**Lab 4**

In this lab, you will build a sine wave lookup table in the DSP, output it using the ePWM module, and use a simple analog filter to reconstruct the sine wave.

1) For this lab, the following tools and components will be of value:
   a. Your BitScope or other oscilloscope
   b. A protoboard or other means of connecting components
   c. (1) 10kΩ resistor
   d. (1) 100nF capacitor
   e. (1) 1μF capacitor

2) The ePWM module in the DSP board allows you to generate pulse width modulated (PWM) signals that can be used to control the duty cycle of power electronics systems. In the examples provided by TI (such as the ones you’ve already used for toggling GPIO pins, and for ADC), there are also a number of examples for generating PWM signals. In this lab we are going to modify the project in example folder ‘epwm_up_aq’. You may wish to make a copy of the folder prior to modifying the project ‘Example_2833xEPwmUpAQ’. Other ePWM projects could be chosen as well. This just happens to be the one the instructor employed.

3) In project ‘Example_2833xEPwmUpAQ’, six ePWM channels are used to generate waveforms where independent modulation occurs between ePWMxA and ePWMxB. In addition, two sets of counters (CMPA and CMPB) are incremented or decremented. We, however, are only concerned with one PWM channel and one counter (i.e., Epwm1A). And, we don’t want to increment (or decrement) a counter. Rather, we would like to generate a sine wave and send it out through the ePWM channel, essentially performing sinusoidal pulse width modulation (SPWM). At the output, using a simple analog filter, we can reconstruct the sine wave. A simple block diagram shows the system:

![Block Diagram](image)

Thus, we are able to use the ePWM module as a substitute for a digital-analog (D/A) converter.

4) Using your program from Lab 3 as a guide, build a lookup table (buffer) (in your ‘main’ code) for your sine terms. To help you, consider the following:
   a. You will need to add a header file called ‘math.h’. This will allow you to use various math functions (such as the sine function) in your code.
b. For now, allocate 512 words of memory for the size of the table, and assume each value will correspond to a 1 degree change (i.e., 360 values will yield a complete cycle of sine terms). Note that one of the questions in Lab3 asked ‘How many values are stored in your buffer?’. The answer was 999. In other words, the number of samples placed in the buffer was less than the size of the buffer (2048). This approach should be maintained here.

c. The amplitude of each sine wave term should be relative to the value of the time-based period (TBPRD). Refer to http://focus.ti.com/lit/ug/spru791d/spru791d.pdf, Figure 2-13, for an understanding of this term. In this program, TBPRD has been set to 2000. You will thus want to have your sine wave terms vary between 0 and 2000.

5) You can start reducing some of the code. First, avoid modifying any code associated with the following:
   i. Function ‘InitEPWM1Example’.
   ii. Function ‘epwm1_isr’

After that, any code that calls out the following can be removed to simplify/shorten the amount of code:
   iii. ePWM2 (including interrupt service routines (ISR) and initialization)
   iv. ePWM3 (including interrupt service routines and initialization)
   v. Compare A and B (CMPA and CMPB)
   vi. Function ‘update_compare’

6) Function ‘update_compare’ is incrementing and decrementing counters, but we just wish to place each sine term into the proper compare register, and use that for each PWM carrier cycle. When the sine term is small, say around 0 degrees, the value placed into the compare register should be small. When the sine term is large, say around 90 degrees, the value placed into the compare register should be large. In any case, the value placed into the compare register must be between 0 and the value of TBPRD. Refer again to http://focus.ti.com/lit/ug/spru791d/spru791d.pdf, Figure 2-13, for a pictorial view of the “up-count” mode and the effect of the CMP register. So, a number of changes will be required to the overall program:
   i. Function ‘update_compare’ was called by the ISR for epwm1. Now all of the code will be inside the ISR. The code will need to do the following:
   ii. Increment counter EPwm1TimerIntCount
   iii. Put the sine value pointed to by this counter into the proper compare register (namely, EPwm1Regs.CMPA.half.CMPA). This might be coded as follows:
      \[ EPwm1Regs.CMPA.half.CMPA = SineTable[EPwm1TimerIntCount] \]
   iv. After this line, you should compare the value of the counter with the size of the number of samples, and reset the counter when it reaches the limit (in this case, 360).
   v. Make sure you keep the two lines of code to clear the interrupt flag and acknowledge the interrupt.

7) The function ‘InitEPWM1Example’, among other things, can be used to modify the PWM (carrier) frequency. We know that the F28335 has an internal system clock (SYSCLKOUT) of 150MHz. The time-based clock used by the Epwm module (TBCLK) can be reduced by using the equation:
TBCLK = SYSCLKOUT / (HSPCLKDIV x CLKDIV)

In this example, you can see from function ‘InitEPWM1Example’ that TBCLK is 37.5MHz. The actual PWM frequency can be found by dividing TBCLK by TBPRD (refer to Section 2.2.3 of spru791d identified above). In this case, the actual PWM frequency is 18.75kHz. Since your sine wave is made up of 360 points, your reference wave frequency at the output of Epwm1A will be approximately (but not exactly) 50Hz.

8) Try compiling and running the program (hopefully you have removed any extraneous variables that aren’t required in your modified program). Connect a low-pass filter to output pin Epwm1A/GPIO0. Use the 10kΩ resistor and 100nF capacitor. Before the filter, when you look at the output of ePWM1A (at GPIO0), can you estimate the PWM frequency? After the filter, can you see your reference wave signal of approximately (but not exactly) 50Hz? If the DSP has been programmed correctly, the 50Hz signal should oscillate between approximately 0V and approximately 3.3V.

9) Replace the 100nF capacitor with a 1μF capacitor. Observe the reference wave again. Is this a good filter design? Why or why not?

10) Modify the program to output exactly 50Hz.

11) You are now able to build a sine wave inside the DSP board, modulate the signal using the ePWM module in the DSP, and reconstruct the signal using a simple low-pass filter to utilize the ePWM module as a D/A converter. This ends Lab 4. Be sure the DSP board is disconnected from the computer and from the oscilloscope prior to removing power.