Distributed Arithmetic FIR Filter V4.0.0

November 3 2000 Product Specification

Xilinx, Inc.
2100 Logic Drive
San Jose, CA 95124
Phone: +1 408-559-7778
FAX: +1 408-559-7114
URL: www.xilinx.com/ipcenter
Support: www.support.xilinx.com

1 Features

- Drop-in module for Virtex™, Virtex™-E, Spartan™-II and Virtex™-II 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
- Highly parameterizable
- 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 tradeoff FPGA logic resources for sample rate in order to generate an optimal design
- Data-flow style core interface and control
- Incorporates Xilinx Smart-IP technology for maximum performance
- To be used with version 3.1i or later of the Xilinx CORE Generator System

2 General Description

The Xilinx filter Core is a highly parameterizable, area efficient high-performance FIR filter. Several highly optimized filters can be realized with the filter 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 Eq. (1)

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

where *N* is the number of filter coefficients. The conventional tapped delay line realization of this inner-product calculation is shown in Figure 1.

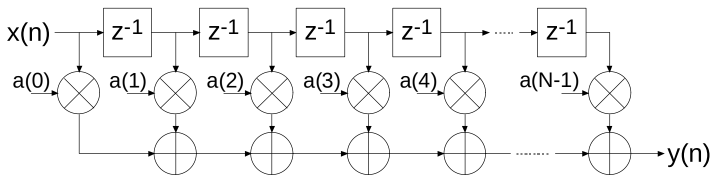


Figure 1: Conventional tapped-delay line FIR filter mechanization.

Even though 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. With this approach there are no explicit multipliers employed in the design, only look-up tables (LUTs), shift registers and a scaling accumulator.

2.1 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 TSB's are used as address inputs to a look-up table. This LUT stores all possible partial products [2] over the filter coefficient space.

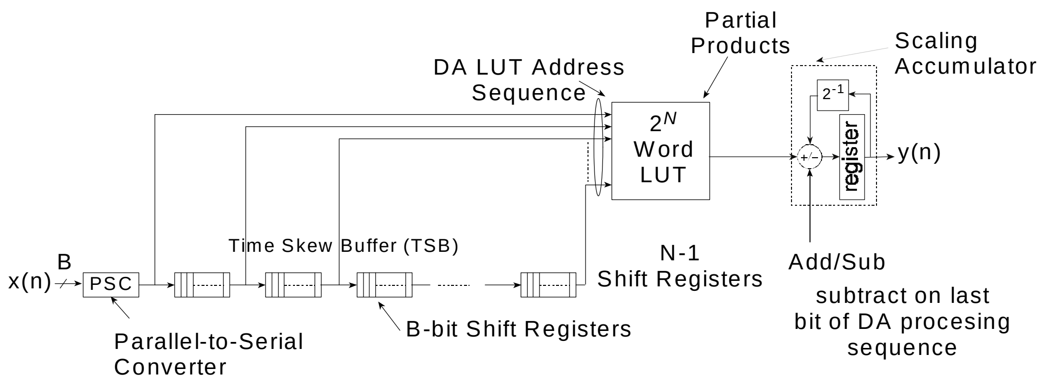


Figure 2: Serial distributed arithmetic FIR filter.

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 non-symmetrical 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 non-symmetrical 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 de-coupled 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.

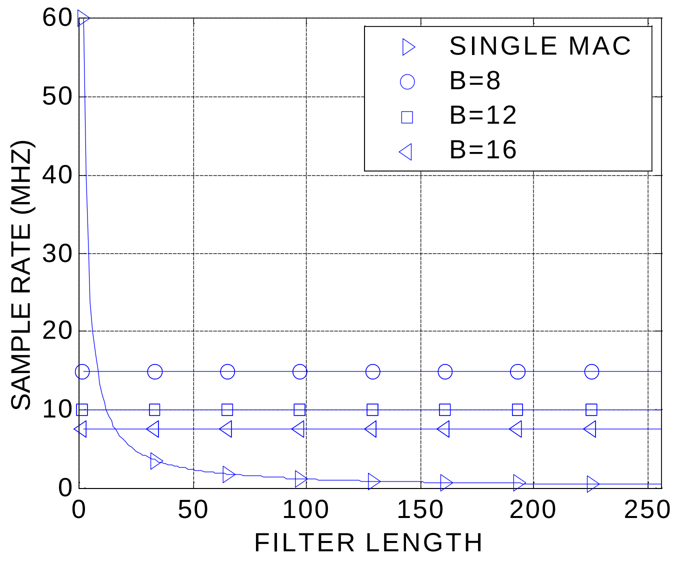


Figure 3: Throughput (sample rate) comparison of single-MAC based FIR and DA FIR as a function of filter length. B is the DA FIR input sample precision. The clock rate is 120 MHz.

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

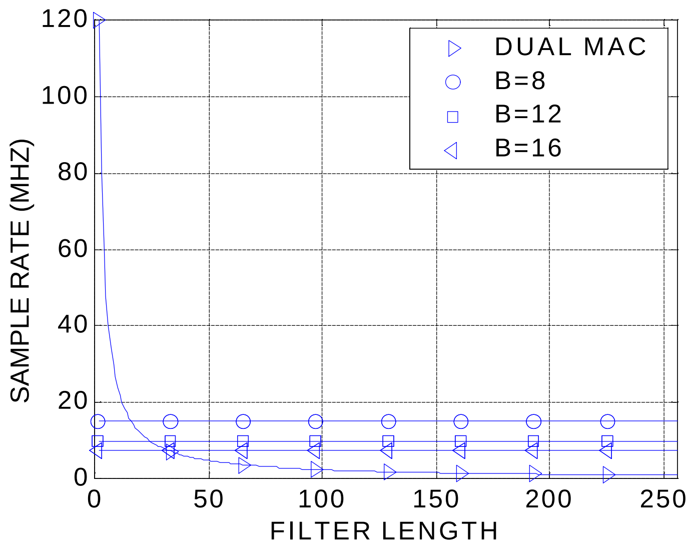


Figure 4: Throughput (sample rate) comparison of dual-MAC based FIR and DA FIR as a function of filter length. B is the DA FIR input sample precision. The clock rate is 120 MHz.

2.2 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 non-symmetrical impulse response). Additional speed may 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 non-symmetrical 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 tradeoff silicon area with filter throughput.

2.3 Exploiting Filter Symmetry

The impulse response for many filters possess 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.

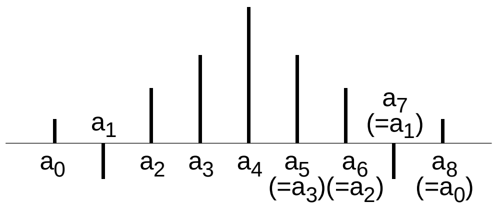


Figure 5: Symmetric FIR – odd number of terms.

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 $\lceil N/2 \rceil$ multiplications and approximately *N* additions. This significant reduction in the computation workload can be exploited to generate efficient filter hardware implementations.

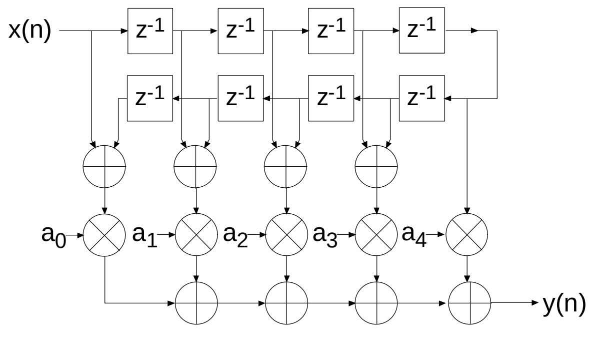


Figure 6: Exploiting coefficient symmetry – odd number of filter taps.

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

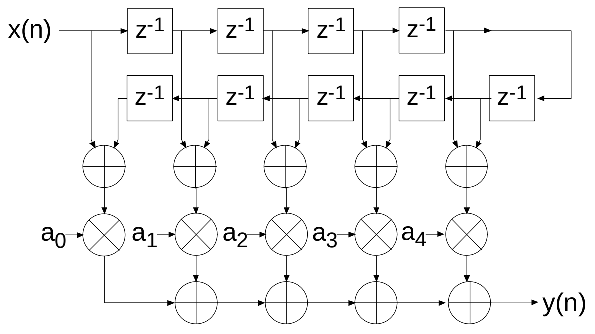


Figure 7: Exploiting coefficient symmetry – even number of filter taps.

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

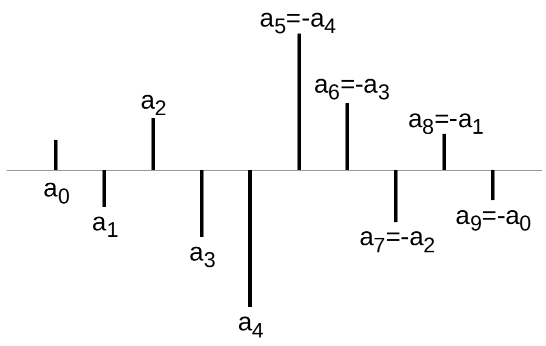


Figure 8: Negative Symmetric impulse response.

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.

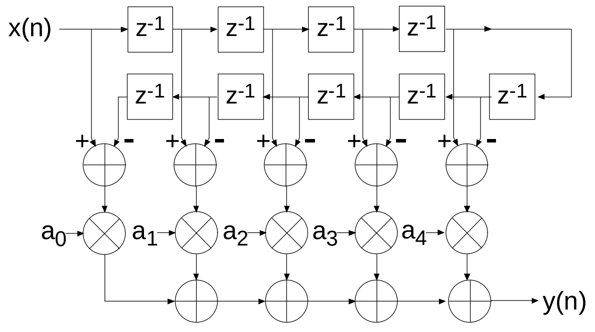


Figure 9: FIR architecture exploiting negative symmetry.

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% savings in area.

3 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 non-symmetrical 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, then 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 simply 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 non-symmetrical 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 (non-symmetrical FIR).

If the input sample precision is changed to 8 bits, with *L*=8, the filter sample rate for a non-symmetrical filter would be 100/8 = 12.5 MHz.

4 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 each processed by the same filter.

