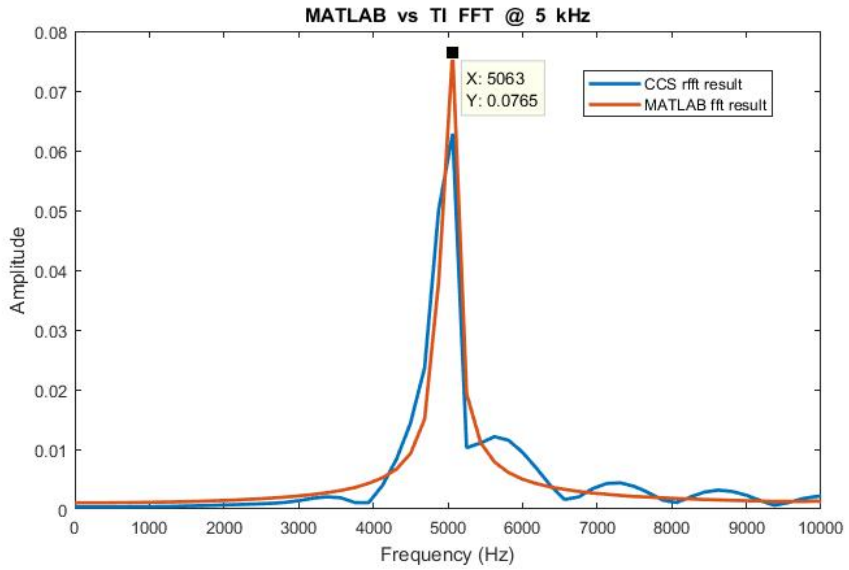

Figure 4: MATLAB and CCS comparison of FFT results at 5000 Hz Tone

4 Appendix

```
/*
 *  Copyright 2010 by Texas Instruments Incorporated.
 *  All rights reserved. Property of Texas Instruments Incorporated.
 *  Restricted rights to use, duplicate or disclose this code are
 *  granted through contract.
 *
 */
/*****************************************************************************/
/*                                                                           */
/*     H E L L O . C                                                         */
/*                                                                           */
/*     Basic LOG event operation from main.                                  */
/*                                                                           */
/*****************************************************************************/

#include <std.h>

#include <log.h>

#include "hellocfg.h"
#include "ezdsp5535.h"
#include "ezdsp5535_i2s.h"
#include "ezdsp5535_lcd.h"
#include "ezdsp5535_led.h"
#include "ezdsp5535_sar.h"
#include "csl_i2s.h"
#include "stdint.h"
#include "aic3204.h"

extern CSL_I2sHandle    hI2s;
extern void audioProcessingInit(void);


void main(void)
{
    LOG_printf(&trace, "hello world!");

```

```
38      /* Initialize BSL */
39      EZDSP5535_init( );
40
41      /* init LEDs and set to off*/
42      EZDSP5535_LED_init( );
43      EZDSP5535_LED_setall(0x0F);
44
45      /* init dip switches */
46      EZDSP5535_SAR_init( );
47
48          /* Initialize OLED display */
49      EZDSP5535_OSD9616_init( );
50      EZDSP5535_OSD9616_send(0x00,0x2e);  // Deactivate Scrolling
51      EZDSP5535_OSD9616_send(0x00,0x2e);  // Deactivate Scrolling
52
53      // configure the Codec chip
54      ConfigureAic3204();
55
56      /* Initialize I2S */
57      EZDSP5535_I2S_init();
58
59      /* enable the interrupt with BIOS call */
60      C55_enableInt(14); // reference technical manual, I2S2 tx interrupt
61      C55_enableInt(15); // reference technical manual, I2S2 rx interrupt
62
63      audioProcessingInit();
64
65      // after main() exits the DSP/BIOS scheduler starts
66  }
```

Listing 4: main.c

```
1  /*
2   * audioProcessing.h
3   *
4   *  Created on: Mar 29, 2018
5   *      Author: Zach
6   */
7
8  #include "stdint.h"
9  #include "string.h"
10 #include "dsplib.h"
11 #include "filters.h"
12
13 /*****************************************************************************
14  ********************************  DEFINES  **********************************
15  *****************************************************************************/
16
17 #define NX              48                          // number of samples filtered
       per myfir call
18 #define N               256                         // N-point FFT
19 #define N2              512                         // N doubled
20 #define NBY2    128                             // N halved
21
22 /*****************************************************************************
23  ********************************  FUNCTIONS  ********************************
24  *****************************************************************************/
25
26 extern CSL_I2sHandle    hI2s;
27 extern void myfir(const int16_t* input, const int16_t* filterCoeffs, int16_t* output,
       int16_t* delayLine, uint16_t  nx, uint16_t  nh);
28 extern int16_t myNCO(uint16_t f_tone);
29 extern void clearPA(void);
30 extern void initLCDVertAddr( void );
31 extern void fftDisplay(int16_t * samples, int numSamps, int16_t maxVal);
32
33 /*****************************************************************************
34  ***************************  ARRAYS + POINTERS  *****************************
35  *****************************************************************************/
36
37 int16_t rxPingPong[96];                                     // array of received
       audio (2 blocks of 48)
38 int16_t txPingPong[96];                                     // array of audio to
       transmit (2 blocks of 48)
39
40 int16_t delayLineLPF[NX + LPF_NH - 1];          // delay line for low-pass filter
