2013-14

Efficient Parallel Real-Time Upsampling of Ultrasound Vectors

Authors: William D. Richard, Ph.D.

Corresponding Author: wdr@wustl.edu

Abstract: Upsampling is required prior to the summation step in most receive digital beamforming implementations to produce an accurate summed RF line or vector. This is true in both annular and linear array systems where receive echos are digitized first and then time delayed in the digital domain to achieve proper signal alignment. The efficient, parallel, real-time upsampling circuit presented here produces M upsampled values per ADC clock, where M is the desired upsampling factor. A circuit implementation that upsamples by a factor of M=4 is presented as an example of the more general technique.

Type of Report: Other

Efficient Parallel Real-Time Upsampling of Ultrasound Vectors

William D. Richard, Ph.D.

Washington University in St. Louis
One Brookings Drive
St. Louis, MO 63021

Corresponding Author: William D. Richard
Washington University in St. Louis
Campus Box 1045
St. Louis, MO 63130
wdr@wustl.edu

Upsampling is required prior to the summation step in most receive digital beamforming implementations to produce an accurate summed RF line or vector. This is true in both annular and linear array systems where receive echos are digitized first and then time delayed in the digital domain to achieve proper signal alignment. The efficient, parallel, real-time upsampling circuit presented here produces M upsampled values per ADC clock, where M is the desired upsampling factor. A circuit implementation that upsamples by a factor of M=4 is presented as an example of the more general technique.

KEY WORDS: Efficient; parallel; real-time, upsampling

SHORT TITLE: Efficient Parallel Real-Time Upsampling

I. INTRODUCTION

Modern ultrasound systems do receive digital beamforming (DBF) by aligning sampled RF echos from multiple receive channels in the digital domain, unlike early systems that used analog time delays prior to an analog summation.[1] In order to produce an accurate summed RF line or vector, the individual digitized RF echos must be upsampled so that the effective sample rate satisfies the overall system signal-to-noise requirements. Simple interpolation approaches only provide acceptable results for very low frequency imaging systems where the front end analog-to-digital converter (ADC) can run at many times the transducer center frequency. As transducer frequencies push higher and higher, an efficient, parallel, real-time approach to upsampling is needed.

According to the Nyquist-Shannon sampling theorem,[2] any bandwidth-limited signal that has been sampled at twice the highest frequency in that signal is ***completely described*** by the corresponding set of time domain samples. This means that not only are the points on the time domain signal actually sampled by the ADC described by the corresponding set of samples, but every point in between those sampled points is described as well. How one efficiently and accurately generates those “in between” points for use in the summation step is the subject of this paper.

The easiest way, conceptually, to upsample a vector of ultrasound data by a factor of M is to zero pad the discrete Fourier transform (DFT)[3] of the vector with (M-1) times as many zeros as there are actual frequency components and then transform the zero-padded

vector back into the time domain.[3,4] Under the assumption that the original, time-domain signal was bandlimited and the Nyquist-Shannon criteria satisfied, the zero amplitude frequency components added by the padding process would have been zero anyway could the original signal have been sampled fast enough to capture M-times as many samples during the same time period. Once transformed back into the time domain, the new, upsampled data points produced by this method lie perfectly on the original signal waveform, with (M-1) upsampled data points between each of the original samples.

While upsampling by zero padding the DFT is conceptually simple, it requires considerable compute power to perform in real time, especially when a large number of ultrasound vectors must be processed for each image. In this paper, we present a method for designing an efficient, parallel, real-time circuit that produces M upsampled values per ADC clock, where M is the desired upsampling factor, and give performance results for an example circuit realized in a modern field programmable gate array (FPGA). In many systems, this circuit will immediately follow the ADC, although other architectures are also possible.

II. UPSAMPLING BY ZERO-PADDING IN THE TIME DOMAIN

The general concept on which our parallel upsampling technique is based has been termed “windowed Sinc interpolation” by some authors, and it is described in several excellent papers in the literature.[5,6] Using an example signal and circuit, we develop the concepts necessary for the reader to easily understand the basis for our parallel, real-time upsampling circuit.

For the purposes of illustration, consider the example 16 MHz analog signal shown in figure 1. This signal has the form:

$$f(t) = \cos(2 \pi f t) * e^{(t * t)/\text{constant}} . \quad (1)$$

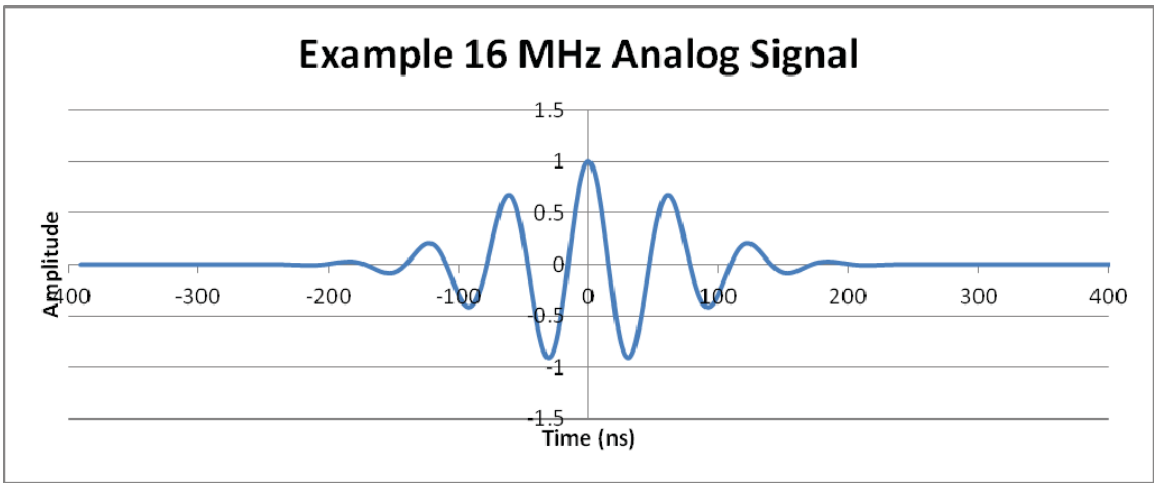

Fig. 1 Example 16 MHz signal used to illustrate the upsampling process.

If the signal shown in figure 1 is sampled at 80 MHz, only five samples are taken per signal period, and the sample data sequence shown in figure 2 results. Using these

samples directly to produce a summed RF vector would provide very poor results, obviously. Upsampling this example data sequence by a factor of four, to an effective sample rate of 320 MHz, would provide twenty samples per signal period, improving the SNR of a summed RF vector formed using the sampled data sequence. While the method described here can be used to upsample by larger factors, upsampling by M=4 will be used here as an example of the more general technique.

