CALIFORNIA STATE UNIVERSITY
LONG BEACH

Nov 2019
EE 489

Lab # 3

Digital Filter Design

prepared by
Brian Fox
Arya Daroui
Inessa Lopez

1. Introduction

The purpose of this lab is to gain firsthand experience designing low pass FIR and IIR filters in MATLABs fdatool, and implementing them on C55x hardware using a combination of C and C55x assembly in Code Composer Studio (CCS) using fixed-point arithmetic. To accomplish this, much of the code is provided by the texbook "Real Time Digital Signal Processing: Implementation and Application, 3rd Ed" by Sen M. Kuo, Bob H. Lee, and Wenshun Tian. IIR filters are commonly used due to fast computation but can only handle a limited amount of cycles. Although FIR filters can take up more memory, they are always stable, can be linear phase, and have no limit on cycles.

2. Method

2.1 FIR

We started by importing Experiment 3.2, the assembly implementation of an FIR filter, from the textbook. As received, this had its own input, which was a short `*.wav` file that sounded like a dual-frequency tone, and its own low pass filter coefficients. After running this, the output created clearly sounded like the higher frequency components were removed. Next, we created our own filter coefficients in MATLABs FDAtool. Following the lab prompt, our filter was created to be an equiripple low pass filter with a pass band of 800 Hz and pass band attenuation of 3 dB, stop band frequency of 1600 Hz and stop band attenuation of 40 dB, and a sample rate of 8 kHz. We set the arithmetic to fixed-point, 16-bit word length, and exported the coefficients to an ASCII coefficient file in decimal. The reason decimal was chosen was because the given code was written with decimal numbers for filter coefficients, scaling up by 32767.0 (or $2^{15} - 1$ since we are using 15 fractional bits and one sign bit for a range of zero to 32,767. We put our coefficients into the `blockFirCoef.h` file, replacing the default ones given by the textbook, and changed the constant `NUM_TAPS` to 11, the length of our FDAtool-generated filter. This also resulted in the audio file to be output with the higher frequency attenuated. Next, we went back to MATLAB to generate the three given inputs, which were a 101 point impulse, a 101 point sine wave with an 800 Hz frequency (exactly at the edge of the pass band), and finally another 101 point sine wave with 1600 Hz frequency, which is the edge of the stop band. These were all generated with a sample frequency of 8 kHz to match the sample rate of the generated filter, and are shown in Figure 1 Since the length of the data is only 101 points at a sample rate of 8kHz, the input, and subsequent output since both our FIR and IIR filters are Linear Time Invariant (LTI) systems, would be only 101/8000, or 0.012625, seconds. This approximately 1/80th of a second signal becomes impossible to use playing as an audio file as a method of testing our filters, so from here on graphs will be used to show both inputs and outputs. Since the same input data is used for both filters and the two are being compared, the inputs as well as both outputs are shown overlaid in Figure 1 Since these input values are 101 points, the constant `NUM_DATA` was also changed to 101. Once the code was run and outputs collected, they were completely unscaled, with the raw values having magnitudes in the thousands. The `*.wav` files created were read in to MATLAB, which scaled them properly to what is shown in the figures below. This scaling factor was 2^{15}, but was computed automatically. The frequency response of the FIR system created by FDAtool is shown in Figure 5. The phase is linear, as it is an odd symmetric FIR filter, making it Type 1, and giving it a constant group delay- a feature that is very important for audio filters. This feature is not explored significantly in this experiment, since the non-impulse inputs were both single frequency and too short to listen to, but in theory there would be the same phase delay at any frequency. The response can be seen to have numerous zero crossings, little stability in either the pass band or the stop band, and a large transition band. While the value is approximately -3 dB at 800 Hz and -40 dB at 1600 Hz, any value more than a few dozen Hz above or below can give wildly different attenuations, as is common with FIR filters. Many frequencies above the cutoff of 1600 Hz even have higher magnitudes than the cutoff magnitude of -40 dB, including at 4000 Hz, the maximum frequency that can be represented without aliasing in an 8kHz sampled system like this. That said, the stop band doesnt go above -30 dB, so for some applications that ripple would be acceptable.

2.2 IIR

Much of what was done was the same for the IIR portion. For this, the same input files we created in matlab were used, and merely had to be added to the particular folder, since we imported it as a workspace. This time, the provided Experiment 4.2 was used, which was fixed-point IIR filtering. FDAtool was again used to create the filter using the same 8kHz sample rate, 800 Hz with 3 dB and 1600 Hz with 40 dB for the pass band and stop band, cutoff and attenuation, respectively. The Difference this time was that we selected an IIR Butterworth filter, instead of an FIR equiripple. This returned both feedforward and feedback terms, which were put into `fixPoint_directIIRTest.c`, replacing the provided example values from the textbook. This time no length of input data or number of taps constants required changing, and the code simply worked. The benefit of a Butterworth filter is its maximally flat pass band, something this experiment did not directly show with much detail. Having many frequencies in the pass band used as inputs would show the extremely consistent nature of the pass band of this filter, or at least one significantly lower than the cutoff frequency of 800 to show the lowering magnitude of the output as it nears the edge. For completeness, the frequency response is shown in Figure 6. As expected, the IIR filter has an incredibly flat pass band attenuation of 0 dB until just before the 800 Hz edge of our pass band. This characteristic would treat nearly any frequency below 800 Hz very similarly in magnitude. It would treat different frequencies differently in terms of the phase, however, since this IIR filter has a nonlinear phase, as nearly all IIR filters do, leading to a variable group delay at various frequencies. The magnitude does hit exactly -40 dB at 1600 Hz, but unlike the FIR filter, it continues to plunge, never rising above that value, and settling at -100 dB as it nears the nyquist rate of 4000 Hz.