41 int16_t delayLineHPF[NX + HPF_NH - 1];          // delay line for high-pass filter
42
43 int16_t * filterPtr;                                       // pointer to select low
       - or high- pass filter
44 int16_t * delayLinePtr;                                    // pointer to select
       delay line for LPF or HPF
45
46 #pragma DATA_ALIGN(fftData,2);
47 DATA fftData[N];
48 #pragma DATA_ALIGN(data1,2);
49 DATA data1[N];
50 int16_t fftMag[NBY2+1];
51
52 /*****************************************************************************
53  *********************************  VARIABLES  *******************************
54  *****************************************************************************/
55
56 int16_t rxIndex;                                // 0: samples 1-48; 1: samples 49-96
57 int16_t txIndex;                                // 0: samples 1-48; 1: samples 49-96
58 int16_t ppIndex;                                // 0: samples 1-48; 1: samples 49-96
59 int16_t fftIndex;                               // 0: samples 1-N; 1: samples N+1-2N
60 int16_t dataIndex;                              // ??
61
```

```
62 uint16_t sw1State  = 0;          // SW1 state
63 uint16_t sw2State  = 0;          // SW2 state
64 uint16_t filtState;                              // 1: low-pass filter; 0: high-pass
       filter
65
66 uint16_t myNH;                                   // length of selected filter
67
68 int displayFreeze = 0;                    // 0: no update; 1: update graph
```

Listing 5: audioProcessing.h

```
1  /*
2   *  Copyright 2010 by Texas Instruments Incorporated.
3   *  All rights reserved. Property of Texas Instruments Incorporated.
4   *  Restricted rights to use, duplicate or disclose this code are
5   *  granted through contract.
6   *
7   */
8  /***************************************************************************/
9  /*                                                                         */
10 /*     H E L L O . C                                                       */
11 /*                                                                         */
12 /*     Basic LOG event operation from main.                                */
13 /*                                                                         */
14 /***************************************************************************/
15
16 #include <std.h>
17
18 #include <log.h>
19
20 #include "stdint.h"
21 #include "string.h"
22 #include "math.h"
23 #include "hellocfg.h"
24 #include "ezdsp5535.h"
25 #include "ezdsp5535_gpio.h"
26 #include "ezdsp5535_i2s.h"
27 #include "ezdsp5535_led.h"
28 #include "ezdsp5535_sar.h"
29 #include "csl_i2s.h"
30 #include "csl_gpio.h"
31 #include "aic3204.h"
32 #include "audioProcessing.h"
33 #include "dsplib.h"
34
35 /*
36  * audioProcessingInit
37  *
38  * @brief:       Initialize arrays used for filtering and transmitting
39  *                      and initialize array indices to 0.
40  */
41 void audioProcessingInit(void)
42 {
43         initLCDVertAddr();
44
45         /* Initialize arrays as empty*/
46         memset(txPingPong, 0, sizeof(rxPingPong) * sizeof(int16_t));
47         memset(txPingPong, 0, sizeof(txPingPong) * sizeof(int16_t));
48         memset(delayLineLPF, 0, sizeof(delayLineLPF) * sizeof(int16_t));
49         memset(delayLineHPF, 0, sizeof(delayLineHPF) * sizeof(int16_t));
50         memset(fftData, 0, sizeof(fftData) * sizeof(DATA));
51
52         /* Initially select low-pass filter */
53         filterPtr = &myLPF[0];
54         delayLinePtr = &delayLineLPF[0];
55         myNH = LPF_NH;
56         filtState = 1;
57
58         /* Initialize rx and tx indices to 0 */
59         rxIndex = 0;
60         txIndex = 0;
61         ppIndex = 0;
62         dataIndex = 0;
63 }
64
65 /*****************************************************************
66  ******************          HWIs          ******************
67  *****************************************************************/
```

```
68
69  /*
70   * HWI_I2S_Rx
71   *
72   * @brief:       Function handle for HWI 15. Stores received samples into a
73   *                      double buffer and post SWIs to perform filtering.
