Analysis of Digital Filters for the Readout Card.
SCUBA2 Multichannel Electronics Technical Report.

Mark Halpern, 28 Sept. 2003
Latest revision, September 28, 2003

Introduction The readout cards in the MCE sample data from the arrays at 20 kHz and are polled for output data at 200 Hz. The detector time constants are of order 1 ms, so the 200 Hz reported data rate is sufficient. The rapid readout rate is dictated by a desire to limit the effects of out of band noise in the front end amplifiers looking at the squid outputs, and also to provide high bandwidth feedback signals for biasing the squids. Filtering the 20 kHz data so that there is little power above the nyquist frequency associated with the 200 Hz data reporting rate is performed by the readout cards. In this note details of an infinite impulse response filter (IIR) and of a finite impulse response filter (FIR) which meet are needs are described. The purpose is to present a plausible filter design so that we may decide if the electronics hardware and firmware are sufficient to the task. No attempt has been made to optimize either filter, and the final filter may well look somewhat different.

Both filters described here run all the time that the electronics are collecting data from the array. Both filters produce an output which is sampled at 20 kHz and is available at the end of any complete cycle through the 41 addresses of the array. This allows sub-sampling of the filtered output to be essentially instant while leaving control of the precise timing up to an external trigger, provided by the Data Valid Pulse.

The IIR filter is simpler to implement since it involves much less storage of data, and no storage external to the RC FPGA. The FIR filter has potentially a shorter impulse response function. Our intention is to implement the IIR in the first round of prototypes, but the schematics and layout of the RC allow for the external memory the FIR would require.

In this analysis, we assume that 32 bit numbers are used throughout. While 24 bits would probably be sufficient, two reads or stores of 16 bit words is likely faster than three reads or writes at 8 bits, so if external (but on the card) memory is required, 32 bit arithmetic is simpler.

Infinite Impulse Response Filters A single RC filter can be mimicked digitally in one step by accumulating a running sum of the input data. For a given input data set $data[m]$,

$$f1[m] = data[m] + \alpha * f1[m-1] \tag{1}$$

is equivalent to convolution with a single exponential with time constant $\alpha = 1.-1/(\tau f_{sample})$. This filter requires one add and one multiply per pole, and only requires storing one previous value of the filtered output, no matter how long a time constant is desired. For this reason IIR filters are often chosen for cases, such as ours, which require rapid filtering of a lot of

data in real time. IIR filters can be unstable if very short impulse response times or subtle passbands are required.

Since the purpose of this analysis is to produce a 'straw man' filter for which we can evaluate resource requirements, I have not implemented either a Butterworth or Bessel filter, but simply a stack of single pole filters. Altera offers recipes for IIR filter implementation on their devices and if more subtle filters are required we will devote the design effort the call for.

The *idl* code below produces filtered outputs corresponding to one, two, three, four and five pole filters.

$$\begin{aligned} f1[m] &= data[m] + \alpha_1 \times fi[m-1] \\ f2[m] &= f1[m] + \alpha_2 \times f2[m-1] \\ f3[m] &= f2[m] + \alpha_3 \times f3[m-1] \\ f4[m] &= f3[m] + \alpha_4 \times f4[m-1] \\ f5[m] &= f4[m] + \alpha_5 \times f5[m-1] \end{aligned} \tag{2}$$

This code requires one floating multiply and one add per pole per address cycle. If this is implemented without external memory on a readout card, the FPGA must store the previous values of $f1$ through $f5$ for each of 8×41 pixels. This requires

$$\text{Storage} = 32 \text{ bits per number } \times 8 \text{ Rows } \times 41 \text{Numbers per row } = 10,496 \text{ bits} \tag{3}$$

within the FPGA, which will not be hard to find. Assuming no caching of data within the FPGA, and two reads or writes per data point to get 32 bits worth of 16-bit words, this algorithm requires 20 reads and 10 writes for each of 8 ADCs every 50 μs address step. This is not a problem, even for the slowest memory we might employ.

The output of this algorithm is shown in Figure 1 as a separate graph of response against frequency for each pole of the filter.

