Project 04 - BIOS Multithread

Introduction to Embedded Systems - University of Nebraska

Zach Swanson

Contents

1 Introduction

The purpose of this project was to introduce tasks (TSK), an additional type of BIOS thread not used in project three. A TSK replaced the software interrupts (SWIs) from project three, which handled the high-pass/low-pass filter audio processing. An additional TSK was used to implement a decimation filter and to display the output of the filter to the LCD display. As in project three, one hardware interrupt thread (HWI) was used to handle receiving audio and another HWI was used to handle transmitting audio. An additional purpose of this project was to introduce mailbox objects (MBX), which were used to communicate samples between the receive HWI and the TSKs. The project also required the design of several filters.

2 Project Description

2.1 Filter Design

For this project, a decimation filter was implemented for the display portion. The 48 kHz sample rate was decimated by an integer factor of 6 to a sample rate of 8 kHz. Therefore, an anti-aliasing filter was implemented to eliminate all frequency components above the Nyquist frequency at 4 kHz. Ideally, the filter's passband and stopband frequency would both equal 4 kHz. However, such a filter cannot be practically implemented due to the large number of coefficients it would require. Therefore, a design decision was made to set the stopband frequency at 4 kHz to ensure that no aliasing would occur. The complete filter specifications are listed below and the filter's magnitude response is shown in Figure **??**.

- Passband frequency: **3000 Hz**
- Stopband frequency: **4000 Hz**
- Passband ripple: **0.2 dB**
- Stopband attenuation: **60 dB**
- Order: **121**

Additionally, the low- and high-pass filters implemented in project three were recycled for project four. The filters had stop and pass bands at 350 Hz/1310 Hz and 2100 Hz/1400 Hz, respectively. And the filters had 125 and 124 coefficients, respectively.

2.2 Program Description

The program consisted of two HWI threads, two TSK threads, two IDL threads and two MBXs. Figure 2 illustrates the overall flow of the program in a block diagram. Audio was sampled by the stereo in the ADC and the samples were managed by the I2S receive HWI. The HWI stored copies of the samples in two arrays: one for audio processing/playback and one for decimation/signal display. Once the necessary number of samples had been collected, the receive HWI posted the array of samples to one of two arrays which corresponded to a specific TSK. The MBXs eliminated the need to pass a global array and posting to the MBX was analagous to posting a SWI. Depending on which MBX was posted, the program could follow two routes.

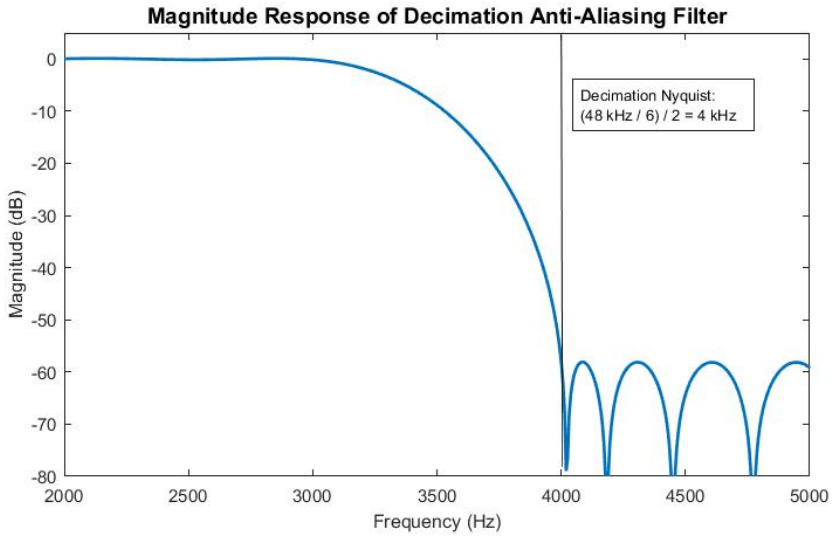

Figure 1: Magnitude response of anti-aliasing filter

Similar to the ping-pong array used in project three, the MBX for the audio processing TSK had a length of two and a size of 48. In such a manner, the HWI could post 48 samples to the MBX and then continue storing new samples to be posted next. Until the HWI posted to the MBX, the audio processing TSK had been essentially waiting or pending on the HWI. After the MBX post, the TSK was no longer pending and began filtering audio using myfir, which stored values to a txPingPong array, similar to project three. The transmit HWI wrote filtered audio samples to the DAC to be played back to the listener. Again, an IDL thread was used to monitor for switch two presses and switched filter types in response.