74   */
75  void HWI_I2S_Rx(void)
76  {
77          volatile int16_t temp;
78          temp = hI2s->hwRegs->I2SRXRT1;
79          rxPingPong[rxIndex++] = hI2s->hwRegs->I2SRXLT1;
80
81          if (rxIndex == 48)              //Have 48 samples been collected
82          {
83                  MBX_post(&MBXAudio, &rxPingPong[0], 0);
84          }
85
86          if (rxIndex == 96)
87          {
88                  MBX_post(&MBXAudio, &rxPingPong[48], 0);
89                  rxIndex = 0;
90          }
91  }
92
93  /*************************************************************************
94   * HWI_I2S_Tx
                                                *
95   *
                                                                      *
96   * @brief:       Function handle for HWI 14. Transmits filtered samples          *
97   *                      from a double buffer.
                                      *
98   *************************************************************************/
99  void HWI_I2S_Tx(void)
100 {
101         /* Transmit filtered samples */
102         hI2s->hwRegs->I2STXLT1 = txPingPong[txIndex];
103         hI2s->hwRegs->I2STXRT1 = txPingPong[txIndex++];
104
105         if (txIndex == 96)              //Have 96 samples been transmitted?
106         {
107                 txIndex = 0;
108         }
109 }
110
111 /*************************************************************************
112  ******************              TSKs              ******************
113  *************************************************************************/
114
115 void TSKPingPongFunc(void)
116 {
117         int i;
118         int16_t ping[48], pong[48];
119
120         while(1)
121         {
122                 if(ppIndex == 0)
123                 {
124                         MBX_pend(&MBXAudio, &ping, SYS_FOREVER);
125
126                         myfir(&ping[0], filterPtr, &txPingPong[0], delayLinePtr, NX,
                                myNH);
127                         ppIndex = 1;
128
129                         for(i = 0; i < 48; i++)
130                         {
131                                 fftData[dataIndex++] = (DATA)txPingPong[i];
```

```
132
133                                     if(dataIndex == N)
134                                     {
135                                             MBX_post(&MBXfft, &fftData[0], 0);
136                                             dataIndex = 0;
137                                     }
138                             }
139                     }
140
141                     else if (ppIndex == 1)
142                     {
143                             MBX_pend(&MBXAudio, &pong, SYS_FOREVER);
144
145                             myfir(&pong[0], filterPtr, &txPingPong[48], delayLinePtr, NX,
                                    myNH);
146                             for(i = 48; i < 96; i++)
147                             ppIndex = 0;
148
149                             for(i = 48; i < 96; i++)
150                             {
151                                     fftData[dataIndex++] = (DATA)txPingPong[i];
152
153                                     if(dataIndex == N)
154                                     {
155                                             MBX_post(&MBXfft, &fftData[0], 0);
156                                             dataIndex = 0;
157                                     }
158                             }
159                     }
160             }
161     }
162
163     void TSK_FFT_Func(void)
164     {
165             int i, j;
166             int16_t max;
167
168             while(1)
169             {
170                     MBX_pend(&MBXfft, &data1, SYS_FOREVER);
171                     max = 0;
172                     rfft(data1, N, SCALE);
173
174                     IDL_run();
175
176                     fftMag[0] = (int16_t)(sqrt( pow((double)(data1[0]), 2) ) );
177                     fftMag[NBY2] = (int16_t)(sqrt( pow((double)(data1[1]), 2) ) );
178
179                     for(i = 1, j = 2; i < NBY2; i++, j += 2)
180                     {
181                             fftMag[i] = (int16_t)(sqrt( pow((double)(data1[j]), 2) + pow((
                                    double)(data1[(j + 1)]), 2) ) );
182
183                             if(fftMag[i] > max)
184                             {
185                                     max = fftMag[i];
186                             }
187                     }
188
189                     IDL_run();
190
191                     if(!displayFreeze)
192                     {
193                             fftDisplay(fftMag, NBY2, max);
194                     }
195             }
196     }
197
```

```
198 /*********************************************************************
199  ******************              IDLs              ******************
200  *********************************************************************/
201
202 void monitorSW1(void)
203 {
204         /* Check SW1 */
205         if(EZDSP5535_SAR_getKey( ) == SW1)              // Is SW1 pressed?
