EECS 452 Lab 5:
IIR filters and Timer Interrupts

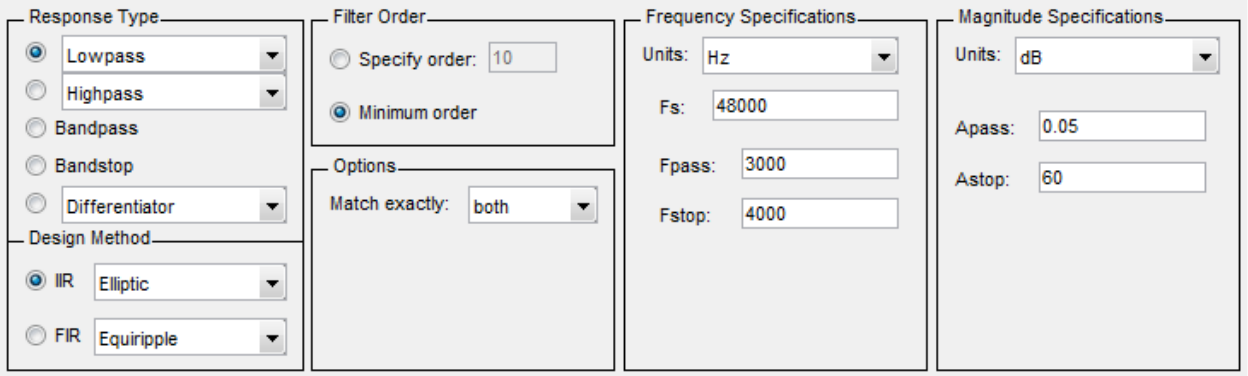
In this lab we will implement IIR filters on both the FPGA and the C5515 processor. In addition you will get to work with some code that uses the timer interrupts on the C5515 processor.

1. Pre-lab

We have a number of different things to cover in the pre-lab. We'll start with a TI-supplied FIR filter. We'll then move onto designing an IIR filter using biquad sections in C. Finally, we'll look at one of TI's IIR filter function.

IIR filters in MATLAB

Using Matlab's fdatool, design the low-pass Elliptic IIR filter shown in the figure below.



Q1. Answer the following questions about the filter

a. What is the order of your filter?
b. How many biquad sections are in your filter?
c. Look at the group delay.
   i. Is it constant?
   ii. In terms of *time* what is the largest the group delay is in the passband?
   iii. In terms of *time* what is the smallest the group delay is in the passband?
   iv. If you had an FIR filter of the same order, what would be the answer to ii & iii?
   v. Is this filter linear-phase in the passband? Explain your answer.

The default structure in MATLAB is direct-form II, SOS (second order sections, also called biquad sections).

Q2. Examine coefficients of the direct-form II SOS filter. (You only need to examine the matrix SOS and ignore G in Matlab)

a. What is the largest absolute value found among these coefficients?

b. Examine the pole/zero plot. Explain how you know that all coefficients (both numerator and denominator) are certainly within the range of -2 to 2. Hint: we proved this for the denominator in class based upon an assumption that happens to be true for the numerator in this case too.
c. Convert the filter out of SOS form and into a single section (in the Edit menu). Discuss the range of (absolute) values now seen for the coefficients.
d. Is it easier to work with the SOS coefficients in fixed point or the single-section coefficients in fixed point? Why?

Q3. Write a MATLAB script to read in SOS matrix, normalize the coefficients for unit DC gain (see below) and prints them in Q15 format so that you can paste them to your verilog and C code. Each row of the SOS matrix contains the coefficients for a second order section.
The coefficients are ordered: b0 b1 b2 a0 a1 a2
In your script first determine the 0 Hz gain for each biquad section and then normalize the numerator values so that each section has unit gain at 0 Hz. Again you can ignore the variable G exported by Fdatool.
To normalize the DC gain of a section to unity multiply the b values for that section by

$$(a0 + a1 + a2) / (b0 + b1 + b2)$$

Q4. Now let us look at the impulse response for a bit. Export the filter to the Workspace as Objects. You can get the impulse response by using the impz function. The syntax would be **`impz(Hd)`** assuming your filter was exported as "Hd". That command causes a graph to pop up, but it also returns some values. Try "**`help impz`**".
a. What is the sum of the absolute values of the impulse response? What does that tell you of interest? Explain why.
b. If you wanted to input values which maximized the output value, what would be the last 25 values you would input (in order from last in to 25th to last)? Assume inputs must be between 1 and -1 (inclusive).

Hopefully answering the above questions provided some insight into the filter itself (and reminded you what you know about filters). Now let's move forward toward designing our filter.

Q5. Print the pole/zero plot.
a. Label which poles should be paired with which zeros.
b. Label each zero/pole pair with a single letter name (A, B, etc.).
c. According to the ordering heuristic we are using for biquad sections, list the order we'd put the biquad section in from closest to the input to closest to the output.
d. Using the same scheme as above, list the order MATLAB put the biquad sections in.

Save your work: we will want it for the in-lab portion portion..