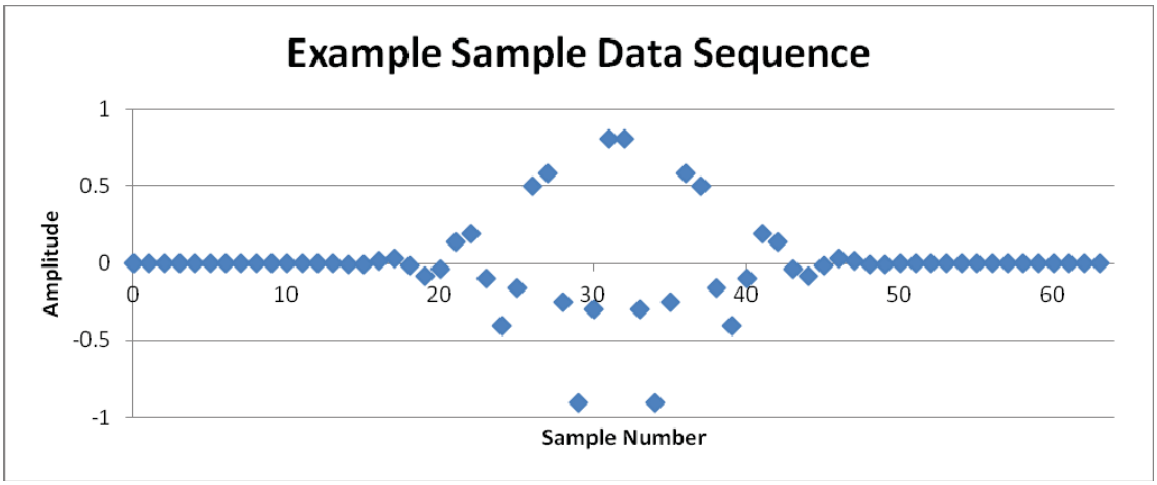

Fig. 2. Example sample data sequence that results from sampling the example analog signal of figure 1 at 80 MHz, or five times per period.

One can generate an (admittedly poorly) upsampled data vector by simply inserting (M-1) zeros in between each actual sample value in the data sequence produced by the ADC as shown in figure 3. This "zero insertion step" corresponds to a replication of the spectrum of the original signal in the frequency domain as shown in figures 4 and 5. And, importantly, it produces a data vector with the desired number of samples. By lowpass filtering the resulting "zero-padded" time domain signal with an ideal brick wall filter so as to eliminate the "replications" of the desired spectrum in the frequency domain, one could theoretically obtain a perfectly upsampled data vector that is identical to the one that would have been produced using zero padding in the frequency domain as described above. In what follows, we develop a realizable, lowpass finite impulse response (FIR) filter and provide performance results.

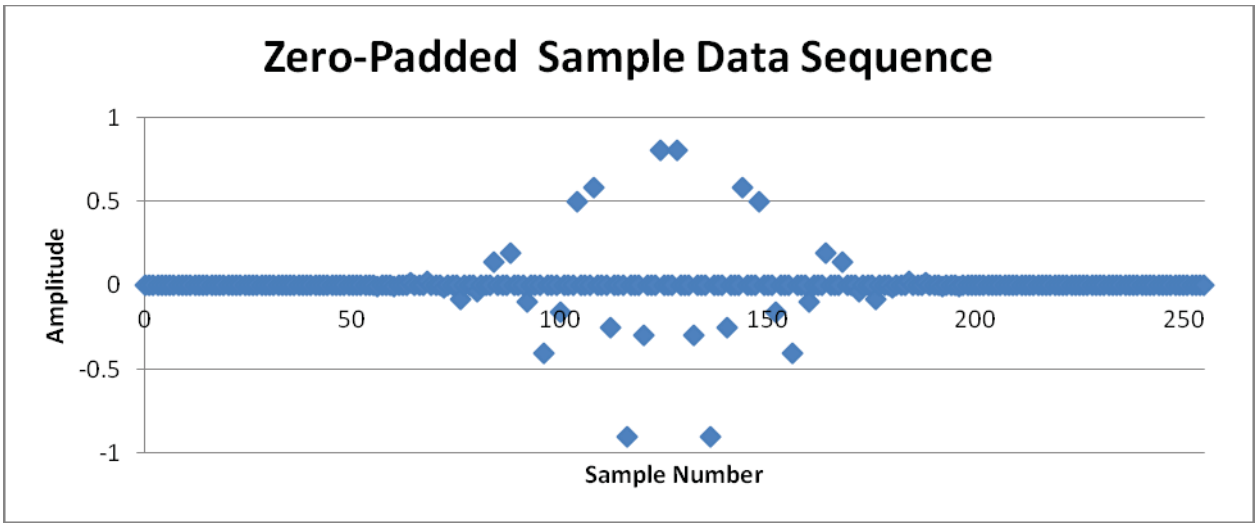

Fig. 3 Zero-padded data sequence produced by inserting M=3 zeros between each actual sample value in the original data sequence of figure 2.

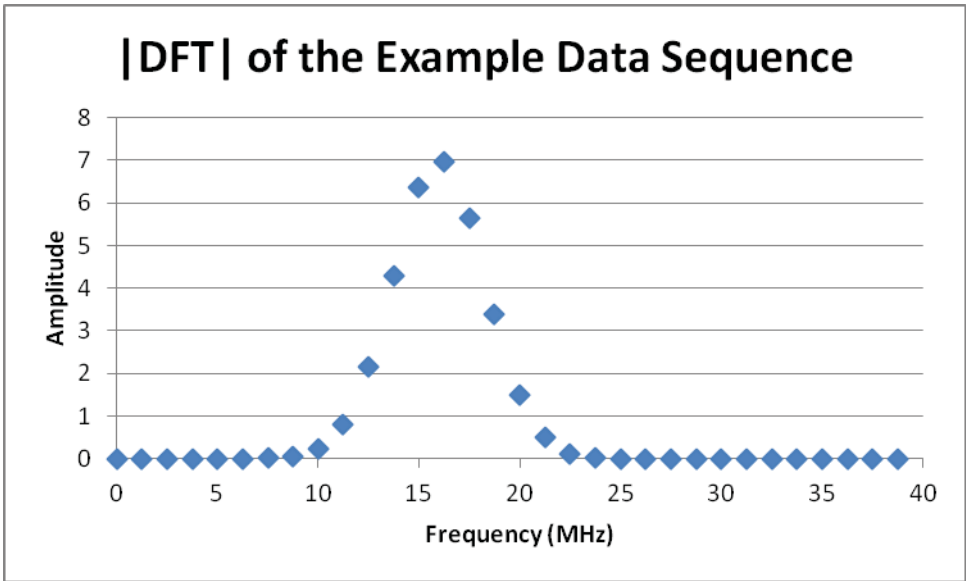

Fig. 4. Magnitude of the DFT of the non-zero-padded example data sequence.

