48-BIT INTEGER PROCESSING BEATS 32-BIT FLOATING POINT FOR PROFESSIONAL AUDIO APPLICATIONS

-

James A. Moorer
Sonic Solutions

Abstract:

The emergence of new consumer formats for the delivery of digital audio into the home places ever more pressure on professional systems for maintaining the highest possible audio quality throughout the production process. For a while, it was thought that the 32-bit floating point processors provided a simple and adequate solution to the issues of dynamic range and scaling inherent in professional audio processing. It is clear now that even more precision is required to produce a finished product of 24-bit PCM at sampling rates of 96 KHz to 192 KHz. This paper discusses the differences between fixed point and floating point computation with respect to the audibility of computational error, and makes a case for the use of 48-bit integer processing as the standard by which professional systems should be measured.

Introduction:

Floating-point has always been admired for its freedom from scaling and dynamic-range problems. It makes it possible to design digital filters without worrying about scaling problems in the state variables. Mixdown busses can be arbitrarily wide in terms of number of channels without fear of overflow. Since the 32-bit IEEE-752 floating point format allows 24 bits of mantissa (including the sign bit), it has been generally thought that this was sufficient for high-resolution audio. This is especially tempting since many floating-point DSP implementations include an ALU with 32-bits of mantissa for somewhat better resolution in the multiply-accumulate operation that is at the core of so many algorithms in professional audio processing.

Indeed, when we compare 32-bit floating point with 24-bit integer, there is some advantage to the floating-point. The resulting noise floor for 24-bit audio processing is of a constant level. It does generally not depend on the level of the signal. At low levels of audio, the noise floor may not be far below the level of the audio itself. In floating-point, the noise floor is directly related to the amplitude of the audio itself. If the audio is at a low level, the noise floor will be at a correspondingly lower level. This is generally a good thing, but there is some disadvantage since it is necessarily a changing, or “pumping” noise floor. The hope is that said noise floor is sufficiently low to be inaudible under all circumstances.

With recent improvements in DSP cost/speed performance, it is now reasonable to use integer DSP processing of two 24-bit words in parallel. We can perform all the calculations in a signal path at 48-bit precision. Of course, only words up to 24 bits can be exchanged with DA/AD converters, although any size sample from 24 through 48 bits (or more) could hypothetically be stored on a hard disk (with corresponding reduction in the audio capacity of the disk). It is worthwhile to compare 48-bit integer processing with 32-bit floating point and characterize the differences.

It is well-known that for linear processing, the error in floating-point computation is equivalent to error in the coefficients that multiply the signal. Analysis of error in linear computation then becomes an analysis of coefficient sensitivity. In integer computation, there is generally a tradeoff between roundoff error and dynamic range. One of the complications in integer computing is that this tradeoff must be considered at every stage of the process, thus placing additional burden on the designer. In floating-point, this tradeoff is made automatically, which can have unanticipated side effects as well.

In 48-bit integer processing, we can divide the word up into three portions: some number of high-order bits will be used for dynamic-range, or “headroom.” Some number of low-order bits will be used to keep track of a the fractional portion of the sample. These are sometimes known as “guard bits.” The remainder of the bits form the audio sample. One reasonable allocation of a 48-bit word would be a 32-bit audio sample with 8-bits of headroom and 8 guard bits. This would allow summing of up to 256 channels of audio without overflow.

In addition to the issue of the width of the data, we can also ask about the precision of the coefficients. If the coefficients can be represented as 24-bit integers, then we avoid the costly 48-by-48 multiply.

Filter State Variables

The issue of how the state variables of a digital filter are scaled attains considerable importance when we consider a practical filter implementation. Besides the usual problems of scaling integers in general, scaling of filter state variables can cause problems in floating point implementations as well. Anyone who reads a standard textbook on digital filtering may come away with the feeling that filter coefficients are constants that are set once and for all for the desired frequency response. They are not. Any real-world filter will have coefficients that change with time. In any implementation, this has the possibility of producing artifacts due to side effects of the scaling of the state variables.

