The 8th International Conference on Signal Processing Applications and Technology, Toronto Canada, September 13-16 1998.

FPGA Interpolators Using Polynomial Filters

Chris Dick
chrisd@xilinx.com

Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124

fred harris
fred.harris@sdsu.edu

Signal Processing Chair
College of Engineering
San Diego State University
San Diego, CA 92182

Abstract: A fractional delay (FD) filter is a device for performing bandlimited interpolation between the samples of a time-series. It finds application in a vast number of signal processing applications including digital modems, echo cancellers, software radios, array processing, speech coding, image processing, transferring data between quasi-synchronous systems and music technology. In many applications the fractional delay value must be continuously variable. The Farrow filter [1] is a multirate filter structure that offers the option of continuously adjustable delay. This paper presents a derivation of the method proposed by Farrow and demonstrates the performance and complexity of resampling filters using his technique. The field programmable gate array (FPGA) implementation of the Farrow architecture is described. This implementation requires only 2.7% of the logic resources of a conventional polyphase decomposition with the same functionality. The paper also develops some important system options made available to the designer as spin-offs of the derivation.

1 Introduction

Multirate filters are designed and implemented as polyphase P-path filters with each path providing a delay of an integer multiple of $1/P$th of the output sample rate for down-sampling or $1/P$th of the input sample rate for up-sampling. Since each path corresponds to different phase slopes, their outputs are normally referred to by their phase index. For an up-sampling filter we say $d(nP+r)$ or $d_r(n)$ is the output from the rth phase in response to the input at time n. Traditionally the coefficients for the bank of the P-path filters are precomputed and stored, and then accessed by a Q-to-1 sequencing rule to implement a P/Q resampling operation. This structure is shown in Figure 1, and while this figure presents the functional operation, the implementation is actually performed as memory management of the P coefficient sets. For filters designed to operate over a large ratio of P/Q, the number of stages P is made large so that time jitter associated with selecting a phase path nearest to the desired output time position is made acceptably small. The output is selected from the stage nearest the desired time location.

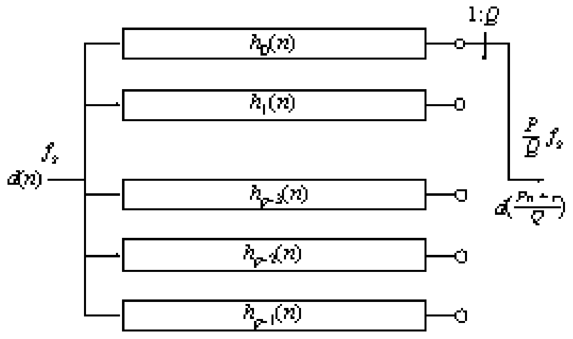

Figure 1. Polyphase upsampling filter.

The nearest neighbor rule is a course interpolator and results in spectral artifacts bounded by $1/(2P)$ [2]. Linear interpolation between two adjacent outputs reduces the level of spectral artifacts to $1/(2P)^2$. For a specific example, to hold spectral artifacts 50 dB below peak signal spectra, we would require 158 stages using the nearest neighbor rule and would only require 9-stages using the linear interpolator.

2 Polynomial Representation of the Filter

One option available to implement an arbitrary resampling filter is to have precomputed the weights of the polyphase filter stages for each resampling ratio and then download them to the processor on startup. Another option is to form a large array representing a highly oversampled filter (large oversampling ratio) with the desired lowpass spectral response and then use the nearest neighbor or linear interpolation rule to form samples of the filter at the desired points required for the specific *P* stage polyphase filter bank. If the filter requires *L* symbols per stage to compute an output, the oversampled prototype is of length 9*L*. As a useful reference, for a square-root Nyquist filter with 0.25 dB implementation loss and with a rolloff $\alpha = 0.4$ and 50 dB sidelobes, *L* is 22, and with a rolloff $\alpha = 0.2$ with the same sidelobes, *L* is 44. This means a prototype table of length 198 to 396 taps would have to available for linear interpolation to an arbitrary set of filter weights.

A third option is to approximate the filter weights in a segment of its impulse response by a low order polynomial and then use the polynomial to compute the filter weights at their desired positions. Consider a 32-stage polyphase filter with 8-taps per stage. This filter is defined by its 256 taps which can be visualized as having been loaded into the polyphase filter set as a two-dimensional array by loading successive columns but processing the data by rows. The *c*-th column of the two-dimensional coefficient set contains the *c*-th coefficient of each polyphase filter. We note that it takes 32 entries to define each column. The filter weights corresponding to the *r*-th stage are located *r*-steps down each column or the fraction (r/P) between input samples. The filter weights corresponding to the $(r+1)$-th stage are of course located $r+1$ steps down each column. This process can be visualized with the aid of Figure 2. Rather than define each column by the 32 samples of the polyphase partition we can pass a low degree polynomial through the coefficients and use the polynomial as an equivalent description.

