

## Lab # 3

---

### Digital Filter Design



## 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 textbook "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 differ-

ent 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.

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

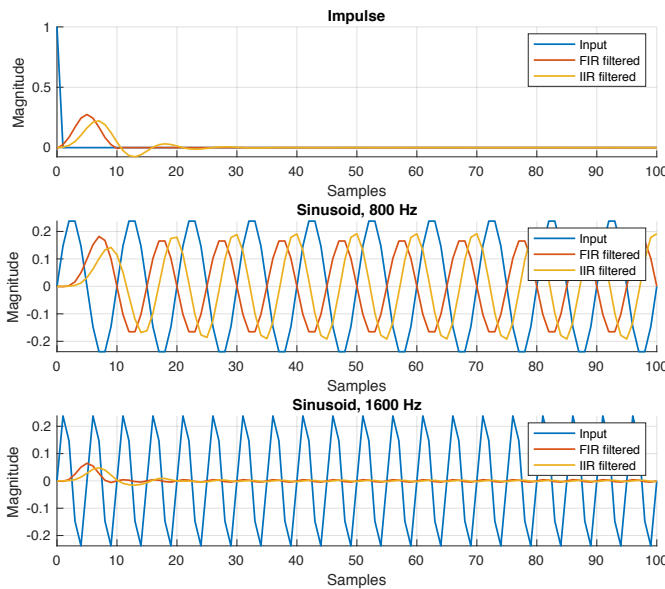
From there, we took the average of the absolute difference to compare the different inputs.

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

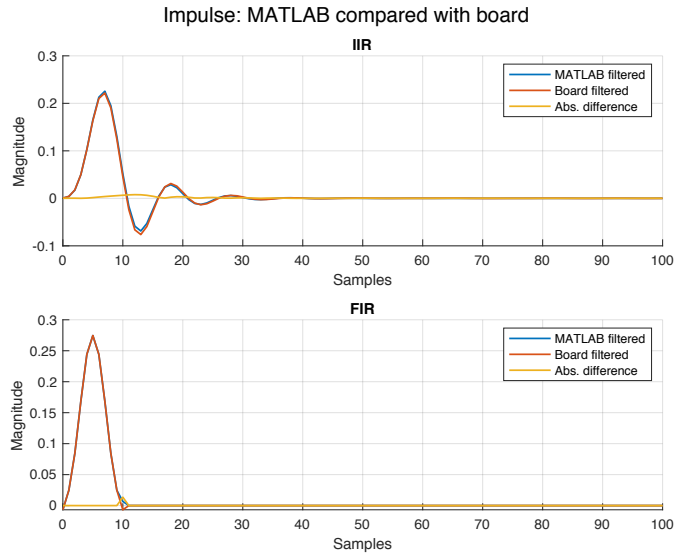
Where  $N$  is the number of samples  $n$  in the absolute difference  $d$ . The final results are shown in Table 1. We chose to use average absolute difference over average percent error because dividing by the extremely small amplitudes made the error seem magnitudes worse than is adequately represented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

## 3. Data

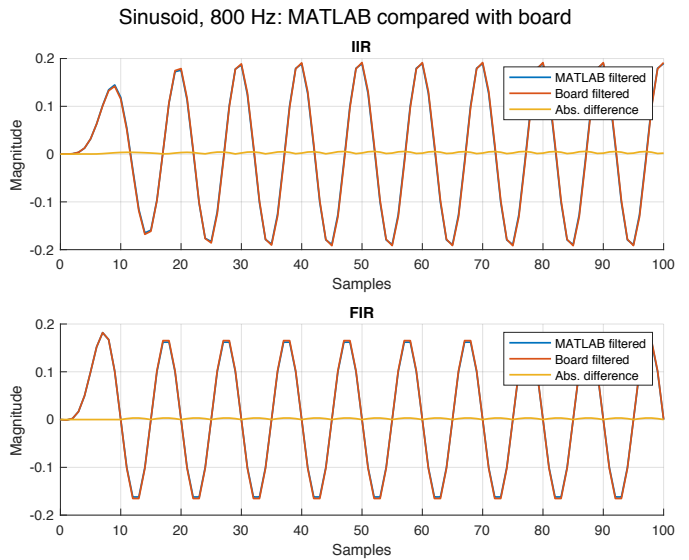
Superimposed inputs and outputs vs. samples



