Quantized FIR Filter Design: A Collaborative Project for Digital Signal Processing and Digital Design Courses

Kishore A. Kotteri, Amy E. Bell, Joan E. Carletta

Virginia Tech / Virginia Tech / The University of Akron

Abstract

We describe a collaborative, team-based filter design project for students in two senior-level electrical and computer engineering courses: Digital Signal Processing (DSP) and Digital Design (DD). The objective of the project is to replicate real-world issues involved in the design and implementation of digital filters. These issues can not be fully addressed without simultaneously considering both signal processing and hardware aspects of the design. First, the DSP team members design a digital filter to meet given filter specifications; next, the DD team members implement the filter in hardware. Finally, the entire team works together to verify and validate their design. A basic solution should be attainable for all of the teams; however, the more advanced teams will be able to refine their design to achieve better hardware performance without sacrificing signal processing performance. We describe the basic solution as well as a refinement. The project addresses joint hardware-algorithm issues like: filter structure, gain compensation, filter coefficient quantization, zero compensation, and canonical signed digit representation of coefficients.

Introduction

The typical electrical and computer engineering curriculum includes courses in both digital signal processing and digital design, but treats the two subjects completely separately. Students in a digital signal processing (DSP) course learn various techniques, such as windowing and the equiripple Park-McClellan methods, for the design of finite and infinite impulse response (FIR, IIR) filters subject to specifications such as passband ripple, stopband attenuation and passband/stopband edge frequencies. A typical DSP filter design project uses MATLAB, MathCAD, or C to create a floating-point design as the end product. The design is considered high quality if it meets the frequency response specifications while minimizing complexity (measured in terms of the order of the filter). On the other hand, students in a digital design (DD) course learn how to implement digital hardware using various hardware description

languages and platforms. A typical DD project might work with a digital filter, starting with a set of filter coefficients, and ending with a hardware implementation. Because fixed-point mathematics can be implemented in faster, smaller hardware than floating-point mathematics, one of the first steps in the design process is to convert the given filter into a fixed-point format. The quality of the hardware design is measured in terms of hardware performance metrics that include size, throughput and power consumption.

This standard approach effectively educates the DSP and DD students separately, but misses the opportunity to reflect the more realistic, industrial design process where filter design and hardware implementation of filters are intertwined. In DSP classes, filters are designed in floating-point software on a personal computer without any consideration of the challenges involved in a fixed-point hardware implementation of the designed filter. In DD classes, hardware implementation changes that increase throughput and decrease size and power consumption are made without much consideration of how those changes alter the filter's signal processing performance. In the real world, the signal processing and hardware aspects must be considered *simultaneously* in order to end up with a high quality filter implementation. The accuracy of the final fixed-point filter implementation depends on both the accuracy of the floating-point design and the approximations made during the conversion from floating-point to fixed-point. When hardware constraints are absent (such as when a floating-point, general purpose processor is the target computational platform), there is no concern regarding the impact of fixed-point approximations on filter performance. However, when hardware limitations exist, so that it is necessary to use fixed-point hardware, the conversion to fixed-point must be made such that the degradation in the resulting filter's performance is minimized (i.e., the performance is as close to the original floating-point filter's performance as possible).

In this paper, we describe a finite impulse response (FIR) filter design project that can be used in collaboration between DSP and DD courses. The project goal is the design and implementation of a fast, high-quality FIR filter subject to both given filter specifications and hardware constraints. Each team is comprised of DSP and DD students. The following sections describe a sample project, a basic solution that every team should be able to achieve, and one possible refinement that leads to an improved solution for advanced teams.

The Design Problem

The problem requires the design of a linear phase, FIR, lowpass filter with a passband edge corresponding to a normalized frequency (such that a frequency of one corresponds to the sampling rate) of f=0.25. The filter specifications are: a stopband attenuation of -20dB and a peak-to-peak passband ripple of 1.25dB. For our example, we use the windowing method (with a rectangular window) to design the floating-point filter, and we use an Altera field programmable gate array (FPGA) as the computational platform. The performance of the fixed-point filter design as implemented in hardware is evaluated with the following metrics.

1. *Mean-squared error (MSE)*: the average of the squared difference between the magnitude response of the fixed-point (quantized) filter and the floating-point (unquantized) filter over all frequencies.
2. *Hardware size*: the total number of logic cells used on the FPGA.

