IP LogiCORE FIR Compiler v5.0

DS534 March 1, 2011 **Product Specification**

Introduction

The Xilinx LogiCORE™ IP FIR Compiler core provides a common interface for users to generate highly parameterizable, area-efficient high-performance FIR filters utilizing either Multiply-Accumulate (MAC) or Distributed Arithmetic (DA) architectures.

Features

- Drop-in module for Kintex™-7, Virtex®-7, Virtex®-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/AN/3A DSP/XA FPGAs
- High-performance finite impulse response (FIR), polyphase decimator, polyphase interpolator, half-band, half-band decimator, half-band interpolator, Hilbert transform, polyphase filter bank, and interpolated filter implementations
- Multiply-Accumulate (MAC) and Distributed Arithmetic (DA) architectures available
- Support for up to 256 sets of coefficients, with 2 to 2048 coefficients per set
- Input data up to 49-bit precision
- Filter coefficients up to 49-bit precision
- Support for up to 64 channels generally and up to 1024 for polyphase filter bank implementations
- Interpolation and decimation factors of up to 64 generally and up to 1024 for single channel filters
- Support for multiple parallel data paths with shared control logic
- DA-based filters support both serial and parallel implementation
- MAC implementations use single or multiple MAC engines to achieve specified filter performance
- Data-flow-style core interface and control
- On-line coefficient reload capability
- User-selectable output rounding available in MAC implementations
- Use with Xilinx CORE Generator™ and Xilinx System Generator for DSP v13.1

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Virtex-7 and Kintex-7, Virtex-6, Virtex-5, Virtex-4, Spartan-6, Spartan-3/XA, Spartan-3E/XA, Spartan-3A/3AN/3A DSP/XA
Supported User Interfaces	Not Applicable
Provided with Core	
Documentation	Product Specification
Design Files	Netlist
Example Design	Not Provided
Test Bench	Not Provided
Constraints File	Not Applicable
Simulation Model	VHDL behavioral model in the xilinxcorelib library VHDL UniSim structural model Verilog UniSim structural model
Tested Design Tools	
Design Entry Tools	CORE Generator tool 13.1 System Generator for DSP 13.1
Simulation	Mentor Graphics ModelSim 6.6d Cadence Incisive Enterprise Simulator (IES) 10.2 Synopsys VCS and VCS MX 2010.06 ISIM 13.1
Synthesis Tools	N/A
Support	
Provided by Xilinx, Inc.	

1. For a complete listing of supported devices, see the release notes for this core.

Overview

A wide range of filter types can be implemented in the Xilinx CORE Generator: single-rate, half-band, Hilbert transform and interpolated filters, in addition to multi-rate filters such as 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 conventional single-rate FIR version of the core computes the convolution sum defined in Equation 1, where N is the number of filter coefficients.

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

Figure 1 illustrates the conventional tapped delay line realization of this inner-product calculation, and although the illustration is a useful conceptualization of the computation performed by the core, the actual FPGA realization is quite different.

Where a MAC realization is selected, one or more time-shared multiply-accumulate (MAC) functional units are used to service the N sum-of-product calculations in the filter. The core automatically determines the minimum number of MAC engines required to meet user-specified throughput. Where a Distributed Arithmetic (DA) realization [Ref 1] [Ref 2] is selected, no explicit multipliers are employed in the design; only look-up tables (LUTs), shift registers, and a scaling accumulator are required.

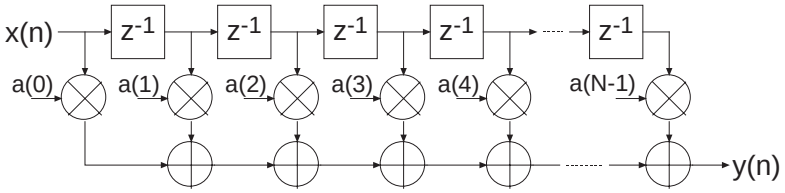


Figure 1: **Conventional Tapped Delay Line FIR Filter Representation**

Feature Support Matrix

The FIR Compiler implements three distinct filter architectures: Distributed Arithmetic, Systolic Multiply-Accumulate, and Transpose Multiply-Accumulate. Feature support is not uniform across these architectures and is summarized in Table 1 and Table 2.

Table 1: **Feature Support Matrix**

Feature	Distributed Arithmetic	Systolic Multiply-Accumulate		Transpose Multiply-Accumulate	
		Virtex-5/6 FPGAs	Other Families	Virtex-5/6 FPGAs	Other Families
Number of Coefficients	2–1024	2–1024	2–1024	2–1024	2–1024
Coefficient Width(1)	1–32	2–49	2–35	2–49	2–35
Data Width(1,2)	1–32	2–49	2–35	2–49	2–35
Number of Channels	1–8	1–64	1–64	×	×
Parallel Data Paths(3)	×	1-16	1-16	1-16	1-16

Table 1: **Feature Support Matrix *(Cont'd)***

Feature	Distributed Arithmetic	Systolic Multiply-Accumulate		Transpose Multiply-Accumulate	
		Virtex-5/6 FPGAs	Other Families	Virtex-5/6 FPGAs	Other Families
Maximum Rate Change *Single Channel* *Multiple Channels*	 8 1	 1024 512	 1024 512	 1024 N/A	 1024 N/A
Fractional Rate Support	✗	✓	✓	✗	✗
Coefficient Reload *Offline* *Online (glitch-free)*	 ✓ ✗	 ✓ ✓	 ✓ ✓	 ✓ ✓	 ✓ ✓
Coefficient Sets	1	1–256	1–256	1–256	1–256
Output Rounding	✗	✓	✓	✓	✓

Notes:

1. Maximum Coefficient Width reduces by one Multiply-Accumulate architectures when the Coefficients are signed. Similarly for Maximum Data Width when the Data values are signed.
2. The allowable range for the Data Width field in the GUI may reduce further in Virtex-5/6 devices to ensure that the accumulator width does not exceed maximum.
3. Maximum Parallel Data Paths reduces to 8 when Coefficient Width or Data Width is greater than 25-bits for Virtex-5/6 FPGAs or 18-bits for other families.

Table 2 shows the classes of filters that are supported for the FIR Compiler core.

Table 2: **Filter Configuration Support Matrix**

Filter Configuration	Distributed Arithmetic	Systolic Multiply-Accumulate	Transpose Multiply-Accumulate
Conventional Single-rate FIR	✓	✓	✓
Half-band FIR	✓	✓	✗
Hilbert Transform [Ref 3]	✓	✓	✗
Interpolated FIR [Ref 4] [Ref 5]	✓	✓	✗
Polyphase Decimator	✓	✓	✓
Polyphase Interpolator	✓	✓	✓
Half-band Decimator	✓	✓	✗
Half-band Interpolator	✓	✓	✗
Polyphase Filter Bank	✗	✓	✗

The supported filter configurations are described in separate sections within this document.

Notable Limitations

In conjunction with Table 1 and Table 2, it is important to note some further limitations inherent in the core.

When selecting the Systolic Multiply-Accumulate architecture, the limitations are as follows:

- Symmetry is not exploited in configurations requiring multiple columns of DSP slices.

- Fractional Rate filters do not currently exploit coefficient symmetry.

When selecting the Transpose Multiply-Accumulate architecture, the limitations are as follows:

- Symmetry is not exploited.
- Multiple channels are not supported.

When selecting the Distributed Arithmetic-based core architecture, the limitations are as follows:

- Symmetry is not exploited for multi-rate filters.

Core Symbol and Port Definitions

Figure 2 displays the schematic symbol for the interface pins to the FIR Compiler module.

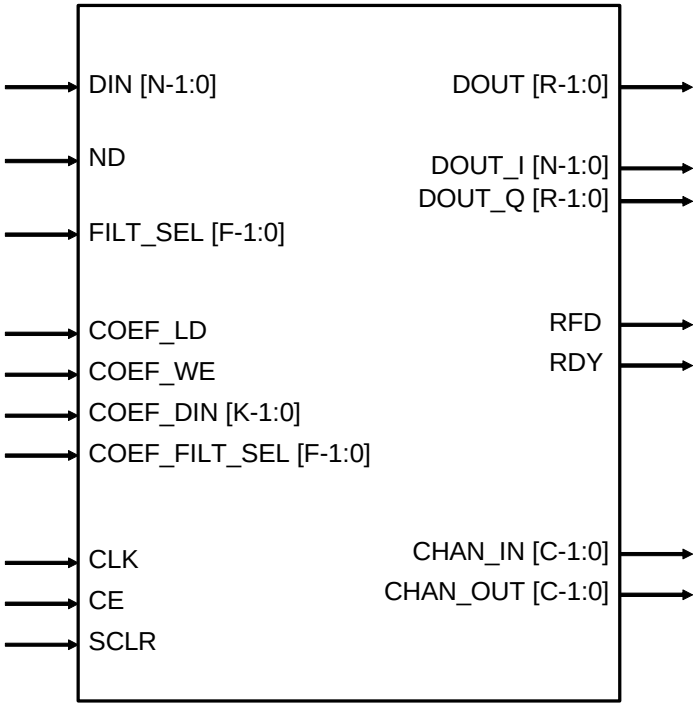


Figure 2: **FIR Compiler Core Pinout**

Filter input data is supplied on the DIN port (N bits wide), and filter output samples are presented on the DOUT port (R bits wide). The maximum output width R is the sum of the data bit width N and the bit growth of the filter; see the Output Width and Bit Growth section for more details. The output width may also be reduced further under user control by truncation or rounding. The CLK signal is the system clock for the core, where the clock rate may be greater than or equal 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. These core interface signals are detailed in "Interface, Control, and Timing."

For Hilbert transform filter implementations, a pair of In-Phase/Quadrature data outputs is provided. The In-Phase data output is N bits wide, as it is a delayed version of the input data, while the Quadrature data output is R bits wide, calculated as described previously. For multiple channel implementations, a pair of indicator signals is provided to specify the currently active input and output channels. These indicator signals are C bits wide, where C is the required bit width to represent the maximum channel value. Where multiple coefficient sets are specified in the .coe file, a filter selection input is available to select the active filter set, and this is F bits wide. F is the required bit

width to represent the maximum filter set value. Coefficient reloading can be achieved by driving the coefficient reload interface, which consists of a load start indicator, a write enable, and a coefficient data bus (K bits wide for most filter types). Where reloading is required with multiple filter sets, the filter set to be reloaded can be specified using the COEF_FILT_SEL port, which is again F bits wide. Resetting the core is achieved by driving the SCLR pin; it does not require the assertion of clock enable (CE). A clock enable pin is available only for the Multiply-Accumulate filter architectures.

Table 3 defines the FIR filter port names and port functional descriptions.

Table 3: **Core Signal Pinout**

Name	Direction	Description
SCLR	Input	SYNCHRONOUS CLEAR Synchronous reset (active high). Asserting SCLR synchronously with CLK resets the filter internal state machines. It does *NOT* reset the filter data memory contents (regressor vector). SCLR has priority over CE. SCLR is an optional pin.
CLK	Input	CLOCK Core clock (active rising edge). Always present.
CE	Input	CLOCK ENABLE Core clock enable (active high). Available for MAC-based FIR implementations.
DIN [N-1:0]	Input	DATA IN N-bit wide filter input sample. Always present. Note that for multi-channel implementations, this input is time-shared across all channels. Separate channel inputs are not provided.
ND	Input	NEW DATA (active high) When this signal is asserted, the data sample presented on the DIN port is accepted into the filter core. ND should not be asserted while RFD is low; any samples presented when RFD is low are ignored by the core.
FILT_SEL [F-1:0]	Input	FILTER SELECT Filter Selection input signal, F-bit wide where F = ceil(log2(filter sets)). Only present when using multiple filter sets.
COEF_LD	Input	COEFFICIENT LOAD Indicates the beginning of a new coefficient reload cycle.
COEF_WE	Input	COEFFICIENT RELOAD WRITE ENABLE WE for loading of coefficients into the filter to allow a host to halt loading until ready to transmit on the interface.
COEF_DIN [K-1:0]	Input	COEFFICIENT RELOAD DATA IN Input data bus for reloading coefficients. K is the core coefficient width for most filter types and coefficient width + 2 for interpolating filters where the symmetric coefficient structure is exploited.
COEF_FILT_SEL [F-1:0]	Input	COEFFICIENT RELOAD FILTER SELECT Filter Selection input signal for reloading coefficients, F-bit wide where F = ceil(log2(filter sets)). Only present when using multiple filter sets and reloadable coefficients.
DOUT [R-1:0]	Output	DATA OUT R-bit-wide output sample bus. 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	READY Filter output ready flag (active high). indicates that a new filter output sample is available on the DOUT port.
RFD	Output	READY FOR DATA Indicator to signal that the core is ready to accept a new data sample. Active high.

Table 3: **Core Signal Pinout *(Cont'd)***

Name	Direction	Description
CHAN_IN [C-1:0]	Output	INPUT CHANNEL SELECT Standard binary count generated by the core that indicates the current filter input channel number.
CHAN_OUT [C-1:0]	Output	OUTPUT CHANNEL SELECT Standard binary count generated by the core that indicates the current filter output channel number.
DOUT_I [N-1:0]	Output	DATA OUT IN-PHASE Hilbert transform only. In-phase (I) data output 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	DATA OUT QUADRATURE Hilbert transform only. Quadrature (Q) data output 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.
DATA_VALID	Output	DATA VALID Indicator signal that can be used in conjunction with or in preference to `RDY`. The signal indicates that a new filter output sample is available on the `DOUT` port that has been generated from a complete data vector. Available for MAC-based FIR implementations.

Parallel Data Paths

Up to 16 parallel data paths are supported when the Multiply-Accumulate filter architectures are selected, as detailed in Parallel Data Paths, page 50. Table 4 defines how the core pinout must be adapted for multiple data path operation.

Table 4: **Single-to-Multiple Data Path Pinout Conversion**

Single Path	Multiple Paths
SCLR	SCLR
CLK	CLK
CE	CE
DIN	DIN_1 .... DIN_16
ND	ND
FILT_SEL	FILT_SEL
COEF_LD	COEF_WE
COEF_DIN	COEF_DIN
COEF_FILT_SEL	COEF_FILT_SEL
DOUT	DOUT_1 .... DOUT_16
RDY	RDY
RFD	RFD
CHAN_IN	CHAN_IN
CHAN_OUT	CHAN_OUT
DOUT_I	DOUT_I_1 .... DOUT_I_16

Table 4: **Single-to-Multiple Data Path Pinout Conversion *(Cont'd)***

Single Path	Multiple Paths
DOUT_Q	DOUT_Q_1 .... DOUT_Q_16
DATA_VALID	DATA_VALID

CORE Generator Graphical User Interface

The FIR Compiler GUI contains four pages used to configure the core plus three informational/analysis tabs.

