SJSU EE138
Introduction to Embedded Control System Design
Lab 4: Digital Filter and Motor Speed Measurement

Readings:

Atmel SAMD20 DataSheet
- Section 27: TC – Timer/Counter
- Section 19: EIC - External Interrupt Controller

Lecture Notes: Chapter 4, 5, 10, and 11. **F**or the interrupts to work properly NVIC must be set correctly and the correct interrupt handler must be used. Reading chapter 11 is very important.

Lab Description:

There are two parts in this lab: (1) Implementation of two digital filters: a 1st order low-pass filter and a 2nd order notch filter. (2) Implementation of a motor speed measurement function. The first part uses timer triggered interrupt and the second part involves both timer and external interrupt.

PART I: Digital Filters

First Order Low Pass Filter:

The transfer function H(s) below is a 1st order low-pass continuous time filter with a bandwidth of b and a DC gain of H(0)=a/b.

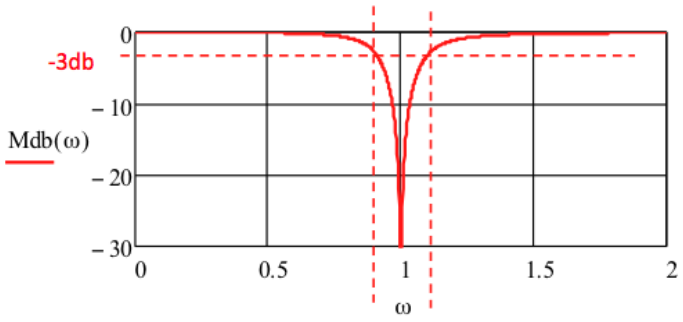
$$H(s) = \frac{a}{s+b}$$

Second Order Notch Filter:

The following transfer function is a 2nd order continuous time notch filter (a.k.a. band reject filter).

$$H(s) = \frac{s^2 + \omega_n^2}{s^2 + (\omega_n/Q)s + \omega_n^2}$$

The frequency ω_n (in rad/sec) is the center of the rejecting band, Q is the quality factor, and ω_n/Q is the bandwidth of the rejection band. The following figure shows this transfer function's magnitude response plot with $\omega_n=1$ and Q=5. The bandwidth of this transfer function is $\omega_n/Q = 1/5$. The x-axis of the plot is in rad/sec and is shown in linear scale. The vertical axis is in dB scale.



Matlab

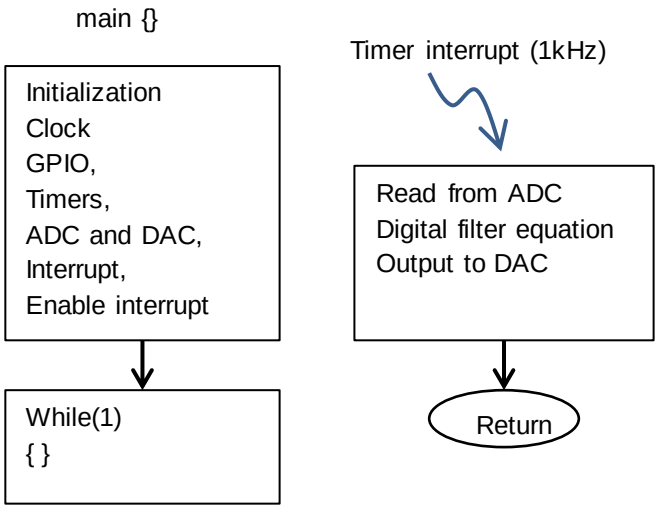
The following steps should be taken in Matlab to transform a continuous time transfer function to its discrete-time counterpart. Only discrete time transfer function can be implemented in software. The following example transfer function is a second-order low pass filter with ω_n = 5Hz

$$H(s) = \frac{(10\pi)^2}{s^2 + 5\pi s + (10\pi)^2}$$

The following Matlab code generates the coefficients for the discrete-time filter (See Chapter 4 for detail).

1) Hc = tf ([(10pi)^2] , [1, 5pi, (10pi)^2]) //This will create Hc as the continuous time transfer function
2) bode(Hc) //To observe the Bode diagram of Hc
3) Hd = c2d(Hc, 0.001) // Hd is the discrete time transfer function with a sampling frequency of 1KHz (sampling interval of Ts=0.001s).

Keep in mind that the Ts used in the c2d() function must match exactly the time interval that the digital filter code is executed. For microcontroller implementation, the only way to assure that a routine is executed at exactly at a certain time interval is to use a timer triggered interrupt. For this lab, a timer should be programmed to generate interrupts at exactly 0.001 second interval (or at 1kHz). The digital filter code should be placed in the interrupt handling routine. The following figure shows the structure of the digital filter program.



