Delay on DSP Board (DSK6713)

Executed by:Kevin Anum
Mailbox:108
DSP Lab
Natural Science Laboratory, Jacobs University Bremen

April 25, 2017

Introduction

Objective

In this lab we learn how to program a real-time DSP blocks in C.In this lab we used our knowledge of implementing a delay block in Matlab.

Background

Programming in C

In the this lab we are implementing a Delay on an output using the delay block.In this lab we used the same structure of implementing block in MATLAB.Our code has two main functions they are:

- delay_int()-This is mainly to initializes the block and return new state structures for the block.
- delay()-The function that generates the output samples.

In programming in C we will like to make our lives easier by defining global definitions in a header file.We also use # define to define values to variables like:

```
# define DELAY_BUFFER_SIZE       16384
```

This is used instead of:

```
int DELAY_BUFFER_SIZE =16384;
```

because it makes are dsp code efficient as possible.

Audio Processing on the TI DSP Boards

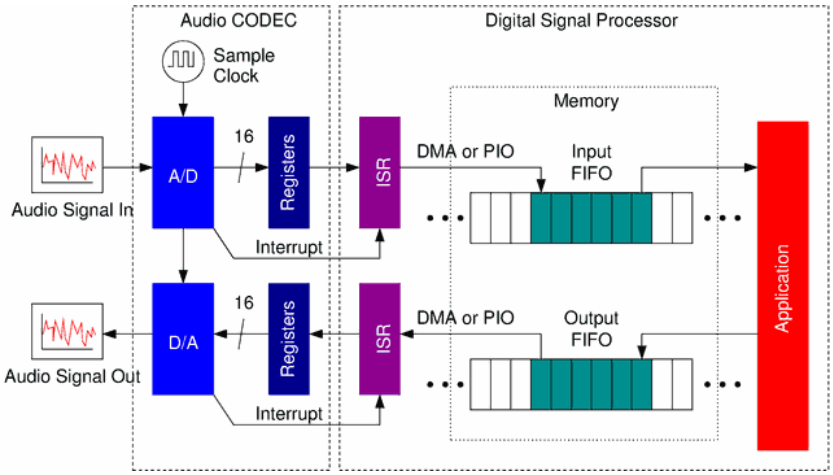

Figure 1: The diagram shows a flow chart of a DSP board.

The main components of the DSP board are:

- connectors for audio inputs
- Analog-to-digital converter
- Digital-to-Analog converter
- Audio CODEC(Coder and decoder)

Lab Write UP

Question 1

A few sentences explaining how to modify the golden project to support an audio streaming application.

In order for the project to support an audio stream we remove the commented the line(part of code now)

```
//#define  DSP_SOURCE   AIC23_REG4_LINEIN
```

and comment :

```
#define  DSP_SOURCE   AIC23_REG4_MIC
```

which are found in dsp_ ap

Since the frequency may defer the sampling rate will so you can change the sampling rate in order to sample the signal.Everything else will remain the same.

Question 2

A brief description of how blocks are implemented to support real-time processing in C. What are the two main functions for each block and what do they do? How is state stored ?

The two main functions are delay_ int() and delay().The state is stored in a struct.Below is an example:

```
typedef struct
{
    float buffer[DELAY_BUFFER_SIZE];
    unsigned int del;
    unsigned int h, t;
} delay_state_def;
```

Question 3

dsp_ap.h

```
/*=====================================================
* dsp_ap.h
* Contains global definitions for your DSP application.
* Here you can control the size and number of
* audio buffers (if needed), the sample rate, and
* the audio source.
*=====================================================*/

#ifndef _dsp_ap_h_
#define _dsp_ap_h_

#include "math.h"
#include "aic23.h"
#include "dsk_registers.h"
```

```
/* DSP_SOURCE
* -----------
* The following lines control whether Line_In or Mic_In is
* the source of the audio samples to the DSP board.  Use Mic_In
* if you want to use the headset, or Line_In if you want to use
* the PC to generate signals.  Just uncomment one of the lines
* below.
*/
//#define  DSP_SOURCE   AIC23_REG4_LINEIN

#define  DSP_SOURCE   AIC23_REG4_MIC

/* DSP_SAMPLE_RATE
* ----------------
* The following lines control the sample rate of the DSP board.
* Just uncomment one of the lines below to get sample rates from
* 8000 Hz up to 96kHz.
*/
#define DSP_SAMPLE_RATE     AIC23_REG8_8KHZ

//#define DSP_SAMPLE_RATE     AIC23_REG8_32KHZ

//#define DSP_SAMPLE_RATE     AIC23_REG8_48KHZ

//#define DSP_SAMPLE_RATE     AIC23_REG8_96KHZ

/*********************************************************/
/* You can probably leave the stuff below this line alone. */
/*********************************************************/

/* Number of samples in hardware buffers.  Must be a multiple of 32. */
#define BUFFER_SAMPLES 128

/* Number of buffers to allocate.  Need at least 2. */
#define NUM_BUFFERS     2

/* Scale used for FP<->Int conversions */
#define SCALE           16384

int dsp_init();
void dsp_process(const float inL[],const float inR[],float outL[], float outR[])
```

```
#endif /* _dsp_ap_h_ */
```

