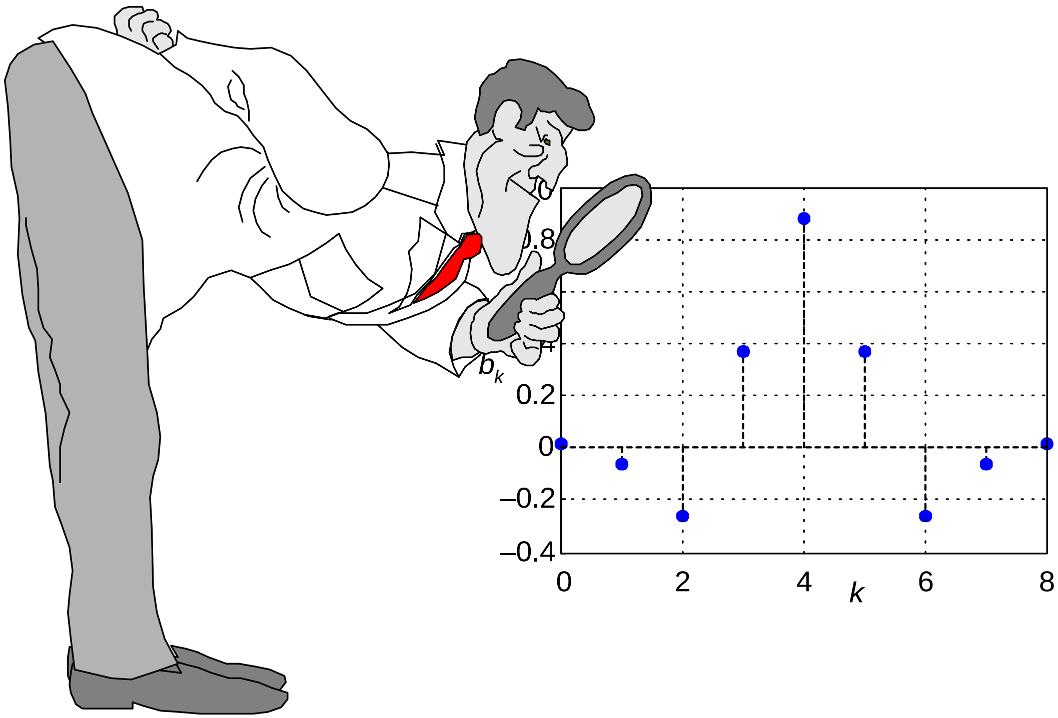
Improving FIR Filter Coefficient Precision

Speaker: Richard Lyons
Besser Associates
E-mail: R.Lyons@ieee.com



Improving FIR Filter Coefficient Precision

- **There is a method for increasing the precision of fixed-point coefficients used in linear-phase finite impulse response (FIR) filters,**
  - **to achieve improved filter performance,**
  - **without increasing either the number of coefficients or coefficient bitwidths.**

- **Thinking about this, such a process does not seem possible.**

- **But to see how, let's first review the behavior of a FIR filter.**

- **Consider an FIR filter's coefficients (impulse response) shown in Figure 1(a).**
- **Such a filter can be implemented as shown in Figure 1(b).**

