# Project 03 - DSP/BIOS based FIR

Introduction to Embedded Systems - University of Nebraska

Zach Swanson

# Contents

# 1 Introduction

The purpose of this project was to introduce the Texas Instruments (TI) real-time operating system BIOS using the TI ezDSP5535 development board (ezDSP). The project required the use of hardware interrupt (HWI) threads, software interrupt (SWI) threads, and idle (IDL) threads. The goal of the project was to receive, filter and retransmit audio samples using HWIs and SWIs. Additional components to change between a low-pass (bass) and high-pass (treble) filter and to transmit a one kHz tone were included to employ the IDL threads. The project built on previous projects by requiring the use of myNCO and myFIR functions designed in Projects 1 and 2, respectively.

# 2 Project Description

## 2.1 Filter Design

Two filters were implemented for this project, a low- and high-pass filter, with the intent of isolating bass tones with the low-pass filter and isolating treble tones with the high-pass filter. The design process was trial-and-error until filters were achieved that produced the distinct bass and treble sounds that were desired.

The low-pass filter was designed with the following specifications:

- Passband frequency: **350 Hz**
- Stopband frequency: **1310 Hz**
- Passband ripple: **0.2 dB**
- Stopband attenuation: **60 dB**
- Order: **125**

The high-pass filter was designed with the following specifications:

- Passband frequency: **2100 Hz**
- Stopband frequency: **1400 Hz**
- Passband ripple: **0.2 dB**
- Stopband attenuation: **60 dB**
- Order: **124**

The frequency response of both filters is shown in Figure 1.

## 2.2 Program Description

TI provides an easy-to-use GCONF tool to configure the BIOS scheduler, which was used to establish the threads that were used for this project. To progress through the project incrementally the HWIs were setup and tested, the SWI was incorporated and finally the IDL threads were included. HWI 14 and 15 were used for I2S transmit (tx) and I2S receive (rx) with function handles_ HWI_I2S_Tx and _HWI_I2S_Rx, respectively. Two SWI threads

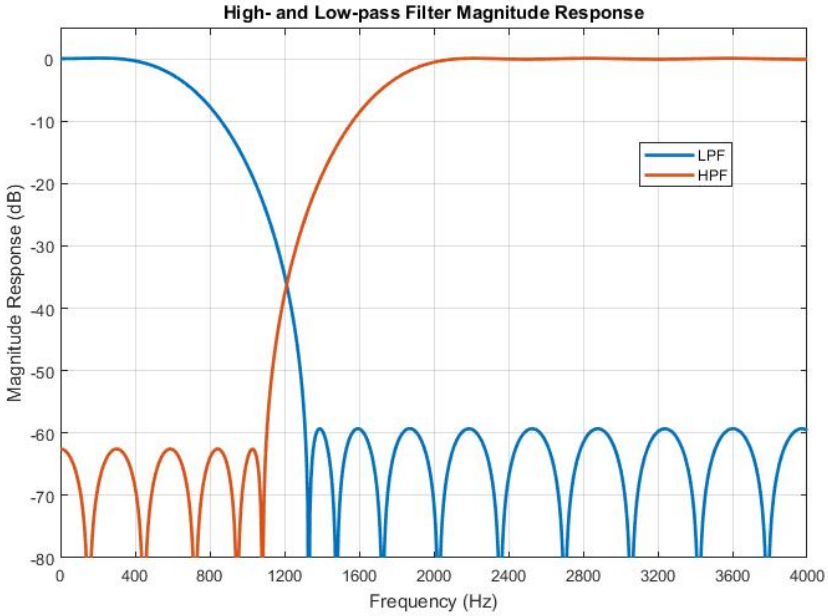


Figure 1: Magnitude response of low- and high-pass filters

with function handles _SWIPingFunc and _SWIPongFunc and equal priority were defined using GCONF. Lastly, the GCONF tool was used establish two IDL threads with function handles _monitorSW1 and _monitorSW2.

After establishing the threads using GCONF, the HWI threads were enabled in main and once the program exited main it launched the DSP/BIOS scheduler, which managed all the threads. The scheduler ensures that threads run in the correct order based on priority. In general, the priority level of thread types is HWI, SWI, IDL, in descending order.