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



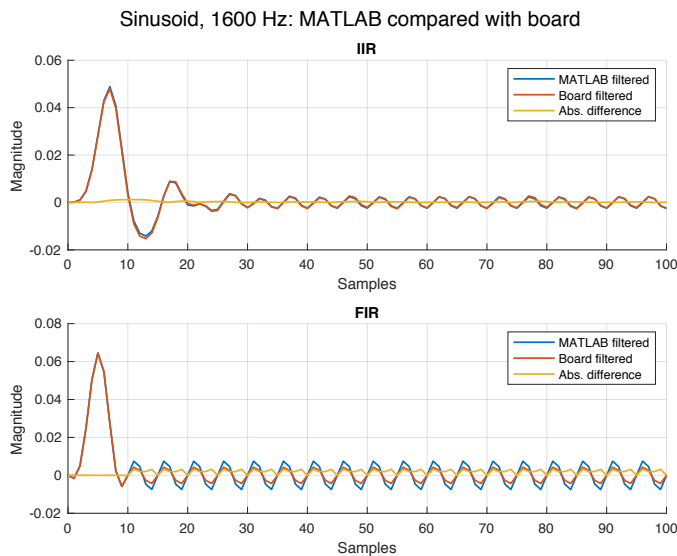
**Figure 2.** Comparison between the C55x filter implementation and MATLAB for the impulse response



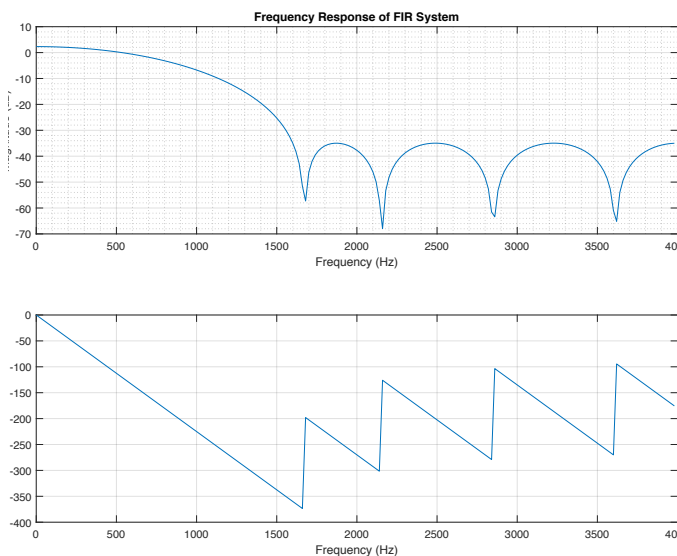
**Figure 3.** Comparison between the C55x filter implementation and MATLAB for the 800 Hz sinusoid

**Table 1.** Comparison of C55x and MATLAB filter implementations

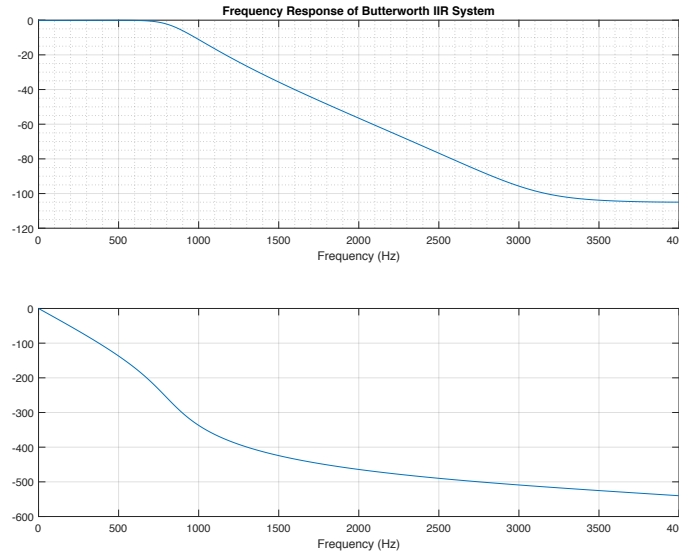
	FIR abs. diff. (%)	IIR abs. diff. (%)
Impulse	0.0135	0.0925
800 Hz	0.1845	0.2935
1600 Hz	0.1848	0.0268



