Goals:

1. Learn how to configure and use the MSP430’s 10-bit SAR ADC.
2. Measure the output voltage of your home-made DAC and compare it to the expected value.
3. Calibrate and use the MSP430’s internal temperature sensor.
4. Design and implement an RC anti-aliasing filter.
5. Solder a microphone circuit to the breakout board and use it to sample audio signals.

Exercise 1:

In this exercise you will configure the MSP430’s 10-bit analog-to-digital converter (ADC) to sample the output of the PWM + low pass filter you designed in lab 2.

This would be a good time to take another good look at, the lecture notes, the MSP 430 User’s Guide ADC10 section, and the “msp430x22x2.h” header file. As you have heard many times already, control register names and predefined constants with (relatively) easy to remember names are declared in the “msp430x22x2.h” header file. For example, consider this source code:

\[
\text{ADC10CTL0} = \text{SREF}_0 + \text{ADC10SHT}_2 + \text{ADC10ON} + \text{ADC10IE};
\]

On line 172 of the “msp430x22x2.h” header file you will find:

\[
\#define \text{SREF}_0 \ (0*0x2000u) \quad \text{/* VR+ = AVCC and VR- = AVSS */}
\]

Obviously, zero times anything is zero and we can assign this value of \text{SREF}_0 to \text{ADC10CTL}, as in

\[
\text{ADC10CTL0} = \text{SREF}_0;
\]

The binary representation of \text{ADC10CTL0} will then be %0000 0000 0000 0000. Note that the three most significant bits are ‘000’ (referring to the MSP 430 User’s Guide) and setting these bits of \text{ADC10CTL0} tells the microprocessor to use \text{VCC} as the positive voltage reference (\text{VR}_+), and \text{VSS} (Ground) as the negative voltage reference (\text{VR}_-).

As you know from Lab #2, the full-scale range of your home-made DAC is \text{VSS} = 0V to \text{VCC} = 3.6V. This means, you can use the example above to help get you started with the configuration of the ADC. Note that because we are using a voltage reference set that is already available on the board (\text{VCC} and GND), you do not need to worry about using any of the reference generator features.

On line 167 of the “msp430x22x2.h” header file you will find:

\[
\#define \text{ADC10SHT}_3 \ (3*0x800u) \quad \text{/* 64 x ADC10CLKs */}
\]
Thus, the binary representation of ADC10SHT_3 = %0001 1000 0000 0000. We could now add SREF_0 and ADC10SHT_3 together, as in

\[ \text{ADC10CTL0} = \text{SREF}_0 + \text{ADC10SHT}_3; \]

Now the binary representation of ADC10CTL0 = %0001 1000 0000 0000. Note that three most significant bits remain ‘000’ but the fourth and fifth most significant bits are now ‘11’. Setting the fourth and fifth most significant bits (bits 12 and 11 of a 16 bit value) of ADC10CTL0 to ‘11’ tells the microprocessor the sample and hold time is equal to 64 cycles of the ADC10 source clock.

The appropriate sample and hold time depends on both the internal input resistance of the microprocessor and the external source resistance associated with any circuitry attached to the ADC input pin. Now refer to section 22.2.5 of the MSP430 user’s guide to determine an appropriate minimum sample and hold time for your situation. The external (source) resistance \( R_S \) is the value of \( R \) in your RC low pass filter. Record the minimum sample and hold time below.

\[ T_{S,\text{min}} = \underline{\text{seconds}} \]

Check the sample-and-hold time with your TA. It is recommended that you use the dedicated ADC oscillator, ADC10OSC, for the ADC10 conversion clock source (assume its speed is exactly 5MHz). Your minimum sample-and-hold time will dictate whether you need to divide the ADC10CLK source with the \( \text{ADC10DIV}_x \) bits. Record the ADC clock rate after any \( \text{ADC10DIV}_x \) divisions below and the resulting number of ADC10 clock pulses (an integer) that correspond to the minimum sample time you recorded above.

\[ \text{ADC10CLK rate after divide: } \underline{\text{MHz}} \quad T_{S,\text{min}} = \underline{\text{ADC10 CLKS}} \]

Check these with your TA.

Continue to determine the remaining configuration parameters according to the following guidelines. You are strongly encouraged to use the predefined constants found in the “msp430x22x2.h” header file and to become comfortable combining them as shown in the examples above (as well as examples from the lecture notes). Doing so will greatly simplify the effort required to properly configure control registers and improve the readability of and portability of your code.

