$\sum_{n=-\infty}^{+\infty}$ ***Digital Sound Labs***

Digital Audio Signal Processing

Practical Considerations in Fixed-Point FIR Filter Implementations

Randy Yates

March 13, 2003 1:20

Typeset using PICTEX

Table of Contents

Suggestions for improvements: 1. Add rightindent to theorem definition 2. Put "Example" in small caps 3. The derivation leading up to theorem 3 is unclear - even I had a hard time following it! Also, the use of alternate symbols in theorem 3 is unclear. 4. Expand the warning after theorem 3 to explain coefficient quantization error effects (frequency response). 5. Add references. 6. Change "greatly" to "significantly". 7. Consider combining this paper and the FP paper into one "book." 8. Replace comma with period in the equation just prior to equation (2). 9. Clarify that "maximum coefficient length" means the size of the register that holds a coefficient. 10.

1 Introduction

1.1 Motivation

The most basic type of filter in digital signal processing is the **F**inite **I**mpulse **R**esponse (FIR) filter. By definition, a filter is classified as FIR if it has a z-transform of the form

$$H(z) = \frac{b_0 z^{N-1} + b_1 z^{N-2} + \ldots + b_{N-2} z + b_{N-1}}{z^{M-1}}, \qquad b_i \in \Re, \quad N, M \in \mathcal{Z}, \quad N > 0, \quad z \in \mathcal{C},$$

where $\Re$ denotes the reals, $\mathcal{Z}$ denotes the integers, and $\mathcal{C}$ denotes the complex numbers. This is referred to as an N-tap FIR filter. In general, an FIR filter can be either causal or non-causal. However, FIR filters are always stable, and indeed that is the chief reason they are widely utilized.

The difference equation that results from $H(z)$ is

$$\begin{aligned} y[n] &= b_0 x[n+N-M] + b_1 x[n+N-M-1] + \ldots + b_{N-2} x[n-M-2] + b_{N-1} x[n-M+1] \\ &= \sum_{i=0}^{N-1} b_i x[n+N-M-i]. \end{aligned}$$

If $N = M$, this simplifies to

$$\begin{aligned} y[n] &= b_0 x[n-0] + b_1 x[n-1] + \ldots + b_{N-2} x[n-(N-2)] + b_{N-1} x[n-(N-1)] \\ &= \sum_{i=0}^{N-1} b_i x[n-i]. \end{aligned}$$

This is the familiar result of the discrete convolution of the filter with the input data.

The equations above are the idealized, mathematical representations of an FIR filter because the arithmetic operations of addition, subtraction, multiplication, and division are performed over the field of real numbers $(\Re, +, \times)$, i.e., in the real number system (or over the complex field if the data or coefficients contain imaginary values). In practice, both the coefficients and the data values are constrained to be *fixed-point rationals* (see "Fixed Point Arithmetic: An Introduction"), a subset of the rationals. While this set is closed, it is not "bit bounded", i.e., the number of bits required to represent a value in the fixed-point rationals can be arbitrarily large. In a practical system one is limited to a finite number of bits in the words used for the filter input, coefficients and filter output. Most current digital signal processors provide arithmetic logic units and memory architectures to support 16 bit, 24 bit, or 32 bit wordlengths, however, one may implement arbitrarily long lengths by customizing the multiplications and additions in software and utilizing more processor cycles and memory. Similar choices can be made in digital hardware implementations. The final choices are governed by many aspects of the design such as required speed, power consumption, SNR, cost, etc.

1.2 Conventions

We shall represent scaled quantities using the $U(a, b)$ and $A(a, b)$ notation described in "Fixed Point Arithmetic: An Introduction".

There are generally two methods of operating on fixed-point data used today - integer and fractional. The integer method interprets the data as integers (either natural binary or signed two's complement) and performs integer arithmetic. For example, the Texas Instruments TMS320C54x DSP is an integer machine. The fractional method assumes the data are fixed-point rationals bounded between -1 and +1. The Motorola 56002 DSP is an example of a machine which uses fractional arithmetic. Except for an extra left shift performed in fractional multiplies, these two methods can be considered equivalent. In this article we shall utilize the integer method because I find it simpler and I am more familiar with it.