delay.c

```
/*
 * delay.c
 *
 *  Created on: Apr 24, 2017
 *      Author: DSP_Lab
 */
/*****************************************************************************
 * delay.c
 *      Implements functions from delay block.
 *****************************************************************************/

#include <std.h>
#include <sys.h>
#include <dev.h>
#include <sio.h>

#include "delay.h"
#include "dsp_ap.h"

/*---------------------------------------------------------------------------
 * delay_init()
 *      This function initializes a delay block with a delay of 0.
 * Inputs:
 *      None.
 * Returns:
 *      0       An error ocurred
 *      other   A pointer to a new delay structure
 *---------------------------------------------------------------------------*/
delay_state_def *delay_init()
{
    delay_state_def *s;

    /* Allocate a new delay_state_def structure.  Holds state and parameters. */
    if ((s = (delay_state_def *)MEM_calloc(DELAY_SEG_ID, sizeof(delay_state_def)
    DELAY_BUFFER_ALIGN)) == NULL)
    {
        SYS_error("Unable to create an input delay
        floating-point buffer.", SYS_EUSER, 0);
        return(0);
```

```
    }

    /* Set initial delay to 0 */
    s->t = 0;
    s->h = 0;

    /* Success.  Return a pointer to the new state structure. */
    return(s);
}

/*-------------------------------------------------------------------------
 * delay_modify()
 *      Change operating parameters of the delay block.
 * Inputs:
 *      s          A pointer to the delay state structure
 *      new_delay   The new delay value
 *-------------------------------------------------------------------------*/
void delay_modify(delay_state_def *s, unsigned int new_delay)
{
    /* Check the requested delay */
    if (DELAY_BUFFER_SIZE < (new_delay + BUFFER_SAMPLES))
    {
        /* Make delay maximum */
        new_delay = DELAY_BUFFER_SIZE-BUFFER_SAMPLES;
    }

    /* Change the head of the buffer to obtain the requested delay.
     Do circular. */
    s->t =( s->t +1+new_delay)&DELAY_BUFFER_CMASK ;
}

/*-------------------------------------------------------------------------
 * delay()
 *      Process one buffer of samples with the delay block.
 *-------------------------------------------------------------------------*/
void delay(delay_state_def *s, const float x_in[], float y_out[])
{
    int i;

    /* Read all input samples into tail of buffer */
    for (i=0; i<BUFFER_SAMPLES; i++)
```

```
    {
        s->buffer[s->t] = x_in[i];
        s->t++; s->t &= DELAY_BUFFER_CMASK;
    }

    /* Read all output samples from head of buffer */
    for (i=0; i<BUFFER_SAMPLES; i++)
    {
        y_out[i]=s->buffer[s->h];
        s->h++; s->h &= DELAY_BUFFER_CMASK;
    }
}
```

delay.h

```
/*
 * delay.h
 *
 *  Created on: Apr 24, 2017
 *      Author: DSP_Lab
 */


#ifndef _delay_h_
#define _delay_h_

/*-- Defines ------------------------------------------------------------*/

/* Size of buffer (samples).  Controls maximum delay. */
#define DELAY_BUFFER_SIZE       16384

/* Mask.  Used to implment circular buffer */
#define DELAY_BUFFER_CMASK      (DELAY_BUFFER_SIZE-1)

/* Which memory segment the data should get stored in */
//#define DELAY_SEG_ID    0  /* IDRAM - fastest, but smallest */
#define DELAY_SEG_ID      1  /* SRAM  - a bit slower, but bigger */

/* Allows alignment of buffer on specified boundary. */
#define DELAY_BUFFER_ALIGN            128

/* Samples required for 1MS of Delay */
#define DSP_SAMPLES_PER_SEC 8000
#define DELAY_SAMPLES_1MS     (DSP_SAMPLES_PER_SEC/1000)

/*-- Structures ---------------------------------------------------------*/
typedef struct
{
    float buffer[DELAY_BUFFER_SIZE];
    unsigned int del;
    unsigned int h, t;
} delay_state_def;

/*-- Function Prototypes ------------------------------------------------*/
```

```
/* Initializes the delay block */
delay_state_def *delay_init();

/* Change delay parameters */
void delay_modify(delay_state_def *s, unsigned int new_delay);

/* Processes a buffer of samples for the delay block */
void delay(delay_state_def *s, const float x_in[], float y_out[]);

#endif /* _delay_h_ */
```