**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 that's 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 there's 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 board's steady state amplitude is 0.1917, while the MATLAB's 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  $-6.1035 \times 10^{-5}$ , which seems extremely close to zero, until it is compared to the MATLAB result of  $4.5747 \times 10^{-9}$ . 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

```

1  /*
2  * Experiment assembly implementation of block FIR filter - Chapter 3
3  * blockFirCoef.h
4  *
5  * Description: This is the filter coefficient file for assembly FIR filter
6  *
7  *   Created on: May 13, 2012
8  *   Author: BLEE
9  *
10 *       For the book "Real Time Digital Signal Processing:
11 *                   Fundamentals, Implementation and Application, 3rd Ed"
12 *                   By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
13 *                   Publisher: John Wiley and Sons, Ltd
14 */
15
16 Int16 blockFirCoef[NUM_TAPS]={
17 (Int16)(-0.00677490234375*32767.0),(Int16)(0.024810791015625*32767.0),
18 (Int16)( 0.0844268798828125*32767.0),(Int16)(0.1685943603515625*32767.0),
19 (Int16)( 0.2441253662109375*32767.0),(Int16)(0.2745819091796875*32767.0),
20 (Int16)( 0.2441253662109375*32767.0),(Int16)(0.1685943603515625*32767.0),
21 (Int16)( 0.0844268798828125*32767.0),(Int16)(0.024810791015625*32767.0),
22 (Int16)(-0.00677490234375*32767.0)
23 };
24
25
26 /*
27 * Experiment assembly implementation of block FIR filter - Chapter 3
28 * blockFir.h
29 *
30 * Description: This is the header file for fixed-point FIR filter
31 *
32 *   Created on: May 13, 2012
33 *   Author: BLEE
34 *
35 *       For the book "Real Time Digital Signal Processing:
36 *                   Fundamentals, Implementation and Application, 3rd Ed"
37 *                   By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
38 *                   Publisher: John Wiley and Sons, Ltd
39 */
40
41 #define  NUM_TAPS    11
42 #define  NUM_DATA    101
43
44 void blockFir(Int16 *x, Int16 blkSize,
45              Int16 *h, Int16 order,
46              Int16 *y,
47              Int16 *w, Int16 *index);
48
49 /*
50 ; * Experiment assembly implementation of block FIR filter - Chapter 3

```

```

51 ; * blockFir.asm
52 ; *
53 ; * Description: This is the assembly language implementation of block FIR filter
54 ; *
55 ; * Created on: May 13, 2012
56 ; * Author: BLEE
57 ; *
58 ; * For the book "Real Time Digital Signal Processing:
59 ; * Fundamentals, Implementation and Application, 3rd Ed"
60 ; * By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
61 ; * Publisher: John Wiley and Sons, Ltd
62 ; */
63
64 .mmregs
65
66 .sect ".text:fir"
67 .align 4
68
69 .def _blockFir
70
71 ;-----
72 ; void blockFir(Int16 *x,          => AR0
73 ;               Int16 blkSize,     => T0
74 ;               Int16 *h,          => AR1
75 ;               Int16 order,       => T1
76 ;               Int16 *y,          => AR2
77 ;               Int16 *w,          => AR3
78 ;               Int16 *index)      => AR4
79 ;-----
80
81 _blockFir:
82     pshm ST1_55          ; Save ST1, ST2, and ST3
83     pshm ST2_55
84     pshm ST3_55
85
86     or #0x340,mmap(ST1_55); Set FRCT,SXMD,SATD
87     bset SMUL            ; Set SMUL
88     mov mmap(AR1),BSA01   ; AR1=base address for coeff
89     mov mmap(T1),BK03     ; Set coefficient array size (order)
90     mov mmap(AR3),BSA23   ; AR3=base address for signal buffer
91     or #0xA,mmap(ST2_55) ; AR1 & AR3 as circular pointers
92     mov #0,AR1           ; Coefficient start from h[0]
93     mov *AR4,AR3         ; Signal buffer start from w[index]
94 || sub #1,T0             ; T0=blkSize-1
95     mov T0,BRC0          ; Initialize outer loop to blkSize-1
96     sub #3,T1,T0         ; T0=order-3
97     mov T0,CSR           ; Initialize inner loop order-2 times
98 || rptblocal sample_loop-1 ; Start the outer loop
99     mov *AR0+,*AR3       ; Put the new sample to signal buffer
100    mpym *AR3+,*AR1+,AC0   ; Do the 1st operation
101 || rpt CSR              ; Start the inner loop
102    macm *AR3+,*AR1+,AC0
103    macmr *AR3,*AR1+,AC0   ; Do the last operation with rounding
104    mov hi(AC0),*AR2+     ; Save Q15 filtered value