### 2.3 HWI_I2S_Rx

Listing 1 shows the code for the HWI thread that handled I2S reception . It was assumed that the right and left samples were the same; therefore, the right sample was read and disregarded and the left sample was stored into a double buffer. The double buffer was 96 elements and was broken into two 48 element buffers, ping and pong. The purpose of the double buffer was that once ping was full its elements could be filtered while new data was loaded into pong. Otherwise, if a single buffer was used, then data may be lost during the process of filtering. The double buffer is reflected in the code. New data is stored in the double buffer rxPingPong until 48 samples have been received; then, the HWI posts the software interrupt SWIPing which handles filtering of the ping buffer. While SWIPingFunc filters the samples the HWI continues to place new samples into pong, i.e. the latter half of rxPingPong. Samples are stored until 48 more have been received, and the HWI posts a second SWI to filter the pong samples.

```
1  void HWI_I2S_Rx(void)
2  {
3          volatile int16_t temp;
4          temp = hI2s->hwRegs->I2SRXRT1;
5          rxPingPong[rxIndex++] = hI2s->hwRegs->I2SRXLT1;
6
7          if (rxIndex == 48)              //Have 48 samples been collected
8          {
```

```
9                   /* Ping is full */
10                  SWI_post(&SWIPing);
11          }
12          if (rxIndex == 96)
13          {
14                  /* Pong is full */
15                  SWI_post(&SWIPong);
16                  rxIndex = 0;
17          }
18  }
```

Listing 1: I2S receive HWI thread

### 2.4 HWI_I2S_Tx

Listing 2 shows the code for the HWI thread that handled I2S transmission. This thread simply writes the filtered samples to the I2S, checks to see if the end of the array has been reached and resets back to the beginning of the transmit array if it has.

```
1   void HWI_I2S_Tx(void)
2   {
3           hI2s->hwRegs->I2STXLT1 = txPingPong[txIndex];
4           hI2s->hwRegs->I2STXRT1 = txPingPong[txIndex++];
5
6           if (txIndex == 96)              //Have 96 samples been transmitted?
7           {
8                   txIndex = 0;
9           }
10  }
```

Listing 2: I2S transmit HWI thread

### 2.5 SWI_PingFunc and SWI_PongFunc

Listing 3 shows the code for the SWI thread that filtered samples in the ping array. The SWI thread for the filtering pong was not included because it was very similar. The SWI calls the myfir function designed in project 2, which used a delayline approach for implementing a FIR filter. The function took the ping array (i.e. samples 0-47 of rxPingPong), sampled all 48 samples and stored in txPingPong. To be able to switch between filters, a pointer was used to point to either the array of low-pass or high-pass coefficients and another pointer pointed to the filters' corresponding delayline array.

The ping and pong threads only ran when they had been posted by the rx HWI thread.

```
1   void SWIPingFunc(void)
2   {
3           myfir(&rxPingPong[0], filterPtr, &txPingPong[0], delayLinePtr, NX, myNH);
4   }
```

Listing 3: Ping filter SWI thread

### 2.6 montiorSW1

Listing 4 shows the code for the IDL thread that monitored for switch one strokes. When the thread detected a switch stroke, it disabled all SWIs and HWI ensure that a one kHz tone was played for one second without being interrupted and to prevent other samples from being transmitted with tone. The tone was generated using myNCO from project 1. Once the tone

played, both delaylines were cleared and the double buffer indexes were reset before enabling SWIs and restoring the HWIs to their previous state.