We will look at three different filter realizations for a single second-order section. We will use the "transposed" direct form, as shown in Figure 1, the normalized lattice form shown in Figure 2 [1], and the state-variable form [2]. We start with the definition of a simple, second-order sections coefficients as follows:

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

With the direct form, it is well known that the error amplification at low frequencies can be significant. One filter form that is sometimes recommended for low frequencies is the Agarwal-Burrus form [3] shown in Figure 3 (this is actually a slight rearrangement of the original form). This consists simply of substituting integrators for the delays. This dramatically reduces the noise amplification at low frequencies, but increases it for center frequencies that are close to the Nyquist frequency. One could argue that this is a reasonable tradeoff for audio. It is especially attractive since the integrators can be implemented without multiplies. Unfortunately, this form is not stable. To see this, consider that the input is a constant (DC). After the transients die down, the output must be a constant as well. This means that there is always a non-zero constant value feeding each of the integrators. Regardless of the implementation, they will eventually overflow. Even before that, they will cease operating correctly when the ratio between the states of the two integrators exceeds the precision of the arithmetic, so that one or more of the terms no longer has any affect on the output signal. Note that one can "fix up" the filter by making the integrator a "lossy" integrator. This is easily done by substituting a different value in the feedback path than unity. As we vary the feedback coefficient from 0 to 1.0, we get a range of filters with a tradeoff between low-frequency response and dynamic range of the state variables. Furthermore, one could l2-normalize the filter by placing multipliers before each delay element and rescaling the coefficients. After all this, however, it is not clear that this form has any advantage, compared to the normalized lattice or state variable forms, so we will not discus it further at this time.

To show the differences among these filter forms, we will show the results of simulations done in Matlab. These use 64-bit IEEE floating point. This gives us a chance to compare the algorithms in a manner that is as independent of implementation as we can imagine. Whatever results we get from this simulation, we can be assured that any practical implementation on a DSP will perform worse than these simulations in one way or another. For input, white noise (random numbers) is used, since this excites all the modes of the filter. The two classes if difficulties we will examine are low-frequency, high-Q filters, and swept filters – that is, filters with changing coefficients.

Figure 4 shows the output of the transposed direct form with swept coefficients. If this were a 44.1 kHz sampling rate, the figure would represent about 91 milliseconds. For a filter, we choose a presence filter with an 18 dB boost. Over the first 1000 points, the frequency is set to 44.1 Hz and the bandwidth to 22.05 Hz. From point 1000 to point 3000, the center frequency is moved linearly to 441 Hz and the bandwidth to 220.5 Hz. They are held constant for the last 1000 points. There are two important things to note about this result: *even with 64-bit floating point, the filter noise totally swamps the response during the first 1000* *points!* The direct form is so noise-sensitive at low frequencies that even 64-bit floating point cannot save it. The next thing to notice is the oscillatory behavior during the sweep. This is a direct result of the lack of scaling of the state variables. At each change in the coefficients, the state may or may not represent a state That would be “reasonable” for that input, and consequently large deviations in the output can occur.

The normalized lattice form is often suggested, since it has a number of desirable qualities, especially for integer realizations. I enclose here the design equations for a complete second-order realization for convenience, simply because I have not seen them detailed anywhere else.

$$S_1 = b_2$$

$$S_2 = \frac{b_1}{1+b_2}$$

$$C_1 = \sqrt{1-S_1^2}$$

$$C_2 = \sqrt{1-S_2^2}$$

$$v_2 = a_2$$

$$v_1 = \frac{1}{C_1}(a_1 - a_2 b_1)$$

$$v_0 = \frac{1}{C_1 C_2}(a_0 - C_1 S_2 v_1 - S_1 v_2)$$

One might ask why we care whether the state variables are normalized in a floating-point realization. The reason is that the ratio of the levels in the state variables in the lattice form is unbounded. The tap gains (v0, v1, v2) are similarly unbounded in an unnormalized realization. Without normalization, the state variables can be so far apart that doing the sum for the output produces difficult cases like subtracting to large numbers to get a very small (and inaccurate) number. It is relatively easy to show that the sensitivity of the output to the coefficients (in the absence of normalization) is proportional to the reciprocals of the cosine coefficients. Consequently, any time these coefficients approach zero, the output becomes very sensitive.

Figure 5 shows the output to our swept coefficients using the same test simulation setup. Notice that we now see something “reasonable” during the first 1000 points. This is simply the impulse response of the filter in response to a suddenly-applied input. The output stays relatively well-behaved throughout most of the test, although there is still some instability in the middle of the sweep.

