Proc. of SPIE's Photonics East'98 *Configurable Computing: Technology and Applications*, Boston, MA, USA, pp, 227-238, November 1-6 1998.

Implementing Area Optimized Narrow-Band FIR Filters Using Xilinx FPGAs

Chris Dick
chrisd@xilinx.com

Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124, USA

ABSTRACT

This paper describes the implementation of very high-performance narrow-band finite impulse response (FIR) filters using Xilinx[1] field programmable gate arrays (FPGAs). Multirate as well as single rate architectures are considered. By exploiting the narrow-band nature of the filtering process, a very area-efficient FPGA filter mechanization referred to as an *interpolated FIR (IFIR) filter* is developed. Several design examples are presented. A lowpass single rate IFIR consumes only 48% of the logic requirements in comparison to other area efficient FPGA FIR structures that have been reported in the literature. Multirate implementations provide larger logic savings. A 50-to-1 decimator is presented that uses only 13.7% of the logic resources of a standard FPGA polyphase implementation.

Keywords: interpolated FIR, FPGA, DSP, multirate filter

1 INTRODUCTION

Many systems require very high-performance narrow-band linear phase filters. Narrow-band wireless communication systems can use these filters as part of the channelization process. Decimating filters also find application in transmultiplexers where they are used to convert between time-division multiplexed (TDM) and frequency-division multiplexed (FDM) data formats. Multirate filter architectures can also be used to implement narrow-band filtering, with no sample rate change from input to output, by first performing a sample rate reduction followed by a sample rate increase.

This paper addresses the efficient field programmable gate array (FPGA) implementation of linear phase narrow-band single-rate and multirate filters. Rather than employ a direct finite impulse response (FIR) implementation, the *interpolated FIR (IFIR)* approach of Neuvo *et. al.*[3] in conjunction with FPGA technology is shown to be a very powerful combination by producing highly efficient FPGA filter realizations. For single-rate filters, use of the IFIR approach with FPGA hardware produces filters that occupy less than 50% of the logic resources of other FPGA approaches that have been reported in the open literature. A high-performance decimating filter is presented that uses only 13.3% of the logic requirements of a standard polyphase[2] filter FPGA implementation.

First the IFIR filter approach is briefly reviewed. A narrow-band lowpass filter design is then presented.

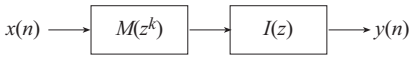

Figure 1: Interpolated FIR (IFIR) architecture.

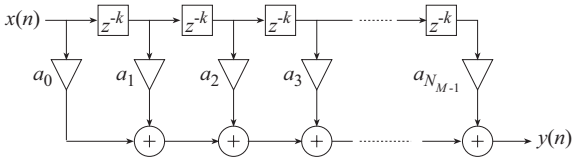

Figure 2: Upsampled FIR architecture $M(z^k)$.

The Xilinx[1] FPGA implementation of the filter is described and the logic resources of this filter are compared to a conventional FPGA implementation that provides the same functionality. Next, a general framework for decimating IFIR structures is presented. This approach is demonstrated by a 50-to-1 decimator design example that rate converts a 100 MHz signal down to 2 MHz. Traditionally this type of filter would be implemented using a polyphase decomposition. The polyphase version of the decimator and the decimating IFIR are compared to illustrate the dramatic reduction in logic resources that is afforded by the IFIR approach.

2 INTERPOLATED FIR - *IFIR*

Linear phase filters are conventionally implemented using a standard tapped delay-line or transversal filter structure. However, many other realization options present themselves if the filter designer is prepared to take advantage of the certain features that may be characteristics of the filter requirements themselves, or that become available when the filter is taken in context with adjacent processing blocks. Filters whose bandwidth is considerably smaller than the system sample rate, or *narrow-band filters*, occur in many practical applications. Several options other than the standard FIR realization can be used to produce efficient narrow-band filter implementations.

The cascade filter structure shown in Figure 1 is a two-stage *interpolated FIR (IFIR)* filter.

The sample rate for both stages is the same as that at the input and output. So although the name alludes to multirate techniques, strictly speaking this is not a multirate filter, since the sampling frequency at all stages is the same. With IFIRs the multirate aspect of the system is embodied in the filter impulse response.

The transfer function $M(z^k)$ of the first stage is a function of z^k. It can be implemented by replacing the unit delay operator z^{-1} in a transversal filter $M(z)$ by z^k, as shown in Figure 2. This means simply replacing each delay element by k units of delay. This is equivalent to inserting $k-1$ zeros between the original impulse response values of $M(z)$. Now recall that a k-fold rate expansion of a time series causes $k-1$ spectral replicates to fold back into the frequency interval $[0 \mapsto 2\pi]$. The same observation is obviously true for $M(z^k)$, since a filter impulse response is just a time-series. The situation is shown in the frequency domain plot of Figure 3 for a rate expansion $k = 4$. In going from $M(e^{j\Omega})$ to $M(e^{jk\Omega})$ the frequency domain response is compressed by the factor k. The critical frequencies in the baseband spectrum of $M(e^{jk\Omega})$ are k times narrower than those of $M(e^{j\Omega})$.

