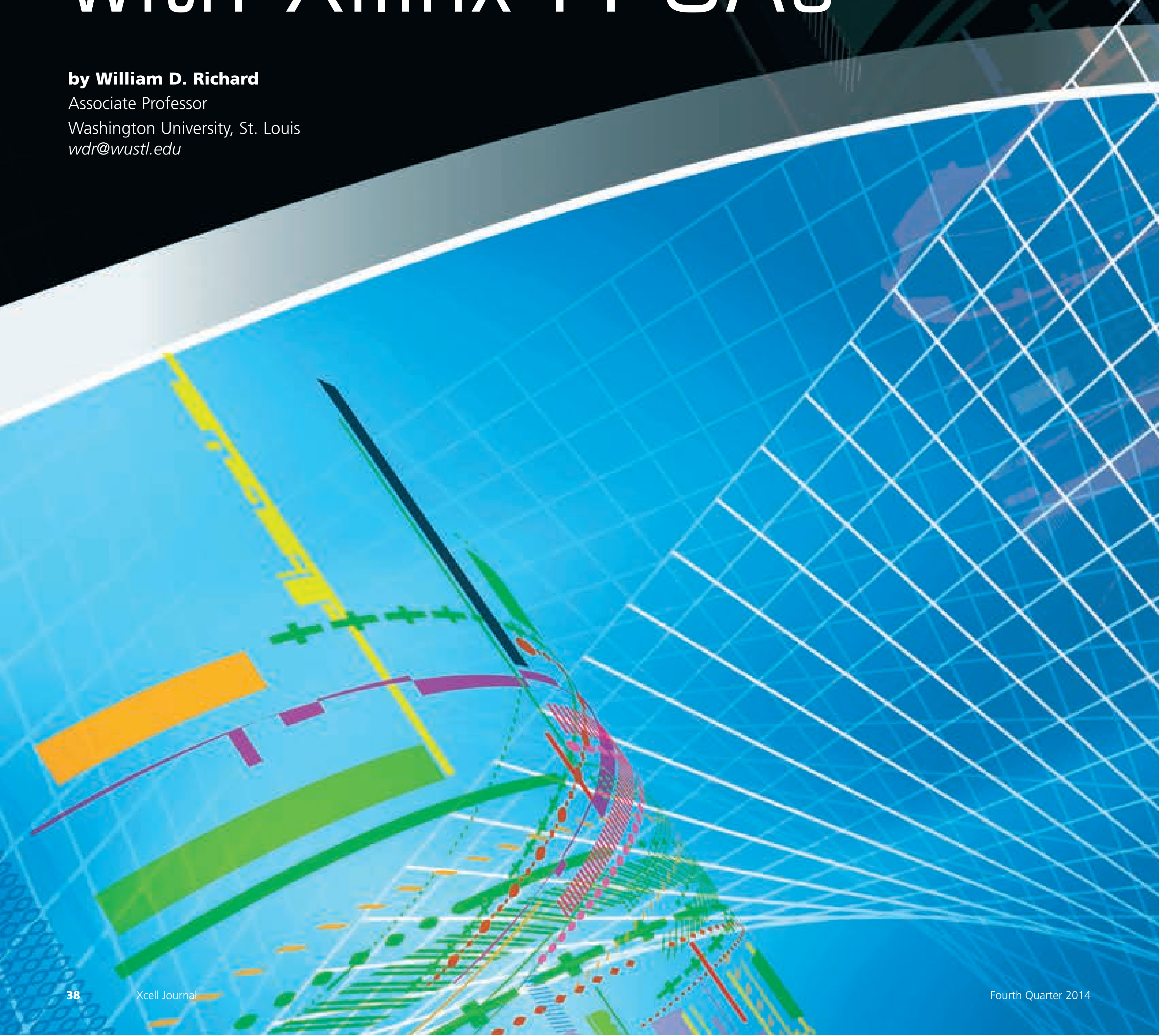
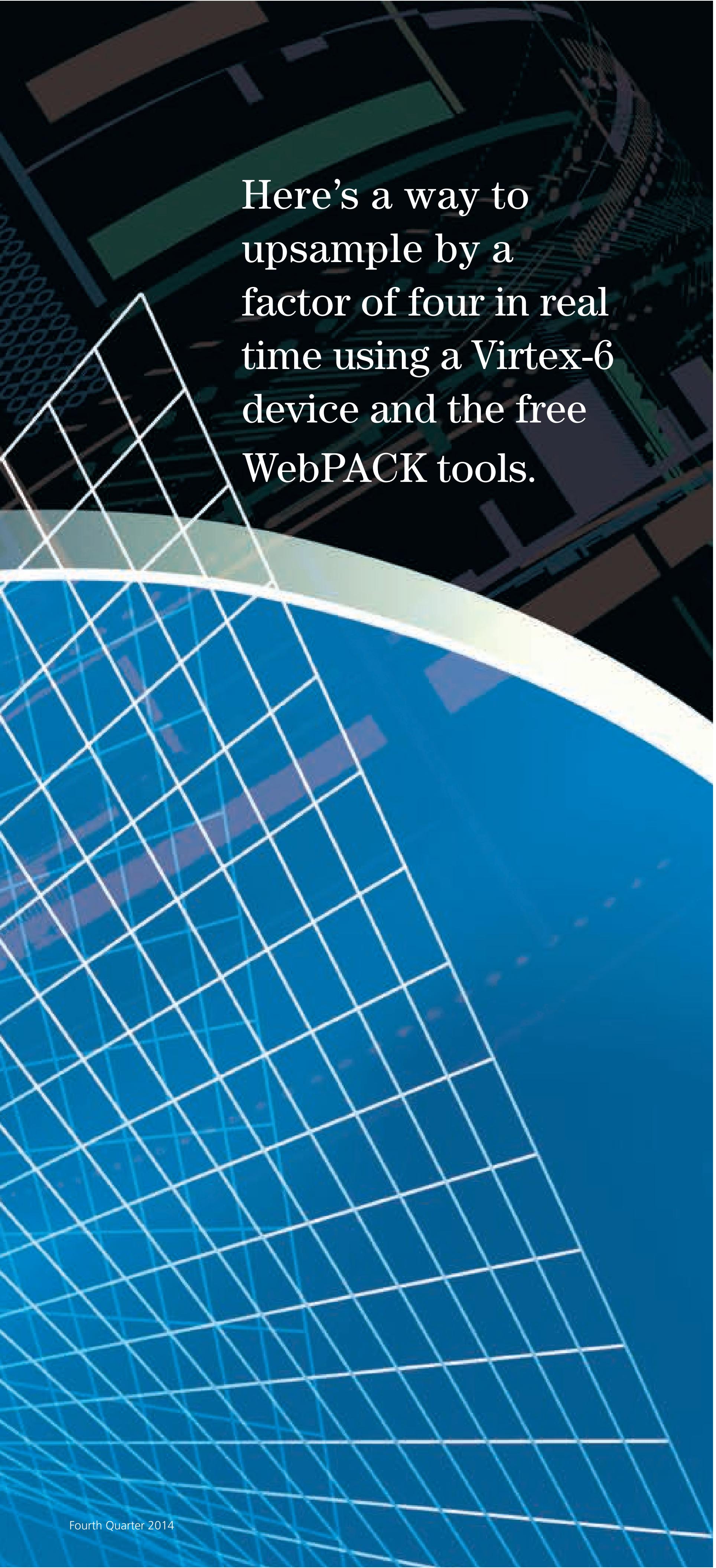
Efficient Parallel Real-Time Upsampling with Xilinx FPGAs