This thread would run when the HWIs and SWIs were not running and occurred sequentially with the other IDL thread.

```
1  void monitorSW1(void)
2  {
3          /* Check SW1 */
4          if(EZDSP5535_SAR_getKey( ) == SW1) // Is SW1 pressed?
5          {
6                  if(sw1State)                // Was previous state not pressed?
7                  {
8                          int16_t msec, sample;
9
10                         /* Disable SWIs and HWIs -- store HWI state */
11                         SWI_disable( );
12                         Uns oldState1 = HWI_disable( );
13
14                         /* nested for loop to play 1 kHz tone for 1 sec */
15                         for ( msec = 0 ; msec < 1000 ; msec++ )
16                         {
17                                 for ( sample = 0 ; sample < 48 ; sample++ )
18                                 {
19                                         /* call myNCO to get 1 kHz tone sample */
20                                         int16_t temp = (myNCO( 1000 ) >> 5);
21
22                                                 /* Write 16-bit left channel Data */
23                                         EZDSP5535_I2S_writeLeft( temp );
24
25                                         /* Write 16-bit right channel Data */
26                                         EZDSP5535_I2S_writeRight( temp );
27                                 }
28                         }
29                         /* clear phase accumulator */
30                         clearPA( );
31
32                         /* clear both delayLines and reset rx and tx indices */
33                         memset(delayLineLPF, 0, sizeof(delayLineLPF));
34                         memset(delayLineHPF, 0, sizeof(delayLineHPF));
35                         rxIndex = 0;
36                         txIndex = 0;
37
38                         /* restore HWIs to previous state and enable SWIs */
39                         HWI_restore( oldState1 );
40                         SWI_enable( );
41
42                         sw1State = 0;      // Set state to 0 to allow only single press
43                 }
44         } else                          // SW1 not pressed
45         {
46                 sw1State = 1;          // Set state to 1 to allow timer change
47         }
48 }
```

Listing 4: IDL thread for monitoring switch one strokes and playing a tone

## 2.7 monitorSW2

Listing 5 shows the code for the IDL thread that monitored for switch two strokes. When the thread detected a switch stroke, it checked to see if the low-pass or high-pass filter was being used and proceeded to implement the alternate filter. As mentioned, two delaylines were used because the filters differed in length, which required different length delaylines. When switching filters the new filter's delay line was cleared. This generated a brief disturbance in

the audio that was noticed if focused on trying to hear the sound. It was determined that the minor disturbance was a lower cost than the noise generated by having a single delayline or the cycles required to dump one delayline into another. Additionally, it provided an easier means of implementation. To make switching between filter and delayline arrays easier, pointer variables were used to simply point to the desired arrays.

```
1  void monitorSW2(void)
2  {
3          /* Check SW2 */
4          if(EZDSP5535_SAR_getKey( ) == SW2) // Is SW2 pressed?
5          {
6                  if(sw2State)           // Was previous state not pressed?
7                  {
8                          if(filtState)           //Was previous state High-pass?
9                          {
10                                 /* Clear Low-pass delayLine */
11                                 memset(delayLineLPF, 0, sizeof(delayLineLPF));
12
13                                 /* Point filter pointer to myLPF */
14                                 filterPtr = &myLPF[0];
15
16                                 /* Point delay line pointer to delayLineLPF */
17                                 delayLinePtr = &delayLineLPF[0];
18
19                                 /* Set myNH to number of low-pass coefficients */
20                                 myNH = LPF_NH;
21
22                                 /* Set filtState to low-pass */
23                                 filtState = 0;
24                         } else                                 //Was previous state low-pass
25                         {
26                                 /* Clear high-pass delay line */
27                                 memset(delayLineHPF, 0, sizeof(delayLineHPF));
28
29                                 /* Point filter pointer to myHPF */
30                                 filterPtr = &myHPF[0];
31
32                                 /* Point delayline pointer to delayLineHPF */
33                                 delayLinePtr = &delayLineHPF[0];
34
35                                 /* Set my NH to number of high-pass coefficients */
36                                 myNH = HPF_NH;
37
38                                 /* Set filtState to high-pass */
39                                 filtState = 1;
40                         }
41                         sw2State = 0;      // Set state to 0 to allow only single press
42                 }
43         } else                          // SW2 not pressed
44         {
45                 sw2State = 1;           // Set state to 1 to allow tone change
46         }
```

Listing 5: IDL thread for monitoring switch two strokes and changing filters

# 3 Results

## 3.1 CPU Usage

Both the rx and the tx HWIs required 16 cycles per sample. With a 100 MHz CPU and 48 kHz codec, 2,083 cycles per sample were available; hence, each HWI used 0.77 percent of the CPU time.