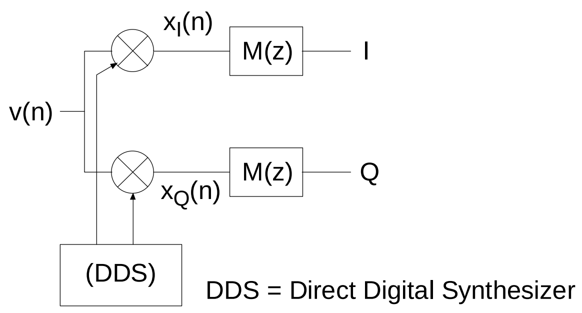


Figure 10: Digital down converter.

One candidate solution to this problem is to simply employ two separate filters. However, this 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. All of the filter classes supported by the filter core provide in-built support for multi-channel processing and can accommodate up to 8 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 will process two sample streams, each with a sample rate of $f_S/2$. A three channel version of the filter will process three data streams, and support a sample rate of $f_S/3$ for each of the streams.

A multi-channel filter implementation is very efficient in terms of the amount of logic resources utilized. A filter with two or more channels can be realized using virtually the same amount of logic resources as a single channel version of the same filter. The tradeoff that needs to be addressed when employing 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 will decrease.

The number of channels to be supported by a filter core is specified through 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.

5 Filter Configurations

Eight classes of filters are supported by the filter compiler: 1. conventional single-rate FIR, 2. half-band FIR, 3. Hilbert transform [5], 4. interpolated FIR [4] [6], 5. Polyphase decimator, 6. Polyphase interpolator, 7. half-band decimator and 8. half-band interpolator. The *interpolated* FIR should not be confused with an *interpolation* filter. Interpolated filters are single rate systems that can be employed to produce efficient realizations of narrow-band filters, and with some minor enhancements, wide-band filters can also be accommodated.

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

5.1 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 the *Interface and Control* section of the product guide.

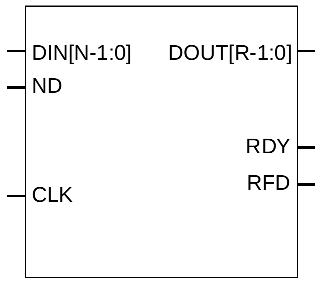


Figure 11: Single channel FIR symbol.

A *P*-channel filter core is shown in Figure 12. The output ports *SEL_I* and *SEL_O* are provided to indicate the active input and output data stream respectively. The *SEL_I* signal can be used to multiplex several input sources on to 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.

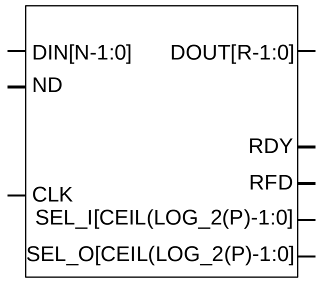


Figure 12: Multi-channel FIR symbol.

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

Table 1: FIR core signal pinout.

Signal Name	**Direction**	**Description**
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 shown in Figure 2 and an inner-product computation is started. *ND* should not be asserted while *RFD* is low. Doing so will corrupt the calculation.
DIN[N-1:0]	Input	FILTER INPUT DATA SAMPLE – N-bit wide filter input sample.
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_O[ceil(log_2(P))-1:0]	Output	OUTPUT CHANNEL SELECT This is a standard binary count generated by the core that indicates the current filter output channel number.
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.
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 simply an access point to the center of the filter memory buffer, it carries the same precision as the input sample data stream, i.e., 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.

5.2 Half-Band FIR

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

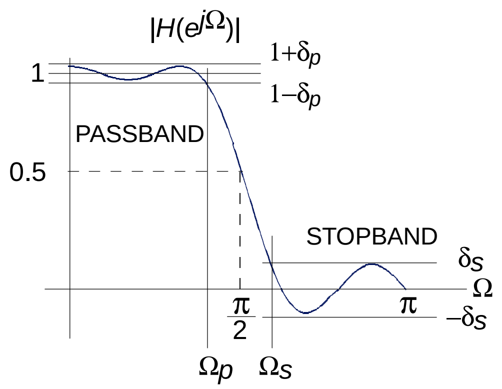


Figure 13: Half-band filter – magnitude frequency response.

Observe from the figure that 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

$$\Omega_p = p - \Omega_s$$

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

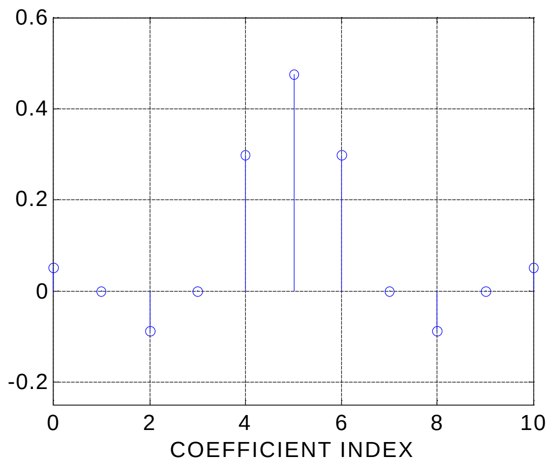


Figure 14: Half-band filter impulse response.

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

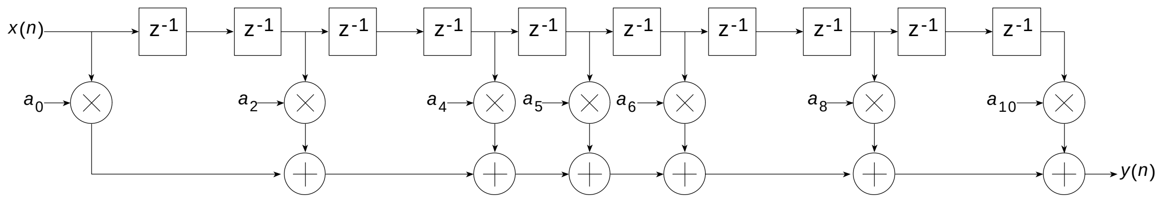


Figure 15: Half-band filter architecture.

This same structure, can of course, 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.

5.3 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 zero's as shown in Figure 16.

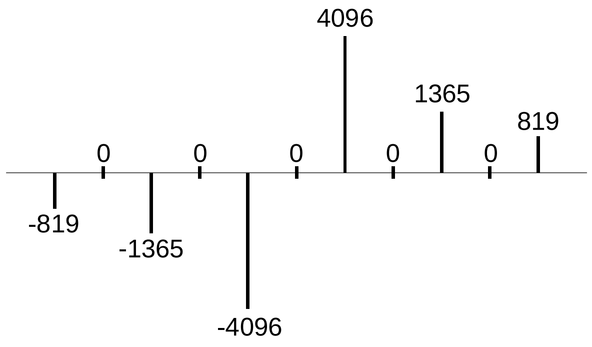


Figure 16: Impulse response of a Hilbert transformer.

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 simply 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.

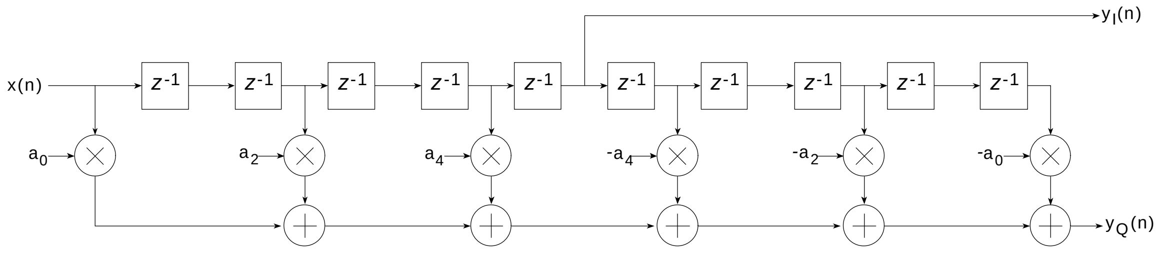


Figure 17: FIR filter realization of a Hilbert transformer.

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

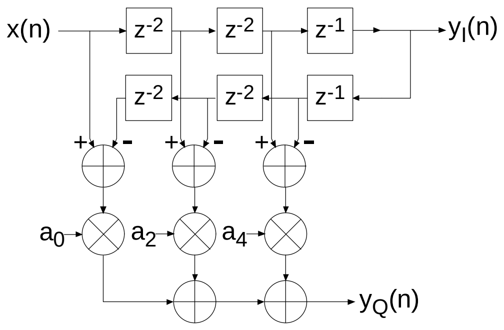


Figure 18: Hilbert transformer exploiting zero-valued filter coefficients and negative symmetry.

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

Figure 19 shows 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.

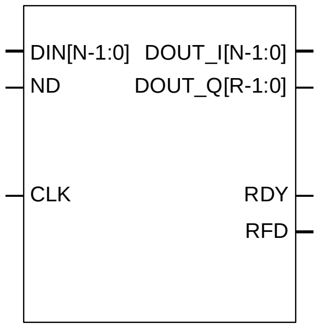


Figure 19: Hilbert transform symbol.

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.

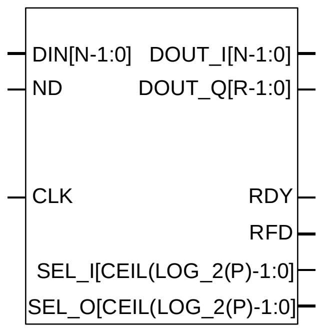


Figure 20: Multi-channel Hilbert transform core.

5.4 Interpolated FIR

An *interpolated FIR* (IFIR) [4] [6] 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.

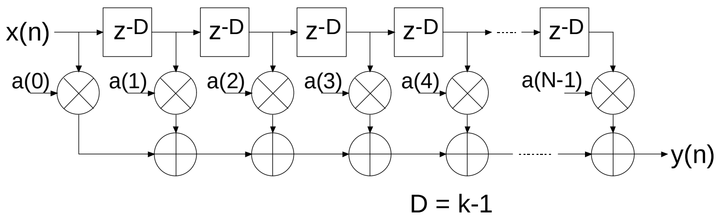


Figure 21:Interpolated FIR (IFIR). The zero-packing factor is *k.*

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.