Tool Tips appear when hovering the mouse over each parameter and a brief description appears, as well as feedback about how their values or ranges are affected by other parameter selections. For example, the Coefficient Structure Tool Tip displays the inferred structure when Inferred is selected from the drop-down list.

Tab 1: IP Symbol

The IP Symbol tab illustrates the core pinout.

Tab 2: Freq. Response

The Freq. Response tab (Figure 3), the default tab when the CORE Generator is started, displays the filter frequency response (magnitude only). The content of the tab can be adjusted to fit the entire window or un-docked (as shown) into a separate window.

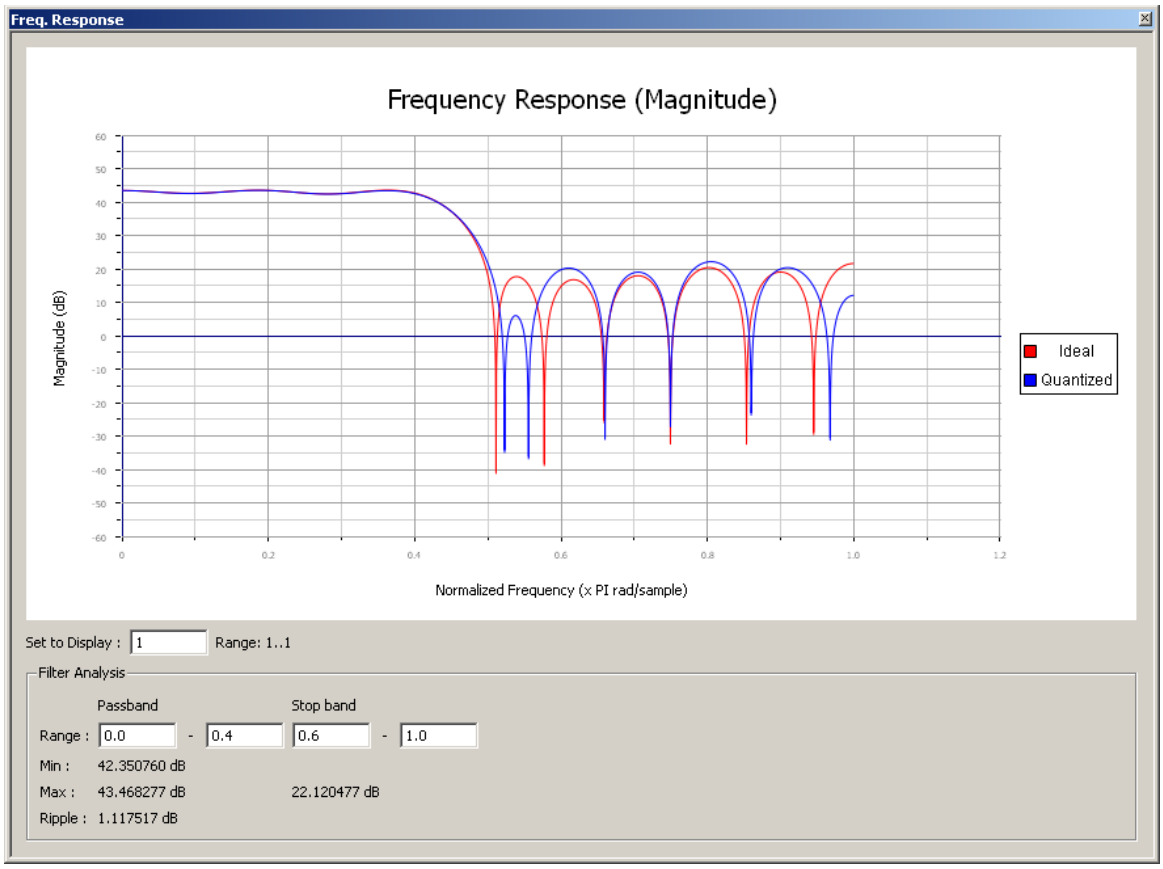


Figure 3: **Freq. Response Tab**

The frequency response of the currently selected coefficient set is plotted against normalized frequency. Where the Quantization option is set to Integer Coefficients, there is only a single plot based on the specified coefficient values. Where the Quantization option has been set to Quantize Only, an ideal plot is displayed based on the provided values alongside a Quantized plot based on a set of coefficient values quantized according to the specified coefficient bit width. Where the Quantization option is set to Maximize Dynamic Range, the coefficients are first scaled to take full advantage of the available dynamic range, then quantized according to the specified coefficient bit width. The quantized coefficients are summed to determine the resulting gain factor over the provided real coefficient set, and the resulting scale factor is used to correct the filter response of the quantized coefficients such that the gain is factored out. The scale factor is reported in the legend text of the frequency response plot and on the Summary page. See the Coefficient Quantization section for more details.

- **Set to Display:** This selects which of multiple coefficient sets (if applicable) is displayed in the Frequency Response Graph.
- **Passband Range:** Two fields are available to specify the passband range, the left-most being the minimum value and the right-most the maximum value. The values are specified in the same units as on the graph x-axis (for example, normalized to pi radians per second). For the specified range the passband maximum, minimum and ripple values are calculated and displayed (in dB).
- **Stopband Range:** Two fields are available to specify the stopband range, the left-most being the minimum value and the right-most the maximum value. The values are specified in the same units as on the graph x-axis (for example, normalized to pi radians per second). For the specified range the stopband maximum value is calculated and displayed (in dB).

The user can specify any range for the passband or stopband, allowing closer analysis of any region of the response. For example, examination of the transition region can be done to more accurately examine the filter roll-off.

Tab 3: Implementation Details

The Implementation Details tab displays Resource Estimation information, core latency, actual calculated coefficients, and the coefficient reload order.

The number of DSP slices/Multipliers is displayed in addition to a count of the number of block RAM elements required to implement the design. Usage of general slice logic is not currently estimated. Resource estimation is currently available only for MAC-based FIR filters.

For some configurations, the number of coefficients calculated by the core may be greater than specified. In this circumstance, the user could increase the number coefficients used to specify the filter at little or no cost in resource usage.

The coefficient reload order is displayed when “Use Reloadable Coefficients” has been selected and “Display Reload Order” is checked. This information is also contained in the <component_name>_reload_order.txt file produced during core generation. See the Coefficient Reload Order section for more details.

It should be noted that the results presented in the Resource Estimation are estimates only using equations that model the expected core implementation structure. The Resource Utilization option within the CORE Generator should be used after generating the core to get a more accurate report on all resource usage. It is not guaranteed that the resource estimates provided in the GUI match the results of a mapped core implementation

Filter Specification Screen

The Filter Specification screen is used to define the basic configuration and performance of the filter.

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

- **Coefficient Source:** Specifies which coefficient input method to use, directly in the GUI via the Coefficient Vector parameter or from a .coe file specified by the Coefficient File parameter.
- **Coefficient Vector:** Used to specify the filter coefficients directly in the GUI. The filter coefficients are specified in decimal using a comma delimited list as for the "coefdata" field in the Filter Coefficient Data file. As with the .coe file, the filter coefficients can be specified using non-integer real numbers which the FIR Compiler quantizes appropriately given the user requirements. See the Coefficient Quantization section for more details.
- **Coefficients File**: Coefficient file name. This is the file of filter coefficients. The file has a .coe extension, and the file format is described in theFilter Coefficient Data section. The file can be selected through the dialog box activated by the Browse.
- **Show Coefficients:** Selecting this button displays the filter coefficient data defined in the specified Coefficient file in a pop-up window.
- **Number of Coefficient Sets**: The number of sets of filter coefficients to be implemented. The value specified must divide without remainder into the number of coefficients derived from the .coe file or Coefficient Vector.
- **Number of Coefficients (per set):** The number of filter coefficients per filter set. This value is automatically derived from the specified coefficient data and the specified number of coefficient sets.
- **Filter Type**: Six filter types are supported: Single-rate FIR, Interpolated FIR, Interpolating FIR, Decimating FIR, Receiver and Transmitter Polyphase filter bank.
- **Rate Change Type:** This field is applicable to Interpolation and Decimation filter types. Used to specify an Integer or Fixed Fractional rate change.
- **Interpolation Rate Value:** This field is applicable to all Interpolation filter types and Decimation filter types for Fractional Rate Change implementations. The value provided in this field defines the up-sampling factor, or P for Fixed Fractional Rate (P/Q) resampling filter implementations.
- **Decimation Rate Value:** This field is applicable to the all Decimation and Interpolation filter types for Fractional Rate Change implementations. The value provided in this field defines the down-sampling factor, or Q for Fixed Fractional Rate (P/Q) resampling filter implementations.
- **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). A zero packing factor of *k* inserts *k*-1 zeros between the supplied coefficient values.
- **Number of Channels:** The number of channels processed by the filter.
- **Hardware Oversampling Specification format:** Selects which format is used to specify the hardware oversampling rate, the number of clock cycles available to the core to process an input sample and generate an output. This value directly affects the level of parallelism in the core implementation and resources used. When "Frequency Specification" is selected, the user specifies the Input Sampling Frequency and Clock Frequency. The ratio between these values along with other core parameters determine the hardware oversampling rate. When "Sample Period" is selected, the user specifies the integer number of clock cycles between input samples.
  - **Input Sampling Frequency:** This field can be an integer or real value; it specifies the sample frequency for one channel. The upper limit is set based on the clock frequency and filter parameters such as Interpolation Rate and number of channels.
  - **Clock Frequency:** This field can be an integer or real value. The limits are set based on the sample frequency, interpolation rate, and number of channels. **Note that this field influences architecture choices only; the specified clock rate may not be achievable by the final implementation.**
  - **Input/Output Sample Period:** Integer number of clock cycles between input samples. When the multiple channels have been specified, this value should be the integer number of clock cycles between the time division multiplexed input sample data stream. When a fixed fractional decimation filter has been specified, this parameter specifies the integer number of clock cycles between output samples. Specifying the output sample period enables a more efficient use of the available clock cycles.

Implementation Options Screen

The Implementation Options screen is used to define which filter architecture and coefficient structure to use and to configure the various data path and coefficient options

- **Filter Architecture**: Three filter architectures are supported: Systolic Multiply Accumulate, Transpose Multiply Accumulate, and Distributed Arithmetic.
- **Use Reloadable Coefficients**: When the "Reloadable" option is selected, a coefficient reload interface is provided on the core.
- **Coefficient Structure:** Five coefficient structures are supported: Non-symmetric, Symmetric, Negative Symmetric, Half-band, and Hilbert transform. The structure can also be inferred from the coefficient file directly (default setting), or specified directly. Note the inference algorithm only analyses the first 2048 coefficients. Only valid structure options, based on analysis of the provided coefficient file, are available for the user to specify directly.
- **Coefficient 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.
- **Quantization:** Specifies the quantization method to be used. Available options are Integer Coefficients, Quantize Only, or Maximize Dynamic Range.
  - The Integer Coefficients option is only available when the filter coefficients have been specified using only integer values.
  - The Quantize Only option simply rounds the provided values to the nearest quantum using a simple rounding towards zero algorithm.
  - The Maximize Dynamic Range option scales all coefficients such that the maximum coefficient is equal to the maximum representable number in the specified bit width, thus maximizing the dynamic range of the filter (note that with the current implementation, overflow is not possible, as the accumulator width is automatically set to accommodate maximum bit growth within the filter). See the Coefficient Quantization section for more information.
- **Coefficient Width:** The bit precision of the filter coefficients. This field can be used with the filter response graph to explore the possibilities for more efficient implementation by limiting coefficient bit width to the minimum required to meet the user's target specification for the filter.
- **Best Precision Fraction Length:** When selected, the coefficient fractional width is automatically set to maximize the precision of the specified filter coefficients. See the Best Precision Fractional Length section for further information.
- **Coefficient Fractional Bits:** Specifies the number of coefficient bits that are used to represent the fractional portion of the provided filter coefficients. The maximum value it supports is the Coefficient Width value minus the required integer bit width. The integer bit width value is static and is automatically determined by calculating the integer bit width required to represent the maximum value contained in the provided coefficient sets. When the coefficient width is less than the required integer bit width, this field reports zero. When the required integer bit width is zero, this parameter may take a value greater than the Coefficient Width. See the Coefficient Quantization section for more information.
- **Number of Paths:** Specifies the number of parallel data paths the filter is to process.
- **Input Data Type:** The filter input data can be specified as either signed or unsigned. The signed option employs conventional two's complement arithmetic.
- **Input Data Width:** The precision (in bits) of the filter input data samples.
- **Input Data Fractional Bits:** The number of Input Data Width bits used to represent the fractional portion of the filter input data samples. This field is for information only. It is used in conjunction with Coefficient Fractional Bits to calculate the filter Output Fractional Bits value.
- **Output Rounding Mode:** Specifies the type of rounding to be applied to the output of the filter.

- **Output Width:** When using Full Precision, this field is disabled and indicates the output precision (in bits) of the filter output data samples, including bit growth. When using any other Rounding Mode, this field allows the user to specify the desired output sample width.
- **Output Fractional Bits:** This field reports the number Output Width bits used to represent the fractional portion of the filter output samples.
- **Allow Rounding Approximation:** When using either of the two Symmetric rounding modes, a spare cycle is normally required to allow determination of the sign of the final accumulated result; however it is possible to approximate symmetric rounding without this spare cycle by checking the sign of the penultimate accumulation value. This checkbox allows the user to specify whether or not such approximation is permitted.
- **Registered Output:** 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.

Detailed Implementation Options Screen

The Detailed Implementation Options screen is used to configure various control and implementation options.

Be aware that using the available control pins can require a moderate increase in resources and can lead to a reduction in maximum achievable clock frequencies; these options should only be used if required. To halt operation of the core, use either CE (which freezes all core operations) or hold ND low (which allows samples currently being processed to be completed) and pause the input data stream until resumption of normal core operation is desired

- **Optimization Goal:** Specifies if the core is required to operate at maximum possible speed (Speed option) or minimum area (Area option). The Area option is the recommended default and normally achieves the best speed and area for the design; however in certain configurations, the Speed setting may be required to improve performance at the expense of overall resource usage (this setting normally adds pipeline registers in critical paths).
- **SCLR:** Specifies if the core is to have a reset pin. This pin can be used with any other pin combination.
- **Use Deterministic SCLR Behavior:** Specifies that the core has deterministic behavior after SCLR has been asserted. See Resetting the Core for more information. This option is only available for Multiply-Accumulate architectures.
- **DATA_VALID:** Specifies that the core is to have a DATA_VALID pin. See Handshake Control Signals for more information. This pin is only available when SCLR has been selected and for Multiply-Accumulate architectures.
- **CE:** Specifies if the core is to have a clock enable pin. This pin can be used with any other pin combination, although it can be used to replace ND as a means to halt core operation, which can lead to significant reductions in resource usage for parallel symmetric filter implementation structures.
- **ND:** Specifies if the core is to have a New Data pin. This pin can be used with any other pin combination. If the ND pin is not present, samples are assumed to be present on the input data bus at specific cycle times according to the designated sample rate, and the input is sampled at those times. This is indicated by the core by RFD pulsing high during those cycles.
- **Generate CHAN_IN Value in Advance**: Specifies that the filter generates the CHAN_IN value a number of input samples in advance. See Input/Output Channel Decoding for more information. This option is only available when multiple channels have been selected and for Multiply-Accumulate architectures.
- **Number of Samples:** Specifies the number of inputs sample in advance that the CHAN_IN value is generated. This field is only available when Generate CHAN_IN Value in Advance has been selected.
- **Memory Options:** The memory type for MAC implementations can either be user-selected or chosen automatically to suit the best implementation options. Note that choosing Distributed may result in shift register implementation where appropriate to the filter structure. Forcing the RAM selection to be either

"Block" or "Distributed" should be used with caution, as inappropriate use can lead to inefficient resource usage. The default "Automatic" mode is recommended for most users.

