ME 461 Laboratory #4
Digital-to-Analog Conversion and Sampling

Goals:

1. Write a driver for interfacing to the TLV5606 10-bit Digital-to-Analog Converter (DAC).
2. Observe the effects of signal aliasing.
3. Play audio signals on headphones using the DAC.

Exercise 1:

In this exercise, you are going to configure the microcontroller to communicate with the TLV5606 Digital-to-Analog Converter (DAC) chip via the Serial Peripheral Interface (SPI) protocol. First, solder the TLV5606, 10kΩ potentiometer and 0.1µF capacitor to the breakout board. From the DAC, connect the CS pin to GND, the DIN pin to SIMO (P3.1), the SCLK pin to SCLK (P3.3), and the FS pin to a GPIO pin. You should set the DAC reference voltage (the output of the potentiometer) to roughly ½ V_{CC} so that the full-scale range of the DAC is 0 – V_{CC}.

By now you should be somewhat familiar with the SPI protocol and the USCI module on the microcontroller. You were asked to configure the USCIB0 module for SPI communication to the DAC in homework 4. Implement that configuration here.

You should also know by now that most of the steps required to implement configurations such as this have been discussed in class, and guidance can be found in the lecture notes. As with previous labs, this would be an excellent time to take another look at the relevant MSP 430 User's Guide section(s) and the "msp430x22x2.h" header file. As you continue to be told, control register names and predefined constants with (relatively) easy to remember names are declared in the "msp430x22x2.h" header file. You are strongly encouraged to start making regular use of them and to become comfortable combining them as shown in many previous examples. Doing so will greatly simply the effort required to properly configure control registers and improve the readability and portability of your code.

Use a 1 MHz SPI clock rate. Set up the transmit (TX) interrupt for communication to the DAC. Recall from the user's guide that the RX and TX interrupts are shared between USCIA0 and USCIB0. This means that the UART serial communication to the PC uses the same interrupt vectors as the SPI communication to the DAC. You may use the prototype below as a starting point, though it should already exist in the code generated by the project creator.

```
// USCI0 TX interrupt service routine
#pragma vector=USCIAB0TX_VECTOR
__interrupt void USCI0TX_ISR(void)
{
      if((IFG2&UCA0TXIFG) && (IE2&UCA0TXIE)) {
            // handler for USCIA0 TX interrupt
      }
```

```
      if((IFG2&UCB0TXIFG) && (IE2&UCB0TXIE)) {
            // handler for USCIB0 TX interrupt
      }
}
```

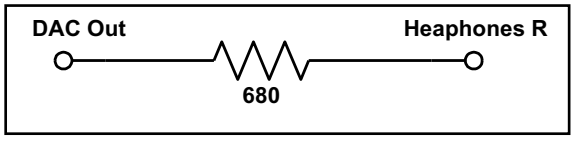
Notice that a read of the `IFG2` register is necessary to distinguish callers of the ISR. An additional read of the `IE2` register is required to ignore flags corresponding to disabled interrupts. For example, the code above protects from the case where a set USCIB0 IFG causes an interrupt (because the USCIB0 interrupt is enabled), but the USCIA0 IFG is also set (though its interrupt is disabled), an errant service of the USCIA0 handler would occur unless the code checks IE2 first. Don't forget to clear the interrupt flags when the ISR is executed, but only if the conditions for clearing them automatically are not met (see section 16.3.8 in the user's guide).

Write the driver for communicating with the DAC. Send the data to the DAC every 1ms. Use the 16-bit data format described in the TLV5606 datasheet which you studied in homework 4. Note that the USCI SPI module requires the data to be sent out 8 bits at a time, so a 16-bit write to the DAC will require 2 write cycles from the microcontroller. Don't forget that you need to control the FS pin according to the DAC's datasheet as well. If you find it helpful, you may write a function called `Write_DAC` to modularize your code.