5.5 Polyphase Decimator

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

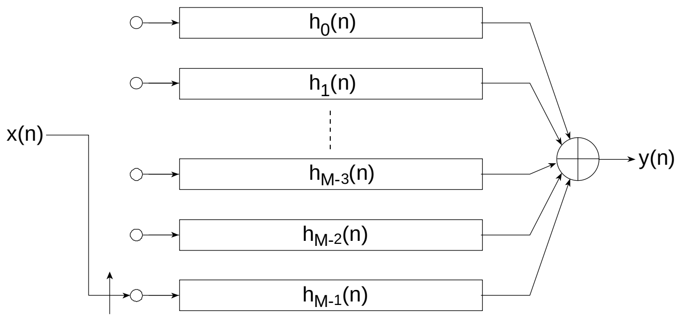


Figure 22: *M*-to-1 polyphase decimator.

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 Eq. (2).

$$h_i(n) = a(i + Mr) \quad i = 0,1,\ldots, M-1 \quad r = 0,1,\ldots, N-M+i \tag{2}$$

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 sub-filters, all operating in parallel, are employed in the filter architecture.

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

5.6 Polyphase Interpolator

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

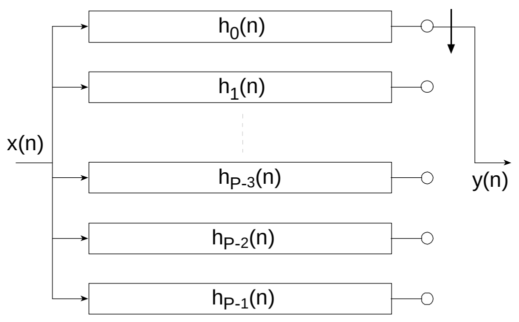


Figure 23: 1-to-*P* polyphase interpolator.

A set of *N* prototype filter coefficients $a_0, a_1, \ldots, a_{N-1}$ are mapped to the P polyphase sub-filters $h_0(n), h_1(n), \ldots, h_{P-1}(n)$ according to Eq. (3).

$$h_i(n) = a(i + \mathrm{Pr}) \quad i = 0,1,\ldots,P-1 \quad r = 0,1,\ldots,N-P+i \tag{3}$$

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

5.7 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.

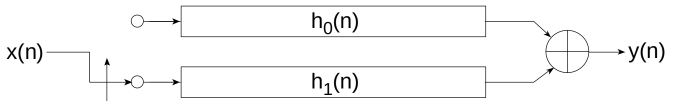


Figure 24: Half-band decimation filter.

The filter is very similar in nature to the polyphase decimator described in 5.5 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 exposes the details of 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 sub-filters 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 represented and described the half-band filter in terms of 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.

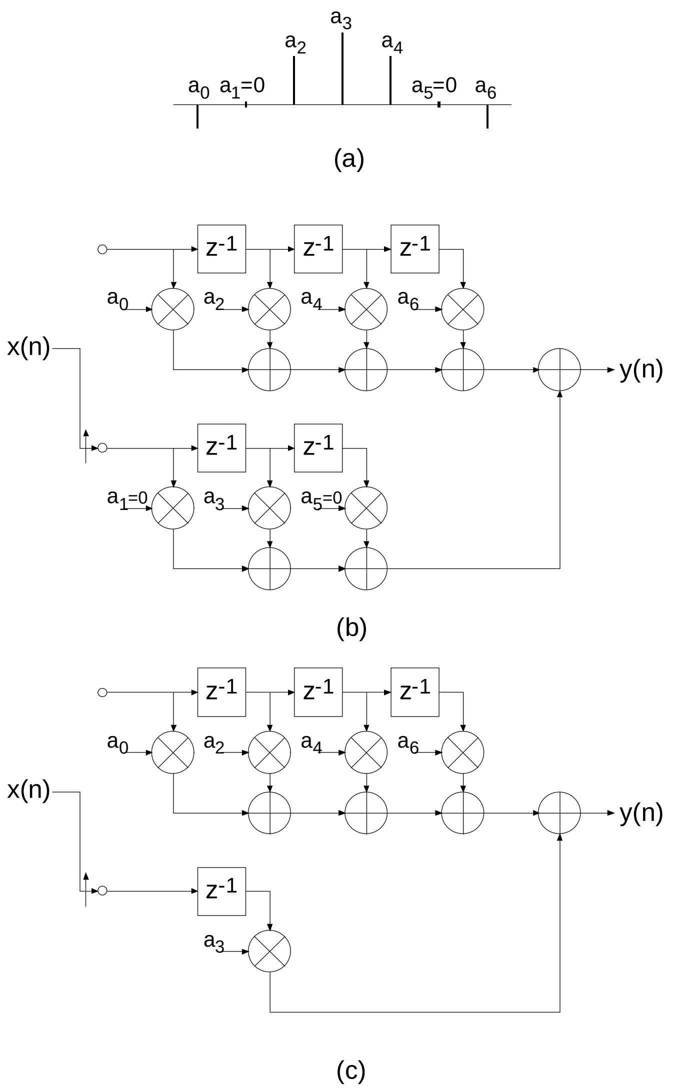


Figure 25: 7-tap half-band decimation filter. (a) Impulse response. (b) Polyphase partition. (c) Reduced complexity (hardware optimized) realization. The high density of zero-valued filter coefficients are exploited in the FPGA realization to produce a minimal area implementation.

5.8 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.

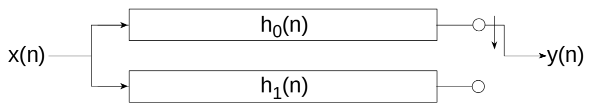


Figure 26: Half-band interpolation filter.

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 only supports single channel input data streams.

5.9 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 non-zero value terms cannot be ignored, the general polyphase decimator or interpolator described in Sections 5.5 and 5.6 respectively using a rate change of two are more appropriate.

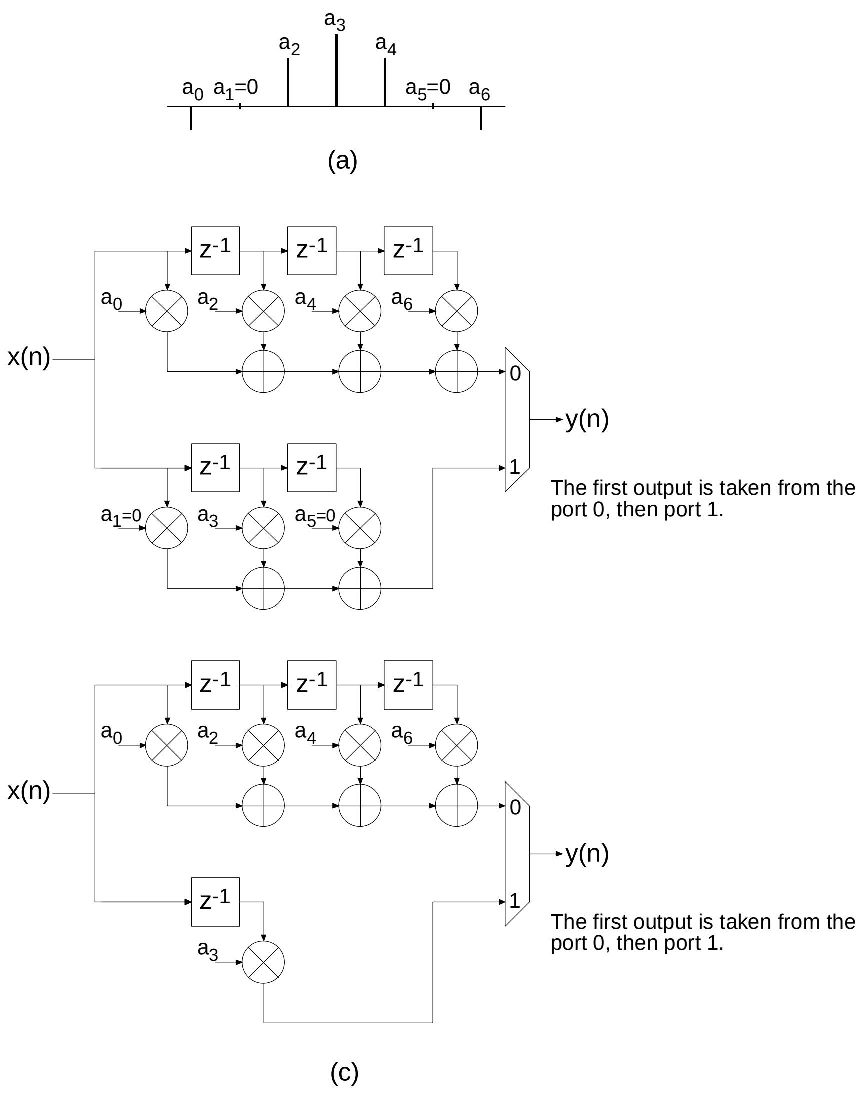
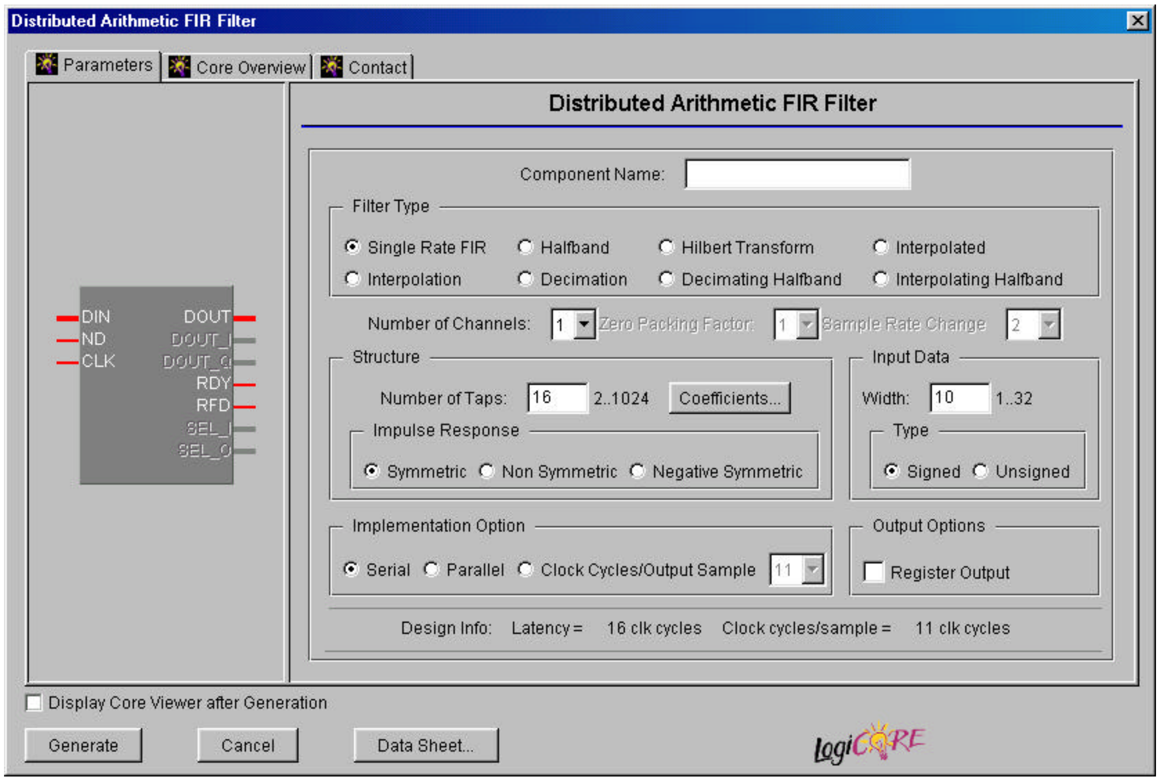

