The University of Queensland
School of Information Technology and Electrical Engineering

ELEC3004/7312: Signals, Systems and Controls

EXPERIMENT 4: IIR FILTERING ON THE NEXYS 2

Aims

In this laboratory session you will:

1. Gain familiarity with designing infinite impulse response (IIR) filters in Matlab;
2. Experiment with and compare a number of different filter realisation structures;
3. Appreciate the use of VHDL for implementing practical **fixed-point** digital filters.

Introduction

Figure 1 shows the canonical (direct form II) implementation of an IIR second order (biquadratic) digital filter. The canonical form of a digital filter is often preferred as it is the realisation which requires the minimum memory to implement a particular transfer function. As we have seen for digital filters the *z*-transform defines the link between the transfer function and the linear difference equation giving, y[n], as a function of present and past input and output samples. The difference equation is important in the sense that it illustrates the fact that the **present output, y[n]**, of a linear discrete system is a linear combination of the weighted present and **N** past **inputs** to the system as well as **M** weighted past **outputs**. Specifically, for a biquadratic filter N = M = 2.

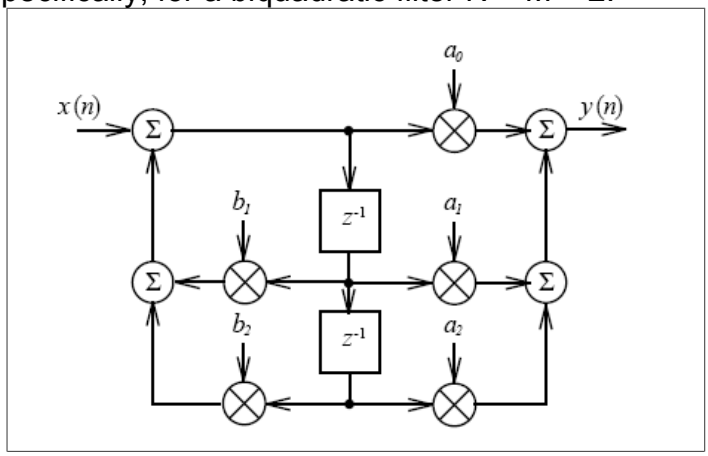

Figure 1: Biquadratic digital filter section, Direct form II (canonical).

In principle, any high-order digital filter design can be synthesised as either a series cascade or parallel connection of second order sections. The series cascade, as shown in Figure 2, is often favoured because:

1. Many second order sections have simple fixed-point (integer) coefficients;
2. Most commercial software design packages assume a cascaded design and more importantly;
3. If each biquadratic section is stable and free from limit cycles then the overall cascaded filter will also be stable and free from limit cycles.

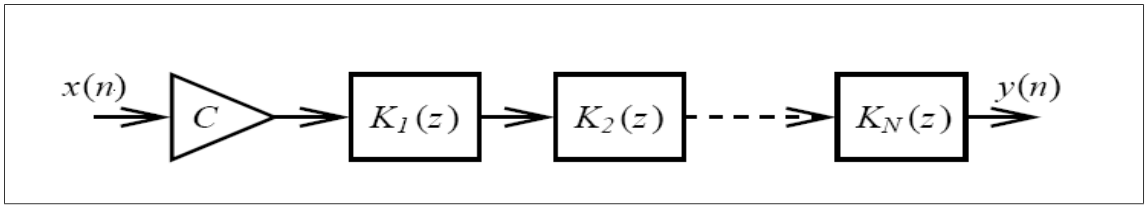

Figure 2: Series cascade of lower order sections.

Equipment

1. PC with Xilinx ISE, Digilent Adept & Matlab;
2. Nexys2 with USB to JTAG cable, PMOD AD1 and DA2 boards plus 2 PMOD CON4 boards
3. DAC712 and ADC712 filter boards
4. Oscilloscope and signal generator;
5. 2 x cables: Mono RCA male to BNC male
6. Powered External speakers + audio jack cable;
7. 1 BNC T-adaptor M to 2F;
8. 1 x cable BNC M to BNC M, 0.5 - 1 metre long.