3. *Throughput*: the rate at which output samples are generated, in samples per second.
4. *Latency*: the elapsed time from the first filter input to the first filter output.
5. *Power consumption*: the average power needed to compute one output sample.

The Design Process and Some Considerations

The recommended overall design approach is shown in the flowchart of Figure 1. Boxes outlined in blue are steps led by the DSP team members; those outlined in green are led by the DD team members, while boxes outlined in both colors are the responsibility of the entire team.

This approach to the filter design problem helps the student teams achieve a basic, working design relatively fast. The iteration in this process allows a team to reconsider their design and investigate alternative designs that offer improved performance; it also promotes interaction and learning between the DSP and DD team members. The iterative process stops when a team believes they have achieved a satisfactory trade-off between hardware performance and filter performance.

Advanced students can create quite sophisticated designs by realizing that both signal processing and hardware design choices combine to play a large role in the quality of the resulting system. Some of the primary design choices that need to be considered can be grouped into the following two categories.

Choice of filter structure: The three most common FIR filter structures are direct, cascade, and lattice. For FIR filters with symmetric coefficients (a typical choice for many applications, because such filters have linear phase characteristics), the lattice structure cannot be used. The direct structure usually performs well, as long as the filter zeros are not very clustered [1], [2]. However, the cascade structure offers at least one advantage when the goal is a fixed-point implementation. When an FIR filter is quantized using a direct structure, the quantization of *one* coefficient affects *all* of the filter's zeros. In contrast, if an FIR filter is quantized with a cascade structure, the quantization of coefficients in one of the cascaded sections affects only those zeros in its section—the zeros in the other cascaded sections are isolated and unaffected. We will exploit this separation of zeros to achieve a closer-to-ideal frequency response for the fixed-point filter implementation than would otherwise be possible.

Choice of hardware architecture: The structure of the hardware impacts its size, speed, and power consumption. Choices range from a very simple but slow scheduling of the multiplication operations serially on a single multiple-accumulate (MAC) unit, to a much faster fully parallel implementation. One popular method of implementing efficient high performance, fully parallel filters is to use a multiplierless architecture, replacing all coefficient multiplications by cheaper and faster shifts and additions. We use a multiplierless architecture in our example.