a3
a2
a4
a0
a1=0
a5=0
a6
(a)
z-1
a0
a2
a4
a6
x(n)
0
y(n)
1
a1=0
a3
a5=0
The first output is taken from the port 0, then port 1.
z-1
a0
a2
a4
a6
x(n)
0
y(n)
1
a3
The first output is taken from the port 0, then port 1.
(c)

Figure 27: 7-tap half-band interpolation filter. (a) Impulse response. (b) Polyphase partition. (c) Reduced complexity (hardware optimized) realization. The high density of zero-valued filter coefficients are exploited in the FPGA realization to produce a minimal area implementation.

6 CORE Generator Parameters

A filter core is generated using the graphical user interface (GUI) shown in Figure 28.



Distributed Arithmetic FIR Filter

Parameters | Core Overview | Contact

Distributed Arithmetic FIR Filter

Component Name:

Filter Type: [✓ Single Rate FIR] Halfband, Hilbert Transform, Interpolated, Interpolation, Decimation, Decimating Halfband, Interpolating Halfband

Number of Channels: 1 Zero Packing Factor: 1 Sample Rate Change: 2

Structure: Number of Taps: 16 2..1024 Coefficients...

Impulse Response: [✓ Symmetric] Non Symmetric, Negative Symmetric

Input Data: Width: 10 1..32

Type: [✓ Signed] Unsigned

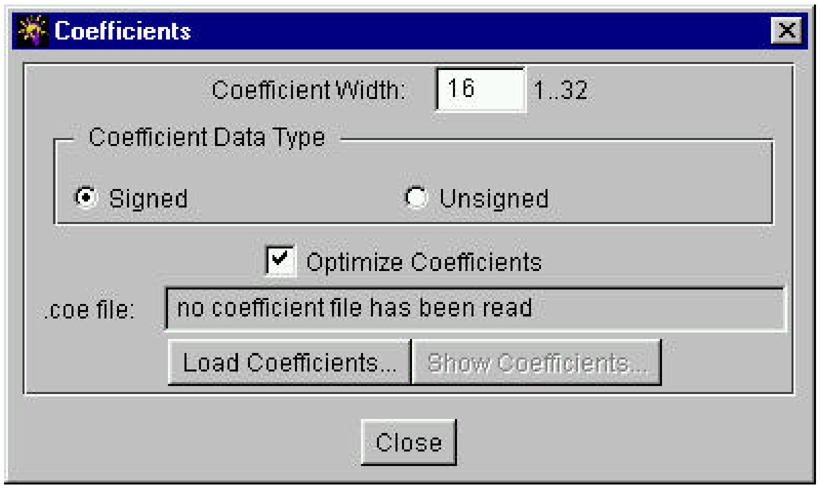
Implementation Option: [✓ Serial] Parallel, Clock Cycles/Output Sample 11

Output Options: Register Output

Design Info: Latency = 16 clk cycles Clock cycles/sample = 11 clk cycles

Display Core Viewer after Generation

Figure 28: Filter parameterization screen.



Coefficients

Coefficient Width: 16 1..32

Coefficient Data Type: [✓ Signed] Unsigned

[✓] Optimize Coefficients

.coe file: no coefficient file has been read

Figure 29: Parameterization screen – *Coefficients* panel. The coefficient parameterization screen is accessed using the *Coefficients* button on the primary GUI shown in Figure 28.

The user supplied parameters are:

- **Component Name:** The user defined filter component name.

- **Filter Type:** Eight filter types are supported 1. Single rate FIR, 2. Half-band FIR, 3. Hilbert transform, 4. Interpolated FIR, 5. Polyphase decimator, 6. Polyphase interpolator, 7. Half-band decimator, and 8. Half-band interpolator.

- **Number of Channels:** The number of channels processed by the filter. One to a maximum of 8 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 0's 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* will insert *k*-1 0's between the supplied coefficient values.

- **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.

- **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 non-symmetrical 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 non-symmetric impulse response.

- **Implementation Option:** Selecting the *Serial* option generates an SDA FIR filter. This is a fully serial DA FIR filter. In this case, if a non-symmetric 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 will result in a filter that generates a new output sample every 3 clock cycles. If *L*=5, new output samples are produced every 5 clock cycles.

  For all of the polyphase filters, including the half-band decimation and half-band interpolation filters, the *Clock Cycles/Output Sample* value refers to the individual filters that are employed to construct all of the multirate architectures.

- **Coefficient Width:** The bit precision of the coefficient data. This is an integer value between 1 and 32 inclusive. The **Coefficient Width** parameter is accessed using the *Coefficients* user interface (UI) shown in Figure 29. This sub-panel is enabled using the *Coefficients* button on the primary GUI.

- **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.

- **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. The **Coefficient Data Type** parameter is accessed using the *Coefficients* user UI shown in Figure 29. This sub-panel is enabled using the *Coefficients* button on the primary GUI.

- **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 detail below. Activating this button brings-up a browser window that lets the user select a coefficient file. The **Load Coefficients** button is accessed using the *Coefficients* user UI shown in Figure 29. This sub-panel is enabled using the *Coefficients* button on the primary GUI.

- **Show Coefficients:** Selecting this button on the GUI activates a dialog box that displays the filter coefficient data.

- **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 only valid when *RDY* is active. At other times the port will change on successive clock cycles.

- **Design Info:** 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 output 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 will be named *my_fir.vho*. If a Verilog flow had been selected this file would be called *my_fir.veo*.

7 XCO File Parameters

The parameters supplied via the filter GUI are captured and logged to the *.xco* file. The full name of this file is simply the *Component Name* with a *.xco* file extension. Table 2 defines the .xco file parameter names and range specifications.

Table 2: 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}
ViewlogicLibraryAlias	Pathname to Viewlogic directory	Valid path name for the user's operating system.

XilinxFamily	The FPGA target device family.	{Virtex | Spartan2}
DesignFlow	HDL flow specifier.	{VHDL | VERILOG}
FlowVendor	Design flow vendor information.	{Other | Synplicity | Exemplar | Synopsis | Foundation}
coefficient_file	File of filter coefficient values. The file format is defined in Section 9.	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 will remain 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 will result in the most compact (minimum FPGA logic resources) implementation. If this parameter is false, no logic optimization is performed.	{true | false}
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 only applicable to *interpolated filters* and controls the number of 0's 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 0's between the user coefficients and so on. For all filters other than the *interpolated* filter this	[1,…,8]

	parameter should be 1.	
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_sample	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 sub-filters (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 will be 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 ot *input_data_width+1.*
filter_type	The filter type specifier.	{ single_rate_fir | half-band | hilbert_transform | interpolated | interpolation | decimation | decimating_half-band | half-band_interpolating }
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 and is further discussed in Section 2.2.	{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}

Figure 30 is an example .xco file. The '#' characters at the start of the first five lines in the example identify in-line comments.

```
# Xilinx CORE Generator 3.1i
# Username = chrisd
# COREGenPath = c:\Xilinx\coregen
# ProjectPath = c:\chrisd\dafir
# ExpandedProjectPath = c:\chrisd\dafir
SET BusFormat = BusFormatAngleBracket
SET SimulationOutputProducts = VHDL
SET ViewlogicLibraryAlias = ""
SET XilinxFamily = Virtex
SET DesignFlow = VHDL
SET FlowVendor = Other
SELECT Distributed_Arithmetic_FIR_Filter Virtex Xilinx,_Inc. 4.0
CSET coefficient_file = D:\prabhu\dafir\test\fir_16_nonsym.coe
CSET coefficient_data_type = signed
CSET number_of_taps = 16
CSET register_output = false
CSET optimize_coefficients = true
CSET component_name = fir_16_nonsym_int_4_ccs_4
CSET zero_packing_factor = 1
CSET impulse_response = non_symmetric
CSET sample_rate_change = 4
CSET number_of_channels = 1
CSET clock_cycles_per_sample = 4
CSET filter_type = interpolation
CSET coefficient_width = 12
CSET input_data_width = 8
CSET implementation_option = clock_cycles_per_output_sample
CSET input_data_type = signed
GENERATE
```

Figure 30: Example *.xco File*.