The second MBX had a length of one and a size of 576. A total of 576 samples were collected before posting to this MBX because it was determined that 576 samples at 48 kHz were necessary to generate 96 samples after decimating by 6 (i.e. 96*6=576). Once the MBX was posted the second TSK stopped pending and began to run. However, it should be noted the audio processing TSK was given a higher priority such that the display TSK wouldn't interfere with the real-time requirement of the audio processing. The TSK took the 576 samples from the MBX and passed them through an anti-aliasing filtering using myfir. Again, when decimating, the sampling frequency is reduced by an integer factor and the Nyquist frequency is also reduced by the same factor. Therefore, all frequency components above the new Nyquist must be removed to avoid aliasing. After performing the filtering, decimation by a factor of six was performed by storing every sixth element of the 576 element array. The anti-aliasing and decimation was performed every time the receive HWI posted to the MBX, in order to have samples continually ready to be displayed.

An additional IDL thread was used to monitor switch one presses. When switch one was pressed, the IDL changed a global state variable that notified the display TSK that it should display an audio signal to the LCD. The LCD functions were taken from an example program provided by TI. To display the audio signal, the correct pixels had to be turned on according to the sample value. Only 16 pixels were available in each column; therefore, samples had to be binned according to their relative value. First, if the sample was positive it was placed in the top eight pixels of a column. Otherwise, it was placed in the bottom eight pixels. A value for the pixel was initially set near the center and a for loop was used to divide max value in

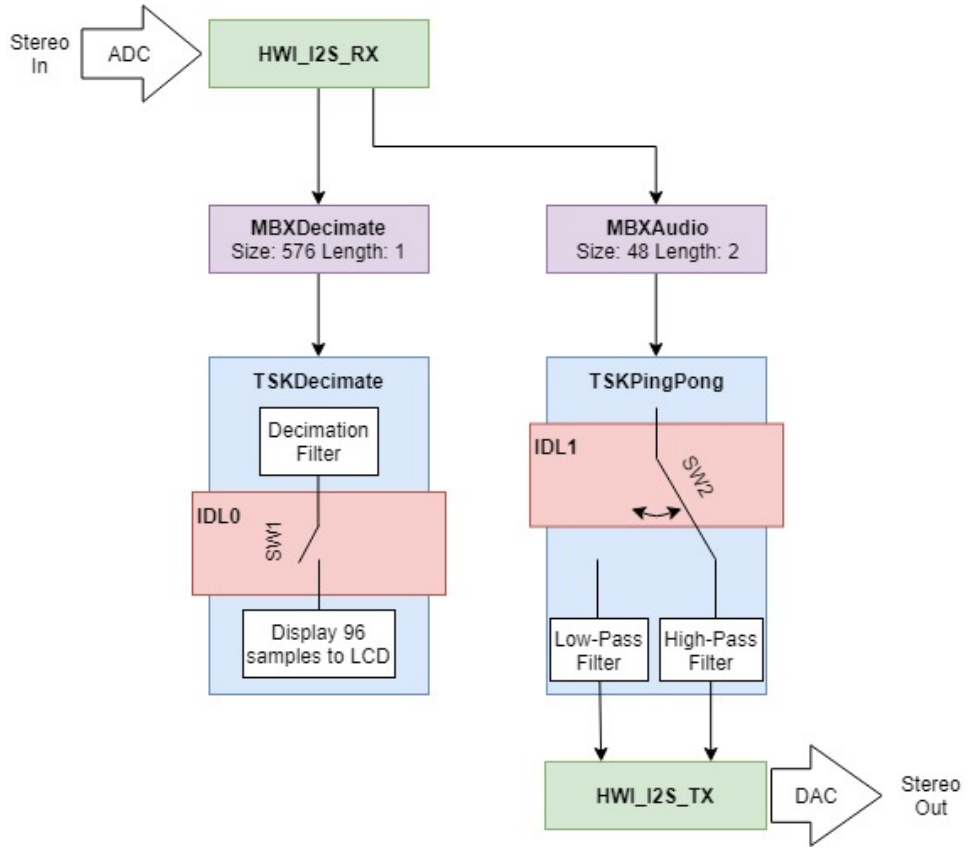

Figure 2: Block diagram of program flow.

either direction 32767 or 32768 into 8 bins. By looping through each area the value was checked against the new minimum absolute value for the bin. If the sample value was larger, then pixel value was bit shifted by one bit in such a manner that it was shifted away from the center. After determining the correct position in the column, the location was sent to the LCD chip and the send function advanced the display to the next column. Before sending the audio signal to the LCD, the display was cleared.