Figure 3 presents the 32 samples of column zero of a prototype filter along with a fourth order polynomial that approximates these samples. Figure 4 presents the error between the samples and the fourth order polynomial expansion. The error is seen to be in the order of 2×10^{-4} or approximately the error of a 12-bit quantizer. Applying the rule of thumb that the sidelobe structure of a filter follows a 5 dB per bit requirement, this 12-bit error is sufficient to support the 60 dB sidelobes of the prototype filter we are currently examining.

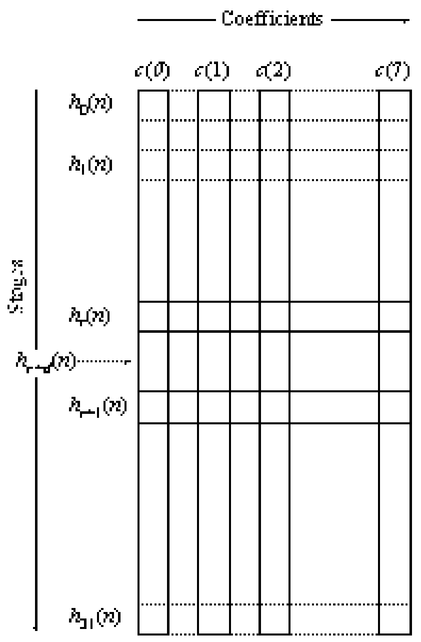

Figure 2: Two-dimensional mapping of polyphase filter coefficient set

In like fashion each column of the polyphase partition is cast into a fourth order polynomial so that the entire 32 stages of the prototype are now represented by 40 coefficients. In fact, filters with an arbitrary number of stages are now represented by the polynomial description. Figure 5 presents the frequency responses of the corresponding filter approximations. As we can see, the spectral artifacts generated by the 4th order polynomial approximation are more than 60 dB below the filters' passband level.

Similarly, the third degree polynomial created 55 dB artifacts while the second order filter generated unacceptably high 28 dB artifacts.

3 Farrow Filter

The filter structure of a polyphase partition is presented in Eq. (1) where the parameter Δ replaces the index r specifying the polyphase stage to emphasize that the displacement Δ is now continuous.

$$d(n+\Delta) = \sum_{k=0}^{7} c_{\Delta}(k) d(n-k) \qquad (1)$$

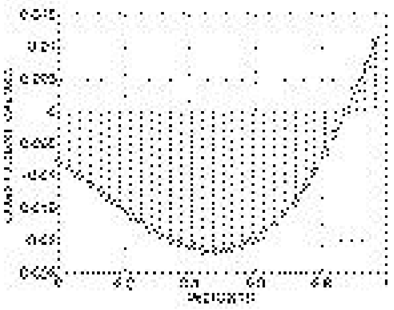

Figure 3: Samples of column zero of polyphase filter and fourth order approximation.

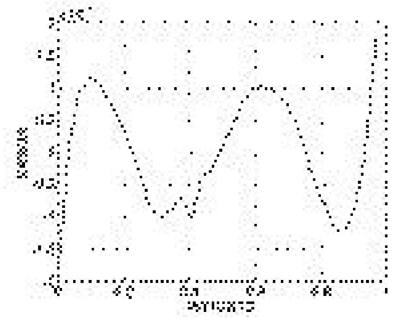

Figure 4: Error between actual samples and fourth order polynomial approximation.

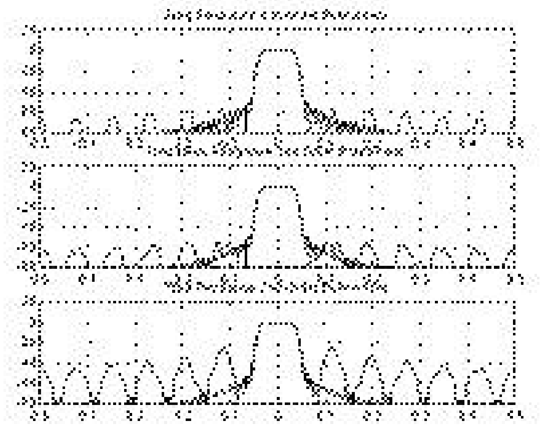

Figure 5: Frequency response of polynomial approximations to prototype filer.