- **Data Buffer Type:** Specifies the type of RAM to be used to store data within a MAC element. Users can select either "Block" or "Distributed RAM" options, or select "Automatic" to allow the core to choose the memory type appropriately.
- **Coefficient Buffer Type:** Specifies the type of RAM to be used to store coefficients within a MAC element. Users can select either "Block" or "Distributed RAM" options, or select "Automatic" to allow the core to choose the memory type appropriately.
- **Input Buffer Type:** Specifies the type of RAM to be used to implement the data input buffer, where present. Users can select either "Block" or "Distributed RAM" options, or select "Automatic" to allow the core to choose the memory type appropriately.
- **Output Buffer Type:** Specifies the type of RAM to be used to implement the data output buffer, where present. Users can select either "Block" or "Distributed RAM" options, or select "Automatic" to allow the core to choose the memory type appropriately.
- **Preference for Other Storage:** Specifies the type of RAM to be used to implement general storage in the data path. Users can select either "Block" or "Distributed RAM" options, or select "Automatic" to allow the core to choose the memory type appropriately. Since this covers several different types of storage, it is recommended that users specify this type of memory directly only if they really need to steer the core away from using a particular memory resource (for example, if they are short of block RAMs in their overall design).

- **Multi-column Support:** For device families with XtremeDSP™ Slices, implementations of large high speed filters might require chaining of DSP slice elements across multiple columns. Where applicable (the feature is only enabled for multi-column devices), the user can select the method of folding of the filter structure across the multiple columns, which can be "Automatic" (based on the selected device for the project) or "Custom" (user specifies the length of each column). The Multiple Column Filter implementation section describes the multi-column implementation in more detail.
- **Device Column Lengths:** Information only. Displays the column length pattern in a comma delimited list for the selected project device.
- **Column Configuration:** Specifies the individual column lengths in a comma delimited list. When "Automatic" has been selected, the column lengths are determined by the GUI starting with the first column in the device column pattern. When "Custom" is selected, the user can specify the desired column pattern. The number of columns may not exceed that available in the selected device, and the individual column lengths must sum to the number of DSP slices utilized by current filter configuration. When the selected device has various column lengths, it maybe desirable to skip a particular column; this can be done by specifying a zero column length, for example 10,0,22. **Note that the specified column configuration does not guarantee that the downstream tools are going to place the columns in the desired sequence.**
- **Inter-column Pipe Length:** Pipeline stages are required to connect between the columns, with the level of pipelining required being dependent upon the required system clock rate, the chosen device, and other system-level parameters. Choice of this parameter is always left for the user to specify.

Summary Screen

The Summary screen provides a summary of core options selected.

Summary: The final page provides summary information about the core parameters selected, which includes information on the actual number of calculated coefficients, including padding; the inferred or specified coefficient structure; the additional gain incurred as data passes through the filter due to maximizing the coefficient dynamic range during quantization; the specified output width along with the full precision width for comparison; the calculated cycle-latency value; and the latency delta from the previous major revision of the core.

Using the FIR Compiler IP Core

The CORE Generator GUI performs error-checking on all input parameters. Resource estimation, implementation details, and filter analysis are also available.

Several files are produced when a core is generated, and customized instantiation templates for Verilog and VHDL design flows are provided in the .veo and .vho files, respectively. For detailed instructions, see the CORE Generator software documentation.

Simulation Models

The core has a number of options for simulation models:

- VHDL behavioral model in the xilinxcorelib library
- VHDL UniSim-based structural simulation model
- Verilog UniSim-based structural simulation model

The models required may be selected in the CORE Generator project options.

Xilinx recommends that simulations utilizing UniSim-based structural models are run using a resolution of 1 ps. Some Xilinx library components require a 1 ps resolution to work properly in either functional or timing simulation. The UniSim-based structural simulation models may produce incorrect results if simulated with a resolution other than 1 ps. See the "Register Transfer Level (RTL) Simulation Using Xilinx Libraries" section in *Chapter 6 of the Synthesis and Simulation Design Guide* for more information. This document is part of the ISE® Software Manuals set available at www.xilinx.com/support/software_manuals.htm.

XCO Parameters

Table 5 defines valid entries for the XCO parameters. Parameters are not case sensitive. Default values are displayed in bold. Xilinx strongly suggests that XCO parameters are not manually edited in the XCO file; instead, use the CORE Generator GUI to configure the core and perform range and parameter value checking. The XCO parameters are helpful in defining the interface to other Xilinx tools.

Table 5: **XCO Parameters**

XCO Parameter	Valid Values
component_name	ASCII text using characters: a..z, 0..9 and "_" ; starting with a letter
CoefficientSource	**Vector**, COE_File
CoefficientVector	ASCII text using characters: 0..9, "." and ","
Coefficient_File	Valid file path
Coefficient_Sets	1 - 256
Filter_Type	**Single_Rate**, Interpolation, Decimation, Interpolated, Polyphase_Filter_Bank_Receiver, Polyphase_Filter_Bank_Transmitter
Rate_Change_Type	**Integer**, Fixed_Fractional
Interpolation_Rate	1 - 1024
Decimation_Rate	1 - 1024
Zero_Pack_Factor	1 - 1024
Number_Channels	1 - 64
RateSpecification	**Frequency_Specification**, Sample_Period

Table 5: **XCO Parameters *(Cont'd)***

XCO Parameter	Valid Values
SamplePeriod	1 - 10000000
Sample_Frequency	0.000001 - 600.0
Clock_Frequency	0.000001 - 600.0
Filter_Architecture	**Systolic_Multiply_Accumulate**, Transpose_Multiply_Accumulate, Distributed_Arithmetic
Coefficient_Reload	**false**, true
Coefficient_Structure	**Inferred**, Non_Symmetric, Symmetric, Negative_Symmetric, Half_Band, Hilbert
Coefficient_Sign	**Signed**, Unsigned
Quantization	**Integer_Coefficients**, Quantize_Only, Maximize_Dynamic_Range
Coefficient_Width	1 - 49; Default is **16**
BestPrecision	**true**, false
Coefficient_Fractional_Bits	0 - 49
Data_Sign	**Signed**, Unsigned
Data_Width	1 - 49; Default is **16**
Data_Fractional_Bits	0 - 49
Number_Paths	1 - 16; Default is **1**
Output_Rounding_Mode	**Full_Precision**, Truncate_LSBs, Non_Symmetric_Rounding_Down, Non_Symmetric_Rounding_Up, Symmetric_Rounding_to_Zero, Symmetric_Rounding_to_Infinity, Convergent_Rounding_to_Even, Convergent_Rounding_to_Odd
Output_Width	1 - 89
Allow_Rounding_Approximation	**false**, true
Registered_Output	**true**, false
Optimization_Goal	**Area**, Speed
Has_SCLR	**false**, true
Has_CE	**false**, true
Has_ND	**false**, true
Has_Data_Valid	**false**, true
SCLR_Deterministic	**false**, true
UseChan_in_adv	**false**, true
Chan_in_adv	0 - 63; Default is **0**
Data_Buffer_Type	**Automatic**, Block, Distributed, Not_Applicable
Coefficient_Buffer_Type	**Automatic**, Block, Distributed, Not_Applicable
Input_Buffer_Type	**Automatic**, Block, Distributed, Not_Applicable
Output_Buffer_Type	**Automatic**, Block, Distributed, Not_Applicable
Preference_For_Other_Storage	**Automatic**, Block, Distributed, Not_Applicable
Multi_Column_Support	**Disabled**, Automatic, Custom
Inter_Column_Pipe_Length	1 - 16; Default is **4**
ColumnConfig	ASCII text using characters: 0..9 and ","

Migrating to FIR Compiler v5.0 from Earlier Versions

Updating from FIR Compiler v4.0 and v3.2

The CORE Generator core update feature may be used to update an existing FIR Compiler XCO file to version 5.0 of the FIR Compiler core. The core may then be regenerated to create a new netlist. See the CORE Generator documentation for more information on this feature.

Port Changes

There are no differences in port naming conventions, polarities, priorities or widths between versions.

Updating from Versions prior to FIR Compiler v3.2

It is not currently possible to automatically update versions of the FIR Compiler core prior to v3.2. Xilinx recommends that customers use the FIR Compiler v5.0 GUI to customize a new core.

System Generator for DSP Graphical User Interface

This section describes each tab of the System Generator GUI and details the parameters that differ from the CORE Generator GUI. See CORE Generator Graphical User Interface for detailed information about all other parameters.

Tab 1: Filter Specification

The Filter Specification tab is used to define the basic filter configuration as on the Filter Specification Screen of the CORE Generator GUI.

- **Coefficients:** This field is used to specify the coefficient vector as a single MATLAB® software row vector. The number of taps is inferred from the length of the MATLAB row vector. It is possible to enter these coefficients using the MATLAB FDATool block. Multiple coefficient sets must be concatenated into a single vector as described in the Multiple Coefficient Sets section.
- **Hardware Oversampling Specification format:** Selects which method is to be used to specify the hardware oversampling rate. This value directly affects the level of parallelism of the core implementation and resources used. When "Maximum Possible" is selected, the core uses the maximum oversampling given the sample period of the signal connected to `DIN` port. When "Hardware Oversampling Rate" is selected, the user can specify the oversampling rate. When "Sample Period" is selected, the core clock is connected to the system clock and the value specified for the Sample period parameter sets the input sample rate that the core supports. The Sample period parameter also determines the hardware oversampling rate of the core. When "Sample Period" is selected, the core is forced to use the `ND` control port. See the Interface, Control, and Timing section for more details on the core control ports.
  - **Sample Period**: Specifies the input sample period supported by the core.
  - **Hardware Oversampling Rate**: Specifies the hardware oversampling rate to be applied to the core.

See Filter Specification Screen for information about the other parameters on this tab.

Tab 2: Implementation

The Implementation tab is used to define implementation options; see the Implementation Options Screen of the CORE Generator GUI for details of all the core parameters on this tab.

- **FPGA Area Estimation:** See the System Generator documentation for detailed information about this section.

See the Implementation Options Screen for information about the other parameters on this tab.

Tab 3: Detailed Implementation

See Detailed Implementation Options Screen for the corresponding CORE Generator GUI screen.

- **rst:** Specifies if the core is to have a reset pin (the equivalent of selecting the SCLR option in the CORE Generator GUI).
- **nd:** Specifies if the core is to have a New Data port (the equivalent of selecting the ND Port option in the CORE Generator GUI). This control pin is only available when "Sample Period" has been selected for the Hardware Oversampling Specification format.
- **en:** Specifies if the core is to have a clock enable pin (the equivalent of selecting the CE option in the CORE Generator GUI).

See Detailed Implementation Options Screen for information about the other parameters on this tab.

Core Use through System Generator

The FIR Compiler core is available through Xilinx System Generator for DSP, a design tool that enables the use of The MathWorks model-based design environment Simulink® for FPGA design. The FIR Compiler core is one of the DSP building blocks provided in the Xilinx blockset for Simulink. The core can be found in the Xilinx Blockset in the DSP section. The block is called "FIR Compiler v5.0." See the System Generator User Manual for more information.

Core Features

Filter Architectures

Multiply-Accumulate

Figure 4 illustrates a simplified view of a MAC-based FIR utilizing a single MAC engine.

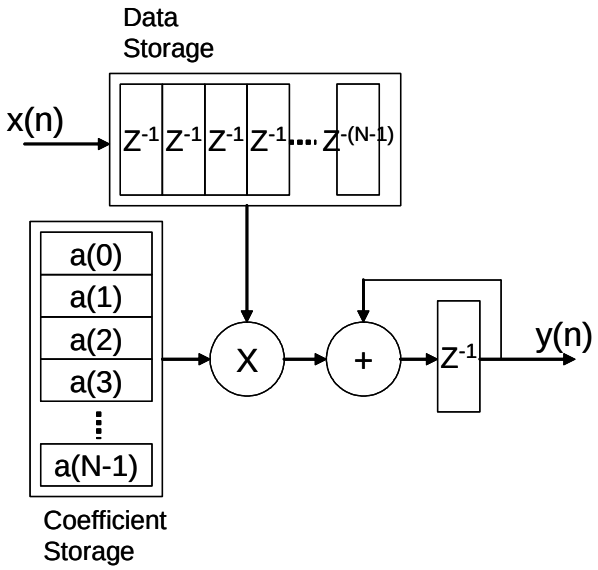


Figure 4: **Single MAC Engine Block Diagram**

The single implementation is extensible to multi-MAC implementations for use in achieving higher performance filter specifications (larger numbers of coefficients, higher sample rates, more channels, etc.).

The number of multipliers required to implement a filter is determined by calculating the number of multiplies required to perform the computation (taking into account symmetrical and half-band coefficient structures and sample rate changes) and then dividing by the number of clocks available to process each input sample. The available clock cycles value is always rounded down and the number of multipliers rounded up to the nearest integer. If there is a non-zero remainder, some of the MAC engines calculate fewer coefficients than others, and the coefficients are padded with zeros to accommodate the excess cycles.

Note that the output samples reflect the padding of the coefficient vector; for this reason, the response to an applied impulse contains a certain number of zero outputs before the first coefficient of the specified impulse response appears at the output. The core automatically generates an implementation that meets the user-defined performance requirements based on the system clock rate, the sample rate, the number of taps and channels, and the rate change. The core inserts one or more multipliers to meet the overall throughput requirements.

Two MAC architectures are available in the FIR Compiler: one that implements a Systolic filter structure and the other a Transpose filter structure.

Systolic Multiply-Accumulate

Figure 5 illustrates the Systolic Multiply-Accumulate architecture implementing a pipelined Direct-Form filter.