2.3 C55x and MATLAB comparison

We compared the C55x board's outputs with outputs from MATLAB to see how accurate the board was. For the plots in Figure 2, Figure 3, and Figure 4, we took a simple absolute difference.

$$\text{Abs. difference} = d = |y_{\text{C55x}} - y_{\text{MATLAB}}|$$

From there, we took the average of the absolute difference to compare the different inputs.

$$\frac{1}{N}\sum_{n=0}^{N} d[n]$$

Where N is the number of samples n in the absolute difference d. The final results are shown in Table 1. We chose to use average absolute difference over average percent error because dividing by the extremely small amplitudes made the error seem magnitudes worse than is adequately represented by the avsolute difference in the plots.

3. Data

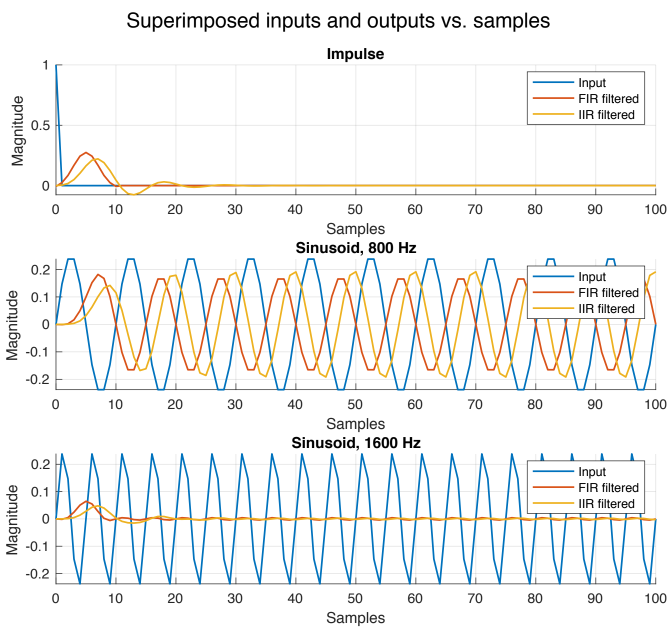

Figure 1. The C55x board's outputs for the different inputs for both FIR and IIR implementations

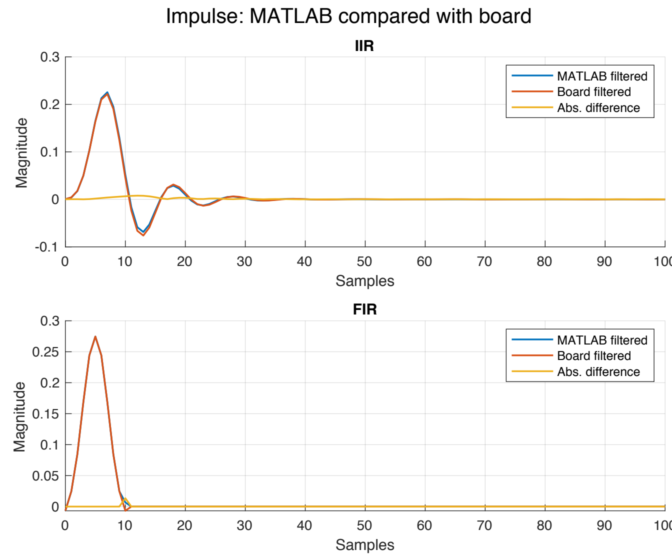

Figure 2. Comparison between the C55x filter implementation and MATLAB for the impulse response

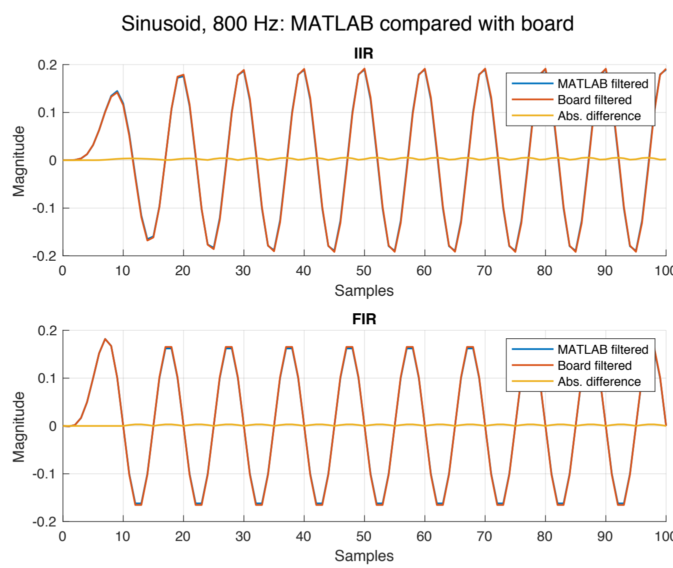

Figure 3. Comparison between the C55x filter implementation and MATLAB for the 800 Hz sinusoid

Table 1. Comparison of C55x and MATLAB filter implementations

	FIR abs. diff. (%)	IIR abs. diff (%)
Impulse	0.0135	0.0925
800 Hz	0.1845	0.2935
1600 Hz	0.1848	0.0268

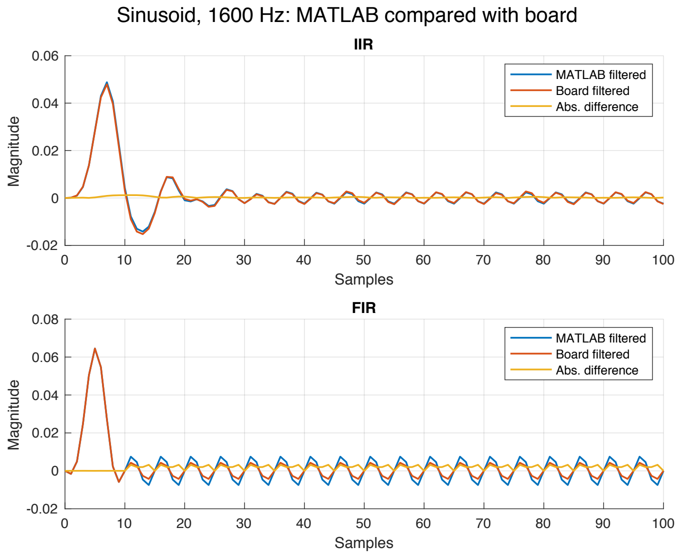

Figure 4. Comparison between the C55x filter implementation and MATLAB for the 1600 Hz sinusoid