The impulse response of the filter can be 'measured' directly in this simulation by setting the input to a single spike at time zero and looking at the output time series. The impulse response associated with each stage of the filter is shown in Fig. 2 for the first four poles. The impulse response has some width, as it must given the desired frequency response. If we were willing to store the filtered data at full sampling rate for roughly 22 ms, or 450 samples per pixel, we could report data which has nearly no delay with respect to the Data Valid Pulse. This would require on-board storage external to the FPGA and introduce ambiguity as to which DVP to associate with which data package. We prefer that delaying the data to match the DVP be handled in software on the linux PCs instead. This issue arises identically for the FIR filter too.

Finite Impulse Response Filters We have simulated the Finite Impulse Response filters we use on BLAST (the Balloon-borne, Large Aperture Stratospheric Telescope). Blast

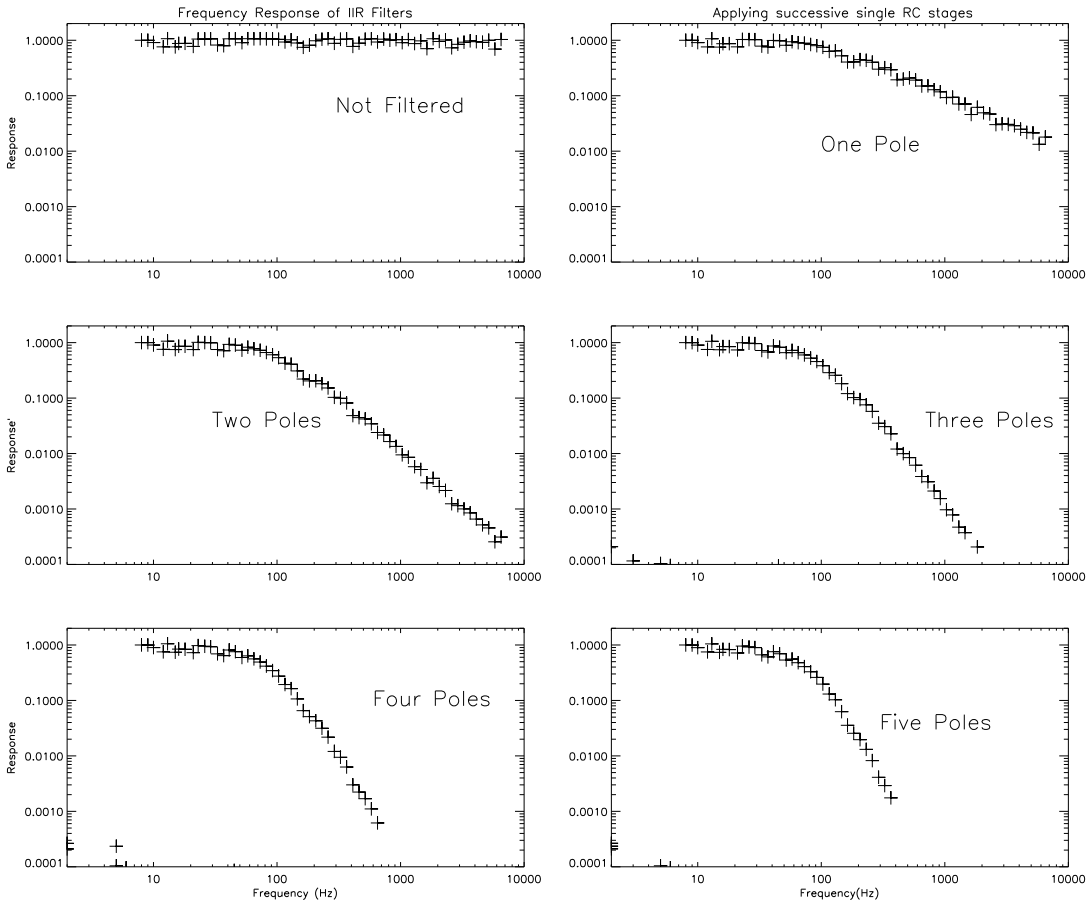

Figure 1: **IIR Filter Frequency Response** The frequency response of the stack of single 'RC' IIR filters is shown. In each case, the response has been normalized to unity at low frequency. The program which generates one minute of data, simulates the filters in Eq.2, calculates the frequency responses and makes the plot above runs in a few seconds on my G3 laptop.