2 Scaling FIR Coefficients

Consider an FIR filter with N coefficients $b_0, b_1, \ldots, b_{N-1}$, $b_i \in \Re$. From "Fixed Point Arithmetic: An Introduction", we see that in fixed-point arithmetic a binary word can be interpreted as an unsigned or signed fixed-point rational. Although there are a number of situations in which the filter coefficients could be the same sign (and thus could be represented using unsigned values), let us assume they are not and hence that we must utilize signed fixed-point rationals for our coefficients. Thus we must find a way of representing, or more accurately, of *estimating*, the filter coefficients using signed fixed-point rationals.

Since a signed fixed-point rational is a number in the form $B_i/2^b$, where B_i and b are integers, $-2^{M-1} \leq B_i \leq 2^{M-1} - 1$, and M is the wordlength used for the coefficients, we determine the estimate b'_i of coefficient b_i by choosing a value for b and then determining B_i as

$$B_i = \text{round}(b_i * 2^b).$$

Then

$$b'_i = B_i/2^b.$$

In general, b'_i is only an estimate of b_i because of the rounding operation. This approximation phenomenom is referred to as *coefficient quantization* because, in a real sense, we are quantizing the coefficients in amplitude just exactly like an A/D converter amplitude quantizes an analog input signal. We can determine the "quantization error" e_i between the estimate and the real value by taking their difference:

$$\begin{aligned} e_i &= b'_i - b_i \\ &= B_i/2^b - b_i \\ &= \frac{\text{round}(b_i * 2^b)}{2^b} - b_i \\ &= \text{round}(b_i, -b) - b_i, \end{aligned}$$

where "round(x, y)" denotes rounding at bit y of the binary value x. The value $y = 0$ rounds at the units bit, with negative values going to the right of the decimal and positive values going to the left of the units bit. For example, round(1.0010110, -5)= 1.00110.

The question we have not yet answered is: *How do we choose* b? In order to answer this, note that the maximum error $e_{i_{max}}$ a quantized coefficient can have will be one-half of the bit being rounded at, i.e.,

$$\begin{aligned} e_{i_{max}} &= 2^{-b}/2 \\ &= 2^{-b-1}. \end{aligned}$$

It is now easy to see that, lacking any other criteria, the ideal value for b is the maximum it can be since that will result in the least amount of coefficient quantization error. Well just what exactly *is* the maximum, anyway? After all, b is from the integers, and the integers go to infinity. So the maximum is infinity, right?

Well, no. Again, considering the coefficient wordlength to be M (bits), note that a signed, two's complement value has a maximum magnitude of $2^{M-1}-1$. Therefore we must be careful not to choose a value for b which will produce a B_i that has a magnitude bigger than $2^{M-1}-1$. When a value becomes too big to be represented by the representation we have chosen (in this case, M-bit signed two's complement), we say that an *overflow* has occurred. Thus we must be careful to choose a value for b that will not *overflow* the largest magnitude coefficient. We may compute this maximum value for b as

$$b = \lfloor \log_2 \left((2^{M-1}-1)/max(|b_i|)\right) \rfloor,$$

where $\lfloor x \rfloor$ denotes the greatest integer less than or equal to x.

In summary we see that, lacking any other criteria, the ideal value for b is the **maximum** value which can be used without overflowing the coefficients since that provides the minimum coefficient quanization error. We emphasize this important result by stating the following

1. ***First Coefficient Scaling Theorem.*** Let b_i be a set of coefficients with scale factor b. Maximum precision is preserved when b is chosen to be the maximum integer possible without overflowing the coefficient representation, i.e.,

$$b = \lfloor \log_2 \left((2^{M-1}-1)/\max(|b_i|)\right) \rfloor, \tag{1}$$

where M is the coefficient wordsize in bits.

Example 1 ———

Consider a 4-tap FIR filter with the following coefficients:

$$\begin{aligned} b_0 &= +1.2830074 \\ b_1 &= -2.3994138 \\ b_2 &= +0.1234689 \\ b_3 &= +0.0029153 \end{aligned}$$

Assuming 16-bit wordlengths, find a) the scaling factor b, and b) the coefficient estimates b'_i using rule 1.

<u>Solution:</u>

$$\begin{aligned} b &= \lfloor \log_2 \left((2^{M-1}-1)/max(|b_i|)\right) \rfloor \\ &= \lfloor \log_2 \left((2^{16-1}-1)/2.3994138)\right) \rfloor \\ &= \lfloor 13.73727399 \rfloor \\ &= 13. \end{aligned}$$

Since $2^{13} = 8192$,

$$\begin{aligned} b'_0 &= \text{round}(+1.2830074 \times 8192)/8192 \\ &= +1.2829589843750 \\ b'_1 &= \text{round}(-2.3994138 \times 8192)/8192 \\ &= -2.3994140625000 \\ b'_2 &= \text{round}(+0.1234689 \times 8192)/8192 \\ &= +0.1234130859375 \\ b'_3 &= \text{round}(+0.0029153 \times 8192)/8192 \\ &= +0.0029296875000 \end{aligned}$$

So that's it, right? We now know everything there is to know about coefficient scaling, right?