by William D. Richard
Associate Professor
Washington University, St. Louis
wdr@wustl.edu





Here's a way to upsample by a factor of four in real time using a Virtex-6 device and the free WebPACK tools.

Upsampling is required in many signal-processing applications. The easiest way, conceptually, to upsample a vector of data by a factor of M is to zero-pad the discrete Fourier transform (DFT) [1] of the data 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 [1, 2]. This approach is computationally expensive, however, and does not lend itself to efficient implementation inside FPGAs. The efficient, parallel, real-time upsampling circuit presented here produces M upsampled values per ADC clock, where M is the desired upsampling factor. Our Xilinx® Virtex®-6 XC6VLX75T FPGA implementation, which upsamples by a factor of M=4, serves as an example of the more general technique.

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 [3, 4].

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

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

Equation 1

If the signal shown in Figure 1 is sampled/quantized at 80 MHz using a 12-bit ADC driven to 97.7 percent of its full-scale input range, only five samples are taken per signal period, resulting in the sample data sequence shown in Figure 2. Upsampling this example data sequence by a factor of four, to an effective sample rate of 320 MHz, would provide 20 samples per signal period. While you can use the method described here to upsample by larger factors, we will upsample by M=4 for purposes of illustration.

Of course, it's possible to generate an (admittedly poorly) upsampled data vector with the desired number of samples by simply inserting (M-1) zeros in between each actual sample value in the data sequence the ADC produces. This "zero insertion step" corresponds to a replication of the spectrum of the original signal in the frequency domain. By low-pass filtering the resulting "zero-padded" time-domain signal so as to eliminate the "replications" of the desired spectrum in the frequency domain, you can obtain an upsampled data vector.

FIR FILTER DESIGN

An ideal (brick wall) low-pass filter in the frequency domain corresponds to convolution in the time domain with an infinite-extent Sinc function. Therefore, let's run our zero-padded time-domain signal through a symmetric, low-pass FIR filter running at M times the ADC clock rate and performing an approximation to the desired convolution operation, using the topology shown in Figure 3 for an example 31-tap FIR filter. In this way, we can produce our upsampled data vector in real time. In Figure 3, R1, R2, …, R31 represent registers clocked at M times the ADC clock rate, and C0, C1, …, C15 represent the coefficients of the FIR filter.

It is important to note that most of the registers in the FIR filter shown in Figure 3 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.

Since (M–1) out of every M samples moving through the FIR filter shown in Figure 3 are zero, you can collapse the filter and produce M outputs in parallel as shown in Figure 4 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.

You can specify the windowed Sinc function coefficients, Cw(n), shown in Figure 4 so as to minimize the number of

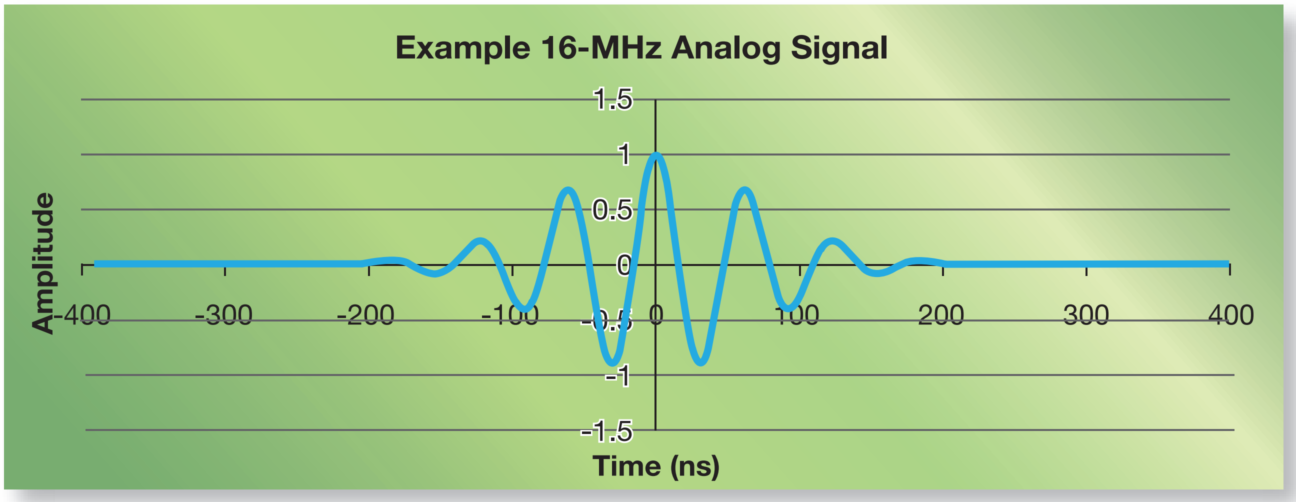


Figure 1 – This example 16-MHz signal illustrates the upsampling process.