3 Results

3.1 Display

Using a GPIO pin, it was determined that it took the approximately 634 ms to clear and write to the display. Based on the fact that a frame is 96 samples, it was estimated that the frame rate was 1.6 frames per second. Ultimately, using the provided functions introduced unneccessary tasks for sending pixels that inflated the time requirement. Also, a more efficient way of clearing the existing pixels would have eliminated a significant portion of the display time.

While many methods of improving frame rate may be applied, the rate is limited by the slow I2C communication line. A typical I2C operates at 100 kbits/s to 400 kbits/s. At 96 samples per frame, 16 bits per sample and a max transmission speed of 400 kbits/s, the maximum frame rate would be 260 frames per second. And if the I2C were limited to 100 kbits/s, then the maximum frame rate would be 65 frames per second. At 100 kbits/s, the current display

method is operating at 2.5% of peak efficiency.

3.2 Execution Graph

Figure 3 shows an execution graph with the I2S rx HWI (top), audio processing TSK (middle), and decimation/disply TSK (bottom). As expected, the audio processing TSK pends unitl 48 samples had been collected, runs, and pends until another 48 samples were collected. Additionally, the execution graph shows decimation TSK running, but it does not show how long the TSK pends. It was expeceted that 576 rx HWI samples would have occured before the TSK quit pending again. Based on that assumption, it would have taken 12 milliseconds.

Furthermore, the execution graph shows the time required by the audio processing TSK to process 48 samples: 142.5 microseconds. That means that the TSK is taking approximately 297 cycles per sample which is within the bounds of operating in real-time. Even the decimation filter is operating in real-time (392 cycles per sample). However, when the display component is introduced the TSK no longer operates in real-time (110462 cycles per sample).

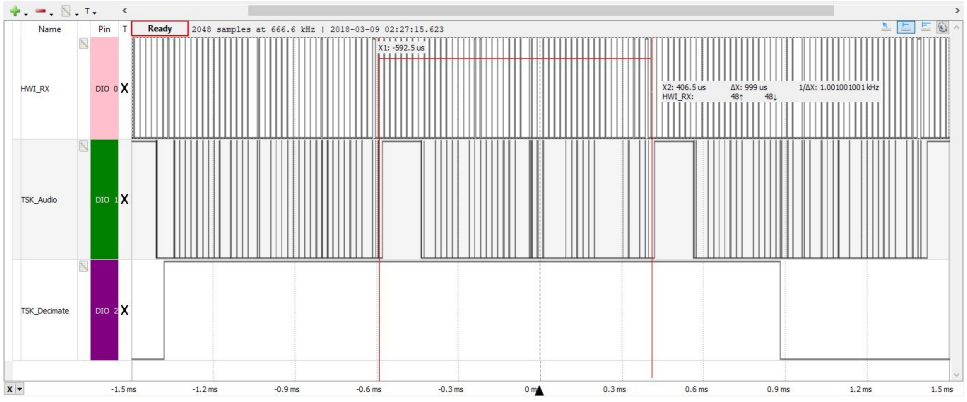

Figure 3: Execution of I2S rx HWI, ping-pong TSK, and decimation TSK

3.3 Decimation

Figure 4 shows a discrete time plot of samples delivered to the decimation TSK before anti-aliasing, after anti-aliasing, and after decimation. The plot is not actually drawn to scale, it was plotted to illustrate that decimated samples align with every sixth sample provided by the anti-aliasing filter, which would be expected when decimating by six. Furthermore, if it were plotted to the actual number of the sample in the signal, then the decimated signal would appear as a compressed version of the original. When observing Figure 5, the decimated signal generated by the DSP board has a frequency that is expanded. Therefore, it makes sense that the time domain signal would compress because expansion in frequency is contraction in time and vice-versa. Furthermore, what is observed in Figure 5 is that at higher sampling rates more frequency components are distributed across the Nyquist range. When the signal is decimated a small portion of the original Nyquist range is essentially expanded to cover the entire Nyquist range (speaking relatively and in terms of periodicity). Therefore, such an observation again shows that a successful decimation filter has been implemented.

