Zhi Shen

Improving FIR Filter Coefficient Precision

"DSP Tips and Tricks" introduces practical design and implementation signal processing algorithms that you may wish to incorporate into your designs. We welcome readers to submit their contributions. Contact Associate Editors Rick Lyons (R.Lyons@ieee.org) or C. Britton Rorabaugh (dspboss@aol.com).

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. The improved performance is accomplished without increasing either the number of coefficients or coefficient bitwidths. At first thought, such a process does not seem possible, but this article shows exactly how this novel filtering process works.

TRADITIONAL FIR FILTERING

To describe our method of increasing FIR filter coefficient precision, let's first recall a few characteristics of traditional linear-phase tapped-delay line FIR filter operation.

Consider an FIR filter whose impulse response is shown in Figure 1(a). For computational efficiency reasons (reduced number of multipliers), we implement such filters using the folded tapped-delay line structure shown in Figure 1(c) [1].

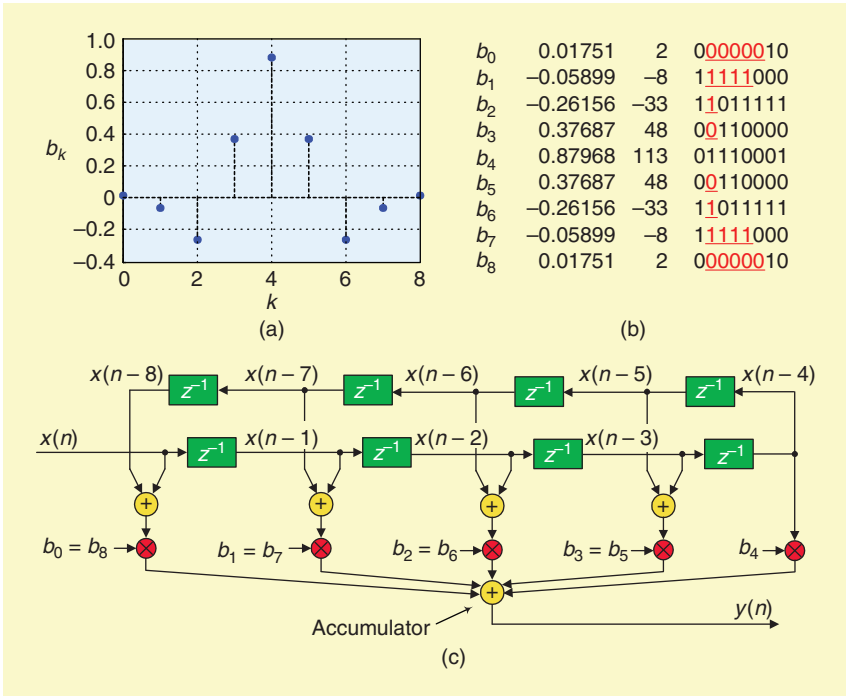
The filter's b_k floating-point coefficients are listed in the second column of Figure 1(b). When quantized to an 8-b two's-complement format, those coefficients are the decimal integers and binary values shown in the third and fourth columns, respectively, in Figure 1(b).

Compared to b_4, the other coefficients are smaller, especially the outer coefficients such as b_0 and b_8. Because of the fixed bitwidth quantization, many high-order bits of the low-amplitude coefficients, the red-font underscored bits in the fourth column of Figure 1(b), are the same as the sign bit. These bits are wasted because they have no effective (no weight) in the calculations. If we can remove those wasted bits (consecutive bits adjacent to, and equal to, the sign bit), and replace them with more significant coefficient bits, we will obtain improved numerical precision for the low-amplitude beginning and ending coefficients.

Replacing a low-amplitude coefficient's wasted bits with more significant bits is the central point of our FIR filtering trick—of course some filter architecture modification is needed as we shall see. So let's have a look at a generic example of what we call a "serial" implementation of our trick.

