Lab #6 A Transfer Function Generator

1. Description

In this laboratory, we implement a real-time system that outputs the result of the response of a Butterworth filter. We learned about how to approximate continuous systems to discrete systems using the difference equation, along with using the ADC and DAC system. We dealt with a SISO LTI dynamic system and we used electrical signals as our inputs and outputs. The figure below describes the functional hierarchy of this laboratory.

```
main()
|_ myRio_Open()
|_ Irq_RegisterTimerIrq()
|_ pthread_create()
    |_Timer_Irq_Thread()
        |_ Irq_Wait()
        |_ NiFpga_UWrite32()
        |_ NiFpga_WriteBool()
        |_ Aio_Read()
        |_ Aio_Write()
        |_ cascade()
        |_ Irq_Acknowledge()
        |_ pthread_exit()
|_ printf_lcd()
|_ getkey()
|_ pthread_join()
|_ Irq_UnregisterDiIrq()
```

Just like the previous lab, this project involves running two threads: main() and the ISR that reads the analog input, calculates, and generates the output. A function generator is used to analyze the input and the output signals created. The main() thread handles the creation and the termination of the ISR, and it loops through while doing nothing in order to let the ISR run, and finally exits the loop and terminates if the backspace key is pressed in the keypad.

The filter we implement to process the input signal is a Butterworth Filter, which has a optimally flat 0dB region. The transfer function of the filter is shown in the figure below.

$$T(s) = \frac{V_{out}(s)}{V_{in}(s)} = \frac{\omega_n^3}{(s+\omega_n)(s^2+2\zeta\omega_n s+\omega_n^2)}$$

Figure 1: Butterworth Filter

This continuous system is approximated as a discrete system by breaking the Nth order system into a series of n_s 2nd order systems called the Biquad

Cascade. It works by using the output of the previous Biquad cascade as the new input for the next one. The equation of the Biquad structure is described below. All the coefficients of this structure is defined in the ISR thread.

$$\begin{aligned} y_i(n) \;=\; & [\ b_{0_i} x_i(n) + \\ & + b_{1_i} x_i(n-1) + b_{2_i} x_i(n-2) + \\ & - a_{1_i} y_i(n-1) - a_{2_i} y_i(n-2)\ ]/a_{0_i} \end{aligned}$$

Figure 2: 2nd order Biquad Structure

The output of this equation is calculated in the cascade() function which is called in the ISR thread. If the output of this equation exceeds the minimum or maximum output limit, it is saturated to a predefined value.

When the filter is active, the main() thread causes the LCD screen to print ”Lab 6 ...” and after the DEL key is pressed, the ISR is terminated along with the main thread and ”Good bye!” is printed to the LCD screen.

2. Testing

State precisely the complete procedure for testing the program. The tests should not be unnecessarily extensive, but should be adequate to confirm that all major functions perform correctly. A code tester will attempt to follow your test procedure exactly.

To test the program, run main.c, and follow the sequence below:

1. Uncomment the section in the program that sends the output signal (marked by the comment:
debugging and comment the MATLAB plotting sections labeled by (”MATLAB Data”)

2. After running main.c, the LCD screen should print ”Lab 6 ...”

3. Observe the function generator. Adjust the voltage increment resolution and the time resolution to get a clear view of A1 and A2. Set the function generator to output a square wave with Vpp = 5 V and frequency = 8 Hz. You should see two signals; one square wave, and a repeating second order step response (output).

4. If the previous step does not work, try changing the VADout here Aio_Write(CO0, VADout) to -5V, and you should see that A1 shows -5V. If it still fails, most commong problem is the wrong setup of the function generator, so try fixing that.

5. Once the appropriate shapes of A1 and A2 is observed, change the function input to a sinusoidal wave of 8 Hz and 5 V.

6. Increase the frequency, the output signal should start attenuating at 35 Hz.

7. Press the backspace once everything is validated, the screen should print "Good Bye!" and the program stops.

8. Once this is confirmed, we can proceed to importing the data to MATLAB.

Once the oscilloscope shows the correct reading, we can uncomment the sections marked by "//MATLAB Data", then we can repeat step 1 through 8 without checking the oscilloscope to gather data.

3. Results

While performing testing for this lab, the first error I encountered was that I did not see A1 on the oscilloscope. After I decreased the voltage resolution, I noticed that A1 is available, however it was greatly attenuated. The reason for this attenuation is due to my printf_lcd() statement before the Aio_Write() command. This statement causes the Aio_Write() function to be hugely delayed, which results in the attenuation of the output signal. I found the cause of this bug by using the debugger.

The other bug I experienced multiple times is segmentation fault. There are a variety of reasons for this fault but mainly it boils down to: missing a semicolon, not initializing a pointer, passing in the wrong data type, and defining variables in the wrong place.