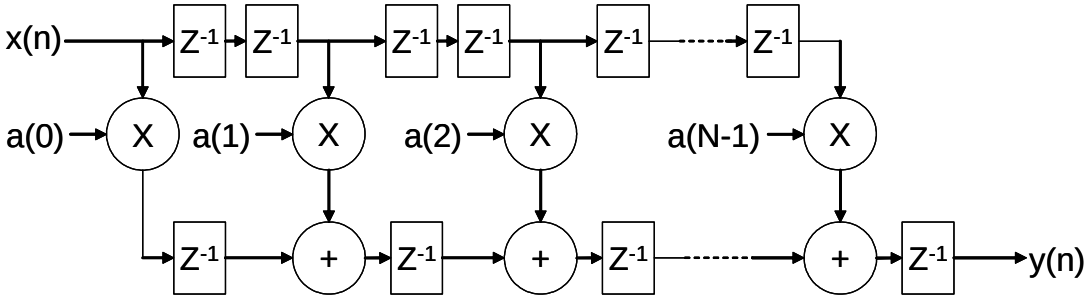


Figure 5: **Pipelined Direct-Form**

Figure 6 illustrates a multi-MAC implementation for this architecture.

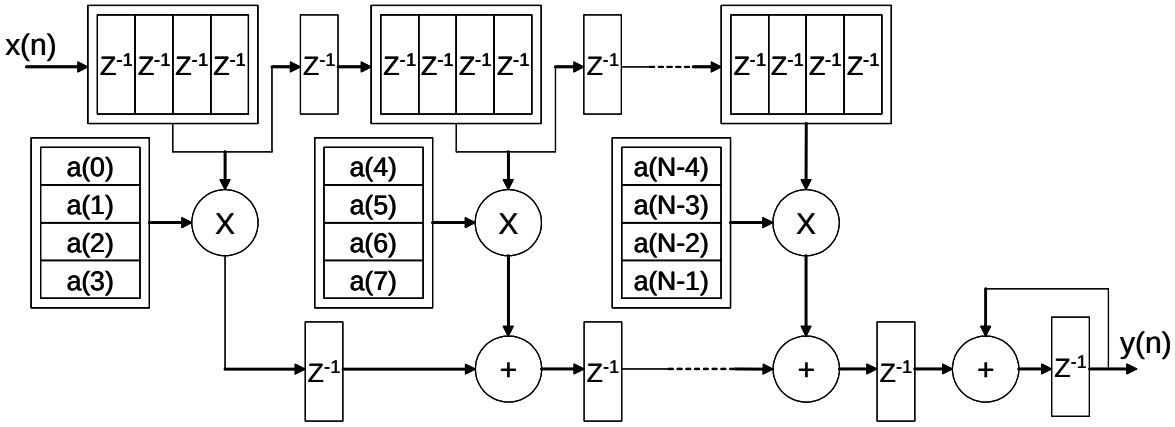


Figure 6: **Systolic Multi-MAC Implementation**

The architecture is directly supported by the XtremeDSP Slice and results in area-efficient and high performance filter implementations. The structure also extends to exploit coefficient symmetry offering further resource savings.

Transpose Multiply-Accumulate

Figure 7 illustrates the Transpose Multiply-Accumulate architecture implementing a Transposed Direct-Form filter.

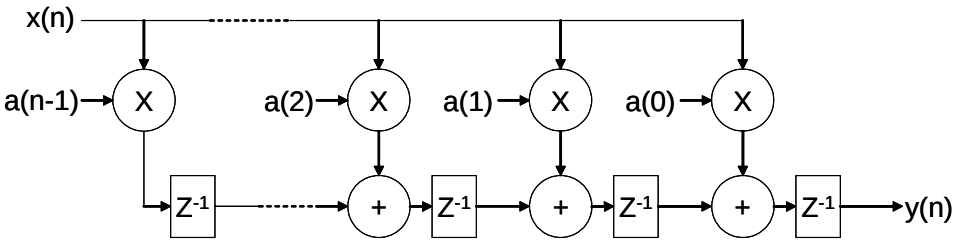


Figure 7: **Transposed Direct-Form**

Figure 8 illustrates a multi-MAC implementation for this architecture.

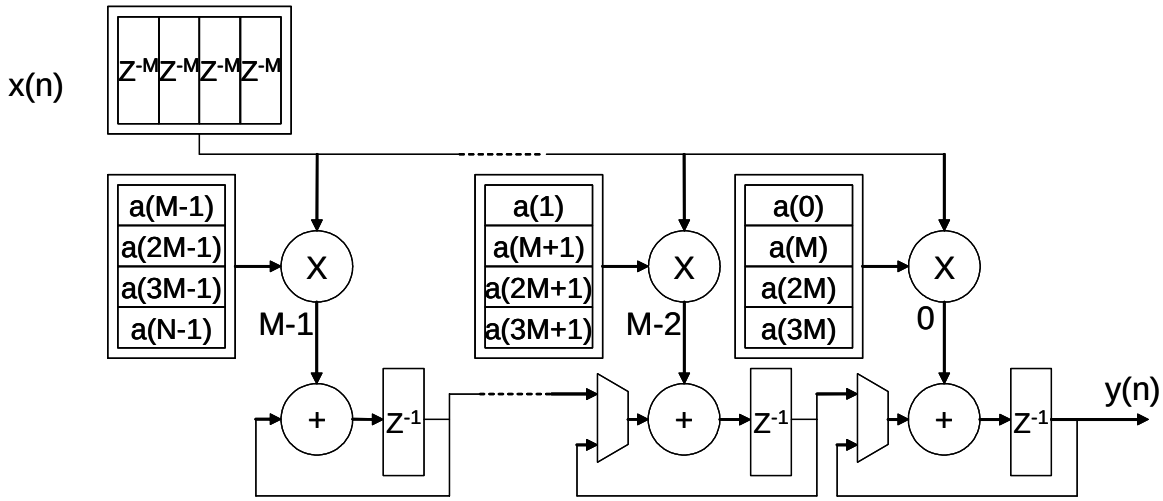


Figure 8: **Transpose Multi-MAC Implementation**

This architecture is also directly supported by the XtremeDSP Slice. This structure offers a low latency implementation, and for some configurations can also offer extra resource savings over the Systolic structure. It does not require an accumulator and can use fewer data memory resources, although it does not exploit coefficient symmetry.

Distributed Arithmetic

Figure 9 displays a simplified view of a DA FIR.

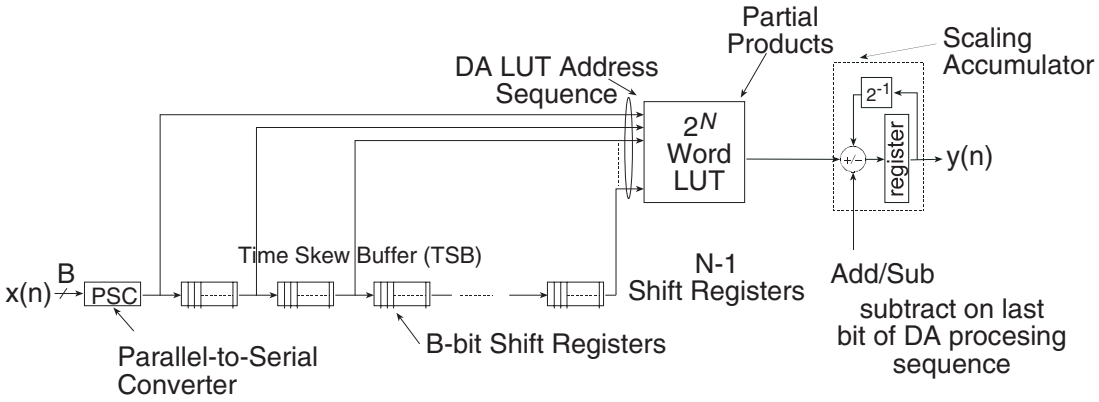


Figure 9: **Serial Distributed Arithmetic FIR Filter**

In their 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 [Ref 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 [Ref 2] over the filter coefficient space.

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 is 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 10 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 tap.

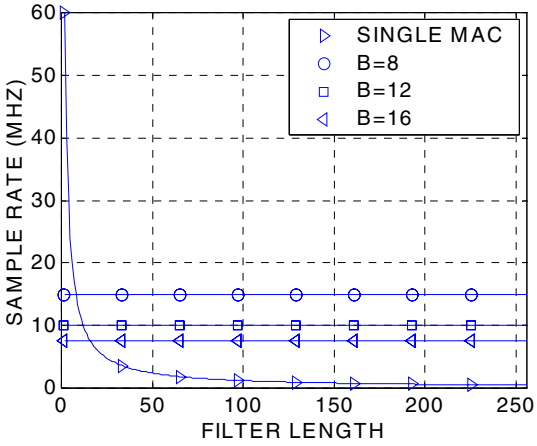


Figure 10: **Throughput (Sample Rate) Comparison of Single-MAC-based FIR and DA FIR as a Function of Filter Length. B is the DA FIR Input Sample Precision. Clock Rate is 120 MHz.**

Figure 11 provides a similar comparison for a dual-MAC architecture.

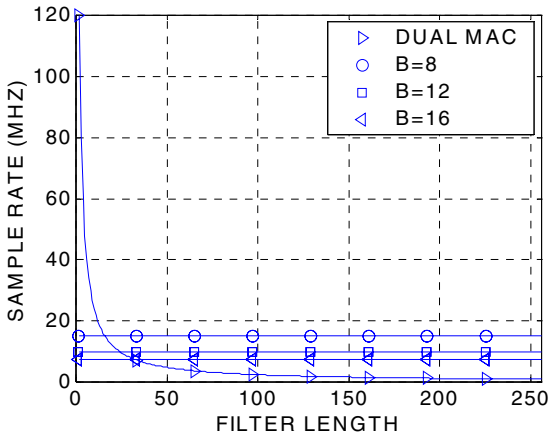


Figure 11: **Throughput (Sample Rate) Comparison of Dual-MAC-based FIR and DA FIR as a Function of Filter Length. B is the DA FIR Input Sample Precision. Clock Rate is 120 MHz.**

Increasing the Speed of Multiplication-Parallel Distributed Arithmetic

In their 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 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 can be designed that process several or 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, 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 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 in return for filter throughput.

DA Filter Throughput

The signal sample rate for a DA type 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, 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 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.

Filter Structures and Optimizations

Filter Symmetry

The impulse response for many filters possesses significant symmetry. This symmetry can generally be exploited to minimize arithmetic requirements and produce area-efficient filter realizations. Figure 12 shows the impulse response for a 9-tap symmetric FIR filter.

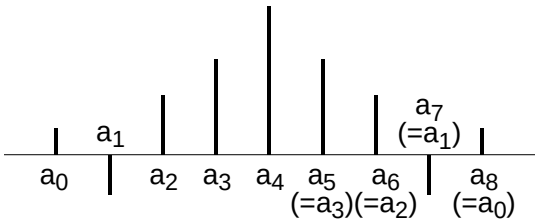


Figure 12: **Symmetric FIR – Odd Number of Terms**

Instead of implementing this filter using the architecture shown in Figure 1, the more efficient signal flow-graph in Figure 13 can be used. In general, the former approach requires *N* multiplications and (*N*-1) additions. In contrast, the architecture in Figure 13 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.

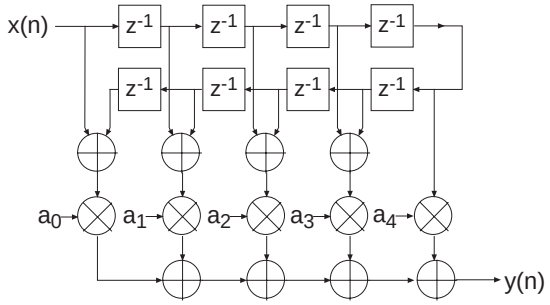


Figure 13: **Exploiting Coefficient Symmetry – Odd Number of Filter Taps**

Coefficient symmetry for an even number of terms can be exploited as shown in Figure 14.

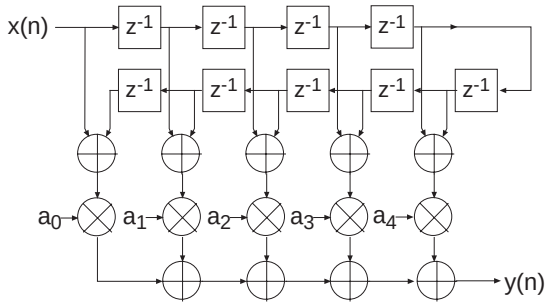


Figure 14: **Exploiting Coefficient Symmetry – Even Number of Filter Taps**

Figure 15 shows the impulse response for a negative, or odd, symmetric filter.

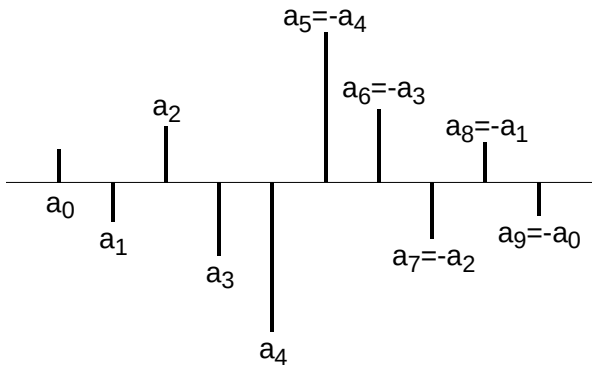


Figure 15: **Negative Symmetric Impulse Response**

This symmetry is easily exploited in a manner similar to that shown in Figure 13 and Figure 14. In this case, the middle layer of adders are replaced by subtracters, as illustrated in Figure 16.

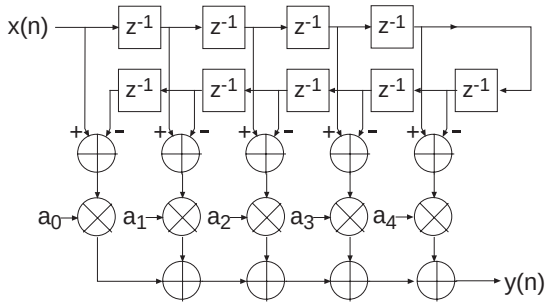


Figure 16: **FIR Architecture Exploiting Negative Symmetry**

Filter coefficient symmetry is inferred by the core GUI from the coefficient definition file. Note that this inferred value can be overridden by the user. When the structure is inferred, the inferred setting is displayed in the Summary page and in the ToolTip for the Coefficient Structure field.

Coefficient Padding

When implementing a filter with symmetric coefficients using the Multiply-Accumulate architecture, users must be aware that the core reorganizes the filter coefficients if required to exploit symmetry, and this **may alter the filter response**. This is only necessary if the core is configured such that all processing cycles are not utilized. For example, when the core has four cycles to process each sample for a 30-tap symmetric response filter, the core pads the coefficient storage out as illustrated in Figure 17.