You should connect the PWM DAC output to one of the analog inputs on the port 2 pins. To set that pin to analog input mode, you need to set the corresponding bit in the \( \text{ADC10AE}_{x} \) register. Also, you must tell the ADC which input channel to convert. You may use any of the conversion triggers (referred to in the user’s guide as the sample-and-hold source) you like. The \( \text{ADC10SC} \) bit is a software trigger that is easy to use and thus recommended at this point, but the other (timer-based) triggers are better suited to fast sample rates (>10kHz) and may be necessary in future lab exercises. Whatever trigger source you use, you should trigger conversions every millisecond. Use the knowledge (and code) you gained from the first half of Lab 2 to generate a 1ms timing interval. For now, use the straight-binary data format for the conversion results.
Finally, make sure you enable the ADC10 interrupt and turn the ADC10 on. For a flow chart procedure of a general ADC conversion, refer to figure 22-5 in the user’s guide. Your program should control ENC and ADC10SC in accordance with this diagram. Only read the conversion result (ADC10MEM) in the interrupt service routine to ensure the result is accurate and current. In your program, print out the raw conversion results and the corresponding integer millivolts to the serial port. Additionally print the expected DAC voltage as you did in Lab 2. When converting from raw ADC results to millivolts, you may not use floating-point numbers, so keep in mind the caveats of integer math (such as 2/3 = 0). **Demonstrate to your TA.**

**Exercise 2:**

Now, change the analog input channel so that the ADC converts the internal temperature sensor’s voltage. Modify the ADC clock input divider and sample-hold time to achieve the large minimum sample period listed in section 22.2.8 of the user’s guide. Print the voltage in millivolts to the terminal window. **Demonstrate to your TA.**

Next, calibrate the temperature sensor assuming the lab is at room temperature. In section 22.2.8 of the user’s guide, Figure 22-13 shows the sensor voltage as a linear function of temperature. You may use the published value of the slope (3.55mV/°C), but you will have to determine the offset voltage using the ambient temperature in lab and the corresponding sensor voltage. Record the value of the temperature sensor offset voltage below.

\[ V_{\text{TEMP}} \text{(mV)} = 3.55 \times \text{Temp(°C)} + \text{________ mV} \]

Once you have found the offset voltage, convert the temperature sensor voltage to degrees Celsius and print the value to the serial port. **Demonstrate to your TA.**
Exercise 3:

In this exercise, you are going to solder a microphone circuit to the breakout board and use it to sample audio signals. The circuit schematic is shown below.

Microphone circuit schematic

Notice that the schematic includes an RC anti-aliasing filter on the output of the amplifier. This is a necessary component when sampling an input signal with a frequency spectrum that is not band-limited to half the sample rate. Before you begin soldering the components, you are going to design the anti-aliasing filter.

Our goal for the microphone circuit is to sample common audio signals. The audio signals you hear from a CD or mp3 player have been sampled at a rate of 44.1 kHz. This ensures that every signal you can hear with your ears is accurately represented. In most cases, though, a sample rate this high is more than enough. It turns out that most speech and common sounds are in the low-frequency range. We are going to use a sample rate of 10 kHz, which means we can capture audio signals in the 0 - 5 kHz range. However, there will still be some sounds and noise outside this range. The goal of the anti-aliasing filter is to attenuate frequencies above the 5 kHz Nyquist frequency.

Note A:
1) Leave Pins 1 & 8 open for Gain = 20.
2) Use \( R_G = 1.2k \) and \( C_G = 10\mu F \) for Gain = 50.
3) Use \( R_G = 0 \) and \( C_G = 10\mu F \) for Gain = 200.

Note B:
Use a 470\( \mu F \) bypass capacitor between \( V_{CC} \) and GND when using the microphone and LM386.

See LM386 datasheet for more information.
Your job is to select an RC value that satisfies the requirement such that at 5 kHz the attenuation should be roughly 50%, that is, only half the amplitude of a 5-kHz signal should be passed through. *Hint:* recall that the transfer function of an RC low pass filter is $G(j\omega) = 1/(1+RCj\omega)$. Record the cutoff frequency and the RC value that achieves this below. Feel free to use MATLAB (with the `tf` and `bode` functions) to check your results.

$$f_c: \quad \text{___________ Hz} \quad \text{RC: \quad \text{___________}}$$

**Check these numbers with your TA.** Now, from the list of components written on the whiteboard, choose a resistor and a capacitor that gives an RC close to what you calculated. Record the values of these components below. **Check with your TA.**

$$R_A: \quad \text{___________} \quad C_A: \quad \text{___________}$$

Now, use the schematic above and the demo board in lab to guide you as you solder the components to your board. For now, leave pins 1 and 8 on the LM386 open for a gain setting of 20. You may need to increase the gain later. When you are done soldering the components to the board, scope the output of the amplifier as you make some noise into the microphone. You should see a “typical” audio signal waveform biased around $\frac{1}{2} V_{CC}$. If the output of the amplifier uses only a small portion of the voltage range ($0V - V_{CC}$) for reasonably audible noises, increase the gain according to the note in the schematic. **Show the audio signal to your TA.** Also, scope the output of the anti-aliasing filter and see if you notice any differences.

Now, sample the microphone circuit’s output at a rate of **10 kHz**. You may use the software trigger bit (AD10SC) as long as you don’t run into lags created by calling an ISR too often. You will have to perform another minimum sample-hold time calculation using the $R_A$ you found above. Change the data format to 2’s complement. This will cause the ADC to read 0 when there is no input to the microphone, since it is biased at $\frac{1}{2} V_{CC}$. Write code to light an LED when the magnitude (absolute value) of the ADC reading is greater than something like half of the maximum. You do not need to convert the ADC reading to a voltage. *Hint:* in 2’s complement format the conversion result ranges from -512 to +511 (i.e. 0V to $V_{cc}$, where $V_{cc}/2$ results in 0). **Demonstrate to your TA.**

---

ME 461

Lab #3