Test the interface by sending several commands to the DAC and scoping the results. Print out the expected voltage to the serial port and compare it to the scope reading. Then generate a sawtooth pattern similar to what you created with the home-made DAC in Lab 2. **Demonstrate to your TA.**

Exercise 2:

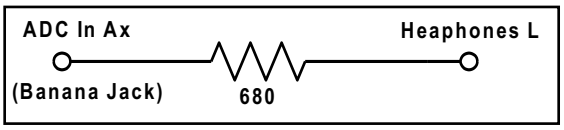
Connect the output of the DAC to the right channel of the stereo jack through a 680Ω resistor as shown in the figure below. If the jack is not already in place, solder it to the board as well.



Connect the headphones at your bench to the stereo jack. Output a square wave signal at an audible (and preferably tolerable) frequency; 100-1000 Hz is recommended. To do this, switch the output of the DAC between its maximum and minimum voltages at a frequency in the range given above. You should hear a tone from the headphones. **Demonstrate to your TA.** You can create a variety of NES-style audio tracks using this simple and (computationally and monetarily) inexpensive tone generator.

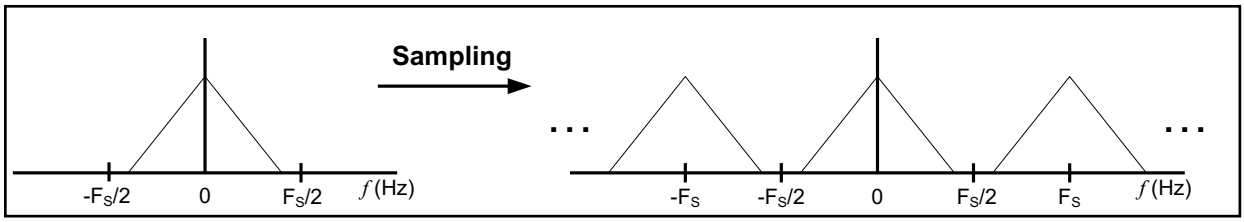
Exercise 3:

In this exercise you are going to explore the effects of discretization and signal aliasing. Solder the circuit shown below.



Configure the function generator to produce a 100-Hz sine wave centered at 1.7V with a peak-peak amplitude of 3V. Confirm these settings by scoping the output of the function generator. **Show the waveform to your TA.** Since the function generator can only output a very small current, it is necessary to increase the impedance of the headphone channel driven by the function generator. This is why you soldered the 680Ω resistor between the banana jack and the left channel of the headphone jack.

Connect the function generator to the banana jack that you wired to an open ADC channel. Sample the ADC channel at a rate of 10 kHz and echo the 10-bit ADC reading to the DAC (also at 10 kHz). Put on the headphones and observe the effects as you increase the function generator output frequency from 100 Hz to 4000 Hz. In your left ear you should hear a pure tone at the frequency you set. In your right ear you should hear this tone plus some higher-pitched ringing. This ringing is caused by the fact that sampling reproduces the spectrum at multiples of the sample frequency. Refer to the figure below.



Signal spectrum (left) and the sampled counterpart (right)

Given a band-limited input signal (Fourier transform is zero outside a finite band of frequencies) on the left, the *ideal* sampled signal produces the spectrum on the right. However, this sampled signal is not what you are hearing in your right ear. What you are hearing is a sampled signal produced by a zero-order hold. The spectrum of the reconstructed signal becomes a crudely lowpass-filtered version of the original spectrum. This is why the ringing is fainter than the pure tone. View the sampled signal on channel 2 of the oscilloscope. **Show it to your TA.** Notice that the signal is a "stair-step" reconstruction of the original signal. As you have observed, this difference has a nontrivial effect on the sound the signal produces. If we had more time, we could design an analog filter to reduce the effects of sampling and more accurately reconstruct the original signal.

Now, increase the input frequency beyond the Nyquist frequency (5 kHz). Note the behavior of the audio signal reproduced in the right channel of the headphones compared to that of the left channel. What do you notice? **Show your TA** the waveforms on the oscilloscope. These are the effects of aliasing. With a sample rate of 10 kHz, it is impossible to retain any components of the original signal above 5 kHz.

Any attempt to sample higher frequencies will result in the signal being wrapped to inside the principal range $[-F_S, F_S]$.