So far we have seen that rate expanding a filter's impulse response has the desired effect of decreasing the baseband filter transition-band without introducing any additional arithmetic. The narrow-band response comes at only the expense of some additional memory to accommodate the increased storage requirements for the input signal time history. However, the spectral replicates or *images* are a problem and must be dealt with. This is the function of the *image rejection filter* $I(z)$ in Figures 2 and 3(c). The resultant transfer function is

$$H(z) = M(z^k)I(z) \tag{1}$$

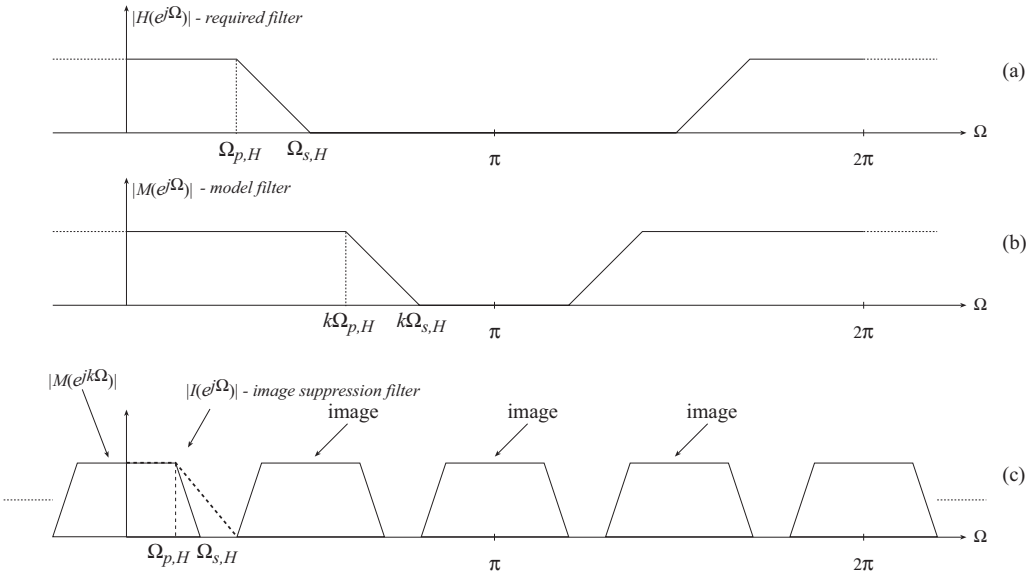

Figure 3: IFIR - frequency domain illustration.

The compressed response reduces the computational complexity by a factor of approximately k. The complexity for constructing $I(z)$ must also be considered. In many situations a suitable choice of k will result in a shallow transition slope for the image suppressor, so that this filter will have substantially fewer filter coefficients than the desired filter $H(z)$ and only contribute modestly to the computation load.

2.1 IFIR Design

The IFIR approach is suitable for any narrow-band filter process - lowpass, bandpass or highpass. The design approach will be described using a lowpass methodology.

Consider the synthesis of a lowpass filter $H(z)$ with a passband edge frequency $\Omega_{p,H}$, stopband edge frequency $\Omega_{s,H}$ and with $\delta_{p,H}$ and $\delta_{s,H}$ passband and stopband ripple respectively.

First a model filter $M(z)$ (Figure 3(b)) is defined having a passband edge frequency $\Omega_{p,M} = k\Omega_{p,H}$, and stopband edge frequency $\Omega_{s,M} = k\Omega_{s,H}$. The passband edge frequency of $I(z)$ must be the same as the required filter $H(z)$, so $\Omega_{p,I} = \Omega_{p,H}$. The stopband edge frequency for the image suppression filter must be chosen so that its magnitude response provides the specified attenuation at the first intercept point with the first image of $M(z^k)$ above baseband. Some simple arithmetic results in this design frequency being defined as $\Omega_{s,I} = 2\pi/k - \Omega_{s,H}$.

The passband ripple for $H(z)$ is distributed between $M(z^k)$ and $I(z)$. Formally we require

$$(1 + \delta_{p,M})((1 + \delta_{p,I})) = 1 + \delta_{p,H} \tag{2}$$

If the ripple $\delta_{p,H}$ is small, then

$$\delta_{p,M} + \delta_{p,I} \approx \delta_{p,H} \tag{3}$$

One possibility for solving Eq. (3) that works in practice is to choose $\delta_{p,M} = \delta_{p,I} = 0.5\delta_{p,H}$. Provided δ_p is small it is possible to make the further approximation

$$\delta_{s,M} = \delta_{s,I} \approx \delta_s \tag{4}$$

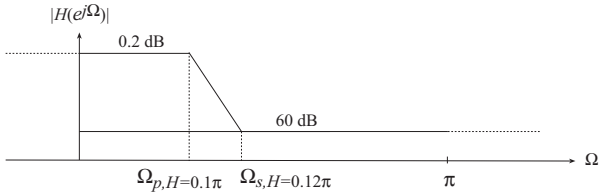

Figure 4: Narrow-Band lowpass filter example - magnitude response requirements.

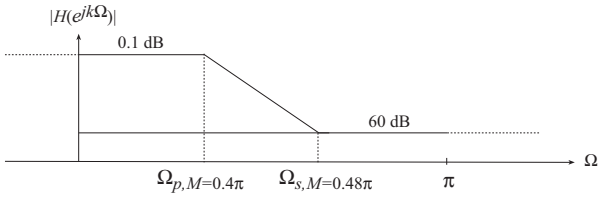

Figure 5: Model filter specifications for the lowpass IFIR example, $k = 4$.