```



```
105 sample_loop
106
107     popm    ST3_55                ; Restore ST1, ST2, and ST3
108     popm    ST2_55
109     popm    ST1_55
110     mov     AR3,*AR4              ; Update signal buffer index
111     ret
112
113     .end
114
115 /*
116 * Experiment assembly implementation of block FIR filter - Chapter 3
117 * blockFirTest.c
118 *
119 * Description: This is the test file for the block FIR filter
120 *
121 * Created on: May 13, 2012
122 * Author: BLEE
123 *
124 * For the book "Real Time Digital Signal Processing:
125 * Fundamentals, Implementation and Application, 3rd Ed"
126 * By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
127 * Publisher: John Wiley and Sons, Ltd
128 */
129
130 #include <stdlib.h>
131 #include <stdio.h>
132 #include "tistdtypes.h"
133 #include "blockFir.h"
134
135 /* Define DSP system memory map */
136 #pragma DATA_SECTION(blockFirCoef, ".const:fir");
137 #pragma DATA_SECTION(w, ".bss:fir");
138
139 #include "blockFirCoef.h"
140
141
142 Int16 w[NUM_TAPS];
143
144 void main()
145 {
146     FILE *fpIn,*fpOut;
147     Int16 i,k,c,
148         index;          // Delay line index
149     Int16 x[NUM_DATA], // Input data
150         y[NUM_DATA];   // Output data
151     Int8 temp[NUM_DATA*2];
152     UInt8 waveHeader[44];
153
154     printf("Exp --- Assembly program_Block FIR filter experiment\n");
155     printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
156     scanf ("%d", &c);
157
158     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
```

```
213 printf("Exp --- Assembly program_Block FIR filter experiment\n");
214 printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
215 scanf ("%d", &c);
216
217 if (c == 2)
218 {
219     fpIn = fopen("../data\\sin1.wav", "rb");
220     fpOut = fopen("../data\\FIR_sin1_out.wav", "wb");
221 }
222 else
223 {
224     fpIn = fopen("../data\\input.pcm", "rb");
225     fpOut = fopen("../data\\output.pcm", "wb");
226 }
227
228 if (fpIn == NULL)
229 {
230     printf("Can't open input file\n");
231     exit(0);
232 }
233
234 if (c == 2)
235 {
236     fread(waveHeader, sizeof(Int8), 44, fpIn);
237     fwrite(waveHeader, sizeof(Int8), 44, fpOut);
238 }
239
240 // Initialize for filtering process
241 for (i=0; i<NUM_TAPS; i++)
242 {
243     w[i] = 0;
244 }
245 index = 0;
246
247
248 // Begin filtering the data
249 while (fread(temp, sizeof(Int8), NUM_DATA*2, fpIn) == (NUM_DATA*2))
250 {
251     for (k=0, i=0; i<NUM_DATA; i++)
252     {
253         x[i] = (temp[k]&0xFF)|(temp[k+1]<<8);
254         k += 2;
255     }
256     // Filter the data x and save output y
257     blockFir(x, NUM_DATA, blockFirCoef, NUM_TAPS, y, w, &index);
258
259     for (k=0, i=0; i<NUM_DATA; i++)
260     {
261         temp[k++] = (y[i]&0xFF);
262         temp[k++] = (y[i]>>8)&0xFF;
263     }
264     fwrite(temp, sizeof(Int8), NUM_DATA*2, fpOut);
265 }
266
```

```
267     fclose(fpIn);
268     fclose(fpOut);
269
270     printf("\nExp --- completed\n");
271
272     printf("Exp --- Assembly program_Block FIR filter experiment\n");
273     printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
274     scanf ("%d", &c);
275
276     if (c == 2)
277     {
278         fpIn = fopen("../data\\sin2.wav", "rb");
279         fpOut = fopen("../data\\FIR_sin2_out.wav", "wb");
280     }
281     else
282     {
283         fpIn = fopen("../data\\input.pcm", "rb");
284         fpOut = fopen("../data\\output.pcm", "wb");
285     }
286
287     if (fpIn == NULL)
288     {
289         printf("Can't open input file\n");
290         exit(0);
291     }
292
293     if (c == 2)
294     {
295         fread(waveHeader, sizeof(Int8), 44, fpIn);
296         fwrite(waveHeader, sizeof(Int8), 44, fpOut);
297     }
298
299     // Initialize for filtering process
300     for (i=0; i<NUM_TAPS; i++)
301     {
302         w[i] = 0;
303     }
304     index = 0;
305
306
307     // Begin filtering the data
308     while (fread(temp, sizeof(Int8), NUM_DATA*2, fpIn) == (NUM_DATA*2))
309     {
310         for (k=0, i=0; i<NUM_DATA; i++)
311         {
312             x[i] = (temp[k]&0xFF)|(temp[k+1]<<8);
313             k += 2;
314         }
315         // Filter the data x and save output y
316         blockFir(x, NUM_DATA, blockFirCoef, NUM_TAPS, y, w, &index);
317
318         for (k=0, i=0; i<NUM_DATA; i++)
319         {
320             temp[k++] = (y[i]&0xFF);
```

```
321         temp[k++] = (y[i]>>8)&0xFF;
322     }
323     fwrite(temp, sizeof(Int8), NUM_DATA*2, fpOut);
324 }
325
326 fclose(fpIn);
327 fclose(fpOut);
328
329 printf("\nExp --- completed\n");
330 }
```