8 Interface, Control and Timing

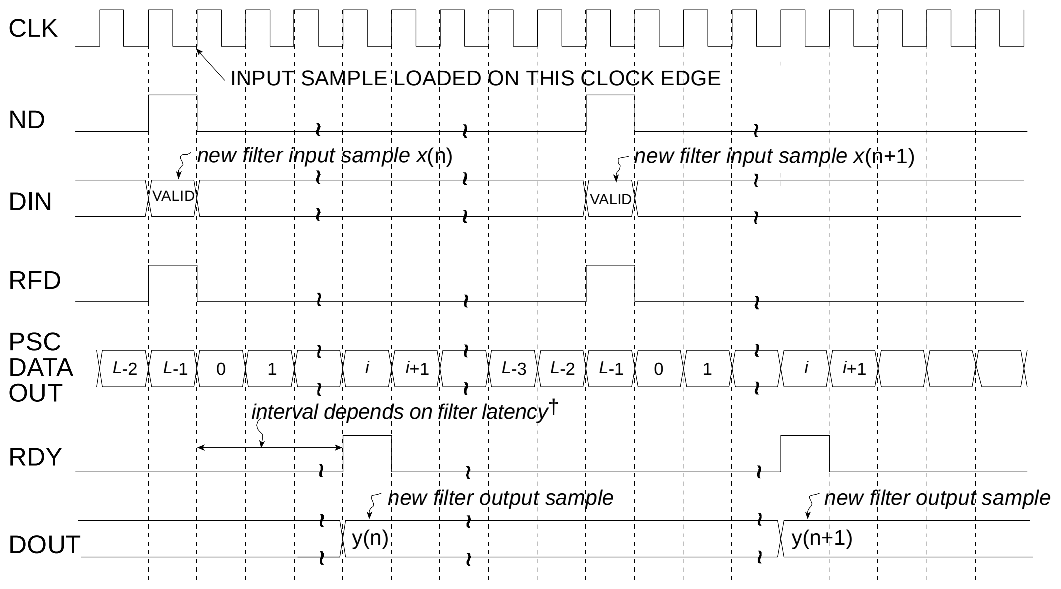
All of the filter classes employ a data-flow style interface for supplying input samples to the core and for reading the filter output port. *ND (New Data), RFD (Read For Data)* and *RDY (Ready)* are used to co-ordinate I/O operations. In addition, for multi-channel filters, *SEL_I* and *SEL_O* are supplied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

8.1 Nomenclature

In the timing diagrams supplied in this section the notation $x(n)$ and $y(n)$ are used 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 only annotated with the index value n.

8.2 Timing: Single Rate 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 31. 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 will remain valid between successive assertions of *RDY.*

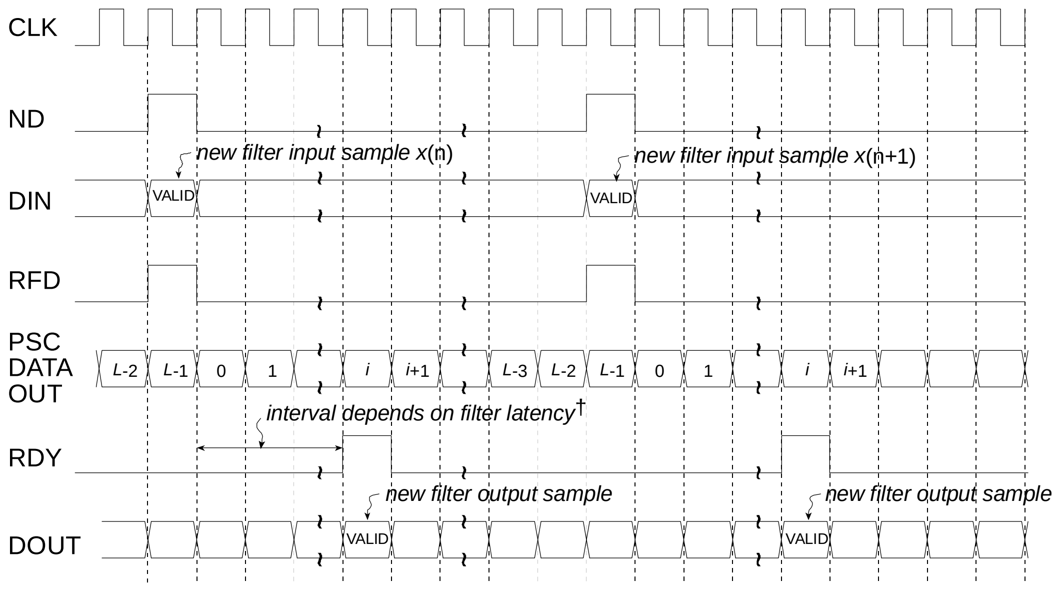

CLK
INPUT SAMPLE LOADED ON THIS CLOCK EDGE
ND
new filter input sample x(n)
new filter input sample x(n+1)
DIN
VALID
VALID
RFD
PSC DATA OUT
L-2
L-1
0
1
i
i+1
L-3
L-2
L-1
0
1
i
i+1
interval depends on filter latency†
RDY
new filter output sample
new filter output sample
DOUT
y(n)
y(n+1)

† The latency is reported on the filter GUI

Figure 31: Single channel FIR filter timing. *L* clock cycles per output sample, registered output.

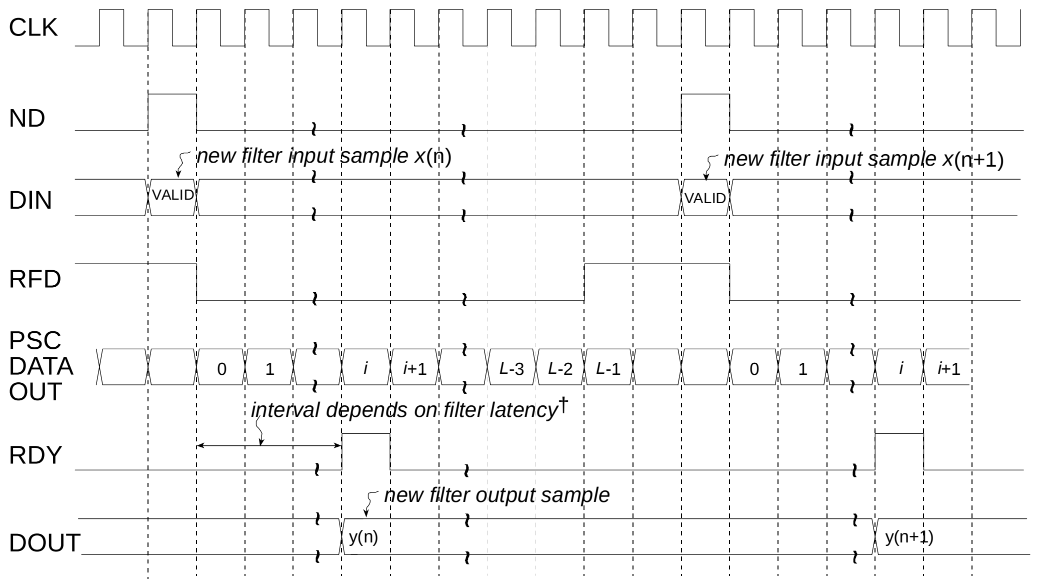
Figure 32 shows the timing for a single channel filter with an unregistered output port. The input timing is the same as for the registered output example, but now the filter result is valid for only a single clock period and is framed by *RDY*.

For the two cases described so far, the host system is supplying input samples at the highest frequency possible, that is, every *L* clock ticks. 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 33. In this example, even though the filter has been designed specifying *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 only supplies a new input sample 2 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* effectively acting as an active high clock enable.


CLK
ND
DIN
RFD
PSC DATA OUT
RDY
DOUT
new filter input sample x(n)
new filter input sample x(n+1)
VALID
L-2
L-1
0
1
i
i+1
L-3
interval depends on filter latency†
new filter output sample

† The latency is reported on the filter GUI

Figure 32: Single channel FIR filter timing. *L* clock cycles per output sample, unregistered output.


CLK
ND
DIN
RFD
PSC DATA OUT
RDY
DOUT
new filter input sample x(n)
new filter input sample x(n+1)
VALID
0
1
i
i+1
L-3
L-2
L-1
interval depends on filter latency†
new filter output sample
y(n)
y(n+1)

† The latency is reported on the filter GUI

Figure 33: 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 34. Ten clock cycles are needed to process each new input sample.

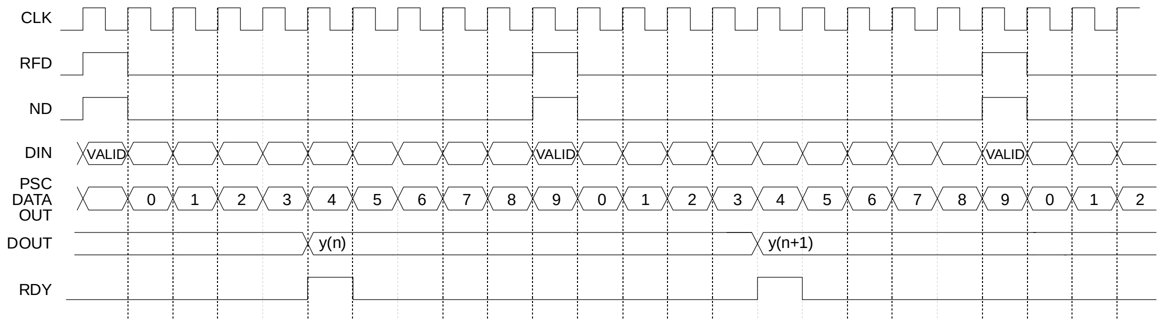


Figure 34: Single channel FIR filter timing. Full serial implementation, 10-bit input samples, registered output. For *L*=10, there are 10 clock periods between successive output samples.

A symmetrical filter with *B*-bit precision input samples requires in general *B*+1 clock periods for a full serial (SDA) implementation. Figure 35 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.