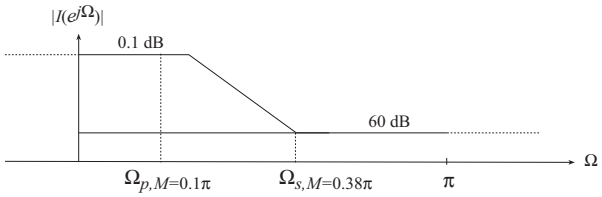

Figure 6: Image rejection filter specifications for a lowpass IFIR, $k = 4$.

2.2 Lowpass IFIR Example

As a first example consider implementing the lowpass filter characterized in Figure 4. Using a normalized sample rate of 2π radians, the passband edge frequency is $\Omega_{p,H} = 0.05 \times 2\pi$, the stopband edge frequency is $\Omega_{s,H} = 0.06 \times 2\pi$, the passband ripple is to be 0.2 dB and the required minimum stopband attenuation is 60 dB. Using filter design software the number of filter taps, N, required to meet the filter requirements is 260. An efficient technique for implementing inner-product calculations using distributed memory FPGA architectures like the Xilinx 4000 series, is *distributed arithmetic*.[4] The mechanization of this type of filter, as well as many other DSP related functions is facilitated by the *Core Generator*[5] design automation tool. The largest FIR module that can be produced automatically is an 80-tap filter. However, these modules are cascadable. The required 260-tap filter can be easily built from available modules by cascading three 80-tap filters and a 20-tap section. Exploiting optimizations based on the symmetry in the filter coefficient data, a single 80-tap filter employing 16-bit arithmetic requires 320 CLBs. A 20-tap filter using 16-bit precision occupies 95 CLBs. Cascading these modules to produce the required 260-tap filter gives a CLB count of 1055.

Now consider an IFIR realization of the same filter. The model filter passband and stopband specifications are $\Omega_{p,M} = k\Omega_{p,H}$ and $\Omega_{s,M} = k\Omega_{s,H}$. To support the required passband ripple requirement $\delta_{p,H}$ of the original filter $H(z)$, the model and image rejection filters do not simply inherit the same value. They must be designed using a smaller value for passband ripple, so that the series cascade combination provides the required performance. For this example the passband ripple requirement will be equally distributed between $M(z)$ and $I(z)$. So

$$\delta_{p,M} = \delta_{p,I} = \frac{1}{2}\left(10^{0.2/20} - 1\right) = 0.0116464 \tag{5}$$

Selecting the up-sampling factor as $k = 4$ results in the model filter specification defined in Figure 5.

The image rejection filter $I(z)$ design frequencies are $\Omega_{p,I} = \Omega_{p,H} = 0.1\pi$ and $\Omega_{s,I} = 2\pi/K - \Omega_{s,H} = 0.38\pi$. This filter is illustrated in Figure 6. To meet the model filter requirements a 69-tap filter is needed. Figure 7 shows the magnitude frequency response for $M(z)$. Upsampling $M(z)$ by a factor $k = 4$ produces the response in Figure 8. The image rejection filter specifications can be met with a 22-tap filter. The magnitude response for

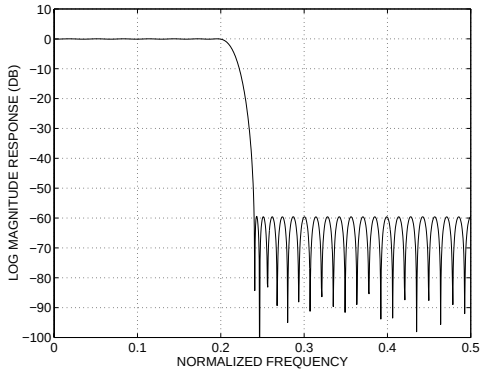

Figure 7: Model filter magnitude response.

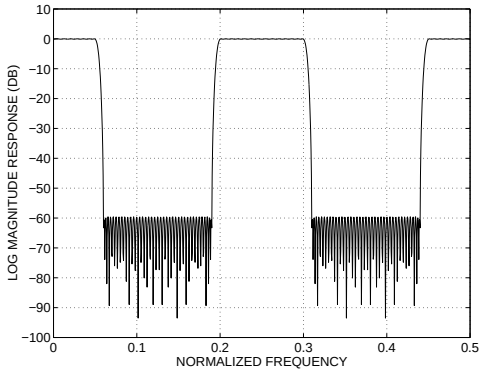

Figure 8: Upsampled model filter response. $k = 4$

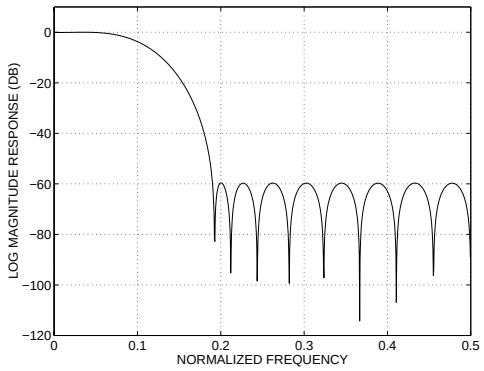

Figure 9: Image filter response.

