# Distributed Arithmetic FIR Filter v9.0



## Features

- Highly parameterizable drop-in module for Virtex™, Virtex-E, Virtex-II, Virtex-II Pro, Virtex-4, Spartan™-II, Spartan-IIE, Spartan-3, and Spartan-3E FPGAs
- High-performance finite impulse response (FIR), half-band, Hilbert transform, interpolated filters, polyphase decimator, polyphase interpolator, half-band decimator and half-band interpolator implementations
- 2 to 1024 taps
- 1- to 32-bit input data precision
- Signed or unsigned input data
- Signed or unsigned filter coefficients
- 1- to 32-bit coefficient precision
- 1 to 8 channels
- Support for interpolation and decimation factors of between 1 and 8 inclusive
- Coefficient symmetry exploited (symmetric/negative-symmetric) to produce compact implementations
- Serial and parallel filters supported. The user may specify the degree of parallelism and trade off FPGA logic resources for sample rate in order to generate an optimal design
- Data-flow-style core interface and control
- On-line coefficient reload capability
- Incorporates Xilinx Smart-IP™ technology for maximum performance
- To be used with v7.1i or later of the Xilinx CORE Generator™ system

## General Description

The Xilinx filter core is a highly parameterizable, area-efficient high-performance FIR filter. Several highly optimized filters can be generated in the Xilinx CORE Generator: single-rate, half-band, Hilbert transform and interpolated filters, in addition to polyphase decimators and interpolators and half-band decimators and interpolators. Structure in the coefficient set is exploited to produce area-efficient FPGA implementations. Sufficient arithmetic precision is employed in the internal data-path to avoid the possibility of overflow. The filter always presents a full-precision result at its output port.

The conventional single-rate FIR version of the core computes the convolution sum defined in Equation 1, where $N$ is the number of filter coefficients.

$$y(k) = \sum_{n=0}^{N-1} a(n)x(k-n) \quad k = 0, 1, \ldots \qquad \text{Equation 1}$$

The conventional tapped delay line realization of this inner-product calculation is shown in Figure 1.

Although the figure is a useful conceptualization of the computation performed by the core, the actual FPGA realization is quite different. A distributed arithmetic (DA) realization [1] [2] is employed. This approach employs no explicit multipliers in the design, only look-up tables (LUTs), shift registers, and a scaling accumulator.

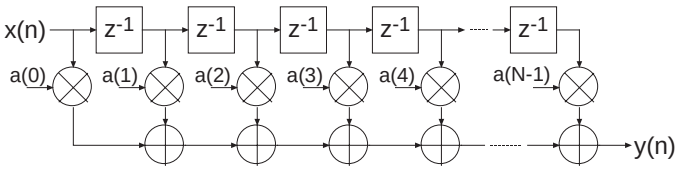



*Figure 1:* **Conventional Tapped-Delay Line FIR Filter Mechanization**

## Filter Realization: Distributed Arithmetic

A simplified view of a DA FIR is shown in Figure 2. In its most obvious and direct form, DA-based computations are bit-serial in nature—serial distributed arithmetic (SDA) FIR. Extensions to the basic algorithm remove this potential throughput limitation [2]. The advantage of a distributed arithmetic approach is its efficiency of mechanization. The basic operations required are a sequence of table look-ups, additions, subtractions and shifts of the input data sequence. All of these functions efficiently map to FPGAs. Input samples are presented to the input parallel-to-serial shift register (PSC) at the input signal sample rate. As the new sample is serialized, the bit-wide output is presented to a bit-serial shift register or *time-skew buffer (TSB).* The TSB stores the input sample history in a bit-serial format and is used in forming the required inner-product computation. The TSB is itself constructed using a cascade of shorter bit–serial shift registers. The nodes in the cascade connection of TSBs are used as address inputs to a look-up table. This LUT stores all possible partial products [2] over the filter coefficient space.

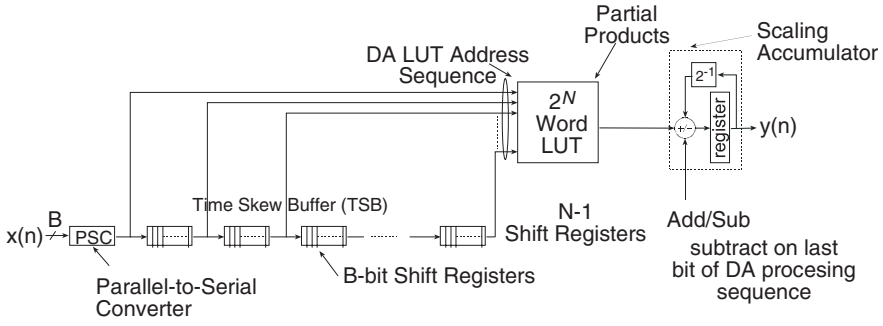



*Figure 2:* **Serial Distributed Arithmetic FIR Filter**

Several observations provide valuable insight into the operation of a DA FIR filter. In a conventional multiply-accumulate (MAC)-based FIR realization, the sample throughput is coupled to the filter length. With a DA architecture, the system sample rate is related to the bit precision of the input data samples. Each bit of an input sample must be indexed and processed in turn before a new output sample is available. For *B*-bit precision input samples, *B* clock cycles are required to form a new output sample for a nonsymmetrical filter, and *B*+1 clock cycles are needed for a symmetrical filter. The rate at which data bits are indexed occurs at the *bit-clock* rate. The bit-clock frequency is greater than the filter sample rate ($f_S$) and is equal to $Bf_S$ for a nonsymmetrical filter and $(B+1)f_S$ for a symmetrical filter. In a conventional instruction-set (processor) approach to the problem, the required number of multiply-accumulate operations are implemented using a time-shared or *scheduled* MAC unit. The filter sample throughput is inversely proportional to the number of filter taps. As the filter length is increased, the system sample rate is proportionately decreased. This is not the case with DA-based architectures. The filter sample rate is decoupled from the filter length. The trade off introduced here is one of silicon

area (FPGA logic resources) for time. As the filter length is increased in a DA FIR filter, more logic resources are consumed, but throughput is maintained.

Figure 3 provides a comparison between a DA FIR architecture and a conventional scheduled MAC-based approach. The clock rate is assumed to be 120 MHz for both filter architectures. Several values of input sample precision for the DA FIR are presented. The dependency of the DA filter throughput on the sample precision is apparent from the plots. For 8-bit precision input samples, the DA FIR maintains a higher throughput for filter lengths greater than 8 taps. When the sample precision is increased to 16 bits, the crossover point is 16 taps.

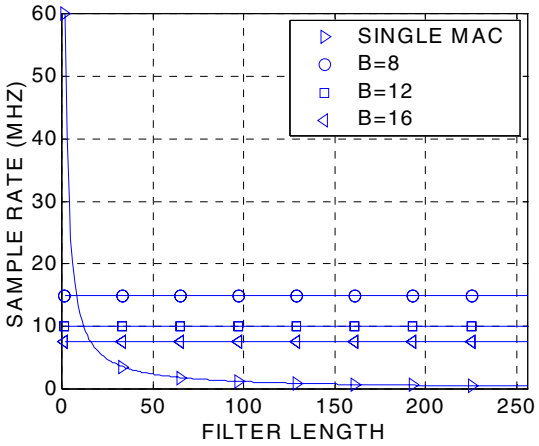



*Figure 3:* **Throughput (Sample Rate) Comparison of Single-MAC-Based FIR and DA FIR as a Function of Filter Length. B is the DA FIR Input Sample Precision. The Clock Rate is 120 MHz.**

Figure 4 provides a similar comparison but for a dual-MAC architecture.

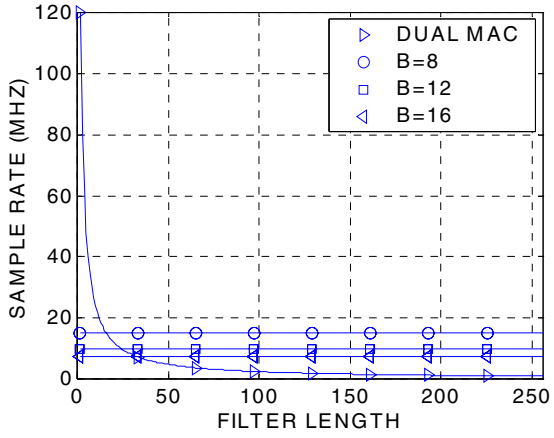



*Figure 4:* **Throughput (Sample Rate) Comparison of Dual-MAC-Based FIR and DA FIR as a Function of Filter Length. B is the DA FIR Input Sample Precision. The Clock Rate is 120 MHz.**

## Increasing the Speed of Multiplication - Parallel Distributed Arithmetic

In its most obvious and direct form, DA-based computations are bit-serial in nature; each bit of the samples must be indexed in turn before a new output sample becomes available (SDA FIR). When the input samples are represented with *B* bits of precision, *B* clock cycles are required to complete an inner-product calculation (for a nonsymmetrical impulse response). Additional speed can be obtained in several ways. One approach is to partition the input words into *M* subwords and process these subwords in parallel. This method requires *M*-times as many memory look-up tables and so comes at a cost of increased storage requirements. Maximum speed is achieved by factoring the input variables into single-bit subwords. The resulting structure is a fully parallel DA (PDA) FIR filter. With this factoring a new output sample is computed on each clock cycle. PDA FIR filters provide exceptionally high performance. The Xilinx filter Core provides support for parallel DA FIR implementations. Filters may be designed that process several bits in a clock period, through to a completely parallel architecture that processes all the bits of the input data during a single clock period. For example, consider a nonsymmetrical filter with 12-bit precision input samples. Using a serial DA filter, new output samples are available every 12 clock periods. If the data samples are processed 2 bits at a time (2-BAAT), a new output sample is ready every 12/2 = 6 clock cycles. With 3-,4-, 6- and 12-BAAT implementations, a new result is available every 4, 3, 2 and 1 clock cycles, respectively.

Another way to view the problem is in terms of the number of clock cycles *L* needed to produce a filter output sample. And indeed, this is how the degree of computation parallelism is presented to the user on the filter design GUI. So, for example, let's consider a filter core with a master system clock (and this is not necessarily the filter sample rate) equal to 150 MHz. Also assume that the input sample precision is 12 bits and that the impulse response is not symmetrical. For this set of parameters, the valid values of *L* (and these are presented on the core GUI) are 12, 6, 4, 3, 2 and 1. The corresponding filter sample rate (or throughput) for each value of *L* is 150/12=12.5, 150/6=25, 150/4=37.5, 150/3=50, 150/2=75 and 150/1=150 MHz respectively. If the filter employs a symmetrical impulse response, the valid values of *L* are different—and this is associated with the hardware architecture that is employed to exploit the coefficient symmetry in order to produce the most compact (in terms of FPGA logic resources) realization. So for a filter with 12-bit precision input samples and a symmetrical impulse response, the valid values of *L* are 13, 7, 5, 4, 3, 2, and 1. Again, using a filter core master clock frequency of 150 MHz, the sample rate for each value of *L* is 11.539, 21.429, 30, 37.5, 50, 75, and 150 MHz respectively.

The higher the degree of filter parallelism (fewer number of clock cycles per output sample or smaller *L*), the greater the FPGA logic resources required to implement the design. Specifying the number of clock cycles per output sample is an extremely powerful mechanism that allows the designer to trade off silicon area with filter throughput.

## Exploiting Filter Symmetry

The impulse response for many filters possesses significant symmetry. This symmetry can be exploited to minimize arithmetic requirements and produce area-efficient filter realizations.

Figure 5 shows the impulse response for a 9-tap symmetric FIR filter:

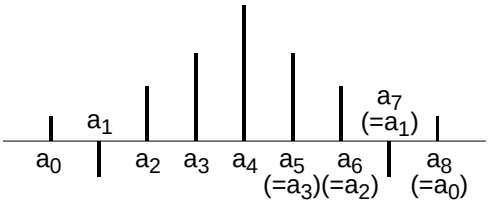



*Figure 5:* **Symmetric FIR - Odd Number of Terms**

Instead of implementing this filter using the architecture shown in Figure 1, the more efficient signal flow-graph in Figure 6 can be used. In general, the former approach requires *N* multiplications and (*N*-1) additions. In contrast, the architecture in Figure 6 requires only [N/2] multiplications and approximately *N* additions. This significant reduction in the computation workload can be exploited to generate efficient filter hardware implementations.

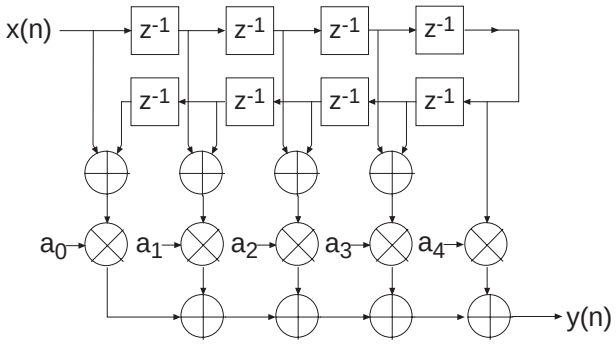



*Figure 6:* **Exploiting Coefficient Symmetry - Odd Number of Filter Taps**

Coefficient symmetry for an even number of terms can be exploited as shown in Figure 7.

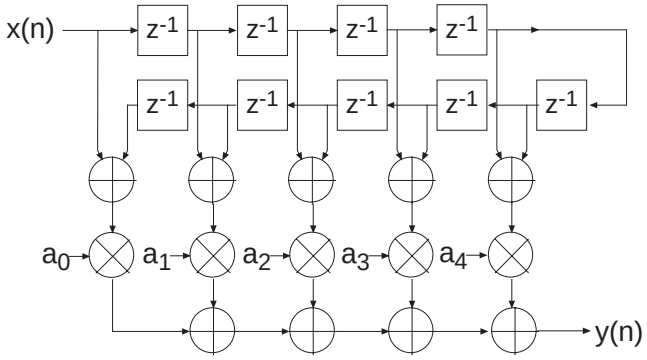



*Figure 7:* **Exploiting Coefficient Symmetry - Even Number of Filter Taps**

The impulse response for a negative, or odd, symmetric filter is shown in Figure 8.

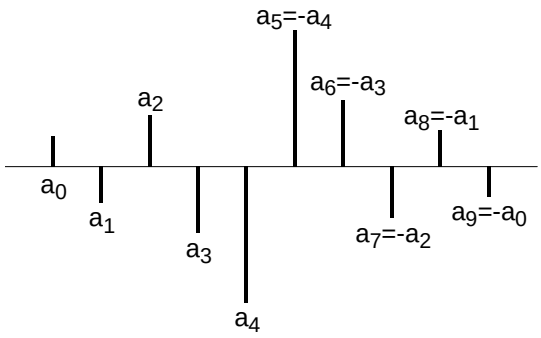



*Figure 8:* **Negative Symmetric Impulse Response**

This symmetry is easily exploited in a manner similar to that shown in Figure 6 and Figure 7. In this case, the middle layer of adders are replaced by subtracters as illustrated in Figure 9.

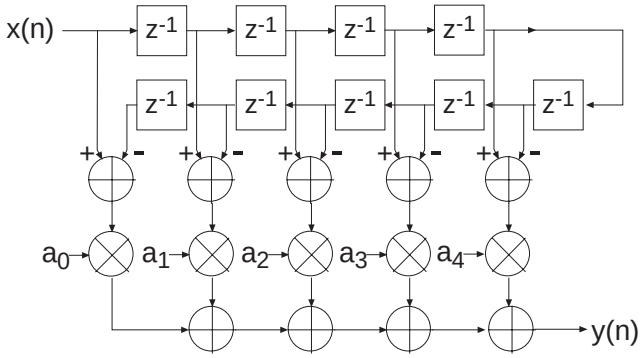



*Figure 9:* **FIR Architecture Exploiting Negative Symmetry**

Again, as highlighted above, the symmetry properties can be utilized to produce an efficient hardware realization.