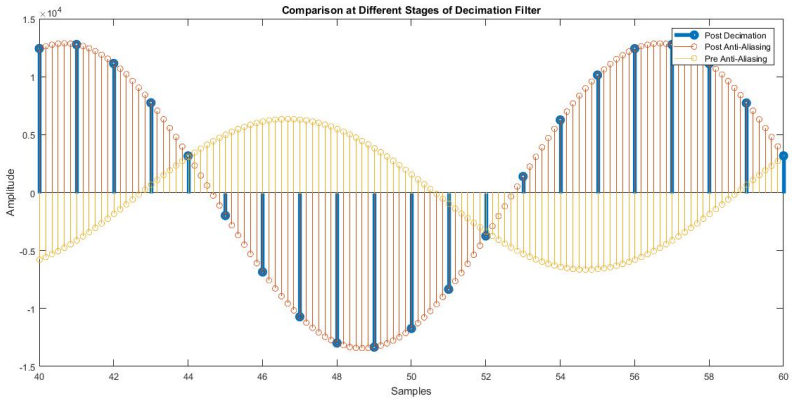

Figure 4: A discrete time plot of data generated from the anti-aliasing filter and the decimation filter

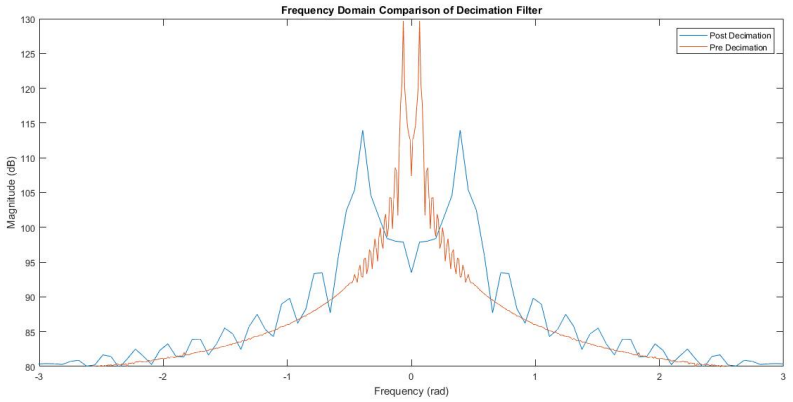

Figure 5: A frequency domain plot of data generated from the anti-aliasing filter and the decimation filter

4 Appendix

```
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 "hellocfg.h"
```

```
21 #include "ezdsp5535.h"
22 #include "ezdsp5535_i2s.h"
23 #include "ezdsp5535_lcd.h"
24 #include "ezdsp5535_led.h"
25 #include "ezdsp5535_sar.h"
26 #include "csl_i2s.h"
27 #include "stdint.h"
28 #include "aic3204.h"
29
30 extern CSL_I2sHandle    hI2s;
31 extern void audioProcessingInit(void);
32
33
34 void main(void)
35 {
36     LOG_printf(&trace, "hello world!");
37
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 1: main.c

```
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 "hellocfg.h"
23 #include "ezdsp5535.h"
24 #include "ezdsp5535_gpio.h"
25 #include "ezdsp5535_i2s.h"
26 #include "ezdsp5535_led.h"
27 #include "ezdsp5535_sar.h"
28 #include "csl_i2s.h"
29 #include "csl_gpio.h"
30 #include "aic3204.h"
31 #include "filters.h"
32
33 #define NX                      48
34 #define BL_ONLY         0x0E
35 #define YL_ONLY         0x0D
36 #define RD_ONLY         0x0B
37 #define GR_ONLY         0x07
38 #define MIN(a,b) ( ((a) <  (b)) ? (a) : (b) )
39
40 extern CSL_I2sHandle   hI2s;
41 extern void myfir(const int16_t* input, const int16_t* filterCoeffs,
42                 int16_t* output, int16_t* delayLine, uint16_t  nx, uint16_t  nh);
43 extern int16_t myNCO(uint16_t f_tone);
44 extern void clearPA(void);
45
46 int16_t rxPingPong[96];
47 int16_t txPingPong[96];
48 int16_t preDecSamps[576], preAA[576], postAA[576];
49 int16_t delayLineAA[576 + 121 - 1];
50 int16_t disp96[96];
51 int16_t finalDisp[96];
52
53 int16_t delayLineLPF[NX + LPF_NH - 1];
54 int16_t delayLineHPF[NX + HPF_NH - 1];
55
56 int16_t rxIndex;
57 int16_t txIndex;
58 int16_t ppIndex;
59 int16_t preDecIndex;
60 int16_t dispIndex;
61 int16_t deciIndex;
62
63 uint16_t sw1State  = 0;         // SW1 state
64 uint16_t sw2State  = 0;         // SW2 state
65 uint16_t filtState = 0;                 // filter state
66
67 int16_t * filterPtr;
```

