FFT + LCD

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

Landon Burk, Evan Cornwell, Tess Jisa
April 19, 2023

Contents

1 Introduction

Unlike the previous two projects, this project takes a detour from focusing on the DSP/BIOS and proceeds to introduce spectral analysis. In order to do a spectral analysis of a signal, the Fourier transform (FT) needs to be implemented. The Fourier transform is a mathematical transform, derived from the Fourier series, that is capable of extracting the frequencies present in a signal. For digital applications, the Discrete Fourier Transform (DFT) needs to be implemented, this transform does the same thing as the FT, but the DFT converts a finite sequence of samples into an equal-length finite sequence of frequency components. The major drawback to using the DFT is that it requires a lot of operations to compute, about O(N^2). To fix this issue the Fast Fourier Transform (FFT) was developed, this transform exploits the symmetry property of sinusoids to reduce the number of operations to about O(2N), making the FFT a much more useful algorithm especially when applied on embedded systems.

To be able to visualize the spectrum, the LCD display built into the eZDSP development board will be utilized. This display is 16-bits high, 96 bits wide, and is communicated with via I2C.

The goal for this project is to implement an FFT program generated using MatLabs simulink software onto the C55xx microcontroller, then output the spectrum on the LCD built onto the eZDSP development board. Additionally, specifications from previous projects such as implementing a high pass and low pass filter, and being able to switch between them using the built-in buttons will be implemented into this project.

2 Program Description

The bulk of our final program was created throughout previous projects. The same structure was kept to filter samples: mailboxes were used to pass unfiltered samples through different filters and a different mailbox posted those samples to be output in real-time. In addition, the filter used was chosen from a user interface task that utilized the I2C bus to monitor button presses. For each button press, the filter changed and there were three different LEDs that would light up to correspond to each filter change.

The major change between this project and the last one was that a third mailbox was used to send audio samples from the audio processing task to a spectral analysis task. The spectral analysis task was added in order to compute the FFT of the samples sent to it and to send those to the LCD to be displayed in real-time.

This program used more memory than previous projects at 90% of DARAM. The priorities for signal processing and user interface stayed the same at 15 and 8 respectively. The FFT task needed to have a lower priority than the user interface so it was chosen to be 7.

2.1 Spectral Analysis Task

The spectral analysis task was implemented in order to compute the FFT of incoming audio samples in real-time. This was done by creating a new BIOS task called "TSKFFTfxn" and then utilizing a mailbox to receive audio samples from the main audio processing task. The mailbox would copy 128 samples at a time into the built-in FFT_U.In1 variable, and utilizing the built-in FFT library, the samples were processed and sent to the spectrum's output.

2.2 Displaying FFT

The spectrum of the signal (the result of the FFT) was displayed on the eZDSP's built-in LCD. This display has a size of 96 pixels wide and 16 pixels high, but for this application, only 64 frequency bins need to be displayed due to the symmetry of the FFT around the Nyquist sampling rate making the other 64 frequency bins redundant. Given that a sampling rate of 48000 samples per second was used, in theory, the frequency range shown on the display is 0 Hz - 24000Hz. In practice, it was found using a tone generator that the maximum frequency that could fully be displayed was about 12,000 Hz. Using the `PRD_getticks()` function, the frame rate was found to be 30.

The first step to displaying the FFT on the LCD was to initialize the display. For this, the LCD initialization function provided in one of the example programs was used. Small adjustments had to be made to the LCD initialization function to make the display most suitable for our applications. First, the addressing mode of the display had to be set to vertical addressing mode. Next, the scan direction of the display had to be changed, this vertically flips the display. Finally, the segment remap function was used to horizontally flip the display. The last two adjustments made it so that the bottom left corner of the display was the 0th frequency bin.

The next step was to assign a frequency magnitude to a binary vector that could be used to output the magnitude on the display. In order to do this an array had to be made to store the bit maps. This array had a length of 16 and included the 16 possible bit maps from 0x0000 to 0xFFFFF. Next, to assign a bit vector to a magnitude, a frequency bin was first divided by 2048, then multiplied by 16. The result of this operation was then used to index the bit map array, and the bit map from that index was stored in another array that would then be sent to the display using a batch send function, which took 128 samples and sent them all at once. Additionally, to be able to utilize more of the LCD, each frequency bin has a width of 2 pixels.

3 Summary

Implementing the fast Fourier transform and displaying that to an LCD in real time were the highlights of this lab. To implement this, a new BIOS task was made to process and take the FFT of 128 incoming samples at a time. Then, these samples were sent to the LCD (128 at a time) utilizing the built-in eZDSP LCD function library. There was a bit of a difficulty with bit mapping and assigning bit vectors to magnitudes to output on the display, but this was completed nonetheless. In addition, the audio sounded clear and all of the filters worked as expected. All in all, this project was a success and this knowledge of real-time FFTs and LCD outputs will prove to be very useful (and quite fun) information moving forward.

