Lab 4 – Data Conversion: Analog-to-Digital and Digital-to-Analog

1 Introduction

PSoC, along with Arduino and many other MCUs, provides multiple digital-to-analog and analog-to-digital converters. These data converters can be critical in many applications for reading sensors, driving actuators, gauges or lights or for enabling real-time digital signal processing.

The Nyquist theorem tells us that we can sample a bandlimited continuous time signal into a discrete sequence of values and attain perfect reconstruction back into continuous time as long as the sampling frequency, f_s, is at least twice the frequency of the highest frequency component present in the input signal. Let the frequency of the highest frequency component be represented by B, then we require that:

$$f_s \geq 2B$$

In practice no signals are perfectly bandlimited (How can I be so sure of that? Look it up!), and some spectral overlap typically occurs. A well designed system will manage the degree of spectral overlap to meet realistic specifications.

The theory of reconstructing continuous-time signals from discrete signals normally assumes impulsive reconstruction (i.e. brickwall lowpass filter applied to impulse train, where heights of impulses correspond to sample values), with lowpass filtering to remove spurious "image" frequencies above $f_s/2$. In practice reconstruction is accomplished by a Digital to Analog Converter (DAC), which performs a stair-step reconstruction, not impulsive reconstruction. The stair-step approach is called a "zero-order hold" function, and effectively imparts some lowpass filtering (along with some undesired roll-off in the band of interest). Additional lowpass filtering is required to minimize the step edges (which is equivalent to saying "remove spurious image frequencies") and attain good quality reconstruction. When high reconstruction accuracy is needed one also applies a certain mid-band boost to compensate for the zero-order-hold-type lowpass filter.

This lab will give you a first-hand look at the effects of sampling and aliasing, and reconstruction filtering. You will use an analog-to-digital converter (ADC) and a digital-to-analog converter (DAC) built into the PSoC chip (they're also built into most other popular microcontrollers). You will observe sampling and reconstruction of low amplitude signals, and signals close to, and over, the Nyquist limit.

You will gather statistics of a 1 KHz periodic signal, and use those to discern whether it is a square wave, sine wave or triangle wave, and display your results on the character LCD.

2 PROCEDURE

You will design a PSoC based system to perform real-time digital signal processing. One goal will be to receive an unknown signal with 1 KHz frequency from a signal generator via the PSoC SAR ADC (successive approximation analog-to-digital converter), and determine the following signal attributes:

a) Minimum and maximum values
b) Mean value
c) Mean-squared values, after subtracting the mean
d) Whether the signal is a sine, triangle, or square wave

1) Paper-and-pencil/MATLAB analysis:
   a. Find the Fourier series representation of a triangle wave, and square wave, at 1 KHz. Comment whether sampling at 100 KHz would be expected to preserve signal integrity.
   b. Given the peak amplitude of the wave, A, compute the normalized mean-squared value for a sine wave, triangle wave and square wave (after removing the DC, i.e. average, component). You can use integrals of continuous-time signals to approximate the summations that will be computed in your system.

      Integral form: $MS(\text{continuous}) = \frac{1}{TA^2}\int_0^T \left(f(t) - \bar{f}\right)^2 dt$

      Discrete form: $MS(\text{discrete}) = \frac{1}{NA^2}\sum_{k=0}^{N-1}\left(f(k) - \bar{f}\right)^2$

      Note that the discrete form does not explicitly sum over a single period – you will sum over many periods so that effects of sampling a non-integer number of periods is minimized.

2) Investigate the various data conversion components available in PSoC Creator. Build a PSoC project that includes a Successive Approximation analog-to-digital converter (SAR ADC) and a voltage digital-to-analog converter (VDAC). Route your signals through analog pin types. Avoid using pins P0[2] and P0[4] (since Cypress has placed 1 μF capacitors on those pins.

   For the ADC, select an input voltage range of 0 to 2.048 volts, and internal reference, bypassed. Configure your ADC for 12-bits and a conversion rate of 600 Ksps, or greater. Sample mode should be "hardware trigger" and clock should be "internal". For the VDAC, choose the "high-speed" output option, data source is CPU, strobe external.