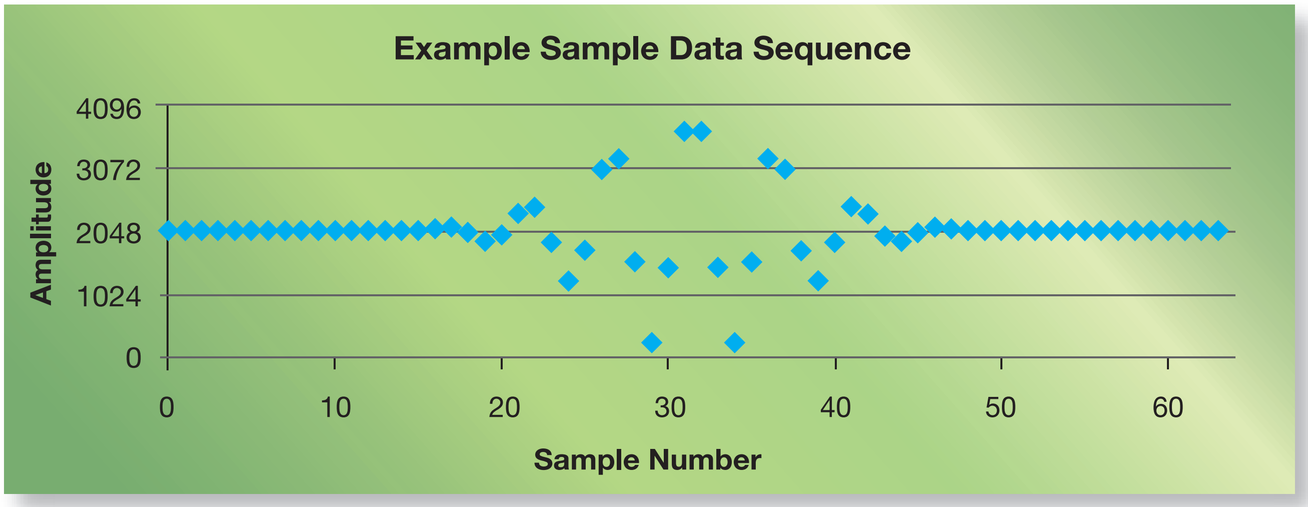


Figure 2 – Here is the example sample data sequence that results from sampling the example analog signal of Figure 1 at 80 MHz, or five times per period, using a 12-bit ADC driven to 97.7 percent of its full-scale input range.