Preparation

Note: preparation will be checked and marked at the start of each laboratory class.

Except in Question 2, the symbol π below here does not mean the value 3.1415... It refers to a NORMALISED FREQUENCY where 2π is your sampling frequency Fs, and π is your Nyquist frequency, Fs/2.

Answer the following questions:

1. Are we using π to represent 3.1415, or are we referring to the Nyquist frequency, (Fs/2)?
2. Derive a difference equation whose transfer function has zeros at exp(±jπ /4) and poles at ±0.9j. Hint: the locations of poles/zeros define the roots of the transfer function that defines the filter difference equation;
3. Next, we will implement the 3rd order Butterworth IIR low-pass filter from Haykin & Van Veen's (Library ref TK5102.5 .H37 2003) Example 8.7, pp. 647–649 & 657, in fixed-point arithmetic on the Nexys2. This filter has a normalised 3dB cut-off frequency of 0.2π and can be designed by hand using the bilinear transform or in MATLAB using the following commands:

```
[b,a] = butter(3,0.2); % design 3rd order Butterworth with cut-off 0.2π
[H,w] = freqz(b,a,512); % calculate frequency response H, 512 values
mag = 20*log10(abs(H)); % calculate magnitude in dB
phi = angle(H); % calculate phase in degrees
phi = (180/pi)*phi; % convert phase to radians
plotyy(w,mag,w,phi) % Plot magnitude and phase on one graph
```

Note: MATLAB will display floating point values for the filter coefficients.

4. When implemented as a filter using the Nexys2, what will be the cut-off frequency of this filter, in Hz, if the sampling frequency is 25 kHz? **You will use this in Part 1(b)**
5. Draw a block diagram of the filter in Direct Form II, i.e., calculate the coefficient values for the general biquadratic digital filter section shown in Figure 1.
6. Modify your design to implement the filter in cascaded form with Direct Form II sections, as per Figure 2. The cascade form is most simply designed in MATLAB using **butter** and **zp2sos** (zero-pole-gain to second-order sections). Alternatively, you can use MATLAB's **fdatool**: by selecting a **Lowpass, IIR, Butterworth, of order 3** with the required sampling and cut-off frequencies. Once the filter is designed you can **Store Filter** and then analyse it with **fvtool** using the **Filter Manager**;
7. Now, without changing the cut-off frequency, increase the order of the filter to 6. Can this be done by cascading 3 of the above 2nd order sections? If so, do it and compare this approach to designing the 6th order Butterworth filter directly using MATLAB.

FIXED POINT MATHS:

The Nexys 2 does not have a convenient or time-efficient method of generating floating point numbers. Therefore, in this experiment we will utilise a **fixed-point (integer)** approximation for each of the coefficients in the biquadratic filter.

One of the simplest ways of implementing integer coefficients is through the use of ratios. For example, if the value 0.3 is required as a coefficient value, this is 3/10 or 300/1000, or even 351/1170. The Nexys 2 is limited to using binary denominators in the ratios due to the type of hardware implementation available, which uses a binary "Right Shift" to divide. That is, division by 64 is 6 right shifts, sometimes expressed as >>6.

For our example of 0.3, we might choose 1024 (i.e., 2^{10} or 10 bits) as the denominator. The numerator is then the nearest integer result of (0.3 * the denominator), so the exact result is $0.3 \times 1024 = 307.2$, which when rounded to the nearest integer becomes 307. Thus 0.3 is represented as 307/1024. In this example the error is quite small at 0.2/1024 = 0.0195%. However, even smaller errors can be achieved by using larger denominators (and consequently larger numerators). This requires striking a balance between accuracy and the potential for overflows to occur in the multiply-accumulate calculations.

