BIOS Hardware Interrupt Based FIR

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

Landon Burk, Evan Cornwell, Tess Jisa
March 10, 2023

Contents

1 Introduction

This project served as an introduction to the concept of BIOS (basic input/output system) while bringing in functions from past projects. BIOS is a multi-thread real time operating system that is used by the eZDSP board, and will be used in this project. BIOS has a convenient hardware and software interrupt system (HWI and SWI, respectively) that will be leveraged to allow for more efficient real time processing of input signals. Having multiple threads allows the user to do concurrent tasks while not losing as much time as a typical build environment. This concept will be utilized in to monitor a button on the eZDP board to perform differing filtering functions on an input audio signal. In addition, a numerically controlled oscillator will be used to output a sinusoid when another button is pressed.

2 Program Description

2.1 IDL Thread

An IDL (idle) thread was used monitor buttons “SW0” and “SW1” on the DSP board. The state of each button was read at the top of the loop and saved so that interrupt flags could be set later on. To combat button bounces, the current button state was checked against the previous button state and if they were different, the change could be handled. When button SW0 registered a press, a NCO flag was set to high to trigger a change to sound being output in the HWI thread. When the SW1 button was pressed, a variable kept track of the number of times it was pressed so that three filter modes could be rotated through with the help of the HW thread. A switch statement changed the LED to correspond which each filter mode.

2.2 HWI Thread

The HWI thread was used to perform the FIR filters, an NCO and write the output samples. In the audio processing file, interrupts would fire to take in left and right samples and write them to the output audio jack. When the NCO flag was set high in the IDL thread, the NCO function was used to output a tone for 500ms in the output interrupt. At the end of the tone samples, the interrupt flag and counters were set back to zero so that the function would be ready for a new button press. When the filter mode changed, a switch statement was used to change between a high pass filter, low pass, and no filter on the sound being output. For both functions, only the left channel sample was output.

The number of cycles for the HWI thread was found with the counter under breakpoint type. The number varied slightly each time but it was around 4500 cycles to go from the top of `HWI_I2S_Rx` to the bottom of `HWI_I2S_Tx`. The CPU% used was $\frac{48k}{300M} * 4500 * 100 = 72\%$. The interrupts used were of type i2c GPIO and the i2c clock was initialized at 100 kHz.

2.3 Filters and NCO

For the high pass filter, an equiripple design was used with 1 dB ripple in the pass-band and 80 dB attenuation in the stop-band. In music, high frequency sound is considered to be 2 kHz and up, so in the high pass filter, the stop-band frequency was chosen to be 2kHz and the pass band frequency to be 4kHz. The longer transition band allowed for the number of filter coefficients to remain relatively low at only 67. The magnitude response is shown in Figure 1.

An equiripple design was used with 1 dB ripple in the pass-band and 80 dB attenuation in the stop-band. There was a passband frequency of 250 Hz and stop band frequency of 2000 Hz. This filter had 70 coefficients and the magnitude response is shown in Figure 2.