The example considered here illustrates a filter with an even number of terms; the filter structure for an odd number of terms is a simple extension of the same principle.

Even though none of the filter classes supported by the filter core use explicit multipliers, the various symmetries can still be exploited using a distributed arithmetic implementation to produce efficient FPGA realizations.

The filter compiler interface allows the filter symmetry to be specified. When the impulse response does exhibit symmetry, the filter logic requirements can be significantly reduced in comparison to an implementation that does not exploit the impulse response structure. For example, a 100-tap non symmetric filter with 12-bit data samples and 12-bit coefficients consumes 519 Virtex logic slices [3]. In contrast, a 100-tap symmetric filter is realized with 354 slices. This represents approximately a 30 percent savings in area.

## Filter Throughput

The signal sample rate for a filter is a function of the core bit clock frequency, fclk Hz, the input data sample precision *B*, the number of channels, the number of clock cycles (*L)* per output sample, and the coefficient symmetry. For a single-channel nonsymmetrical FIR filter using *L*=B clock cycles per output sample, the filter sample frequency, or sample throughput, is fclk/B Hz. If the filter is symmetrical, the sample rate is fclk/(*B*+1) Hz. If the number of clock cycles per output sample is changed to *L*=1, the sample throughput is fclk Hz. For *L*=2, the throughput is fclk/2 Hz.

As a specific example, consider a filter with a core clock frequency equal to 100 MHz, 10-bit input samples, *L*=10 and a nonsymmetrical coefficient set. The filter sample rate is 100/10 = 10 MHz. Observe that this figure is independent of the number of filter taps. If a symmetrical realization had been generated, the sample throughput would be 100/11 = 9.0909 MHz. For *L*=1, the sample rate would be 100 MHz (nonsymmetrical FIR). If the input sample precision is changed to 8 bits, with *L*=8, the filter sample rate for a nonsymmetrical filter would be 100/8 = 12.5 MHz.

## Processing Multiple Channels

In many applications the same filter must be applied to several data streams. A common example is the simple digital down converter shown in Figure 10. Here a complex base-band signal $x(n) = x_I(n) + jx_Q(n)$ is applied to a matched filter M(z). The in-phase and quadrature components are processed by the same filter.

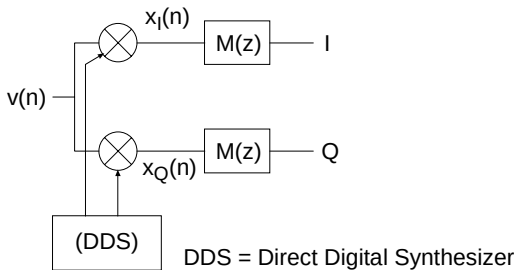



*Figure 10:* **Digital Down Converter**

One candidate solution to this problem is to employ two separate filters. This, however, can be wasteful of logic resources. A more efficient design can be realized using a filter architecture that shares logic resources between multiple sample streams. Several filter classes supported by the filter core provide in-built support for multi-channel processing and can accommodate up to eight independent data streams. As more channels are processed by a filter core, the sample throughput is commensurately reduced. For example, if the sample rate (not the core bit clock *CLK*) for a single-channel filter is $f_S$, a two-channel version of the same filter processes two sample streams, each with a sample rate of $f_S/2$. A three-channel version of the filter processes three data streams and supports a sample rate of $f_S/3$ for each of the streams.

A multi-channel filter implementation is very efficient in logic resources utilization. A filter with two or more channels can be realized using the same amount of logic resources as a single-channel version of the same filter. The tradeoff that needs to be addressed when using multi-channel filters is one of sample rate versus logic requirements. As the number of channels is increased, the logic area remains approximately constant, but the sample rate for an individual input stream decreases. The number of channels supported by a filter core is specified in the filter customization GUI. The multirate filters

(polyphase decimator, polyphase interpolator, half-band decimator, and half-band interpolator) provide support for single-channel operation only.

## Filter Configurations

The filter compiler supports the following eight classes of filters:

- Conventional single-rate FIR
- Half-band FIR
- Hilbert transform [5]
- Interpolated FIR [4] [6]
- Polyphase decimator
- Polyphase interpolator
- Half-band decimator
- Half-band interpolator

The interpolated FIR should not be confused with an interpolation filter. Interpolated filters are single-rate systems employed to produce efficient realizations of narrow-band filters and, with some minor enhancements, wide-band filters can be accommodated.

The filter categories supported by the DA FIR core are described in separate sections below.

### Single-Rate FIR

The basic FIR filter core is a single-rate (input sample rate = output sample rate) finite impulse response filter. Figure 11 shows the schematic symbol for a single-channel instance of this module. Filter input data is supplied on the *DIN* port and filter output samples are presented on the *DOUT* port. The *CLK* signal is the bit-rate clock for the core, and is recognized as being different (higher frequency) to the input signal sample frequency. The *ND*, *RDY*, and *RFD* signals are filter interface/control signals that permit a simple and efficient data-flow style interface for supplying input samples and reading output samples from the filter. The core interface signals are discussed in detail in "Interface, Control, and Timing" on page 30.

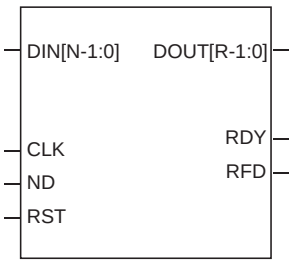



*Figure 11:* **Single-Channel FIR Symbol**

A *P*-channel filter core is shown in Figure 12. The output ports *SEL_I* and *SEL_O* indicate the active input and output data stream respectively. The *SEL_I* signal can be used to multiplex several input sources onto the time-shared input bus *DIN*. *SEL_I* is employed as the multiplexer select signal in this example. In a similar manner, the *SEL_O* signal may be used to de-multiplex the time-division multiplexed filter output bus *DOUT*. This is useful for generating *P* separate filter output samples to present to down-stream processes.

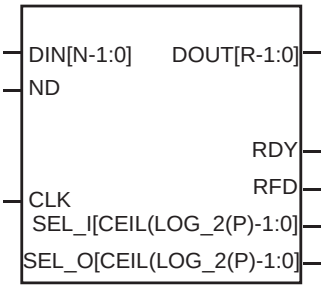



*Figure 12:* **Multi-Channel FIR Symbol**

Table 1 lists the FIR filter port names and port functional definitions.

*Table 1:* **FIR Core Signal Pinout**

| Name | Direction | Description |
|---|---|---|
| DIN[N-1:0] | Input | FILTER INPUT DATA SAMPLE — N-bit wide filter input sample. |
| CLK | Input | BIT CLOCK (active rising edge) |
| ND | Input | NEW DATA (active High) — When this signal is asserted, the data sample presented on the *DIN* port is loaded into the PSC and an inner-product computation is started. *ND* should not be asserted while *RFD* is low. |
| RST | Input | Synchronous reset (active High). Asserting *RST* synchronously with *CLK* resets the filter internal state machines. It does *NOT* reset the filter data memory contents (regressor vector). *RST* resets the counters that control the *SEL_I* and *SEL_O* output signals. *RST* is an optional pin. |
| DOUT[R-1:0] | Output | FILTER OUTPUT SAMPLE R-bit-wide output sample bus for the FIR, half-band and interpolated filters. R depends on the filter parameters (data precision, coefficient precision, number of taps and coefficient optimization selection) and is always supplied as a full-precision output port to avoid any potential for overflow. |
| RDY | Output | FILTER OUTPUT SAMPLE READY (active High) Indicates that a new filter output sample is available on the *DOUT* port. |
| RFD | Output | READY FOR DATA — (active High) Indicates when the final bit of the current data sample is about to be processed and new data may be supplied to the filter. |
| SEL_I[ceil(log_2(P))-1:0] | Output | INPUT CHANNEL SELECT This is a standard binary count generated by the core that indicates the current filter input channel number. *SEL_I* is an optional pin. |
| SEL_O[ceil(log_2(P))-1:0] | Output | OUTPUT CHANNEL SELECT This standard binary count generated by the core indicates the current filter output channel number. *SEL_O* is an optional pin. |
| DOUT_I[N-1:0] | Output | FILTER OUTPUT SAMPLE, Hilbert transform — In-phase (I) component. A Hilbert transform accepts real valued input data and produces a complex result. This port is the real or in-phase component of the result. Since this output port is an access point to the center of the filter memory buffer, it carries the same precision as the input sample data stream, that is, N bits. |
| DOUT_Q[R-1:0] | Output | FILTER OUTPUT SAMPLE, Hilbert transform — quadrature (Q) component. A Hilbert transform accepts real valued input data and produces a complex result. This port is the imaginary or quadrature component of the result. |

## Half-Band FIR

The frequency response for a half-band filter is shown in Figure 13.

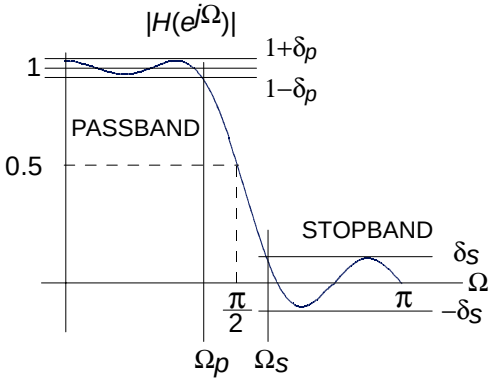



*Figure 13:* **Half-Band Filter—Magnitude Frequency Response**

The magnitude frequency response is symmetrical about quarter sample frequency π/2 radians. The sample rate is normalized to 2π radians/sec. The passband and stopband frequencies are positioned such that

$$\Omega_p = \pi - \Omega_s$$

The passband and stopband ripple, $\delta_p$ and $\delta_s$ respectively, are equal $\delta_p = \delta_s$. These properties are reflected in the filter impulse response. It can be shown [5] that approximately half of the filter coefficients are zero for an odd number of taps. This is illustrated in Figure 14 for an 11-tap half-band filter.

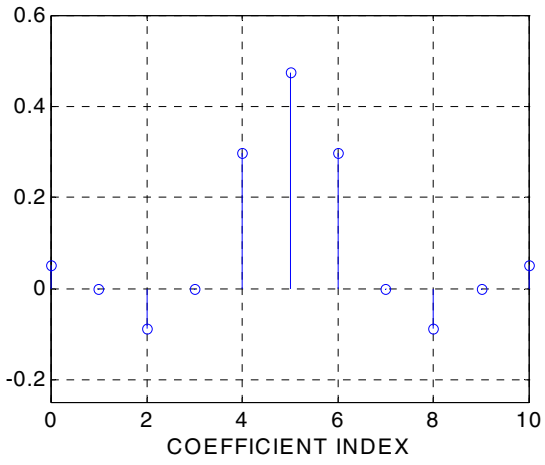



*Figure 14:* **Half-Band Filter Impulse Response**

The interleaved zero values in the coefficient data can be exploited to realize an efficient realization like that shown in Figure 15.

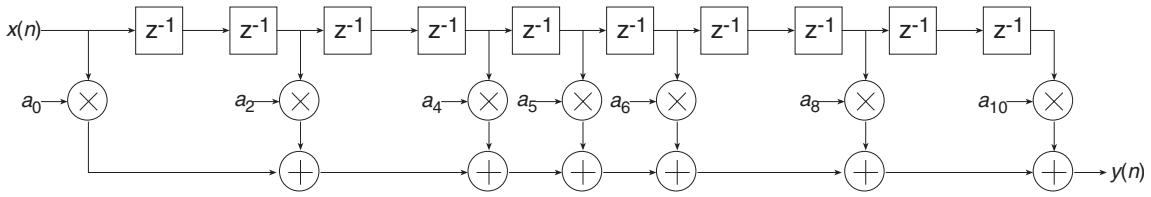



*Figure 15:* **Half-Band Filter Impulse Response**

This same structure can be utilized to generate an efficient DA FPGA implementation. The Half-Band filter selection in the compiler is intended for this purpose. This filter is available in the *Filter Type* field of the user interface. The user must supply the complete list of filter coefficients, including the 0 value samples, when using the half-band filter. The filter coefficient file format is discussed in greater detail in the *Filter Coefficient Data* section.

The half-band filter core has the same port definitions as the single-rate FIR filter.

## Hilbert Transform

Hilbert transformers [5] are used in a variety of ways in digital communication systems.

An ideal Hilbert transform provides a phase shift of 90 degrees for positive frequencies and –90 degrees for negative frequencies. It can be shown [5] that the impulse response corresponding to this frequency domain characteristic is odd-symmetric and has interleaved zeros as shown in Figure 16.

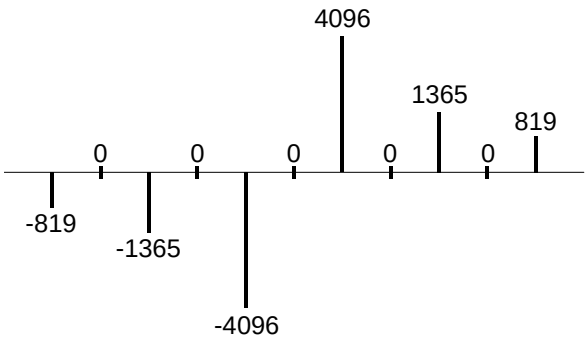



*Figure 16:* **Impulse Response of a Hilbert Transformer**

Both the alternating zero-valued coefficients and the negative symmetry can be utilized to produce an efficient hardware realization. A Hilbert transformer accepts a real-valued signal and produces a complex (I,Q) output signal. The quadrature (Q) component of the output signal is produced by a FIR filter with an impulse response like that shown in Figure 16. The in-phase (I) component is the input signal delayed by an appropriate amount to compensate for the phase delay of the FIR process employed for generating the Q output. This is easily and efficiently achieved by accessing the center tap of the sample history delay of the Q channel FIR filter as shown in Figure 17. In this figure, x(n) is the real-valued input signal and $y_I(n)$ and $y_Q(n)$ are the in-phase and quadrature outputs, respectively.

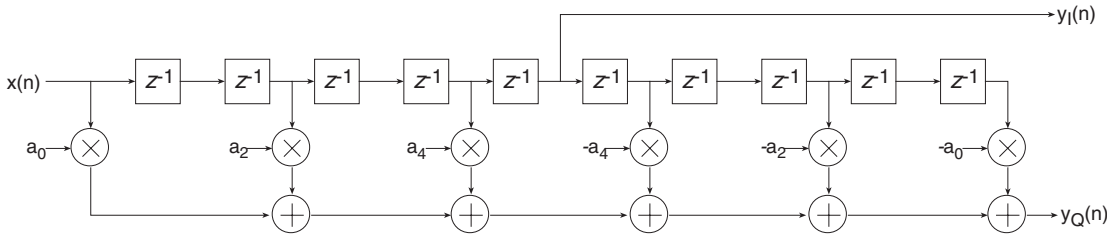



*Figure 17:* **FIR Filter Realization of a Hilbert Transformer**

Figure 18 shows the architecture for a Hilbert transformer that exploits both the zero-valued and the negative symmetry characteristics of the impulse response.