   Read the datasheets for the ADC and DAC to understand the format of the data returned by, or sent through, the API. You will have to scale 12-bit ADC data to fit into the 8-bit DAC range! Note that the values that you will manipulate should all be positive, and that the voltages input and output to/from PSoC must all be positive (but not greater than V_{DD}).

3) Add a clock signal to your design, and configure it to a frequency of 100 KHz. Wire your clock to the "Start of Convert" pin of the ADC and the strobe pin of the DAC.
4) Build your design to get software API support for your schematic components.
5) Read the datasheets for the API regarding reading and writing the SAR ADC and VDAC. You should use the routines:

`ADC_SAR_1_IsEndConversion()`

This call can be blocking on non-blocking, based on a parameter passed. You could use either. Which do you choose, and why, and how?

`ADC_SAR_1_GetResult16()`

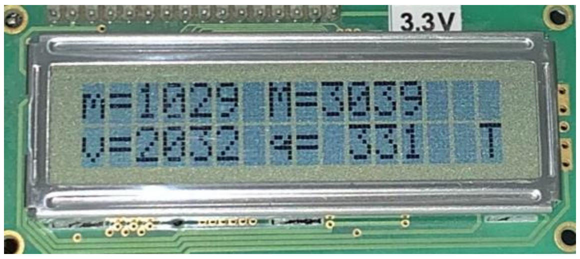
What format does this return – how to interpret the result?

`VDAC8_1_SetValue()`

How should the input data be represented?

In `main()` declare the variables:

```
int32_t sum;            // running sum, zero it out each 100,000 samples
int32_t saved_mean;     // gets sum/N at the end of each batch
int64_t sum_of_sqs;     // running sum of (f(n) - saved_mean)²
int32_t min;            // capture min(f(n)) reset to INT_MAX each batch
int32_t max;            // capture max(f(n)) reset to INT_MIN each batch
```



Your code should:

```
loop forever:
    reset sums, min, max
    for a batch of 100,000 samples:
        wait for ADC data available
        read ADC
        scale value appropriately, and write to DAC
        update values for sum, sum-of-squares, min, max
    compute saved_mean
```

Why did we use **`int64_t sum_of_sqs`**?

Use your waveform statistics (mean square vs. peak-to-peak) to determine whether the wave is sine, triangle or square). Update the LCD to look like the picture, using the following legend:

m = minimum	M = maximum
v = average	q = mean square

T=TRIANGLE	S=SINE	Q=SQUARE

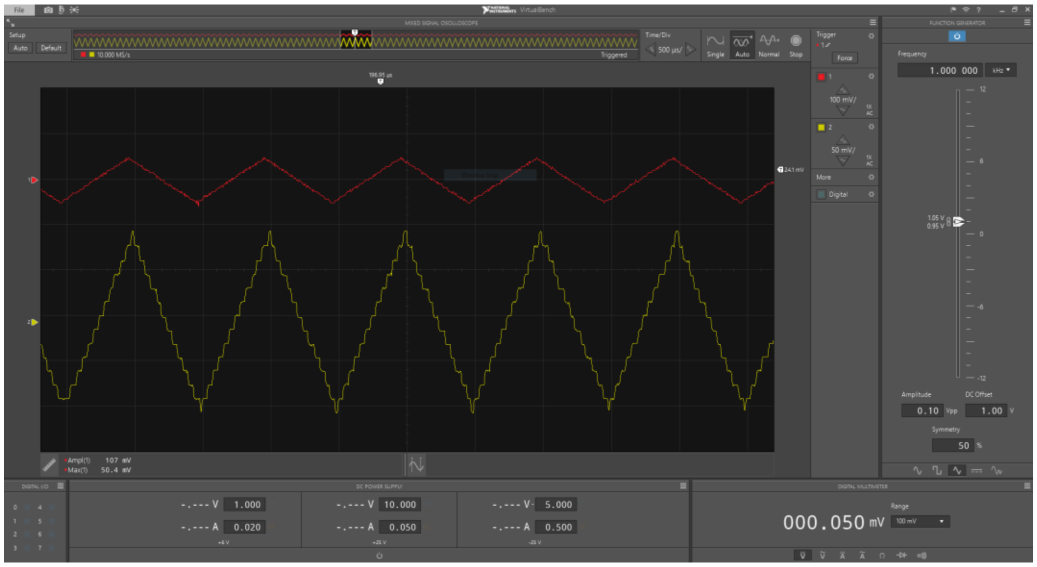
To fit the display, scale as follows:

```
int64_t mean_sq;                            // Need 64 bits to avoid overflow
mean_sq = sum_of_sqs / NUM_SAMPS;           // Normalize by number of samples
mean_sq = 1000*mean_sq/SQR((max - min)/2);  // Normalize amplitude, so 1.0 → 1000
```