SERIAL IMPLEMENTATION

As a simple example of replacing wasted bits, we list the Figure 1(b) b_k coefficients as the floating-point numbers in the upper left side of Figure 2. Assume we quantize the maximum-amplitude coefficient, b_4, to 8 b. In this FIR filter trick we quantize the lower-amplitude coefficients to larger bitwidths than the maximum coefficient (b_4) as shown on



[FIG1] **Generic linear-phase FIR filter: (a) impulse response, (b) coefficients, and (c) structure.**

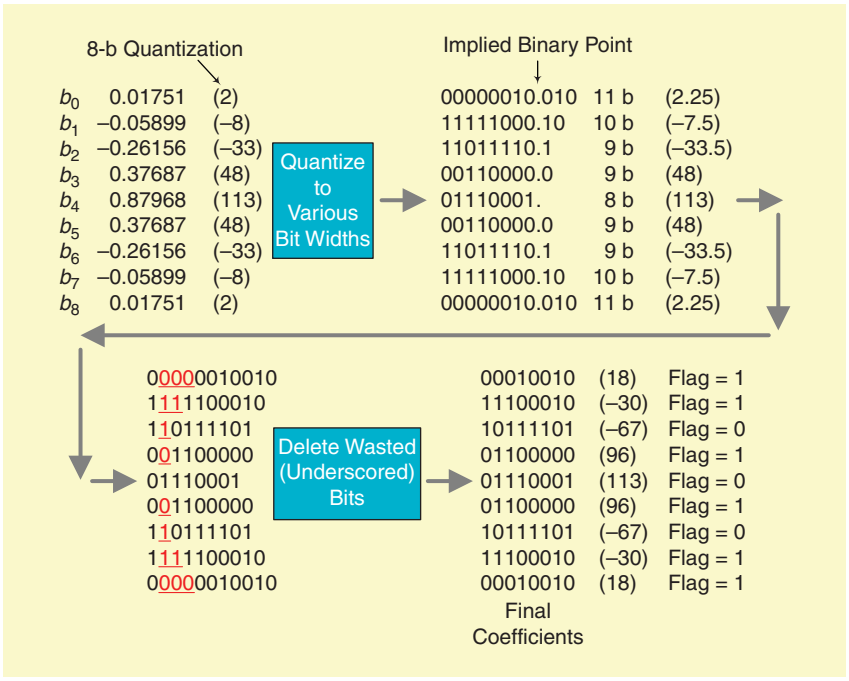
Digital Object Identifier 10.1109/MSP.2010.936777

1053-5888/10/$26.00©2010IEEE

the upper right side of Figure 2. (The algorithm used to determine those variable bitwidths is discussed later in this article.) Next, we eliminate the appropriate wasted bits, the red-font underscored bits in the lower left side of Figure 2, to arrive at our final 8-b coefficients shown on the lower right side of Figure 2.

Appended to each coefficient is a flag bit that indicates whether that coefficient used one more quantization bit than the previous, next larger, coefficient.

Now, you may say: "Stop! You can't do this. The outer coefficients are left shifted, so they are enlarged, and the product accumulations are changed. Using these modified coefficients, the filter results will be wrong!" Don't worry, we correct the filter results by modifying the way we accumulate products. Let's see how.



[FIG2] **Filter coefficients for serial implementation.**

The coefficients and flag bits from Figure 2 are used in the serial implementation shown in Figure 3. The data registers in Figure 3 represent the folded delay-line elements in Figure 1(c). This implementation is called "serial" because there is only one multiplier and, when a new $x(n)$ input sample is to be processed, we perform a series of multiplications and accumulations (using multiple clock cycles) to produce a single $y(n)$ filter output sample.

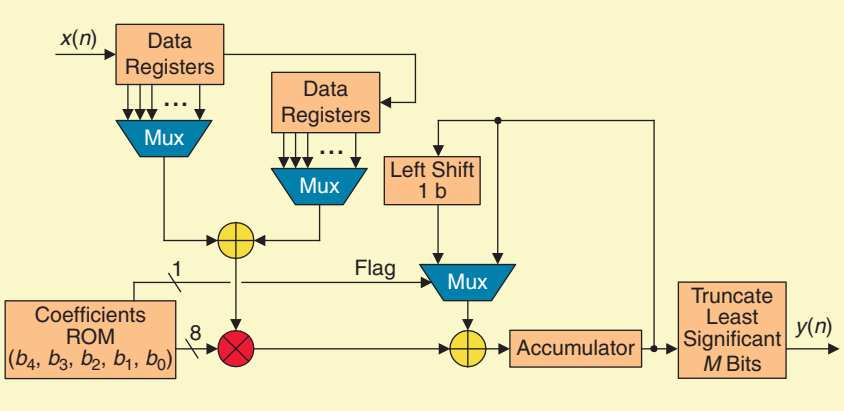
For an N-tap FIR filter, where N is odd, due to our folded delay-line structure only $(N+1)/2$ coefficients are stored in the coefficient read-only memory (ROM). (When N is even, $N/2$ coefficients are stored.) Crucial to this FIR filter trick is that when processing a new $x(n)$ input sample, the largest coefficient, b_4, is applied to the multiplier prior to the first accumulation. Following that is the next smaller coefficient, b_3, and so on. In other words, in this serial implementation the coefficient sequence applied to the multiplier, for each $x(n)$ input sample, is in the order of the largest to the smallest coefficient.

