Programming for Embedded Systems
Lab 2: Digital to Analog Conversion

February 11, 2014

Intro

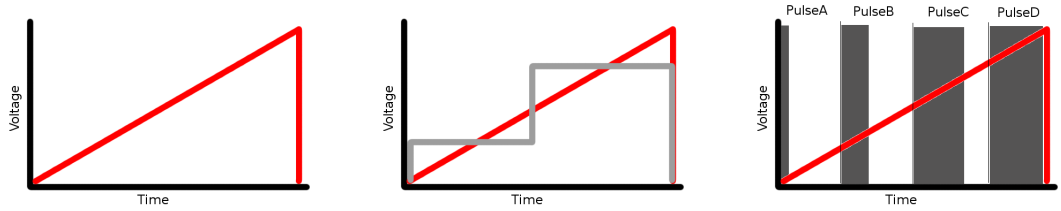
The MSP430 does not have analog output pins so we cannot directly create anything other than Vcc. There are techniques to simulate analog output with digital output though, known collectively as digital-to-analog conversion, or DAC. In this lab we will use a technique known as Pulse Width Modulation.

Pulse Width Modulation

In pulse width modulation we reconstruct a digital output by quanitizing (dividing into pieces) the desired analog output and matching the digital output to the average output power of the signal. Generally some kind of reconstruction filter is then used to smooth out the output pulses so that the output is a analog signal rather than a series of pulses.

The challenging part of PWM is that our output pulses will all be the same power. The only way we can control the average output power is to change the duration of our output – this is the width of our pulses and the source of the name, pulse width modulation. An example of this modulation is in Figure 1.

In PWM the original signal is divided into bins. Inside of each bin a pulse is used to approximate the signal in that bin. The pulse can only go from ground to Vcc, so the PWM pulse varies its width to match the power of the original signal in that bin, as in Figure 1(c). The duration of the pulse itself is also a digital value, which introduces another source of error. The number of bins per second is the sampling rate and the bit resolution used to represent the pulse duration is the bit depth. As an example, audio might be recorded with a 16KHz sampling rate and have 16 bit depth, meaning that there are 16,000 bins per second and the width of a pulse has 2^16 different possible values.



(a) (b) (c)

Figure 1: A sawtooth signal (a), quantized with variable output power (b), or with PWM (c). The width of the pulse in (c) should have area as close as possible to the area of the original signal. The number of bits used to specify the pulse width (the bit depth) determines the quality of the reconstruction.

Part 1: Set 10KHz Samples with 1/4 Width Pulses (35%)

In part 1 we will use TA0CCR0 to set a sampling rate and bit depth while using TA0CCR1 to set the pulse width. Our goal will be to have a sampling rate of 10KHz and a bit depth of about 10 bits for the pulse width. We will set the bin width with TA0CCR0, so with a clock speed o 16MHz we want TA0CCR0 to have a value of 1000. This results in a sample rate of 10KHz with 1001 different values per sample for the pulse width, or about 2^10. We will use TA0CCR1 to set the pulse width, so TA0CCR1 will have a value from 0 to 1000. At the start of every bin we turn on an output pin. Once the TAR timer register reaches TA0CCR1 we will set the pin low. Thus the ratio of TACCR1 to TACCR0 sets the fraction of the bin that the pulse is on.

Setting up TA0CCR0

We need to set the clock source in TA0CTL. We want to use "up" mode to count to the value in TACCR0 and use SMCLK as the clock source. We also want to run the CPU as fast as possible so we will set the clock speed to 16MHz and will not divide the clock source for Timer A.

Setting up those registers will look like this:

```
BCSCTL1 = CALBC1_16MHZ;
DCOCTL = CALDCO_16MHZ;
TA0CTL = MC_1 | ID_1 | TASSEL_2;
```

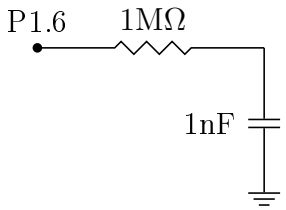


Figure 2: A basic reconstruction filter at the output of Pin 1.6. Measure across the capacitor to observe the reconstructed analog signal. The exact values of the resistor and capacitor are not vital to this lab. Choosing any resistor and capacitor pair with about the same RC time constant is sufficient.