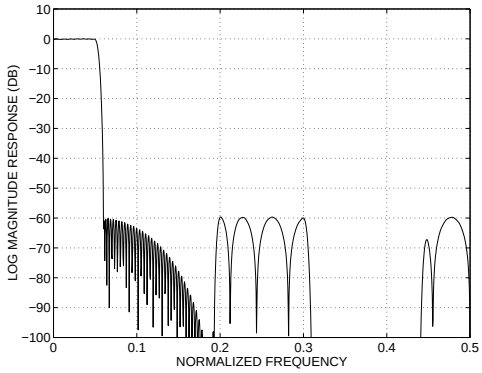

Figure 10: IFIR $M(z^4)I(z)$

$I(z)$ is shown in Figure 9.

The net frequency response of this combination is obtained by multiplying $M(z^4)$ with $I(z)$. This result is shown in Figure 10.

Exploded views of the passband and transition region are provided in Figures 11 and 12 as verification that the IFIR LPF design has satisfied the design requirements.

3 IFIR FPGA Implementation

The architecture of the upsampled filter $M(z^k)$ implemented in an FPGA is shown in Figure 13. Zero-packing the filter impulse response is of course achieved in practice by replacing the unit delay z^{-1} in the model filter with k units of delay. The filter supports a history of the previous $k(N_M - 1)$ input samples, but only N_M multiply-accumulates are required to compute an output value. Taking advantage of the highly efficient implementation of FIR filters in Xilinx FPGAs using distributed arithmetic results in the architecture shown in Figure 13. If the filter impulse is symmetrical the optimized architecture shown in Figure 14 can be used. This figure is for $N = 4$,

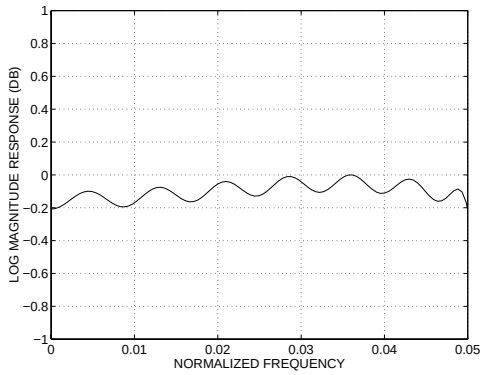

Figure 11: LPF IFIR passband.

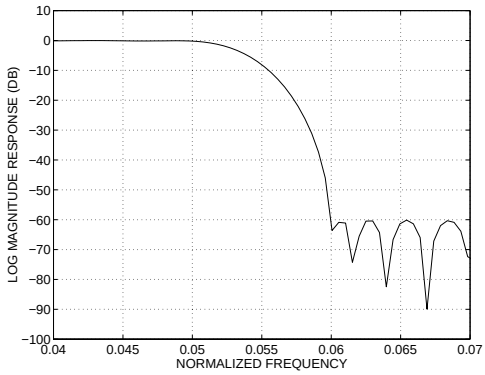

Figure 12: LPF IFIR transition band.

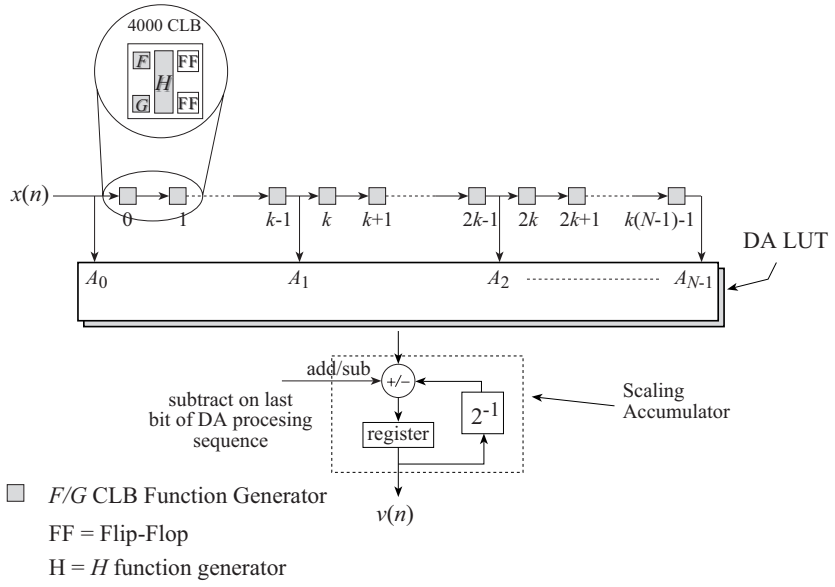

Figure 13: Xilinx FPGA IFIR implementation.