Given these properties, when a new $x(n)$ sample is to be processed we clear the current accumulator value and multiply the sum of the appropriate data registers by the b_4 coefficient. That product is then added to the accumulator. On the next clock cycle 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 then the current multiplier's output is added to the shifted accumulator value. (If the current coefficient's flag bit is zero the accumulator word is not shifted prior to an addition.) We continue these multiplications, possible left shifts, and accumulations for the remaining b_2, b_1, and b_0 coefficients.

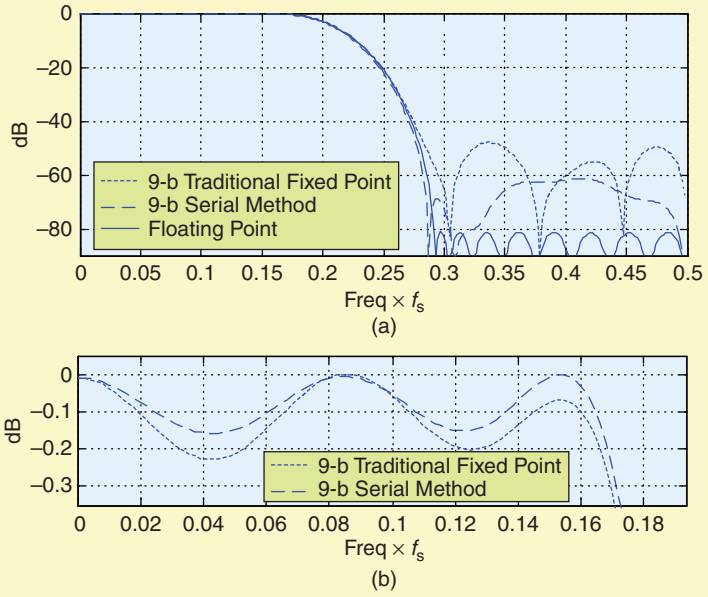
THE LEFT SHIFTING OF AN ACCUMULATOR VALUE IS THE KEY TO THIS ENTIRE FIR FILTER TRICK.

The left shifting of an accumulator value is the key to this entire FIR filter trick. To minimize truncation errors due to right shifting a multiplier output word, we preserve precision by left shifting the previous accumulator word.

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



[FIG3] **Serial implementation with 8-b coefficients.**



[FIG4] Low-pass serial method filter frequency responses: (a) full frequency range and (b) passband detail.

in the ROM memory, to produce a $y(n)$ output sample. Now let's have a look at an actual FIR filter example.