After fixing the bugs, the filter works as intended. For frequencies between 0 Hz to 35 Hz, the output signal has the same magnitude as the input signal and it is greatly attenuated afterwards. This is the behavior of a Butterworth filter.

Step Response

I imported the input and output of the system for the square wave input with 5 V amplitude and 8 Hz frequency. The measured data is plotted on top of the simulated data to compare the performance of the digital vs. continuous system. The result is described in the figure below. The figure shows that the behavior of the filter is very similar to the theoretical behavior for a continuous system. Therefore, using the Interrupt Service Routine and implementing the Biquad approximation is a great method to perform real time control.

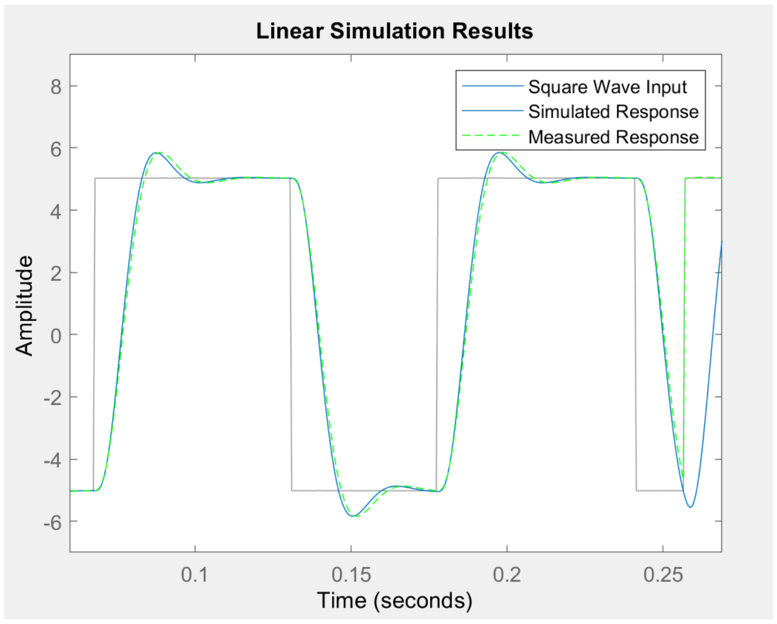

Figure 3: Square Wave Response

Frequency Response

The next part of the project is to measure the magnitude and phase shift of the transfer function at frequencies of [5, 10, 20, 40, 60, 100, 140, 200] Hz. The figure below shows the data points superposed on the theoretical Bode Plot of both the magnitude and phase of the transfer function. The magnitudes of the measured attenuation is very close with the theoretical Bode Plot, with slightly decreasing accuracy towards higher frequencies. This phenomenon occurs because closer to higher frequencies, the measured signal is getting closer to be aliased, which may cause the peak modulated signal to be sampled incorrectly. This also explains the fact that the amplitude of the modulated output signal at high frequencies fluctuate.

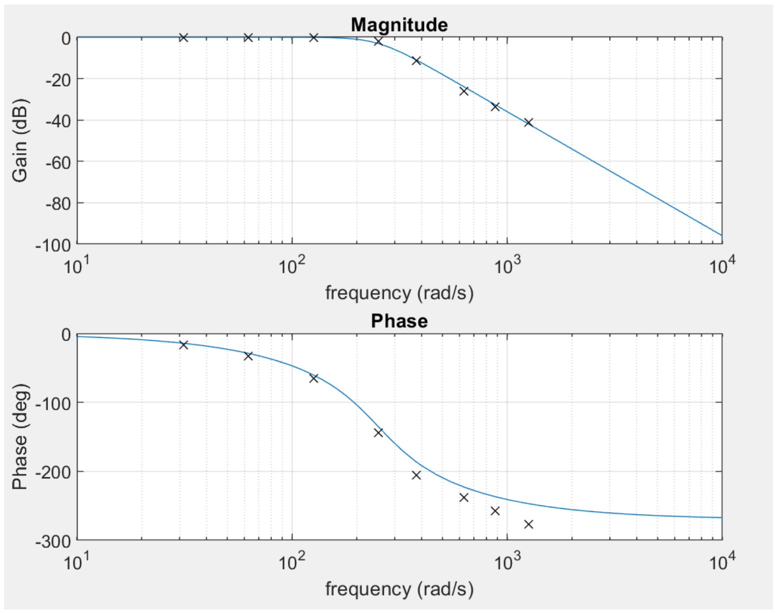

Figure 4: Bode Plot

Furthermore, the phase shift of the transfer function also lies closely to the theoretical phase plot, but with more inaccuracies towards higher frequency. One explanation of this behavior is due to the fact that it is hard to exactly pinpoint the peak of each signal, especially at higher frequencies where the time resolution is very small, which causes higher error. We can also see that the phase shift is always higher (more negative) than the predicted value, which can be explained by the fact that the communication of myRio and the C code introduces transport lag of the signals, which increases the phase shift of the output signal.