Because we do not have direct access to the hardware registers in this implementation, an overflow can occur. However, we can add some VHDL to test for an overflow condition and choose whether to allow the overflow, or saturate the signal at either the maximum or minimum value available.

The best approach is to design the system so as not to overflow in the first place. There are three sources of overflow:

1. The input signal (being too large);
2. Numerator summation overflow, e.g. (a/x +b/x +c/x) is > 1
3. Denominator division overflow (Remember division by zero errors? Well, choosing too small a value for the denominator will cause similar problems).

The VHDL you will use has been tested for various combinations of numerator and denominator to find the limits of overflow in the implementation.

Part 1 of this experiment implements this 3rd order filter using a cascade of a second and first order section. Using the MATLAB fdatool you will obtain coefficient values in a similar form to the ones in the image:

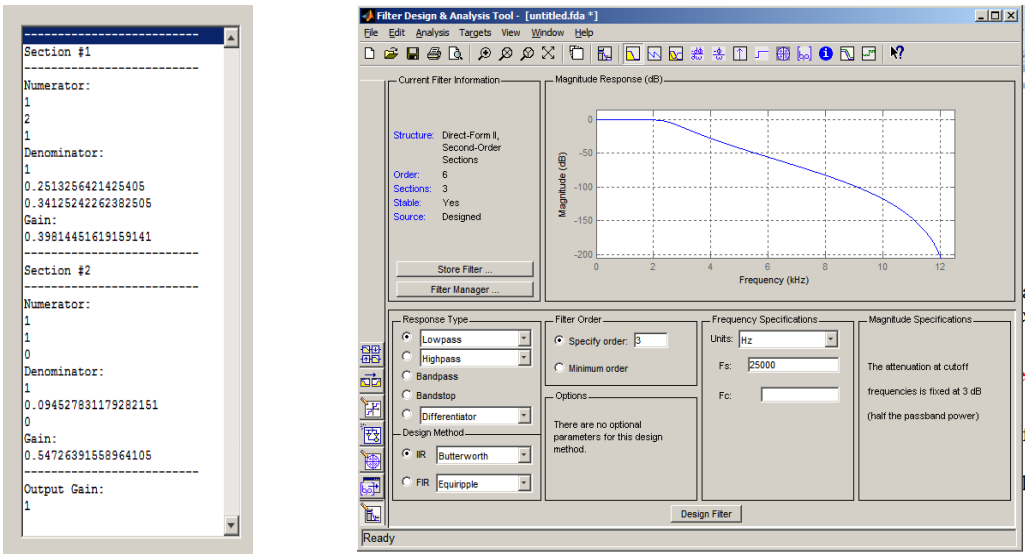

Figure 3. fdatool coefficients output

You can see from figure 3, that the Numerators for Section #1 are **1 2 1.** The values you will need to enter in the vhdl modules will depend on the value of the denominator chosen.

For this example, we have chosen to use 131072 (2^{17}). This means that for Numerators of **1 2 1**, we get a_0 = **1** × denominator, a_1 = **2** × denominator, a_2 =**1** × denominator, resulting in values of 131072, 262144 and 131072 respectively for a_0, a_1 and a_2. The values for b_0, b_1 and b_2 will simply be those shown in the denominator section of fdatool, again multiplied by 131072.

For the example above, the value of 0.2513256421425405 is multiplied by 131072 and converted to an integer, which results in a value of 32942. In MATLAB, if you multiply a floating-point number by an integer you will usually get a floating-point result, which we convert to integer with this command:

int32(floating-point number * integer)

This process is repeated for the Second filter section.

Procedure

Part 1: 3RD ORDER Filter using cascaded 1st and 2nd order sections

Ensure you have the following files (downloadable from the Course website) stored in a local folder:

AD1RefComp.sym, AD1RefComp.vhd
DA2RefComp.sym, DA2RefComp.vhd
DAC_CTRL.sym, DAC_CTRL.vhd
Echo_IIR_1.sym, Echo_IIR_2_B.sym
clockdiv1.sym, clockdiv_1.vhd
ELEC3004top.ucf
IIR_1_Filter_a.vhd
IIR_2_Filter_b.vhd
Prac4_Part1a_2012.xise
Prac4_Part1a_2012.ise
Prac4_Part1a_2012.sch

Part 1a: Confirm Module Operation: - Tasks and questions

The supplied modules contain initial values that allow you to confirm that output = input.