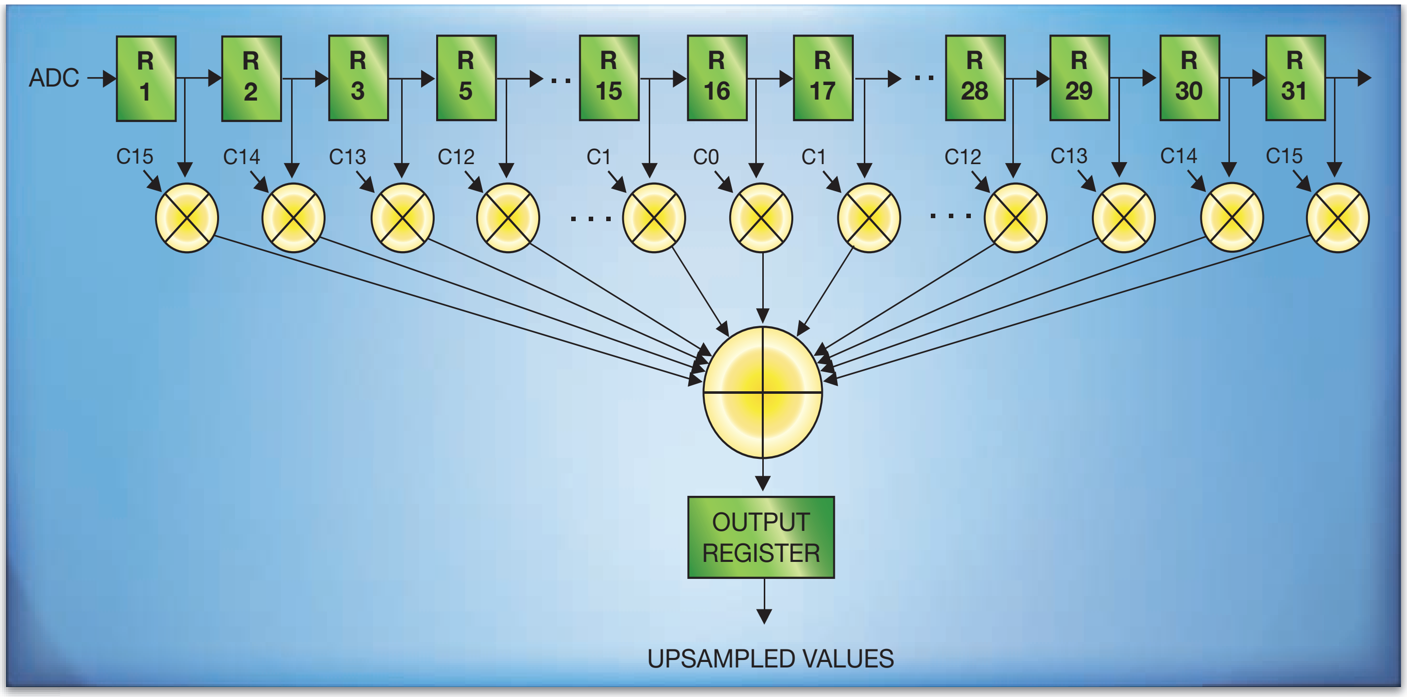


Figure 3 – You could use a 31-tap FIR filter to generate one upsampled data value per clock period if clocked at M-times the base ADC clock rate with zero insertion.

multipliers required to implement the FIR filter. For a T-tap, low-pass FIR filter, the optimal coefficients are given by:

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

Equation 2

Here, the Hanning window coefficients are given by:

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

Equation 3

The windowed Sinc function coefficients, Cw(n), are then found by multiplying corresponding values of C(n) and H(n), i.e.,

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

Equation 4

For M=4, when the coefficients for a 31-tap FIR filter are calculated as described above, C0 = 1.0 and C4 = C8 = C12 = C15 = 0, the nine multipliers associated with these coefficients in Figure 4 are not needed. In addition, by recognizing that each coefficient is used twice to generate UPSAMPLED VALUE(1), you can “fold” the implementation—add R1 to R8 before multiplying, for example—and eliminate four additional multipliers. The end result is a design that requires a total of only 18 multipliers to produce four upsampled values per clock period. It is important to note that, as a result of the filter design technique described above, each original sample value exits the parallel filter unmodified.

We used the synthesizable VHDL [5] model in Figure 5 to evaluate the performance of the circuit shown in Figure 4. This VHDL implementation assumes 12-bit sample data, as an Analog Devices AD9670 eight-channel ultrasound front-end integrated circuit [6] might produce. Filter coefficients are represented as 25-bit fixed-point constants to match the size of the multipliers integrated onto the FPGA die. Input samples from the ADC are clocked into a register (R1 in Figure 4) connected to input pins, and upsampled output values use registers tied to output pins. Registers R2 to R8 are internal to the chip. Registers R1 to R8 are intentionally 15 bits wide so that the synthesized logic has the headroom to perform the calculation. The design checks for overflow or underflow and clamps the results so they remain within the valid range.

NO NEED FOR PIPELINING

Figure 6 plots the upsampled data sequence that results when the VHDL model is simulated using the ISim simulator in version 14.7 of the free Xilinx WebPACK™ tools [8] and fed the sampled/quantized 12-bit data sequence of Figure 2. Each of the original 12-bit samples is unchanged, as explained above, and three new samples have been inserted on the original waveform between each actual sample.

The worst-case error in the computed (upsampled) values from the ideal values in the original analog signal is 0.464 percent of the full-scale range, while the average error is 0.070

The placed-and-routed design used 19 DSP48E1 blocks but fewer than 1 percent of the Virtex-6's slices. It ran at 107 MHz without pipelining.

percent of the full-scale range. Of course, there is as much as ½ LSB of error in the sampled/quantized 12-bit source vector data values (or 0.012 percent of the full-scale range) due to the initial quantization step.

We implemented the upsampler in a Xilinx XC6VLX75T-3FF484 Virtex-6 FPGA [7] using version 14.7 WebPACK tools. The placed-and-routed design used 19 of the 288 DSP48E1 blocks in the part but fewer than 1 percent of the slices. The final upsampling circuit was able to run at 107 MHz. It was not necessary to pipeline the filter to achieve this performance. We also developed a pipelined version that ran at over 217 MHz.