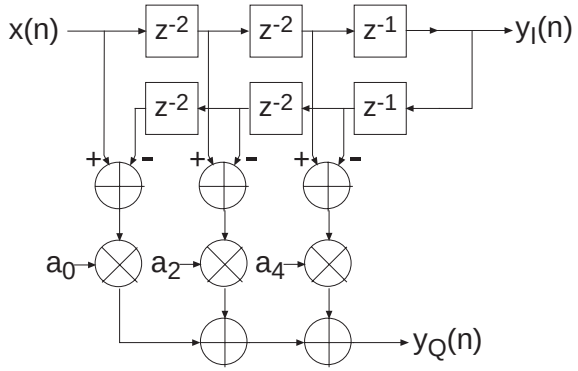



*Figure 18:* **Hilbert Transformer Exploiting Zero-Valued Filter Coefficients and Negative Symmetry**

The DA equivalent of this architecture is used for realizing the Xilinx Hilbert transformer.

Figure 19 is the symbol for the Hilbert transform core. The *DIN* port is the filter input signal, and the ports *DOUT_I* and *DOUT_Q* are the I and Q outputs respectively.

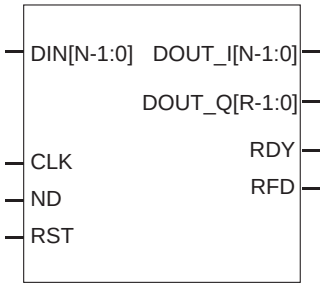



*Figure 19:* **Hilbert Transform Symbol**

The Hilbert transform core has the same data-flow interface and control signals (*ND, RDY,RFD*) as the single-rate FIR filter core. The Hilbert transform core also supports multiple channels as shown in Figure 20.

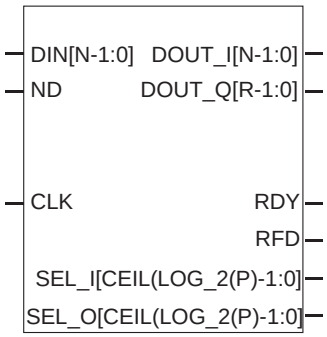



*Figure 20:* **Multi-Channel Hilbert Transform Core**

## Interpolated FIR

An *interpolated FIR* (IFIR) [4] has a similar architecture to a conventional FIR filter, but with the unit delay operator replaced by *k*-1 units of delay. *k* is referred to as the zero-packing factor. An *N*-tap IFIR filter is shown in Figure 21.

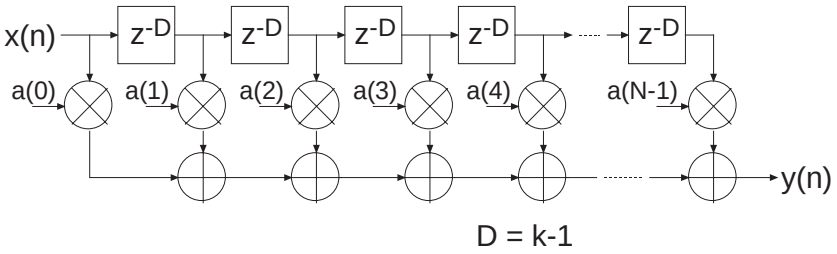



*Figure 21:* **Interpolated FIR (IFIR). The Zero-Packing Factor is *k*.**

This architecture is functionally equivalent to inserting *k*-1 zeros between the coefficients of a prototype filter coefficient set.

Interpolated filters are useful for realizing efficient implementations of both narrow-band and wide-band filters. A filter system based on an IFIR approach requires not only the IFIR but also an image rejection filter. References [4] and [6] provide the details of how these systems are realized, and how to design the IFIR and the image rejection filters.

The IFIR filter core takes advantage of the *k*-1 zeros in the impulse response to realize and area-efficient FPGA implementation. The FPGA area required by an IFIR filter is not a strong function of the zero-packing factor.

**THE IFIR FILTER IS A SINGLE-RATE STRUCTURE. IT DOES NOT PROVIDE AN EMBEDDED SAMPLE RATE CHANGE; THE INPUT SAMPLE RATE IS THE SAME AS THE OUTPUT SAMPLE RATE.**

## Polyphase Decimator

The *polyphase decimation filter* option implements the computationally efficient *M*-to-1 polyphase decimating filter shown in Figure 22.

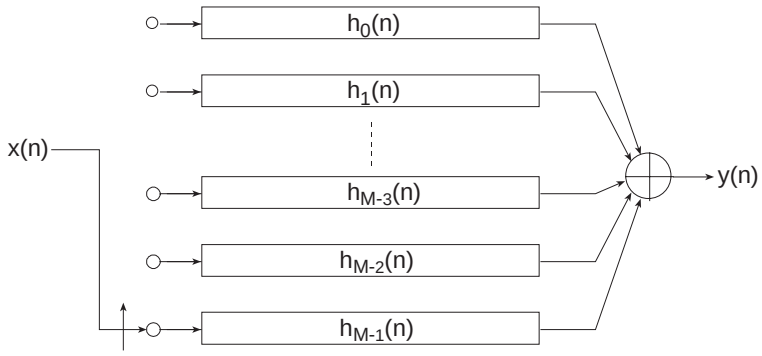



*Figure 22:* ***M*-to-1 Polyphase Decimator**

A set of $N$ prototype filter coefficients $a_0, a_1, \ldots, a_{N-1}$ are mapped to the $M$ polyphase sub-filters $h_0(n), h_1(n), \ldots, h_{M-1}(n)$ according to Equation 2.

$$h_i(n) = a(i + Mr) \quad i = 0, 1, \ldots, M-1 \quad r = 0, 1, \ldots, N-M+i \qquad \text{Equation 3}$$

The polyphase segments are accessed by delivering the input samples $x(n)$ to their inputs via an input commutator which starts at the segment index $i = M-1$ and decrements to index 0. After the commutator has executed one cycle and delivered $M$ input samples to the filter, a single output is taken as the summation of the outputs from the polyphase segments. The output sample $f_s'$ rate is $f_s' = \frac{f_s}{M}$ where $f_s$ is sample rate of the input data stream $x(n), n = 0, 1, 2, \ldots$

We observe that each of the polyphase segments is operating at the low output sample rate $f_s'$ (compared to the high input sample rate $f_s$) and a total of $N$ operations are performed per output point.

In the Xilinx decimator, the polyphase segments are realized using distributed arithmetic techniques. $M$ subfilters, all operating in parallel, are employed in the filter architecture. The polyphase decimator provides support for single-channel operation only.

## Polyphase Interpolator

The polyphase interpolation filter option implements the computationally efficient 1-to-*P* interpolation filter shown in Figure 23.

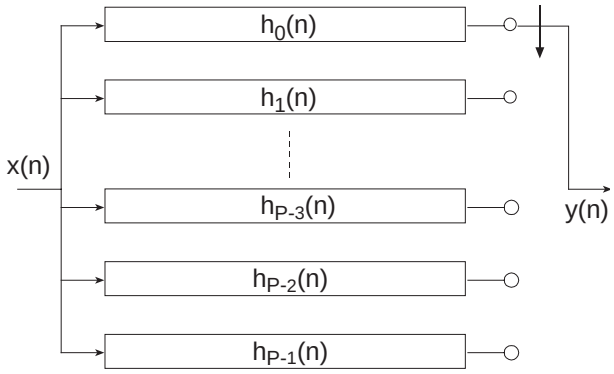



*Figure 23:* **1-to-*P* Polyphase Interpolator**

A set of $N$ prototype filter coefficients $a_0$, $a_1$, ..., $a_{N-1}$ are mapped to the $P$ polyphase subfilters $h_0(n)$, $h_1(n)$, ..., $h_{p-1}(n)$ according to Equation 3.

Each new input sample $x(n)$ engages all of the polyphase segments in parallel. For each input sample delivered to the filter, $P$ output samples, one from each segment, are delivered to the filter output port as indicated by the commutator in Figure 23.

The output sample $f_s'$ rate is $f_s' = f_s P$ where $f_s$ is sample rate of the input data stream $x(n), n = 0, 1, 2, \ldots$. We observe each of the polyphase segments operating at the low input sample rate $f_s$ (compared to the high output sample rate $f_s'$) and a total of $N$ operations performed per output point. Like the polyphase decimator, each filter segment in the interpolator is constructed using distributed arithmetic techniques. $P$ concurrently operating segments are employed in the filter realization.

The polyphase interpolator provides support for single-channel operation only.

## Half-Band Decimator

The half-band decimator is a polyphase filter with an embedded 2-to-1 downsampling of the input signal. The structure is shown in Figure 24.

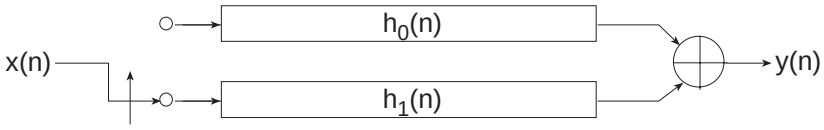



*Figure 24:* **Half-Band Decimation Filter**

The filter is very similar to the polyphase decimator described in "Polyphase Decimator" on page 14 with the decimation factor set to *M*=2. However, there is a subtle difference in the implementation that makes the half-band decimator a more area efficient 2-to-1 down-sampling filter when the frequency response reflects a true half-band characteristic.

The frequency and time response of a half-band filter are shown in Figure 13 and Figure 14 respectively. Observe the alternating zero-valued coefficients in the impulse response.

Figure 25 details a 7-tap half-band polyphase filter when the coefficients are allocated to the two polyphase segments $h_0(n)$ and $h_1(n)$ in Figure 24. Figure 25 (a) is the filter impulse response; note that $a_1 = 0 = a_5$. Figure 25 (b) provides a detailed illustration of the polyphase subfilters and shows how the filter coefficients are allocated to the two polyphase arms. In the bottom arm, $h_1(n)$, the only non-zero coefficient is the center value of the impulse response $a_3$. Figure 25 (c) shows the optimized architecture when the redundant multipliers and adders are removed. The final structure has a reduced computation workload in contrast to a more general 2:1 down-sampling filter. The number of multiply-accumulate (MAC) operations required to compute an output sample has been lowered by a factor of approximately two.

The arithmetic optimizations described above are exploited in the Xilinx half-band decimating filter to minimize the logic requirements of the FPGA implementation.

Even though the previous description and associated figures have described the half-band filter in MAC operations, and the signal flow-graphs indicate explicit multiply operations, as with all of the filters discussed in this document, the underlying implementation is done using distributed arithmetic techniques.

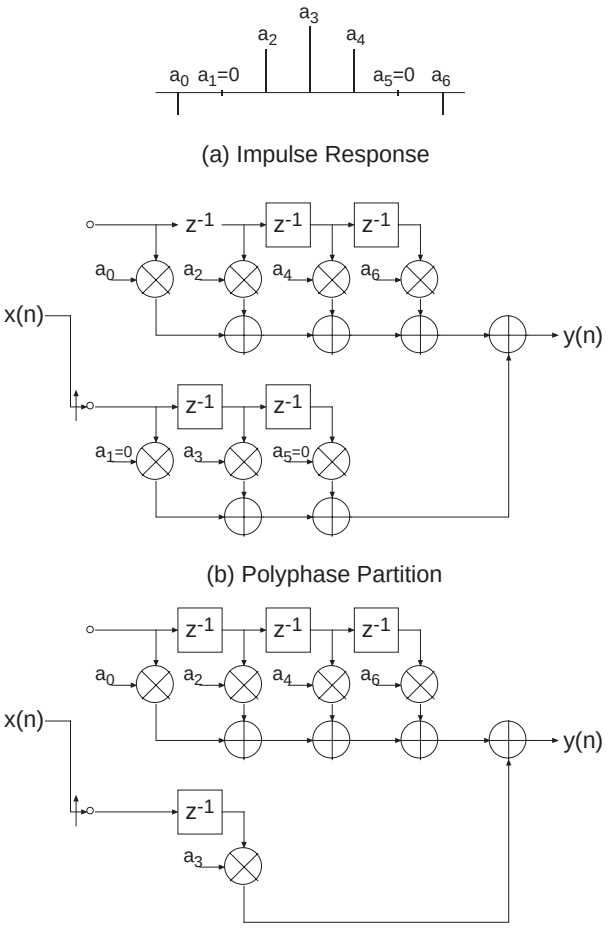



*Figure 25:* **7-Tap Half-Band Decimation Filter; the High Density of Zero-Valued Filter Coefficients is Exploited in the FPGA Realization to Produce a Minimal Area Implementation**

## Half-Band Interpolator

Just as the half-band decimator is an optimized version of the more general polyphase decimation filter, the half-band interpolator is a special case of a polyphase interpolator. The half-band interpolator is shown in Figure 26.

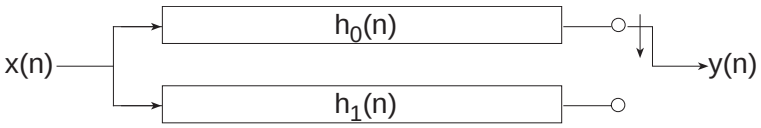



*Figure 26:* **Half-Band Interpolation Filter**

The coefficient set for a true half-band interpolator is identical to that of a half-band decimator with the same specifications. The large number of zero entries in the impulse response is exploited in exactly the same manner as with the half-band decimator to produce hardware-optimized half-band interpolators. The process is presented in Figure 27. Figure 27(a) is the impulse response, Figure 27(b) shows the polyphase partition, and Figure 27(c) is the optimized architecture that has taken full advantage of the 0 entries in the coefficient data.

Like the polyphase decimator and interpolator, the half-band interpolator supports only single-channel input data streams.

## Small Non-Zero Even Terms in a Half-Band Filter Impulse Response

Certain filter design software may result in small non-zero values for the odd terms in the half-band filter impulse response. In this situation, it may be useful to force these values to 0 and re-evaluate the frequency response to assess if it is still acceptable for the intended application. If the odd terms are not identically zero, the hardware optimizations described above are not possible. If the small nonzero value terms cannot be ignored, the general polyphase decimator or interpolator described in "Polyphase Decimator" on page 14 and "Polyphase Interpolator" on page 15, respectively using a rate change of two, are more appropriate.

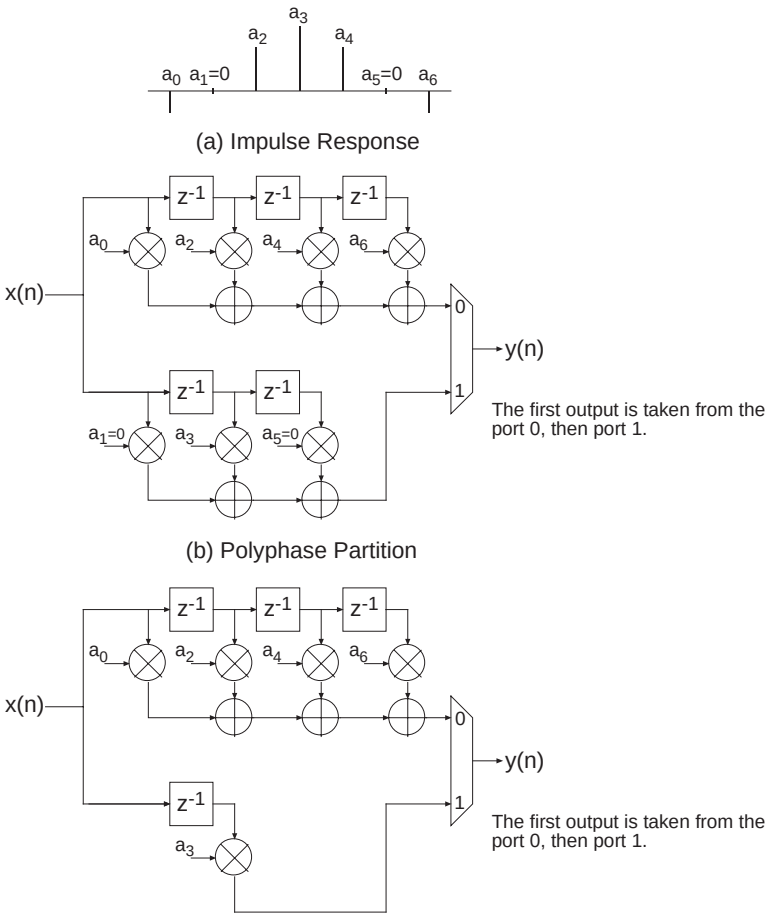



*Figure 27:* **7-Tap Half-Band Interpolation Filter; the High Density of Zero-Valued Filter Coefficients is Exploited in the FPGA Realization to Produce a Minimal Area Implementation**

## On-Line Coefficient Reload

All of the filters provide an interface for loading new coefficient data. While the new coefficient values are being loaded, and some internal data structures are subsequently initialized, the filter ceases to process input samples. The coefficient reload time is a function of the filter length and type.