206         {
207                 if(sw1State)                                    // Was previous state
                        not pressed?
208                 {
209                         if(displayFreeze)
210                         {
211                                 displayFreeze = 0;
212                         } else {
213                                 displayFreeze = 1;
214                         }
215
216                         sw1State = 0;                                   // Set state to
                            0 to allow only single press
217                 }
218         } else                                          // SW1 not pressed
219         {
220                 sw1State = 1;                                   // Set state to 1 to
                        allow timer change
221         }
222 }
223
224 void monitorSW2(void)
225 {
226         /* Check SW2 */
227         if(EZDSP5535_SAR_getKey( ) == SW2)              // Is SW2 pressed?
228         {
229                 if(sw2State)                                    // Was previous state
                        not pressed?
230                 {
231                         Uns olstate = HWI_disable();
232                         TSK_disable();
233
234                         if(filtState)                                   //Was previous
                            state High-pass?
235                         {
236                                 /* Clear Low-pass delayLine */
237                                 memset(delayLineLPF, 0, sizeof(delayLineHPF)*sizeof(
                                    int16_t));
238
239                                 /* Point filter pointer to myLPF */
240                                 filterPtr = &myLPF[0];
241
242                                 /* Point delay line pointer to delayLineLPF */
243                                 delayLinePtr = &delayLineLPF[0];
244
245                                 /* Set myNH to number of low-pass coefficients */
246                                 myNH = LPF_NH;
247
248                                 /* Set filtState to low-pass */
249                                 filtState = 0;
250                         } else                                  //Was previous state low-pass
251                         {
252                                 /* Clear high-pass delay line */
253                                 memset(delayLineHPF, 0, sizeof(delayLineHPF)*sizeof(
                                    int16_t));
254
255                                 /* Point filter pointer to myHPF */
256                                 filterPtr = &myHPF[0];
257
258                                 /* Point delayline pointer to delayLineHPF */
```

```
259                                 delayLinePtr = &delayLineHPF[0];
260
261                                 /* Set myNH to number of high-pass coefficients */
262                                 myNH = HPF_NH;
263
264                                 /* Set filtState to high-pass */
265                                 filtState = 1;
266                             }
267
268                             HWI_restore(olstate);
269                             TSK_enable();
270                             sw2State = 0;                                   // Set state to
                                    0 to allow only single press
271                         }
272                 } else                                              // SW2 not pressed
273                 {
274                         sw2State = 1;                                       // Set state to 1 to
                            allow tone change
275                 }
276 }
```

Listing 6: audioProcessing.c

```c
/*
 * oscDisplay.c
 *
 *  Created on: Mar 8, 2018
 *      Author: Zach
 */

#include "stdint.h"
#include "ezdsp5535_lcd.h"
#include "ezdsp5535_gpio.h"
#include "ezdsp5535_i2c.h"

void initLCDVertAddr( void )
{
        Uint16 cmd[4];

        cmd[0] = 0x00 & 0x00FF;
        cmd[1] = 0x20;
        cmd[2] = 0x01;
    EZDSP5535_waitusec( 250 );
        EZDSP5535_I2C_write(0x3C, cmd, 3);

        cmd[0] = 0x00 & 0x00FF;
        cmd[1] = 0x21;
        cmd[2] = 0x00;
        cmd[3] = 0x5f;
    EZDSP5535_waitusec( 250 );
        EZDSP5535_I2C_write(0x3C, cmd, 4);

        cmd[0] = 0x00 & 0x00FF;
        cmd[1] = 0x22;
        cmd[2] = 0x00;
        cmd[3] = 0x01;
    EZDSP5535_waitusec( 250 );
        EZDSP5535_I2C_write(0x3C, cmd, 4);
}

void pixLoc(int16_t sample, Uint16 * bot, Uint16 * top, int maxVal)
{
        int i;
        Uint32 temp = 0x01;

        if(maxVal < 100)
        {
                maxVal = 32767;
        }

        for( i = 1; i < 16; i++)
        {
                if(sample > (i * (maxVal / 16)))