```
68  int16_t * delayLinePtr;
69  uint16_t myNH;
70
71
72  Uint16 idl = 0, swi = 0;
73
74  int displayGraph = 0;
75
76  extern void oscDisplay(int16_t * samples, int numSamps);
77
78  /*
79   * audioProcessingInit
80   *
81   * @brief:      Initialize arrays used for filtering and transmitting
82   *                      and initialize array indices to 0.
83   */
84  void audioProcessingInit(void)
85  {
86          /* Initialize arrays as empty*/
87          memset(txPingPong, 0, sizeof(txPingPong));
88          memset(delayLineLPF, 0, sizeof(delayLineLPF));
89          memset(delayLineHPF, 0, sizeof(delayLineHPF));
90
91          /* Initially select low-pass filter */
92          filterPtr = &myLPF[0];
93          delayLinePtr = &delayLineLPF[0];
94          myNH = LPF_NH;
95
96          /* Initialize rx and tx indices to 0 */
97          rxIndex = 0;
98          txIndex = 0;
99          ppIndex = 0;
100         preDecIndex = 0;
101         deciIndex = 0;
102         dispIndex = 0;
103 }
104
105 /************************************************************************
106  ******************              HWIs              ******************
107  ************************************************************************/
108
109 /*
110  * HWI_I2S_Rx
111  *
112  * @brief:      Function handle for HWI 15. Stores received samples into a
113  *                      double buffer and post SWIs to perform filtering.
114  */
115 void HWI_I2S_Rx(void)
116 {
117 //      EZDSP5535_GPIO_setOutput( 14, 1 );
118
119         /* Read right sample and disregard. Read left sample and store
120          * in rxPingPong.
121          * Ping -> first 48 samples in array (0 - 47)
122          * Pong -> second 48 samples in array (48 - 97)
123          */
124         volatile int16_t temp;
125         temp = hI2s->hwRegs->I2SRXRT1;
126         rxPingPong[rxIndex++] = hI2s->hwRegs->I2SRXLT1;
127         preDecSamps[preDecIndex++] = temp;
128
129         if (rxIndex == 48)              //Have 48 samples been collected
130         {
131                 /* Ping is full -> Post SWIPing to run SWI that will
132                  * filter the ping samples.
133                  */
134                 MBX_post(&MBXAudio, &rxPingPong[0], 0);
135         }
```

```
136
137         if (rxIndex == 96)
138         {
139                 /* Pong is full -> Post SWIPong to run SWI that will
140                  * filter the pong samples. Clear rxIndex so rxPingPong
141                  * will begin filling ping again.
142                  */
143                 MBX_post(&MBXAudio, &rxPingPong[48], 0);
144                 rxIndex = 0;
145         }
146
147         if(preDecIndex == 576)
148         {
149                 MBX_post(&MBXDecimate, &preDecSamps[0], 0);
150                 preDecIndex = 0;
151         }
152
153         EZDSP5535_GPIO_setOutput( 14, 0 );
154 }
155
156 /*
157  * HWI_I2S_Tx
158  *
159  * @brief:       Function handle for HWI 14. Transmits filtered samples
160  *                      from a double buffer.
161  */
162 void HWI_I2S_Tx(void)
163 {
164         /* Transmit filtered samples */
165         hI2s->hwRegs->I2STXLT1 = txPingPong[txIndex];
166         hI2s->hwRegs->I2STXRT1 = txPingPong[txIndex++];
167
168         if (txIndex == 96)              //Have 96 samples been transmitted?
169         {
170                 /* Set index to beginning of tx array */
171                 txIndex = 0;
172         }
173 }
174
175 /************************************************************************
176  ******************                TSKs                ******************
177  ************************************************************************/
178
179 void TSKPingPongFunc(void)
180 {
181         int16_t ping[48], pong[48];
182
183     /* Filter a frame of 48 received samples and store output in tx buffer
184      *  using myfir. Variables filterPtr and delayLinePtr point to the desired
185      *  filter (LPF or HPF) and it's corresponding delayline (selected in second IDL
186      *  thread).
187      */
188         while(1)
189         {
190                 if(ppIndex == 0)
191                 {
192                         MBX_pend(&MBXAudio, &ping, SYS_FOREVER);
193 //                      EZDSP5535_GPIO_setOutput( 15, 1 );
194                         myfir(&ping[0], filterPtr, &txPingPong[0], delayLinePtr, NX,
                                myNH);
195                         ppIndex = 1;
196                 } else if (ppIndex == 1)
197                 {
198                         MBX_pend(&MBXAudio, &pong, SYS_FOREVER);
199 //                      EZDSP5535_GPIO_setOutput( 15, 1 );
200                         myfir(&pong[0], filterPtr, &txPingPong[48], delayLinePtr, NX,
                                myNH);
201                         ppIndex = 0;
```