A high-level view of the reloadable DA FIR architecture is shown in Figure 28. Observe that the DA LUT build engine, in addition to resources to store the new coefficient vector (*coefficient buffer*), is integrated with the FIR filter engine.

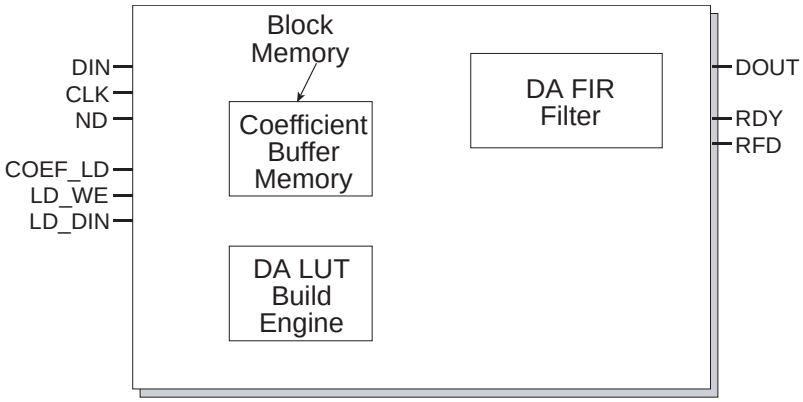



*Figure 28:* **High-Level View of DA FIR with Reloadable Coefficients**

Figure 29 is the symbol for a single-rate FIR supporting coefficient reload. The signals that support the reload operation are *LD_DIN, COEF_LD* and *LD_WE*. The *LD_DIN* port is used to supply the new vector of coefficients to the core. *COEF_LD* is asserted to initiate a load operation and *LD_WE* is a write enable signal for the internal coefficient buffer.

When a coefficient load operation is initiated, the new vector of coefficients is first written to an internal buffer—the coefficient buffer. Once the load operation has completed, the DA LUT build-engine is automatically started. The build-engine uses the values in the coefficient buffer to reinitialize the DA LUT.

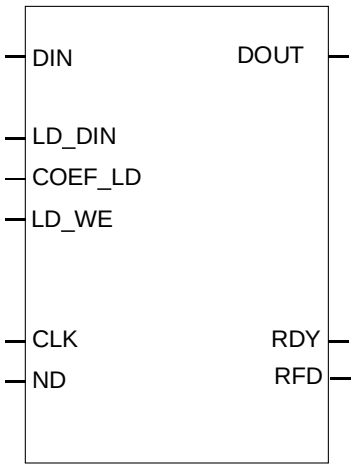



*Figure 29:* **Single-Rate FIR Filter with Coefficient Reload Functionality**

Figure 30 shows the timing for a coefficient reload operation. *COEF_LD* is asserted to start the procedure. The new vector of coefficients is then written to the internal memory buffer synchronously with the core master clock *CLK*. *LD_WE* may be used to control the flow of coefficient data from the external coefficient source—for example, a microprocessor—to the core. *LD_WE* performs a clock-enable function for the load process.

Asserting *COEF_LD* forces *RFD* to the inactive state (Low), indicating that the core cannot accept any new input samples. Note that during the reload operation the filter inner-product engine is suspended. Once the new coefficients have been loaded and the DA LUT build engine has constructed the new partial-product lookup tables, *RFD* is asserted indicating the core is ready to accept new input samples and resume normal operation. The filter sample history buffer (regressor vector) is cleared when a new coefficient vector is loaded.

Asserting *COEF_LD* also forces *RDY* to the inactive state (Low). *COEF_LD* may be reasserted again at any point during an update procedure (even once the DA LUT build-engine is running) to start a new coefficient configuration.

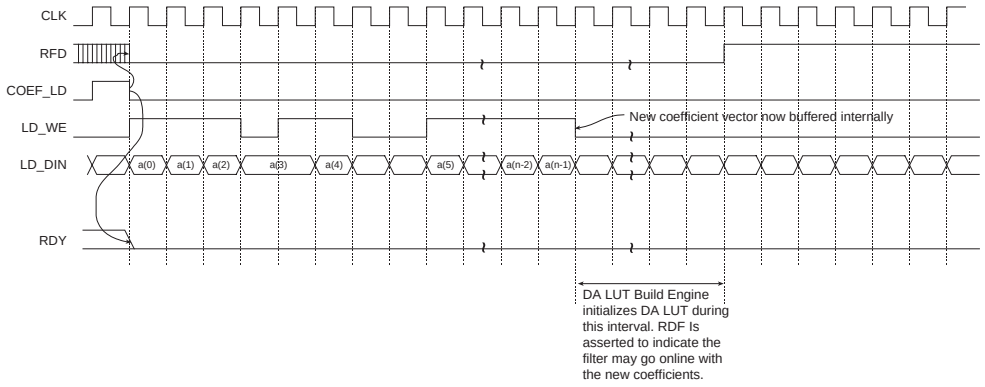



*Figure 30:* **Coefficient Reload Timing**

The number of clock cycles required to load a coefficient vector is a function of several variables, including the filter length and filter type. Table 2 presents the reload time (in clock cycles) for each filter class.

## Coefficient Reload—Typical Use Model

The typical sequence of events that engage the coefficient reload are:

1. Pulse *COEF_LD* for a single clock cycle to initiate a coefficient load operation.
2. Supply a length *N* vector new coefficient data on the *LD_DIN* port. The coefficients can be written to the internal buffer at a rate of one value per clock cycle. The coefficient source may use *LD_WE* to control the rate at which coefficients are delivered. This is useful for systems in which the coefficient source may not be able to accommodate the core bit clock; remember, the coefficients are written to the internal buffer synchronously with the core master clock signal *CLK*.
3. Wait until *RFD* is asserted, indicating the filter may now be put back on-line, and process input samples with the new coefficient vector.

*Table 2:* **Coefficient Reload Times as a Function of Filter Type**

| Filter Type | Latency $L$[1] |
|---|---|
| Single-Rate FIR [2,3] | $L = \left( \left\lfloor \frac{N+3}{4} \right\rfloor \times 64 \right) + 18$ |
| Halfband | $L = \left( \left\lfloor \frac{\left\lfloor \frac{N+1}{2} \right\rfloor + 4}{4} \right\rfloor \times 64 \right) + 18$ |
| Hilbert Transform | $L = \left( \left\lfloor \frac{\left\lfloor \frac{N+1}{2} \right\rfloor + 3}{4} \right\rfloor \times 64 \right) + 18$ |
| Interpolated | $L = \left( \left\lfloor \frac{N+3}{4} \right\rfloor \times 64 \right) + 18$ |
| Interpolation Decimation[4] | $L = (S \times 64) + 18$ <br> $Y = N - \left\lfloor \frac{N}{4R} \right\rfloor \times 4R$ <br> if $Y = 0$, then $S = \frac{N}{4}$ <br> if $0 < Y < R$, then $S = \left( \left\lfloor \frac{N}{4R} \right\rfloor \times R \right) + Y$ <br> if $Y \geq R$ and $Y \neq N$, then $S = \left( \left\lfloor \frac{N}{4R} \right\rfloor + 1 \right) \times R$ <br> if $Y = N$, then $S = R$ |
| Decimating Halfband Interpolating Halfband | $L = \left( \left\lfloor \frac{\left\lfloor \frac{N+1}{2} \right\rfloor + 3}{4} \right\rfloor \times 64 \right) + 82$ |

1. Latency equations calculate number of cycles between the last coefficient written into block memory and RFD being asserted.

2. $\lfloor x \rfloor$ is the symbol for rounding $x$ down to the nearest integer (for example, $\lfloor 3.2 \rfloor = 3$)

3. $N$ is the effective number of taps:

   - for Non Symmetric and Negative Symmetric filters, $N$ = Number of Taps
   - for Symmetric filters $N = \left\lfloor \frac{\text{Number of Taps} + 1}{2} \right\rfloor$:
   - $R$ is the Sample Rate Change ( $S$ and $Y$ are temporary variables).

## CORE Generator Parameters

A filter core is customized using a configuration wizard. The wizard screens are shown in Figure 31 through Figure 33.

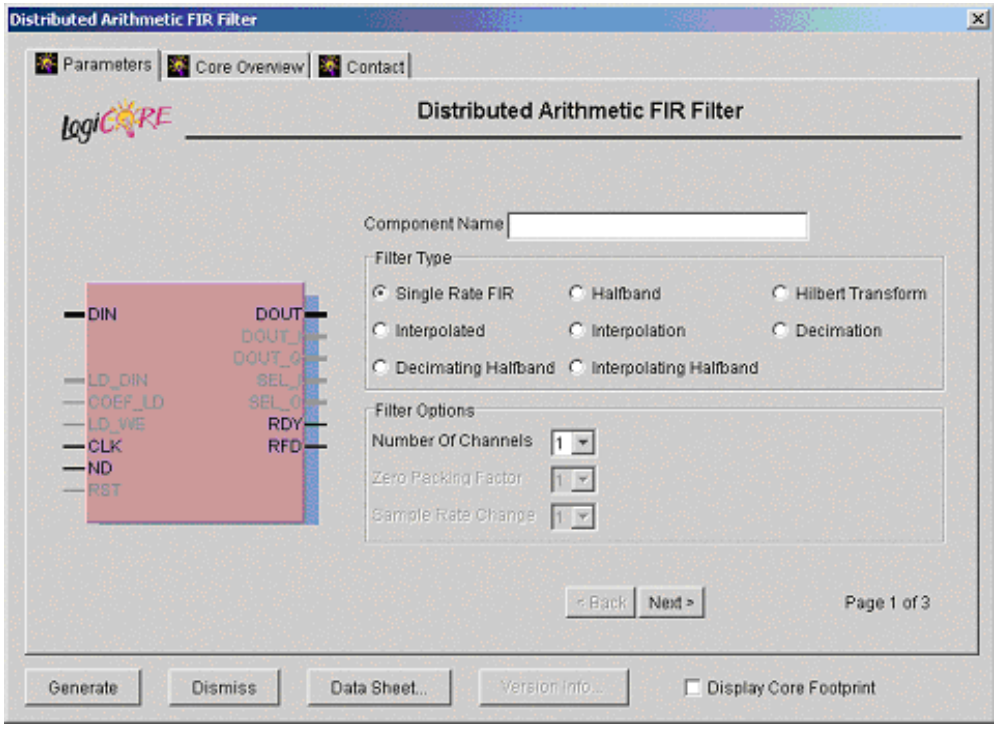



*Figure 31:* **Filter Parameterization Screen—Page 1**

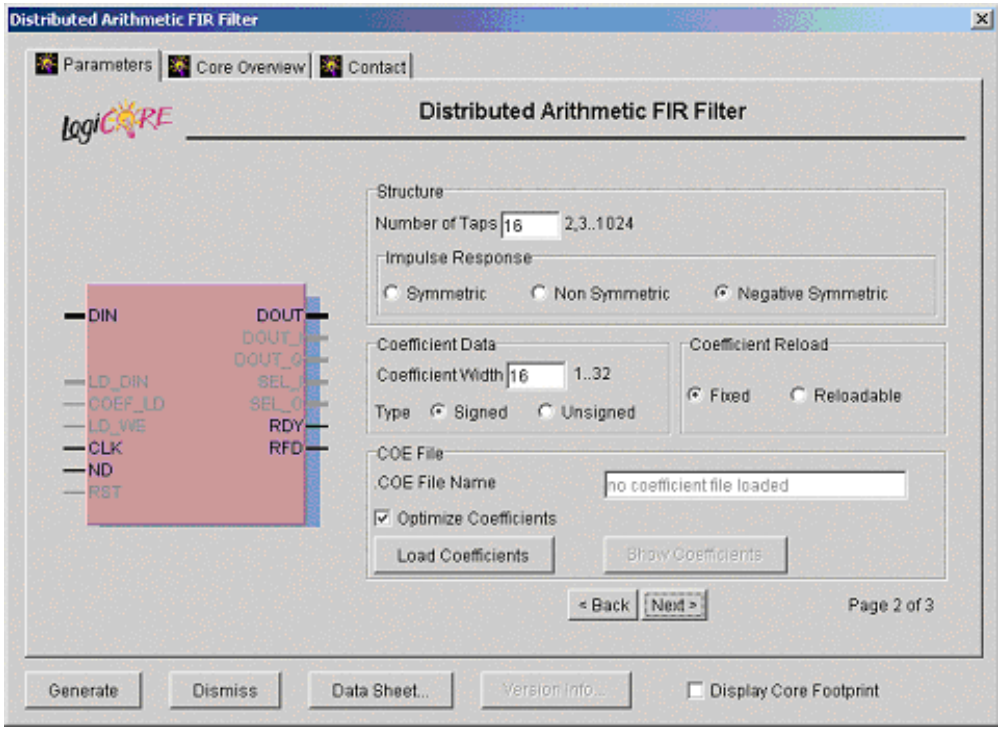



*Figure 32:* **Parameterization Screen—Page 2**

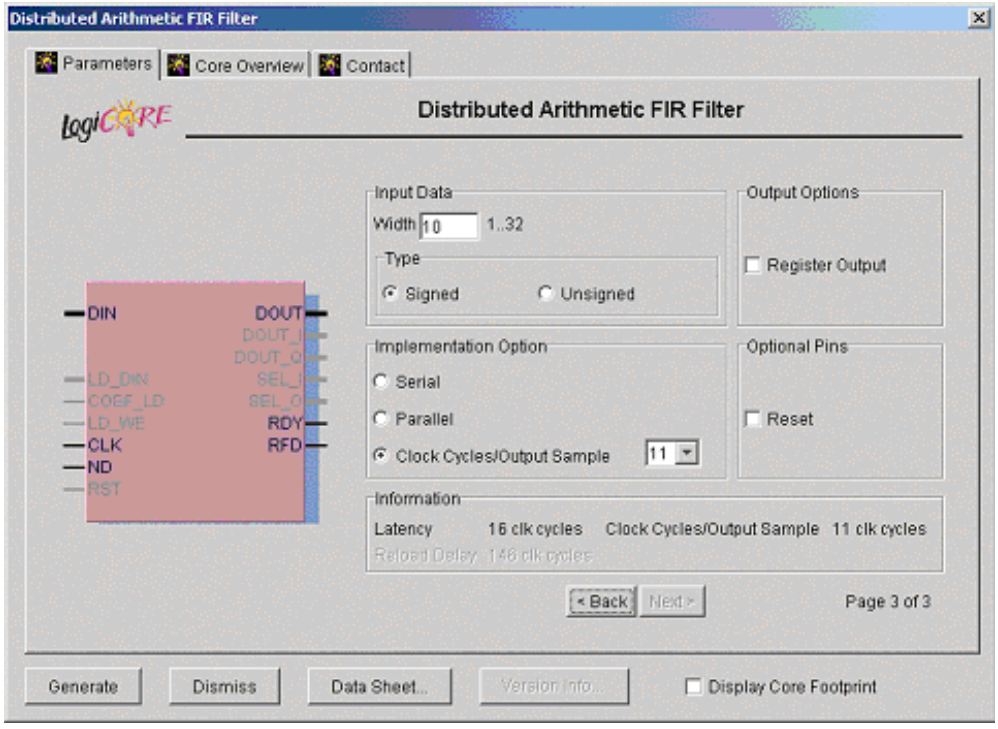



*Figure 33:* **Filter Parameterization Screen—Page 3**

The user-supplied parameters are:

- **Component Name:** The user-defined filter component name
- **Filter Type:** Eight filter types are supported:
  - Single-rate FIR
  - Half-band FIR
  - Hilbert transform
  - Interpolated FIR
  - Polyphase decimator
  - Polyphase interpolator
  - Half-band decimator
  - Half-band interpolator.
- **Number of Channels:** The number of channels processed by the filter. One to eight channels can be accommodated by a single filter core. The polyphase decimator and polyphase interpolator provide single-channel support only.
- **Zero Packing Factor:** This field is applicable to the interpolated filter only. The zero packing factor specifies the number of 0s inserted between the coefficient data supplied by the user in the .coe (filter coefficient file). This is an integer value between 2 and 8 inclusive. A zero packing factor of *k* inserts *k*-1 0s between the supplied coefficient values.
- **Sample Rate Change:** This field is applicable to the polyphase decimator and interpolator structures. When the decimator is selected, the Sample Rate Change value defines the decimation factor. For the interpolation filter, it defines the up-sampling factor. Sample rate changes of between 1 to 8 inclusive are supported for both up-sampling and down-sampling.
- **Number of Taps:** The number of filter taps. For a symmetric impulse response (either even or odd symmetric) the number of filters taps is between 2 and 1024 inclusive. For a nonsymmetrical coefficient set the range is 2 to 1024 inclusive.
- **Impulse Response:** Indicates structure present in the coefficient set. The user may specify a symmetric, negative (odd)-symmetric or nonsymmetric impulse response.
- **Coefficient Width:** The bit precision of the filter coefficients. This is an integer value between 1 and 32 inclusive.
- **Coefficient Data Type:** The coefficient data can be specified as either signed or unsigned. When the signed option is selected, conventional two's complement representation is assumed.
- **.COE File Name:** Coefficient file name. This is the file of filter coefficients. The file has a *.coe* extension and the file format is described in "Filter Coefficient Data" on page 38.
- **Coefficient Reload:** When the Fixed radio button on the Coefficient Reload panel is selected, the filter Core is generated without a coefficient reload interface. When the Reloadable button is selected, a coefficient reload interface is provided on the Core.
- **Optimize Coefficients:** The look-up tables employed in the filter mechanization can be optimized to minimize the amount of FPGA logic fabric employed by the core. The optimization is data (filter coefficient set) dependent.
- **Load Coefficients:** The filter coefficients are supplied in a coefficient or *coe* file. This is an ASCII file with a ".coe" extension. The file format is described in "Filter Coefficient Data" on page 38.

Activating this tab presents a browser window that lets the user select a coefficient file.

- **Show Coefficients:** Selecting this tab displays the filter coefficient data.
- **Input Data Width:** The precision (in bits) of the filter input data samples. The input sample precision is an integer value between 1 and 32 inclusive.
- **Input Data Type:** The filter input data can be specified as either signed or unsigned. The signed option employs conventional two's complement arithmetic.
- **Output Options:** The filter output bus can be registered or unregistered. When the registered output option is selected, the filter output bus *DOUT* is maintained at the core output between successive assertions of *RDY*. In the unregistered mode, the output sample is valid only when *RDY* is active. At other times, the port changes on successive clock cycles.
- **Implementation Option:** Selecting the *Serial* option generates an SDA FIR filter. This is a fully serial DA FIR filter. In this case, if a nonsymmetric impulse response is specified, *B* (*B* is the bit precision of the input data) clock cycles are required to generate a new output sample (*B* clock cycles per output point). If a symmetric impulse response is employed, *B*+1 clock cycles are required per output point.

  If the Parallel filter is specified, a fully parallel PDA filter is produced. The fully parallel filter produces a new output sample on every clock edge. Choosing the Clock Cycles/Output Sample option allows the degree of filter parallelism to be specified using the associated pull-down menu. The menu presents the valid set of values (L) that can be selected to specify the number of cycles per output sample. For example, selecting L=3 for a polyphase decimator results in a filter where each internal DA sub-filter generates a new output sample every three clock cycles irrespective of the filter input sample precision.

  For all of the polyphase decimation filters, including the half-band decimator, the Clock Cycles/Output Sample value refers to the individual filters that are employed to construct the multirate structure.
- **Information:** This field reports the filter latency (the number of clock cycles between presenting an input data sample and the corresponding filter output sample) and the number of clock cycles per sample. The filter latency is also available in the component instantiation file. This file has a base-name that is the same as the filter component name, with a .vho extension for a VHDL design flow or a .veo extension for a Verilog flow. For example, if the filter component name is my_filt, and a VHDL flow has been selected, the instantiation file is named my_fir.vho. If a Verilog flow had been selected this file would be called my_fir.veo.

  The information field also reports the number of cycles that the coefficient reload process requires.

## XCO File Parameters

The parameters supplied via the filter customization wizard are captured and logged to the .xco file. The full name of this file is the *Component Name* with an *.xco* file extension. Table 3 defines the .xco file parameter names and range specifications.

*Table 3:* **XCO File Parameter Names, Definitions, and Range Specifications**

| Parameter Name | Definition | Range |
|---|---|---|
| BusFormat | Controls the notation employed for identifying buses in the output edif netlist file. | {BusFormatAngleBracket \| BusFormatParen} |
| SimulationOutputProducts | Core HDL simulation selection—either VHDL or Verilog. | {VHDL \| VERILOG} |
| XilinxFamily | The FPGA target device family. | {Virtex \| Virtexe \| Spartan2 \| Virtex2 \| Virtex2p \| Spartan3} |
| DesignFlow | HDL flow specifier. | {VHDL \| VERILOG} |
| FlowVendor | Design flow vendor information. | {Other \| Synplicity \| Exemplar \| Synopsis \| Foundation \| ISE \| Innoveda} |
| coefficient_file | File of filter coefficient values. | Any valid file name for the user's operating system consisting of the letters a…z, 0…9 and "_." |
| coefficient_data_type | The filter coefficient data type. When the type *signed* is selected, conventional 2's complement arithmetic is employed. | {signed \| unsigned} |
| number_of_taps | The number of filter taps. | [2,…,1024] |
| register_output | When *true,* an output register is inserted at the output of the filter datapath. In this case, the filter output remains valid during successive transitions of the filter output port(s). When this parameter is *false,* the filter output is not registered and the output sample is valid only during the clock cycle demarcated by the *RDY* control signal. | {true \| false} |
| optimize_coefficients | When *true,* logic optimization is performed on the filter look-up tables. Selecting optimization results in the most compact (minimum FPGA logic resources) implementation. If this parameter is false, no logic optimization is performed. | {true \| false} |

*Table 3:* **XCO File Parameter Names, Definitions, and Range Specifications *(Continued)***

| Parameter Name | Definition | Range |
|---|---|---|
| component_name | Textbox that defines the filter component name. | Any valid file name for the user's operating system consisting of the letters a…z, 0…9 and "_." |
| zero_packing_factor | This field is applicable to *interpolated filters* only and controls the number of 0s inserted between the user-supplied coefficient values. A value of 2 results in a single 0 valued entry between the user coefficients; a value of 3 inserts 2 0s between the user coefficients, and so on. For all filters other than the *interpolated* filter, this parameter should be 1. | [1,…,8] |
| impulse_response | This parameter allows the user to identify structure in the filter coefficient data. Coefficient vectors that are identified (explicitly by the user) as being structured (symmetric or negative symmetric) result in minimal size hardware implementations. | {non_symmetric \| symmetric \| negative_symmetric} |
| sample_rate_change | This parameter specifies the sample rate change embedded in the filter. For all single-rate filters, the rate change is considered to be 1. | [1,…,8] |
| number_of_channels | The number of channels supported by the filter. All multirate filters support only a single channel. | [1,…,8] |
| clock_cycles_per_output | Number of clock cycles required to generate a filter output sample. In the context of any of the multirate filters, this value is associated with the subfilters (polyphase segments) and not the final output result. | The valid set of values for this parameter is a function of several other parameters, including *input_data_width* and *impulse_response*. This value is a minimum of 1, corresponding to a full parallel implementation in which a new output sample is available on every clock edge, to a maximum of *input_data_width+1*. |
| number_of_taps | The number of filter taps. | [2,3,…,1024] |
| filter_type | The filter type specifier. | {single_rate_fir \| halfband \| hilbert_transform \| interpolated \| interpolation \| decimation \| decimating_halfband \| halfband_interpolating} |

*Table 3:* **XCO File Parameter Names, Definitions, and Range Specifications *(Continued)***

| Parameter Name | Definition | Range |
|---|---|---|
| coefficient_width | Number of bits used to represent the filter coefficient values. | [1,2,…,32] |
| input_data_width | Number of bits used to represent the filter input samples. | [1,2,…,32] |
| implementation_option | This field defines the degree of filter parallelism. | {clock_cycles_per_output_sample \| parallel \| serial} |
| input_data_type | The filter input sample data type. When the type *signed* is selected, conventional 2's complement arithmetic is employed. | {signed \| unsigned} |
| coefficient_reload | This parameter controls the presence (or not) of a coefficient reload interface. When defined as *stop_during_reload,* the interface is included. When defined as *fixed_coefficients,* no coefficient reload feature is present. | {stop_during_reload \| fixed_coefficients} |
| reset | Indicates if a synchronous reset input (*RST)* is included. | {true \| false} |

Figure 34 is an example .xco file. The "#" character at the start of the first five lines in the example identify in-line comments.

```
# Xilinx CORE Generator 5.1i; Cores Update # 2
# Username = chrisd
# COREGenPath = C:\Xilinx\coregen
# ProjectPath = C:\ip_portfolio\DA_FIR\datasheet\eip2\xilinx
# ExpandedProjectPath = C:\ip_portfolio\DA_FIR\datasheet\eip2\xilinx
# OverwriteFiles = False
# Core name: rrc
# Number of Primitives in design: 4189
# Number of CLBs used in design cannot be determined when there is no RPMed logic
# Number of Slices used in design cannot be determined when there is no RPMed
logic
# Number of LUT sites used in design: 1360
# Number of LUTs used in design: 1191
# Number of REG used in design: 1680
# Number of SRL16s used in design: 169
# Number of Distributed RAM primitives used in design: 0
# Number of Block Memories used in design: 0
# Number of Dedicated Multipliers used in design: 0
# Number of HU_SETs used: 0
#
SET BusFormat = BusFormatParen
SET SimulationOutputProducts = VHDL
SET XilinxFamily = Virtex2
SET OutputOption = DesignFlow
SET DesignFlow = VHDL
SET FlowVendor = Synplicity
SET FormalVerification = None
SELECT Distributed_Arithmetic_FIR_Filter Virtex2 Xilinx,_Inc. 7.0
CSET implementation_option = Serial
CSET optimize_coefficients = true
CSET zero_packing_factor = 1
CSET input_data_type = Signed
CSET number_of_channels = 1
CSET register_output = true
CSET component_name = rrc
CSET sample_rate_change = 8
CSET coefficient_reload = Fixed_Coefficients
CSET reset = false
CSET filter_type = Interpolation
CSET input_data_width = 10
CSET impulse_response = Non_Symmetric
CSET clock_cycles_per_output_sample = 10
CSET coefficient_data_type = Signed
CSET coefficient_file =
C:\ip_portfolio\DA_FIR\datasheet\eip2\xilinx\rrc_interp.coe
CSET number_of_taps = 129
CSET coefficient_width = 14
GENERATE
```

*Figure 34:* **Example .xco File**

## Interface, Control, and Timing

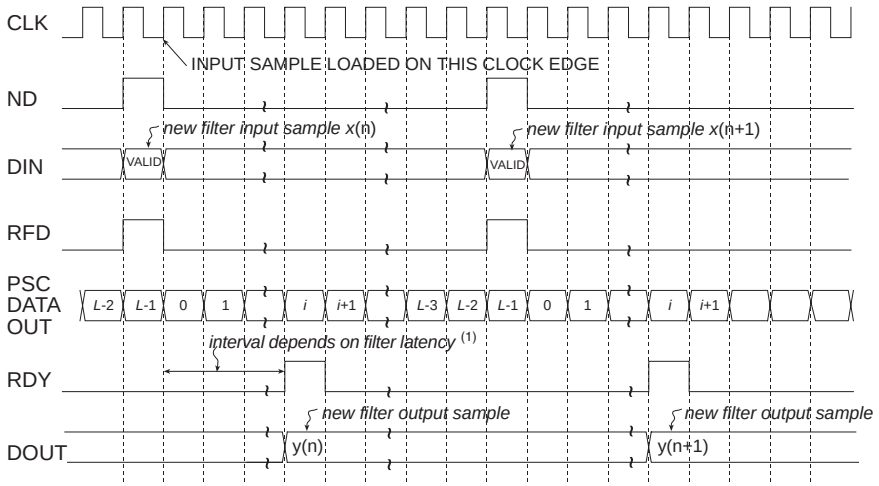
All of the filter classes employ a data-flow style interface for supplying input samples to the core and for reading the filter output port. *ND (New Data)*, *RFD (Read For Data)* and *RDY (Ready)* are used to co-ordinate I/O operations. In addition, for multi-channel filters, *SEL_I* and *SEL_O* indicate the active input and output stream respectively.

### Nomenclature

In the timing diagrams supplied in this section, the notation, $x(n)$ and $y(n)$ to denote the filter input and output samples respectively. In some diagrams, for space reasons, the variable name ($x$ or $y$) has been omitted and the diagram is annotated only with the index value $n$.

### Timing: Single-Channel and Multi-Channel Filters

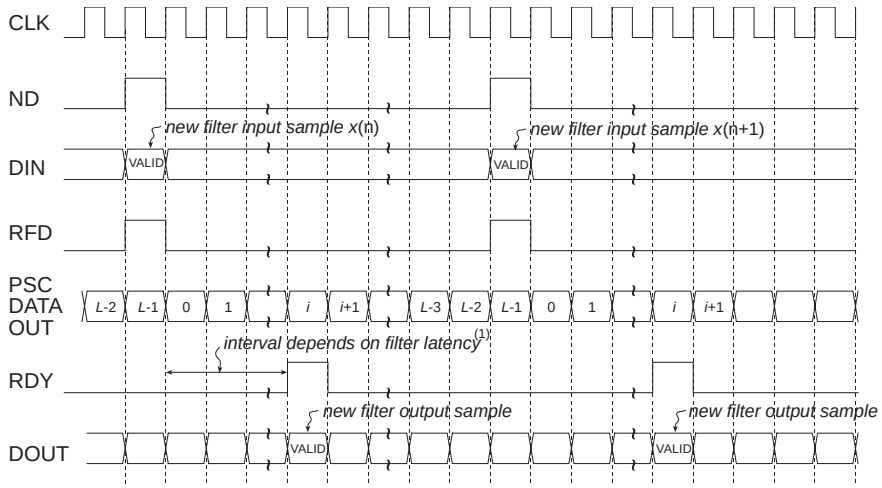
The timing for a single-channel filter, with *L* clock cycles per output sample and a registered output port, is shown in Figure 35. The *ND* input signal is used for loading a new input sample into the filter. It is effectively used internally as a clock enable, and the actual sample load operation occurs on the rising of the clock (*CLK*). When the core is ready to accept a new input sample, the *RFD* signal is asserted. When a new output sample is available, *RDY* is asserted for a single clock period. When the registered output option is selected, the output sample remains valid between successive assertions of *RDY*.