Well, no. Remember when I said, "...lacking any other criteria..."? Well, guess what - there are other criteria.

Adding two J-bit values requires $J+1$ bits in order to maintain precision and avoid overflow when there is no *a priori* knowledge about the values being added. For example, if the 16-bit signed two's complement values 21,583 and 12,042 are summed, the result is 33,625. Since the maximum value for a 16-bit signed two's complement number is 32,767, we must add an extra bit to avoid overflowing. Also, since the result is odd, the least-significant bit (bit 0) is set, so we cannot simply take the upper 16 bits of the 17 bit result without losing precision. As a counterexample, consider processing a stream of data in which any two adjacent samples are known to be of opposite signs. In this case, we would be able to guarantee that the sum of two adacent J-bit samples would never overflow J bits.

We may easily extend this rule to sums of multiple values and state the result as the

2. ***Fixed-Point Summation Theorem.*** The sum of N J-bit values requires $J + \lceil \log_2 N \rceil$ bits to maintain precision and avoid overflow if no information is known about the values.

Let us consider an N-tap FIR filter which has L-bit data values and M-bit coefficients. Then using the relations above, the final N-term sum required at each time interval n,

$$y[n] = b'_0 x[n] + b'_1 x[n-1] + b'_2 x[n-2] + \ldots + b'_{N-1} x[n-N+1],$$

requires $L + M + \log_2 N$ bits in order to maintain precision and avoid overflow if no information is known about the data or the coefficients. For example, a 64-tap FIR filter ($N = 64$) with 16-bit coefficients and data values ($L = M = 16$) requires $L + M + \log_2(N) = 32 + \log_2(64) = 32 + 6 = 38$ bits in order to maintain precision and avoid overflow.

Most processors and hardware components provide the ability to multiply two M-bit values together to form a $2M$-bit result. For example, the Integrated Device Technolgy 7210 multiplier-accumulator performs 16x16 multiplies to a 32-bit result. Most general purpose and some DSP processors provide an accumulator that is the same width as the multiplier output. For example, the Texas Instruments TMS320C50 DSP provides a 16x16 multiplier and a 32-bit accumulator. Some DSP processors provide a $2M + G$-bit accumulator, where G denotes "guard bits" (to be explained shortly). For example, the Texas Instruments TMS320C54x DSP provides a 16x16 multiplier with a 32-bit output and a 40-bit accumulator ($M = 16$, $G = 8$).

Therefore another criteria in the design of FIR filters is that the final convolution sum fit within the accumulator. To put it algebraically, we require that

$$2M + \log_2 N \leq 2M + G,$$

where we have assumed that the coefficient wordlength and the data wordlength is the same (M bits), and where we have assumed we have no information about the data or the coefficients. The key point here is that **the number of bits required for the filter output increase with the length of the filter.**

For those situations in which $G = 0$ (e.g., the TMS320C50), we see that we immediately have a problem for even a two-tap FIR filter since that filter requires $2M + \log_2 2 = 2M + 1$ bits and the accumulator is only $2M$ bits. This is precisely why the extra G bits which are available on some processors are called "guard bits" - they guard against overflow when performing summations. However, even though the accumulator may have guard bits, it is still possible to overflow the accumulator if $\log_2 N > G$, i.e., if we attempt to use a filter that is longer than 2^G taps.

The easiest solution is to simply decree that we shall maintain an optimistic outlook. In other words, we will acknowledge that our filter won't work for "the most general case" and hope and pray that those cases (i.e., those combinations of N data values) which would result in overflow for our filter will never occur. However, this is rather like sticking one's head in the sand, because if and when overflows occur, they can be catastrophic. In signed two's complement systems, overflows cause abrupt variations in output levels which, in the case of digital audio, are very audible to say the least and extremely rude to be more accurate.

Another solution is to redesign the filter to use fewer taps. However, if there are no guard bits, then the filter would be reduced to a gain control (i.e., 1 tap), and even with guard bits, the number of filter taps is usually at a premium to begin with anyway (i.e., we can almost always use more taps to implement a better filter).