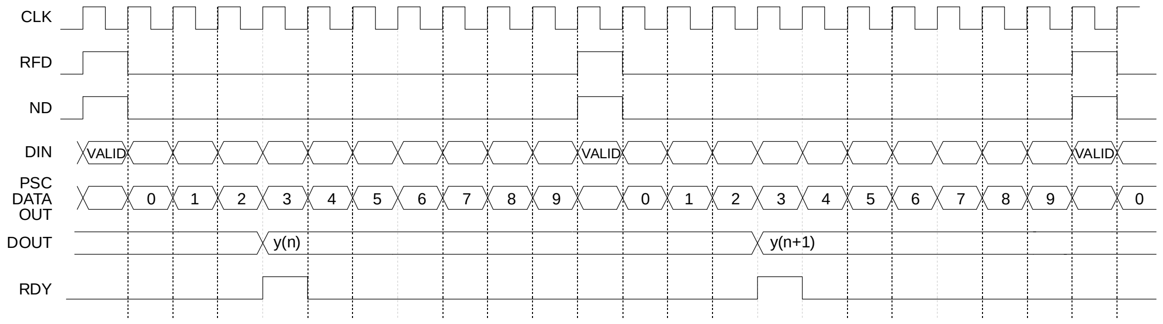


Figure 35: Single channel FIR filter timing. Full serial implementation, 10-bit input samples, symmetrical impulse response, registered output. 11 clock periods are required to process each new input sample.

The previous two figures illustrated 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 36, Figure 37 and Figure 38 illustrate the timing diagrams for a filter with *B*=10 bit precision input samples, with *L*=2, 4 and 6 respectively.

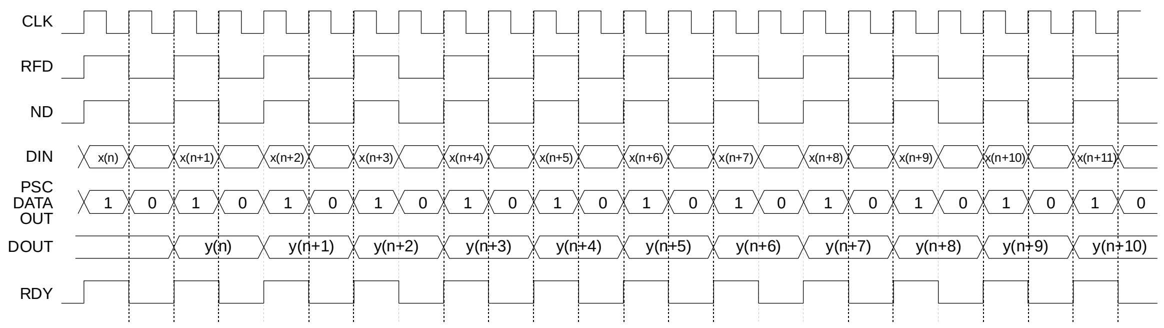


Figure 36: Single channel FIR filter timing. PDA FIR with *B*=10-bit input samples, *L*=2 clock cycles per output sample, registered output. There are 2 clock periods between successive output samples.

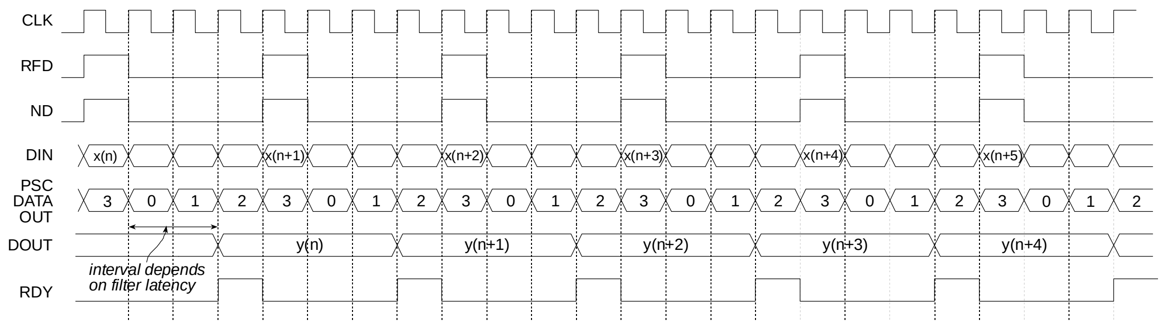


Figure 37: Single channel FIR filter timing. PDA FIR with *B*=10-bit input samples, *L*=4 clock cycles per output sample, registered output. There are 4 clock periods between successive output samples.

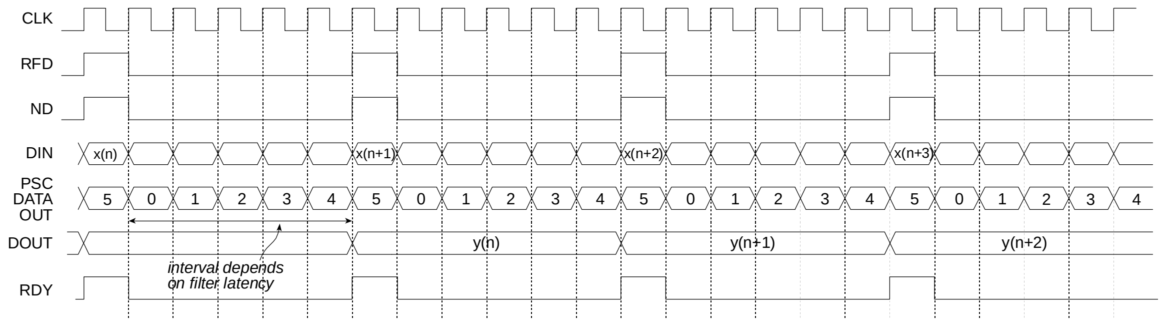


Figure 38: Single channel FIR filter timing. Symmetrical PDA FIR with *B*=10-bit input samples, *L*=6 clock cycles per output sample, registered output. There are 6 clock periods between successive output samples.

Figure 39 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 Section 7) 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 will resume normal operation when *ND* is re-applied (placed in the active again).

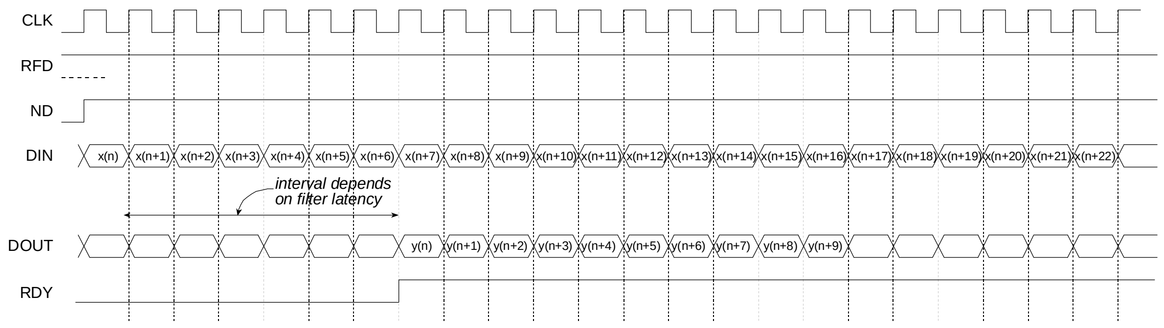


Figure 39: Fully parallel implementation. Single channel filter. With a fully parallel implementation a new output sample is available on each clock edge (after the start-up latency) independent of the filter length or the bit precision of the input data samples.

Figure 40 and Figure 41 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 40 illustrates a filter with an unregistered output while Figure 41 shows the timing for registered output samples.

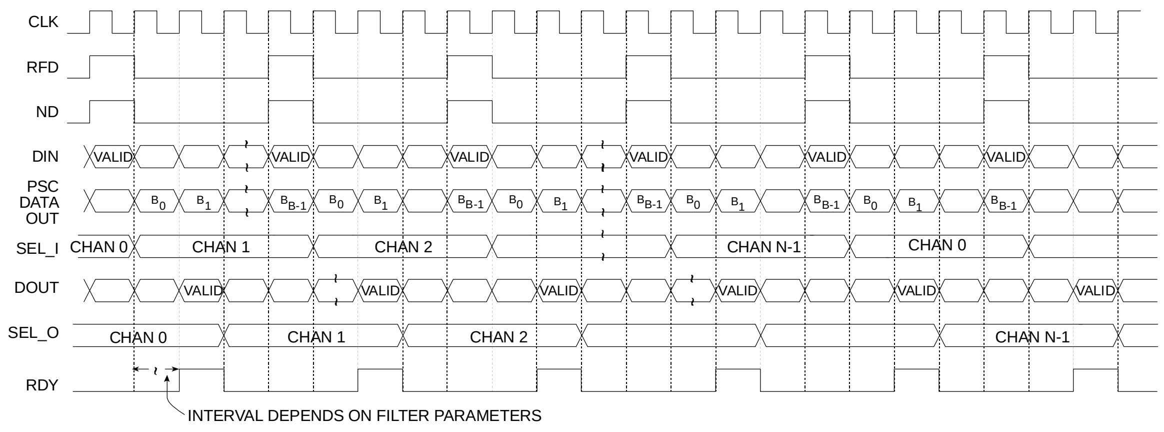


Figure 40: Multi-channel FIR filter timing. Non-symmetrical impulse response, *B*-bit input samples, unregistered output.

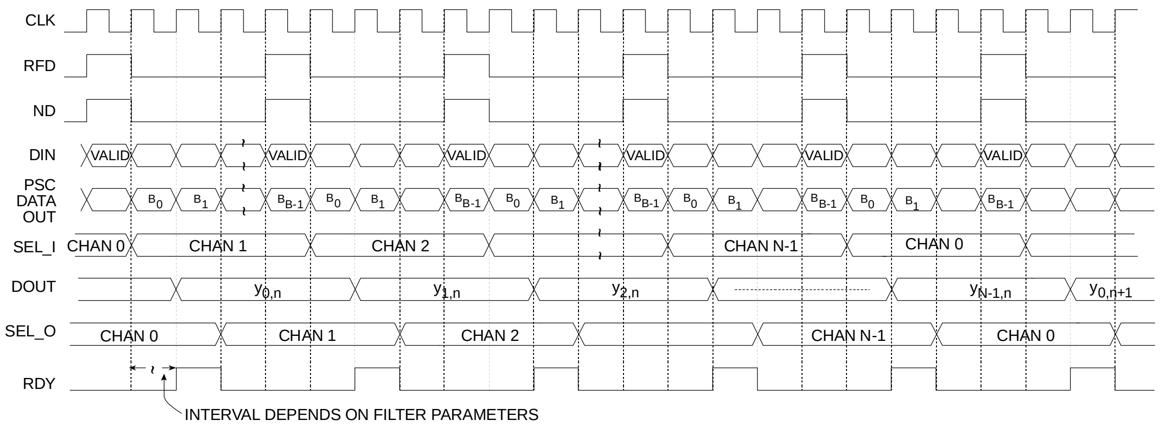


Figure 41: Multi-channel FIR filter timing. Non-symmetrical impulse response, *B*-bit input samples, registered output.