Eq. (1) is recast in the polynomial form in Eq. (2). Exchanging the order of the summations we obtain Eq. (3) which we write more compactly as Eq. (4).

$$d(n+\Delta) = \sum_{k=0}^{7} \sum_{l=0}^{4} b(k,l)\Delta^{l} \; d(n-k) \qquad (2)$$

$$d(n+\Delta) = \sum_{l=0}^{4} \sum_{k=0}^{7} b(k,l) d(n-k) \; \Delta^{l} \qquad (3)$$

$$d(n+\Delta) = \sum_{l=0}^{4} h_{l}(d)\Delta^{l} \qquad (4)$$

It is a convenient perspective to think of the polynomial expansion of the coefficient sets as their Taylor series (actually Tchebyshev) expansion. From this perspective we conclude that the terms $h_l(d)$, defined in Eq. (5), are the result of processing the data with the Taylor series expansion of the data.

$$h_{l}(d) = \sum_{k=0}^{7} b(k,l) d(n-k) \qquad (5)$$

This is more obvious if we examine the two dimensional list of polynomial coefficients $b(k,l)$ where k is the index defining the tap (or column in Figure 2.) and l is the index within the kth column.

The structure of the Farrow filters is shown in Figure 6. Here the data is delivered to each differentiating filter. The outputs of these five filters are the Taylor series expansion of the data, which evaluates the output at offset Δ by Horner's rule as shown in Eq. (6).

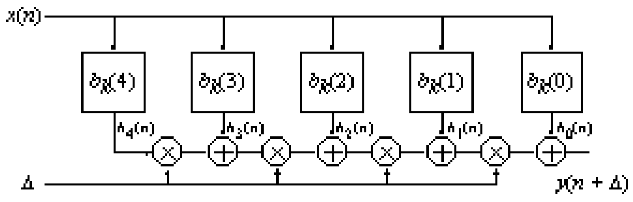

Figure 6: Farrow filter employing 5 polynomial filters.

$$\begin{aligned} y(n+\Delta) &= h_0 + h_1\Delta + h_2\Delta^2 + h_3\Delta^3 + h_4\Delta^4 \\ &= h_0 + \Delta(h_1 + \Delta(h_2 + \Delta(h_3 + \Delta(h_4)))) \end{aligned} \quad (6)$$

Figure 7 presents the spectrum of a sinusoid passed through the Farrow filter to perform a 1-to-6 upsampling operation. The figure also shows the spectrum of the impulse response of the same 1-to-6 filter. Note the sidelobes are suppressed to the 50 dB level for which the prototype filter had been designed.

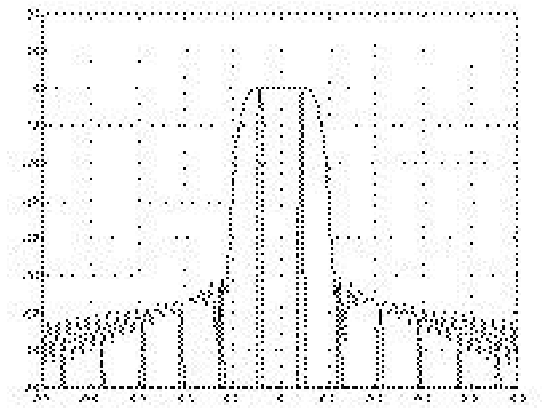

Figure7: Spectrum of Farrow filter output and filter response for 1-to-6 upsampling.

4 FPGA Implementation of the Farrow Filter

The filter requires *m* differentiating filters $b_k(i), i = 0,...m$ and a sum-multiply datapath that combines the sub-filter outputs to form the final output $y(n+\Delta)$ according to Eq. (6). Each of the filters $b_k(i), i = 0,...m$ are 8- tap finite impulse response (FIR) filters. These are most efficiently implemented with Xilinx [4] FPGA technology using a distributed arithmetic [3] (DA) approach. Using a DA implementation, each filter occupies approximately 69 Xilinx 4000 series configurable logic blocks (CLBs). If the input data *x(n)* is kept to 16-bit precision, and a serial distributed arithmetic approach is used to construct the differentiating filters, each sub-filter will produce a new output every 17 clock cycles. This suggests an approach for building the multiply-add datapath: since multiple clock cycles are available to form $y(n+\Delta)$, trade area for speed and use an area efficient serial-parallel (SP) Booth recoded multiplier to form the required products. If the phase offset Δ is kept to a precision of b_Δ bits, and a radix-3 serial-parallel multiplier is used, a product is formed every $b_\Delta / 2$ clock cycles. For $b_\Delta = 8$, 4 clock cycles are required to compute an 8-by-16 product. Therefore the 4 multiplies needed to form $y(n+\Delta)$ can be implemented with one radix-3 multiplier. This unit occupies 50 CLBs. The logic to realize the complete multiply-add datapath can be implemented with 124 CLBs. The entire Farrow filter structure shown in Figure 6 can be realized with $5 \times 69 + 124 = 469$ CLBs.