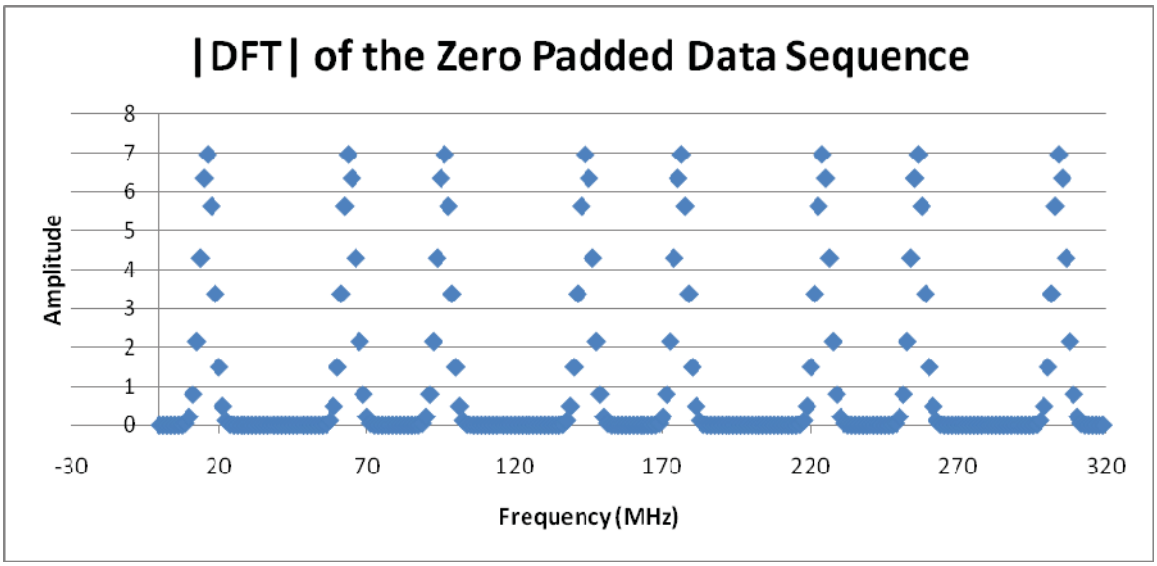

Fig. 5. Magnitude of the DFT of the zero-padded data sequence.

A brick wall lowpass filter in the frequency domain corresponds to convolution in the time domain with a Sinc function. Therefore, if we can run our "zero padded" time domain signal through a symmetric, lowpass FIR filter running at M times the ADC clock rate performing the appropriate convolution operation, as shown in figure 6 for an example 31-tap FIR filter, we can produce our upsampled data vector in real time. In figure 6, R1, R2, …, R31 represent registers clocked at M times the ADC clock rate, and C0, C1, …, C15 represent the coefficients of the FIR filter.

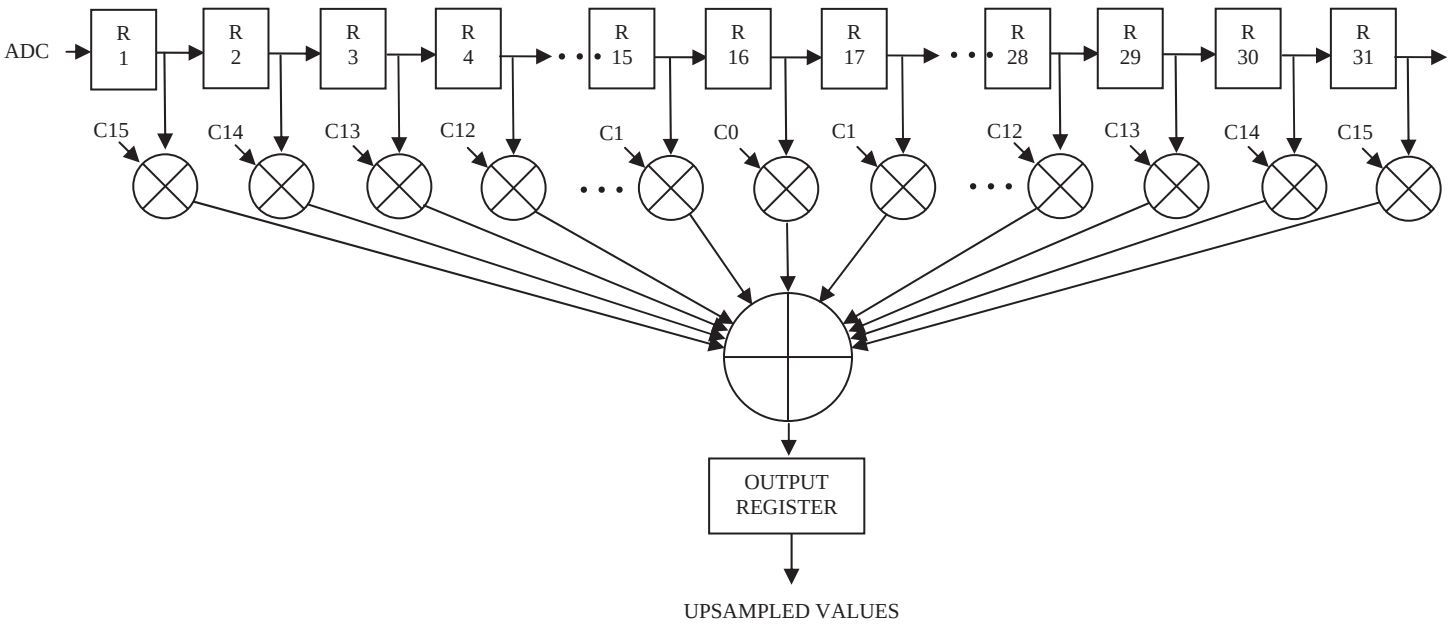

Fig. 6. A 31-tap FIR filter that could be used to generate one upsampled data value per clock period if clocked at M-times the base ADC clock rate.

It is important to note that most of the registers in the FIR filter shown in figure 6 will contain zero, and not actual sample data, during any particular clock interval. For M=4, as an example, when R1 contains actual sample data, R2, R3, and R4 will contain zero. When R1 contains actual sample data, so will R5, R9, R13, R17, R21, R25, and R29, and the remaining registers will contain zero. During the next clock interval, R2, R6, R10, R14, R18, R22, R26, and R30 will contain actual sample data. This observation leads to the parallel implementation discussed below that runs at the base ADC clock rate.

Since it is impossible to produce a perfect brick wall filter response with a finite length FIR filter, our challenge is to design acceptable filters we can implement in available technologies.