Even though the XC6VLX75T-3FF484 is the smallest member of the Xilinx Virtex-6 family, it contains 288 DSP48E1

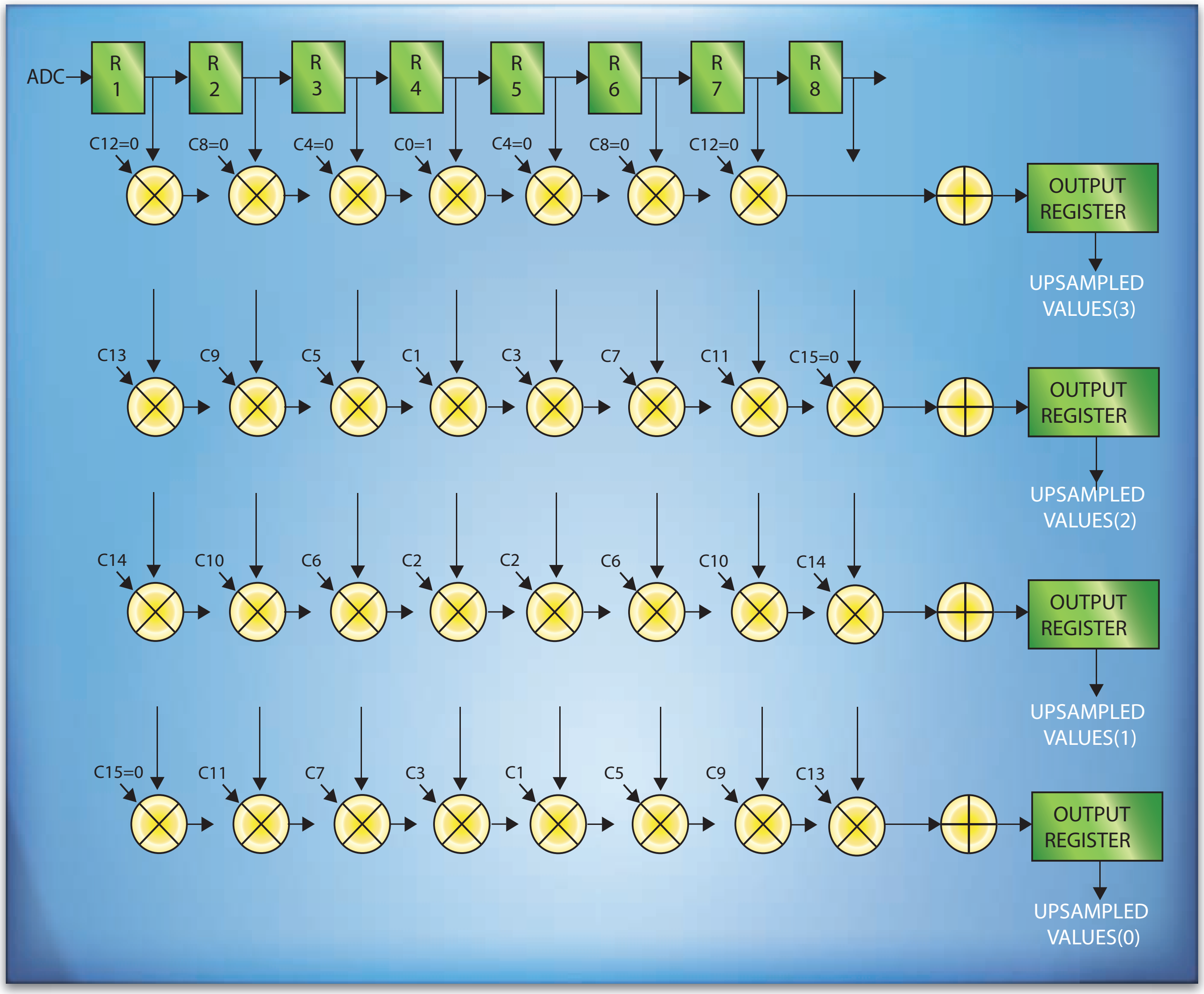


Figure 4 – By observing that only one out of every four registers in Figure 3 holds non-zero data during any given clock period, it is possible to collapse the filter and produce four outputs in parallel while running the filter at the base ADC clock rate.

```
LIBRARY IEEE ;
USE IEEE.STD_LOGIC_1164.ALL ;
USE IEEE.STD_LOGIC_ARITH.ALL ;
USE IEEE.STD_LOGIC_UNSIGNED.ALL ;

ENTITY upsample IS
   PORT (clk         : IN STD_LOGIC ;
         r_ext       : IN STD_LOGIC_VECTOR(11 DOWNTO 0) ;
         d0,d1,d2,d3 : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)) ;
   END upsample ;

ARCHITECTURE mine OF upsample IS
   SIGNAL   r1,r2,r3,r4,r5,r6,r7,r8             : STD_LOGIC_VECTOR(14 DOWNTO 0) ;
   SIGNAL   d0int,d1int,d2int                   : STD_LOGIC_VECTOR(39 DOWNTO 0) ;
   CONSTANT c1  : STD_LOGIC_VECTOR(24 DOWNTO 0) := “1110001111110110011100110” ;
   CONSTANT c2  : STD_LOGIC_VECTOR(24 DOWNTO 0) := “1001101111101110000000011” ;
   CONSTANT c3  : STD_LOGIC_VECTOR(24 DOWNTO 0) := “0100010101111101100111001” ;
   CONSTANT c5  : STD_LOGIC_VECTOR(24 DOWNTO 0) := “0010001010010010011110000” ;
   CONSTANT c6  : STD_LOGIC_VECTOR(24 DOWNTO 0) := “0010001110001110010111001” ;
   CONSTANT c7  : STD_LOGIC_VECTOR(24 DOWNTO 0) := “0001001000101111000010111” ;
   CONSTANT c9  : STD_LOGIC_VECTOR(24 DOWNTO 0) := “0000100011011001000000101” ;
   CONSTANT c10 : STD_LOGIC_VECTOR(24 DOWNTO 0) := “0000100000100110000100111” ;
   CONSTANT c11 : STD_LOGIC_VECTOR(24 DOWNTO 0) := “0000001101110111011000010” ;
   CONSTANT c13 : STD_LOGIC_VECTOR(24 DOWNTO 0) := “0000000011000100001100100” ;
   CONSTANT c14 : STD_LOGIC_VECTOR(24 DOWNTO 0) := “0000000001000001000111111” ;

BEGIN

   flops:PROCESS(clk)
   BEGIN
      IF (clk = ‘1’ AND clk’EVENT) THEN
         r1 <= “000” & r_ext ;
         r2 <= r1 ;
         r3 <= r2 ;
         r4 <= r3 ;
         r5 <= r4 ;
         r6 <= r5 ;
         r7 <= r6 ;