Yet another solution is to scale down the data values by K bits before applying the filter, thus allowing 2^K more taps in the filter before overflowing. This is, in general, a horrible idea because it greatly degrades the signal-to-noise ratio of the signal path by 6 dB per bit.

A better solution is to modify the way in which we use equation (1) to scale the coefficients. Since the M we use in equation (1) is effectively the number of bits used for the coefficients, we can simply use an alternate value that is smaller than M bits available in our hardware. After all, just because we have an M-bit wordlength available for the coefficients doesn't mean we have to use all M bits. Therefore let us use M' bits for the coefficients, where $M' \leq M$.

What size shall we make M'? Let us (lettuce?) calculate it based on the width of the accumulator:

$$M + M' + \log_2 N = 2M + G \quad \Longrightarrow \quad M' = \min(M, M + G - \log_2 N),$$

We summarize this section with

3. ***Second Coefficient Scaling Theorem.*** If no information is known about the data or the coefficients, then the coefficient wordlength M' must be

$$M' = \min(M, A - L - \log_2 N), \tag{2}$$

in order to avoid overflow and preserve precision in an N-tap FIR filter output, where M is the maximum coefficient wordlength, A is the accumulator wordlength, and L is the data wordlength.

WARNING: There is a cost associated with this solution: increased coefficient quantization error. This fact should not be overlooked when weighing the options.

Example 2

We continue with the 4-tap FIR filter we used in example 1, Assume the maximum coefficient wordlength is 16 bits, the data wordlength is 16 bits and the accumulator wordlength is 32 bits.

a. Find the value for M', i.e., the effective coefficient wordlength that will avoid overflow and guarantee precision is preserved in the filter output using rule 2.

b. Substitute this result into coefficient scaling rule 1 to obtain b', the new coefficient scaling.

<u>Solution:</u>

a. Simply plug the numbers into equation (2):

$$\begin{aligned} M' &= \min(M, A - L - \log_2 N) \\ &= \min(16, 32 - 16 - \log_2 4) \\ &= \min(16, 32 - 16 - 2) \\ &= \min(16, 14) \\ &= 14. \end{aligned}$$

b. Substitute M=14 into equation (1):

$$\begin{aligned} b' &= \lfloor \log_2 \left((2^{M-1} - 1)/max(|b_i|) \right) \rfloor \\ &= \lfloor \log_2 \left((2^{14-1} - 1)/2.3994138 \right) \rfloor \\ &= \lfloor 11.73731892 \rfloor \\ &= 11. \end{aligned}$$

We see that the reduction of wordlength by 2 bits in part a also results in a reduction in the coefficient scale factor by 2 bits and thus increases the coefficient quantization error. This is the price paid for ensuring the result will not overflow.

So now we're really done, right? We certainly must now know everything there is to know about coefficient scaling, right?

Well, no. Remember when I said, "...if no information is known about the data or coefficients..."? It is often the case that the coefficient values are known at design time (and won't change). Therefore we *do* have information about the coefficients. How can we use this information to improve our filter architecture?

Since we are constantly concerned about overflow in fixed-point digital signal processing, let us begin by considering what combination (or combinations) of N input data values will provide maximum output from a given N-tap FIR filter. In order to answer this, recall the triangle inequality:

$$|a + b| \leq |a| + |b|.$$

Using the obvious relation $a + b \leq |a + b|$, we then have

$$a + b \leq |a + b| \leq |a| + |b| \qquad \Rightarrow \qquad a + b \leq |a| + |b|.$$

We may generalize this 2-term sum to an N-term sum. This means that the signs of $x[k]$ that will make the terms $b_i x[n-i]$ all positive in the convolution sum

$$y[n] = \sum_{i=0}^{N-1} b_i x[n-i],$$

will result in larger output. This occurs when $sgn(x[n-i]) = sgn(b_i)$. We may therefore rewrite the set of $x[n-i]$s that maximize the output as

$$x[n-i] = sgn(b_i)|x[n-i]|.$$

Our convolution sum now looks like this:

$$\begin{aligned} y[n] &= \sum_{i=0}^{N-1} b_i x[n-i] \\ &= \sum_{i=0}^{N-1} b_i (sgn(b_i)|x[n-i]|). \end{aligned}$$