While FIR filters can be designed using automated tools like the MATLAB Filter Design Toolbox,[7] in many cases encountered in ultrasonic imaging systems, a conceptually simple approach based on coefficients generated by sampling and windowing a Sinc function will prove adequate. Figure 7 shows a Sinc function sampled optimally for the case M=4 assuming a 31-tap FIR filter topology. In general, for a T-tap, lowpass, FIR filter, the coefficients are given by:

$$C(n) = \mathrm{Sinc}[(n * \pi) / M],\ n = 0 \text{ to } (T-1)/2. \qquad (2)$$

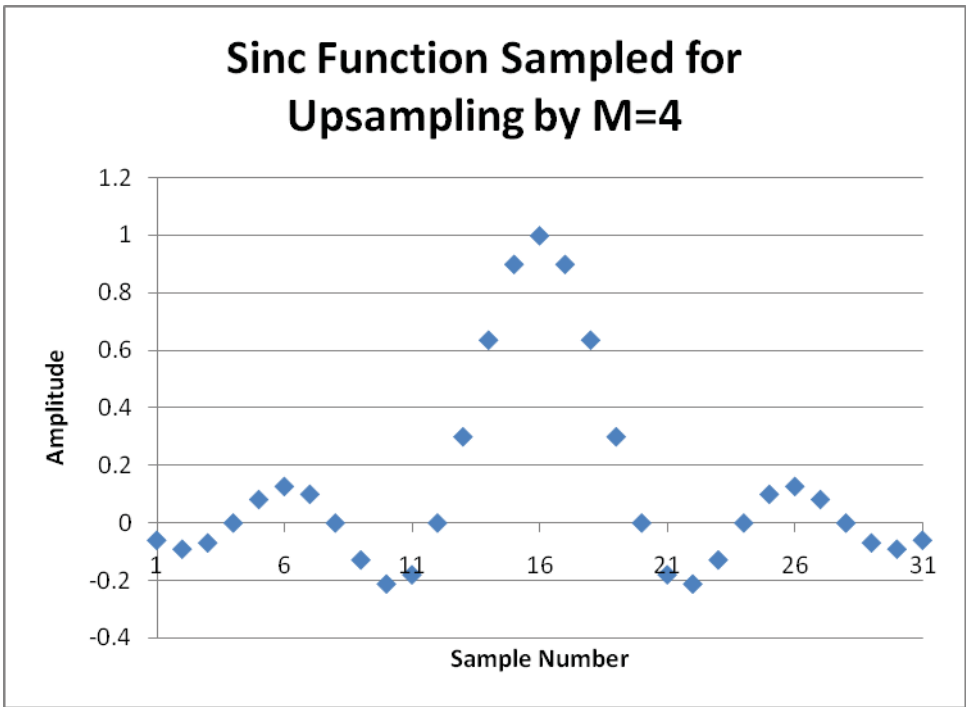

Fig. 7. Sinc function sampled for use with a 31-tap filter when M=4.

Note that the Sinc function is sampled so that every fourth sample is at a zero of the Sinc function (except for the center sample). As these samples are convolved with the zero-padded time domain signal using the FIR filter topology described above, it is conceptually easy to see that the original samples are replicated exactly when they align with the center filter tap. Upsampled values are produced when the original samples do not align with the center FIR filter tap and corresponding filter tap zeros.

Figure 8 shows the frequency response associated with a 31-tap filter designed using the method described above. Since the FIR filter is symmetric, the filter will have linear phase, so only the magnitude of the response is plotted here.

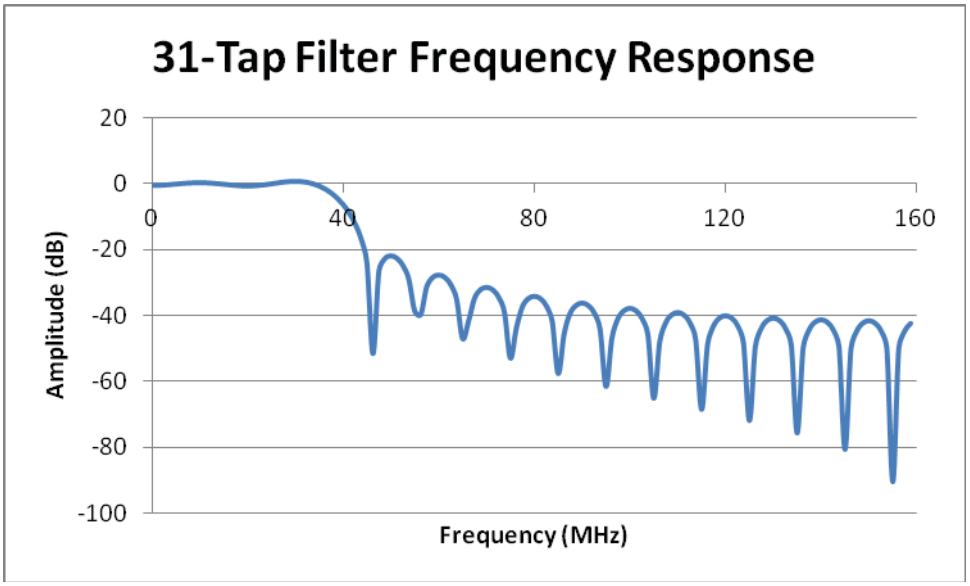

Fig. 8. Frequency response of a 31-tap FIR filter based on a sampled Sinc function.

By applying a Hanning window of the type shown in figure 9 to the sampled Sinc function of figure 7 to produce the windowed, sampled Sinc function shown in figure 10, the filter response can be improved. As shown in figure 11, using coefficients based on a windowed, sampled Sinc function reduces the passband ripple and increases the stopband attenuation at the cost of a slightly wider transition band. Here, the Hanning coefficients are given by:

$$H(n) = [1-(COS((2* \pi * (n + ((T-1)/2)))/ (T-1)))] / 2,\ n = 0 \text{ to } (T-1)/2. \quad (3)$$

The windowed coefficients, $C_w(n)$, are found by multiplying corresponding values of C(n) and H(n), i.e.,

$$C_w(n) = C(n) * H(n),\ n = 0 \text{ to } (T-1)/2. \quad (4)$$

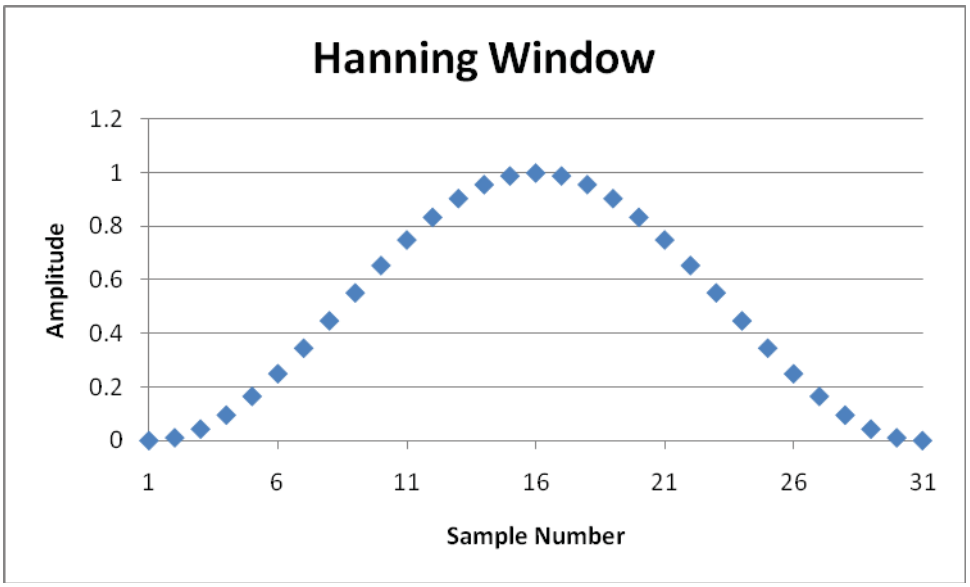

Fig. 9. Hanning window appropriate for a 31-tap FIR filter.