8.3 Polyphase Decimator Timing

Figure 42 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.*

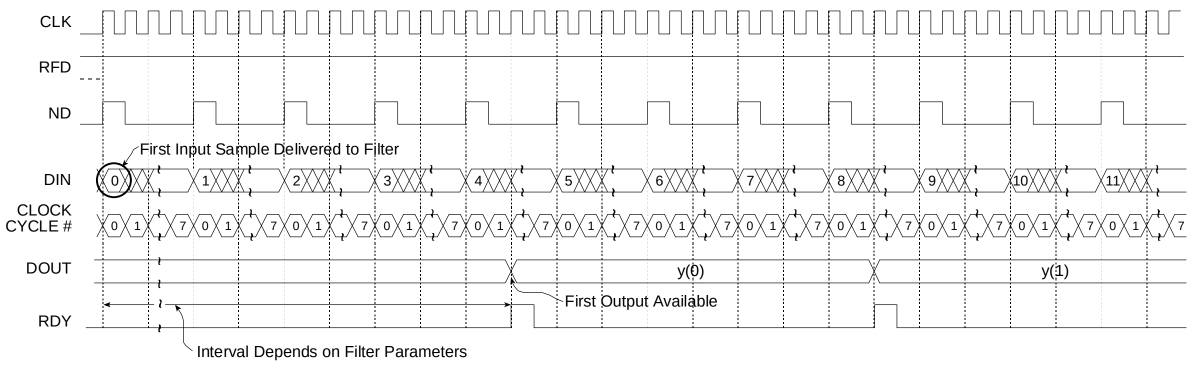

CLK
RFD
ND
DIN
CLOCK CYCLE #
DOUT
RDY
First Input Sample Delivered to Filter
First Output Available
Interval Depends on Filter Parameters
y(0)
y(1)

Figure 42: Polyphase decimator timing. 8-bit precision input samples, down-sampling factor *M*=4.*L*=8.

Figure 43 illustrates the timing for a 4-to-1 polyphase decimator with similar parameters to the filter considered in Figure 42, 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 42, samples can be presented to filter every 4 clock cycles, in contrast to every 8 clock periods in the previous example. The filter supports double the input sample rate, and hence twice the bandwidth, of the filter with *L*=8.

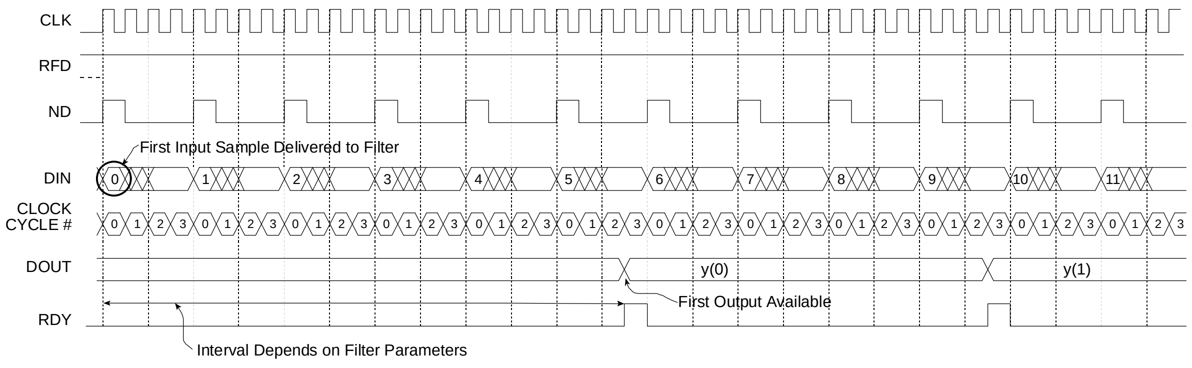

CLK
RFD
ND
DIN
CLOCK CYCLE #
DOUT
RDY
First Input Sample Delivered to Filter
First Output Available
Interval Depends on Filter Parameters
y(0)
y(1)

Figure 43: Polyphase decimator timing. 8-bit precision input samples, down-sampling factor *M*=4.*L*=4.

8.3.1 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 44 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 previously been 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.*

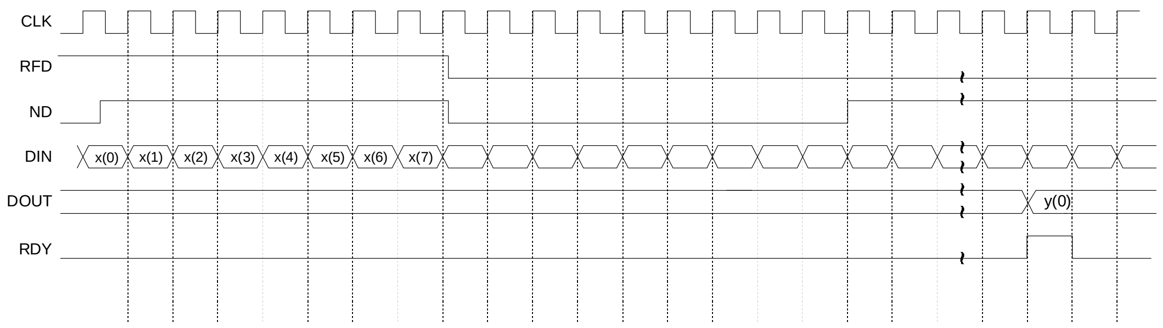


Figure 44: Polyphase decimator timing. 12-bit precision input samples, down-sampling factor *M*=4, *L*=12. Burst input data operation. Diagram shows the timing when the filter is started from from rest, that is, no data has previously been applied to the input port.

Once the filter has moved out of this start-up state input samples must obey the timing diagram shown in Figure 45. Only 4 samples can be supplied in each data burst.

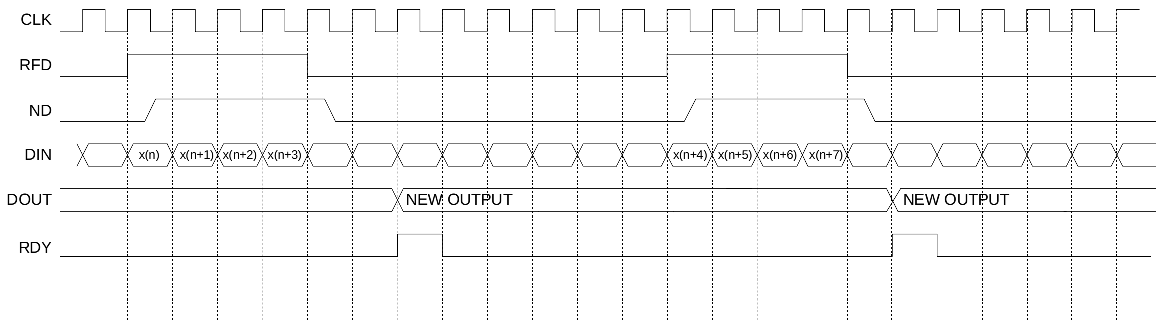


Figure 45: Polyphase decimator timing. 12-bit precision input samples, down-sampling factor *M*=4, *L*=12. Burst input data operation. Diagram shows the timing after the filter has moved out of the start-up timing shown in Figure 44.

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.

8.4 Polyphase Interpolator Timing

Figure 46 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, just like the polyphase decimator, the number of clock cycles specified per output point is associated with the individual sub-filters 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 are available, they 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 46, the vector of *P* output samples is validated by the core output control signal *RDY*.

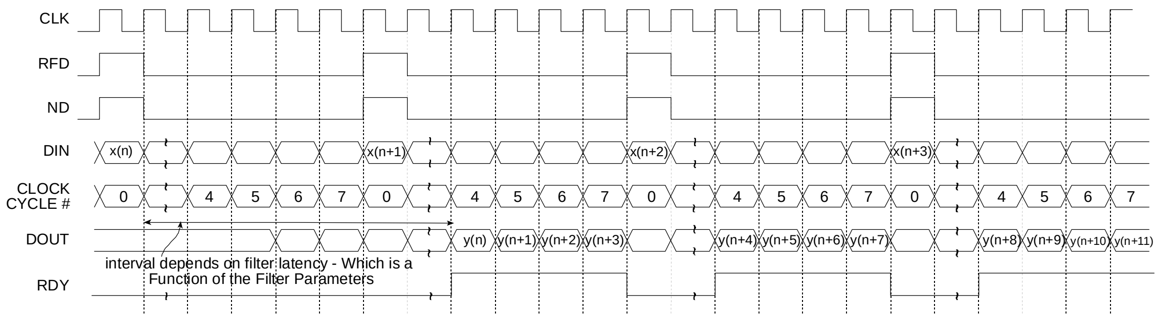

CLK
RFD
ND
DIN
CLOCK CYCLE #
DOUT
RDY
x(n)
x(n+1)
x(n+2)
x(n+3)
0 4 5 6 7 0 4 5 6 7 0 4 5 6 7 0 4 5 6 7
y(n) y(n+1) y(n+2) y(n+3) y(n+4) y(n+5) y(n+6) y(n+7) y(n+8) y(n+9) y(n+10) y(n+11)
interval depends on filter latency - Which is a Function of the Filter Parameters

Figure 46: Polyphase interpolator timing. 8-bit precision input samples, up-sampling factor *P*=4. *L*=8.

Figure 47 shows the timing for an interpolator with similar parameters to the example demonstrated in Figure 46, 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 .