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 "hellocfg.h"
#include "ezdsp5502.h"
#include "ezdsp5502_i2cgpio.h"
#include "stdint.h"
#include "aic3204.h"
#include "ezdsp5502_mcbsp.h"
#include "csl_mcbsp.h"
#include "lcd.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);
```

```
57  void main(void)
58  {
59      /* Initialize BSL */
60      EZDSP5502_init( );
61
62      // configure the Codec chip
63      ConfigureAic3204();
64
65      /* Initialize I2S */
66      EZDSP5502_MCBSP_init();
67
68      /* enable the interrupt with BIOS call */
69      C55_enableInt(7); // reference technical manual, I2S2 tx interrupt
70      C55_enableInt(6); // reference technical manual, I2S2 rx interrupt
71
72      osd9616_init( ); //lcd init
73      osd9616_send(0x00,0x2e);  // Deactivate Scrolling
74
75      /* Fill page 0 */
76      Int16 i;
77      i = osd9616_send(0x00,0x00);   // Set low column address
78      osd9616_send(0x00,0x10);   // Set high column address
79      osd9616_send(0x00,0xb0+0); // Set page for page 0 to page 5
80      for(i=0;i<192;i++)
81      {
82          osd9616_send(0x40,0xff);
83      }
84      /* Write to page 0 */
85      osd9616_send(0x00,0x00);   // Set low column address
86      osd9616_send(0x00,0x10);   // Set high column address
87      osd9616_send(0x00,0xb0+0); // Set page for page 0 to page 5
88      for(i=0;i<192;i++)
89      {
90          osd9616_send(0x40,0x00);  // Spaces
91      }
92
93      volatile int j = 0;
94      for(j=0;j<1000; j++){}// hard delay
95
96      osd9616_send(0x00,0xA0); //column address 0 mapped to seg0
97
98
99      audioProcessingInit();
100     //switch init
101     EZDSP5502_I2CGPIO_configLine(  SW1, IN );
102     //init leds
103     EZDSP5502_I2CGPIO_configLine(  LED0, OUT );
104     EZDSP5502_I2CGPIO_configLine(  LED1, OUT );
105     EZDSP5502_I2CGPIO_configLine(  LED2, OUT );
106
107     memset(delayLineLP, 0, sizeof delayLineLP);
108     memset(delayLineHP, 0, sizeof delayLineHP);
109
110     //declare filter pointers
111     delayLineLPptr=delayLineLP;
112     delayLineHPptr=delayLineHP;
113     demoFilterptr=demoFilter;
114     highPassptr=highPass;
115
```

```
116     // after main() exits the DSP/BIOS scheduler starts
117 }
```

Listing 1: Main Code

```
1  #include <stdio.h>
2  #include <stdint.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 #include "FFT.h"
16 #include "lcd.h"
17
18
19
20
21
22 extern ushort fir2(DATA *, DATA *, DATA *, DATA *, ushort, ushort);
23 void *memcpy(void *dest, const void * src, size_t n);
24
25 extern MCBSP_Handle aicMcbsp;
26
27 int16_t rxRightSample;
28 int16_t rxLeftSample;
29 int16_t leftRightFlag = 0;
30 int16_t txleftRightFlag = 0;
31
32 //int16_t output;
33 int16_t outputLP;
34 int16_t outputHP;
35 int16_t filteredLeftSample[48]={0};
36 int16_t msg[48]={0};
37 int16_t output[48]={0};
38 int16_t spectrum[128]={0};
39 int16_t FFTSamps[128]={0};
40 int16_t wave[128]={0}; //FOR TESTING PURPOSES
41 uint16_t spectrumOut[128]={0};
42 int16_t display[64]={0};
43 uint16_t buffcount=0;
44 Int16 filteredLeftSampleOutput;
45 LgUns ticksbefore;
46 LgUns ticksafter;
47
48 const uint16_t sally
       [16]={0,1,3,7,15,31,63,127,255,511,1023,2047,4095,8191,16383,32767};
49
50 extern int NCO;
51 extern int filterMode;
52 extern int16_t* delayLineLPptr;
53 extern int16_t* delayLineHPptr;
```

