1. CPE 325: Laboratory Assignment #11 Digital-Analog Converter

Objectives: This tutorial will review the MSP430 12-bit digital to analog converter (DAC12) device. In this lab you will learn to configure and operate the device with arbitrary waveform outputs. Specifically, you will learn:

- How to optimize the output resolution of a digital to analog converter
- How to create waveform lookup tables in MATLAB
- Configuration of the DAC12 peripheral
- Operation of the DAC12 peripheral

Note: this lab requires all the previous tutorials, especially tutorials on UART communication, timer A, and getting started with MSP430F IAR Embedded Workbench. Further, reading the user guide for TI Experimenter's Board is useful.

1.1. Digital to Analog Converters

In the previous lab, we explored analog to digital converters which express an analog input as a numerical value that can be stored by the microcontroller. Conversely, digital to analog converters take a numerical value that has been programmed by the user and output it as an analog voltage. The concepts of resolution and reference voltages are seen here again and should be understood to make the best use of the digital to analog converter.

The MSP430FG4618 has two 12-bit digital to analog (DAC12) modules available. Each module has one control register and one data register. The control register determines the output pin, output level, reference voltage, 8- or 12-bit resolution, when the output latch triggers, the amplifier settings, and the interrupt settings. The data register holds the value to be converted to an analog voltage output.

The DAC12 operation is much simpler than the ADC12. There is no timer associated with the device, so one of the MSP's timers such as the TimerA must be used. Likewise, there is no internal voltage reference, so it is easiest to configure the ADC12 voltage reference to be used. Once the DAC12 is configured and enabled, a timer interrupt can periodically call an ISR that writes a new value to the DAC12 data register. By default, when the data register is written, a new output voltage is generated.

As discussed in the previous lab, it is important to optimize your signal's vertical (time) and horizontal (voltage) resolution for your requirements. Figure 1 below is an example of a D/A converter output waveform.

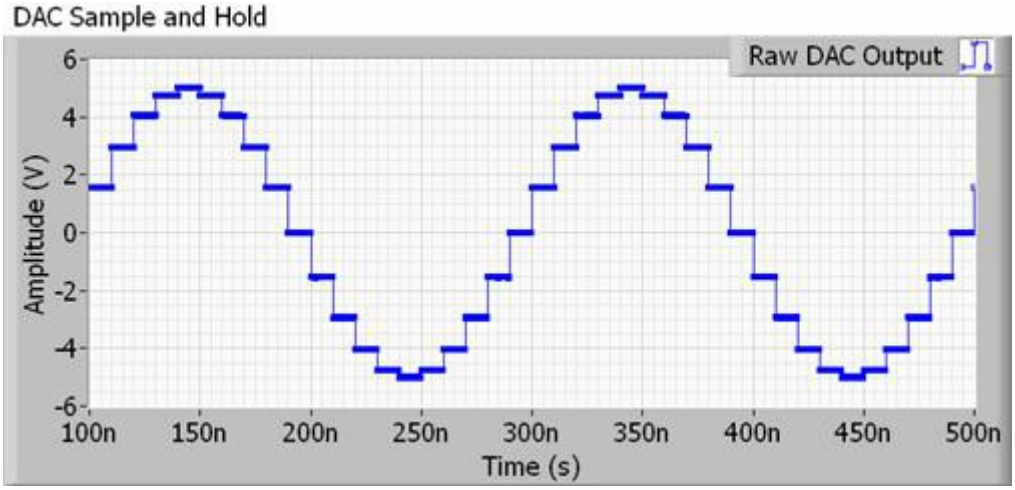

Figure 1. D/A converter output waveform

Notice how the output looks very blocky. With better resolution, the "blockiness" of the signal is minimized. The resolution is a function of how often the signal data is updated, the resolution of the converter, and the output waveform algorithm.

In this project, we will create a lookup table using MATLAB. The lookup table is a large array stored in memory that holds individual values that will be passed to the DAC12 in sequence in order to output a signal. While creating our waveform lookup table, it is important to keep in mind that we can increase our output resolution by increasing the number of samples used per period of our waveform. That also requires us to use a faster timer to refresh our DAC12 module in order to maintain the correct frequency.

MATLAB works by manipulating matrices. Below you will see an example of how to create and save a lookup table consisting of 256 values for a sine wave.

1.2. Sinusoidal Wave Generator

Let us consider the following application. Your task is to develop a sinusoidal wave generator using the TI Experimenter's board. The function you should implement is described as follows: y = 1.25*(1+sin(x)). The maximum voltage is consequently 2.5 V and the minimum voltage is 0 V. The frequency of the sine wave should be 10 Hz, that this one period should take 10 ms.