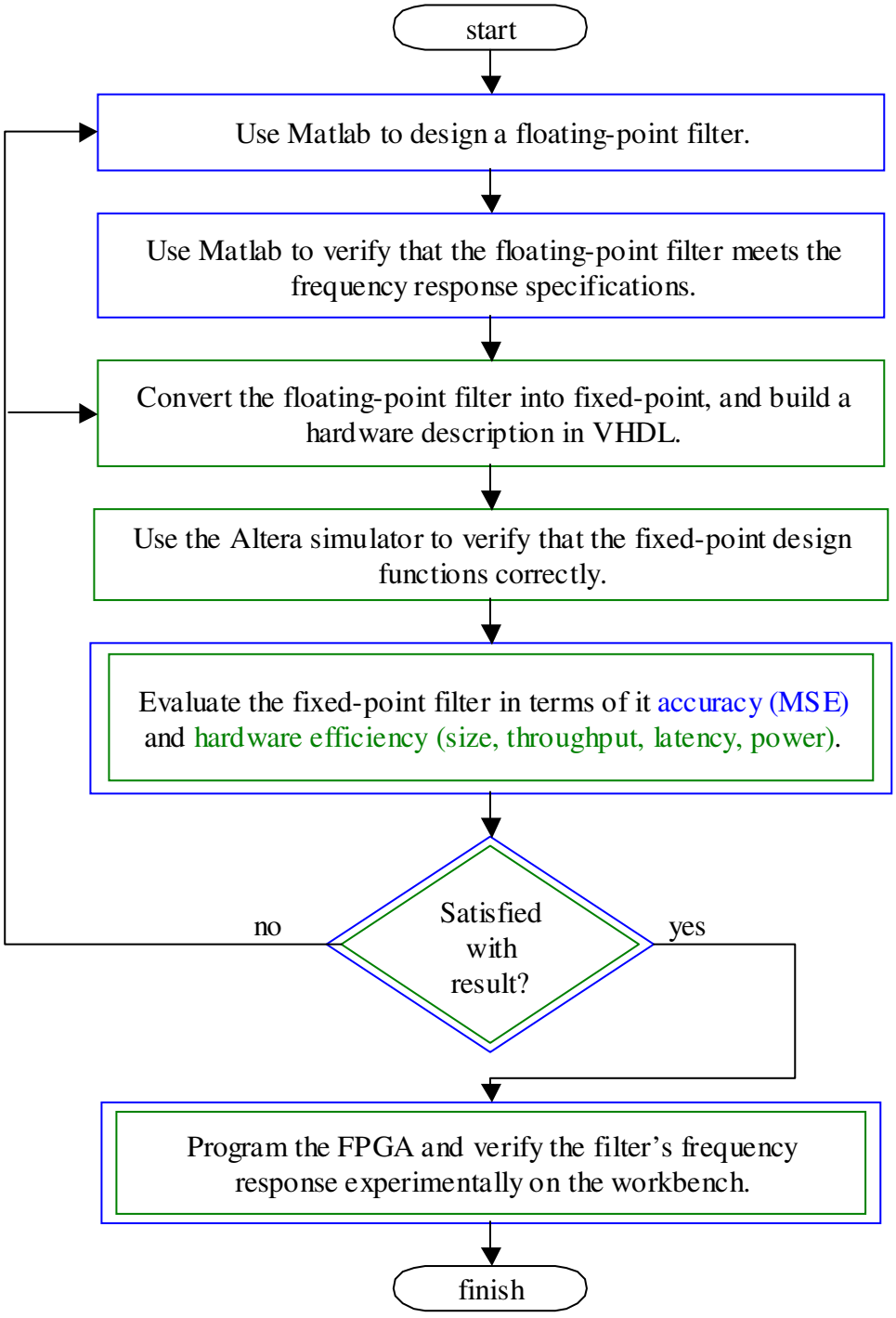

Figure 1: Flowchart of sample project.

Proceedings of the 2004 American Society for Engineering Education Annual Conference & Exposition

The goal of the quantized filter design is to achieve a small magnitude mean-squared error (MSE) while keeping hardware costs low. In general, the more closely the floating-point coefficients are approximated in fixed-point (by using more bits of precision), the more hardware the system will require, and the smaller the magnitude MSE. Hence, there is a trade-off between signal processing performance and hardware cost.

Floating-point design

Table 1 lists the floating point coefficients for a lowpass, symmetric, length-19 FIR filter that meets the required specifications. It was designed in Matlab using a rectangular window. Figure 1 shows the magnitude response of this floating-point (unquantized) filter design.

The direct filter structure is straightforward: it is comprised of the coefficients in Table 1. A cascade structure separates the zeros of the filter into a number of separate sections, in this case two sections, with coefficients $c_1(n)$ and $c_2(n)$. Figure 2 illustrates the pole-zero plot for the original $h(n)$. To separate the zeros of $h(n)$ into the two cascaded sections, the z-plane is scanned from $\omega = 0$ to $\omega = \pi$. As the zeros are encountered, they are placed alternately in the two sections. The first zero encountered is at $z = 0.66e^{0.324\,j}$. This zero, its conjugate, and the two reciprocals are put in one section. The next zero at $z = 0.69e^{0.978\,j}$, its conjugate, and the reciprocal pair are placed in the other section. This proceeds until all of the zeros of the unquantized filter are divided among the two cascade sections. It is best to make the section with fewer zeros the first section in the cascade, $c_1(n)$, and the section with more zeros the second section, $c_2(n)$. In Figure 2 the eight blue zeros go to $c_1(n)$, which has length nine, and the ten red zeros go to $c_2(n)$, which has length eleven. This method of splitting up the zeros has the advantage of keeping the zeros relatively spread out within each section, thereby minimizing the quantization effects within each section. Because complex conjugate pairs and reciprocal pairs of zeros are kept together in the same cascade section, the two resulting sections have symmetric, real-valued coefficients.

Table 1. Unquantized windowed FIR filter coefficients, *h(n)*.

n	*h*(*n*)
0, 18	0.03536776513153
1, 17	-1.94908591626e-017
2, 16	-0.04547284088340
3, 15	1.94908591626e-017
4, 14	0.06366197723676
5, 13	-1.9490859163e-017
6, 12	-0.10610329539460
7, 11	1.94908591626e-017
8, 10	0.31830988618379
9	0.50000000000000

