CALIFORNIA STATE UNIVERSITY
LONG BEACH

Nov 2019
EE 489

# Lab # 3

## Digital Filter Design

prepared by
Brian Fox
Arya Daroui
Inessa Lopez

## 1. Introduction

## 2. Method

## 3. Data

**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 |

## 4. Discussion

## 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)
```

```
159     {
160         fpIn = fopen("..\\data\\impulse.wav", "rb");
161         fpOut = fopen("..\\data\\FIR_imp_out.wav", "wb");
162     }
163     else
164     {
165         fpIn = fopen("..\\data\\input.pcm", "rb");
166         fpOut = fopen("..\\data\\output.pcm", "wb");
167     }
168 
169     if (fpIn == NULL)
170     {
171         printf("Can't open input file\n");
172         exit(0);
173     }
174 
175     if (c == 2)
176     {
177         fread(waveHeader, sizeof(Int8), 44, fpIn);
178         fwrite(waveHeader, sizeof(Int8), 44, fpOut);
179     }
180 
181     // Initialize for filtering process
182     for (i=0; i<NUM_TAPS; i++)
183     {
184         w[i] = 0;
185     }
186     index = 0;
187 
188 
189     // Begin filtering the data
190     while (fread(temp, sizeof(Int8), NUM_DATA*2, fpIn) == (NUM_DATA*2))
191     {
192         for (k=0, i=0; i<NUM_DATA; i++)
193         {
194             x[i] = (temp[k]&0xFF)|(temp[k+1]<<8);
195             k += 2;
196         }
197         // Filter the data x and save output y
198         blockFir(x, NUM_DATA, blockFirCoef, NUM_TAPS, y, w, &index);
199 
200         for (k=0, i=0; i<NUM_DATA; i++)
201         {
202             temp[k++] = (y[i]&0xFF);
203             temp[k++] = (y[i]>>8)&0xFF;
204         }
205         fwrite(temp, sizeof(Int8), NUM_DATA*2, fpOut);
206     }
207 
208     fclose(fpIn);
209     fclose(fpOut);
210 
211     printf("\nExp --- completed\n");
212 
```

```
    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");
```

```
95      }
96      else
97      {
98          fpIn = fopen("..\\data\\input.pcm", "rb");
99          fpOut = fopen("..\\data\\output.pcm", "wb");
100     }
101     // Open file for read input data
102     if (fpIn == NULL)
103     {
104         printf("Can't open input data file\n");
105         exit(0);
106     }
107
108     if (c == 2)        // Create WAVE data file header
109     {
110         fread(waveHeader, sizeof(Int8), 44, fpIn);
111         fwrite(waveHeader, sizeof(Int8), 44, fpOut);
112     }
113
114     // Clear delay lines
115     for(i=0; i<NL; i++)
116     {
117         x[i] = 0;
118     }
119     for(i=0; i<DL; i++)
120     {
121         y[i] = 0;
122     }
123
124     printf("Exp --- IIR filter experiment\n");
125
126     // Filter test
127     while (fread(temp, sizeof(Int8), 2, fpIn) == 2)
128     {
129         in = (temp[0]&0xFF)|(temp[1]<<8);
130         inputIIR[count] = in;
131
132         // Filter the data
133         fixPoint_IIR(in, x, y, num, NL, den, DL);
134         outputIIR[count] = *y;
135         temp[0] = (y[0]&0xFF);
136         temp[1] = (y[0]>>8)&0xFF;
137         fwrite(temp, sizeof(Int8), 2, fpOut);
138
139         count++;
140     }
141     fclose(fpIn);
142     fclose(fpOut);
143     printf("Exp --- completed\n");
144
145
146 //    Int16  in,i,c;
147 //    FILE   *fpIn,*fpOut;
148 //    Int8   temp[2];
```