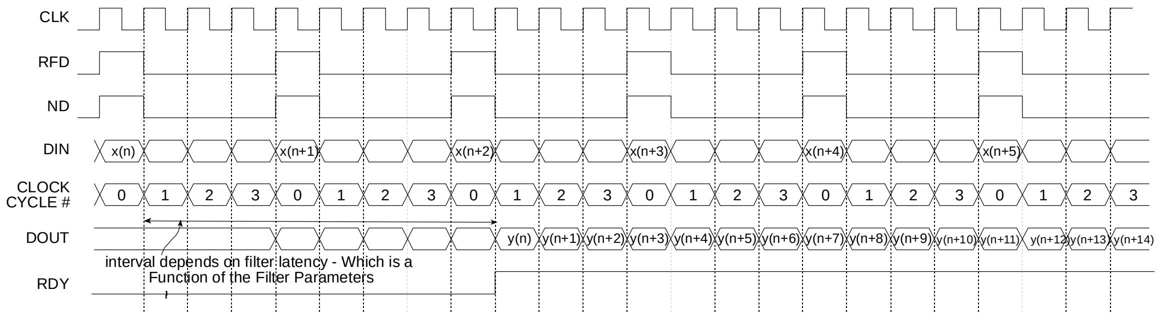

CLK
RFD
ND
DIN
CLOCK CYCLE #
DOUT
RDY
x(n)
x(n+1)
x(n+2)
x(n+3)
x(n+4)
x(n+5)
0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3 0 1 2 3
y(n) y(n+1) y(n+2) y(n+3) y(n+4) y(n+5) y(n+6) y(n+7) y(n+8) y(n+9) y(n+10) y(n+11) y(n+12) y(n+13) y(n+14)
interval depends on filter latency - Which is a Function of the Filter Parameters

Figure 47: Polyphase interpolator timing. 8-bit precision input samples, up-sampling factor *P*=4. *L*=4.

9 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 48 for an *N*-tap filter.

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

Figure 48: 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 49.

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

Figure 49: 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.

9.1 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 50.

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

Figure 50: 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 51 is presented in Figure 52.

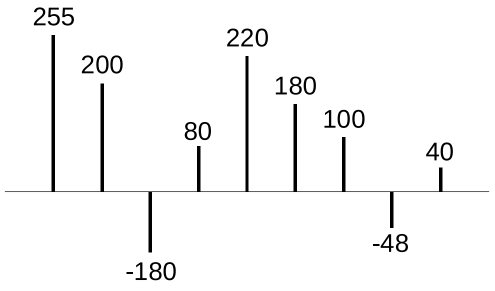


Figure 51: Non-symmetric impulse response.

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

Figure 52: Coefficient file for the non-symmetric impulse response in Figure 51.

The coefficient file for the negative-symmetric filter characterized by the impulse response in Figure 53 is shown in Figure 54.

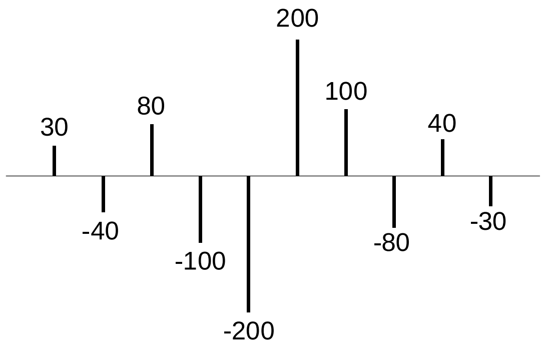


Figure 53: Symmetric impulse response.

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

Figure 54: Coefficient file for the symmetric impulse response in Figure 53.

9.2 Half-Band Filter

As described in a previous section, every second filter coefficient for a half-band filter with an odd number of terms will be zero. When specifying the filter coefficient data for this filter class, the zero value entries need to be included in the coefficient file. For example, the filter coefficient file that specifies the filter impulse response in Figure 55 is shown in Figure 56.

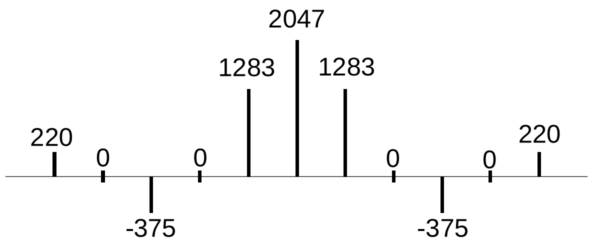


Figure 55: 11-tap half-band filter impulse response.

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

Figure 56: Coefficient file for the half-band filter impulse response shown in Figure 55.

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 will be flagged as an error and the filter will not be generated. A dialog box will be 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 57. However, observe that these values do not contribute to the result.

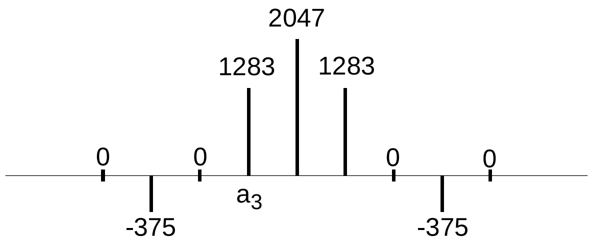


Figure 57: 9-tap half-band filter impulse response.

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.

9.3 Hilbert Transform

The impulse response for a 10-term approximation to a Hilbert transformer is shown in Figure 58. 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 58 is shown in Figure 59.

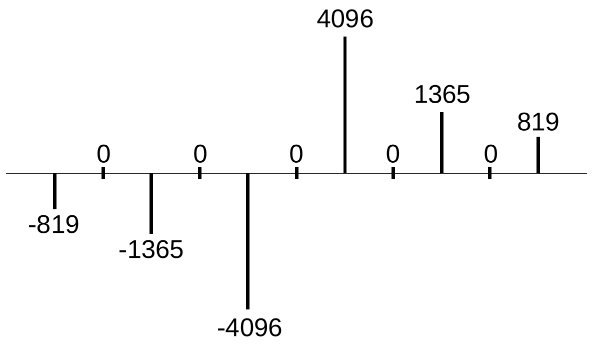


Figure 58: Hilbert transform – impulse response.

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

Figure 59: Coefficient file for the Hilbert transformer with the impulse response shown in Figure 58.

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.

9.4 Interpolated Filter

In a previous section it was 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 60.

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

Figure 60: Prototype coefficient data for IFIR example.

If a zero-packing factor of *k*=2 is specified, the equivalent filter impulse response will be as shown in Figure 61.

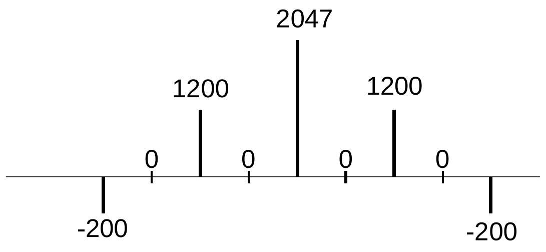


Figure 61: Equivalent IFIR impulse response for the coefficient data shown in Figure 60 with a zero-packing factor *k*=2.

If the zero-packing factor is changed to *k*=3, the impulse response will be as shown in Figure 62.

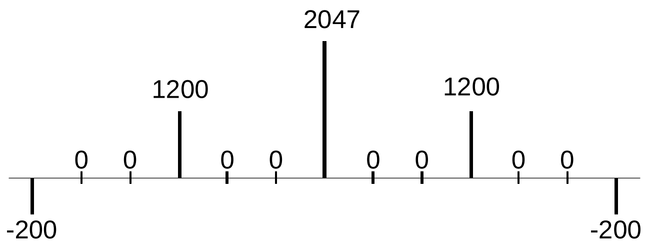


Figure 62: Equivalent IFIR impulse response for the coefficient data shown in Figure 60 with a zero-packing factor *k*=3.

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

10 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 2 through Table 8 provide logic resource requirements for a number of filter configurations.

Table 3: Virtex logic slice utilization for several filter FIR filter configurations. 10-bit filter coefficients. Filter coefficient optimization is 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
	Non-symmetrical	29	33	36	43	67
8	Symmetrical	36	38	44	49	72
	Non-symmetrical	45	50	53	60	82
32	Symmetrical	103	108	113	117	157
	Non-symmetrical	141	146	151	154	196
80	Symmetrical	247	251	255	261	332
	Non-symmetrical	363	369	373	376	454
128	Symmetrical	370	377	380	385	493
	Non-symmetrical	532	536	537	543	646
256	Symmetrical	731	747	740	749	940

Table 4: Virtex logic slice utilization for several filter FIR filter configurations. 12-bit filter coefficients. Filter coefficient optimization is 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
	Non-symmetrical	30	35	39	45	66
8	Symmetrical	36	41	45	52	75
	Non-symmetrical	50	53	56	62	87
32	Symmetrical	111	114	118	125	166
	Non-symmetrical	160	161	168	173	214
80	Symmetrical	268	273	277	279	353
	Non-symmetrical	408	414	413	424	498
128	Symmetrical	402	415	417	421	521
	Non-symmetrical	595	601	599	607	718
256	Symmetrical	797	806	819	810	1003

Table 5: Virtex logic slice utilization for several half-band filter configurations. 14-bit filter coefficients. Filter coefficient optimization is 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 6: Virtex logic slice utilization for several Hilbert transformer configurations. 14-bit filter coefficients. Filter coefficient optimization is 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 7: Virtex logic slice utilization for several interpolated filter configurations. 16-bit filter coefficients. Filter coefficient optimization is 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
	Non-symmetrical	56	66	71	84	122
32	Symmetrical	146	170	198	201	303
	Non-symmetrical	189	214	239	264	366
80	Symmetrical	359	410	474	477	705
	Non-symmetrical	488	550	609	668	897

Table 8: Virtex logic slice utilization for several PDA FIR filter configurations. 12-bit filter coefficients. 12-bit input data, 60-taps. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output, non-symmetrical impulse response. Filter master clock frequency is 150 MHz.

Number of Clock Cycles per Output Sample	Slice Count	Filter Sample Rate† (MHz)
1	3072	150
2	1571	75
3	994	50
4	802	37.5
6	511	25
12	268	12.5

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

11 Ordering Information

This core is downloadable free of charge from the Xilinx IP Center (www.xilinx.com/ipcenter), for use with version 3.1i and later versions of the Xilinx Core Generator System. The Core Generator System is bundled with the Alliance and Foundation implementation tools.

To order Xilinx software contact your local Xilinx sales representative. For information on the Xilinx sales office nearest you, please refer to:

http://www.xilinx.com/company/sales.htm

12 Reference

[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