SERIAL METHOD EXAMPLE

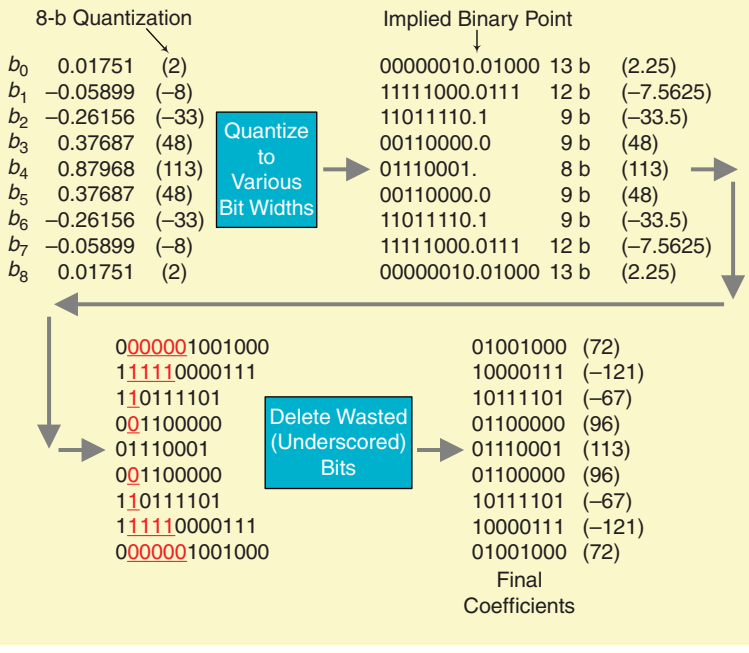
Suppose we want to implement a low-pass filter whose cutoff frequency is $0.167f_s$ and whose stopband begins at $0.292f_s$, where f_s is the input data sample rate. If the filter has 29 taps (coefficients), and is implemented with floating-point coefficients, its frequency magnitude response will be that shown by the solid curve in Figure 4(a). Anticipating a hardware implementation using an Altera field-programmable gate array (FPGA) having 9-b multipliers, when using coefficients that have been quantized to 9-b lengths in a traditional manner (with no wasted coefficient bits removed), the filter's frequency magnitude response is the dotted curve in Figure 4(a).

When we use our FIR filter trick's serial implementation, with its enhanced-precision 9-b coefficients (not counting the flag bit) obtained in the manner shown in Figure 2, the filter's frequency magnitude response is the dashed curve in Figure 4(a). We see in the figure that, relative to the traditional fixed point implementation, the serial method provides:

- improved stopband attenuation
- reduced transition region width
- improved passband ripple performance.

All of these improvements occur without increasing the bitwidths of our filter's multiplier or coefficients, nor the number of coefficients. Because it preserves the impulse response symmetry of the original floating-point filter, the serial implementation filter exhibits phase linearity [2].

It is possible to improve upon the stopband attenuation of our compressed-coefficient serial method FIR filter. We do so by implementing what we call the "parallel method."



[FIG5] Filter coefficients for parallel implementation.

PARALLEL IMPLEMENTATION

In the above serial method of filtering, adjacent filter coefficients were quantized to a precision differing by no more that one bit. That's because we use a single flag bit to control the 1-b shifting of the accumulator word prior to a single accumulation. In the parallel method, described now, adjacent coefficients can be quantized to a precision differing by more than 1 b. Figure 5 shows an example of our parallel method's coefficient quantization process.

Again we list the Figure 1(b) b_k coefficients as the floating-point numbers in the upper left side of Figure 5. In this parallel method, however, notice that the expanded quantized b_1 and b_2 words differ by more than one

bit in the upper right side of Figure 5. Coefficients b_2 and b_6 are quantized to 9 b while the b_1 and b_7 coefficients are quantized to 12 b. While we only deleted some of the wasted coefficient bits in Figure 2, in our parallel method all the wasted coefficient bits are deleted. As such, our final 8-b coefficients are those listed in the lower right side of Figure 5.

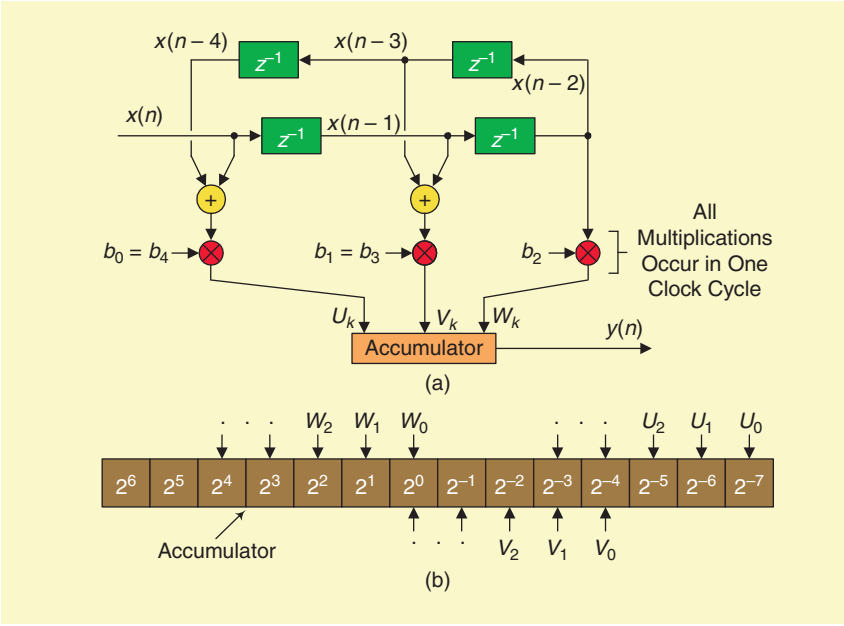
We are all familiar with the operation known as bit extension—the process of extending the bit length of a binary word without changing its value or sign. With that process in mind, we can refer to our trick's operation of removing wasted bits as "bit compression."