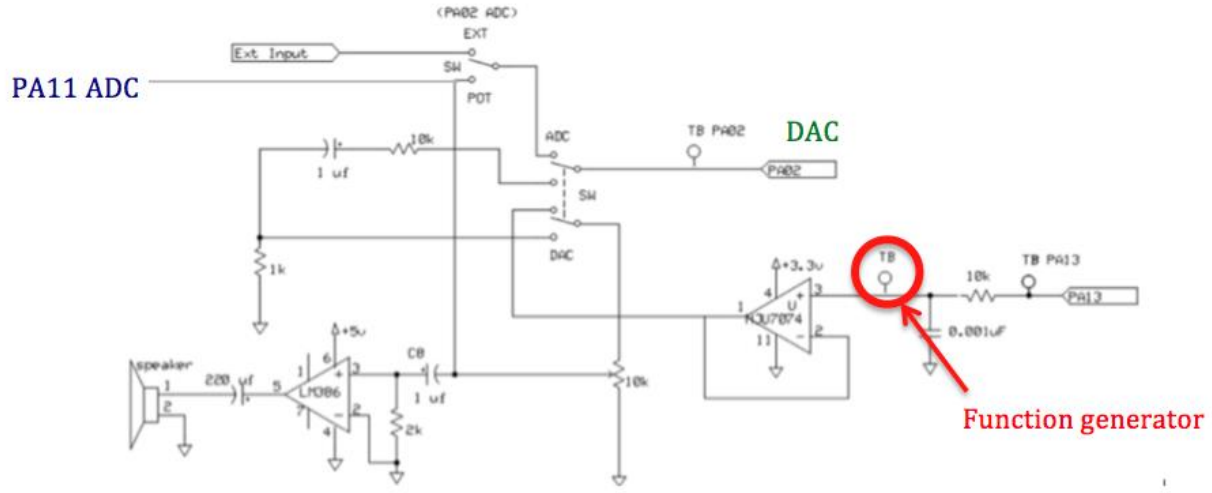
Variable Types:

For simplicity, all variables used in the filter equation should be declared as floating point data type variables. The analog input (from ADC) is in integer format so it must be converted to the floating point data format before it is used in the filter equation and the result from the filter equation should be converted to integer before it is sent to DAC. The main drawback of declaring a variable as floating point is that a floating point math operation can take a much as 100 times longer than that is required to perform an integer operation. This is true for microcontrollers that don't have a dedicated hardware floating point arithmetic processor (like SAMD20). For SAMD20, an integer multiplication only takes 1 CPU clock cycle. At the 8 Mhz clock, it only takes 1/8 uS to complete an integer multiplication. In this lab, you will determine the time that takes SAMD20 to perform a floating point multiplication.

Circuit connections:

For Version C board, the input to the digital filter should be a sinusoidal signal to the external connector (the green connector) pin-1. This signal is amplified by a factor of 2 and a DC offset is added by an on-board operational amplifier circuit before it reaches the PA03 input pin. See Figure 2.1 in Lab 2. Due to the x2 amplifier, the peak-to-peak sinusoidal waveform input should be less than 1.5V. The output of the digital filter should be sent to DAC on PA02 which can be probed directly through the PA02 test point. This signal is also connected through a voltage-follower op-amp circuit to the pin-2 of the external connector.

For version B board, the input signal should be injected into the circuit via the Test Point (TB) as shown in the following figure. This input passes through the POT and reaches the analog input pin PA11. The position of the ADC/DAC switch must be in the ADC position. Unlike the version C board, there is no amplification or added offset. So, it is important that the input signal must have a DC offset of 1.5v and the input sinusoidal waveform should have a peak-to-peak value of less than 3V. The output from the digital filter should be sent to PA02 which can be directly probed at the PA02 test point on the board.



Required Tasks: ***please comment your code**

Task 1: Implement a digital 1st order-low pass filter with a bandwidth of your choice. The sampling frequency should be set to 500 Hz, i.e., Ts=0.001 seconds. Note that ω_n of the input to the c2d() function (i.e., the continuous time transfer (H(s)) must be in rad/sec, not in Hertz.

Task 2: Implement a digital 2nd order notch filter that rejects a 60 Hz (~377 rad/sec) frequency component with a bandwidth of 10Hz and a sampling frequency of 500 Hz.

For Task 1 and 2 do the following:

- Work through Matlab to create the proper discrete time transfer functions.
    - Observe and get screenshots of the bode diagram for the transfer functions

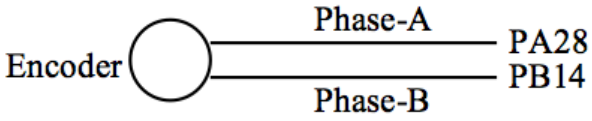
- Observe and get Oscilloscope screenshots of
    - The bandwidth
    - Rejection frequency
    - The aliasing effect

- Compare the bandwidth from the Matlab bode diagram to the measure bandwidth
- Determine the time it takes SAMD20 to carry out the computation for the follow tasks.
    - Filter equation
    - ADC
    - DAC
    - Anything else in your ISR