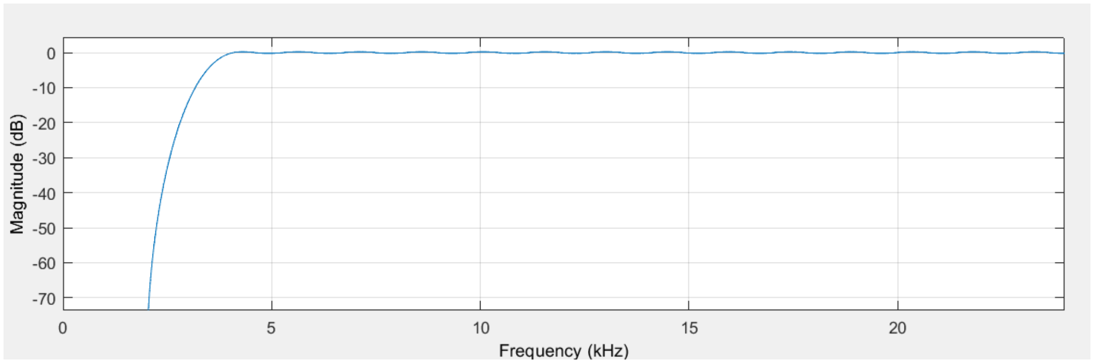

Figure 1: High Pass Filter Magnitude Response

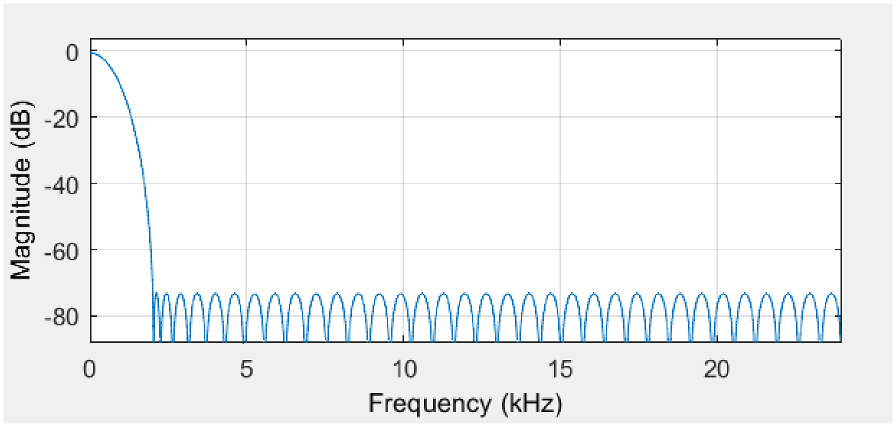

Figure 2: Low Pass Filter Magnitude Response

The NCO used for this project was the same NCO function that was used in the first project for this class. The sine wave output was chosen with a frequency of 1000Hz and an attenuation factor of zero that could be adjusted throughout the testing period.

3 Summary

Overall, this project was a success, the concept of BIOS was introduced and utilized to create efficient implementations of previously designed functions. This was done by using hardware interrupt threads to execute on different events, like a button being pressed, or a buffer filling up. With an incoming audio signal, the software was tested. And indeed, the results show that the processor processed the incoming audio samples quite efficiently. In addition, with the filter function, one could tell, just by listening to the audio, that it was being processed quite quickly in real-time, to the point that there was no perceptible latency from the input to the output (this was very fun to listen to with the different filters!) Overall, this project was a success, and interrupts via BIOS were added to the programming toolbelt; a useful tool indeed!

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

extern void audioProcessingInit(void);

volatile int counter = 0;
int switch0;
int switch1;
int switch0Prev=1;
int switch1Prev=1;
int NCO;
int filterMode=0;
int16_t delayLineLP[70]={0};
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 k;

void *memset(void *str, int c, size_t n);
void main(void)
{
    /* Initialize BSL */
    EZDSP5502_init( );

    // configure the Codec chip
    ConfigureAic3204();

    /* Initialize I2S */
    EZDSP5502_MCBSP_init();

    /* enable the interrupt with BIOS call */
    C55_enableInt(7); // reference technical manual, I2S2 tx interrupt
    C55_enableInt(6); // reference technical manual, I2S2 rx interrupt

    //audioProcessingInit();

    EZDSP5502_I2CGPIO_configLine(  SW0, IN );
    EZDSP5502_I2CGPIO_configLine(  SW1, IN );
    //init leds
    EZDSP5502_I2CGPIO_configLine(  LED0, OUT );
    EZDSP5502_I2CGPIO_configLine(  LED1, OUT );
    EZDSP5502_I2CGPIO_configLine(  LED2, OUT );

    //init NCO
    nco_set_frequency(1000);
    nco_set_attenuation(3);

    memset(delayLineLP, 0, sizeof delayLineLP);
    memset(delayLineHP, 0, sizeof delayLineHP);

    delayLineLPptr=delayLineLP;
    delayLineHPptr=delayLineHP;
    demoFilterptr=demoFilter;
    highPassptr=highPass;
    // after main() exits the DSP/BIOS scheduler starts
}

Void taskFxn(Arg value_arg)
{
    LgUns prevHtime, currHtime;
    uint32_t delta;
    float ncycles;