```
54  extern const int16_t* demoFilterptr;
55  extern const int16_t* highPassptr;
56  //volatile int indexIn;
57  int txcounter=0;
58
59  extern ExtU_FFT_T FFT_U;
60
61  int start;
62  int stop;
63  int time;
64
65  int16_t bufferIn[48]={0};
66  volatile uint16_t indexIn=0;
67
68  void *memcpy(void *dest, const void *src, size_t n);
69
70  //have to protect the two lights with a semaphore
71  void audioProcessingInit(void)
72  {
73      rxRightSample = 0;
74      rxLeftSample = 0;
75  }
76
77
78  void HWI_I2S_Rx(void)
79  {
80
81      if (leftRightFlag == 0)
82      {
83          if (indexIn<48)
84          {
85              bufferIn[indexIn] = MCBSP_read16(aicMcbsp);
86              leftRightFlag = 1;
87              indexIn++;
88          }
89
90          if(indexIn>=48)
91          {
92              indexIn=0;
93              MBX_post(&MBXAudio, bufferIn, 0);
94          }
95
96      }
97      else
98      {
99          rxRightSample = MCBSP_read16(aicMcbsp);
100         leftRightFlag = 0;
101     }
102 }
103
104 void HWI_I2S_Tx(void)
105 {
106     if (txleftRightFlag == 0)
107     {
108         if(txcounter<48)
109         {
110             filteredLeftSampleOutput=output[txcounter];
111             EZDSP5502_MCBSP_write(filteredLeftSampleOutput);
112             txcounter++;
```

```
113             txleftRightFlag = 1;
114         }
115         if(txcounter>=48)
116         {
117             txcounter=0;
118             MBX_pend(&MBXOutput, output, 0);
119         }
120     }
121     else
122     {
123         //rxRightSample = MCBSP_read16(aicMcbsp);
124         EZDSP5502_MCBSP_write(filteredLeftSampleOutput);
125 
126         txleftRightFlag = 0;
127     }
128 }
129 
130 void TSKAudioProcessorFxn(Arg value_arg)
131 {
132     while(1)
133     {
134         MBX_pend(&MBXAudio, msg, SYS_FOREVER);
135 
136         switch(filterMode){
137         case 1:
138             fir2((DATA *)&msg,
139                  (DATA *)demoFilterptr,
140                  (DATA *)&filteredLeftSample,
141                  (DATA *)delayLineLPptr,
142                  (ushort)48,
143                  (ushort)70);
144             break;
145         case 2:
146             fir2((DATA *)&msg,
147                  (DATA *)highPassptr,
148                  (DATA *)&filteredLeftSample,
149                  (DATA *)delayLineHPptr,
150                  (ushort)48,
151                  (ushort)67);
152             break;
153         default:
154             memcpy(filteredLeftSample,msg,48);
155             break;
156         }
157 
158         MBX_post(&MBXOutput, filteredLeftSample, SYS_FOREVER);
159 
160 //      volatile int x=0;
161 //      for(x;x<128;x++)
162 //      {
163 //          wave[x]=nco_run_sinusoid();
164 //      }
165         volatile int i=0;
166         for(i;i<48;i++)
167         {
168             spectrum[buffcount]=filteredLeftSample[i];
169             //spectrum[buffcount]=wave[i];
170             buffcount++;
171             if(buffcount==128)
```

```
172             {
173                 MBX_post(&MBXFFT, spectrum, 0);
174                 buffcount=0;
175             }
176         }
177     }
178 }
179
180 void TSKFFTfxn(Arg value_arg)
181 {
182     //call init
183     FFT_initialize();
184
185     uint16_t steve=0;
186     uint16_t sarah[128] = {0};
187     while(1)
188     {
189         MBX_pend(&MBXFFT, FFTSamps, 0);
190
191         memcpy(FFT_U.In1, FFTSamps, 128);
192
193         FFT_step();
194
195         memcpy(spectrumOut, FFT_Y.Out1, 128);
196
197         volatile int i=0;
198         volatile int j=0;
199         for(i=0;i<128;i+=4)
200         {
201             steve=(uint16_t)(((double)spectrumOut[j]/128.0));
202
203             sarah[i] = sally[steve&0xf]>>8;
204             sarah[i+1] = sally[steve&0xf] & 0xFF;
205             sarah[i+2] = sally[steve&0xf]>>8;
206             sarah[i+3] = sally[steve&0xf] & 0xFF;
207             j++;
208         }
209
210         SEM_pend(&SEMI2C, SYS_FOREVER);//blocking the leds from writing to i2c
                bus during screen transmission
211
212         //ticksbefore = PRD_getticks();
213         osd9616_send(0x00,0x21); //setting start address
214         osd9616_send(0x00,0x20);
215         osd9616_send(0x00,0x60); //end column
216         osd9616_send(0x00,0x22); //start page
217         osd9616_send(0x00,0x00);
218         osd9616_send(0x00,0x01);//end page
219
220
221         myosd9616_multiSend((Uint16*)sarah, 128);//send fft results to screen
222
223         //ticksafter = PRD_getticks();
224         SEM_post(&SEMI2C);
225     }
226 }
```

Listing 2: Audio Processing