The SWIs required approximately 20,000 cycles per frame of samples. At 48 samples per frame, each sample required 417 cycles and the CPU load for the SWIs was 20 percent.

### 3.2 Execution Graph

To implement an execution graph, a LED was turned off when the BIOS scheduler entered a thread and was turned back on when it returned from the thread. Therefore, on a logic analyzer the line was pulled high when entering a thread and pushed low when exiting. The red LED was not used because the hardware pulled the line low enough to turn off the LED but did not pull it low enough to register as low on the logic analyzer. As a result, on three threads were monitored at a time.

Figure 2 shows an execution graph of the I2S rx HWI (top), both ping and pong SWIs (middle), and both IDL threads (bottom). The HWI and the SWI graphs were similar to what was expected. As the measurement shows, every 48 times the HWI occured another SWI was posted and the HWI continued to occur while the SWI was running. In reality, each time the tx or rx HWI occurred the SWI graph would go low since the HWIs would have higher priority. Such a graph was generated, but it was difficult to decipher the graph and the graph proivded was generated instead. It was interesting to observe the amount of time that it took each IDL thread to run. As the lowest priority thread type, the IDL was continually interrupted and took longer to execut as a result. It became obvious that if too many higher priority threads were running then the IDL thread may never be reached.

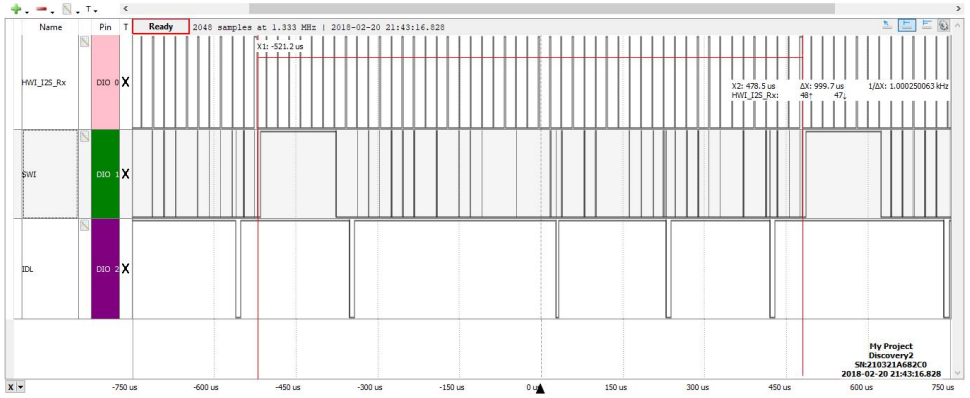


Figure 2: Execution of I2S rx HWI, ping and pong SWI, and both IDL threads

Figure 3 shows another execution graph with the I2S tx HWI (bottom) instead of the IDL threads. Note the measurement, which inidicates that the HWI is clocked at the codec sampling frequency of 48 kHz as expected.

Additionally, the time between the SWI thread going high and going low was measured to 168.7 microseconds. This time correspongs to a CPU load of 16.87 percent, which varied slightly from the aforementioned CPU load.

### 3.3 Demonstration

The project demonstrated as expected, except for the function of the buttons. When pressing switch two, especially when pressing rapidly, the board would register the press as switch one and perform the wrong operation. After investigating the issue, it was discovered that the design of the switches is less than desireable. Switch one is connected by a 20 kΩ resistor to an