```
149 //    Uint8  waveHeader[44];
150 //    Int16 inputIIR[101];
151 //    Int16 outputIIR[101];
152     count = 0;
153 //
154 //    printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
155 //    scanf ("%d", &c);
156     c = 2;
157 
158     if (c == 2)
159     {
160         fpIn = fopen("..\\data\\sin1.wav", "rb");
161         fpOut = fopen("..\\data\\IIR_sin1_out.wav", "wb");
162     }
163     else
164     {
165         fpIn = fopen("..\\data\\input.pcm", "rb");
166         fpOut = fopen("..\\data\\output.pcm", "wb");
167     }
168     // Open file for read input data
169     if (fpIn == NULL)
170     {
171         printf("Can't open input data file\n");
172         exit(0);
173     }
174 
175     if (c == 2)        // Create WAVE data file header
176     {
177         fread(waveHeader, sizeof(Int8), 44, fpIn);
178         fwrite(waveHeader, sizeof(Int8), 44, fpOut);
179     }
180 
181     // Clear delay lines
182     for(i=0; i<NL; i++)
183     {
184         x[i] = 0;
185     }
186     for(i=0; i<DL; i++)
187     {
188         y[i] = 0;
189     }
190 
191     printf("Exp --- IIR filter experiment\n");
192 
193     // Filter test
194     while (fread(temp, sizeof(Int8), 2, fpIn) == 2)
195     {
196         in = (temp[0]&0xFF)|(temp[1]<<8);
197         inputIIR[count] = in;
198 
199         // Filter the data
200         fixPoint_IIR(in, x, y, num, NL, den, DL);
201         outputIIR[count] = *y;
202         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;
    }
```

```
257
258     printf("Exp --- IIR filter experiment\n");
259
260     // Filter test
261     while (fread(temp, sizeof(Int8), 2, fpIn) == 2)
262     {
263         in = (temp[0]&0xFF)|(temp[1]<<8);
264         inputIIR[count] = in;
265
266         // Filter the data
267         fixPoint_IIR(in, x, y, num, NL, den, DL);
268         outputIIR[count] = *y;
269         temp[0] = (y[0]&0xFF);
270         temp[1] = (y[0]>>8)&0xFF;
271         fwrite(temp, sizeof(Int8), 2, fpOut);
272
273         count++;
274     }
275     fclose(fpIn);
276     fclose(fpOut);
277     printf("Exp --- completed\n");
278 }
279
280
281
282 /*
283 * fixPoint_directIIR.c
284 *
285 *  Created on: May 25, 2012
286 *      Author: BLEE
287 *
288 *  Description: This is the fixed-point IIR filter in direct form-I realization
289 *
290 *  For the book "Real Time Digital Signal Processing:
291 *                Fundamentals, Implementation and Application, 3rd Ed"
292 *                By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
293 *                Publisher: John Wiley and Sons, Ltd
294 *
295 */
296
297 #include "tistdtypes.h"
298 #include "fixPointIIR.h"
299
300
301 void fixPoint_IIR(Int16 in, Int16 *x, Int16 *y, Int16 *b, Int16 nb, Int16 *a, Int16 na)
302 {
303     Int32 z1,z2;
304     Int16 i;
305
306     for(i=nb-1; i>0; i--)         // Update the delay line x[]
307     {
308         x[i] = x[i-1];
309     }
310     x[0] = in;                    // Insert new data to delay line x[0]
```

```
311
312     for(z1=0, i=0; i<nb; i++)     // Filter the x[] with coefficient b[]
313     {
314         z1 += (Int32)x[i] * b[i];
315     }
316
317     for(i=na-1; i>0; i--)         // Update the y delay line
318     {
319         y[i] = y[i-1];
320     }
321
322     for(z2=0, i=1; i<na; i++)     // Filter the y[] with coefficient a[]
323     {
324         z2 += (Int32)y[i] * a[i];
325     }
326
327     z1 = z1 - z2;                 // Q15 data filtered using Q11 coefficients
328     z1 += 0x400;                  // Rounding
329     y[0] = (Int16)(z1>>11);       // Place the Q15 result into y[0]
330 }
```

### 6.3 MATLAB