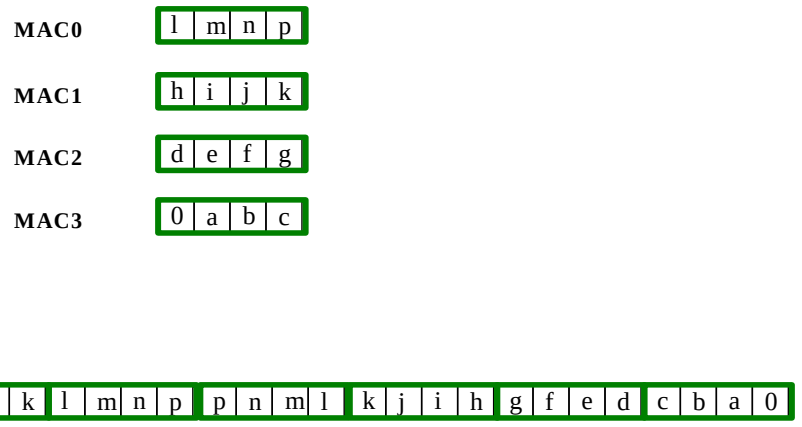


Figure 17: **Filter Padding to Facilitate Symmetric Structure Exploitation**

The appended zeroes after the non-zero coefficients do not affect the filter response, but the prepended zero coefficients do alter the phase response of the filter implementation when compared to the ideal coefficients. There are two ways to avoid this issue: First, and simplest, the user can force the Coefficient Structure to be Non-Symmetric. This avoids the issue of prepending zero coefficients to the coefficient vector, and only appended zeroes are used to pad out the filter response to the required number of cycles. Second, and more efficient, the user can increase the number of taps implemented by the filter **at little or no cost in resource usage.** In the previous example, the filter could process 32 taps in the same time, with the same hardware resources, and with the same cycle latency as the 30-tap implementation, and the phase response of the 32-tap filter would be unaltered.

The core GUI displays the actual number of coefficients calculated on the Implementation Details tab. Users can use this information to determine if they can increase the number of coefficients used by their filter definition.

Single-rate FIR Filter

The basic FIR filter core is a single-rate (input sample rate = output sample rate) finite impulse response filter. This is the simplest of filter types and is the default at the start of parametrization in the CORE Generator software.

Half-band FIR Filter

Figure 18 illustrates the general frequency response for a half-band filter.

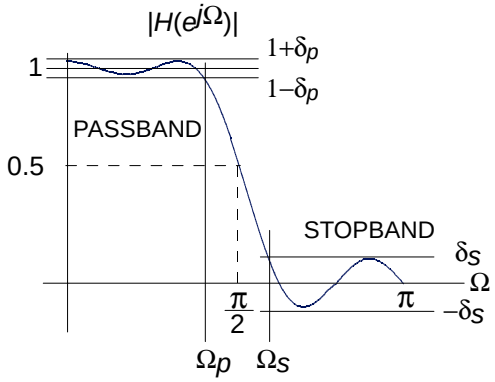


Figure 18: **Half-band Filter Magnitude Frequency Response**

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, δ_p and δ_s respectively, are equal $\delta_p = \delta_s$. These properties are reflected in the filter impulse response. It can be shown [Ref 5] that approximately half of the filter coefficients are zero for an odd number of taps, as illustrated in Figure 19 for an 11-tap half-band filter.

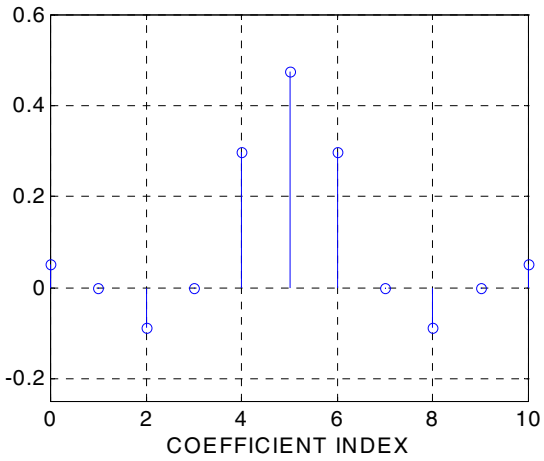


Figure 19: **Half-band Filter Impulse Response**

The interleaved zero values in the coefficient data can be exploited to realize an efficient realization, as shown in Figure 20.

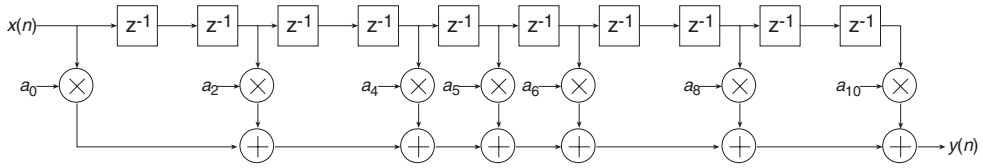


Figure 20: **Half-band Filter Impulse Response**

The half-band filter selection in the compiler is intended for this purpose. This filter is available in the *Coefficient Structure* 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.

Hilbert Transform

Hilbert transformers [Ref 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 [Ref 5] that the impulse response corresponding to this frequency domain characteristic is odd-symmetric and has interleaved zeros as shown in Figure 20. 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 21. 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 22. 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.

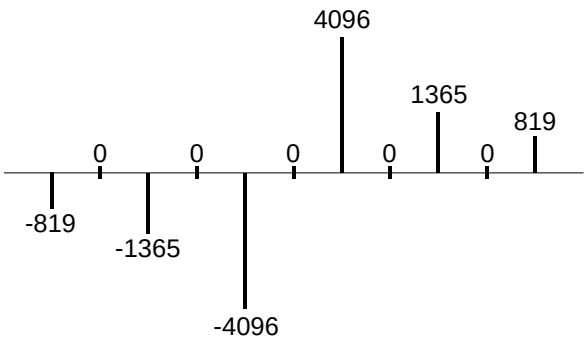


Figure 21: **Hilbert Transformer Impulse Response**

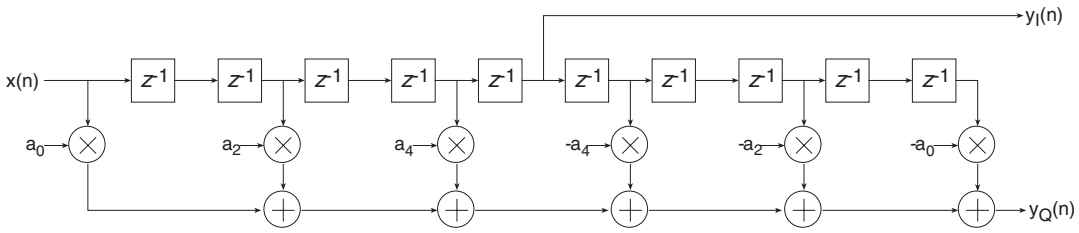


Figure 22: **Hilbert Transformer FIR Filter Realization**

Figure 23 shows the architecture for a Hilbert transformer that exploits both the zero-valued and the negative symmetry characteristics of the impulse response.

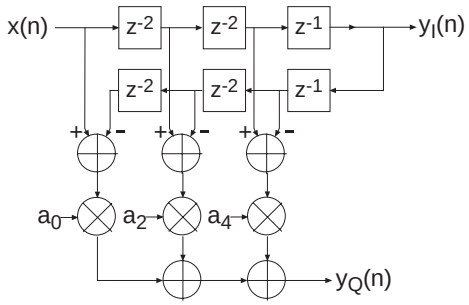


Figure 23: **Hilbert Transformer Exploiting Zero-valued Filter Coefficients and Negative Symmetry**

Interpolated FIR Filter

An *interpolated FIR* (IFIR) filter [Ref 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. Figure 24 illustrates a *N*-tap IFIR filter. This architecture is functionally equivalent to inserting *k*-1 zeros between the coefficients of a prototype filter coefficient set.

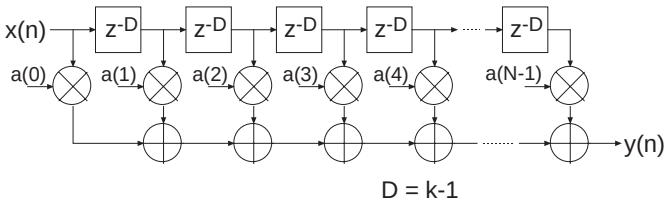


Figure 24: **Interpolated FIR (IFIR). The Zero-packing Factor is *k*.**

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 [Ref 4] and [Ref 6] provide the details of how these systems are realized, and how to design the IFIR and the image rejection filters.

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

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. There is no inherent rate change when using an interpolated filter – the input rate is the same as the output rate.

Polyphase Decimator

Figure 25 illustrates the polyphase decimation filter option which implements the computationally efficient *M*-to-1 polyphase decimating filter.

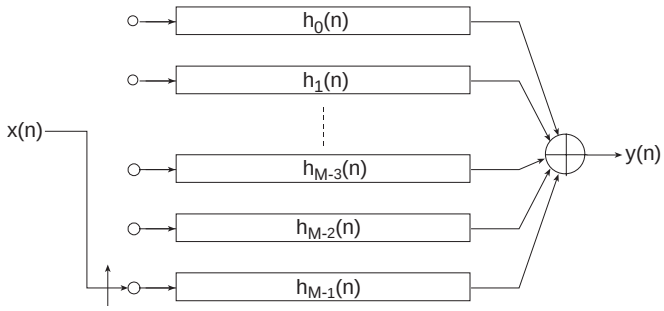


Figure 25: ***M*-to-1 Polyphase Decimating Filter**

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

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

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 the sample rate of the input data stream $x(n),\ n = 0, 1, 2, \ldots$.

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 is performed per output point.

Polyphase Interpolator

Figure 26 illustrates the polyphase interpolation filter option which implements the computationally efficient 1-to-*P* interpolation filter.

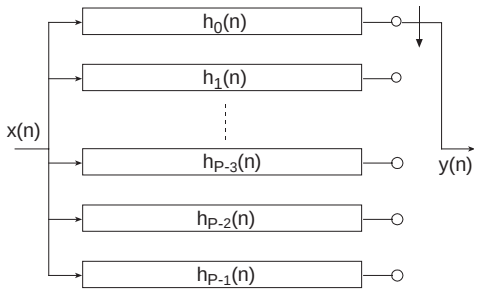


Figure 26: **1-to-*P* Polyphase Interpolator**

A set of N prototype filter coefficients $a_0, a_1, \ldots, a_{N-1}$ is mapped to the P polyphase sub-filters $h_0(n), h_1(n), \ldots, h_{p-1}(n)$ according to Equation 2, as in the decimation case.

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 26.

The output sample f_s' rate is $f_s' = f_s P$ where f_s is the sample rate of the input data stream $x(n), n = 0, 1, 2, \ldots$. 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.

Polyphase Interpolator Exploiting Symmetric Pairs

The *symmetric pairs* technique [Ref 8] is used to exploit coefficient symmetry when implementing an Interpolation filter in the Systolic Multiply-Accumulator architecture. When P polyphase sub-filters are generated from symmetric filter coefficients, not all the sub-filters contain a set of coefficients that are themselves symmetric. The symmetric pairs technique observes that by adding and subtracting two corresponding non-symmetric phases produces two new phases containing symmetric coefficients.

The following example demonstrates this technique for a 15-tap interpolate by 3 filter. The filter coefficients:

a , b , c , d , e , f, g , h , g , f , e , d , c , b , a

Produce the following sub-filters:

h_0 = a , d , g , f , c

h_1 = b , e , h , e , b

h_2 = c , f , g , d , a

Sub-filters h_0 and h_2 are not symmetric. Applying the symmetric pairs technique produces the following sub-filters:

h_0 = a+c , d+f , g+d , f+d , c+a

h_1 = b , e , h , e , b

h_2 = c-a , f-d , g-h , d-g , a-c

Now both h_0 and h_2 are symmetric with h_2 being negative symmetric. The filter can now be implemented utilizing symmetry, giving the associated resource savings. The output from sub-filters h_0 and h_2 must be added and subtracted and then scaled by a factor of 0.5 to produce the original filter output. Figure 27 illustrates the resulting structure.

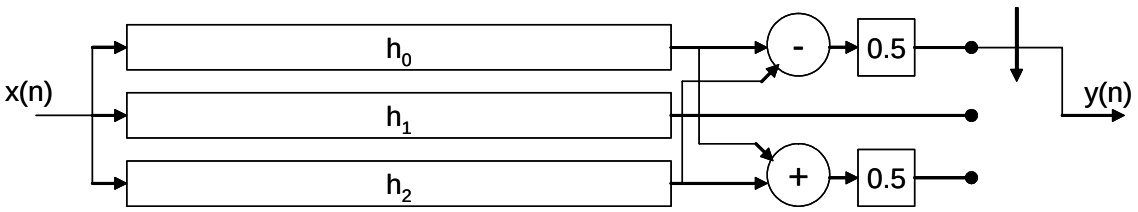


Figure 27: **Symmetric Pairs**

Note: When interpolating by 2 with an odd number of symmetric coefficients, this technique is not required as the resulting polyphase sub-filters are symmetric.

Coefficient Padding

As with the general symmetric filter case, if the combination of rate and number of filter taps results in a sub-filter which is not fully populated with coefficients, the reorganization of the filter coefficients results in a change in the phase response of the filter. The impulse response is shifted by a number of output samples as a result. In the 14 tap, interpolate by 4 case, padding a zero coefficient to the front of the coefficient response would be required to align the phases such that symmetry can be exploited, resulting in a smaller implementation, but this results in a different phase response for the filter. The methods to avoid this change in response, if such a change cannot be accommo-

dated in the user's application system, are also similar to the general symmetry case – the user can either force non-symmetric structure implementation or make use of the extra coefficients which can be supported in the structure. Figure 28 illustrates several example cases in and is extensible to larger filters.