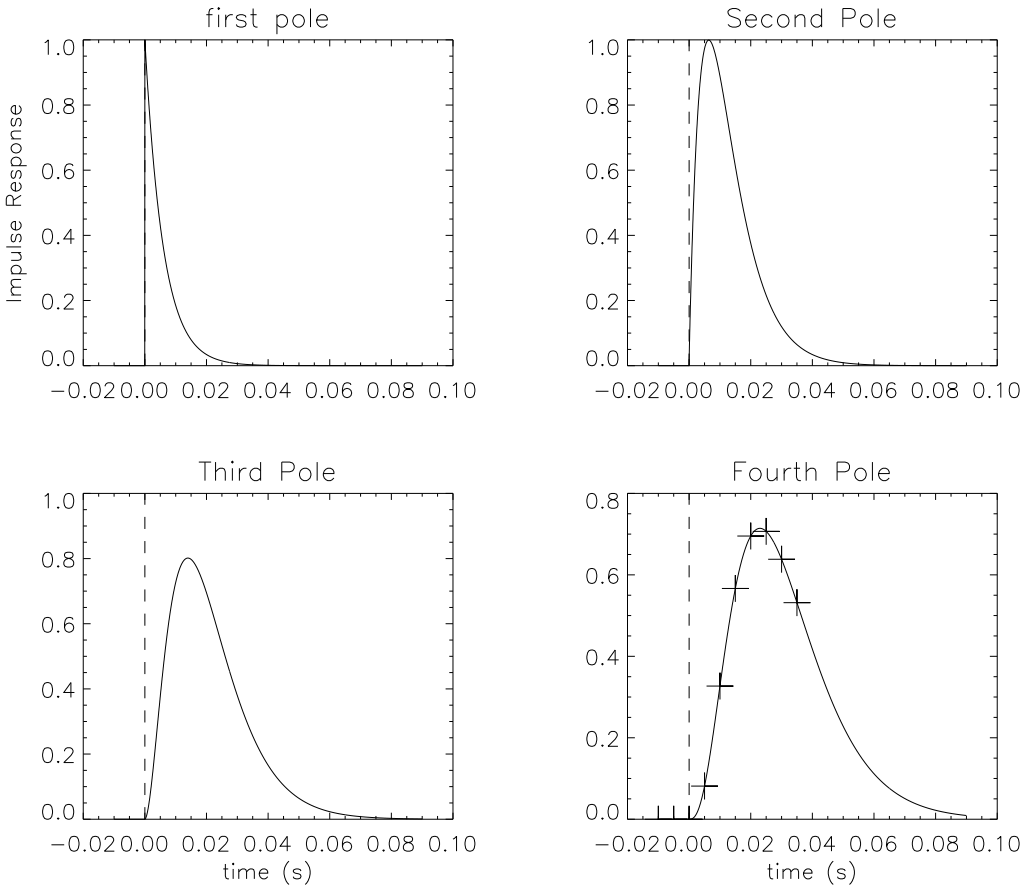

Figure 2: **IIR Filter Impulse Response** The impulse response of the stack of single 'RC' IIR filters is shown. The smooth underlying curve shows the impulse response at 20 kHz, while the plusses in the fourth box show the locations of 200 Hz samples. If the vertical dashed line is the time at which the output is reported, the smooth curve gives the weight associated with detector signals observed at *previous* times.