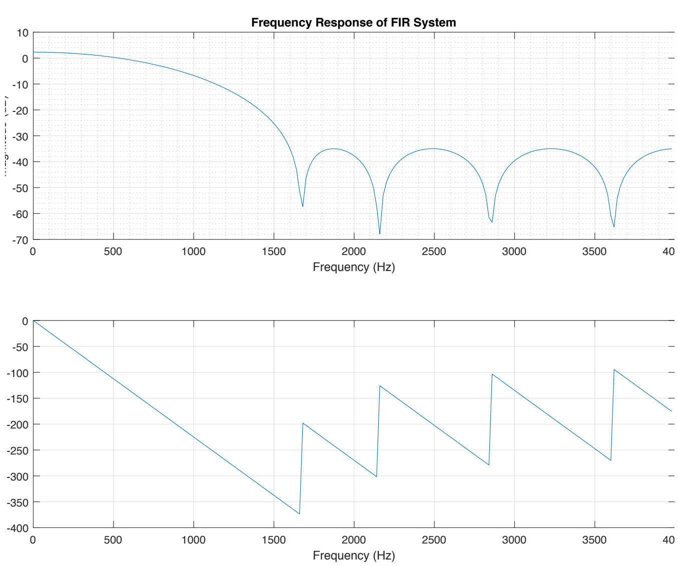

Figure 5. Frequency response of FIR system

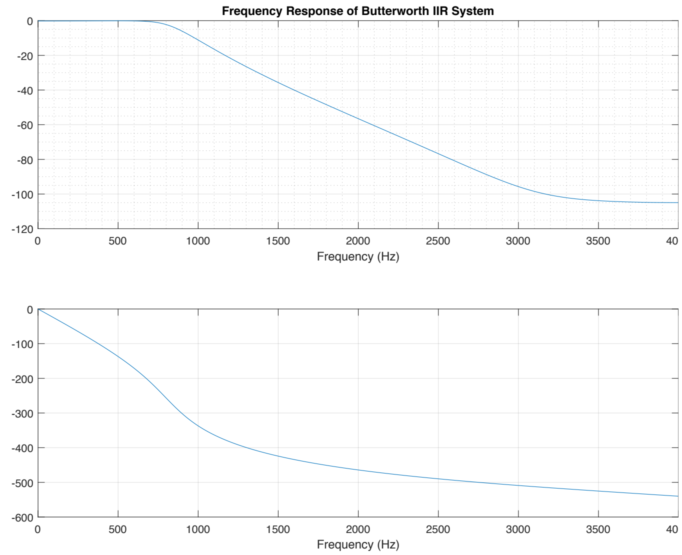

Figure 6. Frequency response of Butterworth IIR system

4. Discussion

4.1 FIR

The results of the FIR filter to the impulse response, or the FIR filters impulse response, is shown in Figure 2 This impulse response is finite and only lasts 11 samples. This is the length of the filter, which is exactly as expected since the eleven coefficients can be directly converted to eleven delayed impulses (from $n = 0$ to $n = 10$) in the $h(n)$ impulse response. After this, the impulse effects die away. It peaks around 0.2746 for both the MATLAB part and from the board.

The result of the FIR filter to the 800 Hz sinusoid is a scaled sinusoid of the same frequency, as expected for an LTI system. The 800 Hz frequency is at the edge of the pass band, and should be scaled down by the 3 dB pass band attenuation selected in the filter design portion. The output, at steady state, has a reduced amplitude of about 0.1652, down from the 0.25 amplitude of the input. This is expected since the 3 dB attenuation results in approximately

$$0.25 \times 10(-3/20) = 0.177$$

Samples happen to land on each side of the peak, which logically accounts for this discrepancy of approximately 6%. The transient has little impact, since its value is low compared to the steady state, but it does cause the first peak to be misshapen compared to the rest, with a peak of 0.1819, or 0.0167 higher than steady state. The result of the FIR filter to the 1600 Hz sinusoid is a scaled sinusoid of the same frequency, as expected for an LTI system. The 1600 Hz frequency is at the edge of the stop band, and should be scaled down by the 40 dB stop band attenuation selected in the filter design portion. The output, at steady state, has a reduced amplitude of about .004272, down from the 0.25 amplitude of the input. This is expected since the

40 dB attenuation results in approximately

$$0.25 \times 10(-40/20) = .0025$$

The transient value looks extremely high, but thats only in relation to the very small steady state amplitude. The transient mirrors the impulse response, peaking the same way and at the same time, because the impulse response is showing what happens when theres a sudden input applied from a previous zero input. It is actually much smaller than the impulse response, peaking at a value of around 0.06 vs the 0.27 of the impulse response. This difference in height is due to the impulse being the max value of $2^{15} - 1$, whereas the first value of the 1600 Hz sinusoid is actually zero, and the input ramps up slowly to never get close to the impulse value.

4.2 IIR

The results of the IIR filter we created to an impulse input, or its impulse response, continues infinitely, as the name would suggest. It starts off by ramping up quickly, but by the time more of the 7 feedback terms are able to take effect the response turns sharply back down from around sample 9 on. The feedback continues to get stronger, turning the impulse response back toward positive after the peak negative value at about sample 12. This negative feedback continues to envelop the oscillating signal, damping the amplitude further and further, never reaching zero. Figure 2 shows that the IIR impulse response peaks at 0.2217 for both the MATLAB and from the TI board. Much like the FIR filters response to the 800 Hz sinusoidal input, the IIRs response ends up being an 800 Hz sinusoid at steady-state, with an amplitude of approximately 3 dB lower than the input. This is because the IIR filter also has 3 dB pass band attenuation and an 800 Hz pass band cutoff frequency, leaving the input in said pass band, as well as the IIR filter being LTI as discussed earlier. The boards steady state amplitude is 0.1917, while the MATLABs is 0.1879. The expected is the same

$$0.25 \times 10(-3/20) = 0.177$$

from the FIR, or about 8.3% and 6.1% error from the board and MATLAB respectively. Transient is actually lower than the steady state, at 0.1419. Everything about the IIRs response to the 800 Hz sinusoidal input applies to the 1600 Hz input, except that it is at the stop band cutoff, with a stop band attenuation of 40 dB. The interesting thing about this result is that the initial spike before the die down to around an amplitude of 0.002747 looks very similar to the IIR impulse response, no doubt because it is the basic response of the system to an instantaneous change from no input to input. After this transient dies down, thanks to the feedback terms and all delays having actual data to read instead of padded zeros, it ends in a steady state response like a scaled down version of the input, as expected from LTI system theory. The expected value, as with the FIR, is

$$0.25 \times 10(-40/20) = .0025$$

at steady state, which is surprisingly close to the 0.002747 result achieved, for an error of 9.88% from the board. These numbers are 0.002398 and 4.1% respectively for the MATLAB.