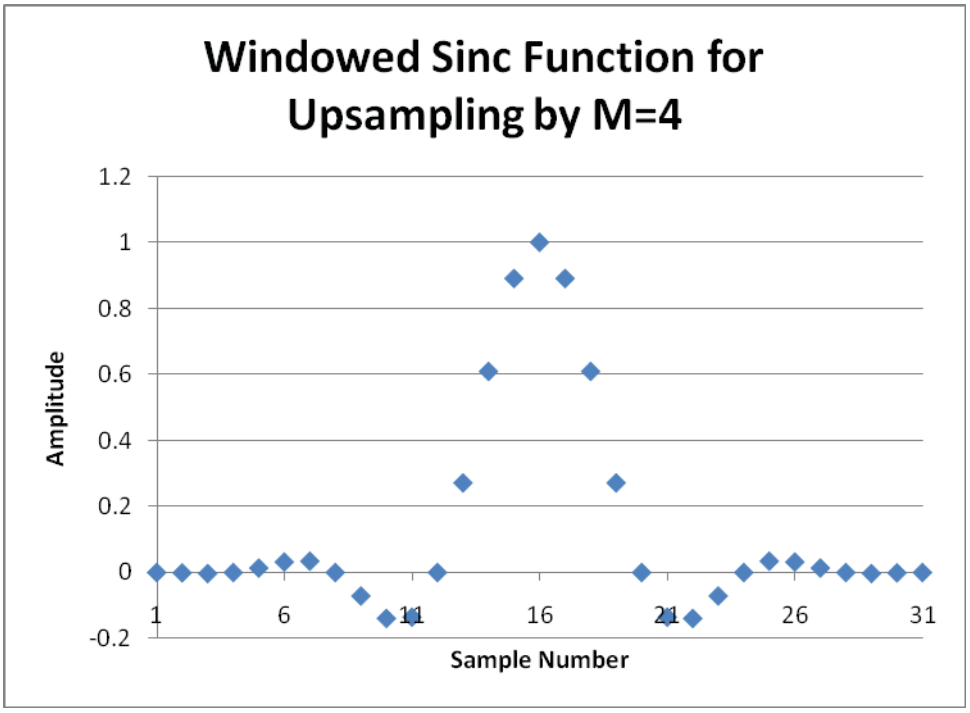

Fig. 10. Windowed Sinc function designed for M=4.

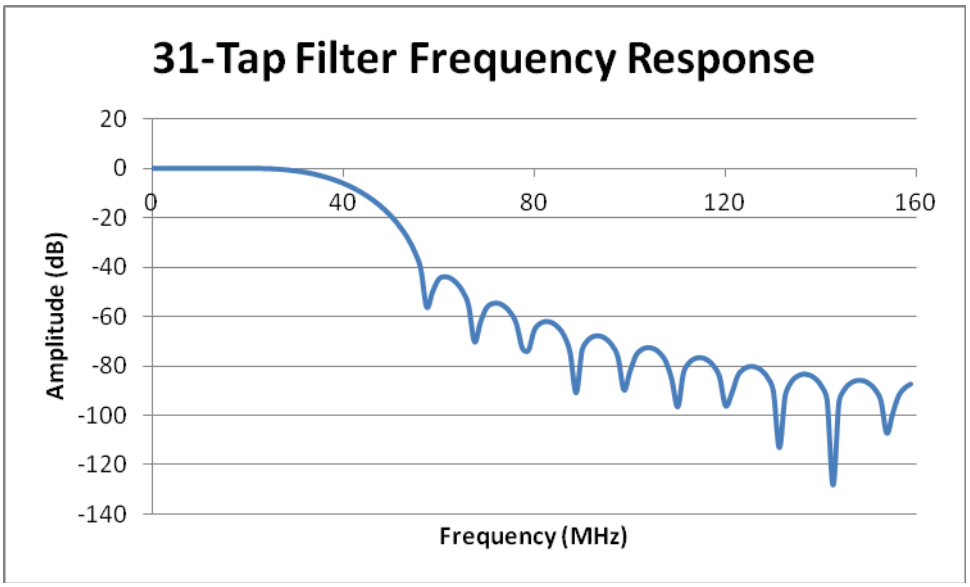

Fig. 11 Frequency response of the 31-tap FIR filter using windowed coefficients.

FIR filters constructed using the methodology presented here will always have a cutoff frequency equal to half that of the base ADC clock, or at 40 MHz in the example presented here where an 80 MHz ADC clock rate was assumed. This is illustrated in figures 8 and 11. And, if we were to run our zero-padded ADC sample data stream through the 31-tap filter clocked at 4 times the base ADC clock rate, the spectral "copies" produced by the zero-padding would be attenuated and the original spectrum passed. The result would be our desired upsampled data vector produced at 320 MHz. We can take advantage of the zeros in the zero-padded data, however, and not require a faster FIR filter clock and generate M samples in parallel as described in the next section.

FIR filters constructed using the windowed Sinc function methodology presented here will also always have very small passband ripple. To illustrate this point, the frequency response of a 63-tap filter designed using the same approach is shown in 12. The 31-tap filter has a worst-case peak passband ripple of 0.0546 dB, while the 63-tap filter has a worst-case peak passband ripple of 0.0535 dB. FIR filters with more taps will, however, have a sharper cutoff as is illustrated in figures 11 and 12.

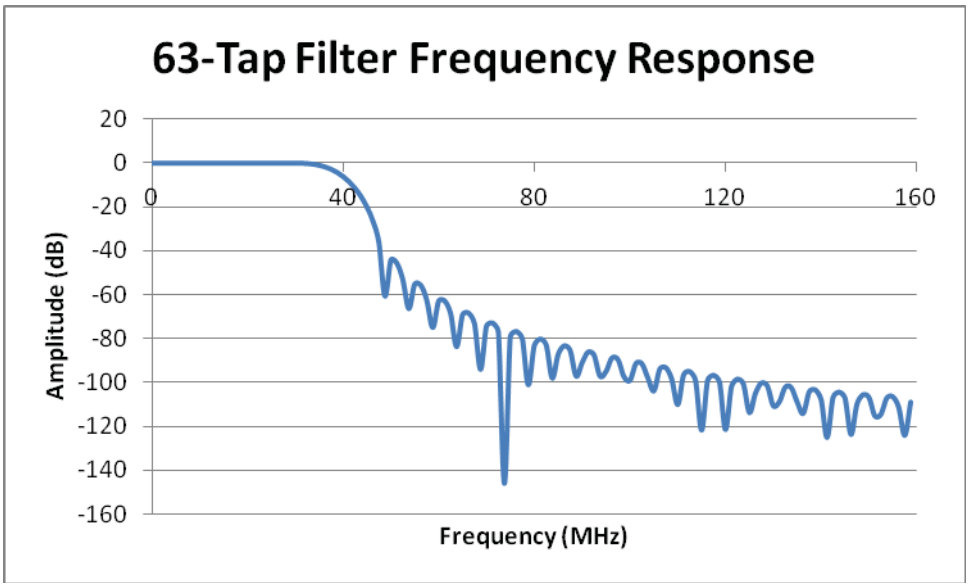

Fig. 12 Frequency response of the 63-tap FIR filter using windowed coefficients.