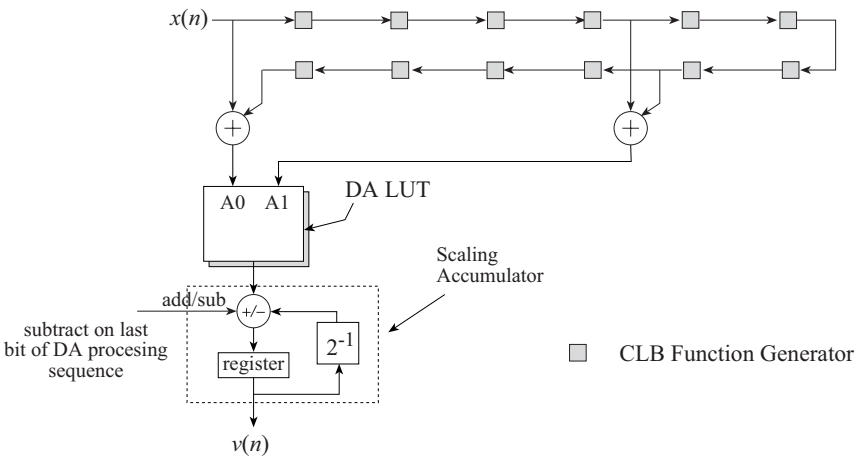

Figure 14: Xilinx FPGA IFIR implementation. $N = 4$, up-sampling factor $k = 4$, symmetric impulse response.

$k = 4$. Employing this architecture reduces the size of the DA lookup table.

The structure is recognized as being close in nature to a conventional DA FIR filter, but now every k'th input sample is used to form the appropriate address sequence for the DA lookup table (LUT).

3.1 Logic Requirements

The IFIR implementation requires an input sample delay-line or shift-register, and two FIR structures for $M(z)$ and $I(z)$. The delay-line must have taps at intervals of the up-sampling factor k. This structure supports the input sample precision B, and its length is the product of k and the model filter length $N_{M(z)}$. The shift-register is implemented with Xilinx technology using RAM-cell based shift-registers - function generators. Two shift-register stages of up to 16-bit precision can be supported in a single 4000 series configurable logic block (CLB). Excluding the minimal amount of control required for the input shift-register, the CLB count Λ for an IFIR filter is

$$\Lambda = \frac{kN_{M(z)}}{2} + \Xi(M(z)) + \Xi(I(z)) \tag{6}$$

where $\Xi(f)$ denotes the number of CLBs required to implement the functional unit f. Using the LPF example above, with $B = 16$, the input delay-line is $4 \times 69 = 276$ samples deep, and requires 136 CLBs to implement. The filter $M(z)$ is implemented with 270 CLBs and $I(z)$ requires 99 CLBs. The total CLB count is $\Lambda_1 = 509$. Comparing this figure to the 1010 CLB requirement for a direct implementation of $F(z)$ for the above example shows that the IFIR approach requires only $509/1055 \times 100 = 48\%$ of the logic resources. This is a considerable saving.

3.1.1 Performance - Computation Rate

The entire IFIR architecture will support a sample rate of 6 MHz. This equates to a computation throughput of 1.5 Giga-MACs (multiply-accumulates) per second. Re-phrasing this number, it is equivalent to 3 Mega-MACs per CLB - this is a large amount of filtering using only a modest amount of silicon real estate!

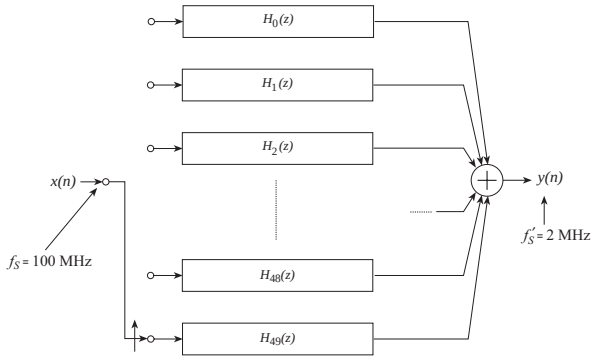

Figure 15: Polyphase 50-to-1 decimator.

4 Multistage IFIR Decimation and Interpolation Filters

In the previous example above there is no sample rate change in the system. If a decimation or interpolation filter is required, the IFIR approach in combination with a polyphase decomposition is useful for FPGA multirate filter implementations.

To illustrate this useful combination of multistage IFIR filters and FPGA implementation, consider the following example. A 100 MHz signal is to be decimated by a factor of $M = 50$. The filter requirements are a passband ripple $\delta_p = .01$ and the stopband ripple is to be $\delta_s = 0.001$. The passband and stopband edge frequencies are 850 kHz and 1 MHz respectively. An equiripple design requires a filter order $N = 2282$. To achieve the required output sample rate, an SDA FIR is the appropriate choice rather than time-shared MAC units based on parallel multipliers. Conventionally, this filter would be implemented using a multirate filter like the polyphase 50-to-1 decimator shown in Figure 15. In a microprocessor implementation, the inner-product calculations would be performed by appropriate time management of the filter coefficients. The M outputs would then be summed to form the final result. In an FPGA implementation a parallel approach is employed, where each polyphase segment is instantiated in the FPGA and all of the segments operate in parallel. An adder tree is used to sum the segment outputs. The number of filter taps per segment is $P = \lceil N/M \rceil = \lceil 2282/50 \rceil = 46$. Using the *Core Generator* design automation tool, the 46-tap filter is most easily implemented using a cascade of a 40-tap and 6-tap section. Using 16-bit precision for the input samples and the filter coefficients, the 40-tap module requires 302 CLBs. The 6-tap filter is implemented with 58 CLBs. In both filters the full precision of the calculation is carried internally and is truncated to 16-bits at the output.