## 6.2 IIR

```
1  /*
2  * fixPointIIR.h
3  *
4  *   Created on: May 25, 2012
5  *   Author: BLEE
6  *
7  *   Description: This is the header file for the fixed-point IIR filter in direct form-I
8  *
9  *   For the book "Real Time Digital Signal Processing:
10 *       Fundamentals, Implementation and Application, 3rd Ed"
11 *       By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
12 *       Publisher: John Wiley and Sons, Ltd
13 *
14 */
15
16 void fixPoint_IIR(Int16 in, Int16 *x, Int16 *y,
17                 Int16 *b, Int16 nb, Int16 *a, Int16 na);
18
19 /*
20 * fixPoint_directIIRTest.c
21 *
22 *   Created on: May 25, 2012
23 *   Author: BLEE
24 *
25 *   Description: This is the test program for fixed-point direct form-I IIR filter
26 *
27 *   For the book "Real Time Digital Signal Processing:
28 *       Fundamentals, Implementation and Application, 3rd Ed"
29 *       By Sen M. Kuo, Bob H. Lee, and Wenshun Tian
30 *       Publisher: John Wiley and Sons, Ltd
31 *
32 */
33
34 #include <stdio.h>
35 #include <stdlib.h>
36 #include "tistdtypes.h"
37 #include "fixPointIIR.h"
38
39 // Coefficient length
40 #define NL 7
```

```
41 #define DL 7
42 #define Q11 2048 // For making Q11 format filter coefficients
43 #define RND 0.5
44
45 // Filter coefficients obtained from MATLAB script
46 /*
47     Rp=0.1; % Passband ripple
48     Rs=60; % Stopband attenuation
49     [N,Wn]=ellipord(836/4000,1300/4000,Rp,Rs); % Filter order & scaling factor
50     [b,a]=ellip(N,Rp,Rs,Wn); % Lowpass IIR filter
51     [num,den]=iirlp2bp(b,a,0.5,[0.25, 0.75]); % Bandpass IIR filter
52
53 Int16 num[NL] = =
54 (Int16)(0.0004*Q11+RND),
55 (Int16)(0.0024*Q11+RND),
56 (Int16)(0.0060*Q11+RND),
57 (Int16)(0.0081*Q11+RND),
58 (Int16)(0.0060*Q11+RND),
59 (Int16)(0.0024*Q11+RND),
60 (Int16)(0.0004*Q11+RND)
61 };
62
63 Int16 den[DL] = {
64 (Int16)(1.0000*Q11+RND),
65 (Int16)(-3.4943*Q11+RND),
66 (Int16)(5.4250*Q11+RND),
67 (Int16)(-4.6889*Q11+RND),
68 (Int16)(2.3579*Q11+RND),
69 (Int16)(-0.6499*Q11+RND),
70 (Int16)(0.0764*Q11+RND)
71 };
72
73 // Filter delay lines
74 Int16 x[NL],y[DL];
75
76 void main()
77 {
78
79     Int16 in,i,c;
80     FILE *fpIn,*fpOut;
81     Int8 temp[2];
82     UInt8 waveHeader[44];
83     Int16 inputIIR[101];
84     Int16 outputIIR[101];
85     int count = 0;
86
87     // printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
88     // scanf ("%d", &c);
89     c = 2;
90
91     if (c == 2)
92     {
93         fpIn = fopen("../data\\impulse.wav", "rb");
94         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);
```



```