The signal is generated using the MSP430's DAC12 digital-to-analog converter. Periodically we will be sending a new digital value to the DAC12. These values are prepared in advance in a constant table. Let us first discuss how to create this lookup table. Our first step is to determine the number of samples we want to have and the number of bits to represent them. More samples, means a better quality of the generated sine wave. Let us assume that we want to have 256 samples of the function y in a lookup table, each sample with a value in the range of [0 ... 4095]. The sample with value 0 corresponds to 0 V, and samples with the value 0xFFF (4095) corresponds to 2.5 V. We can use MATLAB to generate the lookup table for x=0, Δx, 2Δx,..2*pi, where Δx=2*pi/256. Figure 2 shows the MATLAB program. We generate 256 samples (actually 257) in the range x=[0...2*pi] and calculate the integer values that correspond to each sample x, using the original function y=1.25*(1+sin(x)). The values are rounded to nearest integers and the samples are written back to a file. Note: the actual number of samples

is 257, so we should remove the last one (the same as the first). We should also ensure that no samples are larger than 4095 (the largest number that can be represented by 12-bits). These constants will be used to initialize the lookup table visible to your program (here we create a header file named sine_lut_256.h). The sine_lut_256.h file can be found on ANGEL.

```
1. x=(0:2*pi/256:2*pi)
2. y=1.25*(1+sin(x))
3. dac12=y*4095/2.5
4. dac12r = round(dac12)
5. dlmwrite('sine_lut_256.h',dac12r, ',')
```

Figure 2. Matlab program to generate a 256-entry lookup table.

The next step is to determine the trigger period. We want 256 samples to spread over 2*pi range of x. The required period of the sine wave is 10 Hz (0.1sec). That means that the trigger period is 0.1/256 sec. We will use a TimerA device to generate triggers. Assuming that the default clock frequency on SMCLK is used as the timer clock (1048576 Hz), we can determine the value that needs to be written to the TimerA counter (0.1/256)*FSMCLK = 410.

Figure 3 shows the complete program. We stop the watchdog time, initialize the ADC12 to give a reference voltage of 2.5 V, and initialize the timer to raise an interrupt every 0.1/256 sec. The TimerA ISR wakes the processor, we read the next sample from the table, and output it to the DAC12.

```
6. //*******************************************************************************
7. //   MSP430xG461x Demo - DAC12_0, Output Voltage SINEWAVE on DAC0
8. //   Device: TI Experimeter's Board
9. //
10.//   Description: Using DAC12_0 and 2.5V ADC12REF reference with a gain of 1,
11.//   output sinisoidal wave on P6.6, y=1.25(1+sin(x)).
12.//   Normal mode is LPM0 with CPU off.  WDT used
13.//   to provide ~0.064ms interrupt used to wake up the CPU and update the DAC
14.//   with software. Use internal 2.5V Vref.
15.//   ACLK = 32kHz, SMCLK = MCLK = WDTCLK = default DCO 1048576Hz
16.//
17.//
18.//               MSP430xG461x
19.//            -----------------
20.//         /|\|              XIN|-
21.//          | |                 | 32kHz
22.//          --|RST          XOUT|-
23.//            |                 |
24.//            |        DAC0/P6.6|--> sine (10Hz)
25.//            |                 |
26.//
27.// Authors: Aleksandar Milenkovic, milenkovic@computer.org
28.//          Max Avula, ma0004@uah.edu
29.//        UAHuntsville
30.// Date:  March 2012
31.// Built with IAR Embedded Workbench IDE Version: 6.3.11.2079
```

```
32. //*****************************************************************************
33. #include "msp430xG46x.h"
34. #include "sine_lut_256.h"  /*256 samples are stored in this table */
35.
36. void main(void)
37. {
38.   unsigned int i;
39.
40.   WDTCTL = WDTPW + WDTHOLD;                    // Stop WDT
41.   ADC12CTL0 = REF2_5V + REFON;                 // Internal 2.5V ref on
42.   for (i=50000; i>0; i--);                     // Delay to allow Ref to settle
43.   __disable_interrupt();                       // Disable Interrupts
44.   DAC12_0CTL = DAC12IR + DAC12AMP_5 + DAC12ENC;   //Sets DAC12
45.   CCTL0 = CCIE;                                 // CCR0 interrupt enabled
46.   CCR0 = 410;                                // Sets Timer Freq (1048576*0.1sec/256)
47.   TACTL = TASSEL_2 + MC_1;                    // set clock to SMCLK, up mode
48.   i=0;
49.   while (1)
50.   {
51.     __bis_SR_register(LPM0_bits + GIE);     // Enter LPM0, interrupts enabled
52.     DAC12_0DAT = LUT256[i];
53.     DAC12_0DAT &= 0xFFF;
54.     i=(i+1)%256;
55.   }
56. }
57.
58. #pragma vector = TIMERA0_VECTOR
59. __interrupt void TA0_ISR(void)
60. {
61.   __bic_SR_register_on_exit(LPM0_bits);     // Exit LPMx, interrupts enabled
62. }
```

Figure 3. C program that generates a sine wave output using the DAC12 (Lab11_D6.c)

1.3. References

In order to understand more about the DAC12 peripheral and its configuration, please reference the following materials:

- Davies Text, pages 485-492
- MSP430FG4618 User's Guide, Chapter 31, pages 869-886

1.4. Assignment

Write a C program that creates a waveform generator. It should start by outputting a sine wave to DAC0. Each time SW1 is pressed then released, the wave form should change, first to a triangle wave, then a sawtooth wave, then back to a sine wave (repeating forever). Each time SW2 is pressed and released, the output amplitude should be adjusted by a factor of 2 (full amplitude, then half amplitude, then quarter amplitude, then back to full amplitude, etc.).

Use the 2.5V internal reference from the ADC12 module. The full amplitude for each wave form should be from 0V to 2.5V, one period should contain 256 samples, and the frequency should be 15 Hz.

You should use MATLAB to create waveform lookup tables for the sine and triangle waveforms. For the sawtooth wave, you can either use MATLAB or an internal function to generate the waveform.