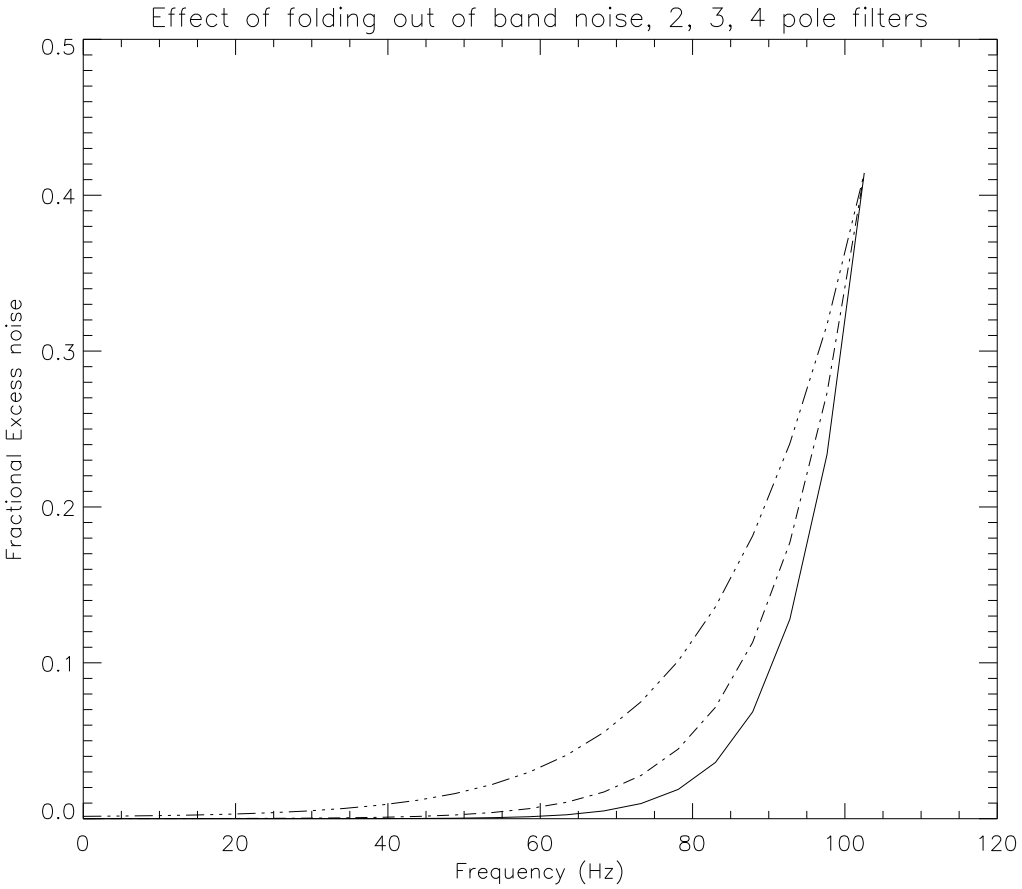

Figure 3: **Contribution of excess noise from beyond the folding frequency.** In any measurement in which the noise is white and extends beyond the Nyquist frequency of a given sampling strategy, the last valid point in the spectrum has $\sqrt{2}$ excess noise. The first data point beyond folding is added incoherently and has the same amplitude as the last valid spectral element. If the data audio filter has a steep slope, the amount of excess noise added due to folding drops rapidly away from f_{nyq}. This figure shows the excess noise associated with each of the impulse responses shown in Figure 2.

employs arrays of conventional NTD Ge spider-web bolometers which are coupled to AC amplifiers. The data are sampled rapidly, but only telemetered to ground at 100 Hz. A single card handles about 25 detectors with a smaller FPGA than we use. The filtering itself is performed in aDSP chip dedicated to the purpose. The SCUBA2 FPGAs can handle the computation involved if extra on-board SRAM is added to store the data and intermediate filter products.

The BLAST algorithm, suggested by Barth Netterfield, is a convolution of four box-cars. It has the advantages of needing no multiplication and of providing quite a narrow impulse response function.

The box-based FIR filter is simulated in *idl* with the code

$$\begin{aligned} f1[m] = &\quad fi[m-1] + data[m] - data[m-w_1] \\ f2[m] = &\quad f2[m-1] + f1[m] - f1[m-w_2] \\ f3[m] = &\quad f3[m-1] + f2[m] - f2[m-w_3] \\ f4[m] = &\quad f4[m-1] + f3[m] - f3[m-w_4] \end{aligned} \tag{4}$$

where the set of box widths is w_n. Notice that the boxes are *not* centered; they share an edge at the most recent sample. The values of w_n are chosen to put nulls of $\sin x/x$ logarithmically uniformly through the first octave in frequency above the nyquist frequency. For SCUBA2 sampling at 200 Hz this implies $w = [119, 140, 168, 200]$, and those are the box widths used below. I doubt this choice is ideal, as the resulting frequency response is flat through this octave and a superior filter has a steep slope at the nyquist frequency.