6) Test your design with sine, triangle and square waves at 1 KHz, with peak-to-peak swings of 0.2V, 1 V, and 2 V.

   Take a screenshot of the scope showing input and output waveforms, and take a picture of your LCD display, under the 9 conditions.

7) Challenge yourself: with a sine- or triangle-wave input, investigate the effect of increasing frequency (up to and beyond half the sampling rate), increasing amplitude beyond 2 V peak-to-peak, and decreasing the amplitude, e.g. down to 50 mV peak-to-peak.


Time/Div 500 µs/
10.000 MS/s
1: 100 mV/ 1X AC
2: 50 mV/ 1X AC
Frequency 1.000 000 kHz
Amplitude 0.10 Vpp
DC Offset 1.00 V
Symmetry 50 %
Ampl(1) 107 mV
Max(1) 50.4 mV
-.--- V 1.000
-.--- A 0.020
-.--- V 10.000
-.--- A 0.050
-.--- V 5.000
-.--- A 0.500
000.050 mV
Range 100 mV

3 NOTES

1) You will need to pay attention to the signal amplitude and offset. The PSoC ADC cannot handle negative signals, and it has a limited positive range.

4 WRITE A REPORT

The report is to include, but not limited to the following:

a) **(1.5 points) Cover Sheet** with Title, Class, Names, etc.
   For each member of your group – list the name, and how you contributed to each part of the work – theory, coding, lab, and writeup.
b) **(0.5 points) Introduction** – describe the technology and procedures involved in this lab.
c) **(0.5 points)** Brief recap of **Procedure**. You can simply distill down Part 2, above.
d) **Design** information – explain your choices and give full documentation:

a. **(0.5 points)** PSoC schematics
b. Equations:
    i. **(0.5 points)** Fourier series for triangle wave and square wave
    ii. **(0.5 points)** Mean-square values for sine, triangle and square waves
c. **(1 points)** Timing diagram. **For this lab you must also include a timing diagram that shows, at least, the clock, the EOC signal, the software polling loops, ADC read and DAC write events, and the DAC output. The timing diagram x-axis is time, and you will show multiple scope-like traces that describe signals or events.** For an example of a hardware/software timing diagram see http://zone.ni.com/reference/en-XX/help/370466Y-01/mxcncpts/controlappcase6/. Just show two ADC/DAC sample periods. You **must show curvy arcs** that indicate causality – show which event leads to what action. I will show examples in lab of what I expect here! If you're not sure what I expect, ask, and ask me to check your diagram before you submit it.
d. **(2 points)** Paste in all of your code. Use Courier New font, 8 or 9 points and format to eliminate line wrapping.

e) **(2 points) Results – waveforms and LCD displays for all 9 cases. Put captions on figures explaining conditions, and stating volts/div and time/div! Record measurements (e.g. values from your LCD display) in tables.**
f) **(0.5 points)** A **Conclusion** including discussion of your work and anything you found especially interesting or not expected.
g) **(0.5 points)** Overall professionalism

REPORT NOTES:

- One report per team
- You may use the IEEE paper format, if you like: (template is 2014_04_msw_usltr_format.doc). In this case the cover sheet info is embedded at the top.
- Microsoft Word .docx file