III. EFFICIENT PARALLEL REAL-TIME UPSAMPLING

Since (M – 1) out of every M samples moving through the FIR filter shown in figure 6 are zero, one can collapse the filter and produce M outputs in parallel as shown in figure 13 for the M=4 case when using a 31-tap FIR filter. With this implementation, the parallel FIR filter runs at the base ADC clock rate, not at M times the ADC clock rate.

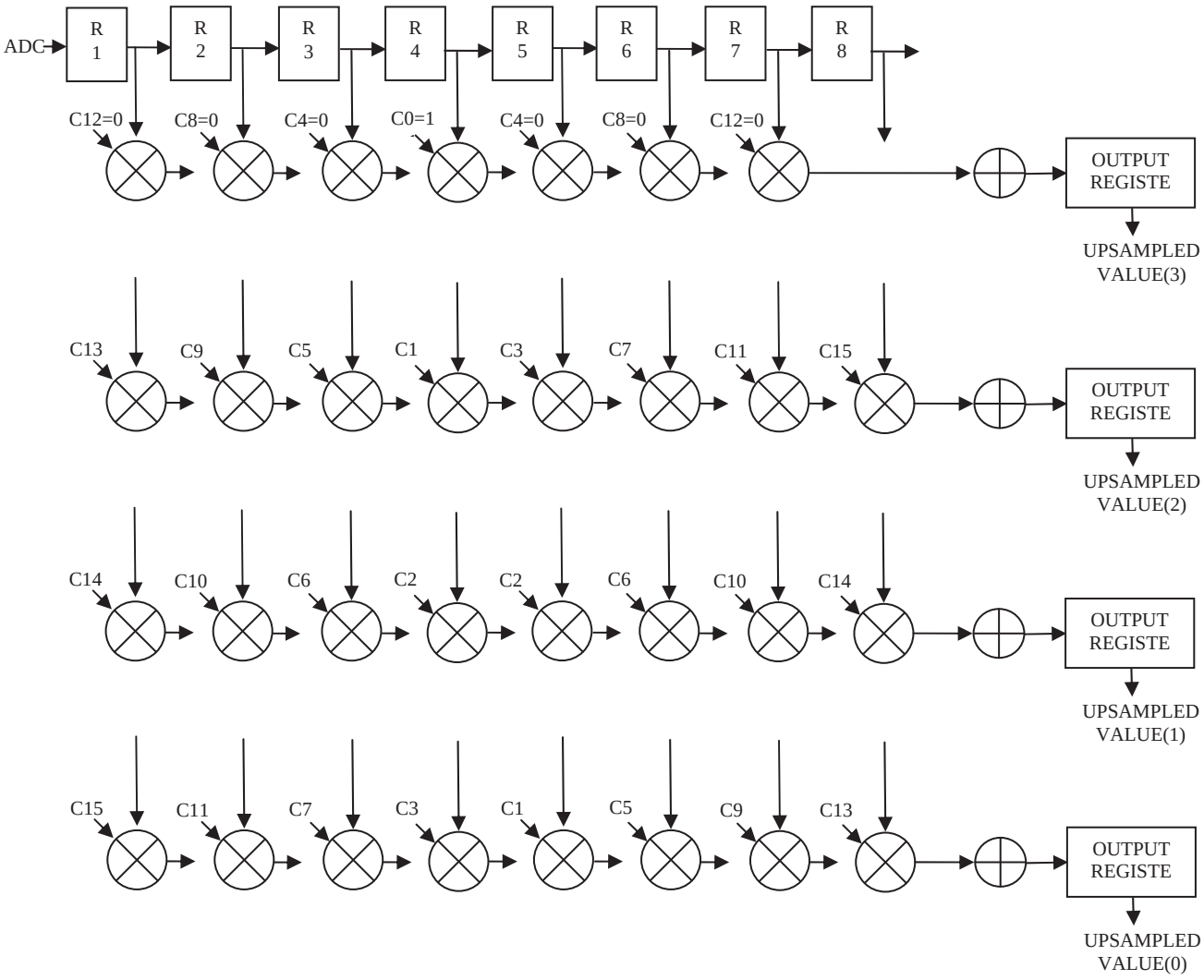

Fig. 13 By observing that only 1 out of every 4 registers in figure 6 holds non-zero data during any given clock period, it is possible to collapse the filter and produce 4 outputs in parallel while running the filter at the base ADC clock rate.

For M=4, when the coefficients for a 31-tap FIR filter are calculated by sampling and windowing a Sinc function as described above, C0 = 1.0, and C4 = C8 = C12 = 0. In addition, if a pure Hanning window (as opposed to a raised Hanning window) is applied to force the windowed function to zero at the ends of the filter, C15 will equal zero as well. If a pure Hanning window is used, then, the three multipliers associated with coefficients C0 and C15 in figure 13 are not needed, simplifying the design and further reducing the number of multipliers required. In addition, by recognizing that each

coefficient is used twice to generate UPSAMPLED VALUE(2), one can “fold” the implementation, add R1 to R8 before multiplying, for example, and eliminate four additional multipliers. The end result, as shown in figure 14, is a design that requires a total of only 18 multipliers to produce four upsampled values per clock period. If a raised windowing function is used, 20 multipliers are required to implement the 31-tap filter (since C15 will be non-zero). It is important to note that each original sample value exits the parallel filter unmodified, a result of the filter design technique described above.