Excluding the adder-tree, the number of CLBs required for the filter segments is $K_{\text{poly}} = 308 \times 50 = 15,400$! This is obviously an excessive requirement for current generation FPGAs. In addition to area considerations, the system clock rate required by the design needs to be examined. While data is delivered to the filter at the input sample rate $f_s = 100$ MHz, the polyphase filter segments are only required to operate at the output sample rate $f_s/M = f'_s = 2$ MHz. This is easily achievable. The same is true for the elements in the adder-tree - each registered adder is clocked at a modest 2 MHz. However, data must be delivered in a round-robyn fashion to each of the filter segments. This circuit must operate at the full input rate of 100 MHz. This clock frequency can be accommodated using Xilinx high-performance XL family components. If the designer chooses not to support the full input sample rate within the FPGA, some simple de-multiplexing implemented externally to the FPGA can be employed so that two 50 MHz sample streams are used to present the data to the polyphase filter. One stream consists of the even numbered samples from the data source, while the second consists of the odd numbered samples. Even though the required clock frequencies can be catered for in the design, the large amount of logic resources presents a problem - it is equivalent to approximately 5 XC4085XL devices.

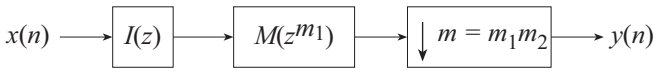

Figure 16: Two-stage IFIR.

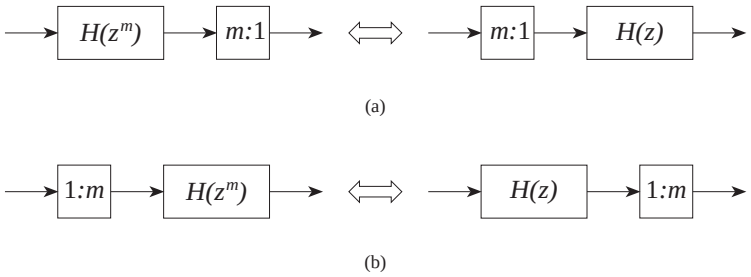

Figure 17: The Noble Identities.

Now consider another approach based on a multistage IFIR decomposition. Figure 16 shows the decimator cascaded with an IFIR architecture. Recall the *Noble Identities*[6] shown in Figure 17. The identity in Figure 17 (a) can be applied to transform the system in Figure 16 to that of Figure 18. For the current example it is convenient to choose $m_1 = 25$ and $m_2 = 2$. Using this factorization of the decimation ratio in combination with the Noble Identities results in the architecture shown in Figure 19. The cascade of $I(z)$ and the downsampling by 25 is implemented using a polyphase filter. The same approach is used for realizing the filter $M(z)$ in cascade with the final output decimate by 2. The resulting filter structure is shown in Figure 20.

Now consider the design of the model and image rejection filters. The model filter must have a passband ripple $\delta_{p,M}$ that is half that of the desired filter $H(z)$. So $\delta_{p,M} = 0.5\delta_p = 0.005$. The stopband requirement is the same as $H(z)$, $\delta_{s,M} = 0.001$. The corner frequency specifications are scaled by the factor $m_1 = 25$. The normalized values for the passband edge frequency $f_{p,M} = 0.0085 \times 0.25 = 0.2125$ and stopband edge frequency $f_{p,M} = 0.25$. An 80-tap equiripple filter will satisfy these requirements. The frequency response of this filter is shown in Figure 21.

The image rejection filter passband ripple is $\delta_{p,I} = 0.5\delta_p = 0.005$, its stopband requirements are the same as the desired filter $H(z)$, $\delta_{s,I} = 0.001$. The passband edge frequency is the same as for $H(z)$, $f_{p,I} = f_{p,H} = 0.0085$. The stopband edge frequency must be selected so that the first image of $M(z^{25})$ is rejected. This value is calculated as $f_{s,I} = 0.03$. The 140-tap image suppressor shown in Figure 22 will meet these specifications. It is instructive to consider some intermediate frequency domain plots to develop an understanding of the IFIR filtering process. Figure 23 is the response $M(z^{25})$ obtained after zero-packing the impulse response $M(n)$. $M(z^{25})$ is obtained by inserting 24 zeroes between samples of $M(z)$'s impulse response. The image suppression filter is applied to $M(z^{25})$ in a multiplicative fashion to produce the final filtered result $\hat{H}(z) = M(z^{25})I(z)$, where $\hat{H}(z)$ is the approximation to our desired filter $H(z)$. The net filter response $\hat{H}(z)$ is shown in Figure 24. To verify that $\hat{H}(z)$ satisfies the original requirements its transition band and passband should be examined. This is done in

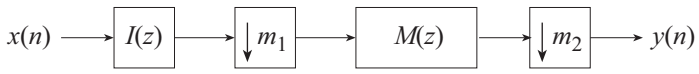

Figure 18: Transformed two-stage IFIR decimator.