         r8 <= r7 ;
         IF d0int(39) = ‘1’ THEN
            d0 <= “000000000000” ;
         ELSIF d0int(38) = ‘1’ OR d0int(37) = ‘1’ THEN
            d0 <= “111111111111” ;
         ELSE
            d0 <= d0int(36 DOWNTO 25) ;
         END IF ;
         IF d1int(39) = ‘1’ THEN
            d1 <= “000000000000” ;
         ELSIF d1int(38) = ‘1’ OR d1int(37) = ‘1’ THEN
            d1 <= “111111111111” ;
         ELSE
            d1 <= d1int(36 DOWNTO 25) ;
         END IF ;
         IF d2int(39) = ‘1’ THEN
            d2 <= “000000000000” ;
         ELSIF d2int(38) = ‘1’ OR d2int(37) = ‘1’ THEN
            d2 <= “111111111111” ;
         ELSE
            d2 <= d2int(36 DOWNTO 25) ;
         END IF ;
         d3 <= r4(11 DOWNTO 0) ;
      END IF ;
   END PROCESS ;

   d0int <= r2*c11 - r3*c7 + r4*c3 + r5*c1 - r6*c5 + r7*c9 - r8*c13 ;
   d1int <= (r2+r7)*c10 - (r1+r8)*c14 - (r3+r6)*c6 + (r4+r5)*c2 ;
   d2int <= r2*c9 - r1*c13 - r3*c5 + r4*c1 + r5*c3 - r6*c7 + r7*c11 ;

END mine ;
```

Figure 5 – The VHDL source uses a single process and 25-bit fixed-point coefficients to implement the filter topology of Figure 4.

This straightforward FIR filter design methodology eliminates the need for sophisticated filter design tools while providing excellent results.

blocks with 25x18-bit multipliers integrated onto the die, or enough to theoretically implement 15 parallel upsampling FIR filters of the type shown in Figure 4. We have constructed a prototype annular-array ultrasound system that uses eight copies of the upsampler running at 80 MHz in an XC6VLX75T FPGA to upsample data from an eight-channel Analog Devices AD9670 ultrasound front-end chip prior to beam forming. In this system, the upsampler works as predicted by simulation and enables real-time beam forming using data upsampled to 320 MHz while running at the base AD9670 ADC clock rate of 80 MHz.

The largest Xilinx Virtex-6 FPGA, the XC6VSX475T, contains 2,016 25x18-bit multipliers, allowing it to theoretically implement 106 upsampling filters of the type shown in Figure 4 in a single chip.