But note that $sgn(r)r = |r|$ for any real value r. Therefore $b_i sgn(b_i) = |b_i|$, and we have

$$y[n] = \sum_{i=0}^{N-1} |b_i||x[n-i]|.$$

What further property could we assign to $x[n-i]$ that would maximize this sum? It should be obvious that if we maximize all the magnitudes of $x[n]$, then we maximize the sum. Therefore let $|x[n-i]| = x_{MAX}$, where x_{MAX} denotes the maximum magnitude possible for $x[n-i]$. Then

$$\begin{aligned} y_{MAX}[n] &= \sum_{i=0}^{N-1} |b_i| x_{MAX} \\ &= x_{MAX} \sum_{i=0}^{N-1} |b_i|. \end{aligned}$$

Also, denote the sum on the right by α, i.e.,

$$\alpha = \sum_{i=0}^{N-1} |b_i|.$$

Let's pause and consider our results so far. Basically, we see that the maximum output value of a filter is a function of the "coefficient area" (α) in the filter. This seems intuitively obvious. Also obvious is the fact that reducing the overall gain of the filter reduces the maximum filter output as well.

So far we have been operating in the infinite-precision (i.e., real) domain. Now express this result in terms of the unscaled integers X and B_i, where x is scaled $A(a_x, b_x)$ and b_i is scaled $A(a_b, b_b)$ so that $x = X/2^{b_x}$ and $b_i = B_i/2^{b_b}$:

$$\begin{aligned} y_{MAX}[n] &= (x_{MAX})(\sum_{i=0}^{N-1} |b_i|) \\ &= (x_{MAX})(\alpha) \\ &= (X_{MAX}/2^{b_x})(\alpha). \end{aligned}$$

Let us use the previous notation of A for accumulator wordlength, L for the data wordlength, and M for the coefficient wordlength. Rules of fixed-point arithmetic dictate that the scaling of the result $y_{MAX}[n]$ will be $A(A - b_x - b_b - 1, b_x + b_b)$. Thus

$$\begin{aligned} Y_{MAX}[n]/2^{b_x+b_b} &= y_{MAX}[n] \\ &= (X_{MAX}/2^{b_x})(\alpha) \\ Y_{MAX}[n] &= 2^{b_b}\alpha X_{MAX}. \end{aligned}$$

We know that the maximum of any T-bit signed two's complement integer is $2^{T-1} - 1$, which, when $T >> 0$, can be approximated as simply 2^{T-1}. Therefore we can express the last result as

$$2^{A-1} \geq 2^{b_b}\alpha 2^{L-1}.$$

Note the use of the inequality since α will seldom be an exact power of two. Take the log base 2 of both sides and solve for b_b:

$$\begin{aligned} A - 1 &\geq b_b + \log_2 \alpha + L - 1 \\ b_b &\leq A - L - \log_2 \alpha \\ \Rightarrow b_b &= A - L - \lceil \log_2 \alpha \rceil. \end{aligned}$$

This important result says that in order to avoid overflow in the output the maximum value for the coefficient scale factor b_b is established by the accumulator wordlength A, the data wordlength L, and the coefficient area α.

Let us take stock then of the current situation. There are three criteria that the coefficient scale factor b_b seeks to satisfy:

1. We seek to **maximize** b_b in order to reduce coefficient quantization error.
2. Given a maximum coefficient filter length M, we seek to **constrain** b_b in order that the coefficient with the largest magnitude is representable.
3. Given the accumlator wordlength A, the data wordlength L, and the information about the coefficients we call the coefficient area α, we seek to **constrain** b_b so that overflows in the convolution sum are avoided.

Hence we see that the value for b_b that meets all three criteria is given by the following function:

$$b_b = \min(\lfloor \log_2 \left((2^{M-1} - 1)/max(|b_i|) \right) \rfloor, A - L - \lceil \log_2 \alpha \rceil) \tag{3}$$

Example 3

Consider the 16-tap FIR filter $b_0 = 1$ and $b_1, b_2, \ldots, b_{15} = 0$. Assuming an accumulator wordlength of 32 bits, a data wordlength of 16 bits, and a coefficient wordlength of 16 bits, use equation (3) to establish the optimum value for the coefficient scale factor b_b.

Solution:

Calculate α:

$$\begin{aligned}\alpha &= \sum_{i=0}^{15} |b_i| \\ &= 1.\end{aligned}$$

Then

$$\begin{aligned}b_b &= \min(\lfloor \log_2 \left((2^{M-1} - 1)/max(|b_i|)\right) \rfloor, A - L - \lceil \log_2 \alpha \rceil) \\ &= \min(\lfloor \log_2 \left((2^{16-1} - 1)/1\right) \rfloor, 32 - 16 - \lceil \log_2 1 \rceil) \\ &= \min(14, 16) \\ &= 14.\end{aligned}$$

In this case we see that the limiting factor is that which allows the coefficients to be representable.

