LIGO-T1300492: Notes on upsampling in CDS

Tobin Fricke

May 22, 2013

1 Upsampling

While the CDS "user models" may run at sample rates of 2048 Hz, 4096 Hz, 16384 Hz, 32768 Hz, or 65536 Hz ("64 kHz"), the digital-to-analog converter itself is always run at 65536 Hz. In order to increase the sample rate of the data stream, we must generate the "missing" samples. Two approaches[1] that are in use are:

1. "Sample-and-hold." The most recent sample is simply repeated to produce the higher rate signal. For example, after upsampling by a factor of 4, the signal [1, 2, 3, 4] becomes [1, 1, 1, 1, 2, 2, 2, 2, 3, 3, 3, 3, 4, 4, 4, 4].

2. "Zero-padding." Here we simply insert zeros in order to increase the sample rate. For example, after upsampling by a factor of 4, the signal [1,2,3,4] becomes [1, 0, 0, 0, 2, 0, 0, 0, 3, 0, 0, 0, 4, 0 0 0].

After performing the upsampling[2] by one of these two means, we filter the new signal through an anti-image (or, perhaps more properly, "interpolation") filter. The job of this filter is to smooth out the signal, ideally removing all spectral content from between the old Nyquist rate and the new Nyquist rate.

Sample-and-hold is in some sense the simplest, most obvious approach. Its disadvantage is that it adds phase equivalent to half a sample delay. For example, using sample-and-hold to upsample from 2048 Hz to 64 kHz adds a delay of (2/(2048 Hz))=1 ms. On the other hand, the general approach of zero-padding followed by interpolation allows a flexible trade-off between phase-loss[3] and production of spurious signals. It has a few surprising features that have caught more than one user off-guard.

CDS currently uses zero padding by default. Sample-and-hold may be chosen instead by putting the option "`no_zero_pad=1`" in the Simulink model parameters block.

2 Strange effects

One of the most unexpected effects of the "zero padding" approach to interpolation is that a constant output from the user model *does not* produce a constant output to the DAC.

After zero-padding, any signal with a significantly nonzero mean value will contain potentially severe harmonics of the old sampling rate. It's the job of the interpolation / anti-image filter to get rid of them. I contend that the only way to completely eliminate these unwanted harmonics is to use sample-and-hold.

An example showing the time series and spectra resulting from a "constant" output value is shown in figure 2.

[1] It turns out that "sample and hold" is just a special case of "zero padding" with a particular choice of interpolation filter.

[2] Of course, on the input side we must also downsample from 65536 Hz to the rate of the user model. CDS currently uses the same filters for both upsampling and down samping.

[3] Phase loss can also be reduced simply by running the model at a faster rate.

3 Filter coefficients

The digital filters in CDS are implemented in second order sections (SOS). A second-order-section implements one or two poles and one or two zeros. Each second-order-section is described by a gain term and a set of coefficients, implementing the z-domain transfer function

$$H(z) = g\frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{1}$$

In CDS, the leading coefficients a_0 and b_0 are assumed to be equal to one. For each filter module, we store four coefficients per second order section and one overall gain term for the whole module. Each filter module can contain ten(?) second order sections. The coefficients stored in the Foton file are $\{a_1, a_2, b_1, b_2\}$. Until recently these coefficients were also used directly in the filter implementation.

Recently we switched to a new filter topology that we call "biquad"[4] which is implemented using a set of coefficients called $\{a_{11}, a_{12}, c_1, c_2\}$. This filter topology nominally implements the same digital filter, but in a manner that introduces less numerical noise. The biquad coefficients may be derived from the direct form coefficients via the following mapping (G0900928, page 11):

$$\begin{aligned} a_{11} &= -a_1 - 1 \\ a_{12} &= -a_2 - a_1 - 1 \\ c_1 &= b_1 - a_1 \\ c_2 &= b_2 - a_2 + b_1 - a_1 \end{aligned}$$

The inverse map is:

$$\begin{aligned} a_1 &= -a_{11} - 1 \\ a_2 &= a_{11} - a_{12} \\ b_1 &= -a_{11} + c_1 - 1 \\ b_2 &= a_{11} - a_{12} - c_1 + c_2 \end{aligned}$$

If biquad filters are enabled, then the coefficients are translated into biquad form as the Foton file is read, in `drv/fmReadCoeff.c`[5]. However, the upsamping/downsampling filter coefficients must be given verbatim in the C code in the correct format (direct-form or biquad).

