Real-Time Block Implementation on DSP board

Digital Signal Processing Lab

3/6/2013

Jacobs University Bremen
Joshan Chaudhary and Grishma Raj Pandeya

Lab Write Up

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

The golden project was used in the lab after some modification for real time audio streaming. The files dsp_ap.h and dsp_ap.c needs to be modified.

In the header file dsp_ap.h the following need to be modified.

- DSP_SOURCE which sets whether the input samples come from the line in or the microphone input on the DSP board.
- DSP_SAMPLE_RATE which allows one to change the sample rate from 8 KHz to 96 KHz.

In dsp_ap.c we have to modify two functions dsp_init() and dsp_process().
We put additional code in dsp_init() to allocate or initialize variables before running the algorithm. The function is called once and returns a 0 to indicate success or nonzero to indicate error.
The other function is :

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

The input samples are provided in the floating point arrays inL and inR (for the left and right channels respectively) and each of these has BUFFER_SAMPLE elements, which is already defined. So we basically process one or both of these arrays to generate output samples, which are put in outL and outR. The const keyword is used to make sure that inL and inR cannot be changed.

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 blocks are implemented with an object oriented approach to the circular buffers method. There are header files that contain the global definitions, constants for the block and a source file that contains the actual code that implements that block. This organization of blocks makes it easier to test them individually and then later connect them to support real-time processing.
The two main functions are [blockname]_init and [blockname]. The first one initiliazes the block with initial values for the parameters of the block and returns a new state structure. The second function is the actual implementation of the block where the actual DSP operation is performed and output buffers are generated. For the delay block and FIR filter block implemented in the lab the following main functions were used: delay_init() initializes the delay block with a delay of 0 and returns a new state structure for the block. delay() processes a buffer of input samples with the delay block and generates an output buffer. fir_int() initializes the FIR filter block and returns a new state structure for the block. fir() processes the buffer of input samples with the FIR filter block and generates the output buffer.

A state is stored in struct in C. We used the following struct in lab to store a state.

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

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

Before the checkout we tried to debug the code and tried something more but it did not work. So, before the check out we just pressed the undo button several times to go back to the state when it partially worked and I really hope that we saved the code on the state when it was running.

```
/*****************************************************************************
 * 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, but bigger */

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

/*-- Structures ---------------------------------------------------------*/
typedef struct
{
   float buffer[FIR_BUFFER_SIZE];
   float len;
   float *h;
```