```
202                 }
203
204 //                EZDSP5535_GPIO_setOutput( 15, 0 );
205         }
206 }
207
208 void TSKDecimateFunc(void)
209 {
210         int i;
211
212         while(1)
213         {
214                 MBX_pend(&MBXDecimate, &preAA, SYS_FOREVER);
215 //                EZDSP5535_GPIO_setOutput( 17, 1 );
216                 myfir(&preAA[0], sharpAA, &postAA[0], delayLineAA, 576, 121);
217                 IDL_run();
218
219                 disp96[0] = postAA[0];
220
221                 for(i = 6; i < 576; i++)
222                 {
223                         if((i % 6) == 0)
224                         {
225                                 disp96[i/6] = postAA[i];
226                         }
227                 }
228
229                 IDL_run();
230                 if(displayGraph == 1)
231                 {
232                         oscDisplay(disp96, 96);
233                         displayGraph = 0;
234                 }
235
236 //                EZDSP5535_GPIO_setOutput( 17, 0 );
237         }
238 }
239
240 /*********************************************************************
241  ******************              IDLs              ******************
242  *********************************************************************/
243
244 void monitorSW1(void)
245 {
246         /* Check SW1 */
247         if(EZDSP5535_SAR_getKey( ) == SW1)              // Is SW1 pressed?
248         {
249                 if(sw1State)                                    // Was previous state
                        not pressed?
250                 {
251                         displayGraph = 1;                               //Tell TSK to
                            display graph
252
253                         sw1State = 0;                                   // Set state to
                            0 to allow only single press
254                 }
255         } else                                          // SW1 not pressed
256         {
257                 sw1State = 1;                                   // Set state to 1 to
                        allow timer change
258         }
259 }
260
261 void monitorSW2(void)
262 {
263         /* Check SW2 */
264         if(EZDSP5535_SAR_getKey( ) == SW2)              // Is SW2 pressed?
265         {
```