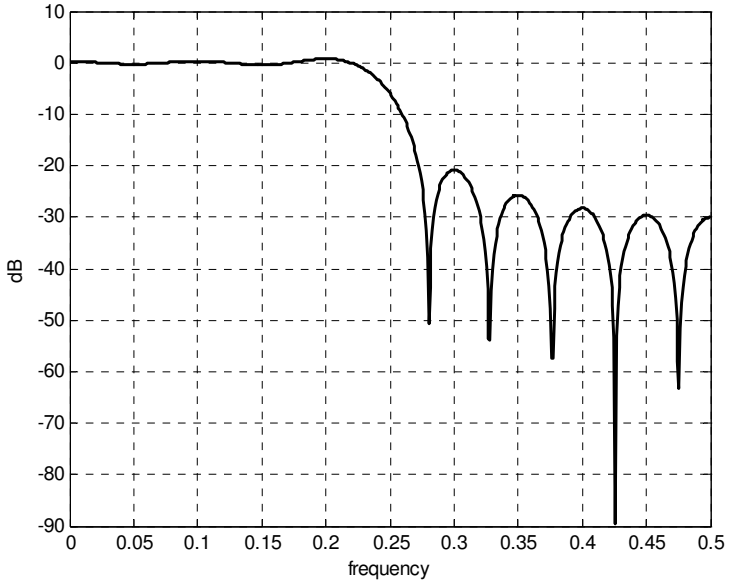

Figure 1: Frequency magnitude response of $h(n)$ for the unquantized FIR filter.

The resulting floating-point cascade $c_1(n)$ and $c_2(n)$ coefficients are shown in Table 2. Sections $c_1(n)$ and $c_2(n)$ have been normalized so that the first and last coefficients in each section have value one; this will help later when we move the filter to fixed-point, since at least two of the coefficients in each cascade section can be implemented with essentially no multiplication. As a result of the normalization, it is necessary to include a gain factor, k in Table 2, following the

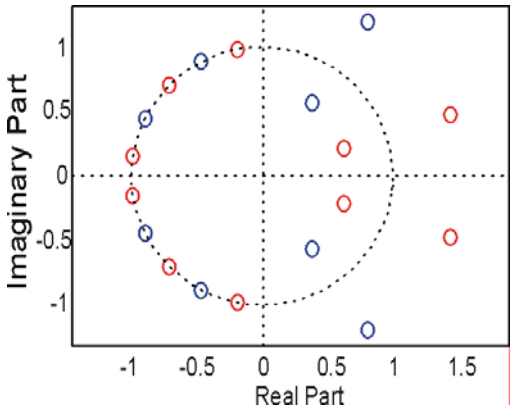

Figure 2: Pole-zero plot for $h(n)$. The zeros are divided into two cascaded sections by placing the blue zeros in the first section $c_1(n)$, and the red zeros in the second section $c_2(n)$.

cascade sections. Figure 3 depicts the block diagram for the original $h(n)$, along with a diagram for the equivalent cascaded form. Before quantization, both the original direct structure and the cascade structure have identical magnitude response shown in Figure 1.

Table 2. Unquantized and simple-quantized cascaded coefficients for $h(n)$.							
Unquantized				simple-quantized with T=25			
$c_1(n)$		$c_2(n)$		$c'_1(n)$ (T = 9)		$c'_2(n)$ (T = 14)	
n_1		n_2		Decimal	CSD	Decimal	CSD
0, 8	1.0000000	0, 10	1.0000000	1.00	001.00	1.000	001.000
1, 7	0.3373269	1, 9	-0.3373269	0.25	000.01	-0.250	000.0$\underline{1}$0
2, 6	0.9886239	2, 8	-2.1605488	1.00	001.00	-2.000	0$\underline{1}$0.000
3, 5	1.9572410	3, 7	-0.8949404	2.00	010.00	-1.000	00$\underline{1}$.000
4	3.0152448	4, 6	1.8828427	4.00	100.00	1.875	010.00$\underline{1}$
		5	3.5382228			3.500	100.$\underline{1}$00
	k = 0.0353678			k' = 0.0351563		0.00001001 (T = 2)	

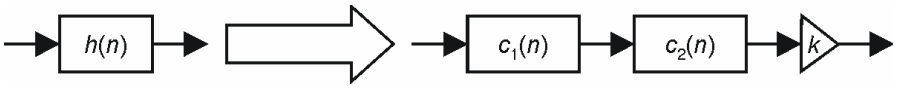

Figure 3: Direct form of $h(n)$ and the equivalent cascade form using $c_1(n)$, $c_2(n)$ and k.