Because no flag bits are used in the parallel method, this filtering method is easiest to implement using FPGAs with their flexible multidata bus routing capabilities.

For example, consider the filter structure shown in Figure 6(a) where we perform the three multiplications in parallel (in a single clock cycle) and that is why we use the phrase "parallel method." Instead of shifting the accumulator word to the left as we did in the serial method, here we merely reroute the multiplier outputs to the appropriate bit positions as they are added to the accumulator word as shown in Figure 6(b). In our hypothetical Figure 6 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 7 b, relative to the W_k product bits, before being added to the accumulator word.

PARALLEL METHOD EXAMPLE

With the solid curve, Figure 7 shows our parallel method's performance in implementing the desired low-pass filter used in the above serial method implementation example. For comparison, we have also included the 9-b traditional fixed point (no bit compression) and the serial method magnitude responses in Figure 7.



[FIG6] **Parallel method implementation: (a) filter structure; (b) accumulator organization.**

The enhanced precisions of the parallel method's quantized coefficients, beyond their serial method precisions, yield improved filter performance. The parallel method of our FIR filter trick

IT IS POSSIBLE TO IMPROVE UPON THE STOPBAND ATTENUATION OF OUR COMPRESSED-COEFFICIENT SERIAL METHOD FIR FILTER.

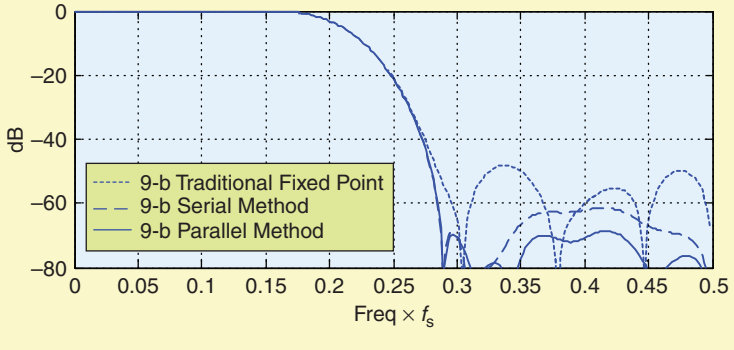
achieves a stopband attenuation improvement of 21 dB beyond the traditional fixed-point implementation—again, without increasing the bitwidths of our filter's multipliers or coefficients, nor the number of coefficients.

COMPUTING COEFFICIENT BITWIDTHS

Determining the bitwidths of the quantized filter coefficients in our DSP trick depends on whether you are implementing the serial or the parallel filtering method.

SERIAL METHOD COEFFICIENT QUANTIZATION

In the serial method, let's assume we want our ROM to store coefficients whose bitwidths are integer B (not counting the flag bit).



[FIG7] **Traditional fixed-point, serial method, and parallel method filter frequency responses.**

[TABLE 1] SERIAL METHOD QUANTIZATION EXAMPLE.

COEFFICIENT BEING QUANTIZED	CURRENT SCALE	ALL UNQUANTIZED COEFFICIENTS LESS THAN SCALE/2?	NEW SCALE	K	ROM EQUIVALENT COEFFICIENT BITWIDTH	FLAG BIT	LEFT SHIFT AND ROUND
$b_4 = 0.87968$	1	NO $\lvert b_4\rvert > 1/2$	1	7	8	0	$B_4 = \text{ROUND}\,[b_4 \times 2^7] = 113$
$b_3 = 0.37687$	1	YES $\lvert b_3\rvert, \lvert b_2\rvert, \lvert b_1\rvert, \lvert b_0\rvert < 1/2$	0.5	8	9	1	$B_3 = \text{ROUND}\,[b_3 \times 2^8] = 96$
$b_2 = -0.26156$	0.5	NO $\lvert b_2\rvert > 0.5/2$	0.5	8	9	0	$B_2 = \text{ROUND}\,[b_2 \times 2^8] = -67$
$b_1 = -0.05899$	0.5	YES $\lvert b_1\rvert, \lvert b_0\rvert < 0.5/2$	0.25	9	10	1	$B_1 = \text{ROUND}\,[b_1 \times 2^9] = -30$
$b_0 = 0.01751$	0.25	YES $\lvert b_0\rvert < 0.25/2$	0.125	10	11	1	$B_0 = \text{ROUND}\,[b_0 \times 2^{10}] = 18$

