JACOBS UNIVERSITY

LAB 1: DELAY ON DSP BOARD DSK6713

Jacobs University Bremen

CO27-300231 DSP & Communications Lab

Spring Semester 2020

Prof. Fangning Hu

Kelan Garcia

April 27, 2020

Mailbox Number: XC-316

Contents

0.1 Introduction

Objective

The objective of this lab is to learn how to program a real-time dsp delay block in C. Following all dsp coding and oriented programming practices.

Background

- Structured Programming of DSP Blocks in C

  An individual DSP block can be coded by writing the following functions:

  1. blockname_init(): A function that initializes the block and returns a new state structure for the block.
  2. blockname(): A function that processes buffers of input samples to generate buffers of output samples.
  3. blockname_modify(): An optional function that allows the operation of the block to be modified at runtime.

- File Organization For each block, you write it in two files:

  1. blockname.h: A "header" file that contains global definitions for the block. This file can then be "included" in other C files that need to use the block.
  2. blockname.c: The actual code that implements the block.

0.2 Tasks

Task 1: Try to understand how two programs in the gold package: dsp_top.c and dsp_ap.c interact with each other.

In dsp_top.c file the function void init() will called the function dsp_init() that is in the dsp_ap.c file. the function dsp_init() will return 0 if the file was initialized correctly and it will return 1 if an error was ocurred. Then if dsp_top.c recive 1 then the if statement will called an error.

Later in the dsp_top.c file in the function void io() we check if the switch 3 is up and if not then we call the dsp_process() function that is in dsp_ap.c file.

```
//                              dsp_top.c
void init(){
    .
    .
    if (dsp_init()){ //checking if it was initialized correctly
    flash_error(200);//error if dsp_init return 0
    }
    .
    .
}
void io(){
    .
    .
    /* Check for loop-back mode */
    if ((USER_REG >> 4) & (0x8)){ //if switch 3 up is true then:
        /* Copy input to output. */
        for (i=0; i<BUFFER_SAMPLES; i++){
            outL[i] = inL[i];
            outR[i] = inR[i];
        }
    }
    else{//if switch 3 is down then dsp_process is called
        /* Call the user's process routine. */
        dsp_process(inL, inR, outL, outR);
    }
    .
    .
}
```

As it can be seen in the dsp_ap.c script below function dsp_init() will return 0 if it was well initialized. And dsp_process(nL , inR , outL , outR) will copy the input buffers (Left and Right buffers) of the channels into the output buffers.

```c
//                              dsp_ap.c

int dsp_init(){
    /* Add code here if needed. */
    return(0);
}

void dsp_process(const float inL[],const float inR[],float outL[],
    float outR[]){
    int i;
    /* EXAMPLE: Copy input to output. */
    for (i=0; i<BUFFER_SAMPLES; i++){
        outL[i] = inL[i];
        outR[i] = inR[i];
    }
}
```

Task 2: Try to find out what happens when we press Down or Up of Switch 3?

Table : CPLD USER_REG Register

Bit Name	R/W	Description
7 USER_SW3	R	User DIP Switch 3(1 = Off(UP), 0 = On(DOWN))
6 USER_SW2	R	User DIP Switch 2(1 = Off(UP), 0 = On(DOWN))
5 USER_SW1	R	User DIP Switch 1(1 = Off(UP), 0 = On(DOWN))
4 USER_SW0	R	User DIP Switch 0(1 = Off(UP), 0 = On(DOWN))
3 USER_LED3	R/W	User-defined LED 3 Control (0 = Off, 1 = On)
2 USER_LED2	R/W	User-defined LED 2 Control (0 = Off, 1 = On)
1 USER_LED1	R/W	User-defined LED 1 Control (0 = Off, 1 = On)
0 USER_LED0	R/W	User-defined LED 0 Control (0 = Off, 1 = On)

By looking at the table on top, it can be seen that switch 3 is the 7th bit. This tells us the position of the bit that describes the state of the switch. If the bit in position 7 is 1 then the switch is Off (up), and if it is 0 then the switch is On (down). Switch 3 in binary:

$$\text{Off(Up)} : 10000000$$
$$\text{On(On)} : 00000000$$

In the file dsp_top.c is an if statement checking the value of the bit by using an and bitwise operator. If the switch is up then it copies the input to output, but if the switch is down then it calls the dsp_process() function that is in the dsp_ap.c file.

Task 3: Review the design of the circular buffer in matlab if you forget it:

I reviewed the topic from all the following sources:

Delay and FIR Lab Manual from Professor's Web Page[3]
Communications Lab, Delay and FIR Lab Report by Kelan Garcia[2][1]
Simulate Delay Lab Manual from Professor's Slide[4]

For the exact description of a circular buffer check my lab report fromm Communications Basics Lab (second link) on page 2-3

Task 4: Read the chapters:

1.) Structured Programming of DSP Blocks in C
2.) File Organization
3.) Delay Block Example
4.) Function definitions in delay.c

and at the end, fill in the code in delay_modify() and delay()

I read, understood and complete both of the following codes:

```
/*                            delay.h
            Header defines for implementing a delay block.
                                                                    */

#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_ */
```

```
/*                             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_state_def *delay_init(){

    /*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 *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 + 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;

```

```c
    /* 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 + 1) &  DELAY_BUFFER_CMASK;
   }
}
```