Hardware design

Once the design of the floating-point filter is complete, the coefficients of the chosen structure must be quantized to fixed-point, so that a hardware implementation can be designed. In what follows, we describe a high performance, multiplierless architecture implemented on an Altera FPGA. We facilitate easy exploration of filters that use different fixed-point coefficients by providing filter synthesis software, that given a set of fixed-point coefficients generates a synthesizable description of the filter in the hardware description language VHDL; this software and documentation for its use is available on our web site [4]. However, the project could use other architectures, either written by hand by the project team, or generated by commercial FIR filter compilers.

Multiplierless implementations use what is called a canonical signed digit (CSD) representation for the filter coefficients. Each quantized filter coefficient is expressed as a sum of positive and negative powers of two; because multiplication by a power of two is simply a data shift, multiplication by a coefficient can then be replaced by shifts and additions or subtractions. Each non-zero power of two in a coefficient corresponds to a term that must be added or subtracted. (By allowing negative powers of two in the representation, CSD is able to use fewer terms – two-thirds fewer on average – than a standard binary representation [3].) Before a quantized filter design is implemented in hardware, the hardware complexity (size, etc.) is estimated in terms of T, the total number of non-zero terms used when writing all of the filter coefficients in CSD format. For application specific integrated circuit and FPGA filter implementations, a fully parallel hardware implementation requires T-1 adders. In general, the smaller T is, the smaller

and faster the hardware implementation is, but the more error is inherent in approximating the floating-point coefficients, and so the larger the magnitude mean-squared error (MSE) is.

The VHDL description generated by our filter synthesis tool is a structural one. For each section in a filter, a chain of registers is used to shift the data in, and the data is shifted in accordance with the filter coefficients before being summed. For example, if one of the filter coefficients were $18 = 2^4+2^1$, the corresponding data word would go to two shifters and be shifted four and one places, respectively, before being summed. A pipelined tree of carry save adders (CSAs) is used for the summation. The CSA tree produces two outputs that must be summed, or "vector-merged", to produce the final filter output. For the results presented here, we use a ripple carry adder for the vector merge, taking care to exploit special purpose routing ("carry chains") provided on Altera FPGAs to make ripple carry addition fast. The software automatically chooses appropriate fixed-point bit widths for internal signals such that errors due to truncation and overflow are avoided.

Using the filter synthesis tool or another FIR filter compiler, the hardware design process can be simply one of specifying the fixed-point values of the coefficients, running the compiler to generate VHDL, and synthesizing that VHDL into hardware so that the system's size, speed and power consumption can be determined. Use of a filter compiler makes it easy to investigate the effects of different quantization schemes on system quality. We now describe two different quantizations of the cascade structure of Table 2. Both use similar amounts of hardware; this is achieved by setting T, the number of terms used when representing all filter coefficients in CSD (i.e., the coefficients for both cascaded sections and for the gain), to 25 for both systems. Twenty-five terms results in small hardware while still providing a reasonable approximation to the desired filter. The first quantization is simple but results in some degradation of the frequency response of the filter, while the second, which requires the application of more sophisticated digital signal processing ideas, gives a higher quality result.

A Simple Quantization

The process of quantizing the floating-point filter coefficients involves allocating T=25 CSD terms among all of the coefficients in the two unquantized cascaded sections and the unquantized gain factor. Allocating more terms to a particular coefficient makes it possible to approximate that coefficient more closely. While it is possible to allocate any number of terms to any coefficient, some distributions of terms to coefficients are clearly unreasonable; for example, it would not make sense to use all T terms for one coefficient, and none for the others (setting them all to zero). Reasonable distributions are "mostly uniform": all coefficients receive roughly the same number of terms (here, at least one). Once each coefficient has been allocated one term, extra terms are generally best allocated to those coefficients that are most different (in terms of percent difference) from their unquantized values. The fixed-point value of each coefficient is then found by choosing the closest value that can be represented in CSD using the allocated number of terms.

In the design that we describe here, we search through all "reasonable" distributions to find the one that gives the best result, measured as the smallest mean-squared error (MSE) in the magnitude of the frequency response of the resulting system. (Note: this does not require an optimization technique; for reasonably small values of T, it is simple to organize a search that

looks in the area around the floating-point coefficients, which is the only area where high quality solutions lay). Alternatively, for a student project, students could simply allocate CSD terms as they see fit. Once they have decided on a distribution, a simple Matlab analysis can show the impact of their decision on the frequency response. By trying a number of distributions, they can better understand the role that quantization plays in the frequency response.