Note: To measure execution time, bracket the code with ***outset*** *and* ***outclr*** *to an output pin and use an oscilloscope to determine the time between these two instructions. Do not comment out some instructions between the brackets and try to determine the time for the instructions by the difference in measured times. Doing so, the complier might generate different executable codes and that makes the comparison of the timing measurements meaningless.*

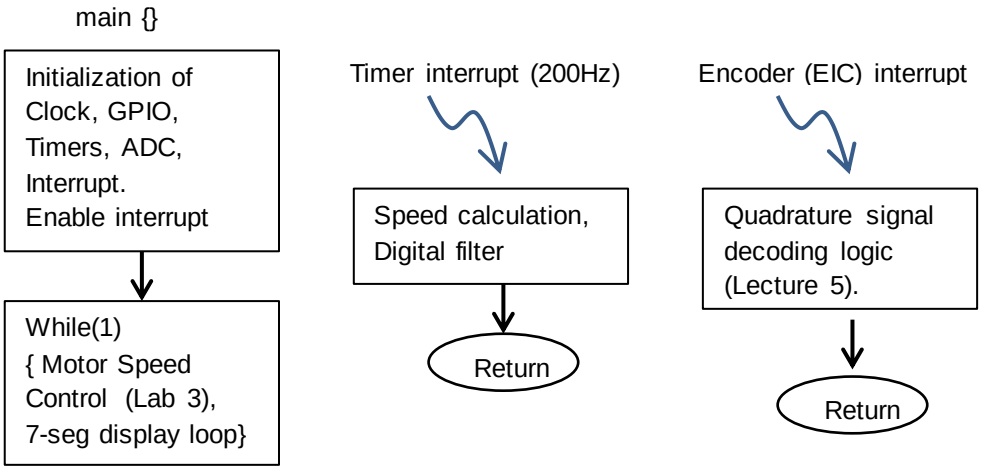
PART II: Motor Speed Measurement

The quadrature signals (Phase A and B) from the motor encoder are connected to PA28 and PB14 respectively as shown below. At least one of these two pins should be used as an external interrupt input pin and hence should be configured accordingly in the initialization routine. The EIC handler routine should implement the quadrature decoder logic as explained in Lecture note, Chapter 5.



As explained in Chapter 5, for speed calculation, a fixed sampling frequency is required. This fixed sampling frequency can be obtained by using a timer triggered interrupt. If the 'raw' calculated speed is directly displayed on the 7-segment display, the value will fluctuate and difficult to read. To smooth out the speed display, the calculated speed should be filtered by a 1Hz low pass filter before it is sent to the 7-segment display routine.

The quadrature signal decoding logic and the digital filter algorithm can be implemented in the same timer interrupt handler routine. The frequency of this routine should be set to 200Hz for this lab. The motor speed control program from Lab 3 should be used to control the motor speed. The 7-segment display routine program from Lab 1 should be used to display the filtered speed measurement. Both the motor speed control code and the 7-segment code can be executed in the main() (since the frequency of execution of these two programs is not critical) or they can be executed in the timer interrupt handler routine (if a more precise frequency is desired) together with the speed calculation and digital filtering routine. The following figure shows the structure of the program.



Required steps:

Step 1: Reconfigure the digital filter program developed in Part I so that the sampling frequency is 200Hz and the bandwidth is 1Hz. Put a 1Hz square wave input to the filter. Observe the output, it should be a filtered version of a square-wave.

Step 2: Incorporate the 7-segment display routine into the reconfigured digital filter program developed in Step 1. Verify that a variable can be correctly displayed on the 7-segment display. At this point, the digital filter routine and display routine are two independent routines.

Step 3: Set up the external interrupt for the quadrature signals from the motor encoder (on PA28 and PB14) and implement the decoding logic (but NOT the speed calculation part) as described in Lecture note, Chapter 5. Display the value of the position counter on the 7-segment display. Check that, if only one edge of phase A (or B) is used in the decoding logic, when hand-turned the motor shaft for one full revolution, the counter should increase by about 400 counts (since the encoder used in the lab is 400-line encoder). If both edges are used (both edges trigger interrupt), the incremental count should be 800. If both edges of both phase A and B signals are used, the increment should be 1600.

Step 4: Incorporate the motor speed control (from Lab 3) into the program. Make sure that the motor speed can be controlled by the pot. Now, there are three independent routines running ---- digital filter, display, and motor control.

Step 5: Implement the speed calculation (as described in Chapter 5) and display the result of the calculation on the 7-segment display directly (don't put it though the filter yet). Use an optical tachometer to verify the speed readout. The accuracy of your speed reading should be within few percentages of the readout from the optical tachometer. You will see that the display fluctuates so much, it is hard to read the last two digits.

Step 6: Pass the calculated speed though the 1Hz bandwidth low-pass filter before displaying it. The speed readout should now be much more stable and easier to read.