    /* get cpu cycles per htime count */
    ncycles = CLK_cpuCyclesPerHtime();

    while(1)
    {
        TSK_sleep(1);
        LOG_printf(&trace, "task running! Time is: %d ticks", (Int)TSK_time());

        prevHtime = currHtime;
        currHtime = CLK_gethtime();

        delta = (currHtime - prevHtime) * ncycles;
        LOG_printf(&trace, "CPU cycles = 0x%x %x", (uint16_t)(delta >> 16), (
            uint16_t)(delta));

    }
}

void myIDLThread(void)
{
    counter++;


    switch0=EZDSP5502_I2CGPIO_readLine(SW0);
    switch1=EZDSP5502_I2CGPIO_readLine(SW1);
    if(switch0 != switch0Prev)
    {
        if(!switch0)
        {
            NCO=1;
        }
        switch0Prev=switch0;
    }

    if(switch1 != switch1Prev)
    {
        if(!switch1)
        {
            filterMode++;
            if(filterMode>2)
            {
                filterMode=0;
            }
            switch(filterMode){
            case 0:
                EZDSP5502_I2CGPIO_writeLine(   LED0, LOW );
                EZDSP5502_I2CGPIO_writeLine(   LED1, HIGH );
                EZDSP5502_I2CGPIO_writeLine(   LED2, HIGH );
            break;
            case 1:
                EZDSP5502_I2CGPIO_writeLine(   LED0, HIGH );
                EZDSP5502_I2CGPIO_writeLine(   LED1, LOW );
                EZDSP5502_I2CGPIO_writeLine(   LED2, HIGH );
            break;
            case 2:
                EZDSP5502_I2CGPIO_writeLine(   LED0, HIGH );
                EZDSP5502_I2CGPIO_writeLine(   LED1, HIGH );
                EZDSP5502_I2CGPIO_writeLine(   LED2, LOW );
            break;
            }
        }
        switch1Prev=switch1;
    }
}
```

Listing 1: Main Code

```

#include <std.h>

#include <log.h>

#include "hellocfg.h"
#include "ezdsp5502.h"
```

```c
#include "stdint.h"
#include "aic3204.h"
#include "ezdsp5502_mcbsp.h"
#include "csl_mcbsp.h"

#include "myNCO.h"
#include "myFIR.h"


extern MCBSP_Handle aicMcbsp;

int16_t rxRightSample;
int16_t rxLeftSample;
int16_t leftRightFlag = 0;
int16_t txleftRightFlag = 0;

int16_t output;
int16_t outputLP;
int16_t outputHP;
int16_t filteredLeftSample;

extern int NCO;
extern int filterMode;
extern int16_t* delayLineLPptr;
extern int16_t* delayLineHPptr;
extern const int16_t* demoFilterptr;
extern const int16_t* highPassptr;
volatile int NCO_counter=0;

void audioProcessingInit(void)
{
    rxRightSample = 0;
    rxLeftSample = 0;
}


void HWI_I2S_Rx(void)
{

    if (leftRightFlag == 0)
    {
        rxLeftSample = MCBSP_read16(aicMcbsp);
        leftRightFlag = 1;

        switch(filterMode){
        case 1:
            myFIR(&rxLeftSample,
                    demoFilterptr,
                    &filteredLeftSample,
                    delayLineLPptr,
                    1,
                    70);
                EZDSP5502_MCBSP_write(filteredLeftSample);
                break;
            case 2:
                myFIR(&rxLeftSample,
                        highPassptr,
                        &filteredLeftSample,
                        delayLineHPptr,
                        1,
                        67);
                EZDSP5502_MCBSP_write(filteredLeftSample);
                break;
            default:
                filteredLeftSample=rxLeftSample;
                EZDSP5502_MCBSP_write(filteredLeftSample);
                break;
            }

    }
    else
    {
        rxRightSample = MCBSP_read16(aicMcbsp);
        leftRightFlag = 0;
    }
}

void HWI_I2S_Tx(void)
{
        if(NCO)
        {
            if(NCO_counter<24000)
            {
                filteredLeftSample=nco_run_sinusoid();
                EZDSP5502_MCBSP_write(filteredLeftSample);
                NCO_counter++;
            }
            if(NCO_counter>=24000)
            {
                NCO_counter=0;
                NCO=0;
            }
        }
}
```

Listing 2: Audio Processing