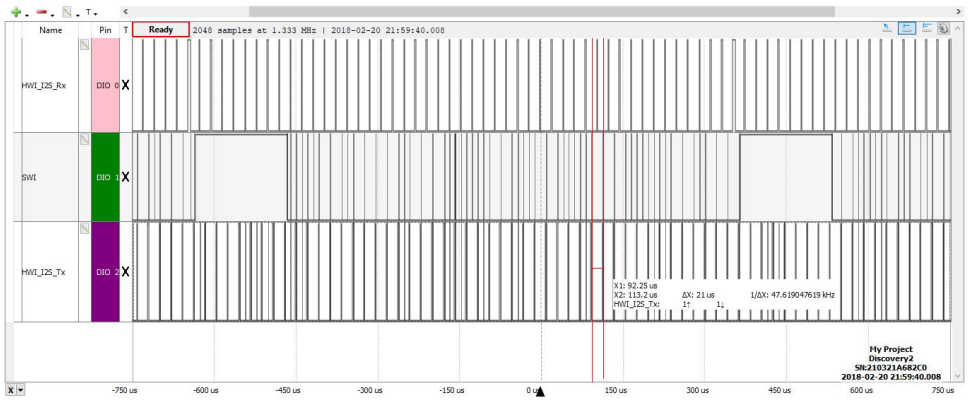


Figure 3: Execution of I2S rx HWI, ping and pong SWI, and I2S tx HWI threads

ADC and switch two is connected by a 10 kΩ resistor to the same ADC. The ADC values for switch one and two are close enough that switch two will occassionally generate a value higher than the switch one value. Therefore, the problem could not be fixed by changing the threshold values and the issue pursued no further.

# 4 Summary

This project introduced and showed the significance of using real-time operating systems. After completing the project, I have ideas of trying to implement RTOSs in other projects where I believe the scheduling nature may be useful and appropriate.

# 5 Appendix

```c
#include <std.h>

#include <log.h>

#include "hellocfg.h"
#include "ezdsp5535.h"
#include "ezdsp5535_led.h"
#include "ezdsp5535_sar.h"
#include "ezdsp5535_i2s.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!");

    /* Initialize BSL */
    EZDSP5535_init( );

    /* init LEDs and set to off*/
    EZDSP5535_LED_init( );
    EZDSP5535_LED_setall(0x0F);

    /* init dip switches */
    EZDSP5535_SAR_init( );

    // configure the Codec chip
    ConfigureAic3204();

    /* Initialize I2S */
    EZDSP5535_I2S_init();

    /* enable the interrupt with BIOS call */
    C55_enableInt(14); // reference technical manual, I2S2 tx interrupt
    C55_enableInt(15); // reference technical manual, I2S2 rx interrupt

    audioProcessingInit();

    // after main() exits the DSP/BIOS scheduler starts
}
```