203     temp[1] = (y[0]>>8)&0xFF;
204     fwrite(temp, sizeof(Int8), 2, fpOut);
205
206     count++;
207 }
208 fclose(fpIn);
209 fclose(fpOut);
210 printf("Exp --- completed\n");
211
212
213 //  Int16  in,i,c;
214 //  FILE   *fpIn,*fpOut;
215 //  Int8   temp[2];
216 //  Uint8  waveHeader[44];
217 //  Int16  inputIIR[101];
218 //  Int16  outputIIR[101];
219 count = 0;
220 //
221 //  printf("Enter 1 for using PCM file, enter 2 for using WAV file\n");
222 //  scanf ("%d", &c);
223 c = 2;
224
225 if (c == 2)
226 {
227     fpIn = fopen("../data\\sin2.wav", "rb");
228     fpOut = fopen("../data\\IIR_sin2_out.wav", "wb");
229 }
230 else
231 {
232     fpIn = fopen("../data\\input.pcm", "rb");
233     fpOut = fopen("../data\\output.pcm", "wb");
234 }
235 // Open file for read input data
236 if (fpIn == NULL)
237 {
238     printf("Can't open input data file\n");
239     exit(0);
240 }
241
242 if (c == 2)          // Create WAVE data file header
243 {
244     fread(waveHeader, sizeof(Int8), 44, fpIn);
245     fwrite(waveHeader, sizeof(Int8), 44, fpOut);
246 }
247
248 // Clear delay lines
249 for(i=0; i<NL; i++)
250 {
251     x[i] = 0;
252 }
253 for(i=0; i<DL; i++)
254 {
255     y[i] = 0;
256 }
```

```

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);
```

```
31     plot(n, FIR_imp_out, 'linewidth', 1.25);
32     plot(n, IIR_imp_out, 'linewidth', 1.25);
33     grid on
34     title("Impulse")
35     xlabel("Samples");
36     ylabel("Magnitude");
37     legend(["Input", "FIR filtered", "IIR filtered"])
38 hold off
39
40 subplot(3, 1, 2);
41 hold on
42     plot(n, sin1In, 'linewidth', 1.25);
43     plot(n, FIR_sin1_out, 'linewidth', 1.25);
44     plot(n, IIR_sin1_out, 'linewidth', 1.25);
45     grid on
46     title("Sinusoid, 800 Hz")
47     xlabel("Samples");
48     ylabel("Magnitude");
49     legend(["Input", "FIR filtered", "IIR filtered"])
50 hold off
51
52 subplot(3, 1, 3);
53 hold on
54     plot(n, sin2In, 'linewidth', 1.25);
55     plot(n, FIR_sin2_out, 'linewidth', 1.25);
56     plot(n, IIR_sin2_out, 'linewidth', 1.25);
57     grid on
58     title("Sinusoid, 1600 Hz")
59     xlabel("Samples");
60     ylabel("Magnitude");
61     legend(["Input", "FIR filtered", "IIR filtered"])
62 hold off
63 sgtitle("Superimposed inputs and outputs vs. samples")
64
65 %%
66
67 IIR_filt_imp = filter([0.000406742095947265625, 0.002439975738525390625,
68     0.006099700927734375, 0.0081329345703125, 0.006099700927734375,
69     0.002439975738525390625, 0.000406742095947265625], [ 1, -3.494384765625,
70     5.425048828125, -4.68896484375, 2.35791015625,