dsp_ap.c

```
/************************************************************
* dsp_ap.c
*  You should edit this file to contain your DSP code
* and calls to functions in other files.
************************************************************/

#include "dsp_ap.h"
#include "delay.h"
/* Make 2 delay blocks for left/right channel */
delay_state_def *delay_left;
delay_state_def *delay_right;
/*
 * State of DIP switches.   Set initial value to force
 * update of delay state.
 */
unsigned int switch_state = 0xff;
/* Global Declarations.  Add as needed. */
float mybuffer[BUFFER_SAMPLES];

/*-----------------------------------------------------------
* dsp_init
* This function will be called when the board first starts.
* In here you should allocate buffers or global things
*   you will need later during actual processing.
* Inputs:
* None
* Outputs:
* 0 Success
* 1 Error
*-----------------------------------------------------------*/
int dsp_init()
{
    /* Initialize the delay block */
    if ((delay_left = delay_init()) == 0)
    {
        /* Error */
        return(1);
    }
```

```
    /* Initialize the delay block */
    if ((delay_right = delay_init()) == 0)
    {
        /* Error */
        return(1);
    }

    /* Success */
    return(0);
}



/*-------------------------------------------------------
* dsp_process
* This function is what actually processes input samples
* and generates output samples.  The samples are passed
* in using the arrays inL and inR, corresponding to the
* left and right channels, respectively.  You
* can read these and then write to the arrays outL
* and outR.  After processing the arrays, you should exit.
* Inputs:
* inL Array of left input samples.  Indices on this
* and the other arrays go from 0 to BUFFER_SAMPLES.
*
* Outputs:
* 0 Success
* 1 Error
*-------------------------------------------------------*/
void dsp_process(
const float inL[],
const float inR[],
float outL[],
float outR[])
{
    unsigned int switch_state_new;
    unsigned int delay_mult;

    /*
     * Check if the state of the DIP switches changed.  DIP switches are upper
     * 4 bits of USER_REG.  We use the 3 least sig. bits to indicate delay in
```

```
     * powers of 2.
     */
    switch_state_new = (USER_REG >> 4) & 0x7;
    if (switch_state_new != switch_state)
    {
        /* State of switches changed.  Update delay block. */
        switch_state = switch_state_new;

        /*
         * Compute new delay according to switch state
         *   Do in powers of 2 according to lower 3 DIP switches.  Allows us to
         *   try a wide range of delays.
         *    000 = 0 => 1
         *    001 = 1 => 2
         *    010 = 2 => 4
         *    011 = 3 => 8 ...
         *    111 = 7 => 128
         */

        delay_mult = 1 << switch_state;

        /* Update delay blocks */
        delay_modify(delay_left,  10*DELAY_SAMPLES_1MS*delay_mult);
        delay_modify(delay_right, 10*DELAY_SAMPLES_1MS*delay_mult);
    }

    /* Run the samples through the delay block. */
    delay(delay_left, inL, outL);
    delay(delay_right, inR, outR);
}
```

Question 4

Any problems you ran into and how you solved them.

Writing and code in C was initially difficult but after using the MATLAB code from last semester I was able to see the similarities between the codes and so it was easy to progress.It was also difficult find the switches but with the help of the professor I was able to

find them.Initially the the task looked complicated but as I began consulting with the TAs the difficulty began to decrease

Conclusion

In conclusion we saw that programming a real-time DSP blocks in C is very similar to that of implementing it in MATLAB in terms of the coding aspect.

Reference

"Delay & FIR On DSP Board (DSK6713) | DSP And Communication Lab". Dsp-fhu.user.jacobs-university.de. N.p., 2017. Web. 2 May 2017.