4 Where to find the upsampling/downsampling filters

The coefficients for the interpolation filters are found in `src/fe/controller.c`, in variables called `feCoeff2x`, `feCoeff4x`, `feCoeff16x`, and `feCoeff32x`, giving the coefficients used when up- or down-sampling by factors of 2, 4, 16, and 32, respectively.

For example, here is one set of coefficients found in that file[6]:

```
/* Coeffs for the 4x downsampling (16K system) filter */
```

[4] Much to my confusion, the filter topology used is *not* the filter topology you find when you google for "biquad". For example, the filter topologies on the "Digital biquad filter" Wikipedia entry are *not* the "biquad" topology we are using. "Biquad" is just a synonym for "second order section," indicating that both the numerator and denominator of the transfer function are quadratic (in z since we are talking about digital filters).

[5] I suggest moving the coefficient-conversion routine to its own function, both (1) so that it doesn't clutter the Foton-file-parsing code, and (2) so that the digital anti-aliasing/anti-image filters (i.e., `feCoeffxxx`) can be stored in the C code in one canonical format and converted to the other as needed (i.e. "don't repeat yourself"). Currently the direct form coefficients and the biquad-form coefficients are stored independently and chosen by `#ifdefs`. It's not immediately obvious from looking at the C code that they implement the same filter.

[6] I suggest we add comments to controller.c indicating how these filters were designed, or even perhaps even read them from a Foton file so that the design can be inspected in Foton.

```
static double __attribute__ ((unused)) feCoeff4x[9] = {0.014805052402446, 0.7166258547451800,
-0.0683289874517300, 0.3031629575762000, 0.5171469569032900, 0.6838596423885499, -0.2534855521841101,
1.6838609161411500, 1.7447155374502499};
```

The first number in this array is the gain factor for the entire filter. Filter coefficients come next, in groups of four. Here there are coefficients for two second-order-sections. If `CORE_BIQUAD` is defined, then each group of four coefficients is interpreted as $\{a_{11}, a_{12}, c_1, c_2\}$. If not, then they are interpreted as $\{a_1, a_2, b_1, b_2\}$. This particular sample comes from a region surrounded by `#if defined(CORE_BIQUAD)`, so these are biquad coefficients.

5 Properties of Sample-and-Hold

It seems intuitive that the sample-and-hold technique would be the only upsampling technique that introduces no ripple. To analyze the effect of sample-and-hold, we can think of sample-and-hold as being implemented by zero-padding followed by a filter that implements the sample repetition.

For example, to upsample a 2048 Hz signal by 32×, a finite-impulse-response (FIR) filter consisting of 32 ones (aka a rectangle or boxcar) would do the job. We can learn about most of the properties of sample-and-hold simply by looking at the transfer function of this filter

This FIR filter consists of a rectangle that starts at $\tau = 0$. The frequency response of a rectangle centered on the origin is the sinc function. So the frequency response of sample-and-hold (a delayed rectangle) is a sinc function combined with a delay of one-half the slower sampling period[7]. It turns out that the nulls of this sinc function are at exactly the harmonics of the lower sampling rate: 2048, 4096, 6144, 8192, 10240 Hz, ... up to the new Nyquist rate.

It appears that any interpolation filter that kills all of these harmonics will end up looking something like "sample and hold." (While the sample-and-hold takes care of the harmonics of the original sample rate, we still need to use an anti-image filter to take care of broadband noise, since the envelope of the sinc function only goes like $\sim 1/f$.)

6 The Analog Anti-Aliasing/Anti-Image filter

The job of the digital interpolation / anti-alias / anti-image filter is to remove frequency content between the old and the new Nyquist rates after converting from one sample rate to another. The job of the analog anti-alias / anti-image filter is to remove frequency content above the Nyquist rate of the ADC, i.e. everything above 32 kHz. Additionally, the noise of the analog filter should be less than the noise of the ADC/DAC.