As noted before, in a floating-point representation, the roundoff error is equivalent to the coefficient sensitivity of the filter form. From this we can conclude that although the normalized lattice form performs much better than the direct form, it is still more sensitive than we might like since the first section coefficients are derived from the square of the radius of the pole location. We might ask if there is a filter form where none of the coordinates of the roots occurs in higher powers. Indeed, the so-called “coupled” form, which is a special case of the state-variable form exhibits exactly this property. Since the fully-general state-variable form has 9 coefficients and we have 5 parameters that determine the root positions and gain of the desired filter, that leaves us 4 degrees of freedom to optimize the performance of the filter. The literature on this is relatively rich, so I will just point the interested reader to the references [2, 4]. The state variables can be easily normalized [4]. Figure 6 shows the response of a normalized, state-variable filter to our test signal. Visually, it is indistinguishable from the output of the normalized lattice form. A close inspection of the numbers shows that the state variable form differs from the normalized lattice form in about the way that we would expect, given the difference in coefficient sensitivity due to the squares of the root coordinates involved in the normalized lattice form. For uniform unit random numbers as input (that is, random numbers between 0 and 1), there is a difference between the output of the two forms that hits a maximum value of about 0.2%. *This is even with using 64-bit floating point!* There is no hope for realizing this performance using either 32-bit floating point or 24-bit integer. We either have to eliminate these options from our system (that is, not allow users to select filters with those parameters, or with parameters that change this rapidly), or take great care in our filter form. The normalized state variable form appears to be the best choice for any realization. The 4 additional degrees of freedom give the designer considerable freedom to make the best tradeoffs for the desired performance. Additionally, since the filter coefficients generally relate to the root coordinates in a linear fashion, 24 bits are perfectly adequate for the coefficients, giving a resolution down to about 0.02 Hz, even at a 192 kHz sampling rate. This resolution is not possible using the other implementations discussed here.

Conclusions:

The first observation is that digital filtering when we allow the user to select high-Q, very low-frequency filters is difficult at the best of times. Even 64-bit floating point can produce significant error energy if the best filter forms are not used. Even for floating point, it is important to use forms that have normalized state variables so that imbalances in the state values do not lead to further degradation of the precision of the result. Clearly, the performance of 32-bit floating point and 24-bit integer will be considerably inferior to that of 64-bit floating point, so we might conclude that *it is not possible to achieve high-quality results for these extreme filter settings*. Furthermore it is shown that sweeping the settings of a filter with time excites some aberrant behavior when the state variables are not normalized, even with 64-bit floating-point arithmetic. 48-bit integer is proposed as a compromise between economic realizability and ultimate precision. The increased headroom and guard bits allowed by the format provide enough precision to allow some extreme filter settings and still preserve a 24-bit result after several stages of processing.

References:

[1] Vaidyanathan, P. P. “Robust Digital Filter Structures,” in *Handbook for Digital Signal Processing*, ed. By Mitra, S.K., and Kaiser, J.F., John Wiley and Sons, New York, 1993

[2] Roberts, R. A., and Mullis, C. T. “Digital Signal Processing,” Addison-Wesley, Reading, Massachusetts, 1987

[3] Agarwal, R.C. and Burrus, C. S. “New Recursive Digital Filter Structures Having Very Low Sensitivity and Roundoff Noise,” IEEE Trans. On Circuits and Systems, CAS-22, #12, pp921-927, December 1975

[4] Jackson, L. B. “Limit Cycles in State-Space Structures for Digital Filters,” IEEE Trans. On Circuits and Systems, CAS-26, pp67-68, Jan. 1979

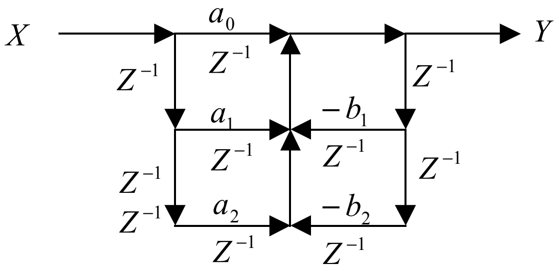

Figure 1: Signal flow diagram of the canonical "transposed" form of the second-order section.

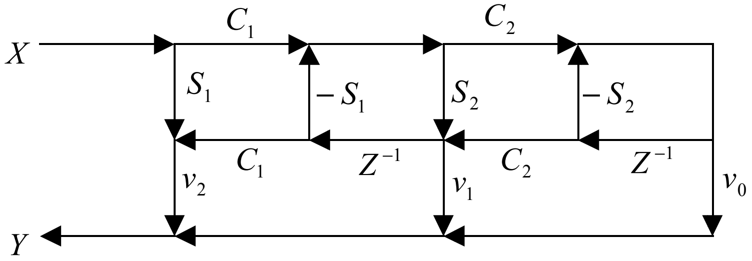

Figure 2– Normalized ladder form of a second-order section, showing links necessary for realization of arbitrary zeros of transmission. Without normalization, these coefficients (*v0*, *v1*, v2) are not bounded.

