JACOBS UNIVERSITY BREMEN

DIGITAL SIGNAL PROCESSING LAB

SPRING SEMESter 2020

FIR Implementation on DSP Board

Instructor :
Ph.D. Fangning Hu

Author :
Haseeb Ahmed

JACOBS UNIVERSITY

1 Introducion [1]

1.1 Audio Processing on the TI DSP Boards

Real-time processing (non-stop, and no samples missed) in C is implemented with the digital signal processor and the Audio Codec as shown below in figure (1).

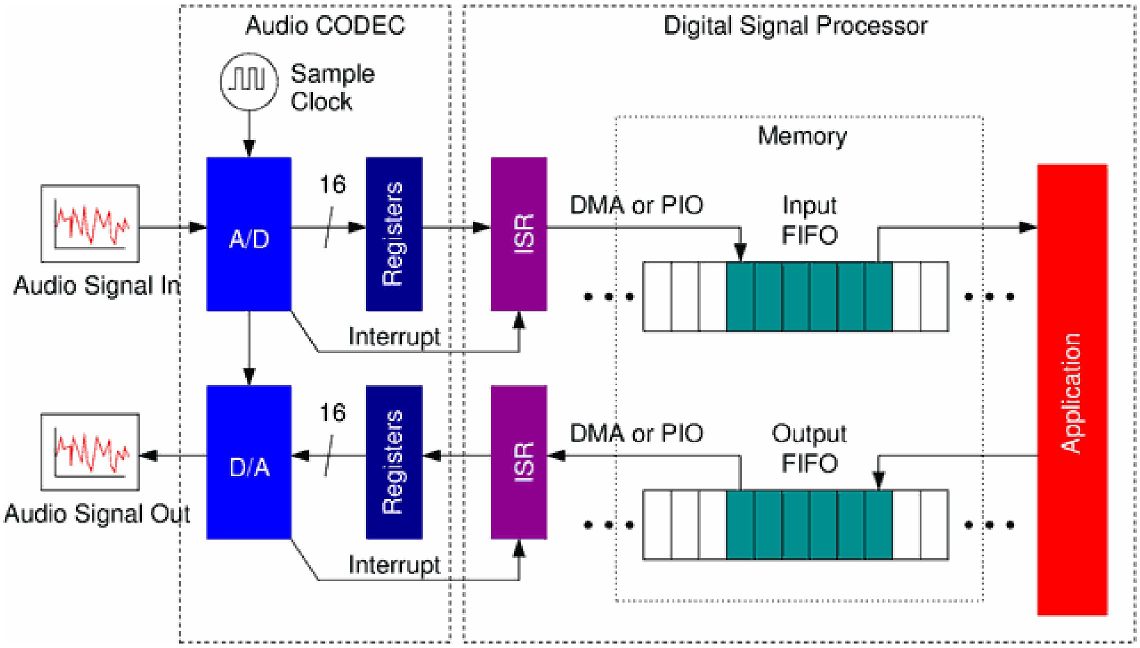

FIGURE 1 – Audio Process Diagram : Input and output process The

main functions of the Audio Codec block include :

—Sampling of the audio signal by an A/D converter.

—Converting analog to digital signals and converting voltage samples into 16- bit signals integers and storing them in registers.

The main functions of the DSP include :

—storing samples temporarily in the memory in a first-in first-out (FIFO) buf- fer and

—processing the data using various algorithms and applications.

1. Most of the processing algorithm of the FIR filter has been discussed in the previous (Matlab) report including the buffer wrapping with bit masking, in C. Please have a look at that report to check at what I did in the Execution parts.

2 Execution and Evaluation

2.1 Problem 1

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

The Golden project copies the input signal to form an output signal. We add a header file and a source file to the project. Then by changing $dsp\_ap.h$ and $dsp\_ap.c$ files, we modify the $dsp\_ap.out$. In the $dsp\_ap.h$, we can choose if the input signal is coming from the microphone or from a computer program(line in). Also, the samping rate can be changed from this file. Properly initialising the block is done by making changes to the $dsp\_ap.c$ and $dsp\_ap.h$ files. Each new buffer of samples is done in the $dsp\_process()$ function, so adjusting it ensures proper processing of the input and so the output will function properly.

2.2 Problem 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 ?

In Real-time processing in C, blocks are implemented using a header file and a source file for each block. The first is $[block\_name].h$ and $[block\_name].c$

The two main functions are $[block\_name]\_init()$ for initializing and $[block\_name]()$ for implementing.

State is stored in a structure and referenced using a pointer.