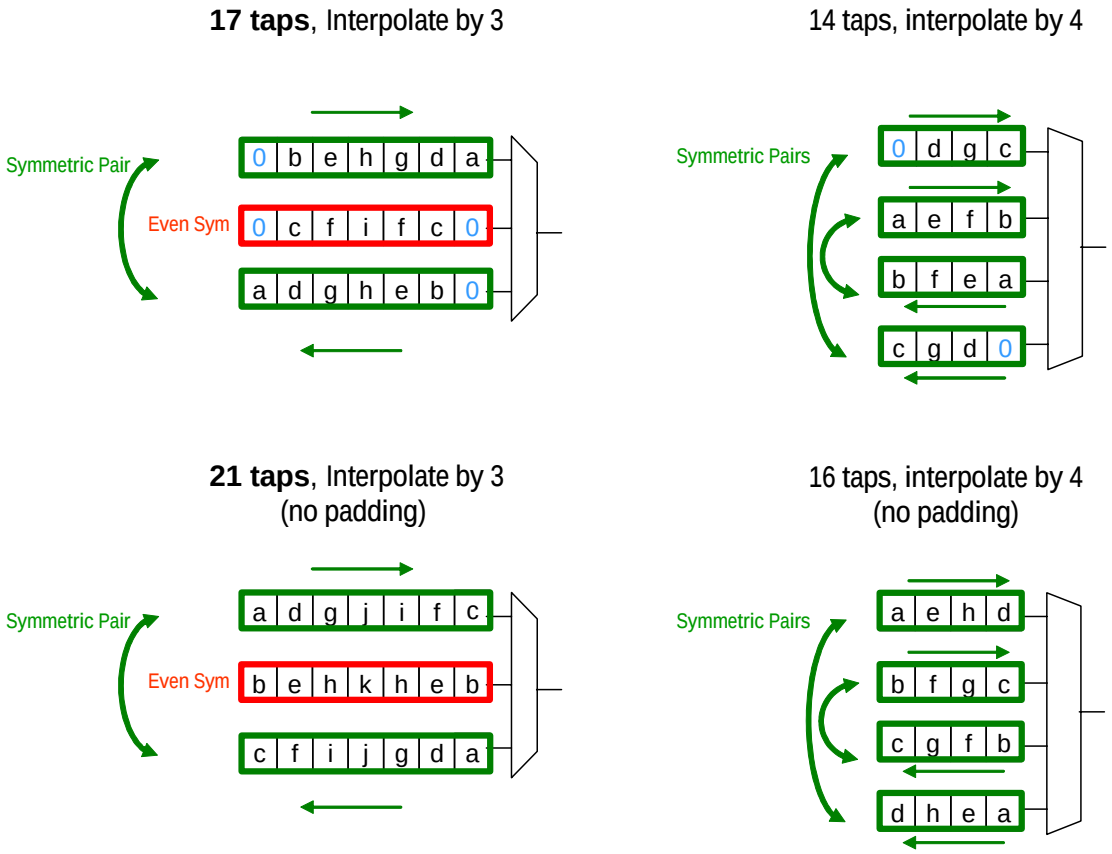


Figure 28: **Filter Padding to Facilitate Symmetric Pairing**

Half-band Decimator

The half-band decimator is a polyphase filter with an embedded 2-to-1 down-sampling of the input signal. Figure 29 illustrates the structure.

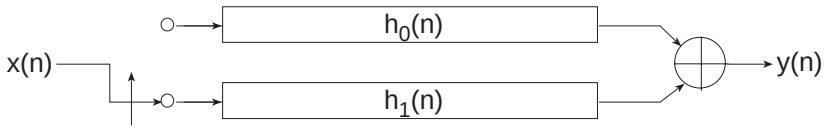


Figure 29: **Half-band Decimation Filter**

The filter is very similar to the polyphase decimator described in Polyphase Decimator 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 18 and Figure 19, respectively. Observe the alternating zero-valued coefficients in the impulse response. Figure 29 details a 7-tap half-band polyphase filter when the coefficients are allocated to the two polyphase segments $h_0(n)$ and $h_1(n)$ shown in Figure 29. Figure 30 (a) is the filter impulse response; note that $a_1 = 0 = a_5$. Figure 30 (b) provides a detailed illustration of the poly-

phase 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 30 (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. In this figure, note that the high density of zero-valued filter coefficients is exploited in the FPGA realization to produce a minimal area implementation.

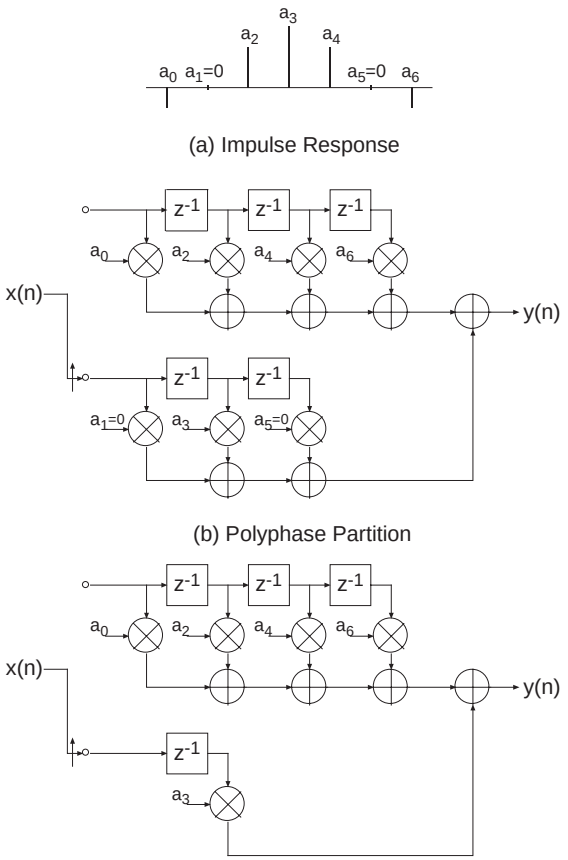


Figure 30: **7-Tap Half-band Decimation Filter**

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. Figure 31 displays the half-band interpolator.

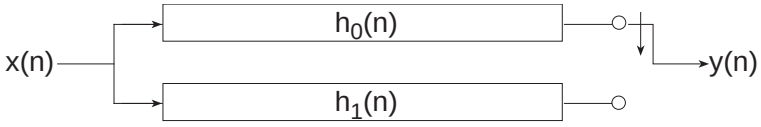


Figure 31: **Half-band Interpolation Filter**

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 32. Figure 32(a) is the impulse response, Figure 32(b) shows the polyphase partition, and Figure 32(c) is the optimized architecture that has taken full advantage of the 0 entries in the coefficient data. Note that the high density of zero-valued filter coefficients is exploited in the FPGA realization to produce a minimal area implementation.

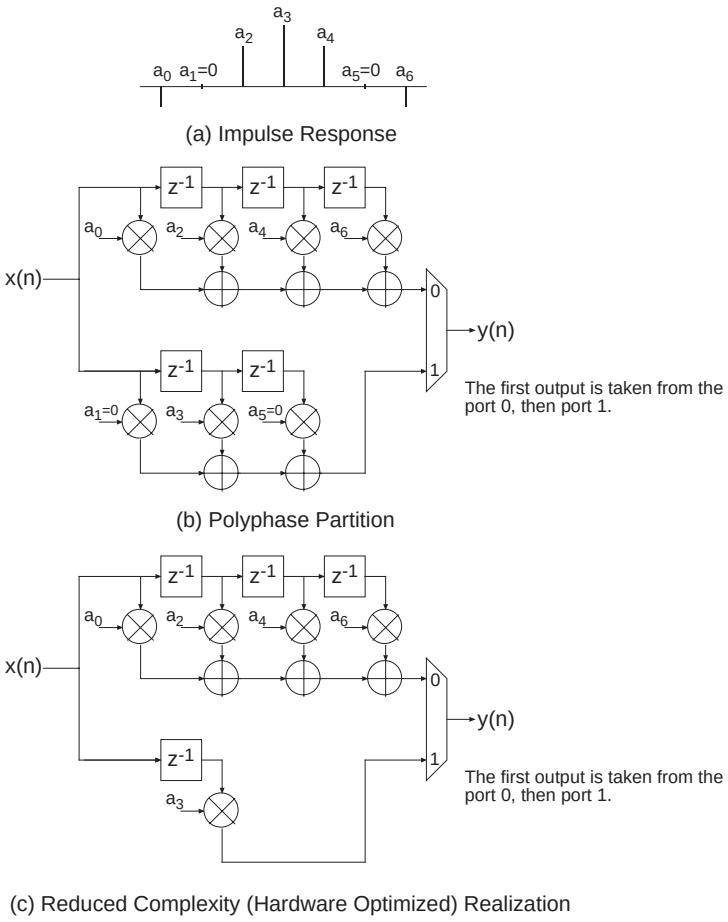


Figure 32: **7-Tap Half-band Interpolation Filter**

Small Non-zero Even Terms in a Half-band Filter Impulse Response

Certain filter design software can result in small non-zero values for the odd terms in the half-band filter impulse response. In this situation, it can 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 previously are not possible. If the small non-zero value terms cannot be ignored, the general polyphase decimator or interpolator described in Polyphase Decimator and "Polyphase Interpolator," using a rate change of two, is more appropriate.

Fixed Fractional Rate Resampling Filters

FIR filters that implement resampling of a data stream at a fixed fractional rate P/Q, where P and Q are integers up to 64, are available for the Systolic Multiply-Accumulate architecture. In Figure 33, the operation of an interpolation filter with interpolation rate P=5 is contrasted conceptually with the operation of a fixed fractional rate filter with rate P/Q=5/3.

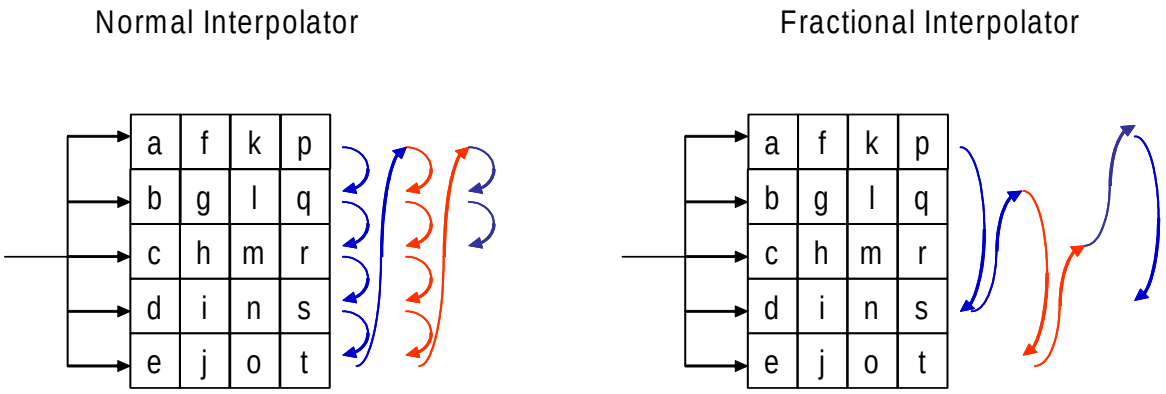


Figure 33: **Interpolation Filters for Integer and Fractional Rates**

The normal (integer rate) interpolator passes the input sample to all P phases and then produces an output from each of the phase arms of the polyphase filter structure. In the fractional rate version, the output is taken from a phase arm which varies according to a stepping sequence with step size Q.

Figure 34 illustrates a similar conceptual method for implementing fractional rate decimators. The integer decimation rate for the left-hand diagram is Q=5, while the fractional-rate illustrated on the right is P/Q=3/5.

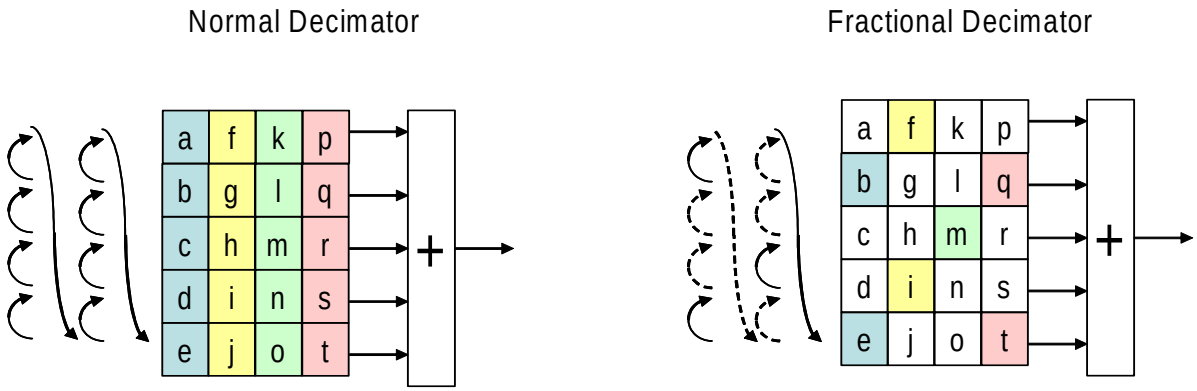


Figure 34: **Decimation Filters for Integer and Fractional Rates**

The integer rate decimator passes the input samples in sequence to each of the Q phase arms in turn, with the data being shifted through the filter, and the output is generated from the summation of the outputs from each phase arm of the polyphase filter. For the fractional rate implementation, the filter passes the input samples to phases in a stepping sequence based on a step size of P, with zero samples being placed into the skipped phases. The summation across the various phase arms remains the same, but is based on fewer actual calculations. The implementation details differ somewhat from these conceptual illustrations, but the resulting behavior of the filter is the same. Symmetry is not currently exploited when using the fractional rate structures.

Polyphase Filter Bank for Channelizer implementations

The polyphase channelizer is an efficient architecture to implement a multi-channel digital receiver (or transmitter) for a set of frequency division multiplexed (FDM) channels that exists in a single sampled data stream **[Ref 7]**. If we consider the receiver operation, Figure 35 illustrates the input spectrum of a FDM signal containing M channels.

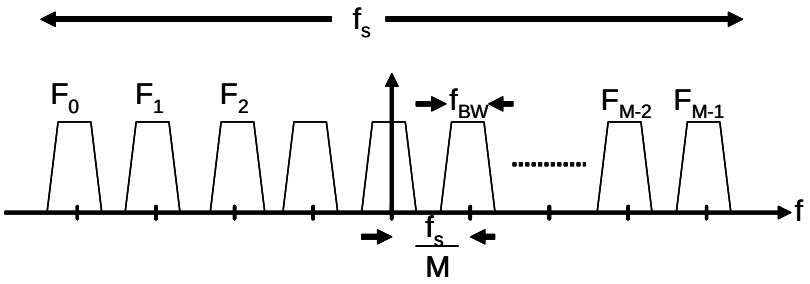


Figure 35: **Input Spectrum**

Figure 36 illustrates a conventional channelizer consisting of an independent mixer, baseband filter, and down sampler per channel.

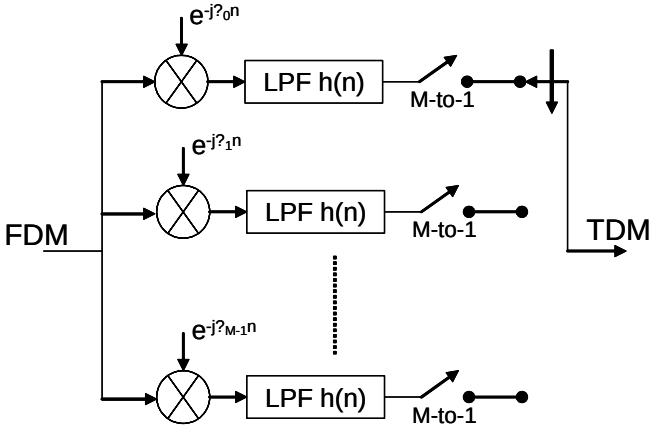


Figure 36: **Conventional Channelizer**

Figure 37 illustrates the polyphase filter bank channelizer. Reference [Ref 7] reviews the conversion process from the conventional channelizer to the polyphase filter bank implementation.