Task 5: Explain each part of dsp_ap.c

```c
//                           dsp_ap.c

#include "dsp_ap.h" //Including header of this file
#include "delay.h"  //Including the header of the delay script

//                      Global Declarations

// Pointers to global struct variables
delay_state_def *delay_left;
delay_state_def *delay_right;

//State of DIP switches.
unsigned int switch_state = 0xff; //Set initial value to force update of delay state.

float mybuffer[BUFFER_SAMPLES];

int dsp_init(){

    /*-------------------------------------------------------------
    *dsp_init
    * This function will be called when the board first starts.
    * Inputs:              Outputs:
    * None                 0 Success
    *                      1 Error
    *------------------------------------------------------------*/

    //              Initialize the left delay block
```

```
    if ((delay_left = delay_init()) == 0)
    {
        /* Error */
        return(1);
    }

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

    /* Success */
    return(0);
}

void dsp_process(
  const float inL[],
  const float inR[],
  float outL[],
  float outR[]){

    /*------------------------------------------------------------
    * dsp_process
    * This function is what actually processes input samples
    * and generates output samples.
    * Inputs:
    * inL,inR   Array of left and Right input samples.
    * outL,outR   Array of left and Right output samples.
    *
    * Outputs:
    * 0 Success
    * 1 Error
    *----------------------------------------------------------*/


  //               DECLARING NEEDED VARIABLES
  unsigned int switch_state_new;
  unsigned int delay_mult;

    /*Check if the state of the DIP switches changed.  DIP switches
```

```
70      *are upper 4 bits of USER_REG.  We use the 3 least sig. bits
71      *to indicate delay inpowers of 2.                            */
72 
73     switch_state_new = (USER_REG >> 4) & 0x7;
74 
75     if (switch_state_new != switch_state){
76         //State of switches changed.  Update delay block.
77         switch_state = switch_state_new;
78 
79         /*Compute new delay according to switch state
80         *   Do in powers of 2 according to lower 3 DIP switches.
81         *   Allows us to try a wide range of delays.
82         *posible switch_state, left shifts, delay_mult value
83         *          000        =      0      =>        1
84         *          001        =      1      =>        2
85         *          010        =      2      =>        4
86         *          011        =      3      =>        8 ...
87         *          111        =      7      =>      128 */
88 
89         delay_mult = 1 << switch_state;
90 
91         /* Update delay blocks */
92         delay_modify(delay_left,  10*DELAY_SAMPLES_1MS*delay_mult);
93         delay_modify(delay_right, 10*DELAY_SAMPLES_1MS*delay_mult);
94     }
95       /* Run the samples through the delay block. */
96   delay(delay_left, inL, outL);
97   delay(delay_right, inR, outR);
98 }
```

In dsp_ap.c

First we use the #include in order to include all the headers of the files that we need, that are: "dsp_ap.h" and "delay.h" this is done in code lines 3-4.

Then, we define the pointers to a delay state structure. Since, we need 2 structs (for left and right channel) then in this case we create 2 states structure pointers (codelines 9-10). Also, we need more global variables which are the state of the switches and the buffer which were declared in codelines 13 and 15.

Next the dsp_init() function is declared. This funcitons will be called by dsp_top.c when the dsp board starts. This funciton will create the left and right delay blocks and will check if where created correctly. If not it will return 1 and if yes it will return 0.

After that, the dsp_process() function is declared. This function is called by dsp_top.c every time switch 3 is down. This function takes inR, inL, as input parameters and outL and outR as outputs (pointers). This funcion checks the state of the switches 0-2 if they are different from the last state then we need to update the delay blocks by replaceing the last switch_state value with the new one and re-calculating the new delay_multi based on the binary value of switches 0-2 in order to update the delay blocks.

Finally, when the delay blocks are updated with the corresponding delay_mult the inputs inL and inR are delayed by the function delay() and stored at outL, outR.

Task 6: Calculate how much delay (ms) you get for different switches down?

This was calulated by left shifting 1 with the switch states decimal value and we will get the delay_mult value, then this value is multiply times the delay_samples_1ms and times ten, then the result of the milliseconds delay is the result.

$$\text{delay\_mult} = 1 \ll \text{switch\_state}$$
$$\text{ms} = \text{delay\_mult} * \text{delay\_samples\_1ms} * 10$$

Switches States	Left shifts to 1	delay_mult value	milliseconds
000	0	1	10 ms
001	1	2	20 ms
010	2	4	40 ms
011	3	8	80 ms
100	4	16	160 ms
101	5	32	320 ms
110	6	64	640 ms
111	7	128	1280 ms

Table 1: Shows how much will be the delay in ms depending on the switches states

0.3 Conclusion

In conclusion programming in C a dsp block is really similar to do it in Matlab. The difference is that the code runs faster in C but is more complicated to program it than in Maltab, but after this lab we saw that in reality is not that dificult to code it, is basiclly the same structure than the Matlab code and also every time we call a function in C we do it in a more efficient way than in matlab because we use pointers to the struct instead of passing the state to every function. I didn't met any problems in the lab.

Bibliography

[1] Kelan Garcia. Lab Report 3: Delay Matlab, 2019.

[2] Fangning Hu. Protected: Delay and FIR (Matlab Part), 2015.

[3] Fangning Hu. Protected: Delay and FIR on DSP Board (DSK6713), 2015.

[4] Fangning Hu. DSPlab Assistant Material, 2020.