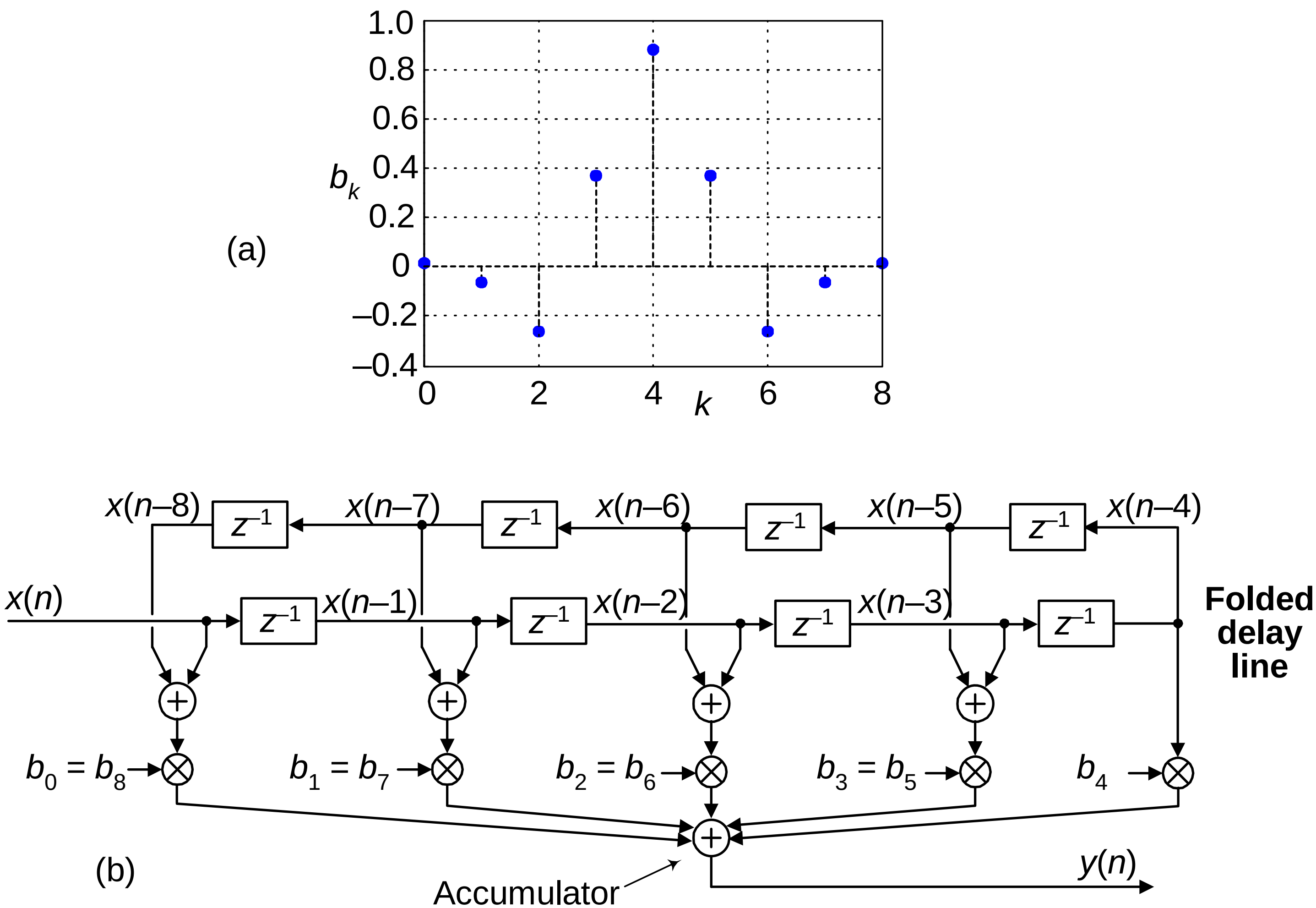
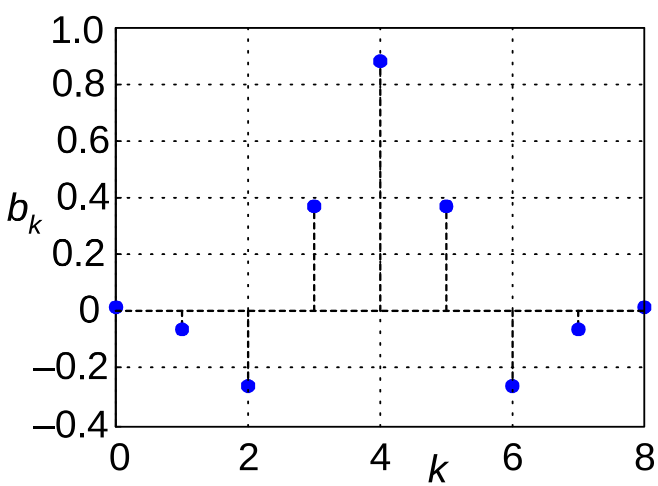


Figure 1

- **Quantizing those coefficients in, an 8-bit two's complement format,**
  - **yields the decimal integer and binary values in Figure 2(b).**

- **The beginning, and ending, coefficients are small in amplitude.**

- **Many high-order bits of the low-amplitude coefficients, the red-font underscored bits, are the same as the sign bit.**
  - **That's because of the fixed bitwidth quantization.**



(a)

Floating-point coefficients

sign bit

b_0	0.01751	2	0<u>00000</u>10
b_1	–0.05899	–8	1<u>1111</u>000
b_2	–0.26156	–33	1<u>1</u>011111
b_3	0.37687	48	0<u>0</u>110000
b_4	0.87968	113	01110001
b_5	0.37687	48	0<u>0</u>110000
b_6	–0.26156	–33	1<u>1</u>011111
b_7	–0.05899	–8	1<u>1111</u>000
b_8	0.01751	2	0<u>00000</u>10

(b)

Figure 2

- **Those underscored bits are "wasted" bits.**
  - **They have no effect (no weight) on the calculation of filter output *y*(*n*).**

sign bit ↓

b_0	0.01751	2	00000010
b_1	–0.05899	–8	11111000
b_2	–0.26156	–33	11011111
b_3	0.37687	48	00110000
b_4	0.87968	113	01110001
b_5	0.37687	48	00110000
b_6	–0.26156	–33	11011111
b_7	–0.05899	–8	11111000
b_8	0.01751	2	00000010

