BIOS Multi-thread FIR

ECEN 498: Real-Time Digital Signal Processing
University of Nebraska-Lincoln
Department of Electrical and Computer Engineering

Landon Burk, Evan Cornwell, Tess Jisa
April 3, 2023

Contents

1 Introduction

In previous projects using the DSP/BIOS, a combination of both hardware interrupts and task threads to filter input audio and output the filtered samples, was used. There were different filters being toggled using one of the switches built onto the ezDSP board. This project utilizes more of the functions build-in to the DSP/BIOS. The added functions that will be used for this project are Mailboxes (MBX), Periodic Functions (PRD), and Semaphores (SEM).

Mailboxes are a special memory location that one or more tasks can use to transfer data, or more generally for synchronization. The difference between the pend operation and simply polling the mailbox location is that the pending task is suspended while waiting for the data to appear. Generally, three operations are possible with a mailbox: initialization, sending a message(POST), and waiting for a message (PEND). An important thing to note about mailboxes is that although multiple tasks can post to a mailbox, only one can receive the message. Additionally, a task desiring a message from an empty mailbox is suspended and placed on a waiting list until a message is received.

Periodic functions are a special type of software interrupt that provides non-preemptive scheduling. Perodic functions operate on a set interval, meaning that a desired time interval is set using the TCONF tool and the thread is executed whenever the specified time has passed. These types of functions are ideal for tasks that need to be run at set intervals.

Semaphores are a signal used to control access to tasks and interrupts as well as avoid problems with critical sections of code, this makes them ideal for task synchronization. The most common type of semaphore is a binary semaphore, which is used when there is only one shared resource. Another type of semaphore is the counting semaphore, which is used to handle more than one shared resource at the same time. A semaphore has three operations Create, Acquire (PEND), and Release (POST).

2 Program Description

The goal for this project is to utilize the concepts from previous projects like Hardware interrupts, software interrupts, and audio filtering, as well as the new concepts mentioned above to be able to efficiently read, process, and write samples. Additionally, another important aspect of this project is to manage the user interface in a way that does not disrupt the audio processing. Finally, measurements of the MIPS (Million Instructions Per Second) and memory usage will be gathered to quantify the performance of the project.

2.1 HWI Threads

There were two HWI threads used for this project, one for the RX line and one for the TX line. The RX thread would check to see which channel was open, save 48 samples from the left channel into a buffer, and post that buffer to a mailbox. The TX thread waited to receive the samples posted in a second mailbox from the audio processing task. Since the samples came in an array, each one of the 48 was parsed out and written to the output. For both the RX and TX thread, it was decided to only work with left-channel audio. The right channel still had to be read and have something written to it in order for the flags to change so the right channel samples in RX were ignored and the left channel samples were written to the right channel in TX.

2.2 Mailboxes

Mailboxes were introduced in this project. A mailbox are useful because the message that is sent to it can be generic, as in it doesn't matter what type of message is being sent to it. Mailboxes were implemented in a few different places. First, the Rx HWI thread utilized mailboxes by reading in some samples and sending them to the audio processing task by posting the samples in the RX mailbox. Then, at the beginning of the audio processing task, the mailbox would wait for a new set of samples so they could be filtered by pending the RX mailbox. After the filtering was complete, the filtered samples were sent (posted) to the Tx HWI thread via a second TX mailbox, where the Tx HWI thread would receive the samples to post them to the output. It is important to note that there were two mailboxes needed to complete this, one for the raw samples, and one for the filtered samples. These mailboxes introduced a nice and efficient way to receive and process incoming samples.

2.3 Task Threads

A few different task threads were implemented in this project. Task objects are nice because they allow the user to store local variables and make blocking possible. In addition, they allow interrupts to run on the system stack. This concept was used to implement the user interface and the audio processing function, which is different from the previous project.