1. Start Xilinx ISE
2. Start Digilent Adept and check that you can connect to the Nexys2 board
3. Use **File** | **Open Project** and load **Prac4_Part1a_2012.xise**
4. Double-click the schematic file **Prac4_Part1a_2012.sch**. You will see a diagram like this

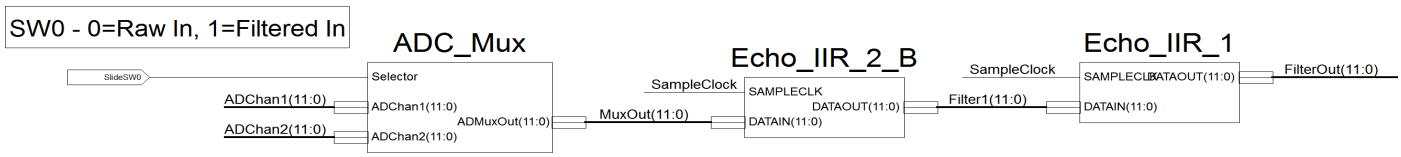

Figure 4: Filter control schematic.

5. Connect the signal generator to CH1 of the oscilloscope and to the AD1 input of the Nexys2, and CH2 to the DAC output. **If the filter boards are available**, set SW0 on the Nexys2 to off (Raw In), but **use Filter Out on the DAC712 Filter board.** Use **Generate Programming File** and download to the Nexys 2 board using Digilent Adept.
6. As per previous labs, using a Sinewave, choose a DC offset value of about 2.5 V and a maximum pk-pk value of just under 5V. Adjust it so that you have no distortion in the output.
7. Sweep the frequency from about 100 Hz up to about 10 kHz and **confirm that the output signal is basically the same as your input signal.** Now sweep from 10 kHz to about 31 kHz and observe the effects of aliasing.
8. **Set SW0 to On (Filtered In)**. Repeat 7 and observe the effect of the input filter upon aliasing

Using the modules as filters

There are two modules named **IIR_1_Filter_a.vhd** and **IIR_2_Filter_b.vhd.**

In each file you will see a section that looks like:

```
constant Denom: integer range 0 to 16777216:= 131072;
```

You will also see a section like this:

```
ECHO:process (SAMPLECLK)

   begin

a0 <= 524288/k;  -- k =4, so 524288/k = 131072
a1 <= 0/k;
a2 <= 0/k;
b0 <= 0/k;
b1 <= 0/k;
b2 <= 0/k;
```

The values you see here are DUMMY values which are used to confirm that the hardware works. You will need to enter the correct values to act as a filter. K can also be changed

This is where you enter the coefficient values. Referring to the values shown in the Appendix, the top row shows the values for a_0, a_1, a_2, then b_0, b_1, b_2 respectively. For example;

The DUMMY filter coefficients:

1 2 1 1 -0.41885608448176648 0.35544676217239063

1 1 0 1 -0.15838444032453636 0

when multiplied by 131072 and then converted from floating point to integer become

131072 262144 131072 131072 -54900 46589

131072 131072 0 131072 -20760 0

For the 1st order filter this would become:

```
a0 <=131072/k;
a1 <= 131072/k;
b0 <= 131072/k;
b1 <= -20760/k;
```

For the 1st order filter, you will see text like this:

```
Yreg(0) <= (a0*Xreg(0)/Denom + a1*Xreg(1)/Denom) - (b1*Yreg(1))/Denom; --calculate difference equation

Yreg(1) <= Yreg(0); --shift data

Xreg(1) <= Xreg(0);
```