```
1  clear; clc
2  samplingRate = 8E3;
3  n = 0:100;
4
5  % impulse = int16([hex2dec('7FFF'), zeros(1,100)])
6  % sin1 = int16(round(0.25 * sin(2 * pi * 800/samplingRate * n) * 2^15))
7  % sin2 = int16(round(0.25 * sin(2 * pi * 1600/samplingRate * n) * 2^15))
8
9  % audiowrite('impulse.wav', impulse,samplingRate)
10 % audiowrite('sin1.wav',sin1, samplingRate)
11 % audiowrite('sin2.wav',sin2, samplingRate)
12
13 impIn = audioread('impulse.wav');
14 sin1In = audioread('sin1.wav');
15 sin2In = audioread('sin2.wav');
16
17 FIR_imp_out = audioread('FIR_imp_out.wav');
18 FIR_sin1_out = audioread('FIR_sin1_out.wav');
19 FIR_sin2_out = audioread('FIR_sin2_out.wav');
20
21 IIR_imp_out = audioread('IIR_imp_out.wav');
22 IIR_sin1_out = audioread('IIR_sin1_out.wav');
23 IIR_sin2_out = audioread('IIR_sin2_out.wav');
24
25
26
27 figure
28 subplot(3, 1, 1);
29 hold on
30     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([0.00040674209594726562 5, 0.00243997573852539062 5,
 0.00609970092773437 5, 0.008132934570312 5, 0.00609970092773437 5,
  0.00243997573852539062 5, 0.00040674209594726562 5], [ 1, -3.494384765625,
   5.425048828125, -4.68896484375, 2.35791015625,
    -0.64990234375, 0.076416015625], impIn);

IIR_filt_sin1 = filter([0.00040674209594726562 5, 0.00243997573852539062 5,
 0.00609970092773437 5, 0.008132934570312 5, 0.00609970092773437 5,
  0.00243997573852539062 5, 0.00040674209594726562 5],
   [ 1, -3.494384765625, 5.425048828125, -4.68896484375, 2.35791015625,
    -0.64990234375, 0.076416015625], sin1In);

IIR_filt_sin2 = filter([0.00040674209594726562 5, 0.00243997573852539062 5,
 0.00609970092773437 5, 0.008132934570312 5, 0.00609970092773437 5,
  0.00243997573852539062 5, 0.00040674209594726562 5], [ 1, -3.494384765625,
   5.425048828125, -4.68896484375, 2.35791015625, -0.64990234375, 0.076416015625],
    sin2In);

```

```
85
86
87  FIR_filt_imp = filter([-0.00677490234375, 0.02481079101562, 0.08442687988281,
88   0.16859436035156, 0.24412536621093, 0.27458190917968, 0.24412536621093,
89   0.16859436035156, 0.08442687988281, 0.02481079101562, 0.00677490234375], [1], impIn);
90
91  FIR_filt_sin1 = filter([-0.00677490234375, 0.02481079101562, 0.08442687988281,
92   0.16859436035156, 0.24412536621093, 0.27458190917968, 0.24412536621093,
93   0.16859436035156, 0.08442687988281, 0.02481079101562, 0.00677490234375], [1],
94   sin1In);
95
96  FIR_filt_sin2 = filter([-0.00677490234375, 0.02481079101562, 0.08442687988281,
97   0.16859436035156, 0.24412536621093, 0.27458190917968, 0.24412536621093,
98   0.16859436035156, 0.08442687988281, 0.02481079101562, 0.00677490234375], [1],
99   sin2In);
100
101 figure
102 subplot(2,1,1)
103 hold on
104     plot(n, IIR_filt_imp, 'linewidth', 1.25)
105     plot(n, IIR_imp_out, 'linewidth', 1.25)
106     plot(n, abs(IIR_filt_imp - IIR_imp_out), 'linewidth', 1.25)
107     grid on
108     title("IIR")
109     xlabel("Samples");
110     ylabel("Magnitude");
111     legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
112 hold off
113 subplot(2,1,2)
114 hold on
115     plot(n, FIR_filt_imp, 'linewidth', 1.25)
116     plot(n, FIR_imp_out, 'linewidth', 1.25)
117     plot(n, abs(FIR_filt_imp - FIR_imp_out), 'linewidth', 1.25)
118     grid on
119     title("FIR")
120     xlabel("Samples");
121     ylabel("Magnitude");
122     legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
123 hold off
124 sgtitle("Impulse: MATLAB compared with board")
125
126 figure
127 subplot(2,1,1)
128 hold on
129     plot(n, IIR_filt_sin1, 'linewidth', 1.25)
130     plot(n, IIR_sin1_out, 'linewidth', 1.25)
131     plot(n, abs(IIR_filt_sin1 - IIR_sin1_out), 'linewidth', 1.25)
132     grid on
133     xlabel("Samples");
134     ylabel("Magnitude");
135     legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
136     title("IIR")
137 hold off
138 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
```