- **So the idea here is to replace those "wasted" bits with more significant bits,**
  - **to give us improved numerical precision for the low-amplitude beginning and ending coefficients.**

- **OK, let's look at an example,**
  - **of what's called a "serial" implementation of this whole idea.**

"Serial" Method

- **Assume we quantize the maximum-amplitude coefficient, b_4, to eight bits.**

- **Next, we quantize the lower-amplitude coefficients to larger bitwidths than the max-amplitude coefficient b_4.**

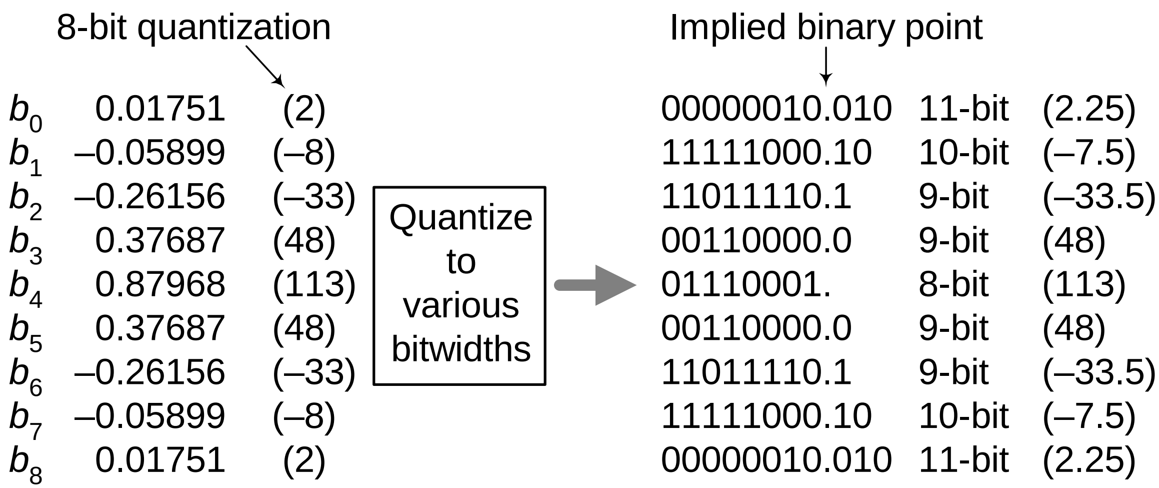


Figure 3

- **We'll discuss how to choose the coefficients' variable bitwidths in a moment.**

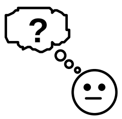
- **Next we delete the appropriate "wasted" (red-underscored) bits,**
  - **to arrive at our final 8-bit coefficients.**

	Delete wasted (underscored) bits →	Final coefficients		
00000010010		00010010	(18)	Flag = 1
11111100010		11100010	(–30)	Flag = 1
110111101		10111101	(–67)	Flag = 0
001100000		01100000	(96)	Flag = 1
01110001		01110001	(113)	Flag = 0
001100000		01100000	(96)	Flag = 1
110111101		10111101	(–67)	Flag = 0
11111100010		11100010	(–30)	Flag = 1
00000010010		00010010	(18)	Flag = 1

Figure 4

- **Appended to each coefficient is a flag bit,**
  - **indicating whether that coefficient used one more quantization bit than the previous (next larger) coefficient.**

- **The question now is, "How do we use those "oddball" coefficients in a filter?"**



- **Figure 5 shows us the answer.**
- **This implementation is called "serial" because there is only one multiplier.**