2.3.1 User Interface

The handling of the user interface is done using a task thread. This user interface first calls the "TSK_sleep()" such that the task will only be called at least every 50 milliseconds. Then the task pends on a semaphore. Next, the SW0 switch is read from the GPIO. If the button is being pressed it will then increment the `filterMode` global variable. Finally, the `filterMode` variable is passed trough a switch statement to change the LEDs to reflect the filter being used. Then, depending on `flgaHighakadanger` from the PRD thread, the fourth LED is toggled on or off. Finally, the semaphore post function is called to release control. The priority selected for the user interface was 1 because if there was any slight latency, it is not something that would be noticed by the user.

2.3.2 Audio Processing

To process the incoming audio, a new task was created. A while loop was entered because the task was not going to be returned from. After receiving samples from the mailbox, the global variable tracking button presses were used to pass the audio through a low pass filter, high pass filter, or no filter. The function used to filter the incoming audio was from the dsplib called "fir2()". This function could take and process more than one sample at a time using the coefficients derived from previous projects. Once completed, the filtered samples were posted to a new mailbox specifically for outgoing samples. The memory usage of this task was 2,560 bytes in DARAM. When measuring the MIPs used in the thread, the software and hardware interrupts had to be turned off so that only the task is measured. Without going through a filter, the number of cycles was 199 or .000199 MIPS. Passing through a filter was 3,511 cycles or .003511 MIPS. The priority for this thread was chosen to be 8. It was important that this task had a higher priority than user interface because audio processing needs to happen much faster but it did not need to have the absolute highest level.

2.4 PRD

The PRD BIOS module is a periodic function manager. For this project it was used to blink a light at 2Hz, and this tick frequency was specified in the .tcf file. In the userInterface task, this module was called to set flags and turn an led on and off every .5 seconds. It was chosen to do this because protecting the thread would be difficult. Adding a PRD automatically adds a software interrupt, which is incompatible with the `SYS_FOREVER` timeout option. The PRD would have to be protected because there was another thread trying to write to the i2c bus and if the two actions were to clash, the code could break down. Thus, the LEDs were actually written high and low in the user interface task thread.

3 Summary

All in all, this project was a success. Many different changes were added to the previous project in order to make this one more efficient. New ideas like tasks and mailboxes were introduced to do this. Mailboxes were an especially powerful tool that was used to increase the cycle efficiency of the audio processing. In addition, a periodic function was introduced to blink an LED while the audio processing was happening. All filters worked as expected and the periodically blinking LED blinked at the expected rate. Many valuable new tools were gained in this lab and they will be useful for the future of this class.

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 <clk.h>
#include <tsk.h>
#include <gbl.h>
#include <c55.h>
//#include "clkcfg.h"

#include "hellocfg.h"
#include "ezdsp5502.h"
#include "ezdsp5502_i2cgpio.h"
#include "stdint.h"
#include "aic3204.h"
#include "ezdsp5502_mcbsp.h"
#include "csl_mcbsp.h"
//#include "Dsplib.h"

#include "demo_filt.h"
#include "highPass.h"

extern void audioProcessingInit(void);

#pragma DATA_SECTION(delayLineLP, ".dbufferLP")
int16_t delayLineLP[70]={0};
#pragma DATA_SECTION(delayLineHP, ".dbufferHP")
int16_t delayLineHP[67]={0};

const int16_t* restrict demoFilterptr;
int16_t* restrict delayLineLPptr;
int16_t* restrict delayLineHPptr;
const int16_t highPass[];
const int16_t* restrict highPassptr;
volatile int counter = 0;


volatile int k;