Listing 6: main.c

```c
/*
 *  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 "stdint.h"
#include "string.h"
#include "hellocfg.h"
#include "ezdsp5535.h"
#include "ezdsp5535_gpio.h"
#include "ezdsp5535_led.h"
#include "ezdsp5535_sar.h"
#include "ezdsp5535_i2s.h"
#include "csl_i2s.h"
#include "csl_gpio.h"
#include "aic3204.h"
#include "filters.h"

#define NX                      48
#define BL_ONLY         0x0E
#define YL_ONLY         0x0D
#define RD_ONLY         0x0B
#define GR_ONLY         0x07
#define MIN(a,b) ( ((a) <  (b)) ? (a) : (b) )

extern CSL_I2sHandle    hI2s;
extern void myfir(const int16_t* input, const int16_t* filterCoeffs,
                int16_t* output, int16_t* delayLine, uint16_t  nx, uint16_t  nh);
extern int16_t myNCO(uint16_t f_tone);
extern void clearPA(void);

int16_t rxPingPong[96];
int16_t txPingPong[96];

int16_t delayLineLPF[NX + LPF_NH - 1];
int16_t delayLineHPF[NX + HPF_NH - 1];

int16_t rxIndex;
int16_t txIndex;

uint16_t sw1State  = 0;         // SW1 state
uint16_t sw2State  = 0;         // SW2 state
uint16_t filtState = 0;                 // filter state

int16_t * filterPtr;
int16_t * delayLinePtr;
uint16_t myNH;

Uint16 idl = 0, swi = 0;

/*
 * audioProcessingInit
 *
```

```
68  * @brief:      Initialize arrays used for filtering and transmitting
69  *                      and initialize array indices to 0.
70  */
71 void audioProcessingInit(void)
72 {
73         /* Initialize arrays as empty*/
74         memset(txPingPong, 0, sizeof(txPingPong));
75         memset(delayLineLPF, 0, sizeof(delayLineLPF));
76         memset(delayLineHPF, 0, sizeof(delayLineHPF));
77
78         /* Initially select low-pass filter */
79         filterPtr = &myLPF[0];
80         delayLinePtr = &delayLineLPF[0];
81         myNH = LPF_NH;
82
83         /* Initialize rx and tx indices to 0 */
84         rxIndex = 0;
85         txIndex = 0;
86 }
87
88 /*************************************************************************
89  ******************                                    HWIs
        ******************
90  *************************************************************************/
91
92 /*
93  * HWI_I2S_Rx
94  *
95  * @brief:      Function handle for HWI 15. Stores received samples into a
96  *                      double buffer and post SWIs to perform filtering.
97  */
98 void HWI_I2S_Rx(void)
99 {
100        /* Blue LED off */
101        EZDSP5535_GPIO_setOutput( 14, 1 );
102
103        /* Turn on yellow (SWI) and green (IDL) LED */
104        EZDSP5535_GPIO_setOutput( 15, 0 );
105        EZDSP5535_GPIO_setOutput( 17, 0 );
106
107        /* Read right sample and disregard. Read left sample and store
108         * in rxPingPong.
109         * Ping -> first 48 samples in array (0 - 47)
110         * Pong -> second 48 samples in array (48 - 97)
111         */
112        volatile int16_t temp;
113        temp = hI2s->hwRegs->I2SRXRT1;
114        rxPingPong[rxIndex++] = hI2s->hwRegs->I2SRXLT1;
115
116        if (rxIndex == 48)              //Have 48 samples been collected
117        {
118                /* Ping is full -> Post SWIPing to run SWI that will
119                 * filter the ping samples.
120                 */
121                SWI_post(&SWIPing);
122        }
123        if (rxIndex == 96)
124        {
125                /* Pong is full -> Post SWIPong to run SWI that will
126                 * filter the pong samples. Clear rxIndex so rxPingPong
127                 * will begin filling ping again.
128                 */
129                SWI_post(&SWIPong);
130                rxIndex = 0;
131        }
132
133        /* Blue LED on */
134        EZDSP5535_GPIO_setOutput( 14, 1 );
```

```
135
136         /* Return yellow (SWI) and green (IDL) LED to previous state */
137         EZDSP5535_GPIO_setOutput( 15, swi );
138         EZDSP5535_GPIO_setOutput( 17, idl );
139 }
140
141 /*
142  * HWI_I2S_Tx
143  *
144  * @brief:       Function handle for HWI 14. Transmits filtered samples
145  *                      from a double buffer.
146  */
147 void HWI_I2S_Tx(void)
148 {
149         //EZDSP5535_GPIO_setOutput( 17, 1 );
150
151         /* Transmit filtered samples */
152         hI2s->hwRegs->I2STXLT1 = txPingPong[txIndex];
153         hI2s->hwRegs->I2STXRT1 = txPingPong[txIndex++];
154
155         if (txIndex == 96)              //Have 96 samples been transmitted?
156         {
157                 /* Set index to beginning of tx array */
158                 txIndex = 0;
159         }
160
161         //EZDSP5535_GPIO_setOutput( 17, 0 );
162 }
163
164 /************************************************************************
165  ******************                                    SWIs
         ******************
166  ************************************************************************/
167
168 void SWIPingFunc(void)
169 {
170         /* swi = 1 -> SWI thread is running -> turn yellow LED off */
171         swi = 1;
172     EZDSP5535_GPIO_setOutput( 15, swi );
173
174     /* Turn green (IDL) LED on */
175     EZDSP5535_GPIO_setOutput( 17, 1 );
176
177     /* Filter a frame of 48 received samples and store output in tx buffer
178      *  using my fir. Variables filterPtr and delayLinePtr point to the desired
179      *  filter (LPF or HPF) and it's corresponding delayline (selected in second IDL
180      *  thread).
181      */
182         myfir(&rxPingPong[0], filterPtr, &txPingPong[0], delayLinePtr, NX, myNH);
183
184 //      swi = 0;
185         EZDSP5535_GPIO_setOutput( 15, 0 );
186 //      EZDSP5535_GPIO_setOutput( 17, idl );
187 }
188
189 void SWIPongFunc(void)
190 {
191 //    swi = 1;
192     EZDSP5535_GPIO_setOutput( 15, 1 );
193 //    EZDSP5535_GPIO_setOutput( 17, 0 );
194
195     /* Filter a frame of 48 received samples and store output in tx buffer
196      *  using my fir. Variables filterPtr and delayLinePtr point to the desired