The LIGO analog AA/AI filter is implemented as a third-order low-pass at 10 kHz combined with a twin-T notch at 64 kHz (D070081; also 10-meter logbook page 550).

References

- Matt Evans. "Digital Filter Noise: Why does the textbook tell us not to use direct form 2?" `https://dcc.ligo.org/LIGO-G0900928-v1`

7 Appendix: Frequency response of a SOS

The transfer function of a second-order-section is given by equation 1 in terms of the z-transform parameter. To plot the frequency response, use the relationships $z = \exp sT$where $s = i\omega = 2\pi i f$ is the frequency and $T = 1/f_s$ is the time interval between samples. This is implemented in the Matlab function `sos2freqresp.m` available from `https://gist.github.com/tobin/1141120` and also the built-in Matlab function `freqz`.

[7] Plot it in Matlab with `bode(tf(ones(1,32)/32, 1, 1/65536));`

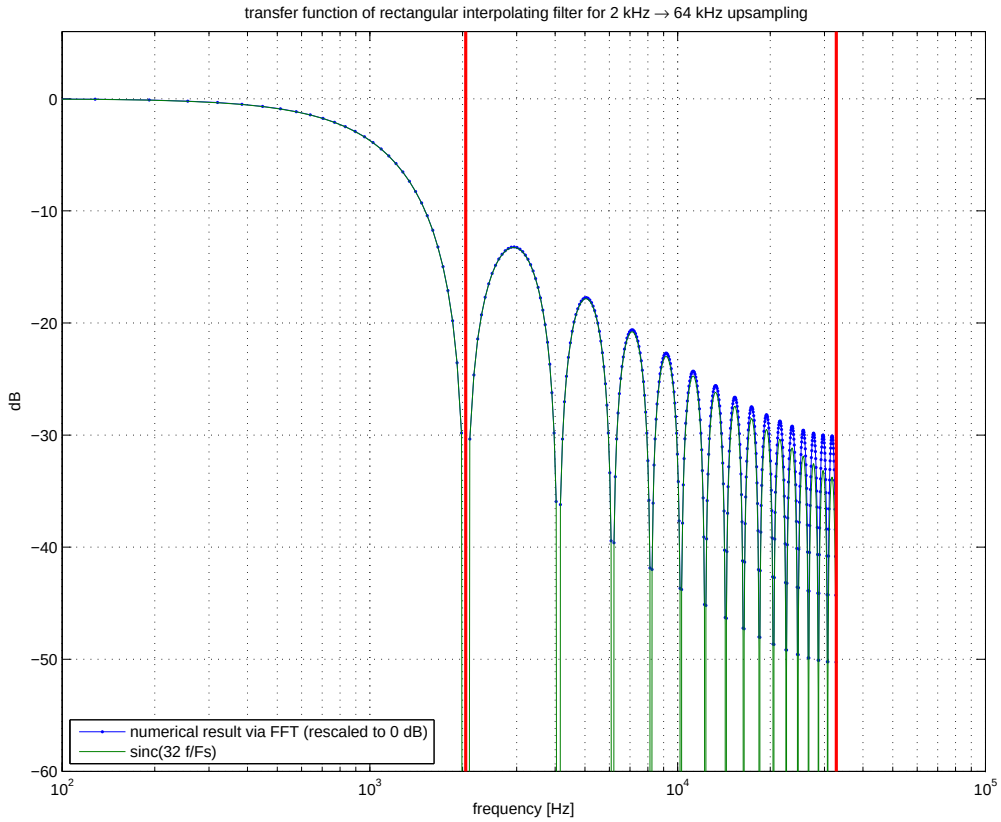

Figure 1: Transfer function of "sample and hold". The old and the new Nyquist rates are indicated with red vertical lines. Figure generated by `rect_FIR.m`.