Q6. Go back to lab 2 and look at how we implemented the FIR filter in lab 2 Part 3.
a. Before sum gets shifted and returned what is the range of representation for sum?
b. What is the range of representation for the return value?
c. Modify the program from lab2 part 3 so that if sum's value is out-of-range for the return value, we saturate the return value. Think about how overflow was handled in lab 4. Do this with simple > and < operators rather than bit selection…

Q7. Think about the issues associated with implementing a filter with an arbitrary number of biquad sections. Describe, in your own words, how to implement an SOS-based IIR filter. How it would likely be different if you are to use only 1 single-section, i.e., the coefficients you obtained in Q2(c)?

And one of TI's IIR filters

Now go back to TI's DSP library for the C55x series (C5515_DSP_Lib_guide.pdf.) Consider the function **`iircas5()`**.

Q8. In your own words, explain the role of each of the arguments to **`iircas5()`**.

Q9. How does the implementation handle internal overflow protection?

Interrupts

When filtering, we are actually wasting a large chunk of our CPU time waiting for new samples. Basically, we do the filtering and then wait until the next sample arrives. That's fine as long as we don't have anything else to do. But we could better utilize our processor if we could go off and work on something else but get notified when a new sample arrives. Luckily for us there is a way to do this: interrupts.

Background

An interrupt is an event that causes the processor to stop what it is doing and handle some other activity. Imagine you had a bunch of studying to do while working at a library desk. You would do your studying until someone came by with a question when you would stop studying and answer their question. Once done, you would go back to where you left off.

We can make it so that when a given event occurs the processor stops whatever it is doing and jumps to a function. There is a table called the Interrupt Vector Pointer Table which has an entry for each possible interrupt source (say a timer or data arriving or something else). You can put an address in that table and the function at that address will be called when the triggering event occurs. Details about this can be found in chapter 1.6 of the C5515 DSP System User's Guide. For this lab, we will use only the general timer. However later we'll want to use the i2s interrupts (i2s is a serial bus and is used to talk to the audio CODEC chip).

There are some peculiarities with both of these interrupts. The timer interrupt has only one enable and one flag bit but there are three general purpose timers. So the interrupt flag gets thrown when any one of these timers is would throw one. How do you which timer threw the interrupt? Well there exists a register called the Timer Interrupt Aggregated Flag Register (TIAFR) which uses its three least significant bits to indicate which timer needs servicing. The timer interrupt flag bit in the interrupt flag register is simply the result of those three bits OR'ed together.

The i2s needs a third register to be set-up properly to tell the processor that it is throwing the interrupt and not the Multi-Media-Card ports. This would be the external bus selection register, whose description can be found in the DSP System Guide in chapter 1.7.3.1.

Read the file "interrupts.c" and "usbstk5515_interrupts.h" from the course website in lab5_files.zip. Look over SPRUFO2 (DSP Timer/Watchdog Timer User's Guide) and the parts of C5515 DSP System User's Guide referenced above.

Q10. Answer the following questions about the following special-purpose registers used in the code

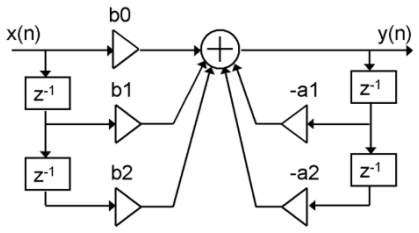
a. TCR0 is the timer control register for timer 0. What do the constants associated with "TIME_STOP" and "TIME_START_AUTOLOAD" do? Explain.

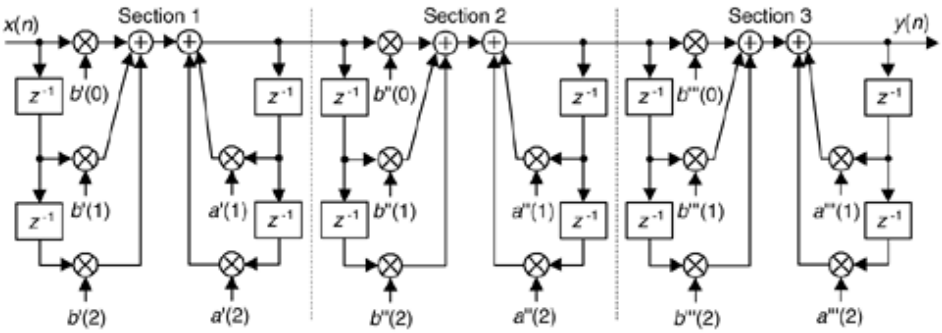
b. TCR0_1 isn't defined in the SPRUFO2 document under that name. Looking over the code and that document, what is the name of that register in the document? What is its role?

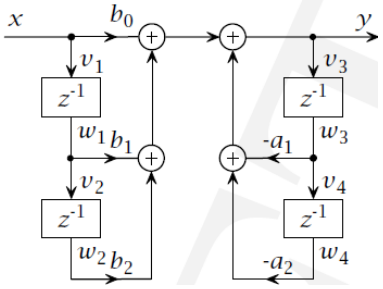
2. In-lab

Writing an IIR filter on C5515 and FPGA

In this section you will be writing two IIR filters. First, you will finish a direct-form 1 biquad filter on C5515. Two equivalent block diagrams are shown on the right and the cascaded form is shown below[1].