1. The latency is reported on the filter GUI

*Figure 35:* **Single-Channel FIR fIlter Timing. *L* Clock Cycles Per Output Sample, Registered Output**

Figure 36 shows the timing for a single-channel filter with an unregistered output port. The input timing is the same as for the registered output example, but now the filter result is valid for only a single clock period and is framed by *RDY*.

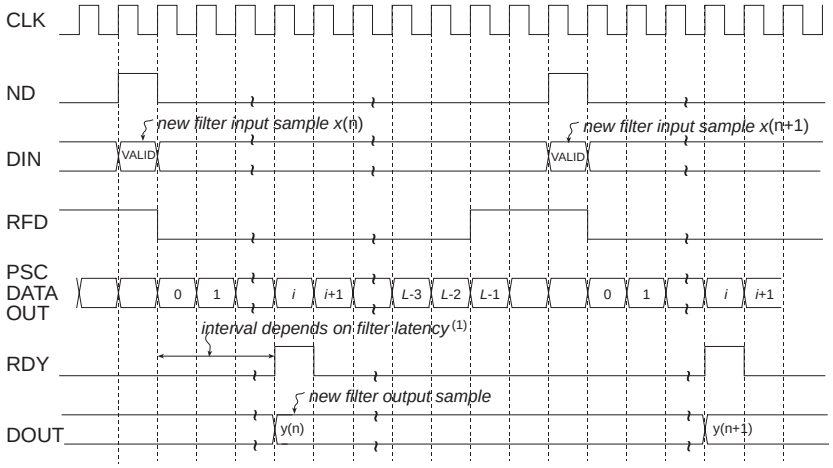



1.The latency is reported on the filter GUI

*Figure 36:* **Single-Channel FIR Filter Timing. *L* Clock Cycles Per Output Sample, Unregistered Output**

In the two previous examples, the host system supplied input samples at the highest frequency possible (every *L* clock tick). This does not have to be the case. Data samples can be supplied at a lower rate without disturbing the operation of the filter, as shown in Figure 37.

In this example, despite the filter being designed to specify *L* clock cycles per output sample, new data is supplied to the filter every *L*+2 clock periods. Observe that *RFD* is still asserted on the *L*th clock cycle of a data sample epoch, but the host system supplies a new input sample only two clock cycles later. *RFD* remains active until the new input sample has been accepted by the filter core. This occurs synchronously with the positive going edge of the clock and with *ND* acting as an active High clock enable.




1.The latency is reported on the filter GUI

*Figure 37:* **Single-Channel FIR Filter Timing. *L* Clock Cycles Per Output Sample, Registered Output. Input Samples Supplied Every *L*+2 Clock Periods.**

As a specific example of the filter interface timing, consider a non symmetric single-channel FIR filter with 10-bit precision input samples and a full serial realization ($L$=10). The timing diagram is shown in Figure 38. Ten clock cycles are needed to process each new input sample.

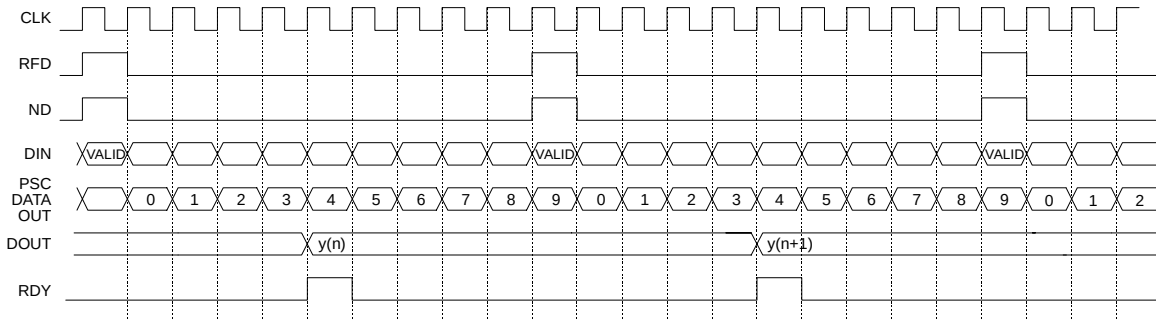



*Figure 38:* **Single-Channel FIR Filter Timing. Full Serial Implementation, 10-bit Input Samples, Registered Output. For *L*=10, there are 10 Clock Periods Between Successive Output Samples.**

A symmetrical filter with $B$-bit precision input samples requires, in general, $B$+1 clock periods for a full serial (SDA) implementation. Figure 39 shows the timing for a single-channel symmetrical FIR employing 10-bit input samples. In this case, eleven clock cycles ($L$=11) are required to process each new piece of data.

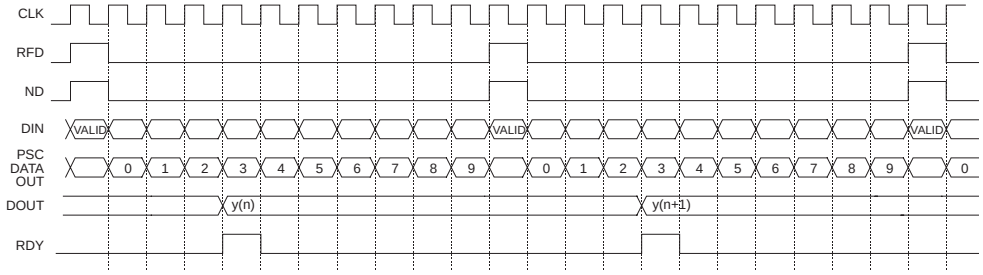



*Figure 39:* **Single-Channel FIR Filter Timing. Full Serial Implementation, 10-bit Input Samples, Symmetrical Impulse Response, Registered Output. 11 Clock Periods are Required to Process Each New Input Sample.**

The previous two figures illustrate the timing for full serial or SDA filter implementations with symmetrical and non symmetrical coefficient data. The Core Generator filter core supports various types of parallel filter realizations. The greater the degree of filter parallelism employed, the higher the filter sample rate. Filter parallelism is specified in terms of the number of clock cycles ($L$) required to compute an output sample. This value is accessed via the filter core GUI when the *Multi clock cycles per output sample* is selected in the *Implementation Option* field. The associated drop-down menu indicates valid options for $L$. The valid options for $L$ depend on the filter parameters, symmetrical/non symmetrical coefficient data and precision of the input samples. For example, for an input sample precision $B$=10 and using a non symmetrical impulse response, the valid values for $L$ are {1, 2, 3, 4, 5, 10}. For $B$=10 and a symmetrical impulse response $L$={1, 2, 3, 4, 6, 11}.

Figure 40, Figure 41, and Figure 42 illustrate the timing diagrams for a filter with $B$=10 bit precision input samples, with $L$=2, 4, and, 6, respectively.

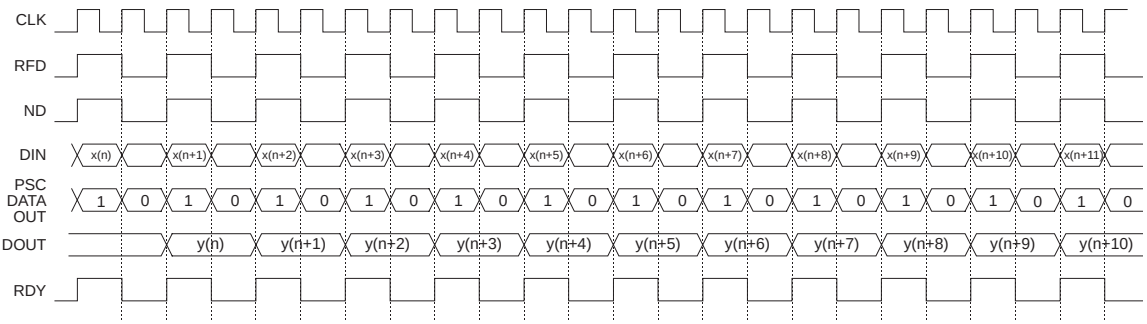



*Figure 40:* **Single-Channel FIR Filter Timing. PDA FIR With *B*=10-Bit Input Samples, *L*=2 Clock Cycles Per Output Sample, Registered Output. There are Two Clock Periods Between Successive Output Samples.**

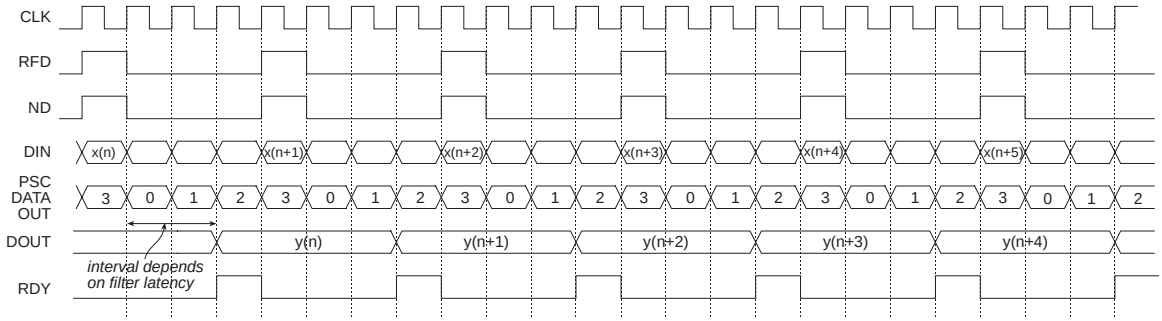



*Figure 41:* **Single-Channel FIR Filter Timing. PDA FIR With *B*=10-Bit Input Samples, *L*=4 Clock Cycles Per Output Sample, Registered Output. There are Four Clock Periods Between Successive Output Samples.**

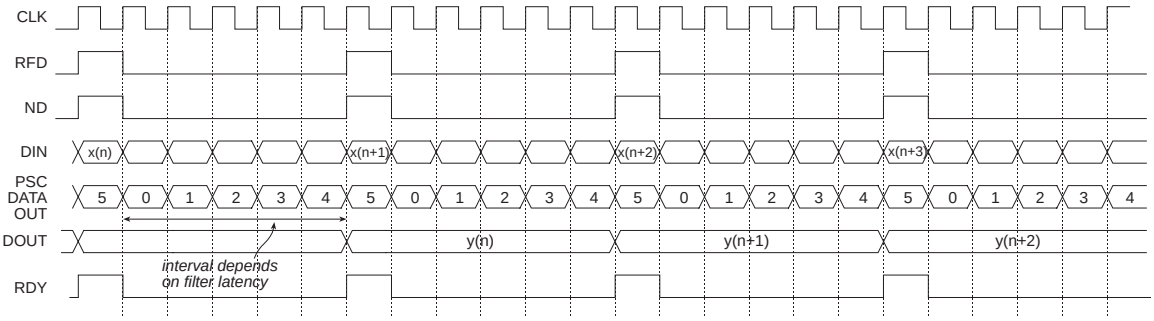



*Figure 42:* **Single-Channel FIR Filter Timing. PDA FIR With *B*=10-Bit Input Samples, *L*=6 Clock Cycles Per Output Sample, Registered Output. There are Six Clock Periods Between Successive Output Samples.**

Figure 43 illustrates the filter timing for a fully parallel DA (PDA) FIR filter. Observe that after the initial start-up latency a new output sample is available on every clock edge. The number of clock cycles in the start-up latency period is a function of the filter parameters. This value is reported in the filter design GUI in addition to the associated .vho (or .veo, refer to "XCO File Parameters" on page 26) file.

The figure shows *ND* valid on every clock edge, so a new input sample is delivered to the filter on each clock edge. Of course, *ND* may be removed for an arbitrary number of clock cycles in order to temporarily suspend the filter operation. No internal state information is lost when this is done, and the filter resumes normal operation when *ND* is reapplied (placed in the active again).

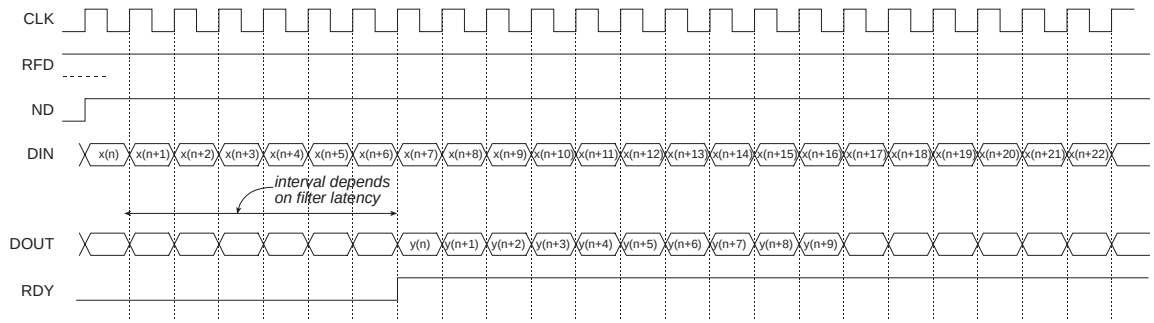



*Figure 43:* **Fully Parallel Implementation. Single-Channel Filter. With a Fully Parallel Implementation, a New Output Sample is Available on Each Clock Edge (After the Start-Up Latency), Independent of the Filter Length or the Bit Precision of the Input Data Samples.**

Figure 44 and Figure 45 demonstrate the timing for a multi-channel filter. Multi-channel filters provide two additional output ports, *SEL_I* and *SEL_O*, that indicate the active input and output channel respectively. Figure 44 illustrates a filter with an unregistered output. Figure 45 shows the timing for registered output samples.