In applying the simple quantization method to the windowed FIR filter example, the unquantized cascade coefficients, $c_1(n)$ and $c_2(n)$, are independently quantized to the simple quantized cascade coefficients, $c'_1(n)$ and $c'_2(n)$. Based on the relative lengths of the sections, nine CSD terms are used for $c'_1(n)$, fourteen terms are used for $c'_2(n)$, and two terms are used for the quantized gain factor k'. The resulting simple quantized coefficients are listed in Table 2 (in the CSD format, an underline indicates that the power of two is to be subtracted instead of added).

The frequency response of the simple quantized filter, $c'_1(n)$ and $c'_2(n)$, is compared to the unquantized filter, $h(n)$, in Figure 4. Although a linear phase response is retained after simple quantization (i.e. the simple quantized coefficients are symmetric), the magnitude response is significantly different from the original, unquantized case. The passband peak-to-peak ripple for the quantized filter is 3.24dB and the stopband attenuation is -18.5dB: both in violation of the desired filter specifications.

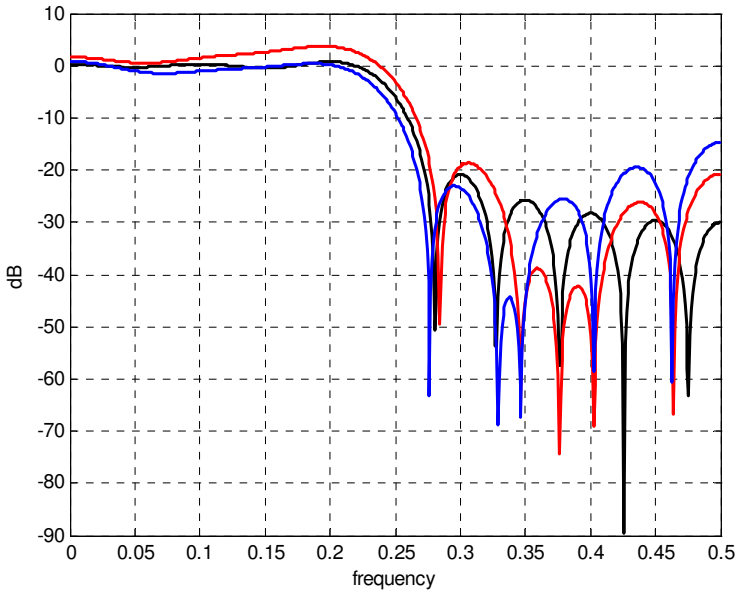

Figure 4: Frequency responses of the unquantized windowed filter $h(n)$ (black), simple quantization (red), and compensating zeros quantization (blue).

By the time they have completed this simple-quantization design, the students have seen the effects of quantization on the filter, and have a baseline to which to compare more sophisticated alternatives. We describe one possible alternative next.

One Refinement

As illustrated in the previous section, a simple approach to quantization of the filter coefficients results in a degradation of the desired floating-point filter frequency response. This section presents a refinement to the cascaded design, called 'compensating zeros', that generates an alternative fixed-point filter design with a closer match to the floating-point frequency response.

The compensating zeros quantization method takes the quantization error of the first cascaded section into account when quantizing the second section. The key to this method is the desire for the frequency response of the quantized cascade structure to match the frequency response of the original, unquantized filter. In essence, we quantize the second section in such a way that it offsets some of the quantization error introduced by the first section.

Quantization using the compensating zeros method begins by using the simple quantization method of the previous section to quantize the first cascade section $c_1(n)$ to $c'_1(n)$ and the gain factor k to k'. We then modify the second section: instead of working with the original second section $c_2(n)$, we compute a new second section $c_{\text{comp}}(n)$ such that $c'_1(n)$ cascaded with $c_{\text{comp}}(n)$ is as close as possible to the original filter $h(n)$. We call coefficients $c_{\text{comp}}(n)$ the *compensating section*, since their aim is to compensate for the performance degradation resulting from the quantization of $c'_1(n)$; the computation of $c_{\text{comp}}(n)$ is developed below. The computed compensating section has floating-point coefficients, and so simple quantization must be used to translate it to fixed-point before hardware implementation.