```
266                 if(sw2State)                                        // Was previous state
                        not pressed?
267                 {
268                         if(filtState)                                       //Was previous
                                state High-pass?
269                         {
270                                 /* Clear Low-pass delayLine */
271                                 memset(delayLineLPF, 0, sizeof(delayLineLPF));
272
273                                 /* Point filter pointer to myLPF */
274                                 filterPtr = &myLPF[0];
275
276                                 /* Point delay line pointer to delayLineLPF */
277                                 delayLinePtr = &delayLineLPF[0];
278
279                                 /* Set myNH to number of low-pass coefficients */
280                                 myNH = LPF_NH;
281
282                                 /* Set filtState to low-pass */
283                                 filtState = 0;
284                         } else                              //Was previous state low-pass
285                         {
286                                 /* Clear high-pass delay line */
287                                 memset(delayLineHPF, 0, sizeof(delayLineHPF));
288
289                                 /* Point filter pointer to myHPF */
290                                 filterPtr = &myHPF[0];
291
292                                 /* Point delayline pointer to delayLineHPF */
293                                 delayLinePtr = &delayLineHPF[0];
294
295                                 /* Set my NH to number of high-pass coefficients */
296                                 myNH = HPF_NH;
297
298                                 /* Set filtState to high-pass */
299                                 filtState = 1;
300                         }
301                         sw2State = 0;                                       // Set state to
                                0 to allow only single press
302                 }
303         } else                                              // SW2 not pressed
304         {
305                 sw2State = 1;                                       // Set state to 1 to
                        allow tone change
306         }
307 }
```

Listing 2: audioProcessing.c

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

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

void pixLoc(int16_t sample, int * topBot, Uint16 * loc8)
{
        int i;

        if(sample < 0 )
        {
                *loc8 = 0x01;
                * topBot = 1;
                sample = sample * (-1);

                for( i = 1; i < 8; i++)
                {
                        if(sample > (i * (32768 / 8)))
                        {
                                * loc8 = (* loc8) << 1;
                        }
                }
        } else if(sample > 0) {
                *loc8 = 0x80;
                * topBot = 0;

                for( i = 1; i < 8; i++)
                {
                        if(sample > (i * (32768 / 8)))
                        {
                                * loc8 = (* loc8) >> 1;
                        }
                }
        } else {
                *loc8 = 0x00;
                *topBot = 0;
        }
}

void oscDisplay(int16_t * samples, int numSamps)
{
//      Uns olstate = HWI_disable();
//      TSK_disable();

//      EZDSP5535_GPIO_setOutput( 14, 0 );

        int i, j;
        int topBot;
        Uint16 loc8;

        /* Fill page 0 */
        EZDSP5535_OSD9616_send(0x00,0x00);    // Set low column address
        EZDSP5535_OSD9616_send(0x00,0x10);    // Set high column address
        EZDSP5535_OSD9616_send(0x00,0xb0+0); // Set page for page 0 to page 5

        for(i=0;i<128;i++)
        {
                EZDSP5535_OSD9616_send(0x40,0x00);
        }

        /* Fill page 1*/
```

```
68          EZDSP5535_OSD9616_send(0x00,0x00);    // Set low column address
69          EZDSP5535_OSD9616_send(0x00,0x10);    // Set high column address
70          EZDSP5535_OSD9616_send(0x00,0xb0+1); // Set page for page 0 to page 5
71
72          for(i=0;i<128;i++)
73          {
74                  EZDSP5535_OSD9616_send(0x40,0x00);
75          }
76
77          EZDSP5535_OSD9616_send(0x00,0x00);    // Set low column address
78          EZDSP5535_OSD9616_send(0x00,0x10);    // Set high column address
79
80          for(j = 0; j < 96; j++)
81          {
82                  pixLoc(samples[j], &topBot, &loc8);
83                  EZDSP5535_OSD9616_send(0x00,0xb0+topBot); // Set page for page 0 to page
                        5
84                  EZDSP5535_OSD9616_send(0x40,loc8);
85          }
86
87  //      EZDSP5535_GPIO_setOutput( 14, 1 );
88
89  //      HWI_restore(olstate);
90  //      TSK_enable();
91  }
```