197      *  filter (LPF or HPF) and it's corresponding delayline (selected in second IDL
198      *  thread).
199      */
200         myfir(&rxPingPong[48], filterPtr, &txPingPong[48], delayLinePtr, NX, myNH);
201
```

```
202         /* swi = 0 -> SWI thread finished -> turn yellow LED on */
203         swi = 0;
204         EZDSP5535_GPIO_setOutput( 15, 0 );
205
206         /* return green (IDL) LED to previous state */
207         EZDSP5535_GPIO_setOutput( 17, idl );
208 }
209
210 /************************************************************************
211  ******************                                    IDLs
         ******************
212  ************************************************************************/
213
214 void monitorSW1(void)
215 {
216         /* idl = 1 -> IDL thread running -> turn green LED off */
217     idl = 1;
218     EZDSP5535_GPIO_setOutput( 17, 1 );
219
220         /* Check SW1 */
221         if(EZDSP5535_SAR_getKey( ) == SW1) // Is SW1 pressed?
222         {
223                 if(sw1State)                // Was previous state not pressed?
224                 {
225                         int16_t msec, sample;
226
227                         /* Disable SWIs and HWIs -- store HWI state */
228                         SWI_disable( );
229                         Uns oldState1 = HWI_disable( );
230
231                         /* nested for loop to play 1 kHz tone for 1 sec */
232                         for ( msec = 0 ; msec < 1000 ; msec++ )
233                         {
234                                 for ( sample = 0 ; sample < 48 ; sample++ )
235                                 {
236                                         /* call myNCO to get 1 kHz tone sample */
237                                         int16_t temp = (myNCO( 1000 ) >> 5);
238
239                         /* Write 16-bit left channel Data */
240                         EZDSP5535_I2S_writeLeft( temp );
241
242                         /* Write 16-bit right channel Data */
243                         EZDSP5535_I2S_writeRight( temp );
244                                 }
245                         }
246                         /* clear phase accumulator */
247                         clearPA( );
248
249                         /* clear both delayLines and reset rx and tx indices */
250                         memset(delayLineLPF, 0, sizeof(delayLineLPF));
251                         memset(delayLineHPF, 0, sizeof(delayLineHPF));
252                         rxIndex = 0;
253                         txIndex = 0;
254
255                         /* restore HWIs to previous state and enable SWIs */
256                         HWI_restore( oldState1 );
257                         SWI_enable( );
258
259                         sw1State = 0;      // Set state to 0 to allow only single press
260                 }
261         } else                          // SW1 not pressed
262         {
263                 sw1State = 1;           // Set state to 1 to allow timer change
264         }
265
266         /* idl = 0 -> IDL thread finished -> turn green LED on */
267     idl = 0;
268     EZDSP5535_GPIO_setOutput( 17, 0 );
```

```
269 }
270
271 void monitorSW2(void)
272 {
273         /* idl = 1 -> IDL thread running -> turn green LED off */
274         idl = 1;
275     EZDSP5535_GPIO_setOutput( 17, 1 );
276
277         /* Check SW2 */
278         if(EZDSP5535_SAR_getKey( ) == SW2) // Is SW2 pressed?
279         {
280                 if(sw2State)            // Was previous state not pressed?
281                 {
282                         if(filtState)           //Was previous state High-pass?
283                         {
284                                 /* Clear Low-pass delayLine */
285                                 memset(delayLineLPF, 0, sizeof(delayLineLPF));
286
287                                 /* Point filter pointer to myLPF */
288                                 filterPtr = &myLPF[0];
289
290                                 /* Point delay line pointer to delayLineLPF */
291                                 delayLinePtr = &delayLineLPF[0];
292
293                                 /* Set myNH to number of low-pass coefficients */
294                                 myNH = LPF_NH;
295
296                                 /* Set filtState to low-pass */
297                                 filtState = 0;
298                         } else                          //Was previous state low-pass
299                         {
300                                 /* Clear high-pass delay line */
301                                 memset(delayLineHPF, 0, sizeof(delayLineHPF));
302
303                                 /* Point filter pointer to myHPF */
304                                 filterPtr = &myHPF[0];
305
306                                 /* Point delayline pointer to delayLineHPF */
307                                 delayLinePtr = &delayLineHPF[0];
308
309                                 /* Set my NH to number of high-pass coefficients */
310                                 myNH = HPF_NH;
311
312                                 /* Set filtState to high-pass */
313                                 filtState = 1;
314                         }
315                         sw2State = 0;      // Set state to 0 to allow only single press
316                 }
317         } else                          // SW2 not pressed
318         {
319                 sw2State = 1;           // Set state to 1 to allow tone change
320         }
321
322         /* idl = 0 -> IDL thread finished -> turn green LED on */
323         idl = 0;
324     EZDSP5535_GPIO_setOutput( 17, 1 );
325 }
326
327 /************************************************************************
328  ******************                                    TSKs
         ******************
329  ************************************************************************/
```