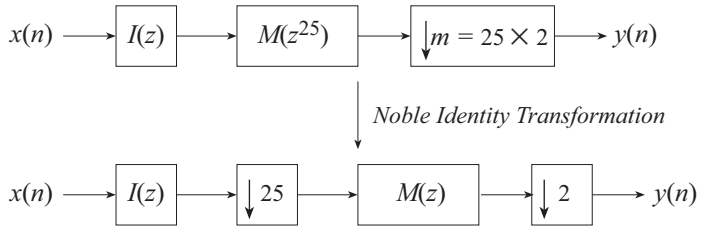

Figure 19: LPF example: transformed two-stage IFIR decimator.

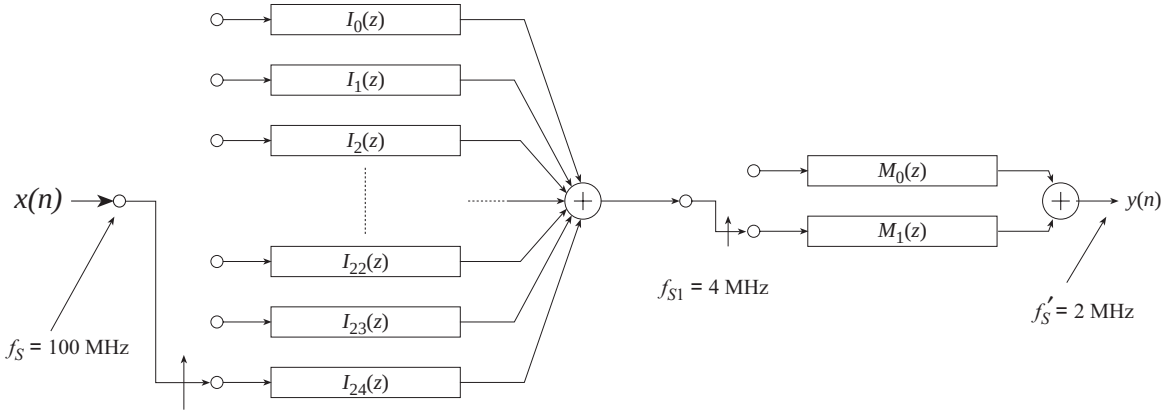

Figure 20: Multistage LPF IFIR decimator using polyphase decomposition.

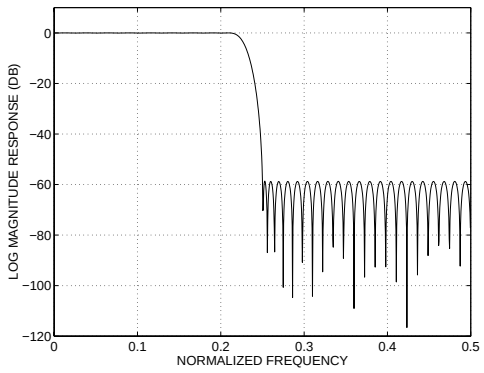

Figure 21: Model Filter.

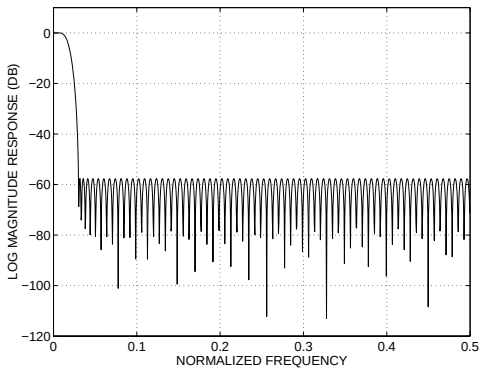

Figure 22: Image filter $I(z)$.

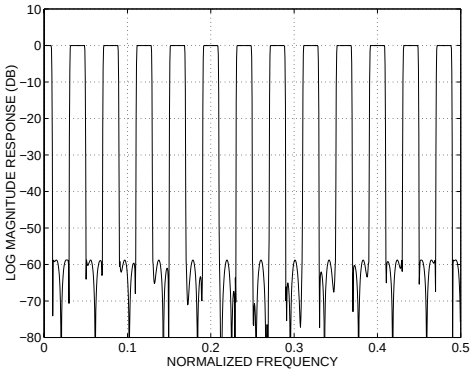

Figure 23: Interpolated model filter $M(z^{25})$.

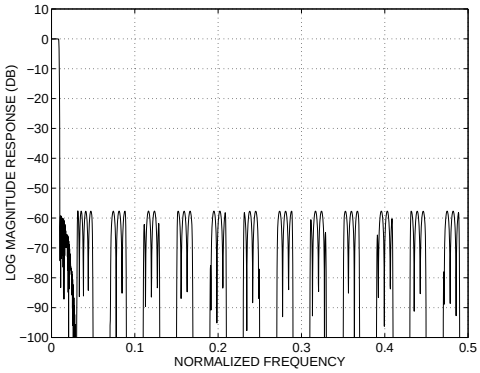

Figure 24: Aggregate response - $M(z^{25})I(z)$.

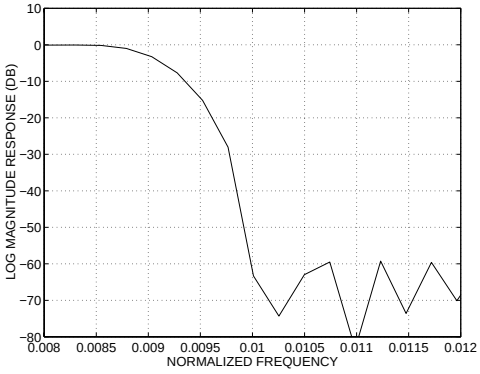

Figure 25: $M(z^{25})I(z)$ - transition band.