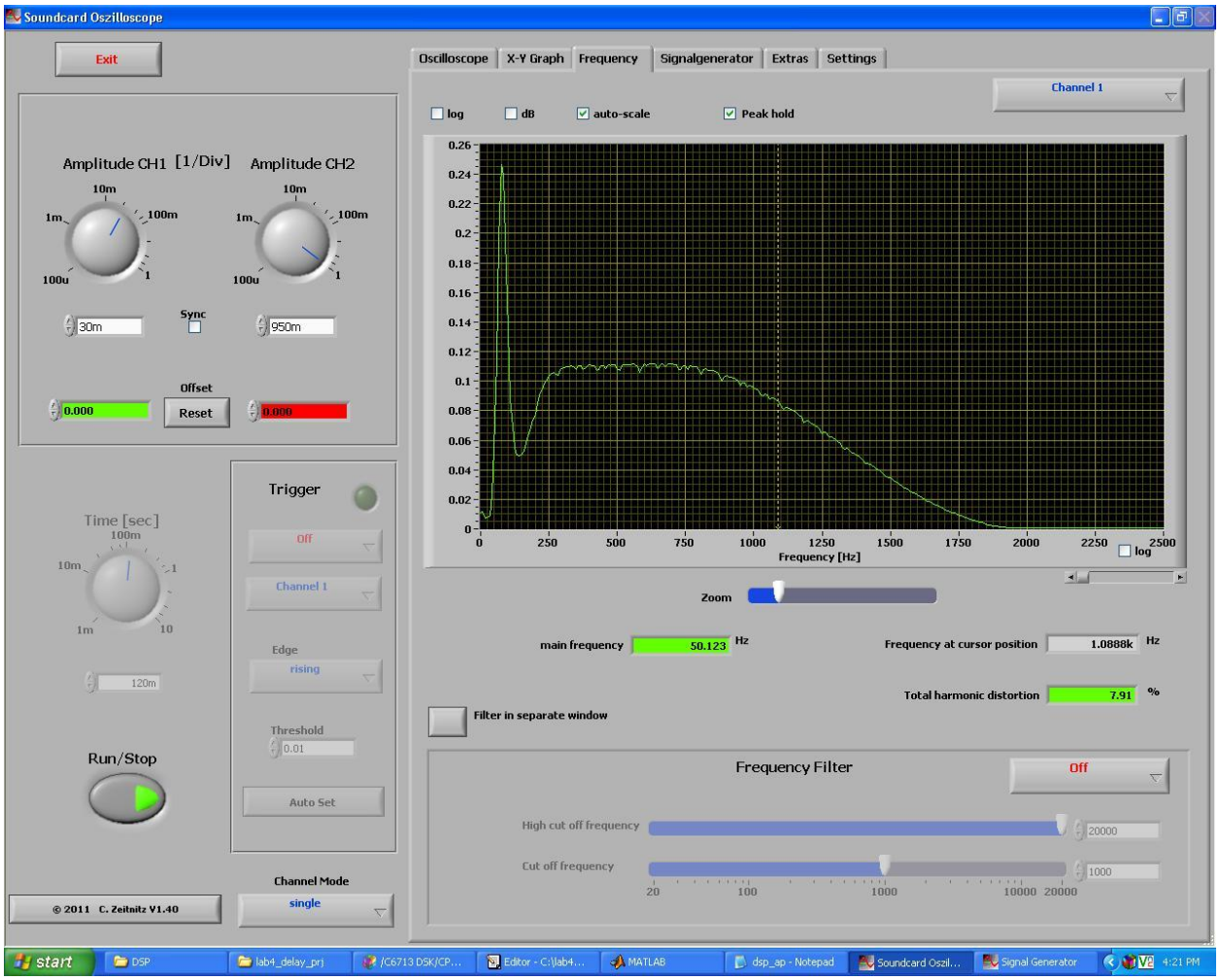
```
   unsigned int t;
} fir_state_def;

/*-- 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, const float x_in[], float y_out[]);

#endif /* _fir_h_ */



/*************************************************************************
 * fir.c
 *     Implements functions from fir block.
 *************************************************************************/

#include<std.h>
#include<sys.h>
#include<dev.h>
#include "fir.h"
#include "dsp_ap.h"


/*-------------------------------------------------------------------------
 * fir_init()
 *     This function initializes a fir block with a fir of 0.
 * Inputs:
 *     None.
 * Returns:
 *     0      An error ocurred
 *     other   A pointer to a new fir structure
 *-------------------------------------------------------------------------*/

fir_state_def *fir_init(int len, float *h)
{
          fir_state_def *s;
          if((s = (fir_state_def *)MEM_calloc(FIR_SEG_ID, sizeof(fir_state_def),
DELAY_BUFFER_ALIGN))=NULL) {
```

```
		SYS_error("unable to create an input delay floating-point buffer.", SYS_EUSER, 0);
	return(0);
	}

	s->t = 0;
	s->h = h;
	s->len = len;

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

void fir(fir_state_def *s, const float x_in[], float y_out[]) {
	int i, j;
	float sum;
	int ptr;

	for (i = 0; i<BUFFER_SIZE; i++) {
		s->buffer[s->t] = x_in[i];
		s->t++; s->t &= FIR_BUFFER_CMASK;
		ptr = s->t-1; ptr &= FIR_BUFFER_CMASK;
		sum =0.0;
		for (j = 0; j<rc1_taps_LEN; j++) {
			sum += s->buffer[ptr] *s->h[j];
			ptr = ptr + FIR_BUFFER_CMASK; ptr &= DELAY_BUFFER_CMASK;
		}
		y_out[i]=sum;
	}
}


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

#include "dsp_ap.h"

/* 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()
{

/* Add code here if needed. */


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[])
{
        int i;

        /* EXAMPLE: Copy input to output. */
        for (i=0; i<BUFFER_SAMPLES; i++)
        {
                outL[i] = inL[i];
                outR[i] = inR[i];
        }

}
```

4. A plot showing the measured response of your filter running on the DSP board compared to the specified response of the filter (basically the output of the test_filter.m script).



The filter did not actually run as it should have been because of some errors in the coding / board. Our output was quite not as expected. This all that we have got as image because the filter response plot on matlab was not even close to working. This however, looks at least somewhat convincing than the one from matlab.

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

During the first time it was difficult remembering the whole sequence of loading the project, building it, loading it to the board and implementing the code. Later we got used to it and also got more experienced.

Then the most difficult part was implementing the filter in C. Realizing the matlab code from the last lab in C was a bit confusing because of different platform and different coding styles. Even though the code looked fine logically, we were not able to get the required output. We also saw some overshoot in the output. In summary, the experiment was quite difficult and even after hours of debugging we were not successful in getting the desired output. The TAs helped us with operating the scope and the result of the scope was quite okay.