4.3 C55x and MATLAB comparison

Due to the extremely small numbers in the results of this experiment, finding an accurate measurement of instantaneous error was a challenge. For example, using absolute error found far less error in the response of the FIR filter to the 1600 Hz sinusoid, even though it can be seen that the difference is among the highest, relative to height. This precluded the use of absolute error, as greater error relative to size is not accounted for, and great changes in size exist. Additionally, when using percent error with the MATLAB considered the actual value, the IIR Impulse response has the highest error at any point, and by a wide margin, with 81% error as the error on the final point of the data. This is because the correct value will be extremely close to 0, even relative to the other inputs. The board result is -6.1035e-05, which seem extremely close to zero, until it is compared to the MATLAB result of 4.5747e-09. This yielded an error of 81%, and was more than 40 times higher than any other percent error calculated by this method, and was not used for this reason. Aside from this section, these errors were extremely low.

5. Conclusion

After utilizing the FDATool block in MATLAB and the code provided from the textbook, we were able to see the different responses the FIR and IIR filters provided using fixed-point arithmetic. As shown previously, the FIR filter takes less samples to reach its steady state response as opposed to the IIR filter. This is due to the recursion of the IIR filter and its feedback loop. We were also able to see the accuracy of the board compared to the MATLAB results and concluded that the error between the two were very low.

6. Code

6.1 FIR

```
/*
* Experiment assembly implementation of block FIR filter - Chapter 3
* blockFirCoef.h
*
* Description: This is the filter coefficient file for assembly FIR filter
*
*  Created on: May 13, 2012
*      Author: BLEE
*
*       For the book "Real Time Digital Signal Processing:
*                    Fundamentals, Implementation and Application, 3rd Ed"
*                     By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
*                     Publisher: John Wiley and Sons, Ltd
*/

Int16 blockFirCoef[NUM_TAPS]={
(Int16)(-0.00677490234375*32767.0),(Int16)(0.024810791015625*32767.0),
(Int16)( 0.0844268798828125*32767.0),(Int16)(0.1685943603515625*32767.0),
(Int16)( 0.2441253662109375*32767.0),(Int16)(0.2745819091796875*32767.0),
(Int16)( 0.2441253662109375*32767.0),(Int16)(0.1685943603515625*32767.0),
(Int16)( 0.0844268798828125*32767.0),(Int16)(0.024810791015625*32767.0),
(Int16)(-0.00677490234375*32767.0)
};


/*
* Experiment assembly implementation of block FIR filter - Chapter 3
* blockFir.h
*
* Description: This is the header file for fixed-point FIR filter
*
*  Created on: May 13, 2012
*      Author: BLEE
*
*       For the book "Real Time Digital Signal Processing:
*                    Fundamentals, Implementation and Application, 3rd Ed"
*                     By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
*                     Publisher: John Wiley and Sons, Ltd
*/

#define  NUM_TAPS   11
#define  NUM_DATA   101

void blockFir(Int16 *x, Int16 blkSize,
                Int16 *h, Int16 order,
                Int16 *y,
                Int16 *w, Int16 *index);

;/*
; * Experiment assembly implementation of block FIR filter - Chapter 3
```

```
; * blockFir.asm
; *
; * Description: This is the assembly language implementation of block FIR filter
; *
; *  Created on: May 13, 2012
; *      Author: BLEE
; *
; *         For the book "Real Time Digital Signal Processing:
; *                        Fundamentals, Implementation and Application, 3rd Ed"
; *                        By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
; *                        Publisher: John Wiley and Sons, Ltd
; */

    .mmregs

    .sect   ".text:fir"
    .align 4

    .def    _blockFir

;----------------------------------------------------------------------
;   void blockFir(Int16 *x,            => AR0
;                 Int16 blkSize,       => T0
;                 Int16 *h,            => AR1
;                 Int16 order,         => T1
;                 Int16 *y,            => AR2
;                 Int16 *w,            => AR3
;                 Int16 *index)        => AR4
;----------------------------------------------------------------------

_blockFir:
    pshm  ST1_55              ; Save ST1, ST2, and ST3
    pshm  ST2_55
    pshm  ST3_55

    or    #0x340,mmap(ST1_55); Set FRCT,SXMD,SATD
    bset  SMUL                ; Set SMUL
    mov   mmap(AR1),BSA01     ; AR1=base address for coeff
    mov   mmap(T1),BK03       ; Set coefficient array size (order)
    mov   mmap(AR3),BSA23     ; AR3=base address for signal buffer
    or    #0xA,mmap(ST2_55)   ; AR1 & AR3 as circular pointers
    mov   #0,AR1              ; Coefficient start from h[0]
    mov   *AR4,AR3            ; Signal buffer start from w[index]
||  sub   #1,T0               ; T0=blkSize-1
    mov   T0,BRC0             ; Initialize outer loop to blkSize-1
    sub   #3,T1,T0            ; T0=order-3
    mov   T0,CSR              ; Initialize inner loop order-2 times
||  rptblocal sample_loop-1  ; Start the outer loop
    mov   *AR0+,*AR3          ; Put the new sample to signal buffer
    mpym  *AR3+,*AR1+,AC0     ; Do the 1st operation
||  rpt   CSR                 ; Start the inner loop
    macm  *AR3+,*AR1+,AC0
    macmr *AR3,*AR1+,AC0      ; Do the last operation with rounding
    mov   hi(AC0),*AR2+       ; Save Q15 filtered value
```

```
sample_loop

    popm  ST3_55             ; Restore ST1, ST2, and ST3
    popm  ST2_55
    popm  ST1_55
    mov   AR3,*AR4           ; Update signal buffer index
    ret

    .end

/*
* Experiment assembly implementation of block FIR filter - Chapter 3
* blockFirTest.c
*
* Description: This is the test file for the block FIR filter
*
*  Created on: May 13, 2012
*      Author: BLEE
*
*       For the book "Real Time Digital Signal Processing:
*                    Fundamentals, Implementation and Application, 3rd Ed"
*                     By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
*                     Publisher: John Wiley and Sons, Ltd
*/

#include <stdlib.h>
#include <stdio.h>
#include "tistdtypes.h"
#include "blockFir.h"

/* Define DSP system memory map */
#pragma DATA_SECTION(blockFirCoef, ".const:fir");
#pragma DATA_SECTION(w, ".bss:fir");

#include "blockFirCoef.h"


Int16 w[NUM_TAPS];

void main()
{
    FILE  *fpIn,*fpOut;
    Int16 i,k,c,
            index;        // Delay line index
    Int16 x[NUM_DATA],  // Input data
            y[NUM_DATA];  // Output data
    Int8  temp[NUM_DATA*2];
    Uint8 waveHeader[44];

    printf("Exp --- Assembly program_Block FIR filter experiment\n");
    printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
    scanf ("%d", &c);

    if (c == 2)
```

```
    {
        fpIn = fopen("..\\data\\impulse.wav", "rb");
        fpOut = fopen("..\\data\\FIR_imp_out.wav", "wb");
    }
    else
    {
        fpIn = fopen("..\\data\\input.pcm", "rb");
        fpOut = fopen("..\\data\\output.pcm", "wb");
    }

    if (fpIn == NULL)
    {
        printf("Can't open input file\n");
        exit(0);
    }

    if (c == 2)
    {
        fread(waveHeader, sizeof(Int8), 44, fpIn);
        fwrite(waveHeader, sizeof(Int8), 44, fpOut);
    }

    // Initialize for filtering process
    for (i=0; i<NUM_TAPS; i++)
    {
        w[i] = 0;
    }
    index = 0;


    // Begin filtering the data
    while (fread(temp, sizeof(Int8), NUM_DATA*2, fpIn) == (NUM_DATA*2))
    {
        for (k=0, i=0; i<NUM_DATA; i++)
        {
            x[i] = (temp[k]&0xFF)|(temp[k+1]<<8);
            k += 2;
        }
        // Filter the data x and save output y
        blockFir(x, NUM_DATA, blockFirCoef, NUM_TAPS, y, w, &index);

        for (k=0, i=0; i<NUM_DATA; i++)
        {
            temp[k++] = (y[i]&0xFF);
            temp[k++] = (y[i]>>8)&0xFF;
        }
        fwrite(temp, sizeof(Int8), NUM_DATA*2, fpOut);
    }

    fclose(fpIn);
    fclose(fpOut);

    printf("\nExp --- completed\n");