Example 4

Consider the 16-tap FIR filter $b_0, b_1, b_2, \ldots, b_{15} = 0.0625$. Assuming an accumulator wordlength of 32 bits, a data wordlength of 16 bits, and a coefficient wordlength of 16 bits, use equation (3) to establish the optimum value for the coefficient scale factor b_b.

Solution:

Calculate α:

$$\begin{aligned}\alpha &= \sum_{i=0}^{15} |b_i| \\ &= 1.\end{aligned}$$

Then

$$\begin{aligned}b_b &= \min(\lfloor \log_2 \left((2^{M-1} - 1)/max(|b_i|)\right) \rfloor, A - L - \lceil \log_2 \alpha \rceil) \\ &= \min(\lfloor \log_2 \left((2^{16-1} - 1)/.0625)\right) \rfloor, 32 - 16 - \lceil \log_2 1 \rceil) \\ &= \min(18, 16) \\ &= 16.\end{aligned}$$

In this case we see that the limiting factor is that which avoids overflow in the accumulator.

Example 5

Consider the 16-tap FIR filter $b_0, b_1, b_2, \ldots, b_{15} = 0.0625$. Assuming an accumulator wordlength of 40 bits, a data wordlength of 16 bits, and a coefficient wordlength of 16 bits, use equation (3) to establish the optimum value for the coefficient scale factor b_b.

Solution:

Calculate α:

$$\begin{aligned}\alpha &= \sum_{i=0}^{15} |b_i| \\ &= 1.\end{aligned}$$

Then

$$\begin{aligned}b_b &= \min(\lfloor \log_2 \left((2^{M-1} - 1)/max(|b_i|)\right) \rfloor, A - L - \lceil \log_2 \alpha \rceil) \\ &= \min(\lfloor \log_2 \left((2^{16-1} - 1)/.0625)\right) \rfloor, 40 - 16 - \lceil \log_2 1 \rceil) \\ &= \min(18, 24) \\ &= 18.\end{aligned}$$

In this case we see that the limiting factor is that which allows the coefficients to be representable, but only because this accumulator has 8 guard bits, otherwise overflow in the accumulator would limit b_b as in example 4. Also note that the extra accumulator guard bits allow the coefficient quantization error to be less than in example 4.

3 Choosing the FIR Filter Output Word

As stated earlier, a DSP or hardware multiplier has an output wordsize that is usually two or more times the size of the input wordsize, e.g., the TI TMS320C54x has a 40-bit accumulator with a 32-bit multiplier and typically operates on data and coefficient wordsizes of 16 bits. It is therfore normally the case that a subset of the accumulator bits must be chosen for the final filter output. How do we choose these bits?

Given a set of N signed, unscaled two's complement coefficients B_i, $i = 0, 1, \ldots, N-1$, and an input wordsize of L bits, the number of bits required to maintain precision while simultaneously avoiding overflow in the convolution sum is

$$\Gamma = L + \log_2(\alpha),$$

where

$$\alpha = \sum_{i=0}^{N-1} |B_i|.$$

Therefore, in order to avoid overflow when truncating this width Γ to the output wordlength K, we extract bits $\Gamma - K$ to $\Gamma - 1$, numbering the LSB of the accumulator as bit 0.

For example, if $L = 16$ and $\alpha = 32$, then

$$\begin{aligned}\Gamma &= L + \log_2(\alpha)\\ &= 16 + \log_2(32)\\ &= 21,\end{aligned}$$

and if the output wordlength $K = 16$, then you must take bits 5 to 20,

If the accumulator size of A is less than Γ, then the filter may overflow the accumulator during the convolution sum. In this case, the best choice is to choose the top K bits of the accumulator.

4 Quantization Noise in FIR Filters

[under construction]

4.1 Truncation

4.2 Rounding

4.3 Dithering

4.4 Noise-shaping

5 Conclusions

My hope is that this article will allow the FIR filter designer to clearly see the effects that the choices of wordlength, scaling, and processing architecture have on signal integrity, and that the material is clear and accurate. Errors, suggestions, etc., should be mailed to `yates@ieee.org`.

6 References