If $C_1(z)$, $C_2(z)$, $C'_1(z)$, $C'_2(z)$ and $C_{\text{comp}}(z)$ are the transfer functions of $c_1(n)$, $c_2(n)$, $c'_1(n)$, $c'_2(n)$ and $c_{\text{comp}}(n)$ respectively, then the transfer function of the unquantized cascaded filter $H(z)$ can be written as

$$H(z) = kC_1(z)C_2(z). \tag{1}$$

where k is the gain factor. The transfer function of the semi-quantized filter using the compensating zeros method is given by

$$H'_{\text{comp}}(z) = k'C'_1(z)C_{\text{comp}}(z). \tag{2}$$

$H'_{\text{comp}}(z)$ is called the semi-quantized filter because $c_{\text{comp}}(n)$ has floating-point coefficients. The goal is for the semi-quantized and unquantized transfer functions to be equal, that is, $H'_{\text{comp}}(z) = H(z)$, or

$$k'C'_1(z)C_{\text{comp}}(z) = kC_1(z)C_2(z). \tag{3}$$

In (3), the quantities on the right-hand side are the known, unquantized cascade filters. $C'_1(z)$ and k' on the left-hand side are also known; they come from the already quantized first cascade section and gain. Thus, (3) can be solved for $c_{\text{comp}}(n)$. Our $c_{\text{comp}}(n)$ has eleven taps, and is symmetric and normalized, so that the first and last coefficients have value 1. Therefore, it can expressed in terms of only five unknowns. In general, for a length-N symmetric filter with normalized leading and ending coefficients, there are M unique coefficients, where

$M = \lceil (N\text{-}2)/2 \rceil$. (The $\lceil x \rceil$ notation means: the next integer larger than x; or if x is an integer, $\lceil x \rceil = x$.)

The five unknowns in $c_{comp}(n)$ can be found by evaluating Equation (3) at $M = 5$ values of z. Since the frequency response is the primary concern, it makes sense to choose five values of z on the unit circle. For our example, (3) is solved at the frequencies $f = 0$, 0.125, 0.2, 0.3 and 0.45 (i.e. $z = 1$, $e^{j0.25\pi}$, $e^{j0.4\pi}$, $e^{j0.6\pi}$, $e^{j0.9\pi}$); other frequencies could be chosen if it were important to match the frequency response of the quantized filter to the unquantized original at particular key points. Table 3 lists the computed floating-point coefficients of $c_{comp}(n)$. Note that the compensating section is significantly different from the original second section, which was shown in Table 2. The compensating section is the ideal second section for the filter, given the quantization decisions that were made for the first and gain sections. MATLAB code for computing the compensating coefficients, and documentation for its use, can be found at our web site [4].

Once $c_{comp}(n)$ has been obtained, it is quantized (using simple quantization and the remaining T) to arrive at $c'_2(n)$. The final, quantized $c'_2(n)$ filter coefficients using this compensating zeros method are also given in Table 3. Thus the compensating zeros quantized filter coefficients are the $c'_1(n)$ and k' from Table 2 in cascade with the $c'_2(n)$ in Table 3.

Table 3. Unquantized $c_{comp}(n)$ and compensating zeros-quantized $c'_2(n)$ for $h(n)$ ($T = 25$).

N	$c_{comp}(n)$	$c'_2(n)$ ($T = 14$)	
		Decimal	CSD
0, 10	1.0000000	1.00	001.00
1, 9	-0.7266865	-0.75	00$\underline{1}$.01
2, 8	-1.1627044	-1.00	00$\underline{1}$.00
3, 7	-0.6238289	-0.50	000.$\underline{1}$0
4, 6	1.2408329	1.00	001.00
5	2.8920707	3.00	10$\underline{1}$.00

The frequency response of the compensating zeros quantized filter is compared to the unquantized filter and the simple quantized filter in Figure 4; the overall frequency response of the compensating zeros quantized implementation is closer to the unquantized filter than the simple approach in the previous section. The compensating zeros magnitude response has a peak-to-peak passband ripple of 1.9dB, and stopband attenuation of -14.6dB. The small value of $T = 25$ employed in this example hampers the ability of even the compensating zeros quantized magnitude response to closely approximate the unquantized magnitude response. Increasing T slightly and redesigning a simple quantized filter and compensating zeros quantized filter would be the next step in the design process. A team should try to find the smallest T for which the filter specifications are satisfied. There is an "art" in how to assign the extra T: these terms should be allocated to the coefficients—in either cascaded section—that are most different (in terms of percent different) from their unquantized values. (Another design alternative in the example would be to consider different frequencies for which to solve equation (3). Notice that

the stopband attenuation problem occurs around f=0.5—a match in (3) at a frequency closer to f=0.5 would help solve this problem).