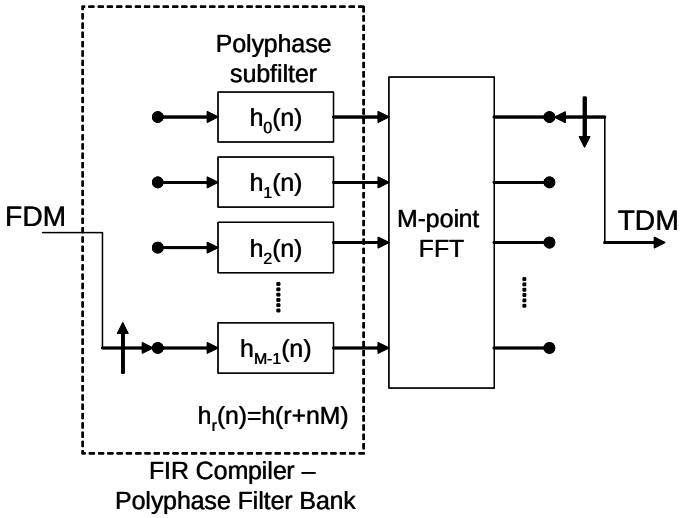


Figure 37: **Polyphase Filter Bank Channelizer**

The polyphase filter bank implementation offers significant resource savings due to the computational efficiency of the FFT algorithm and the polyphase partitioning of the original low pass prototype filter, h(n). The FIR Compiler implementation of the polyphase filter bank is directly compatible with the Xilinx LogiCORE IP Fast Fourier Transform.

The FIR Compiler output has been formatted so it can be connected directly to the Xilinx FFT core for use in a receiver. For use in a transmitter, the FIR Compiler accepts input data in a block-based format as generated by the Xilinx FFT core. Further information about the core input and output format and timing can be found in the Polyphase Filter Bank section of "MAC-based FIR Filter Timing."

Figure 38 shows the pin connections that should be made between the FIR Compiler and the Xilinx FFT core.

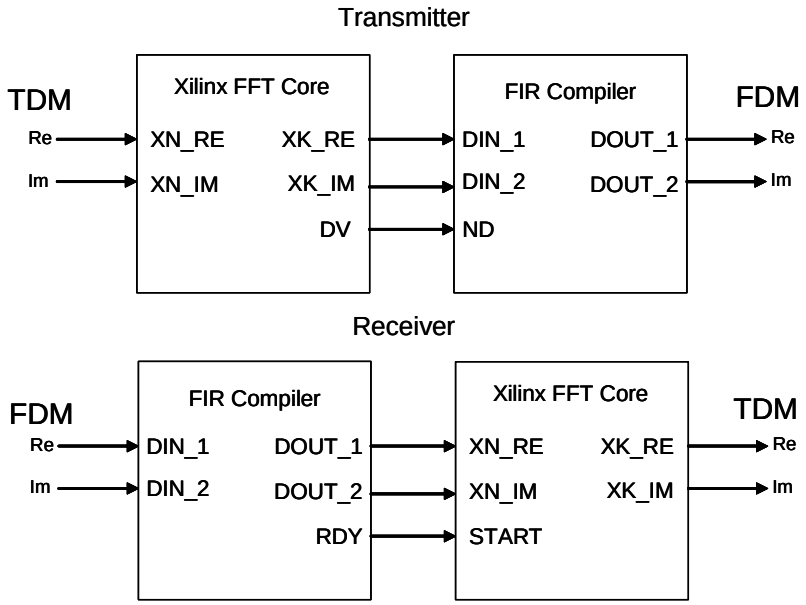


Figure 38: **Pin Connections between FIR Compiler and Xilinx FFT Core**

Filter Coefficient Data

The filter coefficients are supplied to the FIR 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 39 for an *N*-tap filter.

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

Figure 39: **Filter Coefficient File Format**

The filter coefficients can 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. Alternatively, the coefficients can be entered as real numbers (specified to a minimum of one decimal place) in base-10 only. Note that if the user enters signed negative symmetric hexadecimal coefficients, each value should be sign-extended to the boundary of the most significant nibble or hex character. This ensures that coefficient structure inference can be performed correctly (this includes Hilbert transform filter types, which are also negative symmetric).

The coefficient values can also be placed on a single line as shown in Figure 40.

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

Figure 40: **Filter Coefficient File Format – Coefficient Data on a Single Line**

Single-rate 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 41:

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

Figure 41: **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 42 is presented in Figure 43.

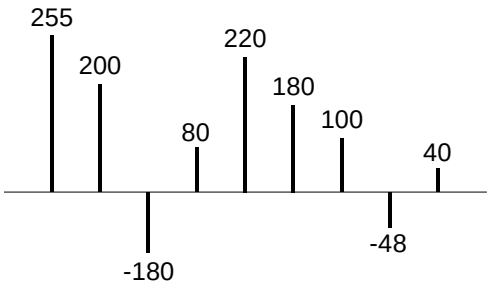


Figure 42: **Non-symmetric Impulse Response**

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

Figure 43: **Coefficient File for the Non-symmetric Impulse Response**

The coefficient file for the negative-symmetric filter characterized by the impulse response in Figure 44 is shown in Figure 45.

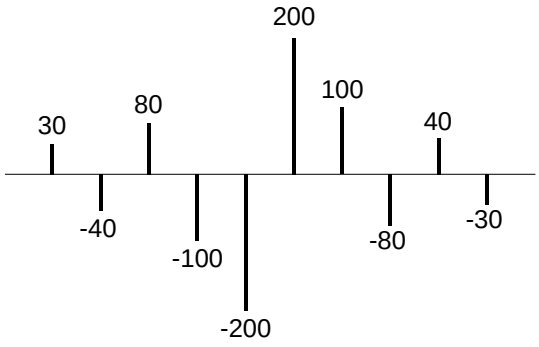


Figure 44: **Symmetric Impulse Response**

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

Figure 45: **Coefficient File for the Symmetric Impulse Response**

Half-band Filter

As previously described, 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 46 is shown in Figure 47.

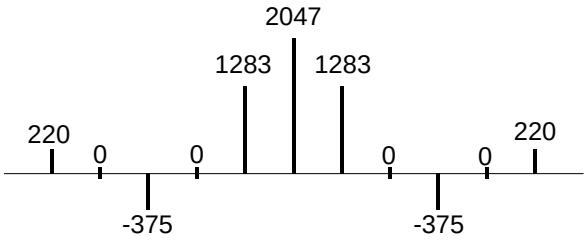


Figure 46: **11-Tap Half-band Filter Impulse Response**

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

Figure 47: **Coefficient File for the Half-band Filter Impulse Response**

The filter coefficient set is parsed by the FIR Compiler. If either the alternating zero entries are absent or the coefficient set is not even-symmetric, this condition 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 48. However, observe that these values do not contribute to the result.

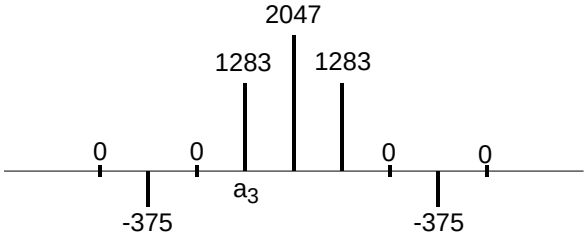


Figure 48: **9-Tap Half-band Filter Impulse Response**

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 can 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 49. 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 49 is shown in Figure 50.

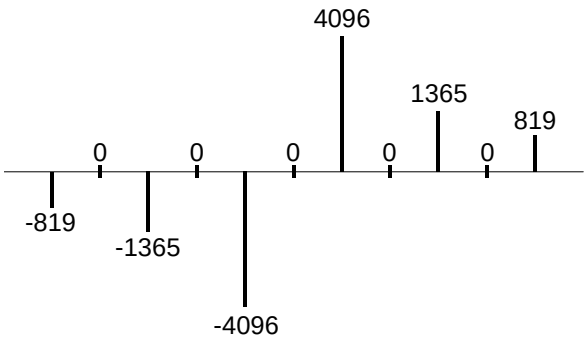


Figure 49: **Hilbert Transform Impulse Response**

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

Figure 50: **Coefficient File for the Hilbert Transformer Impulse Response**

In practice, some optimization methods used for designing a Hilbert transform can lead to the presence of small even-numbered coefficients. If the *Hilbert Transform* filter class is used in the FIR 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 can 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 is 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 51.

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

Figure 51: **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 52.

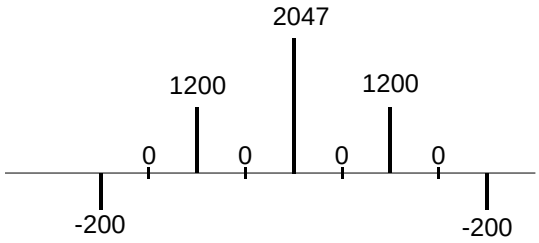


Figure 52: **Equivalent IFIR Impulse Response for the Coefficient Data Shown in Figure 51 with a Zero-packing Factor *k*=2**

If the zero-packing factor is changed to *k*=3, the impulse response is as shown in Figure 53.

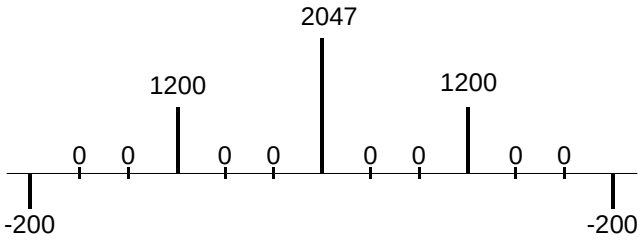


Figure 53: **Equivalent IFIR Impulse Response for the Coefficient Data Shown in Figure 51 with a Zero-packing Factor *k*=3**

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

Multiple Coefficient Sets

For multiple coefficient filters, a single .coe file is used to specify the coefficient sets. Each coefficient set should be appended to the previous set of coefficients.

For example, if a 2-coefficient set, 10-tap symmetric filter was being designed and coefficient set #0 was: `coef data = -1, -2, -3, 4, 5, 5, 4, -3, -2, -1;`

and coefficient set #1 was:

```
coefdata = -9, -10, -11, 12, 13, 13, 12, -11, -10, -9;
```

then the .coe file for the entire filter would be:

```
radix = 10;
coefdata = -1, -2, -3, 4, 5, 5, 4, -3, -2, -1, -9, -10, -11, 12, 13, 13, 12, -11, -10, -9;
```

All coefficients sets in a multiple set implementation must exhibit the same symmetry. For example, if even one set of a multi-set has non-symmetric coefficient structure, then all sets are implemented using that structure. All coefficient sets must also be of the same vector length. If one coefficient set has fewer coefficients, it must be zero padded – either appended with zeros when non-symmetric or prepended and appended with an equal number of zeros when symmetric. See the Coefficient Padding section for further information.

Coefficient Specification Using Non-integer Real Numbers

As indicated previously, the user can specify the coefficient values as non-integer real numbers, with the radix set to 10. For example:

```
radix = 10;
coefdata = 0.08659436542927, 0.00579513928555, -0.06734424313287, -0.04031582111240;
```

The coefficients are then quantized by the core to produce the binary coefficient values used in the filter, based on the user's specified coefficient bit width. This allows the user to supply floating-point values derived from a chosen filter design tool and explore the costs and benefits between performance and resource usage by altering the coefficient bit width and observing the alteration in the quantified frequency response in comparison to the ideal response. The basic quantization function is selected by setting the Quantization field to Quantize_Only. See the Coefficient Quantization section for further details.

The integer values used in the filter implementation can be determined by examining the main core MIF file (<component_name>.mif) which is generated in the CORE Generator project directory. The MIF file is always in binary format.

Multiple Channel Filters

The FIR Compiler core provides support for processing multiple input sample streams using the same implementation. Each input stream is filtered using the same filter configuration (rate change, sample rate, etc.) using the currently selected filter coefficient set.

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 54. 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.

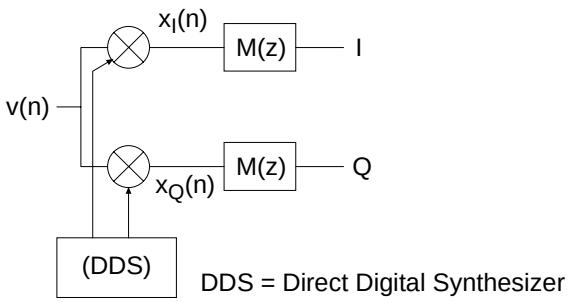


Figure 54: **Digital Down Converter**

One candidate solution to this problem is to employ two separate filters; however, this can waste logic resources. A more efficient design can be realized using a filter architecture that shares logic resources between multiple time division multiplexed (TDM) sample streams. Most filter classes supported by the filter core provide in-built support for multi-channel processing and can accommodate up to 64 TDM data streams. As more channels are processed by a the core, the sample throughput is commensurately reduced. For example, if the sample rate 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 resource utilization. A filter with two or more channels can be realized using a similar amount of logic resources to a single-channel version of the same filter, with proportionate increase in data memory requirements. The trade off 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.

Note the following limitations on multi-channel support:

- Systolic MAC implementations support up to 64 channels.
- DA implementations of single-rate filters support up to 8 channels only.
- DA implementations of multi-rate filters (polyphase decimator, polyphase interpolator, half-band decimator, and half-band interpolator) provide support for single-channel operation only.
- Transpose MAC implementations provide support for single-channel operation only.

Coefficient Reload

Coefficient Reload for DA FIR Implementations

The DA FIR implementation provides a facility for loading new coefficient data, although it is limited in that the filter operation must be halted (the filter ceases to process input samples) while the new coefficient values are loaded and some internal data structures are subsequently initialized. The coefficient reload time is a function of the filter length and type.

Figure 55 shows a high-level view of the reloadable DA FIR architecture. 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.