Start Code Composer Studio. On the course website (in Lab5_Files.zip) there is a file called "IIRmain.c". Create a new project using the standard starting point and add this file. Fill in the missing code (generally indicated with "**TODO**"). Think carefully before you start coding. Read the comments and think about what the filter is supposed to be doing! The coefficients are in the order b0, b1, b2, a0, a1, a2 and using coefficients given in the file, the filter is supposed to be a low-pass filter with Fpass of 6 KHz. Fill in the blanks in the code and make sure the filter functions correctly.

Use the Matlab script you wrote in Q3 of pre-lab to normalize and print the coefficients you designed in Q1 of the pre-lab.

Q1. What should **`biq_32`** be initialized to if you were to build the filter in Q1 of the pre-lab?

G1. Demonstrate your filter with the coefficients you found for Q1 of the in-lab.

Q2. How long does it take your IIR filter to process a single input? Be sure the compiler is fully optimizing (-O3).

Now create a filter made of transposed direct-form II biquad sections.

Note: you need to create a new function and a new buffer for the transposed filter but use the same coefficients as you used for G1.

G2. Demonstrate your transposed direct-form II SOS filter.

Q3. How long does it take your transposed IIR filter to process a single input?

[1] Diagrams are taken from (starting from top right) Wikipedia, Dr. Metzger, and flyby.com.

Using library IIR functions

In the pre-lab assignment you reviewed TI's **`iircas5`** library function. Now we will implement the low-pass filter from G2. The Dsplib.h header file defines the interfaces for those library functions, but the implementations are already compiled into binary in the 55xdspx_LARGE.lib that your linker finds during compilation. The library functions are implemented in TI's assembly language.

Create a new project from Starting_point and copy over the IIR coefficients from **G2**. Notice that these coefficients cannot all fit in a 16-bit Q15 notation. Find the filter coefficient sets (b0, b1, b2…) where this overflow occurs in any of the biquads and for that particular coefficient set, scale the values so that they fit into a 16 bit Q15 notation(scale a1 and b1 to fit in Q15 representation).

Now that you've scaled certain coefficient sets, your filter will have a different transfer function or not work unless you change the implementation of the **`iircas`** function. Find the **`iircasNew.asm`** in the Lab5_Files.zip and add it to your project. The assembly file is in your project so it will get priority from the linker over TI's implementation during compile time. We provide the modified iircas function in iircasNew.asm to accommodate the scaling you have done to coefficients b1 and a1.

Implement a main function where you filter AIC audio data using the **`iircas5`** function. Be sure to scale the input signal down by 32 before you input it to the iircas5New function and scale the output up by 32 before you output to the AIC_write function.

G3. Show your working IIR filter and code to your GSI.

FPGA-based IIR

We suggest you build a biquad section as a single module that takes the input, output and coefficients as parameters and then connect all the biquad sections. Implement the filter you designed in Q1 of the pre-lab. The FIR filter you implemented in lab 4 is a good starting point. Be careful with the coefficient array and think about what representation you want to use since you have more control on the FPGA board (things don't have to be 16 bit).

Note you may ***not*** use a for-loop as you did in the C code. Verilog doesn't allow you to pass in arrays into module so you have to pass in the coefficients individually.

After you've implemented the IIR filter, open up Code Composer Studio and program the C5515 with the TransferFunctionMeasurement.out program (Run->Load Program). Connect a cable from the USB stick stereo out right channel to the DE2-70 FPGA board. Connect the USB stick stereo output left channel to the USB stick stereo in left channel. Then connect the output from the DE2-70 to the USB stick stereo in right channel. Launch the WindowPlot.exe and run the C5515 program.

G4. Demonstrate your transposed direct-form II SOS filter on FPGA. Show that the filter has the correct cut-off frequency. Show your GSI the transfer function measurement and the attenuation.

Q4. What is the worst case delay for your FPGA implementation? How does it compare to your design on C5515?

Interrupts

Create a new project using our standard starting point and include the file "interrupts.c" and "usbstk5515_interrupts.h" from the class file server. Run the program.

Q5. At a high-level, what is going on in this code? Specifically explain when **`Timer_Handler()`** is being called and what that function is doing. How do calls to **`Timer_Handler()`** result in the LEDs flashing?

Q6. What's the pre-scalar value located in TCR0_0? What does it mean?

Now change the program so that the red LED toggles at about 2Hz and the yellow LED toggles at 0.5Hz while the other two LEDs are left off. **Do this by setting one of the LEDs in the interrupt function and the other in the main program**.

G5. Demonstrate your blinking lights. Show your GSI the changes you made to your code.

3. Post-lab

Q7. Go back to the filter coefficients you used for the **G2**. Use FDATool to design that filter again in Matlab. Write a Matlab script to plot out the gain vs. frequency for each delay stage. Use the Matlab DSP library function **`[H,F] = FREQZ(B,A,N,Fs)`** to help you. Plot out the gain vs. frequency for each delay stage. Be sure to set your axis appropriately. Turn in your code and plots.