This filter requires storing sufficient data to subtract the back end of each box. For these values of w_n, 627 32-bit numbers are required per pixel. ($Data$, $f1$, $f2$, and $f3$ are stored. $f4$ is not needed.) The total required data storage is

$$\text{Storage} = 32 \text{ bits per number} \times 8 \text{ Rows} \times 41 \text{Numbers per row} \times 627 \text{ values per cycle} = 6.58 \text{ Mega} - \text{bits} \tag{5}$$

which will certainly not fit in the FPGA, but fits easily in the 2M-byte SRAM which the RC layout allows.

This filter requires eight adds per pixel per cycle, and no multiplies. Four numbers are stored and eight are read, not including the need to shift the whole stack in memory. (A ring might be nice.) Thus, every 1.2μs, 8 ADCs x 12 Operations x 2 Words/Operation = 192 Reads or Writes occur. This requires a shorter access time than we had planned for the SRAM, 6ns instead of 8ns. However, we can cache one full image as we do for the IIR filter. This reduces our storage requirements to a barely feasible level. Because of this timing requirement, this FIR filter is difficult and we will only implement it if a clear reason to do so emerges in teh prototype development cycle.

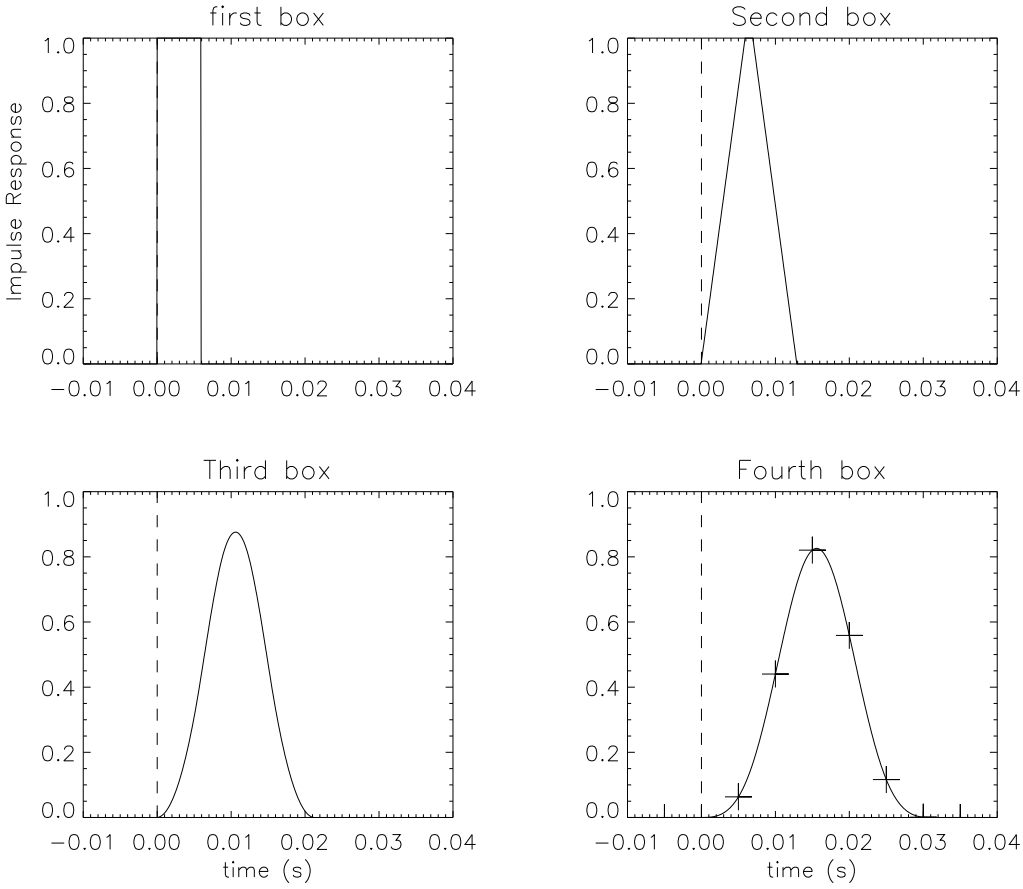

Figure 4: **Impulse Response Function for the Nested-boxes FIR:** Impulse responses are calculated as for the IIR, and the figures all have the same meanings as in Figure 2. Notice that the response function is about 5 ms narrower than for the four-pole IIR filter of comparable frequency response.