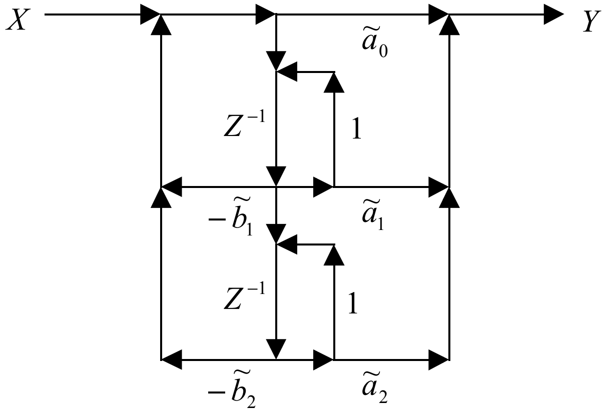

Figure 3: Agarwal-Burrus form is often suggested because of the superior performance at low frequencies. We do not study it here because the state variables are unbounded.

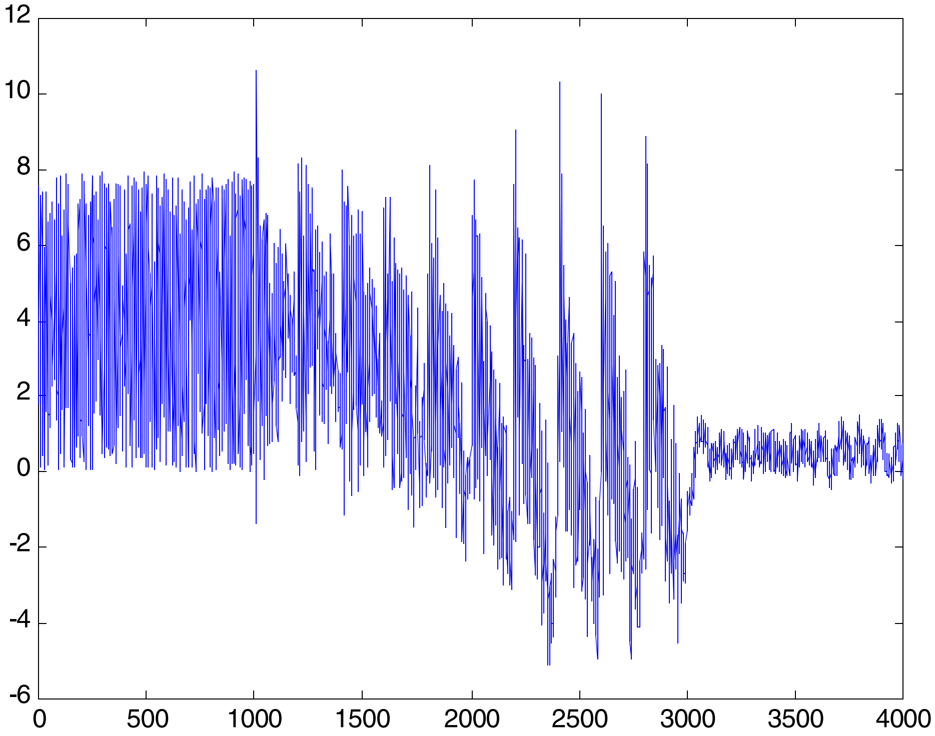

Figure 4: Demonstration of the perils of rapidly changing filter using the transposed (unnormalized) direct form. The filter was set to about 40 Hz for the first 1000 points, then ramped up to about 400 Hz over the next 2000 points. Notice the curious oscillatory behavior. Note that the response during the first 1000 points is swamped by filter noise.