2.3 Problem 3

• A printout of your code for the FIR filter (you don't need to print out the filter coefficients !)

```
/*=====================================================
*  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//Input is from Computer
//#define   DSP_SOURCE        AIC23_REG4_MIC //Input is from 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(constfloat inL[], constfloat inR[], float outL[],                    float
 ↔    outR[]);


#endif/* _dsp_ap_h_ */
```

```
/*************************************************************
*  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"fir.h"
#include"rc1_taps.h"
```

```

/* Global Declarations. Add as needed. */

/* Make 2 delay blocks for left/right channel */
fir_state_def*fir_left;
fir_state_def*fir_right;

float mybuffer[BUFFER_SAMPLES];
/*
  * State of DIP switches.              Set initial value to force
  * update of delay state.
  */
unsignedint   switch_state=0xff;
/*---------------------------------------------------------
* 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 ((fir_left=fir_init(rc1_taps_LEN, rc1_taps))==0)
       {
              /* Error */
              return(1);
       }

       /* Initialize the delay block */
       if ((fir_right=fir_init(rc1_taps_LEN, rc1_taps))==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(
    constfloat inL[],
    constfloat 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. */ fir(fir_left, inL, outL);
        fir(fir_right, inR, outR);
}
```

```
/*************************************************************************
  * fir.h
```

```
 *          Header defines for implementing an FIR block.
 ***************************************************************************/

#ifndef _fir_h_
#define _fir_h_

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

/* Size of buffer (samples). Maximum filter length. */
#define FIR_BUFFER_SIZE          512

/* Mask. Used to implment circular buffer */
#define FIR_BUFFER_CMASK         (FIR_BUFFER_SIZE-1)

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

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

/*-- Structures -------------------------------------------------------------*/
typedefstruct
{
      float buffer[FIR_BUFFER_SIZE];
      float len;
      float *h;
      unsignedint   t;
} fir_state_def;

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

/* Initializes the fir block */
fir_state_def*fir_init( int len, float *h);

/* Processes a buffer of samples for the fir block */
void fir(fir_state_def*s, constfloat x_in[], float y_out[]);

#endif/* _fir_h_ */
```

```
/*
 *  fir.c
 *
 *    Created on: Apr 25, 2017
 *          Author: DSP_Lab
 */
/* Initializes the fir block */
#include<std.h>
#include<sys.h>
```

```
#include<dev.h>
#include<sio.h>


#include"fir.h"
#include"dsp_ap.h"
#include"rc1_taps.h"


fir_state_def*fir_init( int len, float *h){

        fir_state_def*s;

        /* Allocate a new delay_state_def structure. Holds state and parameters. */
        if ((s=(fir_state_def*)MEM_calloc(FIR_SEG_ID,                  sizeof(fir_state_def),
              FIR_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-
        >len=len;
        //****************************************
        //Circular Buffer Implementation s-
        >t=0;
        s->h=h;
    //******************************************
        /* Success. Return a pointer to the new state structure. */
        return(s);


}


/* Processes a buffer of samples for the fir block */
void fir(fir_state_def*s, constfloat x_in[], float y_out[]){

    int i,j;
    float sum;
    unsignedint   ptr;
    /* Read all input samples into tail of buffer */
    for(i=0; i<BUFFER_SAMPLES; i++)
    {
        //****************************************
            //Processing:
        s->buffer[s->t]=x_in[i];//Store a sample
        s->t++;//Increment tail index(circular)
        s->t&=FIR_BUFFER_CMASK;//bit and with the buffer mask
        ptr=(s->t+FIR_BUFFER_CMASK)&(FIR_BUFFER_CMASK);
        sum=0.0;//Assign a 0 float value for later incrementing
        for(j=0; j<s->len; j++)
        {
            sum=sum+s->buffer[ptr]*s->h[j];
            ptr=(ptr+FIR_BUFFER_CMASK)&FIR_BUFFER_CMASK;
```

```
        }

        y_out[i]=sum;//Filter samples are moved into output
        //*****************************************
    }
}
```

2.4 Problem4

• Any problems you ran into and how you solved them.

Writing the code for the FIR implementation was only a bit challenging, as we had the matlab implementation from the previous lab report. Setting up the macros in the fir.h was a mistake that was left unnoticed. These were the fir buffer size, the mask. I had to go through every header and source file to notice that I used the wrong syntax to assign the variables.

3 Conclusion

In DSP board implementation, the FIR filter removes frequencies above a cer- tain frequency, in our case around $2kHz$. In the lab, we used signal generator to test our implementation and run our code using CSS code editor. Above $2Khz$, we can not hear anything as the sound signal has been filtered out.

4 References

[1]http://dsp-fhu.user.jacobs-university.de/?page_id=147

[2]http://www.cengage.com/resource_uploads/downloads/0495082511_315335.pdf