```

```
    printf("Exp --- Assembly program_Block FIR filter experiment\n");
    printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
    scanf ("%d", &c);

    if (c == 2)
    {
        fpIn = fopen("..\\data\\sin1.wav", "rb");
        fpOut = fopen("..\\data\\FIR_sin1_out.wav", "wb");
    }
    else
    {
        fpIn = fopen("..\\data\\input.pcm", "rb");
        fpOut = fopen("..\\data\\output.pcm", "wb");
    }

    if (fpIn == NULL)
    {
        printf("Can't open input file\n");
        exit(0);
    }

    if (c == 2)
    {
        fread(waveHeader, sizeof(Int8), 44, fpIn);
        fwrite(waveHeader, sizeof(Int8), 44, fpOut);
    }

    // Initialize for filtering process
    for (i=0; i<NUM_TAPS; i++)
    {
        w[i] = 0;
    }
    index = 0;


    // Begin filtering the data
    while (fread(temp, sizeof(Int8), NUM_DATA*2, fpIn) == (NUM_DATA*2))
    {
        for (k=0, i=0; i<NUM_DATA; i++)
        {
            x[i] = (temp[k]&0xFF)|(temp[k+1]<<8);
            k += 2;
        }
        // Filter the data x and save output y
        blockFir(x, NUM_DATA, blockFirCoef, NUM_TAPS, y, w, &index);

        for (k=0, i=0; i<NUM_DATA; i++)
        {
            temp[k++] = (y[i]&0xFF);
            temp[k++] = (y[i]>>8)&0xFF;
        }
        fwrite(temp, sizeof(Int8), NUM_DATA*2, fpOut);
    }

```

```
    fclose(fpIn);
    fclose(fpOut);

    printf("\nExp --- completed\n");

    printf("Exp --- Assembly program_Block FIR filter experiment\n");
    printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
    scanf ("%d", &c);

    if (c == 2)
    {
        fpIn = fopen("..\\data\\sin2.wav", "rb");
        fpOut = fopen("..\\data\\FIR_sin2_out.wav", "wb");
    }
    else
    {
        fpIn = fopen("..\\data\\input.pcm", "rb");
        fpOut = fopen("..\\data\\output.pcm", "wb");
    }

    if (fpIn == NULL)
    {
        printf("Can't open input file\n");
        exit(0);
    }

    if (c == 2)
    {
        fread(waveHeader, sizeof(Int8), 44, fpIn);
        fwrite(waveHeader, sizeof(Int8), 44, fpOut);
    }

    // Initialize for filtering process
    for (i=0; i<NUM_TAPS; i++)
    {
        w[i] = 0;
    }
    index = 0;


    // Begin filtering the data
    while (fread(temp, sizeof(Int8), NUM_DATA*2, fpIn) == (NUM_DATA*2))
    {
        for (k=0, i=0; i<NUM_DATA; i++)
        {
            x[i] = (temp[k]&0xFF)|(temp[k+1]<<8);
            k += 2;
        }
        // Filter the data x and save output y
        blockFir(x, NUM_DATA, blockFirCoef, NUM_TAPS, y, w, &index);

        for (k=0, i=0; i<NUM_DATA; i++)
        {
            temp[k++] = (y[i]&0xFF);
```

```
            temp[k++] = (y[i]>>8)&0xFF;
        }
        fwrite(temp, sizeof(Int8), NUM_DATA*2, fpOut);
    }

    fclose(fpIn);
    fclose(fpOut);

    printf("\nExp --- completed\n");
}
```