Table 5 summarizes the hardware performance of the filters. Data formats for all signals are shown as (n,l), where n is the total number of bits, including the sign bit, and 2^l is the weight of the least significant bit. Both filters take in inputs of data format (8,0), i.e., eight-bit two's complement integers. They vary in terms of their output data formats, depending on the precision of the coefficients used. Throughput is the most important performance metric; it measures how many inputs can be processed per second. Latency also bears on performance, but is less critical; it measures how many clock cycles are required from the first input to the first output.

The results of Table 4 show that the compensating zeros quantized filter is slightly smaller (fewer logic cells) and faster (higher throughput) than the simple quantized filter. This is because the coefficients for the compensating zeros quantized case turn out to be slightly less wide (in terms of bits) than those for the simple quantized case, so that the adders also turn out to be slightly less wide.

Table 4. Hardware metrics for the windowed FIR example.

	Simple quantized	Compensating zeros quantized
Hardware complexity (logic cells)	1042	996
Throughput (Mresults/second)	82.41	83.93
Latency (clock cycles)	15	15
Input data format	(8, 0)	(8, 0)
Output data format	(23, -13)	(21, -13)

Remarks

For a fixed value of T, we show two ways to quantize the cascaded coefficients: simple and compensating zeros. If T is large enough, then simple quantization can achieve the desired frequency response. However, when T is restricted (i.e. when hardware size and speed are constrained), compensating zeros quantization provides an alternative that outperforms simple quantization. Simple quantization can be readily implemented by the student teams to generate an initial design; compensating zeros compensation is a more sophisticated design that results in improved performance.

Although T is used as an estimate of hardware performance before implementation of the filter in hardware, final performance depends on a number of other factors such as coefficient bit-width and filter structure. An initial filter coefficient quantization design begins with an initial value for T; this serves as a baseline for performance. Subsequent fixed-point designs are quantized with progressively lower T until a design meets the required specifications with the lowest hardware costs.

Overall, the project introduces students to a real-world problem that is truly interdisciplinary in nature; in embedded systems, where the size and speed of a circuit are of paramount importance,

it is important to consider both signal processing and hardware aspects of the design together. The project pulls concepts that students learn in their undergraduate digital signal processing and digital design courses together in a way that yield new insight into both. Design of fixed-point filters is simple enough to incorporate in a senior level class project, yet provides a good illustration of how the design of signal processing algorithms and their hardware implementations are intertwined.

References

[1] J G Proakis, D G Manolakis, Digital Signal Processing Principles, Algorithms, and Applications, Prentice Hall, third edition, 1995.

[2] A. V. Oppenheim, R. W. Schafer and J. R. Buck, *Discrete-Time Signal Processing*, Prentice Hall, second edition, 1999, pp. 366-510.

[3] C. Lim, R. Yang, D. Li, and J. Song, "Signed power-of-two term allocation scheme for the design of digital filters," *IEEE Transactions on Circuits and Systems – II: Analog and Digital Signal Processing*, Vol. 46, No. 5, pp. 577 – 584, May 1999.

[4] http://www.ecpe.vt.edu/ fac_support/DSPCL/asee04.html

Author Biographies

KISHORE A. KOTTERI received his B.S. in electronics engineering from the University of Mumbai in 1997. After graduation he worked for four years as a software consultant with TATA Consultancy Services, Mumbai. Kotteri is currently pursuing a Ph.D. in electrical engineering at Virginia Tech. His research interests are in the areas of digital signal processing, image processing and communications.

AMY E. BELL is an assistant professor in the department of electrical and computer engineering at Virginia Tech. She received her Ph.D. in electrical engineering from the University of Michigan. Bell conducts research in wavelet image compression, embedded systems, and bioinformatics. She is the recipient of a 1999 NSF CAREER award and a 2002 NSF Information Technology Research award; she has also received two awards for teaching excellence.

JOAN E. CARLETTA is an assistant professor in the Department of Electrical and Computer Engineering at the University of Akron. She received her Ph.D. in computer engineering from Case Western Reserve University in 1995. Her research involves the design of digital hardware for applications that require intensive computation. Her work is funded by two NSF Information Technology Research awards.