Listing 3: oscDisplay.c

```
1  /*****************************************************************************
2  ****                          D E F I N I T I O N S
3  *****************************************************************************/
4  #define LPF_NH            126
5  #define HPF_NH            125
6  /*****************************************************************************
7  ****                        G L O B A L   V A R I A B L E S
8  *****************************************************************************/
9
10 int16_t myLPF[] =
11 {
12         -23,     -13,     -16,     -20,     -24,     -28,     -33,     -38,     -43,     -49,
               -55,     -60,     -66,     -72,     -77,     -82,
13         -86,     -90,     -93,     -95,     -96,     -95,     -93,     -89,     -84,     -77,
               -68,     -56,     -42,     -26,      -8,      13,
14          36,      62,      90,     121,     154,     189,     226,     265,     306,     348,
              391,     435,     480,     525,     571,     615,
15         660,     703,     745,     785,     823,     859,     892,     922,     950,     973,
              993,    1010,    1022,    1030,    1034,    1034,
16        1030,    1022,    1010,     993,     973,     950,     922,     892,     859,     823,
             785,     745,     703,     660,     615,     571,
17         525,     480,     435,     391,     348,     306,     265,     226,     189,     154,
              121,      90,      62,      36,      13,      -8,
18         -26,     -42,     -56,     -68,     -77,     -84,     -89,     -93,     -95,     -96,
               -95,     -93,     -90,     -86,     -82,     -77,
19         -72,     -66,     -60,     -55,     -49,     -43,     -38,     -33,     -28,     -24,
               -20,     -16,     -13,     -23
20 };
21
22 int16_t myHPF[] =
23 {
24         -24,    -147,       6,     -15,      -6,       1,      10,      20,      31,      43,
               53,      62,      69,      73,      73,      69,
25          60,      46,      27,       5,     -21,     -49,     -78,    -106,    -132,    -153,
              -169,    -177,    -176,    -166,    -144,    -113,
26         -71,     -20,      38,     102,     168,     233,     294,     347,     388,     414,
              420,     405,     366,     300,     209,      91,
27         -53,    -219,    -404,    -605,    -817,   -1034,   -1251,   -1460,   -1657,   -1835,
              -1989,   -2115,   -2207,   -2263,   30486,   -2263,
28       -2207,   -2115,   -1989,   -1835,   -1657,   -1460,   -1251,   -1034,    -817,    -605,
             -404,    -219,     -53,      91,     209,     300,
29         366,     405,     420,     414,     388,     347,     294,     233,     168,     102,
               38,     -20,     -71,    -113,    -144,    -166,
30        -176,    -177,    -169,    -153,    -132,    -106,     -78,     -49,     -21,       5,
              27,      46,      60,      69,      73,      73,
31          69,      62,      53,      43,      31,      20,      10,       1,      -6,     -15,
                      6,    -147,     -24
32 };
33
34 /* Anti-aliasing filter for decimation Fpass = 3 kHz 0.2 dB, Fstop = 4 kHz -60 dB */
35 int16_t sharpAA[] =
36 {
37          19,      -5,     -13,     -25,     -39,     -53,     -64,     -67,     -62,     -47,
              -22,       8,      40,      67,      83,      84,
38          67,      33,     -13,     -62,    -105,    -132,    -135,    -110,     -58,      13,
               91,     159,     203,     210,     174,      95,
39         -13,    -133,    -241,    -312,    -326,    -273,    -155,      12,     201,     375,
              494,     525,     449,     263,     -12,    -334,
40        -646,    -880,    -972,    -868,    -540,      11,     753,    1621,    2527,    3374,
             4063,    4513,    4670,    4513,    4063,    3374,
41        2527,    1621,     753,      11,    -540,    -868,    -972,    -880,    -646,    -334,
              -12,     263,     449,     525,     494,     375,
42         201,      12,    -155,    -273,    -326,    -312,    -241,    -133,     -13,      95,
               174,     210,     203,     159,      91,      13,
43         -58,    -110,    -135,    -132,    -105,     -62,     -13,      33,      67,      84,
               83,      67,      40,       8,     -22,     -47,
44         -62,     -67,     -64,     -53,     -39,     -25,     -13,      -5,      19
```

```
45 };
```

Listing 4: filters.h

```
1  /*
2  // ******************************************************************************
3  // ******************************************************************************
4  // **
5  // ** File Name: myfir.c
6  // **
7  // ** Author: David McCreight
8  // **
9  // ** Description:
10 // **
11 // **
12 // ******************************************************************************
13 // ******************************************************************************
14 */
15
16 /*****************************************************************************
17 ****                          INCLUDES
18 *****************************************************************************/
19
20 #include "stdint.h"
21 #include "stdio.h"
22
23 /*****************************************************************************
24 ****                        DEFINITIONS
25 *****************************************************************************/
26
27 /*****************************************************************************
28 ****                   STATIC  VARIABLES
29 *****************************************************************************/
30
31 /*****************************************************************************
32 ****                    GLOBAL  VARIABLES
33 *****************************************************************************/
34
35 /*****************************************************************************
36 ****          FUNCTION  DEFINITIONS
37 *****************************************************************************/
38 void myfir(const int16_t* input,
39             const int16_t* filterCoeffs,
40                         int16_t* output,
41                         int16_t* delayLine,
42                         uint16_t  nx,
43                         uint16_t  nh)
44 {
45
46     uint16_t i;
47     uint16_t j;
48     long sum = 0;
49
50     /*
51      * Assumes delayLine length is nh - 1 + nx
52      */
53
54     // copy input samples to the delay line
55     for (i = 0; i < nx; i++)
56     {
57         delayLine[i + nh - 1] = input[i];
58     }
59
60         for (i = 0; i < nx; i++)
61         {
62                 for (j = 0; j < nh; j++)
63                 {
64                         sum = _smacr(sum, filterCoeffs[j], delayLine[i + nh - 1 - j]);
65                 }
66
67                 output[i] = (int16_t)(sum >> 15);
```

```
68                  sum = 0;
69          }
70
71      // Update delay line for next function call
72          for (i = 0; i < (nh - 1); i++)
73          {
74                  delayLine[i] = delayLine[nx + i];
75          }
76  }
77
78  /*****************************************************************************
79   ****                          E N D   O F   F I L E
80   *****************************************************************************/
```

Listing 5: myfir.c