6.2 IIR

```
/*
* fixPointIIR.h
*
*  Created on: May 25, 2012
*      Author: BLEE
*
*  Description: This is the header file for the fixed-point IIR filter in direct form-I
*
*  For the book "Real Time Digital Signal Processing:
*               Fundamentals, Implementation and Application, 3rd Ed"
*               By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
*               Publisher: John Wiley and Sons, Ltd
*
*/

void fixPoint_IIR(Int16 in, Int16 *x, Int16 *y,
                    Int16 *b, Int16 nb, Int16 *a, Int16 na);

/*
* fixPoint_directIIRTest.c
*
*  Created on: May 25, 2012
*      Author: BLEE
*
*  Description: This is the test program for fixed-point direct form-I IIR filter
*
*  For the book "Real Time Digital Signal Processing:
*               Fundamentals, Implementation and Application, 3rd Ed"
*               By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
*               Publisher: John Wiley and Sons, Ltd
*
*/

#include <stdio.h>
#include <stdlib.h>
#include "tistdtypes.h"
#include "fixPointIIR.h"

// Coefficient length
#define NL  7
```

```
#define DL  7
#define Q11 2048    // For making Q11 format filter coefficients
#define RND 0.5

// Filter coefficients obtained from MATLAB script
/*
    Rp=0.1;                                      % Passband ripple
    Rs=60;                                       % Stopband attenuation
    [N,Wn]=ellipord(836/4000,1300/4000,Rp,Rs); % Filter order & scaling factor
    [b,a]=ellip(N,Rp,Rs,Wn);                     % Lowpass IIR filter
    [num,den]=iirlp2bp(b,a,0.5,[0.25, 0.75]);  % Bandpass IIR filter

Int16 num[NL] = =
(Int16)(0.0004*Q11+RND),
(Int16)(0.0024*Q11+RND),
(Int16)(0.0060*Q11+RND),
(Int16)(0.0081*Q11+RND),
(Int16)(0.0060*Q11+RND),
(Int16)(0.0024*Q11+RND),
(Int16)(0.0004*Q11+RND)
};

Int16 den[DL] = {
(Int16)(1.0000*Q11+RND),
(Int16)(-3.4943*Q11+RND),
(Int16)(5.4250*Q11+RND),
(Int16)(-4.6889*Q11+RND),
(Int16)(2.3579*Q11+RND),
(Int16)(-0.6499*Q11+RND),
(Int16)(0.0764*Q11+RND)
};

// Filter delay lines
Int16 x[NL],y[DL];

void main()
{

    Int16  in,i,c;
    FILE   *fpIn,*fpOut;
    Int8   temp[2];
    Uint8  waveHeader[44];
    Int16 inputIIR[101];
    Int16 outputIIR[101];
    int count = 0;
//
//    printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
//    scanf ("%d", &c);
    c = 2;

    if (c == 2)
    {
        fpIn = fopen("..\\data\\impulse.wav", "rb");
        fpOut = fopen("..\\data\\IIR_imp_out.wav", "wb");
```

```
    }
    else
    {
        fpIn = fopen("..\\data\\input.pcm", "rb");
        fpOut = fopen("..\\data\\output.pcm", "wb");
    }
    // Open file for read input data
    if (fpIn == NULL)
    {
        printf("Can't open input data file\n");
        exit(0);
    }

    if (c == 2)         // Create WAVE data file header
    {
        fread(waveHeader, sizeof(Int8), 44, fpIn);
        fwrite(waveHeader, sizeof(Int8), 44, fpOut);
    }

    // Clear delay lines
    for(i=0; i<NL; i++)
    {
        x[i] = 0;
    }
    for(i=0; i<DL; i++)
    {
        y[i] = 0;
    }

    printf("Exp --- IIR filter experiment\n");

    // Filter test
    while (fread(temp, sizeof(Int8), 2, fpIn) == 2)
    {
        in = (temp[0]&0xFF)|(temp[1]<<8);
        inputIIR[count] = in;

        // Filter the data
        fixPoint_IIR(in, x, y, num, NL, den, DL);
        outputIIR[count] = *y;
        temp[0] = (y[0]&0xFF);
        temp[1] = (y[0]>>8)&0xFF;
        fwrite(temp, sizeof(Int8), 2, fpOut);

        count++;
    }
    fclose(fpIn);
    fclose(fpOut);
    printf("Exp --- completed\n");


//    Int16  in,i,c;
//    FILE   *fpIn,*fpOut;
//    Int8   temp[2];
```

```
//    Uint8  waveHeader[44];
//    Int16 inputIIR[101];
//    Int16 outputIIR[101];
    count = 0;
//
//    printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
//    scanf ("%d", &c);
    c = 2;

    if (c == 2)
    {
        fpIn = fopen("..\\data\\sin1.wav", "rb");
        fpOut = fopen("..\\data\\IIR_sin1_out.wav", "wb");