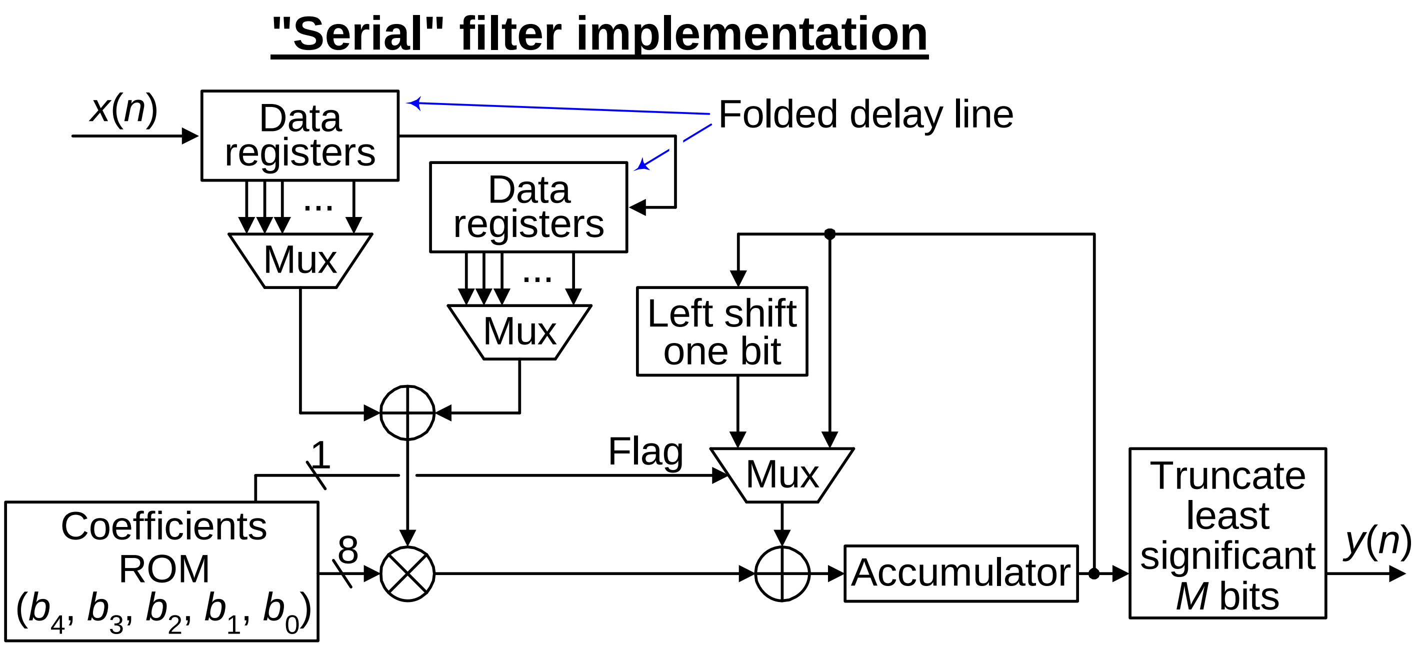


Figure 5

- **For an *N*-tap FIR filter,**
  - **for odd *N*, (*N*+1)/2 coefficients are stored in the coefficient ROM (read-only memory).**
  - **for even *N*, *N*/2 coefficients are stored in the coefficient ROM.**

- **When a new $x(n)$ input sample arrives, we:**
  - **Set the accumulator to zero.**
  - **Multiply the sum of the appropriate data registers by the b_4 coefficient.**
  - **Add that product to the accumulator.**
  - **Next we multiply the sum of the appropriate data registers by the b_3 coefficient.**
    - **If the flag bit of the b_3 coefficient is one, we left-shift the current accumulator value and add the current multiplier's output to the shifted accumulator value.**
    - **If the current coefficient's flag bit is zero the accumulator word is not shifted prior to an accumulation.**
  - **Continue these multiplications, possible left shifts, and accumulations for the remaining b_2, b_1, and b_0 coefficients.**

- **So, when a new *x*(*n*) input sample arrives, we perform a series of multiplications and accumulations (using multiple clock cycles),**
  - **always starting with the largest coefficient (b_4),**
  - **to produce a single *y*(*n*) filter output sample.**
- **To maintain our original FIR filter's gain,**
  - **after the final accumulation we truncate the final accumulator value by discarding its least significant *M* bits,**
  - **where *M* is the total number of flag bits in the ROM memory.**
- **Let's look at this "serial" method in action.**

"Serial" Example

- **Implement a 29-tap lowpass FIR filter,**
  - **whose cutoff frequency is 0.167f_s and whose stopband begins at 0.292f_s.**