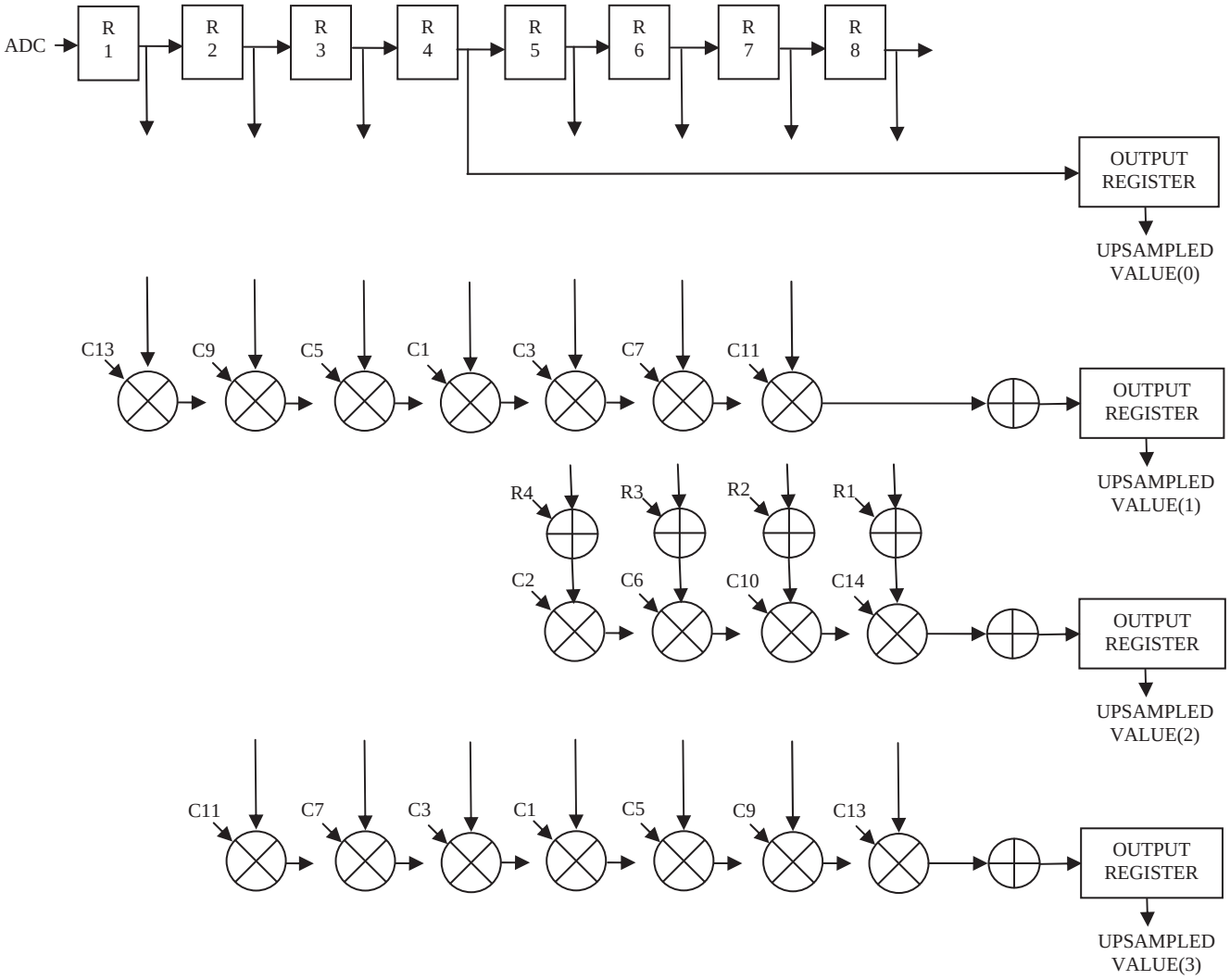

Fig. 14 The final example 31-tap FIR filter design uses eight data registers, seven adders, 18 multipliers, and four output registers.

IV. RESULTS

Figure 15 plots the upsampled data sequence that results when the zero-padded data sequence shown in figure 3 is processed by the 31-tap FIR filter shown in figure 4 using filter coefficients defined by Eq. (4). Each of the original samples has exactly the same value as before, as explained above, and three new samples have been inserted on the original waveform between each actual sample. Figure 16 plots the percent full scale error associated with the upsampled values produced by the filter. The worst case error is 0.342% of the full scale range for this 31-tap (M=4) filter design. For the 63-tap filter of figure 12, the worst case error is only 0.0216% of the full scale range.

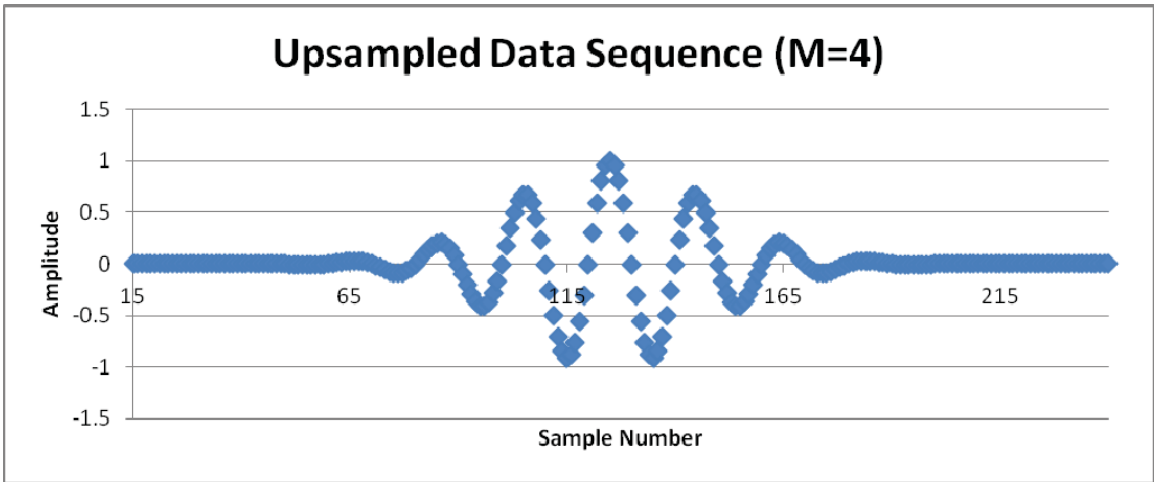

Fig. 15. The upsampled data sequence produced by the circuit of figure 14.

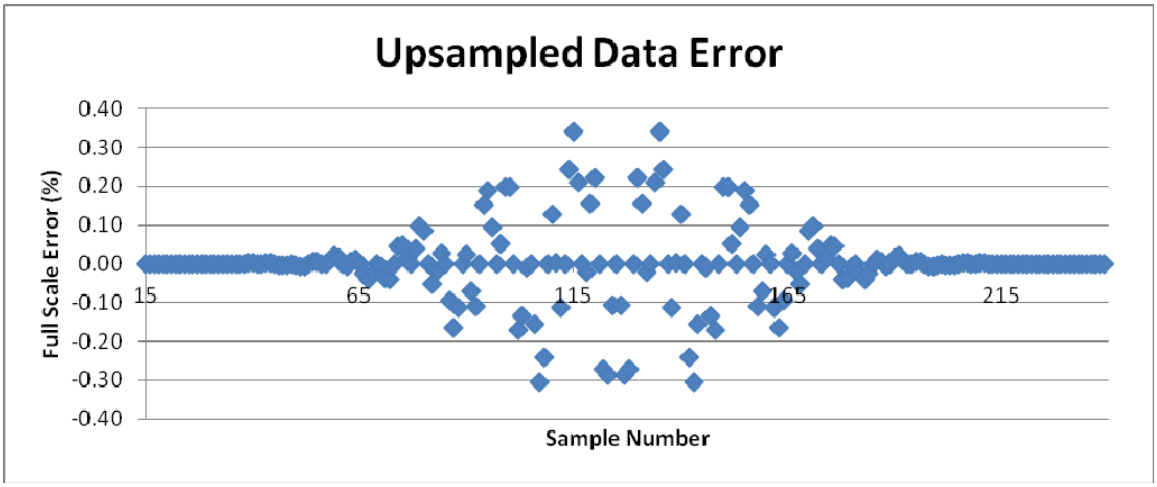

Fig. 16 Upsampled data error versus sample number plotted as a percent of the full scale input range when the data sequence shown in figure 2 is processed by the parallel upsampling circuit of figure 14.