    }
    else
    {
        fpIn = fopen("..\\data\\input.pcm", "rb");
        fpOut = fopen("..\\data\\output.pcm", "wb");
    }
    // Open file for read input data
    if (fpIn == NULL)
    {
        printf("Can't open input data file\n");
        exit(0);
    }

    if (c == 2)        // Create WAVE data file header
    {
        fread(waveHeader, sizeof(Int8), 44, fpIn);
        fwrite(waveHeader, sizeof(Int8), 44, fpOut);
    }

    // Clear delay lines
    for(i=0; i<NL; i++)
    {
        x[i] = 0;
    }
    for(i=0; i<DL; i++)
    {
        y[i] = 0;
    }

    printf("Exp --- IIR filter experiment\n");

    // Filter test
    while (fread(temp, sizeof(Int8), 2, fpIn) == 2)
    {
        in = (temp[0]&0xFF)|(temp[1]<<8);
        inputIIR[count] = in;

        // Filter the data
        fixPoint_IIR(in, x, y, num, NL, den, DL);
        outputIIR[count] = *y;
        temp[0] = (y[0]&0xFF);
```

```
        temp[1] = (y[0]>>8)&0xFF;
        fwrite(temp, sizeof(Int8), 2, fpOut);

        count++;
    }
    fclose(fpIn);
    fclose(fpOut);
    printf("Exp --- completed\n");


//    Int16  in,i,c;
//    FILE   *fpIn,*fpOut;
//    Int8   temp[2];
//    Uint8  waveHeader[44];
//    Int16 inputIIR[101];
//    Int16 outputIIR[101];
    count = 0;
//
//    printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
//    scanf ("%d", &c);
    c = 2;

    if (c == 2)
    {
        fpIn = fopen("..\\data\\sin2.wav", "rb");
        fpOut = fopen("..\\data\\IIR_sin2_out.wav", "wb");
    }
    else
    {
        fpIn = fopen("..\\data\\input.pcm", "rb");
        fpOut = fopen("..\\data\\output.pcm", "wb");
    }
    // Open file for read input data
    if (fpIn == NULL)
    {
        printf("Can't open input data file\n");
        exit(0);
    }

    if (c == 2)        // Create WAVE data file header
    {
        fread(waveHeader, sizeof(Int8), 44, fpIn);
        fwrite(waveHeader, sizeof(Int8), 44, fpOut);
    }

    // Clear delay lines
    for(i=0; i<NL; i++)
    {
        x[i] = 0;
    }
    for(i=0; i<DL; i++)
    {
        y[i] = 0;
    }
```

```

    printf("Exp --- IIR filter experiment\n");

    // Filter test
    while (fread(temp, sizeof(Int8), 2, fpIn) == 2)
    {
        in = (temp[0]&0xFF)|(temp[1]<<8);
        inputIIR[count] = in;

        // Filter the data
        fixPoint_IIR(in, x, y, num, NL, den, DL);
        outputIIR[count] = *y;
        temp[0] = (y[0]&0xFF);
        temp[1] = (y[0]>>8)&0xFF;
        fwrite(temp, sizeof(Int8), 2, fpOut);

        count++;
    }
    fclose(fpIn);
    fclose(fpOut);
    printf("Exp --- completed\n");
}



/*
* fixPoint_directIIR.c
*
*  Created on: May 25, 2012
*      Author: BLEE
*
*  Description: This is the fixed-point IIR filter in direct form-I realization
*
*  For the book "Real Time Digital Signal Processing:
*               Fundamentals, Implementation and Application, 3rd Ed"
*               By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
*               Publisher: John Wiley and Sons, Ltd
*
*/

#include "tistdtypes.h"
#include "fixPointIIR.h"


void fixPoint_IIR(Int16 in, Int16 *x, Int16 *y, Int16 *b, Int16 nb, Int16 *a, Int16 na)
{
    Int32 z1,z2;
    Int16 i;

    for(i=nb-1; i>0; i--)         // Update the delay line x[]
    {
        x[i] = x[i-1];
    }
    x[0] = in;                    // Insert new data to delay line x[0]
```

```

    for(z1=0, i=0; i<nb; i++)     // Filter the x[] with coefficient b[]
    {
        z1 += (Int32)x[i] * b[i];
    }

    for(i=na-1; i>0; i--)         // Update the y delay line
    {
        y[i] = y[i-1];
    }

    for(z2=0, i=1; i<na; i++)     // Filter the y[] with coefficient a[]
    {
        z2 += (Int32)y[i] * a[i];
    }

    z1 = z1 - z2;                 // Q15 data filtered using Q11 coefficients
    z1 += 0x400;                  // Rounding
    y[0] = (Int16)(z1>>11);       // Place the Q15 result into y[0]
}
```