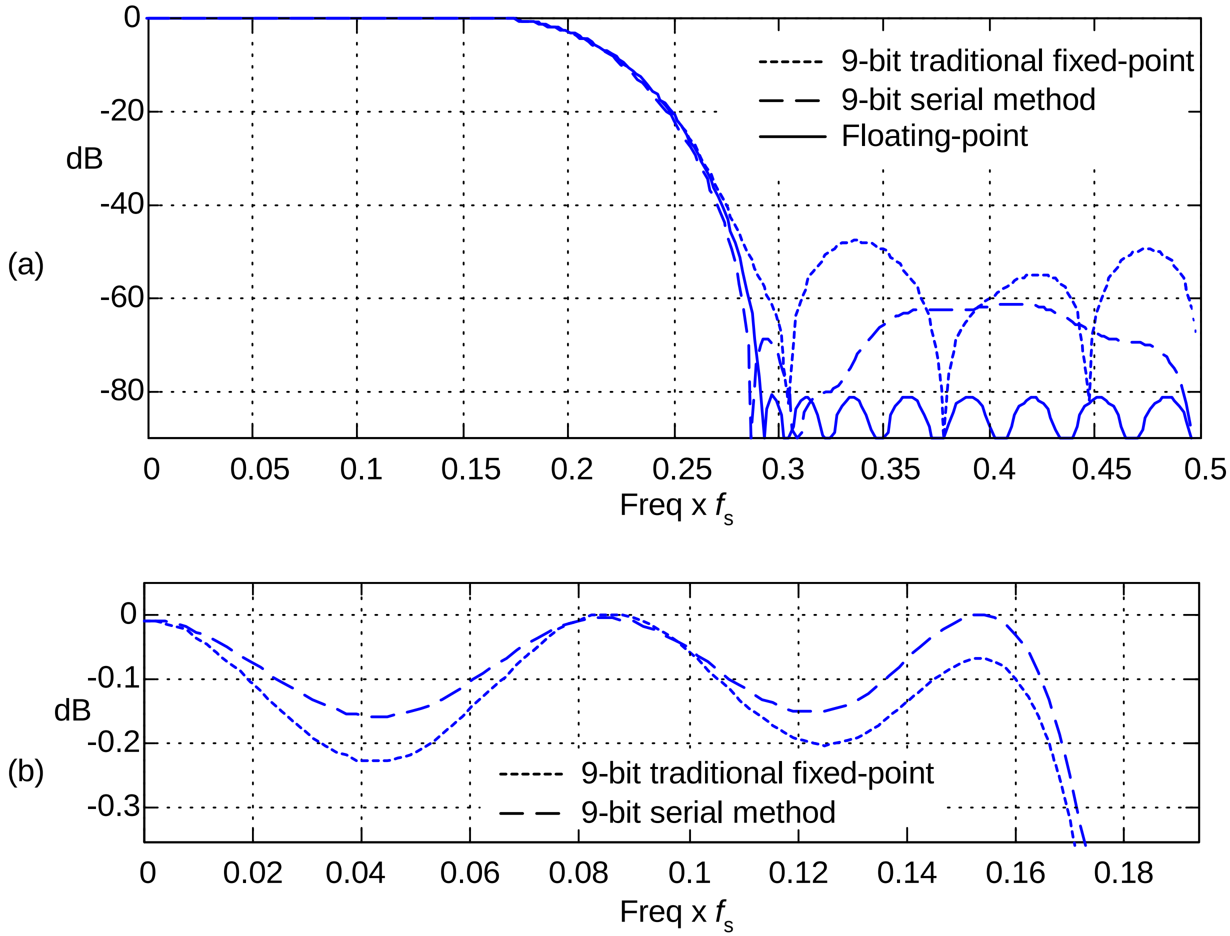
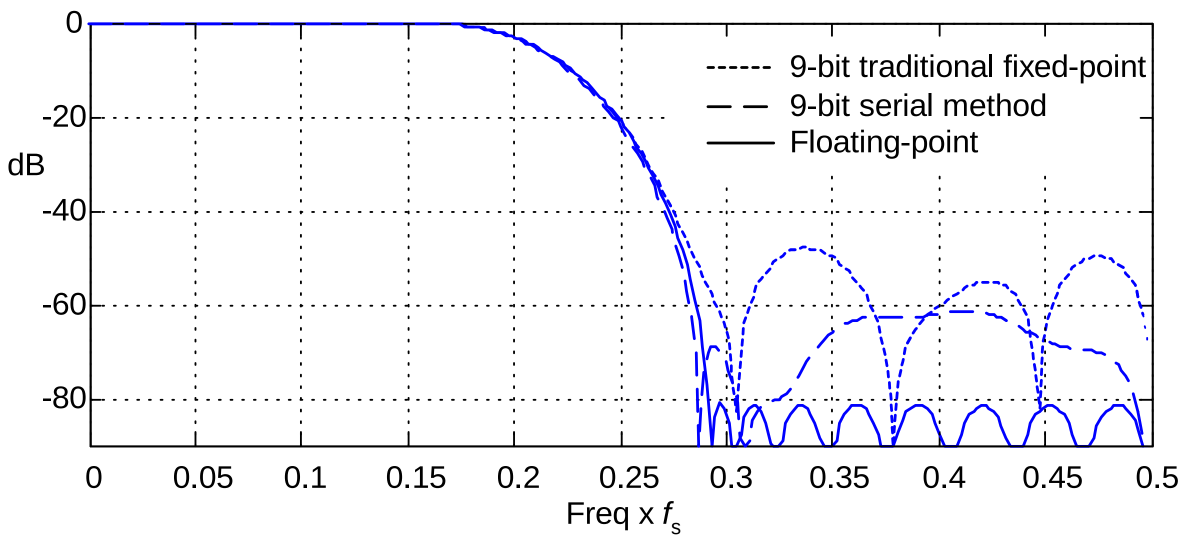


Figure 6

- **Relative to a traditional fixed-point implementation (dotted curve), the "serial method" (dashed curve) provides:**
  - **Improved stopband attenuation,**
  - **Reduced transition region width,**
  - **Improved passband ripple performance.**