71     -0.64990234375, 0.076416015625], impIn);
72
73 IIR_filt_sin1 = filter([0.000406742095947265625, 0.002439975738525390625,
74     0.006099700927734375, 0.0081329345703125, 0.006099700927734375,
75     0.002439975738525390625, 0.000406742095947265625],
76     [ 1, -3.494384765625, 5.425048828125, -4.68896484375, 2.35791015625,
77     -0.64990234375, 0.076416015625], sin1In);
78
79 IIR_filt_sin2 = filter([0.000406742095947265625, 0.002439975738525390625,
80     0.006099700927734375, 0.0081329345703125, 0.006099700927734375,
81     0.002439975738525390625, 0.000406742095947265625], [ 1, -3.494384765625,
82     5.425048828125, -4.68896484375, 2.35791015625, -0.64990234375, 0.076416015625],
83     sin2In);
84
```

```
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 sgttitle("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)
```

```
139 hold on
140     plot(n, FIR_filt_sin1, 'linewidth', 1.25)
141     plot(n, FIR_sin1_out, 'linewidth', 1.25)
142     plot(n, abs(FIR_filt_sin1 - FIR_sin1_out), 'linewidth', 1.25)
143     grid on
144     xlabel("Samples");
145     ylabel("Magnitude");
146     legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
147     title("FIR")
148 hold off
149 sgttitle("Sinusoid, 800 Hz: MATLAB compared with board")
150
151 figure
152 subplot(2,1,1)
153 hold on
154     plot(n, IIR_filt_sin2, 'linewidth', 1.25)
155     plot(n, IIR_sin2_out, 'linewidth', 1.25)
156     plot(n, abs(IIR_filt_sin2 - IIR_sin2_out), 'linewidth', 1.25)
157     grid on
158     xlabel("Samples");
159     ylabel("Magnitude");
160     legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
161     title("IIR")
162 hold off
163 subplot(2,1,2)
164 hold on
165     plot(n, FIR_filt_sin2, 'linewidth', 1.25)
166     plot(n, FIR_sin2_out, 'linewidth', 1.25)
167     plot(n, abs(FIR_filt_sin2 - FIR_sin2_out), 'linewidth', 1.25)
168     grid on
169     xlabel("Samples");
170     ylabel("Magnitude");
171     legend(["MATLAB filtered", "Board filtered", "Abs. difference"])
172     title("FIR")
173 hold off
174 sgttitle("Sinusoid, 1600 Hz: MATLAB compared with board")
175
176 %%
177
178 IIR_imp_comp = (sum(abs(IIR_filt_imp - IIR_imp_out)) / length(n)) * 100
179 FIR_imp_comp = (sum(abs(FIR_filt_imp - FIR_imp_out)) / length(n)) * 100
180
181 IIR_sin1_comp = (sum(abs(IIR_filt_sin1 - IIR_sin1_out)) / length(n)) * 100
182 FIR_sin1_comp = (sum(abs(FIR_filt_sin1 - FIR_sin1_out)) / length(n)) * 100
183
184 IIR_sin2_comp = (sum(abs(IIR_filt_sin2 - IIR_sin2_out)) / length(n)) * 100
185 FIR_sin2_comp = (sum(abs(FIR_filt_sin2 - FIR_sin2_out)) / length(n)) * 100
```