6.3 MATLAB

```
clear; clc
samplingRate = 8E3;
n = 0:100;

% impulse = int16([hex2dec('7FFF'), zeros(1,100)])
% sin1 = int16(round(0.25 * sin(2 * pi * 800/samplingRate * n) * 2^15))
% sin2 = int16(round(0.25 * sin(2 * pi * 1600/samplingRate * n) * 2^15))

% audiowrite('impulse.wav', impulse,samplingRate)
% audiowrite('sin1.wav',sin1, samplingRate)
% audiowrite('sin2.wav',sin2, samplingRate)

impIn = audioread('impulse.wav');
sin1In = audioread('sin1.wav');
sin2In = audioread('sin2.wav');

FIR_imp_out = audioread('FIR_imp_out.wav');
FIR_sin1_out = audioread('FIR_sin1_out.wav');
FIR_sin2_out = audioread('FIR_sin2_out.wav');

IIR_imp_out = audioread('IIR_imp_out.wav');
IIR_sin1_out = audioread('IIR_sin1_out.wav');
IIR_sin2_out = audioread('IIR_sin2_out.wav');



figure
subplot(3, 1, 1);
hold on
    plot(n, impIn, 'linewidth', 1.25);
```

```
    plot(n, FIR_imp_out, 'linewidth', 1.25);
    plot(n, IIR_imp_out, 'linewidth', 1.25);
    grid on
    title("Impulse")
    xlabel("Samples");
    ylabel("Magnitude");
    legend(["Input", "FIR filtered", "IIR filtered"])
hold off

subplot(3, 1, 2);
hold on
    plot(n, sin1In, 'linewidth', 1.25);
    plot(n, FIR_sin1_out, 'linewidth', 1.25);
    plot(n, IIR_sin1_out, 'linewidth', 1.25);
    grid on
    title("Sinusoid, 800 Hz")
    xlabel("Samples");
    ylabel("Magnitude");
    legend(["Input", "FIR filtered", "IIR filtered"])
hold off

subplot(3, 1, 3);
hold on
    plot(n, sin2In, 'linewidth', 1.25);
    plot(n, FIR_sin2_out, 'linewidth', 1.25);
    plot(n, IIR_sin2_out, 'linewidth', 1.25);
    grid on
    title("Sinusoid, 1600 Hz")
    xlabel("Samples");
    ylabel("Magnitude");
    legend(["Input", "FIR filtered", "IIR filtered"])
hold off
sgtitle("Superimposed inputs and outputs vs. samples")

%%

IIR_filt_imp = filter([0.000406742095947265625, 0.002439975738525390625,
 0.006099700927734375, 0.0081329345703125, 0.006099700927734375,
  0.002439975738525390625, 0.000406742095947265625], [ 1, -3.494384765625,
   5.425048828125, -4.68896484375, 2.35791015625,
    -0.64990234375, 0.076416015625], impIn);

IIR_filt_sin1 = filter([0.000406742095947265625, 0.002439975738525390625,
 0.006099700927734375, 0.0081329345703125, 0.006099700927734375,
  0.002439975738525390625, 0.000406742095947265625],
   [ 1, -3.494384765625, 5.425048828125, -4.68896484375, 2.35791015625,
    -0.64990234375, 0.076416015625], sin1In);

IIR_filt_sin2 = filter([0.000406742095947265625, 0.002439975738525390625,
 0.006099700927734375, 0.0081329345703125, 0.006099700927734375,
  0.002439975738525390625, 0.000406742095947265625], [ 1, -3.494384765625,
   5.425048828125, -4.68896484375, 2.35791015625, -0.64990234375, 0.076416015625],
    sin2In);
```

```


FIR_filt_imp = filter([-0.00677490234375, 0.02481079101562, 0.08442687988281,
 0.16859436035156, 0.24412536621093, 0.27458190917968, 0.24412536621093,
 0.16859436035156, 0.08442687988281, 0.02481079101562, 0.00677490234375], [1], impIn);

FIR_filt_sin1 = filter([-0.00677490234375, 0.02481079101562, 0.08442687988281,
 0.16859436035156, 0.24412536621093, 0.27458190917968, 0.24412536621093,
 0.16859436035156, 0.08442687988281, 0.02481079101562, 0.00677490234375], [1],
 sin1In);

FIR_filt_sin2 = filter([-0.00677490234375, 0.02481079101562, 0.08442687988281,
 0.16859436035156, 0.24412536621093, 0.27458190917968, 0.24412536621093,
 0.16859436035156, 0.08442687988281, 0.02481079101562, 0.00677490234375], [1],
 sin2In);

figure
subplot(2,1,1)
hold on
    plot(n, IIR_filt_imp, 'linewidth', 1.25)
    plot(n, IIR_imp_out, 'linewidth', 1.25)
    plot(n, abs(IIR_filt_imp - IIR_imp_out), 'linewidth', 1.25)
    grid on
    title("IIR")
    xlabel("Samples");
    ylabel("Magnitude");
    legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
hold off
subplot(2,1,2)
hold on
    plot(n, FIR_filt_imp, 'linewidth', 1.25)
    plot(n, FIR_imp_out, 'linewidth', 1.25)
    plot(n, abs(FIR_filt_imp - FIR_imp_out), 'linewidth', 1.25)
    grid on
    title("FIR")
    xlabel("Samples");
    ylabel("Magnitude");
    legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
hold off
sgtitle("Impulse: MATLAB compared with board")

figure
subplot(2,1,1)
hold on
    plot(n, IIR_filt_sin1, 'linewidth', 1.25)
    plot(n, IIR_sin1_out, 'linewidth', 1.25)
    plot(n, abs(IIR_filt_sin1 - IIR_sin1_out), 'linewidth', 1.25)
    grid on
    xlabel("Samples");
    ylabel("Magnitude");
    legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
    title("IIR")
hold off
subplot(2,1,2)
```

```
hold on
    plot(n, FIR_filt_sin1, 'linewidth', 1.25)
    plot(n, FIR_sin1_out, 'linewidth', 1.25)
    plot(n, abs(FIR_filt_sin1 - FIR_sin1_out), 'linewidth', 1.25)
    grid on
    xlabel("Samples");
    ylabel("Magnitude");
    legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
    title("FIR")
hold off
sgtitle("Sinusoid, 800 Hz: MATLAB compared with board")

figure
subplot(2,1,1)
hold on
    plot(n, IIR_filt_sin2, 'linewidth', 1.25)
    plot(n, IIR_sin2_out, 'linewidth', 1.25)
    plot(n, abs(IIR_filt_sin2 - IIR_sin2_out), 'linewidth', 1.25)
    grid on
    xlabel("Samples");
    ylabel("Magnitude");
    legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
    title("IIR")
hold off
subplot(2,1,2)
hold on
    plot(n, FIR_filt_sin2, 'linewidth', 1.25)
    plot(n, FIR_sin2_out, 'linewidth', 1.25)
    plot(n, abs(FIR_filt_sin2 - FIR_sin2_out), 'linewidth', 1.25)
    grid on
    xlabel("Samples");
    ylabel("Magnitude");
    legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
    title("FIR")
hold off
sgtitle("Sinusoid, 1600 Hz: MATLAB compared with board")

%%

IIR_imp_comp = (sum(abs(IIR_filt_imp - IIR_imp_out)) / length(n)) * 100
FIR_imp_comp = (sum(abs(FIR_filt_imp - FIR_imp_out)) / length(n)) * 100

IIR_sin1_comp = (sum(abs(IIR_filt_sin1 - IIR_sin1_out)) / length(n)) * 100
FIR_sin1_comp = (sum(abs(FIR_filt_sin1 - FIR_sin1_out)) / length(n)) * 100

IIR_sin2_comp = (sum(abs(IIR_filt_sin2 - IIR_sin2_out)) / length(n)) * 100
FIR_sin2_comp = (sum(abs(FIR_filt_sin2 - FIR_sin2_out)) / length(n)) * 100
```