- **All of these improvements occur:**
  - **<u>without increasing the bitwidths of our filter's coefficients</u>**
  - **<u>without increasing the number of coefficients.</u>** ☺

- **Regarding this "serial method", American actor Robert De Niro would say:**
  - **I like it.**
  - **I like it.**
  - **What did I tell you?**
  - **WHAT DID I TELL YOU?**
  - **I like it!**

- **As it turns out, we can do even better than the "serial method".**

"Parallel" Method

- **In the serial method, adjacent filter coefficients were quantized to a precision differing by no more that one bit.**
  - **That's because we used "flag bits".**

- **In the parallel method, adjacent coefficients can be quantized to a precision differing by more than one bit.**

- **Figure 7 shows an example of our parallel method's coefficient quantization process.**

- **Again, assume we quantize the maximum-amplitude coefficient, b_4, to eight bits.**

- **Next, we quantize the lower-amplitude coefficients to larger bitwidths than the max-amplitude coefficient b_4.**

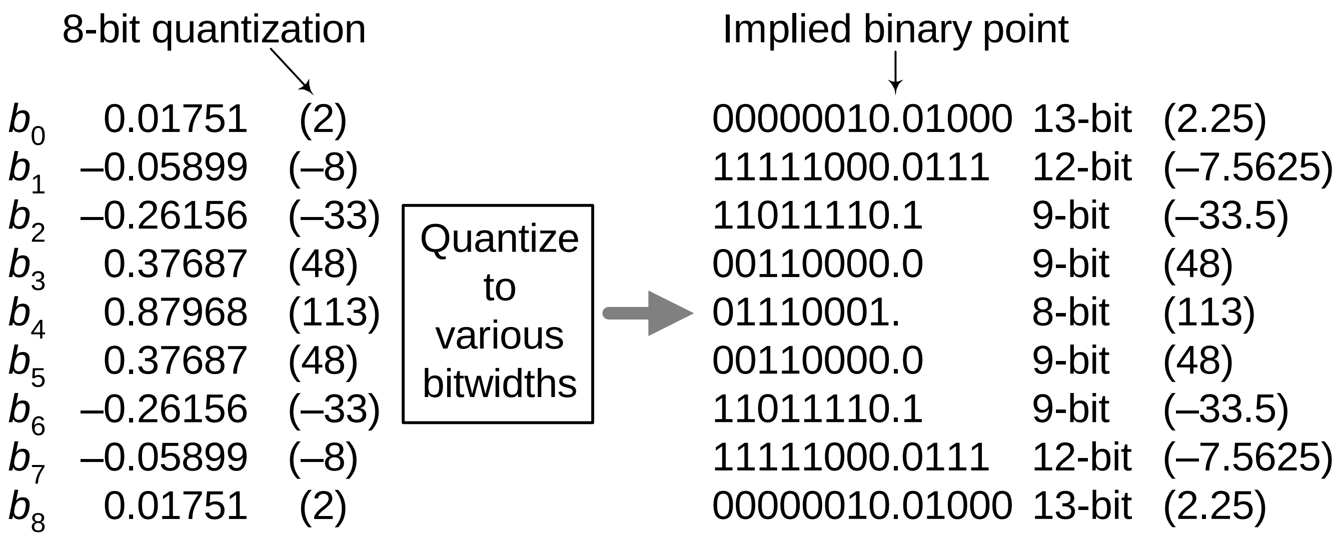


Figure 7

- **Notice that b_2 is quantized to 9 bits, and**
  - **b_1 is quantized to 12 bits.**

- **We'll discuss how to choose the coefficients' variable bitwidths in a moment.**

- **As before, we then delete the appropriate "wasted" (red-underscored) bits,**
  - **to arrive at our final 8-bit coefficients.**