Setting up TA0CCR1 for PWM

The MSP430 natively supports PWM on some output pins. We will configure pin 1.6 for output and turn on P1SEL for that pin. This will make P1.6 output a pwm signal configured by the value in TA0CCR1 once we set the output mode in the control register. Set the TA0CCTL1 register to OUTMOD_7 to set it for PWM output. Since P1.6 is one of the LEDs on the launchpad, you will see it light when this runs. Set the TACCR1 to one quarter of the value in TACCR0 and you should see the LED on 1/4 brightness. You can verify the timing on the oscilloscope. The control registers should be as follows:

```
TA0CCTL1 = OUTMOD_7;
P1DIR |= BIT6;
P1SEL |= BIT6;
```

Verify your bin width with an oscilloscope and record the image. Upload it to sakai along with your code.

Part 2: Building a Reconstruction Filter (15%)

To generate analog output we need a reconstruction filter at the output of the MSP430. We will actually use a simple low-pass or integrating circuit in this case, pictured in Figure 2. Measuring across the capacitor will yield the reconstructed analog signal.

After building the filter capture the output of your 1/4 width pulses using the oscilloscope. Upload a picture with at least one full pulse clearly visible to sakai along with your code.

Part 3: Generating a Sawtooth Output (25%)

To make a sawtooth output you will need to change the value of TA0CCR1 during each bin. The value of TACCR1 will go lineary from 0 to TACCR0, with the total transition settings the sawtooth period. Generate sawtooth signals with periods of one half second. There are 10,000 samples per second, or 5,000 samples per sawtooth. You will need to scale the current sample number down (by dividing) so that the range 0 to 4,999 is mapped into 0 to 1000 (the possible pulse width values).

You should see your LED increasing in intensity and then suddenly turning dark, twice per second and a clean sawtooth signal on the oscilloscope. Upload an image of this waveform to sakai along with your code.

Part 3: Making a Sine Wave Output (25%)

Repeat the previous task with a sawtooth for the sin wave. If we were using the sin function directly, it goes from -1 to 1 so to convert that into bins you would convert the value to the range 0 to 2 and then multiply it by half of the maximum binwidth, giving you a signal centered at $1/2Vcc$ rather than 0.

However, the sin function is too slow to calculate on the MSP430. Instead of using the sin function, create a lookup table as described in the previous lecture. The lookup table will map an index value onto a value of the sin function. You will not be able to generate a lookup table with 5,000 points; that many integer would consume more memory than is available in the MSP430G2553. Instead, a lookup table with only hundreds of points is good enough. Multiple samples will have the same value. For example, for a lookup table with 100 points you would switch from one sample to the next every 50 times the interrupt fires.

You should see your LED fading in and out when your sine wave output is working. Capture an image of the sin wave with a 1/2 second period and upload an image of this waveform to sakai along with your code.

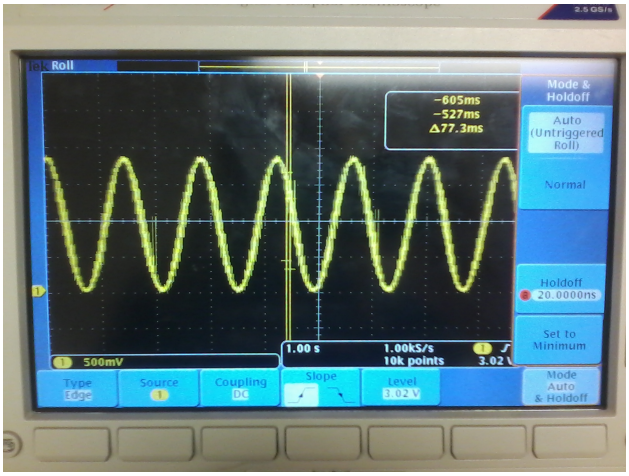


Figure 3: Smoothed output of the sine wave from Part 3 using a reconstruction filter described in Part 4.