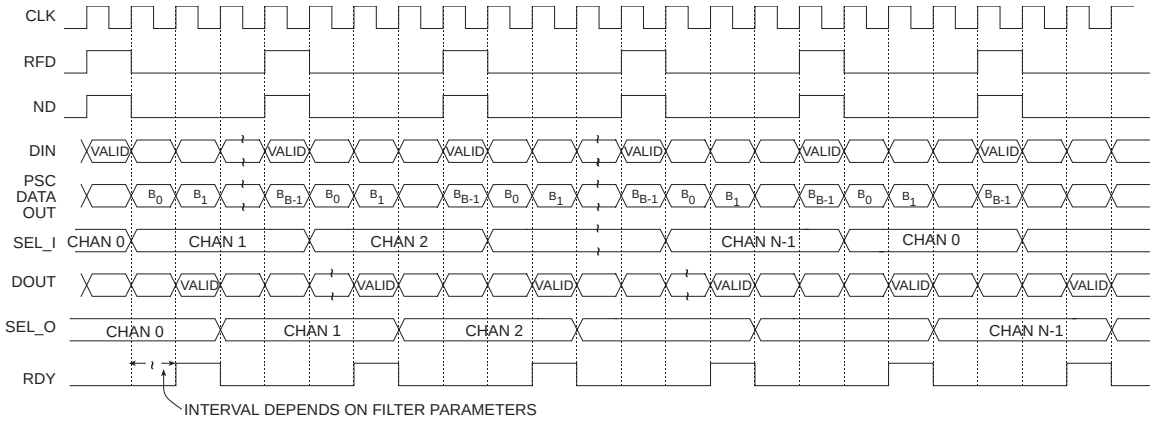



*Figure 44:* **Multi-Channel FIR Filter Timing. Nonsymmetrical Impulse Response, *B*-Bit Input Samples, Unregistered Output.**

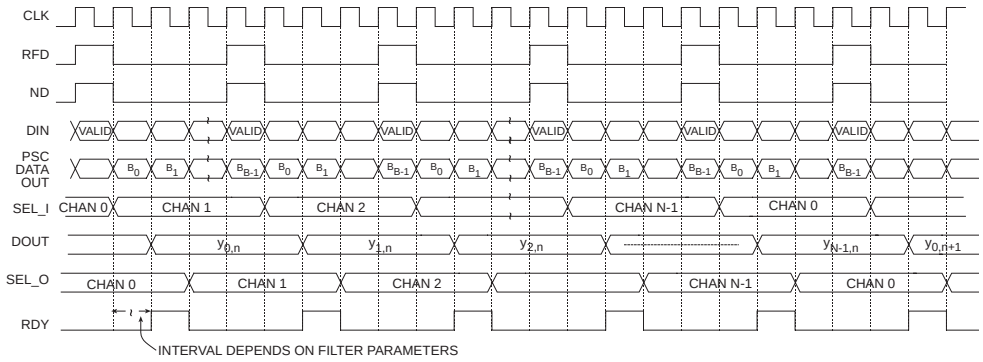



*Figure 45:* **Multi-Channel FIR Filter Timing. Nonsymmetrical Impulse Response, *B*-Bit Input Samples, Registered Output.**

## Polyphase Decimator Timing

Figure 46 demonstrates the timing for a polyphase decimator with $M = 4, B = 8$ and 8 clock cycles per output point (*Clock Cycles/Output Sample=8*). Remember, for all of the multirate filter structures, the number of clock cycles per output point specification (*Clock Cycles/Output Sample*) refers to the individual filter segments that comprise the filter, and is not directly associated with the filter output port *DOUT*. Observe that in this case, the filter is always able to accept input samples, as indicated by *RFD*=1. New output samples become available after *M*, in this case 4, input samples have been delivered to the filter. New output samples are produced in response to each new block of 4 input values. Delivering the final value in each *M*-tuple causes a new inner product calculation to commence. The resulting output sample becomes available a number of clock cycles *k* after the final sample in the *M*-tuple is delivered. The exact value of *k* is a function of the filter parameterization. It is tightly coupled to the input sample bit precision, the value specified for the *Clock Cycles/Output Sample* parameter, in addition to the number of internal pipeline stages and the data buffering depth in the filter. It is always recommended to use the output control signal *RDY* to coordinate all processes that are data sinks for the filter output port *DOUT*.

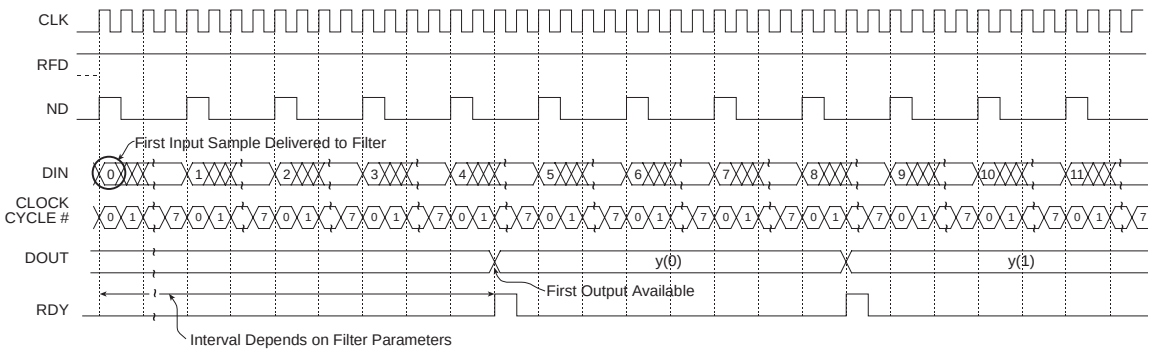



*Figure 46:* **Polyphase Decimator Timing; 8-Bit Precision Input Samples, Down-sampling Factor *M*=4, *L*=8.**

Figure 47 illustrates the timing for a 4-to-1 polyphase decimator with similar parameters to the filter considered in Figure 46, but in this case the number of *Clock Cycles/Output Sample* is *L*=4. Observe that even though the input sample precision (*B*=8) is the same as in the filter demonstrated in Figure 46, samples can be presented to filter every 4 clock cycles, in contrast to every 8 clock periods in the previ-

ous example. The filter supports double the input sample rate and, therefore, twice the bandwidth, of the filter with *L*=8.

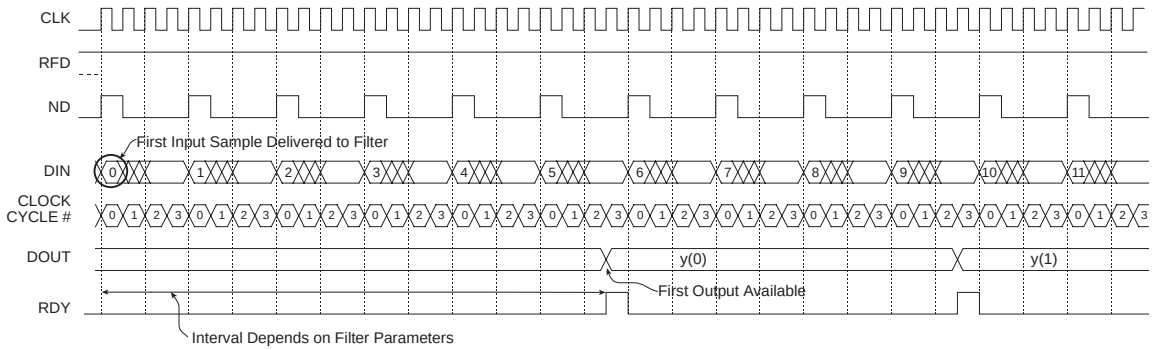



*Figure 47:* **Polyphase Decimator Timing. 8-Bit Precision Input Samples, Down-sampling Factor *M*=4, *L*=4.**

## Polyphase Decimator: Burst Input Mode

Internal buffering in the polyphase decimator allows the user to burst samples into the *DIN* port. This is illustrated in Figure 48 for a down-sampling factor *M*=4, 12-bit input samples, and *L*=12. This figure shows the timing for the filter starting from rest; that is, no data has been previously applied to the input port. Notice in this case that a total of 8 samples may be written to the filter before the device removes *RFD*.

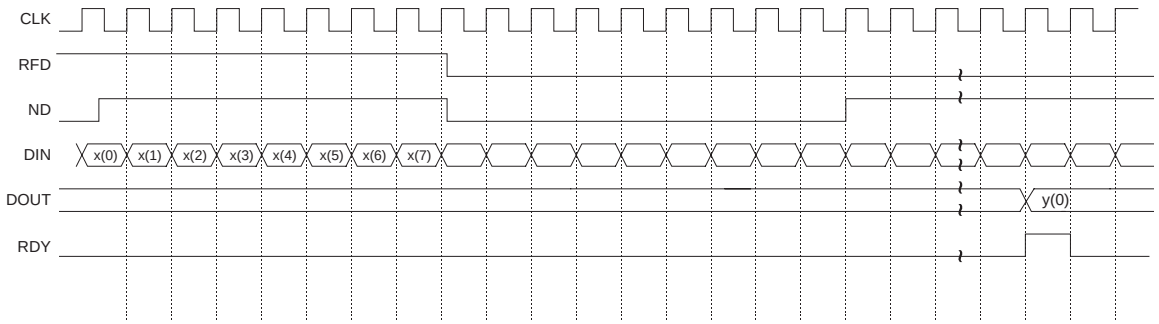



*Figure 48:* **Polyphase Decimator Timing. 12-bit Precision Input Samples, Down-sampling Factor *M*=4, *L*=12. Burst Input Data Operation. Diagram Shows the Timing When the Filter is Started from Rest; that is, No Data Has Previously Been Applied to the Input Port.**

Once the filter has moved out of this start-up state, input samples must obey the timing diagram shown in Figure 49. Only four samples can be supplied in each data burst:

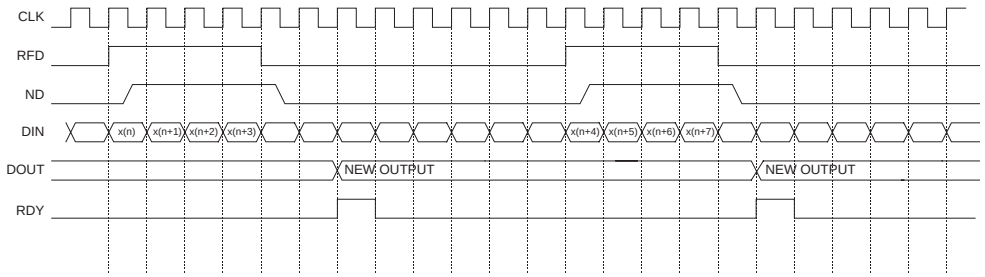



*Figure 49:* **Polyphase Decimator Timing. 12-Bit Precision Input Samples, Down-sampling Factor *M*=4, *L*=12. Burst Input Data Operation. Diagram Shows Timing After the Filter Has Moved Out of the Start-up Timing Shown in Figure 48.**

As with the *Clock Cycles/Output Sample* parameter for the single-rate filters, this parameter can be used with all the multirate filters to tradeoff performance with silicon area.

## Polyphase Interpolator Timing

Figure 50 shows the timing for a polyphase interpolator that supports a sample rate change of *P*=4, eight-bit precision input samples (B=8) and 8 clock-cycles-per-output-point. Again, as with the polyphase decimator, the number of clock cycles specified per output point is associated with the individual subfilters in the polyphase structure. In this example, each subfilter produces a new output sample every 8 clock cycles. The 4 polyphase segments are actually operating concurrently so, in fact, internal to the filter, 4 new output samples are available every 8 clock cycles. When the new block of output samples is available, the samples are sequenced to the filter output port *DOUT* using an internal multiplexor. The multiplexor select signal is referenced to the filter master clock signal *CLK*. As shown in Figure 50, the vector of *P* output samples is validated by the core output control signal *RDY*.

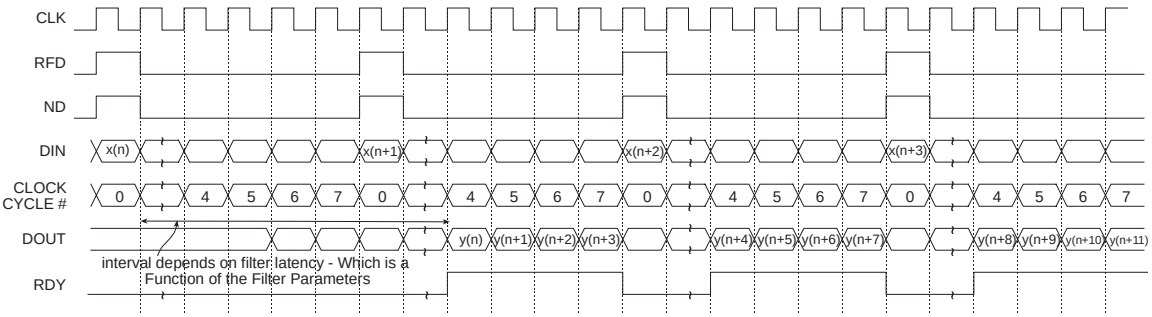



*Figure 50:* **Polyphase Interpolator Timing. 8-Bit Precision Input Samples, Up-sampling Factor *P*=4, *L*=8.**

Figure 51 shows the timing for an interpolator with similar parameters to the example demonstrated in Figure 50, but in this case a value of *L*=4 has been used. This means that each polyphase segment produces a new output sample every 4 clock cycles. In addition, all 4 outputs become available (internally)

in parallel. Observe that after the initial startup latency a new interpolant is available at the filter output port *DOUT* on each successive rising edge of the clock.

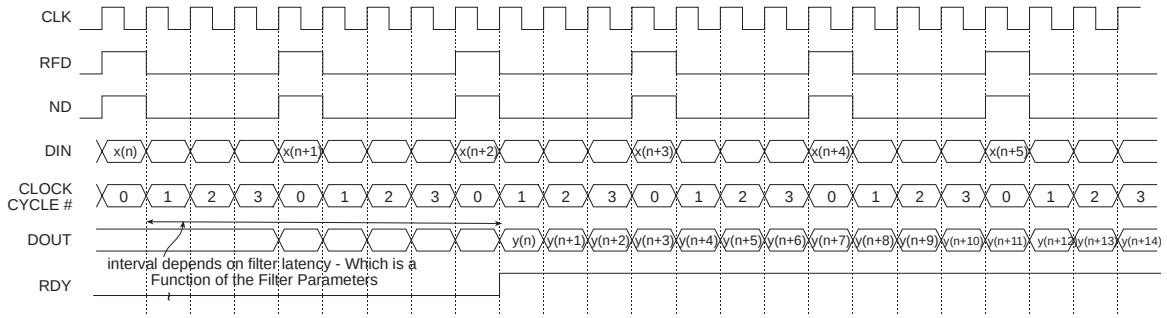



*Figure 51:* **Polyphase Interpolator Timing. 8-Bit Precision Input Samples, Up-sampling Factor *P*=4, *L*=4.**

## Filter Coefficient Data

The filter coefficients are supplied to the filter compiler using a coefficient file with a *.coe* extension. This is an ASCII text file with a single-line header that defines the radix of the number representation used for the coefficient data, followed by the coefficient values themselves. This is shown in Figure 52 for an *N*-tap filter.

```
radix=coefficient_radix;
coefdata=
a(0),
a(1),
a(2),
….
a(N-1);
```

*Figure 52:* **Filter Coefficient File Format**

The filter coefficients must be supplied as integers in either base-10, base-16 or base-2 representation. This corresponds to *coefficient_radix*=10, *coefficient_radix*=16 and *coefficient_radix*=2 respectively.

The coefficient values may also be placed on a single line as shown in Figure 53.

```
radix=coefficient_radix;
coefdata=a(0),a(1),a(2),….,a(N-1);
```

*Figure 53:* **Filter Coefficient File Format—Coefficient Data on a Single Line**

The coefficient file format for each of the filter classes supported by the core are discussed below.

### FIR

The coefficient file for the single-rate FIR filter is straightforward and consists of a one-line header followed by the filter coefficient data. For example, the filter coefficient file for an 8-tap filter using a base-10 representation for the coefficient values is shown in Figure 54:

```
radix=10;
coefdata=20,-256,200,255,255,200,-256,20;
```

*Figure 54:* **Filter Coefficient File—8-Tap Filter, Base-10 Coefficient Values**

Irrespective of the filter possessing positive or negative symmetry, the coefficient file should contain the complete set of coefficient values. The filter coefficient file for the non symmetric impulse response shown in Figure 55 is presented in Figure 56.

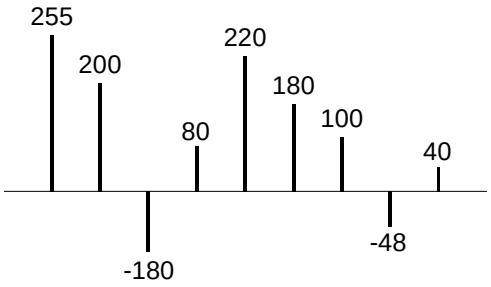



*Figure 55:* **Nonsymmetric Impulse Response**

```
radix=10;
coefdata=255,200,-180,80,220,180,100,-48,40;
```

*Figure 56:* **Coefficient File for the Non symmetric Impulse Response in Figure 55**

The coefficient file for the negative-symmetric filter characterized by the impulse response in Figure 57 is shown in Figure 58.