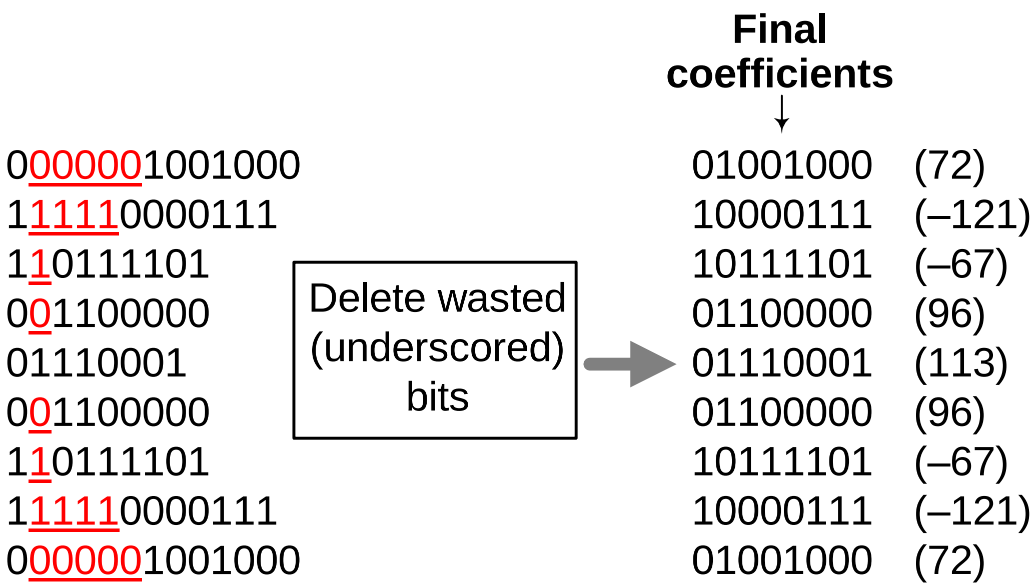


Figure 8

- **Figure 9 shows the implementation of the "parallel" method.**

- **This implementation is called "parallel" because there are multiple multipliers.**
- **To keep our drawings simple, assume we're building a 5-tap filter.**
  - **b_2 is the maximum-amplitude coefficient.**

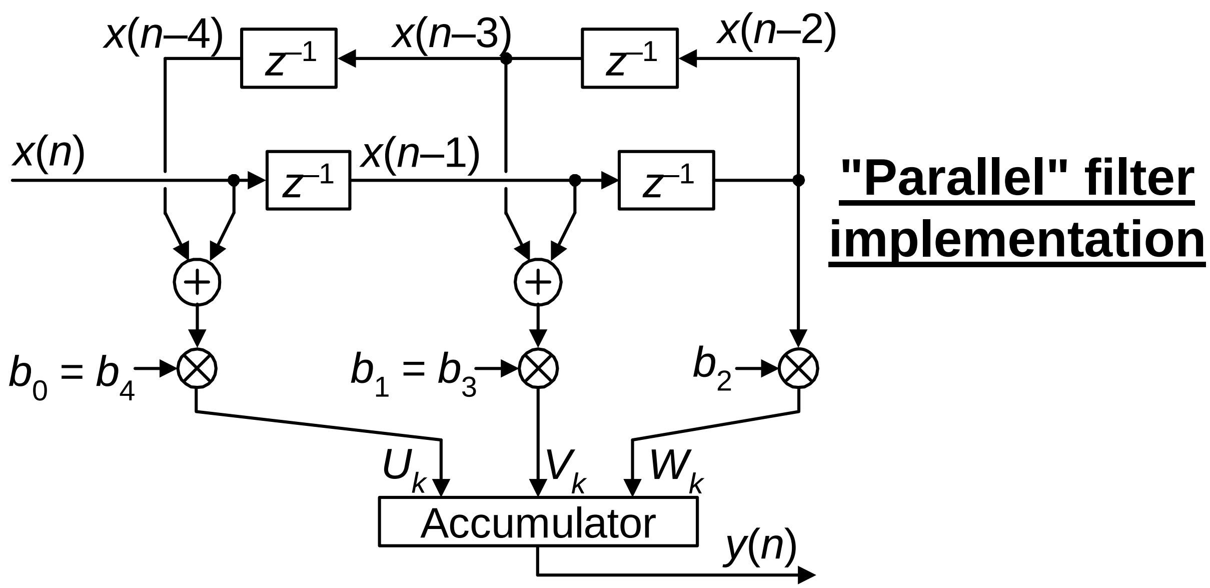


Figure 9

- **When a new $x(n)$ input sample arrives, we:**
  - **Set the accumulator to zero.**
  - **Multiply the sums of the appropriate data registers by the corresponding coefficients.**
    - **All multiplications occur in one clock cycle (i.e., in parallel).**

- **The multiple products are added to the accumulator as shown in Figure 10.**

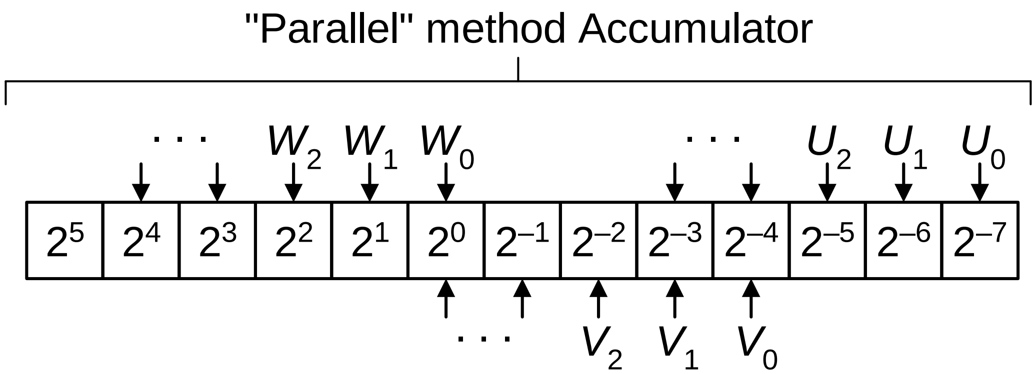


Figure 10

- **For example, if there were four wasted bits deleted from the high-precision b_1 coefficient,**
  - **then the V_k product is shifted to the right by four bits, relative to the W_k product bits, before being added to the accumulator word.**