void *memset(void *str, int c, size_t n);
void main(void)
{
```

```
57      /* Initialize BSL */
58      EZDSP5502_init( );
59
60      // configure the Codec chip
61      ConfigureAic3204();
62
63      /* Initialize I2S */
64      EZDSP5502_MCBSP_init();
65
66      /* enable the interrupt with BIOS call */
67      C55_enableInt(7); // reference technical manual, I2S2 tx interrupt
68      C55_enableInt(6); // reference technical manual, I2S2 rx interrupt
69
70      //audioProcessingInit();
71
72      EZDSP5502_I2CGPIO_configLine(  SW1, IN );
73      //init leds
74      EZDSP5502_I2CGPIO_configLine(  LED0, OUT );
75      EZDSP5502_I2CGPIO_configLine(  LED1, OUT );
76      EZDSP5502_I2CGPIO_configLine(  LED2, OUT );
77
78      memset(delayLineLP, 0, sizeof delayLineLP);
79      memset(delayLineHP, 0, sizeof delayLineHP);
80
81      delayLineLPptr=delayLineLP;
82      delayLineHPptr=delayLineHP;
83      demoFilterptr=demoFilter;
84      highPassptr=highPass;
85      // after main() exits the DSP/BIOS scheduler starts
86
87  }
```

Listing 1: Main Code

```
1
2   #include <std.h>
3
4   #include <log.h>
5   #include <mbx.h>
6   #include <sem.h>
7
8   #include "hellocfg.h"
9   #include "ezdsp5502.h"
10  #include "stdint.h"
11  #include "aic3204.h"
12  #include "ezdsp5502_mcbsp.h"
13  #include "csl_mcbsp.h"
14  #include "Dsplib.h"
15
16
17  extern ushort fir2(DATA *, DATA *, DATA *, DATA *, ushort, ushort);
18  void *memcpy(void *dest, const void * src, size_t n);
19
20  extern MCBSP_Handle aicMcbsp;
21
22  int16_t rxRightSample;
23  int16_t rxLeftSample;
24  int16_t leftRightFlag = 0;
25  int16_t txleftRightFlag = 0;
```

```
26
27 //int16_t output;
28 int16_t outputLP;
29 int16_t outputHP;
30 int16_t filteredLeftSample[48]={0};
31 int16_t msg[48]={0};
32 int16_t output[48]={0};
33 Int16 filteredLeftSampleOutput;
34
35 extern int NCO;
36 extern int filterMode;
37 extern int16_t* delayLineLPptr;
38 extern int16_t* delayLineHPptr;
39 extern const int16_t* demoFilterptr;
40 extern const int16_t* highPassptr;
41 int txcounter=0;
42
43
44 int16_t bufferIn[48]={0};
45 volatile uint16_t indexIn=0;
46
47
48 void audioProcessingInit(void)
49 {
50     rxRightSample = 0;
51     rxLeftSample = 0;
52 }
53
54
55 void HWI_I2S_Rx(void)
56 {
57
58     if (leftRightFlag == 0)
59     {
60         if (indexIn<48) //read in 48 samples to a buffer
61         {
62             bufferIn[indexIn] = MCBSP_read16(aicMcbsp);
63             leftRightFlag = 1;
64             indexIn++;
65         }
66
67         if(indexIn>=48)
68         {
69             indexIn=0;
70             MBX_post(&MBXAudio, bufferIn, 0);
71         }
72
73     }
74     else
75     {
76         rxRightSample = MCBSP_read16(aicMcbsp);
77         leftRightFlag = 0;
78     }
79 }
80
81 void HWI_I2S_Tx(void)
82 {
83     if (txleftRightFlag == 0)
84     {
```