The results shown in figures 15 and 16 were generated using double-precision floating point representations of the input sample sequence and the FIR filter coefficients and,

therefore, represent the best-case performance of a filter of this type. Using the relationship between percent full scale error and effective number of bits (ENOB),[8]

$$\text{ERROR} = (1/2^{\text{ENOB}}) \times 100\ \%, \quad (5)$$

we find that the 31-tap (M=4) filter design produces 8.2 effective bits, while a 63-tap filter design produces 12.2 effective bits.

The circuit shown in figure 14 was implemented in a Xilinx XC6VLX75T-3FF484 Virtex 6 FPGA[9] using the free Xilinx WebPack tools.[10] This design assumed 12-bit sample data as might be produced by an Analog Devices AD9272[12] eight-channel ultrasound front end integrated circuit. Filter coefficients were represented as 25-bit fixed point constants to match the size of the multipliers integrated onto the FPGA die. Input samples from the ADC were clocked into a register (R1 in figure 14) connected to input pins, and upsampled output values used registers tied to output pins. Registers R2-R8 were internal to the chip. Registers R1-R8 are intentionally 15 bits wide so that the synthesized logic has the headroom to perform the calculation, and the design does check for overflow or underflow and clamp the results so they remain within the valid range. The final circuit was able to run at 98 MHz, faster than the highest sample rate (80 MHz) available with the AD9272, and produce four upsampled data values per clock cycle. Pipelining the filter was not required to achieve this performance.

Even though the XC6VLX75T-3FF484 is the smallest member of the Xilinx Virtex 6 family, it contains 288 25x18-bit multipliers integrated onto the die, or enough to theoretically implement 16 parallel upsampling FIR filters of the type shown in figure 14. The largest Xilinx Virtex 6 FPGA, the XC6VSX475T, contains 2016 25x18-bit multipliers, allowing it to theoretically implement 112 upsampling filters of the type shown in figure 14 in a single chip.

V. CONCLUSIONS

It is possible to upsample an ultrasound data vector by a factor of M in real time using an FIR filter implemented in a commodity FPGA running at the base analog-to-digital converter clock rate when the filter is designed using the efficient, parallel topology presented here. The original data samples pass through the filter unmodified, and (M-1) upsampled values are produced in parallel. This approach will let system designers to push transducer frequencies higher while still allowing the use of integrated ultrasound front ends with limited sample rates. The straight-forward FIR filter design methodology presented eliminates the need for sophisticated filter design tools while providing excellent results.

ACKNOWLEDGMENTS

This work was supported by the Washington University in St. Louis Department of Computer Science and Engineering.

The author wishes to thank Edward Richter and Robert E. Morely for their many comments on the drafts of this paper.

REFERENCES

1. Brunner E. How Ultrasound System Considerations Influence Front-End Component Choice, *Analog Dialogue 36*, 9-12 (2002).

2. Shannon CE. Communication in the presence of noise, *Proc. Institute of Radio Engineers 37*, 10-21 (1949).

3. Oppenheim AV, Schafer RW. *Discrete-Time Signal Processing* (Prentice Hall, Englewood Cliffs, NJ, 1989).

4. Stark H, Woods JW, Paul I. An investigation of computerized tomography by direct Fourier inversion and optimum interpolation, *IEEE Trans. Biomed. Engr. 28*, 496-505 (1981).

5. Schafer RW, Rabiner LR. A digital signal processing approach to interpolation, Proc. of the IEEE *61*, 692-702 (1973).

6. Crochiere R, Rabiner LR. *Multirate Digital Signal Processing*, (Prentice-Hall, Englewood Cliffs, NJ, 1983).

7. Lasoda, RA. *Practical FIR Filter Design in MATLAB* (The MathWorks, Inc., Natick, M.A., 2003).

8. Kester, W. Understand SINAD, ENOB, SNR, THD, THD+N, and SFDR So You Don’t Get Lost in the Noise Floor, Tutorial, Analog Devices, 2008.

8. *Virtex-6 Family Overview DS150 (v2.3)* (Xilinx, Inc., San Jose, CA, 2011).

10. *ISE In-Depth Tutorial UG695 (v13.1)* (Xilinx, Inc., San Jose, CA, 2011).

11. Pellerin D, Taylor D. *VHDL Made Easy!* (Prentice-Hall, Upper Saddle River, NJ, 1997).

12. *Analog Devices AD9272 Octal LNA/VGA/AAF/ADC and Crosspoint Switch Datasheet Rev C* (Analog Devices, Norwood, MA, 2009).

FIGURE CAPTIONS

Fig. 1 Example 16 MHz signal used to illustrate the upsampling process.

Fig. 2. Example sample data sequence that results from sampling the example analog signal of figure 1 at 80 MHz, or five times per period.

Fig. 3 Zero-padded data sequence produced by inserting M=3 zeros between each actual sample value in the original data sequence of figure 2.

Fig. 4. Magnitude of the DFT of the non-zero-padded example data sequence.

Fig. 5. Magnitude of the DFT of the zero-padded data sequence.

Fig. 6. A 31-tap FIR filter that could be used to generate one upsampled data value per clock period if clocked at M-times the base ADC clock rate.

Fig. 7. Sinc function sampled for use with a 31-tap filter when M=4.

Fig. 8. Frequency response of a 31-tap FIR filter based on a sampled Sinc function.

Fig. 9. Hanning window appropriate for a 31-tap FIR filter.

Fig. 10. Windowed Sinc function designed for M=4.

Fig. 11 Frequency response of the 31-tap FIR filter using windowed coefficients.

Fig. 12 Frequency response of the 63-tap FIR filter using windowed coefficients.

Fig. 13 By observing that only 1 out of every 4 registers in figure 6 holds non-zero data during any given clock period, it is possible to collapse the filter and produce 4 outputs in parallel while running the filter at the base ADC clock rate.

Fig. 14 The final example 31-tap FIR filter design uses eight data registers, seven adders, 18 multipliers, and four output registers.

Fig. 15. The upsampled data sequence produced by the circuit of figure 14.

Fig. 16 Upsampled data error versus sample number plotted as a percent of the full scale input range when the data sequence shown in figure 2 is processed by the parallel upsampling circuit of figure 14.