                {
                        temp |= temp << 1;
                }

                else
                {
                        i = 16;
                }
        }

        *bot = (Uint16)(temp);
        *top = (Uint16)(temp >> 8);
}

void fftDisplay(int16_t * samples, int numSamps, int16_t maxVal)
{
//      Uns olstate = HWI_disable();
```

```
68 //        TSK_disable();
69
70          int i, j;
71          Uint16 bot, top, cmd[193];
72
73          cmd[0] = 0x40 & 0x00FF;
74
75          for(i = 0, j = 1; i < 96; i++, j+=2)
76          {
77                  pixLoc(samples[i], &bot, &top, maxVal);
78
79                  cmd[j] = bot;
80                  cmd[(j + 1)] = top;
81          }
82
83          EZDSP5535_waitusec( 10 );
84          EZDSP5535_I2C_write(0x3C, &cmd[0], 193);
85
86 //        HWI_restore(olstate);
87 //        TSK_enable();
88 }
```

Listing 7: fftDisplay.c

```
1  /*****************************************************************************
2  ****                          D E F I N I T I O N S
3  *****************************************************************************/
4  #define LPF_NH          128
5  #define HPF_NH          128
6  /*****************************************************************************
7  ****                      G L O B A L   V A R I A B L E S
8  *****************************************************************************/
9
10 ///* 1200 Hz LPF transition to 3600, 32 coefficients */
11 //int16_t myLPF[] =
12 //{
13 //      -232,   -102,    -85,    -30,     74,    233,    448,    719,   1035,   1383,
       1744,   2095,   2413,   2676,   2862,   2959,
14 //      2959,   2862,   2676,   2413,   2095,   1744,   1383,   1035,    719,    448,
        233,     74,    -30,    -85,   -102,   -232
15 //};
16 //
17 ///* 4800 Hz HPF transition to 2400, 32 coefficients */
18 //int16_t myHPF[] =
19 //{
20 //       858,   -392,   -468,   -551,   -554,   -410,    -90,    382,    919,   1388,
       1614,   1411,    568,  -1235,  -5013, -20182,
21 //     20182,   5013,   1235,   -568,  -1411,  -1614,  -1388,   -919,   -382,     90,
        410,    554,    551,    468,    392,   -858
22 //};
23
24 /* 1200 Hz LPF transition to 2400 Hz, 128 coefficients */
25 int16_t myLPF[] =
26 {
27         -3,     -4,     -6,    -10,    -13,    -18,    -22,    -28,    -33,    -38,
              -43,    -47,    -50,    -51,    -50,    -46,
28        -40,    -31,    -18,     -3,     14,     34,     55,     77,     98,    117,
             132,    143,    149,    147,    137,    119,
29         92,     56,     13,    -38,    -93,   -152,   -210,   -266,   -316,   -357,
             -384,   -395,   -387,   -357,   -304,   -225,
30       -121,      7,    158,    330,    519,    721,    931,   1143,   1351,   1550,
            1734,   1897,   2034,   2140,   2214,   2251,
31       2251,   2214,   2140,   2034,   1897,   1734,   1550,   1351,   1143,    931,
             721,    519,    330,    158,      7,   -121,
32       -225,   -304,   -357,   -387,   -395,   -384,   -357,   -316,   -266,   -210,
            -152,    -93,    -38,     13,     56,     92,
33        119,    137,    147,    149,    143,    132,    117,     98,     77,     55,
              34,     14,     -3,    -18,    -31,    -40,
34        -46,    -50,    -51,    -50,    -47,    -43,    -38,    -33,    -28,    -22,
             -18,    -13,    -10,     -6,     -4,     -3
35 };
36
37 /* 3600 Hz HPF transition to 2400 Hz, 128 coefficients */
38 int16_t myHPF[] =
39 {
40         -3,    109,    -59,    -49,    -38,    -26,    -13,      2,     17,     32,
              42,     46,     42,     29,      8,    -18,
41        -44,    -66,    -79,    -78,    -61,    -31,     11,     56,     97,    126,
             134,    119,     79,     18,    -54,   -126,
42       -184,   -216,   -213,   -169,    -88,     20,    139,    249,    328,    358,
             327,    231,     78,   -112,   -311,   -484,
43       -597,   -618,   -529,   -324,    -16,    360,    757,   1112,   1352,   1404,
            1192,    635,   -388,  -2148,  -5666, -20419,
44      20419,   5666,   2148,    388,   -635,  -1192,  -1404,  -1352,  -1112,   -757,
             -360,     16,    324,    529,    618,    597,
45        484,    311,    112,    -78,   -231,   -327,   -358,   -328,   -249,   -139,
             -20,     88,    169,    213,    216,    184,
46        126,     54,    -18,    -79,   -119,   -134,   -126,    -97,    -56,    -11,
              31,     61,     78,     79,     66,     44,
47         18,     -8,    -29,    -42,    -46,    -42,    -32,    -17,     -2,     13,
              26,     38,     49,     59,   -109,      3
```

```
48 };
```

Listing 8: filters.h