5 Why the Farrow Filter for FPGAs?

To understand why the Farrow filter architecture can be advantageous for FPGA implementation of fractional delay filters, it is necessary to understand how the same functionality provided by this filter is achieved using the more conventional polyphase filter approach. Using the precisions defined above, consider the implication of selecting $b_\Delta = 8$. With this choice, a simple change in the value of Δ can generate 1 out of a possible 256 interpolants. Or put another way, using 8 bits of precision for Δ splits the unit delay into 256 phases. To achieve this same phase resolution using a polyphase filter would require 256 polyphase filter segments. For some applications, at any one time only a small number - possibly one - of the segment outputs will be in active use. However, if the output signal phase is to be continuously variable, all 256 filter outputs must be available. One option is to have all 256 filters executing concurrently in the FPGA – this is very expensive in terms of logic resources. Each filter arm is the same complexity as the differentiating filters used in the Farrow architecture. The parallel implementation of the 256 filters alone requires $256 \times 69 = 17664$ CLBs! This is clearly unacceptable for current generation FPGAs. Another strategy that could be adopted is to only maintain a small number of polyphase filter segments executing at any one time. The implication is that all of the coefficient sets needed to implement the entire 1-to-256 interpolating filter be available in external memory, and the appropriate filter coefficients be loaded into the FPGA filter as required. To reduce the amount of storage required, the eight coefficient values that define each polyphase segment could be stored. Although the memory requirements are minimized using the later approach, the DA LUTs must now be calculated on-line by either a functional unit implemented in the FPGA itself or an external microprocessor. Since the main emphasis here is on fast on-line adjustment of the fractional delay, any of the methods that store tables of coefficients are unlikely to be suitable solutions for systems where the fractional delay is required to be continuously variable.

Comparing the CLB count of the Farrow filter with the polyphase filter implementation, shows that the Farrow filter implementation requires only 469/17664 = 2.7% of the logic resources required by a full polyphase filter implementation.

For sample rate conversion processes where the conversion ratio is incommensurate (irrational) the output samples generated using a polyphase filter approach are taken from different polyphase segments for each sample period. If a large number of segments are required, the management, as well as the amount of logic occupied by the DA LUT tables can be a problem with an FPGA implementation. This consideration is of course not a concern for the FPGA realization of the Farrow filter since the coefficients are effectively generated on-line. For this example the area advantage leveraged by the Farrow architecture over a polyphase filter is clear.

6 Conclusion

We have re-derived the Farrow filter which supports continuously variable resampling. The filter accomplishes this in a two step process by efficiently describing partitions of an oversampled polyphase filter into segments corresponding to columns of the underlying two-dimensional mapping. The segments are represented by low order polynomials formed by using the *polyfit* m-file in Matlab.

We discovered that the approximating polynomial had to be of fourth degree to sustain artifacts more than 60 dB below the design passband.

The compact representation of the filter segments has system implications when coefficients have to be computed on the fly or in response to a programmed change in sample rate. Filters operating at low input data rates or with large resampling ratios can be implemented efficiently in the Taylor Series of data form.

The second step in forming the Farrow filter reorders the two dimensional summation which first forms the filter coefficients for a specified Δ and then uses the input data to compute the output for that Δ. In the reordered form, the input data is used to compute the Taylor series expansion in the neighborhood of the current input sample and uses that expansion to compute the output at the desired Δ. We note that when the upsampling ratio is greater than 1-to-5 the Farrow filter will compute the output samples with fewer operations than the standard polyphase form. A significant advantage of the polynomial form of the polyphase filter set is the savings in memory needed to store the stage coefficients.

Finally, the Taylor series form of the filter has significant implications for FPGA implementation. This realization occupies only 2.7% of the logic resources of a conventional polyphase filter. The economy of the structure makes it a serious candidate for systems using fractional delay filters when the delay must be varied dynamically.

7 References

[1] C. W. Farrow, ``A Continuously Variable Digital Delay Element'', *Proc. IEEE Int. Symp. Cir. And Sys. (ISCAS)*, Vol. 3. Pp. 2641-2645, Espoo, Finland, June 6-9, 1988.

[2] f. j. harris, ``Forming Arbitrary Length Windows or Filter Sequences From a Fixed Length Reference'', *Eighteenth Annual Asilomar Conference on Circuits, Systems and Computers,* Pacific Grove, Nov. 1984.

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

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