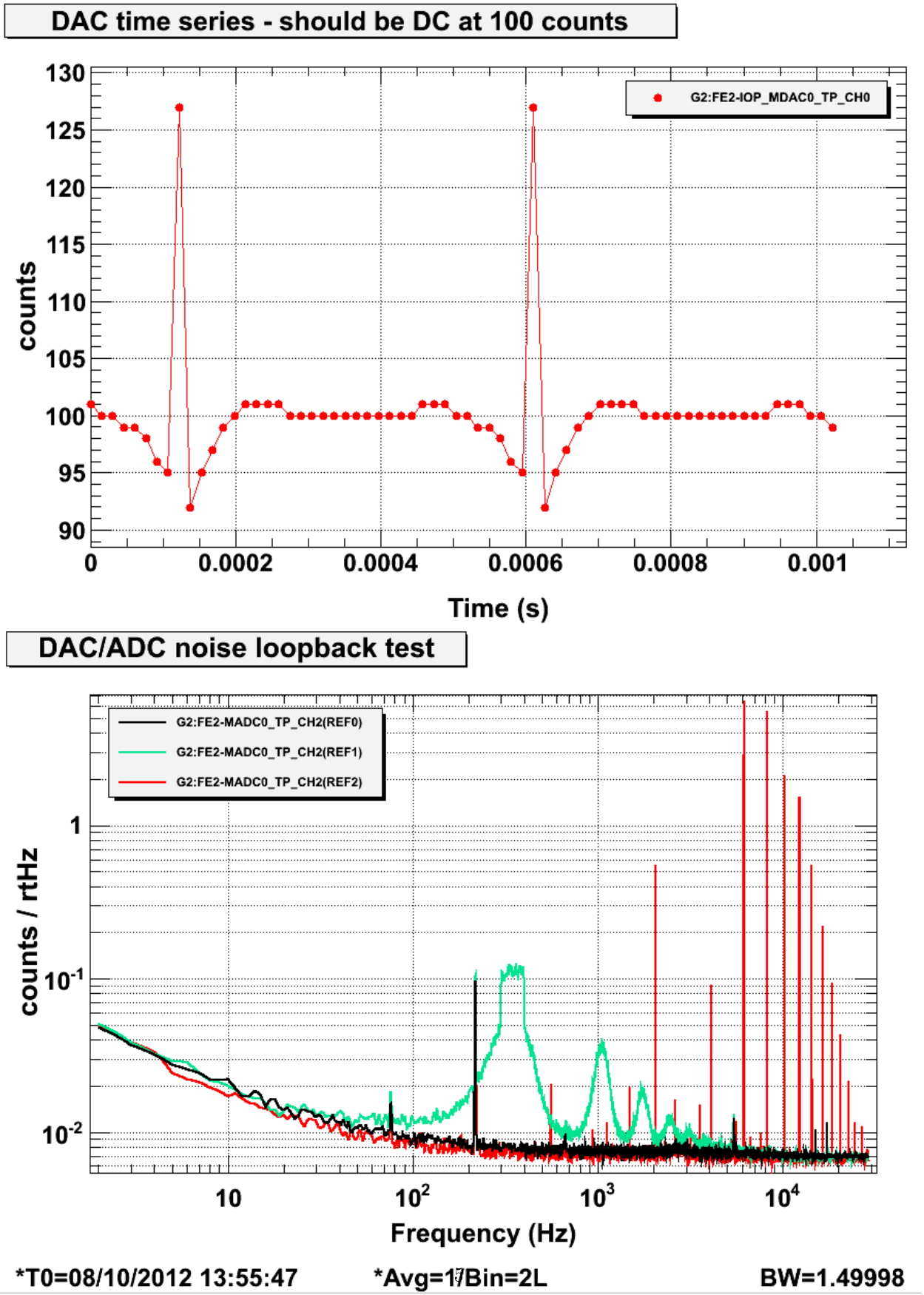

Figure 2: Example time series and spectra showing 100 count DC signal (red) after zero-padding and smoothing with our current digital anti-aliasing filter. The spectra were measured through DAC to ADC loopback. The green trace shows what happens to band-limited noise. 10 meter prototype logbook page 840.