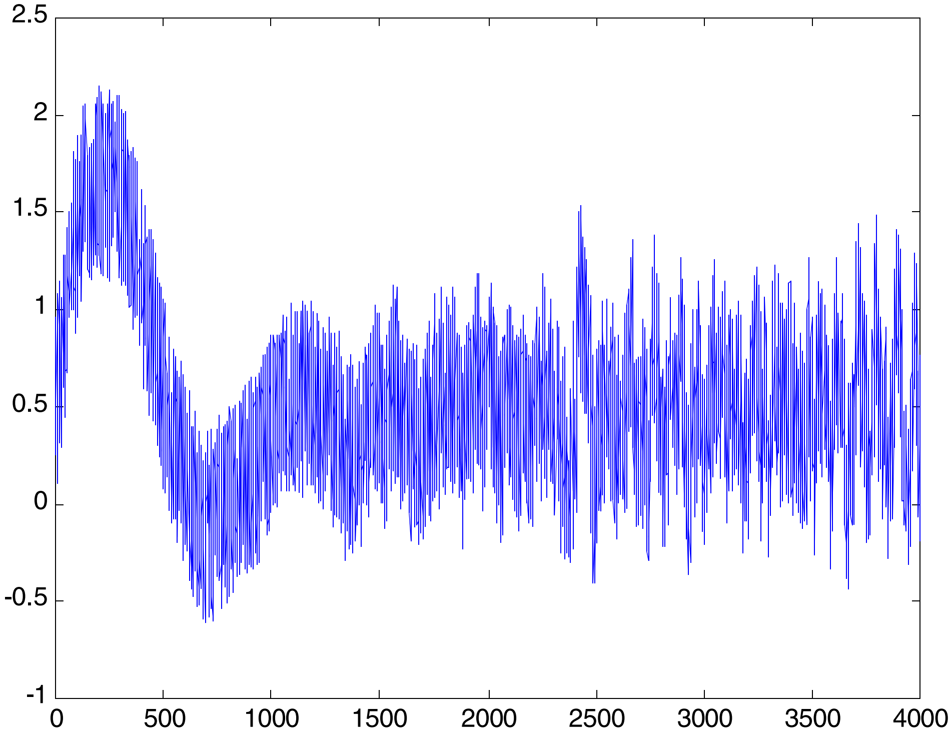

Figure 5: The same filter sweep, but using the normalized lattice realization. The initial sinusoidal behavior is simply the impulse response of the filter due to the onset of the input (random numbers). It is revealed here because it is no longer masked by filter noise. Notice that there is still some curious behavior in the middle of the sweep, but much less than the unnormalized system.

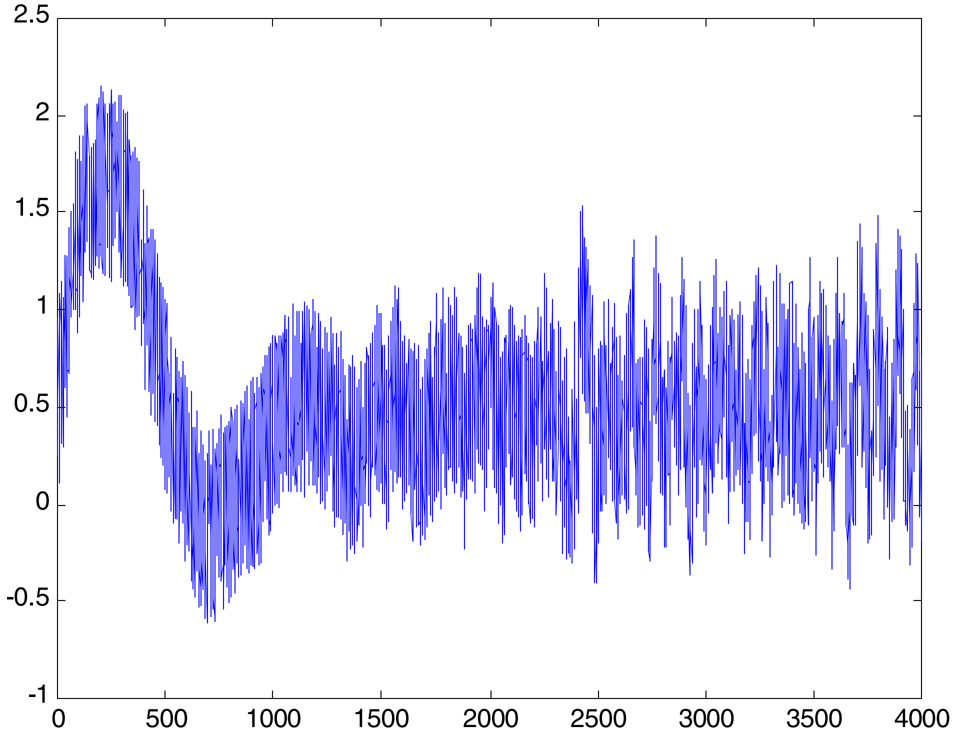

Figure 6: Response of a normalized state-variable filter to the same input. Visually, this is indistinguishable from the output of the normalized lattice realization. A detailed comparison of the numbers show that the normalized lattice output deviates from this output by as much as 0.2% (for an input of random numbers between 0 and 1.0)