```
85          if(txcounter<48)
86          {
87              filteredLeftSampleOutput=output[txcounter];
88              EZDSP5502_MCBSP_write(filteredLeftSampleOutput);
89              txcounter++;
90              txleftRightFlag = 1;
91          }
92          if(txcounter>=48)
93          {
94              txcounter=0;
95              MBX_pend(&MBXOutput, output, 0);
96          }
97      }
98      else
99      {
100         EZDSP5502_MCBSP_write(filteredLeftSampleOutput);
101
102         txleftRightFlag = 0;
103     }
104 }
105
106 void TSKAudioProcessorFxn(Arg value_arg)
107 {
108     while(1)//in general you don't return from task
109     {
110         MBX_pend(&MBXAudio, msg, SYS_FOREVER);
111         /* used for counting cycles
112         SWI_disable();
113         HWI_disable();
114         filterMode=1;
115         */
116
117         switch(filterMode){
118         case 1:
119             fir2((DATA *)&msg,
120                  (DATA *)demoFilterptr,
121                  (DATA *)&filteredLeftSample,
122                  (DATA *)delayLineLPptr,
123                  (ushort)48,
124                  (ushort)70);
125             break;
126         case 2:
127             fir2((DATA *)&msg,
128                  (DATA *)highPassptr,
129                  (DATA *)&filteredLeftSample,
130                  (DATA *)delayLineHPptr,
131                  (ushort)48,
132                  (ushort)67);
133             break;
134         default:
135             memcpy(filteredLeftSample,msg,48);
136             break;
137         }
138
139         MBX_post(&MBXOutput, filteredLeftSample, SYS_FOREVER);
140
141     }
142 }
```

Listing 2: Audio Processing

```
1
2  #include <std.h>
3
4  #include <log.h>
5  #include <clk.h>
6  #include <tsk.h>
7  #include <gbl.h>
8  #include <c55.h>
9  #include <sem.h>
10 #include <prd.h>
11
12 #include "hellocfg.h"
13 #include "ezdsp5502.h"
14 #include "stdint.h"
15 #include "aic3204.h"
16 #include "ezdsp5502_mcbsp.h"
17 #include "csl_mcbsp.h"
18 #include "Dsplib.h"
19 #include "ezdsp5502_i2cgpio.h"
20
21
22 int switch1;
23 int switch1Prev=1;
24 int filterMode=0;
25 int flagHighakadanger=0;
26
27
28 void TSKUserInterfaceFxn(Arg value_arg)
29 {
30     while(1)
31     {
32         TSK_sleep(50);
33
34         SEM_pend(&SEMI2C, SYS_FOREVER);
35
36         //read switches on i2c
37         switch1=EZDSP5502_I2CGPIO_readLine(SW1);
38
39             if(switch1 != switch1Prev)
40             {
41                 if(!switch1)
42                 {
43                     filterMode++;
44                     if(filterMode>2)
45                     {
46                         filterMode=0;
47                     }
48                     switch(filterMode){
49                     case 0:
50                         EZDSP5502_I2CGPIO_writeLine(   LED0, LOW );
51                         EZDSP5502_I2CGPIO_writeLine(   LED1, HIGH );
52                         EZDSP5502_I2CGPIO_writeLine(   LED2, HIGH );
53                     break;
54                     case 1:
55                         EZDSP5502_I2CGPIO_writeLine(   LED0, HIGH );
```

```
56                          EZDSP5502_I2CGPIO_writeLine(   LED1, LOW );
57                          EZDSP5502_I2CGPIO_writeLine(   LED2, HIGH );
58                      break;
59                      case 2:
60                          EZDSP5502_I2CGPIO_writeLine(   LED0, HIGH );
61                          EZDSP5502_I2CGPIO_writeLine(   LED1, HIGH );
62                          EZDSP5502_I2CGPIO_writeLine(   LED2, LOW );
63                      break;
64                      }
65                  }
66                  switch1Prev=switch1;
67              }
68
69          EZDSP5502_I2CGPIO_writeLine(   LED3, flagHighakadanger );
70          SEM_post(&SEMI2C);
71      }
72  }
73
74
75  void PRDLedFxn(void)
76  {
77      //set flags to toggle led
78      if(flagHighakadanger==0)
79      {
80          flagHighakadanger=1;
81      }
82      else
83      {
84          flagHighakadanger=0;
85      }
86  }
```

Listing 3: User Interface