It is possible to upsample by a factor of M=4 in real time using an FIR filter implemented in a Xilinx XC6VLX75T-3FF484 FPGA that runs at 107 MHz when the filter is designed using the efficient, parallel topology presented here. The original data samples pass through the filter unmodified, and (M-1) = 3 upsampled values are produced in parallel. This straightforward FIR filter design methodology eliminates the need for sophisticated filter design tools while providing excellent results. Straightforward extension of the ideas presented here could be used to upsample by larger factors or to reduce the error in the computed upsampled values by using an FIR filter with more taps.

REFERENCES

1. A.V. Oppenheim, R.W. Schafer, *Discrete-Time Signal Processing* (Prentice Hall, Englewood Cliffs, NJ, 1989)
2. H. Stark, J.W. Woods, I. Paul, “An investigation of computerized tomography by direct Fourier inversion and optimum interpolation,” IEEE Transactions Biomedical Engineering 28, 496-505 (1981)
3. R.W. Schafer, L.R. Rabiner, “A digital signal processing approach to interpolation,” Proceedings of the IEEE 61, 692-702 (1973)
4. R. Crochiere, L.R. Rabiner, *Multirate Digital Signal Processing*, (Prentice-Hall, Englewood Cliffs, NJ, 1983)
5. D. Pellerin, D. Taylor, *VHDL Made Easy!* (Prentice-Hall, Upper Saddle River, NJ, 1997)
6. Analog Devices AD9670 Octal Ultrasound AFE with Digital Demodulator Datasheet Rev Sp0 (Analog Devices, 2013)
7. Virtex-6 Family Overview DS150 (v2.3) (Xilinx, Inc., 2011)
8. ISE In-Depth Tutorial UG695 (v13.1) (Xilinx, Inc., 2011)

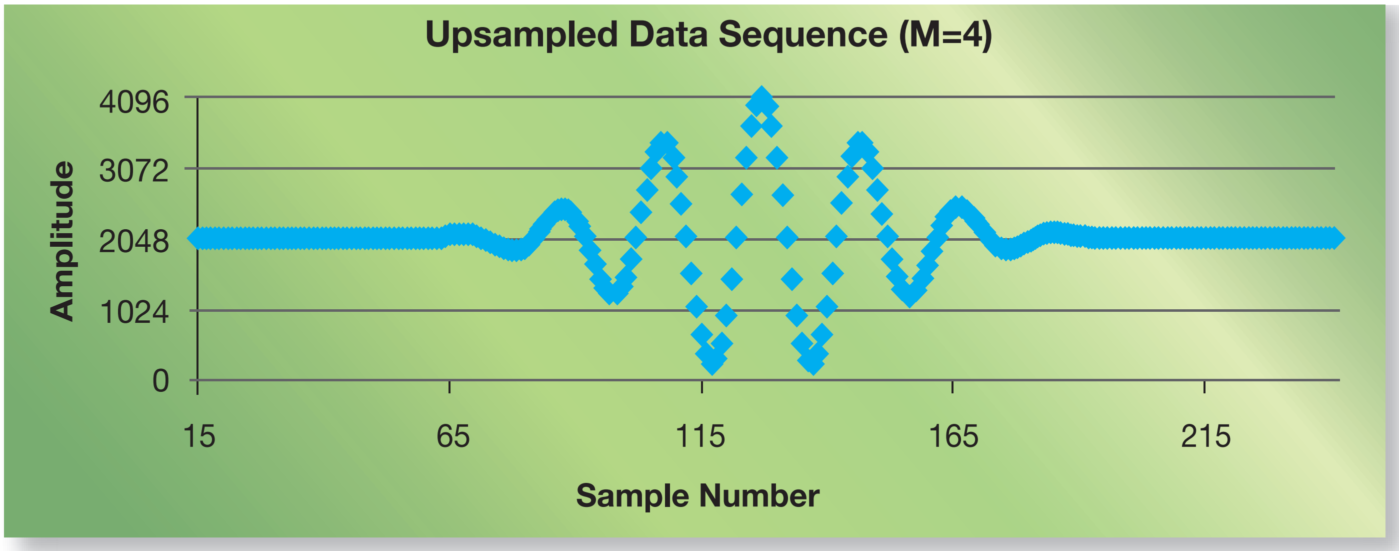


Figure 6 – This graph shows the upsampled data sequence produced by the VHDL model.