The steps in computing the integer ROM coefficients for the serial method are as follows:

- *Step 1*: Set a temporary scale factor variable to SCALE = 1 and temporary bitwidth integer variable to $K = B - 1$. Apply the following quantization steps to the largest-magnitude original b_k floating-point coefficient (for example, b_4 in the upper left side of Figure 2).
- *Step 2*: If the b_k floating-point coefficient being quantized and all the remaining unquantized coefficients are less than the value SCALE/2, set SCALE = SCALE/2, set $K = K + 1$, and set the current coefficient's flag bit to Flag = 1. If the b_k floating-point coefficient being quantized or any of the remaining unquantized coefficients are equal to or greater than SCALE/2, variables SCALE and K remain unchanged, and set the current coefficient's flag bit to Flag = 0.
- *Step 3*: Multiply the b_k floating-point coefficient being quantized by 2^K and round the result to the nearest integer. That integer is our final value saved in ROM.
- *Step 4*: Repeat Steps 2 and 3 for all the remaining original unquantized b_k floating-point coefficients, in sequence from the remaining largest-magnitude to the remaining smallest-magnitude coefficient.

Table 1 illustrates the serial method quantization steps for the floating-point coefficients in Figure 2.

PARALLEL METHOD COEFFICIENT QUANTIZATION

In the parallel method, let's assume we want our ROM to store coefficients whose bitwidths are integer B. (For example, in the lower right side of Figure 5, $B = 8$.) Next, let's define an optimum magnitude range, R, as

$$0.5 \le R < 1. \qquad (1)$$

THE SO-CALLED SERIAL METHOD OF FILTERING IS COMPATIBLE WITH TRADITIONAL PROGRAMMABLE DSP CHIP AND FPGA PROCESSING.

The steps in computing the integer ROM coefficients for the parallel method are as follows:

- *Step 1*: Repeatedly multiply an original b_k floating-point coefficient (the upper left side of Figure 5) by two until the magnitude of the result resides in the optimum magnitude range R. Denote the number of necessary multiply-by-two operations as Q.
- *Step 2*: Multiply the original b_k floating-point coefficient by 2^{B+Q-1} (the minus one in the exponent accounts for the final coefficient's sign bit) and round the result to the nearest integer. That integer is our final value saved in ROM.
- *Step 3*: Repeat Steps 1 and 2 for all the remaining original b_k floating-point coefficients.

CONCLUSIONS

We introduced two novel methods for improving the precision of the fixed-point coefficients of FIR filters. Using the modified (compressed) coefficients, we achieved enhanced filter performance while maintaining phase linearity, without increasing the bitwidths of our filter multiplier or coefficients, nor the number of coefficients. The so-called serial method of filtering is compatible with traditional programmable DSP chip and FPGA processing, while the parallel method is most appropriate with an FPGA implementation.

ACKNOWLEDGMENTS

Many thanks to Rick Lyons for his careful analysis of these filtering methods and his patient assistance with the text of this article.

AUTHOR

Zhi Shen (zhi.m.shen@gmail.com) is pursuing the Ph.D degree with the Department of Electronics and Information Engineering, Huazhong University of Science and Technology, Wuhan, China. As the project leader of the Digital TV Lab, he designed the multicarrier DVB modulator.

REFERENCES

[1] R. Lyons, *Understanding Digital Signal Processing*, 2nd ed. Englewood Cliffs, NJ: Prentice-Hall, 2004, pp. 503–504.

[2] J. Proakis and D. Manolakis, *Digital Signal Processing—Principles, Algorithms, and Applications*, 3rd ed. Englewood Cliffs, NJ: Prentice-Hall, 1996, pp. 620–621.

[SP]