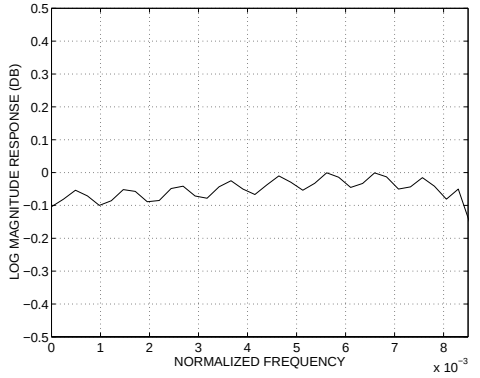

Figure 26: $M(z^{25})I(z)$ - passband.

Figures 25 and 26. The transition band plot shows the response falling the required 60 dB in a bandwidth of 150 kHz. Figure 26 confirms that the passband ripple has been kept acceptably close to the original requirement of $20\log_{10}(0.01+1) = 0.0864$ dB.

The image suppressor is implemented by partitioning $I(z)$ in to $m_1 = 25$ sub-filters I_i, $i = 0, \ldots, m_1 - 1$. Although it is not absolutely necessary for all the filter segments to have the same number of taps, we shall impose this condition for this example. The number of taps per stage is then $N_I = \lceil 140/m_1 \rceil = 6$. The number of CLBs for the 25-to-1 downsampling stage in Figure 20 is $25 \times 58 = 1450$. $M(z)$ and the subsequent decimate-by-2 is implemented with two 40-tap filters $M_0(z)$ and $M_1(z)$. Each filter occupies 302 CLBs, so the number of CLBs required for this processing stage is 604. Excluding the input de-multiplexing and the adder trees that are required in a completed design, the CLB count for the filter modules alone is 2054. Recall from the above calculations that a direct polyphase decimator required $15,400$ CLBs, again excluding the adder tree. The multistage IFIR design represents a savings of $(1 - 2054/15,400) \times 100\% = 86.66\%$. This is obviously a significant savings that more than compensates for the modest increase in the datapath complexity of the design.

5 Why Not IFIRs for Software Programmable DSPs?

It is useful to briefly consider why IFIR filtering is more applicable to FPGA implementation than to realization as software on a DSP microprocessor. Trying to keep the observations as general as possible, no single commercial manufacturers software DSP device will form the focal point of the analysis. Some commonalities can be found across a wide range of these devices to make the discussion broadly applicable.

A key strategy used in software DSPs to facilitate efficient coding and minimize control or branch overheads is the provision of *zero-overhead looping* constructs. Obviously, this is in recognition of the fact that in the majority of the cases, the bulk of the computation burden in a DSP system will be focused in a number of tight loops. A common way to implement an inner-product type calculation in these processors is to loop a MAC instruction. A zero-overhead loop instruction is used to control the calculation, so there is almost no program control penalty. In a typical scenario, the MAC instruction specifies the adjustment to the operand pointers - an increment or decrement by 1. With this mechanization in place it is difficult to take advantage of special cases where one input vector to the calculation may be very sparse - as is the situation with IFIR filters. To try and exploit this type of structure in the impulse response would imply in additional instruction overhead and result in decreasing returns.

This is of course not the case for FPGA implementation where the exact datapath for the IFIR is synthesized, with absolutely no overheads being incurred for taking advantage of zero-packed coefficient data.

6 CONCLUSION

This paper has provided a tutorial review of the IFIR approach to implementing narrow-band filters. The powerful and useful results that are realized when this method is combined with FPGA device technology are evident from the single-rate lowpass filter and 50-to-1 decimator examples that were presented. In the case of the lowpass design, the IFIR architecture provided a 52% increase in hardware efficiency over traditional FPGA implementations that have been previously reported. For the decimator design the results are even more dramatic. Appropriate combining of an IFIR mechanization and polyphase filter decomposition resulted in a 100 MHz to 2 MHz decimation filter that required only 13% of the FPGA logic resources of a polyphase decomposition alone. To the author's knowledge this is the first time that the useful combination of IFIR mechanization and FPGA target hardware have been brought together.

7 REFERENCES

[1] Xilinx Inc., *The Programmable Logic Data Book*, 1998.

[2] J. G. Proakis, and D. G. Manolakis, *Digital Signal Processing Principles, Algorithms and Applications Second Edition*, Maxwell Macmillan International, New York, 1992.

[3] Y. Neuvo, D. Cheng-Yu and S. J. Mitra, "Interpolated Finite Impulse Response Filters", *IEEE Trans. Acoustics Speech and Signal Processing*, Vol. 32, pp. 563-570, June 1984.

[4] S. A. White, "Applications of Distributed Arithmetic to Digital Signal Processing", IEEE ASSP Magazine, Vol. 6(3), pp. 4-19, July 1989.

[5] Xilinx Inc. *Core Solutions Data Book*, 1998.

[6] N. J. Fliege, *Multirate Digital Signal Processing*, John Wiley and Sons, New York, 1996.