For the 2nd order filter, you will see:

```
Yreg(0) <= (a0*Xreg(0)/Denom + a2*Xreg(2)/Denom + a1*Xreg(1)/Denom) - (b1*Yreg(1)/Denom +
b2*Yreg(2)/Denom);
 Yreg(2) <= Yreg(1);      -- shift data
 Yreg(1) <= Yreg(0);
 Xreg(2) <= Xreg(1);
 Xreg(1) <= Xreg(0);
```

The value of k in each VHDL module has been chosen to limit possible numerical overflow, which was mentioned before. If you look at the schematic above in Figure 4, you will notice that IIR_2_Filter_b appears first in the sequence of filters. This is because the results from fdatool show a 2nd order section as Section 1, followed by a 1st order section. Depending on the cut-off frequency (f_c) and sampling frequency (f_s) the order of the sections may change.

Part 1b: Implement Real Filter Coefficients: - Tasks and questions

If you have the filter boards:
Use Filter In (SW0 = 1) and Filter Out (the bottom connector on the PMOD CON4 connector)

Please Note, the symbol π below here does not mean the value 3.1415… It refers to a normalised frequency where 2π is your sampling frequency Fs, and π is your Nyquist frequency.

*See the Appendix for hints on using the **fdatool** output*

To implement **YOUR** filter system, use **fdatool** and enter the values for Fs = 25000 and a normalised cut-off frequency Fc = 0.2π i.e. normalised to Fs = 2π and **calculated as part of the preparation** and **"Specify Order" = 3**. Next follow the instructions from the previous section to enter the (scaled) integer coefficient values into their respective files.

Save the updated filter files and then follow the instructions below.

9. Using the methods shown immediately above, get suitable coefficients from **fdatool**, copy them into the respective VHDL files and ensure the correct value for Denom has been chosen.
10. You should now have a filtered version of the input signal being sent to the audio out port of the Nexys2. You may look at this with your oscilloscope to verify that it is working correctly. To see the VHDL script used to generate the outputs, double-click on any of the VHD files.
11. Follow the instructions immediately below here:

Using the signal generator set for Sinewave, **adjust the offset to about 2.5 V DC, then the amplitude to about 2.6V pk-pk.** Make any adjustments required so you don't see distortion in the output and set frequency knob to minimum.

Select the 10 kHz range on the Function Generator then sweep the range from about 100 Hz up to 25 KHz and observe the output.

1. Now using spot frequencies of 0.125, 0.25, 0.5, 1, 2, 4, 6, 8, 10, and 12 kHz, sketch the magnitude response with the frequency axis scaled in kHz and measure the (3dB) cut-off frequency of the filter.

   Use the output magnitude at 250Hz as your 0dB reference point.

   You may not get a smooth response, so feel free to plot the response and show your tutor where you think the best -3dB point is;

2. Does the magnitude response match that shown in Matlab? If not, why not? Hint: think of the effect on the measured frequency response caused by the limited precision of using fixed-point calculations.

3. Repeat 1 using a squarewave. Make sure to adjust the input signal so that you don't get a distorted output at 4kHz.

Part 2: 6th order filters using 3 × 2nd order sections

(2a) Implement a 6th order filter using 3 × 2nd order sections with the **SAME coefficients**

Open the **Prac4_Part2a_2012** ISE project file. Use fdatool as before but set "**Specify Order" = 2**.

Open IIR_2_B1.vhd, IIR_2_B2.vhd, and IIR_2_B3.vhd in turn and enter the **SAME** coefficients and set **k=4**, then save the files. Compile and Program FPGA as before. **Note: the coefficients for this 6th order cascade filter are the same for each of the sections.**

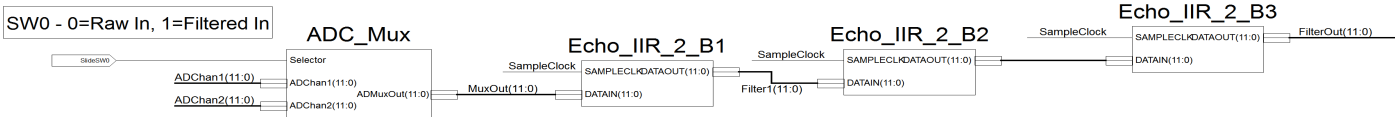

Figure 5. Schematic with all coefficients of the same type.