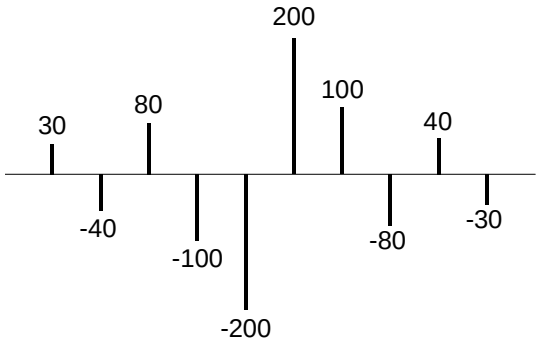



*Figure 57:* **Symmetric Impulse Response**

```
radix=10;
coefdata=30,-40,80,-100,-200,200,100,-80,40,-30;
```

*Figure 58:* **Coefficient File for the Symmetric Impulse Response in Figure 57**

## Half-Band Filter

As described in a previous section, every second filter coefficient for a half-band filter with an odd number of terms is zero. When specifying the filter coefficient data for this filter class, the zero value entries must be included in the coefficient file. For example, the filter coefficient file that specifies the filter impulse response in Figure 59 is shown in Figure 60.

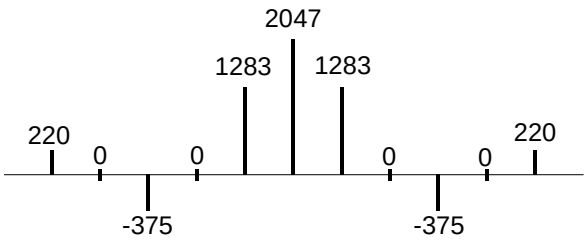



*Figure 59:* **11-Tap Half-band Filter Impulse Response**

```
radix=10;
coefdata=220,0,-375,0,1283,2047,1283,0,-375,0,220;
```

*Figure 60:* **Coefficient File for the Half-band Filter Impulse Response Shown in Figure 59**

The filter coefficient set is parsed by the filter compiler. If either the alternating zero entries are absent or the coefficient set is not even-symmetric, this is flagged as an error and the filter is not generated. A dialog box is presented to indicate the nature of the problem under these circumstances.

Technically, the zero-valued entries for a half-band filter can occur at the filter impulse response extremities as shown in Figure 61. However, observe that these values do not contribute to the result.

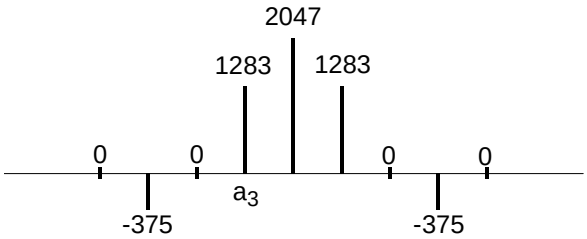



*Figure 61:* **9-Tap Half-band Filter Impulse Response**

This condition is detected when the filter is specified. If the number of taps is such that the zero-valued coefficients form the first and last entry of the impulse response, the filter length is reported as an invalid value. The number of taps N for a half-band filter must obey N = 3 + 4n, where n=0,1,2,3,…. For example, a half-band filter may have 11,15,19 and 23 taps, but not 9, 13, 17 or 21 taps.

## Hilbert Transform

The impulse response for a 10-term approximation to a Hilbert transformer is shown in Figure 62. The odd-symmetry and zero-valued coefficients are both exploited to generate an efficient FPGA realization. The coefficient data file for the Hilbert transform must contain the zero-valued entries. For example, the .coe file corresponding to Figure 62 is shown in Figure 63.

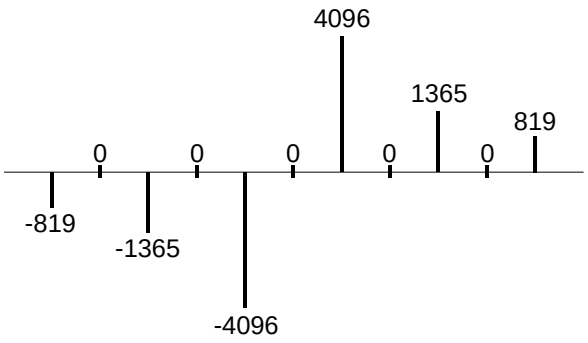



*Figure 62:* **Hilbert Transform - Impulse Response**

```
radix=10;
coefdata=-819,0,-1365,0,-4096,0,4096,0,1365,0,819;
```

*Figure 63:* **Coefficient File for the Hilbert Transformer with the Impulse Response Shown in Figure 62**

In practice, some optimization methods used for designing a Hilbert transform may lead to the presence of small even-numbered coefficients. If the *Hilbert Transform* filter class is used in the filter compiler, these terms must be forced to zero by the user.

Just like the half-band filter, the zero-valued entries for a Hilbert transformer can occur at the filter impulse response extremities. However, these values do not contribute to the result.

This condition is detected when the filter is specified. If the number of taps is such that the zero-valued coefficients form the first and last entry of the impulse response, the filter length is reported as an invalid value. The number of taps N for a Hilbert transformer must obey N = 3 + 4n, where n=0,1,2,3,…. For example, a Hilbert transform filter may have 11,15,19 and 23 taps, but not 9, 13, 17 or 21 taps.

## Interpolated Filter

A previous section explained that an IFIR filter is similar to a conventional FIR, but with the unit delay operator replaced by *k*-1 units of delay. *k* is referred to as the *zero-packing factor.* One way to realize this substitution is by the insertion of *k*-1 zeros between the coefficient values of a prototype filter. When specifying an IFIR architecture, the full set of prototype coefficients are supplied in the coefficient file, without the zeros implied by the zero-packing factor. The zero-packing factor is defined through the filter user interface. For example, consider the filter coefficient data in the .coe file shown in Figure 64.

```
radix=10;
coefdata=-200,1200,2047,1200,-200;
```

*Figure 64:* **Prototype Coefficient Data for IFIR Example**

If a zero-packing factor of *k*=2 is specified, the equivalent filter impulse response is shown in Figure 65.

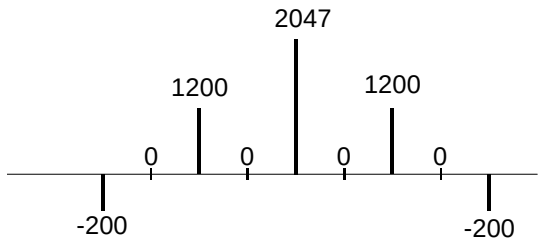



*Figure 65:* **Equivalent IFIR Impulse Response for the Coefficient Data Shown in Figure 64 with a Zero-Packing Factor *k*=2**

If the zero-packing factor is changed to *k*=3, the impulse response is as shown in Figure 66.

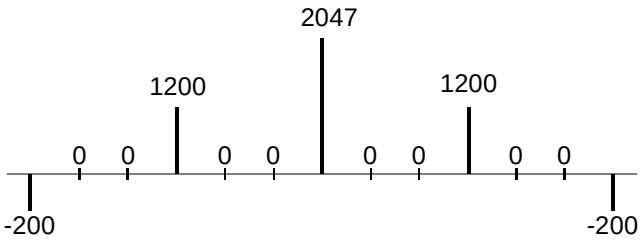



*Figure 66:* **Equivalent IFIR Impulse Response for the Coefficient Data Shown in Figure 64 with a Zero-Packing Factor *k*=3**

These examples have utilized a symmetrical prototype impulse response, this is not a restriction of the filter core. The prototype filter coefficient set can be symmetrical, nonsymmetrical, or negative-symmetric.

## Core Resource Utilization

The logic utilization for a filter is a function of the filter length, coefficient precision, coefficient symmetry, and input data precision. Table 4 through Table 9 provide logic resource requirements for a number of filter configurations.

*Table 4:* **Virtex Logic Slice Utilization for Several FIR Filter Configurations: 10-Bit Filter Coefficients; Filter Coefficient Optimization Off; Single-Channel; Signed Input; Signed Coefficients; Unregistered Output**

| Filter Length | Symmetry | Input Sample Precision | | | | |
|---|---|---|---|---|---|---|
| | | 4-bit | 8-bit | 12-bit | 16-bit | 32-bit |
| 4 | Symmetrical | 31 | 34 | 41 | 43 | 66 |
| | Nonsymmetrical | 29 | 33 | 36 | 43 | 67 |
| 8 | Symmetrical | 36 | 38 | 44 | 49 | 72 |
| | Nonsymmetrical | 45 | 50 | 53 | 60 | 82 |
| 32 | Symmetrical | 103 | 108 | 113 | 117 | 157 |
| | Nonsymmetrical | 141 | 146 | 151 | 154 | 196 |

*Table 4:* **Virtex Logic Slice Utilization for Several FIR Filter Configurations: 10-Bit Filter Coefficients; Filter Coefficient Optimization Off; Single-Channel; Signed Input; Signed Coefficients; Unregistered Output**

| Filter Length | Symmetry | Input Sample Precision | | | | |
|---|---|---|---|---|---|---|
| | | 4-bit | 8-bit | 12-bit | 16-bit | 32-bit |
| 80 | Symmetrical | 247 | 251 | 255 | 261 | 332 |
| | Nonsymmetrical | 363 | 369 | 373 | 376 | 454 |
| 128 | Symmetrical | 370 | 377 | 380 | 358 | 493 |
| | Nonsymmetrical | 532 | 536 | 537 | 543 | 646 |
| 256 | Symmetrical | 731 | 747 | 740 | 749 | 940 |

*Table 5:* **Virtex Logic Slice Utilization for Several Filter FIR Filter Configurations: 12-Bit Filter Coefficients; Filter Coefficient Optimization Off; Single-Channel; Signed Input; Signed Coefficients; Unregistered Output**

| Filter Length | Symmetry | Input Sample Precision | | | | |
|---|---|---|---|---|---|---|
| | | 4-bit | 8-bit | 12-bit | 16-bit | 32-bit |
| 4 | Symmetrical | 34 | 35 | 41 | 47 | 69 |
| | Nonsymmetrical | 30 | 35 | 39 | 45 | 66 |
| 8 | Symmetrical | 36 | 41 | 45 | 52 | 75 |
| | Nonsymmetrical | 50 | 53 | 56 | 62 | 87 |
| 32 | Symmetrical | 111 | 114 | 118 | 125 | 166 |
| | Nonsymmetrical | 160 | 161 | 168 | 173 | 214 |
| 80 | Symmetrical | 268 | 273 | 277 | 279 | 353 |
| | Nonsymmetrical | 408 | 414 | 413 | 424 | 498 |
| 128 | Symmetrical | 402 | 415 | 417 | 421 | 521 |
| | Nonsymmetrical | 595 | 601 | 599 | 607 | 718 |
| 256 | Symmetrical | 797 | 806 | 819 | 810 | 1003 |

*Table 6:* **Virtex Logic Slice Utilization for Several Half-band Filter Configurations. 14-Bit Filter Coefficients; Filter Coefficient Optimization Off; Single-Channel; Signed Input; Signed Coefficients; Unregistered Output**

| Filter Length | Symmetry | Input Sample Precision | | | | |
|---|---|---|---|---|---|---|
| | | 4-bit | 8-bit | 12-bit | 16-bit | 32-bit |
| 7 | Symmetrical | 38 | 42 | 47 | 53 | 77 |
| 31 | Symmetrical | 84 | 96 | 100 | 104 | 147 |
| 79 | Symmetrical | 171 | 194 | 203 | 206 | 274 |

*Table 7:* **Virtex Logic Slice Utilization for Several Hilbert Transformer Configurations: 14-Bit Filter Coefficients; Filter Coefficient Optimization Off; Single-Channel; Signed Input; Signed Coefficients; Unregistered Output**

| Filter Length | Symmetry | Input Sample Precision | | | | |
|---|---|---|---|---|---|---|
| | | 4-bit | 8-bit | 12-bit | 16-bit | 32-bit |
| 7 | Odd-Symmetric | 41 | 49 | 57 | 66 | 99 |
| 31 | Odd-Symmetric | 75 | 88 | 96 | 104 | 157 |
| 79 | Odd-Symmetric | 158 | 187 | 198 | 204 | 289 |

*Table 8:* **Virtex Logic Slice Utilization for Several Interpolated Filter Configurations: 16-Bit Filter Coefficients; Filter Coefficient Optimization Off; Single-Channel; Signed Input; Signed Coefficients; Unregistered Output. Zero Packing Factor Is 4.**

| Filter Length | Symmetry | Input Sample Precision | | | | |
|---|---|---|---|---|---|---|
| | | 4-bit | 8-bit | 12-bit | 16-bit | 32-bit |
| 8 | Symmetrical | 44 | 54 | 63 | 69 | 107 |
| | Nonsymmetrical | 56 | 66 | 71 | 84 | 122 |
| 32 | Symmetrical | 146 | 170 | 198 | 201 | 303 |
| | Nonsymmetrical | 189 | 214 | 239 | 264 | 366 |
| 80 | Symmetrical | 359 | 410 | 474 | 477 | 705 |
| | Nonsymmetrical | 488 | 550 | 609 | 668 | 897 |

*Table 9:* **Virtex Logic Slice Utilization for Several PDA FIR Filter Configurations: 12-Bit Filter Coefficients and Input Data; 60-Taps; Filter Coefficient Optimization Off; Single-Channel; Signed Input; Signed Coefficients; Unregistered Output; Nonsymmetrical Impulse Response. Filter Master Clock Frequency Is 150 MHz.**

| Number of Clock Cycles per Output Sample | Slice Count | Filter Sample Rate[1] (MHz) |
|---|---|---|
| 1 | 3072 | 150 |
| 2 | 1571 | 75 |
| 3 | 994 | 50 |
| 4 | 802 | 37.5 |
| 6 | 551 | 25 |
| 12 | 268 | 12.5 |

1.The filter sample rate is not at all dependent on the number of filter taps.

## Ordering Information

This core may be downloaded from the Xilinx IP Center for use with the Xilinx CORE Generator system v7.1i and later. The CORE Generator system is bundled with the Xilinx Foundation series software packages, at no additional charge.

To order Xilinx software, please visit the Xilinx Xpresso Cafe or contact your local Xilinx sales representative.

Information on additional Xilinx LogiCORE modules is available on the Xilinx IP Center.

## References

1. Peled and B. Liu, "A New Hardware Realization of Digital Filters," *IEEE Trans. on Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 456-462, Dec. 1974.
2. S. A. White, ``Applications of Distributed Arithmetic to Digital Signal Processing," *IEEE ASSP Magazine*, Vol. 6(3), pp. 4-19, July 1989.
3. Xilinx Inc., *Xilinx Product Guide*, Xilinx Inc., San Jose California, 1999.
4. P.P. Vaidyanathan, *Multirate Systems and Filter Banks*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.
5. M. E. Frerking, *Digital Signal Processing in Communication Systems*, Van Nostrand Reinhold, New York, 1994.
6. C. H. Dick, "Implementing Area Optimized Narrow-Band FIR Filters Using Xilinx FPGAs," *SPIE International Symposium on Voice, Video and Data Communications—Configurable Computing: Technology an Applications Stream*, Boston, Massachusetts USA, pp. 227-238, Nov 1-6, 1998. Also available at: http://www.xilinx.com/products/logicore/coredocs.htm

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 03/07/03 | 1.1 | Conversion to new template. |
| 5/21/04 | 1.2 | Updated to support Xilinx software v6.2i and Virtex-4 FPGAs. |
| 4/28/05 | 1.3 | Updated document to add support for Spartan-3E FPGA and Xilinx software v7.1i. |