- **If there were seven wasted bits deleted from the high-precision b_0 coefficient,**
  - **then the U_k product is shifted to the right by seven bits, relative to the W_k product bits, before being added to the accumulator word.**

- **It's the data routing that accounts for the deleted "wasted" bits in Figure 8!**

- **Let's look at this "parallel" method in action.**

"Parallel" Example

- **Implementing the same 29-tap lowpass filter as in the "serial" method example yields the performance curves in Figure 11.**

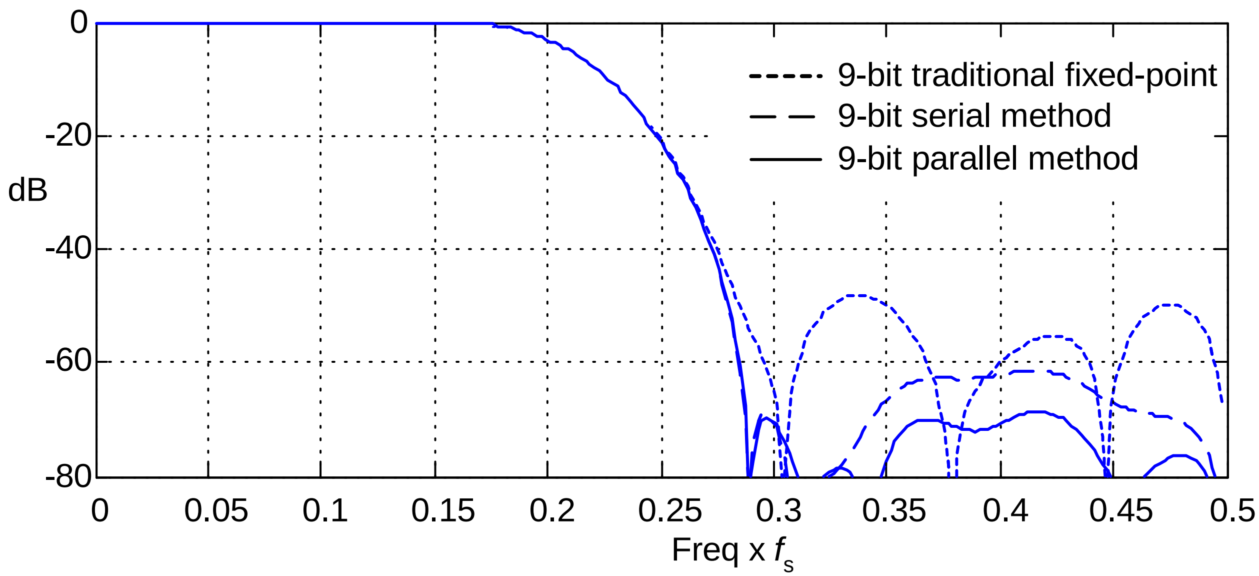


Figure 11

- **Relative to the "serial method" (dashed curve) implementation, the "parallel" method (solid curve) provides:**
  - **even further-improved stopband attenuation.**
  - **Again, without increasing either the bitwidths of our filter's coefficients, or the number of coefficients.** ☺

- **Siskel and Ebert would give this parallel method "Two Thumbs Up."**

Choosing the Number of Bits in Variable Bitwidth Coefficients

- **There are algorithms for determining the number of bits in the variable bitwidth coefficients.**
  - **One algorithm for the "serial" method coeffs. in Figure 3,**
  - **and another algorithm for the "parallel" method coeffs. in Figure 7.**

- **Those algorithms are a bit too intricate (too grueling) to cover in a Conference presentation such as this.**

- **Those algorithms will be published in the "DSP Tips & Tricks" column,**
  - **in the July 2010 issue of the IEEE Signal Processing Magazine.**

- **If you want to learn those algorithms <u>before</u> July, send me an E-mail,**
  - **at: <R.Lyons@ieee.org>.**

- **Please be aware that the Copyrights to the figures in this presentation are, this month, being transferred to the IEEE.**

- **This entire filter coefficient-enhancement idea is not mine.**

- **This is the idea of Zhi Shen.**
  - **Ph.D degree student with the Department of Electronics and Information Engineering, Huazhong Univ.Sci. & Tech., Wuhan, P.R. China.**

- **As far as I know, Mr. Shen has implemented these improved-precision coefficient methods,**
  - **on an Altera FPGA.**