1 Now using spot frequencies of 0.125, 0.25, 0.5, 1, 2, 4, 6, 8, 10, and 12 kHz, sketch the magnitude response with the frequency axis scaled in kHz and measure the (3dB) cut-off frequency of the filter.

Use the output magnitude at 250Hz as your 0dB reference point.

You may not get a smooth response, so feel free to plot the response and show your tutor where you think the best -3dB point is;

2 Do you notice any changes in the frequency response from the desired one? Is the cut-off frequency the same? Can you explain any differences?

(2b) Implement a 6th order filter using 3 × 2nd order sections with **DIFFERENT coefficients.**

Repeat the above using fdatool to design the cascaded 6th order filter from Question 7. of the preparation.

Set **Specify Order = 6**. Open IIR_2_B1.vhd, IIR_2_B2.vhd, and IIR_2_B3.vhd in turn and enter the **DIFFERENT** coefficients and set **k=4**, then save the files. Compile and Program FPGA as before

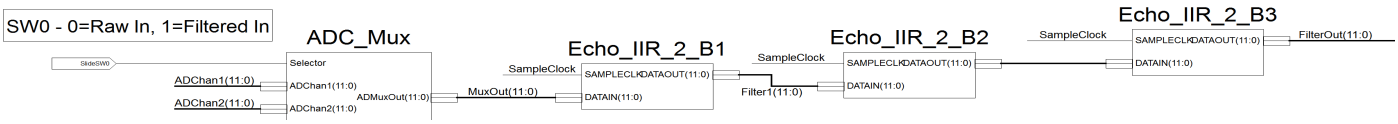

Figure 6. 6th order filter as 3 second order sections.

1 Now using spot frequencies of 0.125, 0.25, 0.5, 1, 2, 4, 6, 8, 10, and 12 kHz, sketch the magnitude response with the frequency axis scaled in kHz and measure the (3dB) cut-off frequency of the filter.

Use the magnitude at 250Hz as your 0dB reference point.

You may not get a smooth response, so feel free to plot the response and show your tutor where you think the best -3dB point is;

2 Do you notice any changes in the frequency response between part 2a and 2b? Is the cut-off frequency the same? Can you explain any differences?

Part 3 Challenge Questions

Challenge 1: Go back to Part1 and try swapping the order of the two filters. Does the filter still work properly? If so, is the frequency response similar to your previous recorded data?

Challenge 2: Try changing the values of k in each of the modules. Divide k by 2 in the first filter module. Do you notice any distortion (overflow) in the output? If not, repeat it for the second filter. Can you explain why the distortion occurs at the physical peak of the output sine wave and only at certain frequencies?

Appendix: Hints for conversion of coefficient values to scaled integers

To simplify the conversion of results from the fdatool use the following process after you have got the coefficients:

> With the **fdatool** window showing, select **File | Export** and **Export to WORKSPACE** using the "**Export as Coefficients**" format. Select **Overwrite Variables** option.

Go the MATLAB workspace and **double click on SOS** so that you see something like these dummy values:

1 2 1 1 -0.753537298108023 0.406307607549862

1 1 0 1 -0.290526856731916 0

Now convert to integer using: int32(SOS*131072) and you will see :

ans =

131072	262144	131072	131072	-98768	53256
131072	131072	0	131072	-38080	0

Note: These are dummy values. You need to run **fdatool** with the value of f_c specified in the experiment notes above.

These values can now be copied and pasted into the respective VHDL modules.