Listing 7: audioProcessing.c

```
1  /*****************************************************************************
2  ****                              I N C L U D E S
3  *****************************************************************************/
4
5  /*****************************************************************************
6  ****                            D E F I N I T I O N S
7  *****************************************************************************/
8  #define LPF_NH          126
9  #define HPF_NH          125
10 /*****************************************************************************
11 ****                        G L O B A L   V A R I A B L E S
12 *****************************************************************************/
13
14 int16_t myLPF[] =
15 {
16       -23,    -13,    -16,    -20,    -24,    -28,    -33,    -38,    -43,    -49,
             -55,    -60,    -66,    -72,    -77,    -82,
17       -86,    -90,    -93,    -95,    -96,    -95,    -93,    -89,    -84,    -77,
             -68,    -56,    -42,    -26,     -8,     13,
18        36,     62,     90,    121,    154,    189,    226,    265,    306,    348,
              391,    435,    480,    525,    571,    615,
19       660,    703,    745,    785,    823,    859,    892,    922,    950,    973,
             993,   1010,   1022,   1030,   1034,   1034,
20      1030,   1022,   1010,    993,    973,    950,    922,    892,    859,    823,
            785,    745,    703,    660,    615,    571,
21       525,    480,    435,    391,    348,    306,    265,    226,    189,    154,
             121,     90,     62,     36,     13,     -8,
22       -26,    -42,    -56,    -68,    -77,    -84,    -89,    -93,    -95,    -96,
             -95,    -93,    -90,    -86,    -82,    -77,
23       -72,    -66,    -60,    -55,    -49,    -43,    -38,    -33,    -28,    -24,
             -20,    -16,    -13,    -23
24 };
25
26 int16_t myHPF[] =
27 {
28       -24,   -147,      6,    -15,     -6,      1,     10,     20,     31,     43,
             53,     62,     69,     73,     73,     69,
29        60,     46,     27,      5,    -21,    -49,    -78,   -106,   -132,   -153,
              -169,   -177,   -176,   -166,   -144,   -113,
30       -71,    -20,     38,    102,    168,    233,    294,    347,    388,    414,
             420,    405,    366,    300,    209,     91,
31       -53,   -219,   -404,   -605,   -817,  -1034,  -1251,  -1460,  -1657,  -1835,
             -1989,  -2115,  -2207,  -2263,  30486,  -2263,
32     -2207,  -2115,  -1989,  -1835,  -1657,  -1460,  -1251,  -1034,   -817,   -605,
           -404,   -219,    -53,     91,    209,    300,
33       366,    405,    420,    414,    388,    347,    294,    233,    168,    102,
             38,    -20,    -71,   -113,   -144,   -166,
34      -176,   -177,   -169,   -153,   -132,   -106,    -78,    -49,    -21,      5,
            27,     46,     60,     69,     73,     73,
35        69,     62,     53,     43,     31,     20,     10,      1,     -6,    -15,
                   6,   -147,    -24
36 };
37 /*****************************************************************************
38 ****          F U N C T I O N   D E C L A R A T I O N S
39 *****************************************************************************/
```

Listing 8: 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 9: myfir.c

```
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 10: myNCO.c