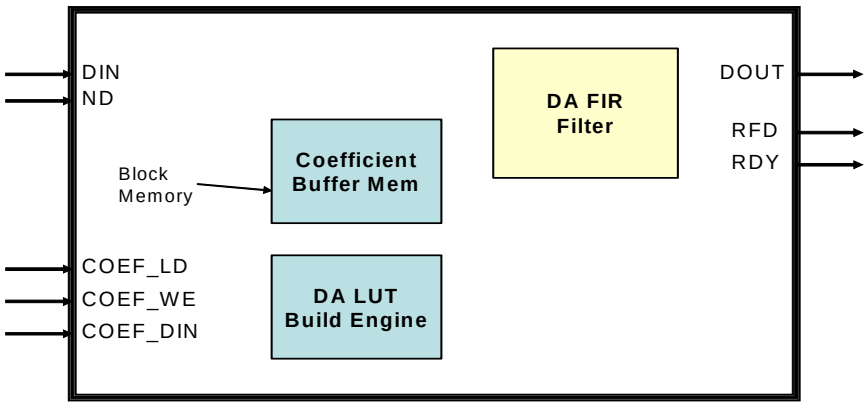


Figure 55: **High-level View of DA FIR with Reloadable Coefficients**

The signals that support the reload operation are COEF_DIN, COEF_LD, and COEF_WE. The COEF_DIN port is used to supply the new vector of coefficients to the core. COEF_LD is asserted to initiate a load operation, and COEF_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. After the load operation has completed, the DA LUT build engine is automatically started. The build engine uses the values in the coefficient buffer to re-initialize the DA LUT.

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. COEF_WE can be used to control the flow of coefficient data from the external coefficient source, for example a microprocessor, to the core. COEF_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 look-up 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 can 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.

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 6 presents the reload time (in clock cycles) for each filter class for the DA filter architecture.

An example timing diagram for DA-based filter reload operation is shown in Figure 56.

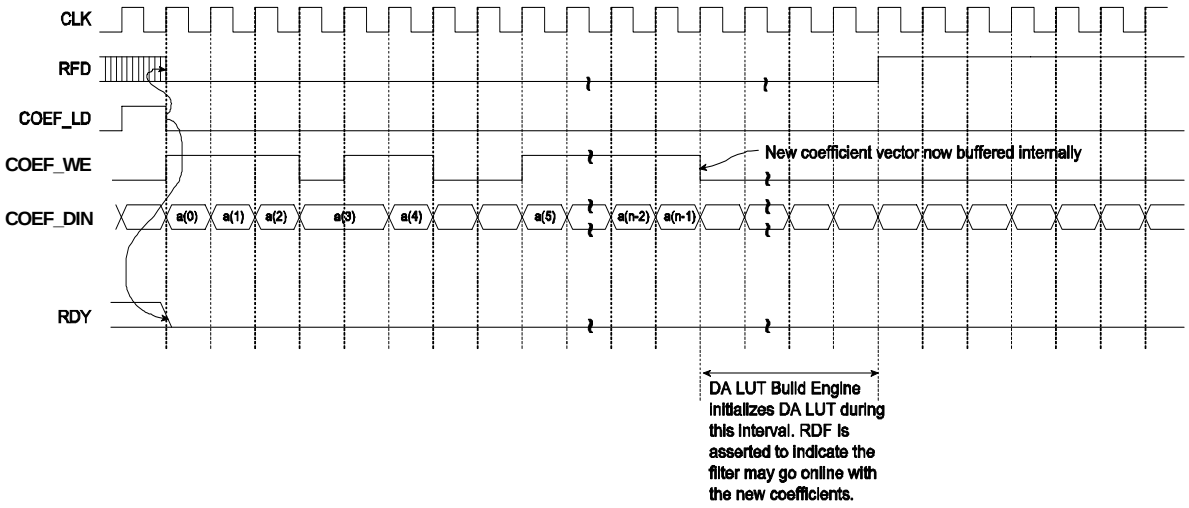


Figure 56: **Coefficient Reload Timing**

Table 6: **Coefficient Reload Times as a Function of Filter Type for DA Architectures**

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

Notes:

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

Coefficient Reload for MAC-based FIR Implementations

When a coefficient load operation is initiated for a MAC-based FIR implementation, the new vector of coefficients is written directly into the coefficient memory. The coefficient memory is split into two pages and the new vector is written into the inactive page. The active page is swapped after the last coefficient is written into the core.

The core operation is not disrupted during coefficient reload and the data buffer is not cleared following a reload. Sample processing proceeds without interruption. The timing for coefficient reload interface signals is illustrated in Figure 57.

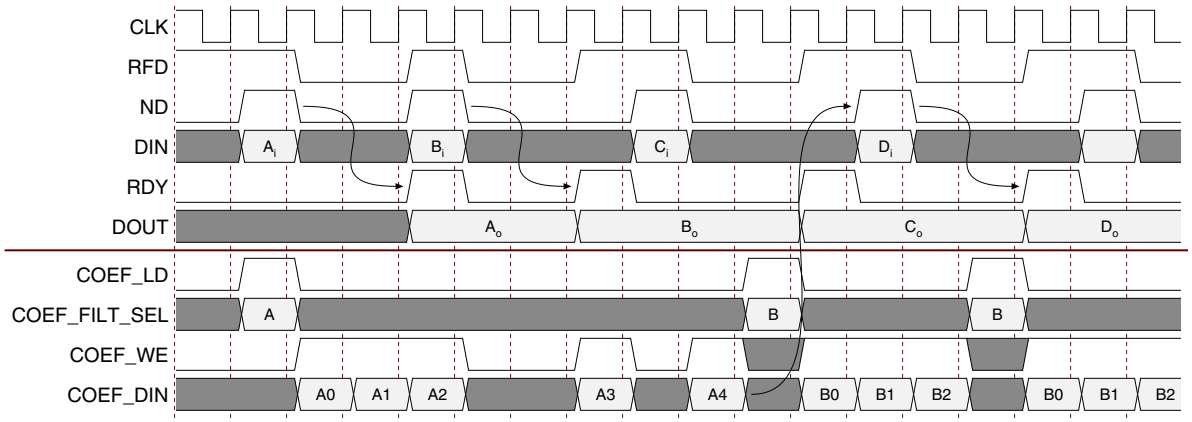


Figure 57: **Coefficient Reload Timing for Multiply-Accumulate Filters**

The number of clock cycles required to reload a coefficient vector is simply equal to the length of the reloaded coefficient vector plus one cycle. The host driving the reload port can load the coefficients over a period of as many samples as required by its application, subject to a minimum requirement equal to the length of the reloaded coefficient vector plus one cycle. The additional cycle is required for the active page to be swapped. To minimize the reload time, it is only necessary to load the first half of the coefficient vector for symmetric coefficient sets, and only non-zero coefficients for half-band or Hilbert coefficient sets.

The timing diagram indicates reloading of multiple filter sets. The `COEF_FILTER_SEL` port value is sampled when the `COEF_LD` signal is pulsed to indicate the start of a reload operation, and that is the filter which is reloaded. The switch to the reload coefficients occurs for each filter set individually. In Figure 57, filter A is reloaded with five new coefficient values. The data samples continue to be processed with the current filter set until the reload is completed (samples Ai, Bi, and Ci leading to outputs Ao, Bo, and Co), after which data samples are processed using the new coefficient set (presuming, of course, that the selected filter set has not changed during that time). After filter set A has been reloaded, the user initiates a reload of filter set B. After loading three of the five coefficients, `COEF_LD` is pulsed once more; this aborts the current reload procedure and signals the start of a new reload procedure, again to filter set B. Note that the level on `COEF_WE` is irrelevant during the `COEF_LD` pulse, as it is ignored along with any data on the `COEF_DATA` port for that clock cycle. The new reload procedure can proceed to completion as indicated previously.

Coefficient Reload Order

To minimize the resources required to implement the coefficient reload feature, it is necessary for users to re-order the coefficients that are to be reloaded to correctly pass each coefficient to its correct storage location in the filter structure. When "Reloadable Coefficients" has been selected, CORE Generator delivers an informational text file to the project area named <component_name>_reload_order.txt, which lists the indices of the coefficients, "Index *x*," in the order they should be reloaded into the filter via the reload port "Reload index *x*."

Care must be take to correctly interpret the reload order, as it is based on the actual number of coefficients calculated by the filter. The Coefficient Padding section of Filter Symmetry discusses how the FIR Compiler sometimes implements a filter with more coefficients than specified. The actual coefficients calculated is displayed on the Implementation Details tab. When the filter is configured to utilize coefficient symmetry, the user must pad the filter response at the beginning and the end with (actual - specified)/2 zeros before applying the reload order. Figure 17 demonstrate a padded filter response. When the filter is non-symmetric, the coefficient set must be padded with (actual - specified) zeros at the end of the filter response before applying the reload order.

In the case of a polyphase interpolating filter utilizing coefficient symmetry, where the Symmetric Pairs technique has been used, the coefficients must be preprocessed before loaded into the filter. The combination of the non-symmetric sub-filters are defined as the sum or difference of two coefficient indices. When the filter configuration requires multiple DSP slices to implement a single Multiply-Accumulate unit, the definition is extended to include bit ranges of the source coefficients.

Figure 58 contains two examples of the _reload_order.txt file, both for a non-symmetric 16-tap single rate filter where the clock rate is four times the input sample rate. Systolic Multiply-Accumulate architecture has been selected for the left-hand example and Transpose Multiply-Accumulate for the right-hand example.

```
Reload index 0 = Index 12
Reload index 1 = Index 13
Reload index 2 = Index 14
Reload index 3 = Index 15
Reload index 4 = Index 8
Reload index 5 = Index 9
Reload index 6 = Index 10
Reload index 7 = Index 11
Reload index 8 = Index 4
Reload index 9 = Index 5
Reload index 10 = Index 6
Reload index 11 = Index 7
Reload index 12 = Index 0
Reload index 13 = Index 1
Reload index 14 = Index 2
Reload index 15 = Index 3
```

```
Reload index 0 = Index 0
Reload index 1 = Index 4
Reload index 2 = Index 8
Reload index 3 = Index 12
Reload index 4 = Index 1
Reload index 5 = Index 5
Reload index 6 = Index 9
Reload index 7 = Index 13
Reload index 8 = Index 2
Reload index 9 = Index 6
Reload index 10 = Index 10
Reload index 11 = Index 14
Reload index 12 = Index 3
Reload index 13 = Index 7
Reload index 14 = Index 11
Reload index 15 = Index 15
```

Figure 58: **Reload Order Text File Format Examples 1 and 2**

Figure 59 contains an example for a symmetric 15-tap interpolate by 3 filter where the clock rate is six times the input sample rate and a coefficient width of 16 bits.

```
Reload index 0 = Index 7
Reload index 1 = Index 10
Reload index 2 = Index 6 - Index 8
Reload index 3 = Index 9- Index 11
Reload index 4 = Index 6 + Index 8
Reload index 5 = Index 9 + Index 11
Reload index 6 = Index 1
Reload index 7 = Index 4
Reload index 8 = Index 0 - Index 2
Reload index 9 = Index 3 - Index 5
Reload index 10 = Index 0 + Index 2
Reload index 11 = Index 3 + Index 5
```

Figure 59: **Reload Order Text File Format Example 3**

Figure 60 contains an example with the same filter configuration as in Figure 59, but with a coefficient width of 30 bits (the width of the reload port is extended when the Symmetric Pairs technique is used, so in this example, the reload port is 33 bits wide).

Contact Xilinx Technical Support if you need any assistance or guidance in implementing the reload coefficient ordering for your specific filter implementation.

```
Reload index 0 (17 downto 0) = "00" & Index 7 (15 downto 0)
Reload index 0 (32 downto 18) = Index 7 (29) & Index 7 (29 downto 16)
Reload index 1 (17 downto 0) = "00" & Index 10 (15 downto 0)
Reload index 1 (32 downto 18) = Index 10 (29) & Index 10 (29 downto 16)
Reload index 2 (17 downto 0) = "00" & Index 6 (15 downto 0) -
                "00" & Index 8 (15 downto 0)
Reload index 2 (32 downto 18) = Index 6 (29) & Index 6 (29 downto 16) -
                 Index 8 (29) & Index 8 (29 downto 16)
Reload index 3 (17 downto 0) = "00" & Index 9 (15 downto 0) -
                "00" & Index 11 (15 downto 0)
Reload index 3 (32 downto 18) = Index 9 (29) & Index 9 (29 downto 16) -
                 Index 11 (29) & Index 11 (29 downto 16)
Reload index 4 (17 downto 0) = "00" & Index 6 (15 downto 0) +
                "00" & Index 8 (15 downto 0)
Reload index 4 (32 downto 18) = Index 6 (29) & Index 6 (29 downto 16) +
                 Index 8 (29) & Index 8 (29 downto 16)
Reload index 5 (17 downto 0) = "00" & Index 9 (15 downto 0) +
                "00" & Index 11 (15 downto 0)
Reload index 5 (32 downto 18) = Index 9 (29) & Index 9 (29 downto 16) +
                 Index 11 (29) & Index 11 (29 downto 16)
Reload index 6 (17 downto 0) = "00" & Index 1 (15 downto 0)
Reload index 6 (32 downto 18) = Index 1 (29) & Index 1 (29 downto 16)
Reload index 7 (17 downto 0) = "00" & Index 4 (15 downto 0)
Reload index 7 (32 downto 18) = Index 4 (29) & Index 4 (29 downto 16)
Reload index 8 (17 downto 0) = "00" & Index 0 (15 downto 0) -
                "00" & Index 2 (15 downto 0)
Reload index 8 (32 downto 18) = Index 0 (29) & Index 0 (29 downto 16) -
                 Index 2 (29) & Index 2 (29 downto 16)
Reload index 9 (17 downto 0) = "00" & Index 3 (15 downto 0) -
                "00" & Index 5 (15 downto 0)
Reload index 9 (32 downto 18) = Index 3 (29) & Index 3 (29 downto 16) -
                 Index 5 (29) & Index 5 (29 downto 16)
Reload index 10 (17 downto 0) = "00" & Index 0 (15 downto 0) +
                "00" & Index 2 (15 downto 0)
Reload index 10 (32 downto 18) = Index 0 (29) & Index 0 (29 downto 16) +
                  Index 2 (29) & Index 2 (29 downto 16)
Reload index 11 (17 downto 0) = "00" & Index 3 (15 downto 0) +
                "00" & Index 5 (15 downto 0)
Reload index 11 (32 downto 18) = Index 3 (29) & Index 3 (29 downto 16) +
                  Index 5 (29) & Index 5 (29 downto 16)
```

Figure 60: **Reload Order Text File Format Example 4**

Coefficient Quantization

The FIR Compiler offers three coefficient quantization options: Integer Coefficient, Quantize Only, and Maximize Dynamic Range. When the coefficients are specified using Radix 2 (binary) and 16 (hexadecimal), only the "Integer Coefficients" option is available, as the coefficients are considered to have already been quantized. When the coefficients are specified using integer numbers, all of the quantization options are available. When the coefficients are specified using non-integer decimal numbers (containing fractional information), only the "Quantize Only" and "Maximize Dynamic Range" options are available.

Integer Coefficients

The "Integer Coefficients" quantization option analyzes the coefficients and determines the minimum number of bits required to represent the coefficients. The coefficient width must be equal to or greater than this value. When more bits are specified than required, the coefficients are sign extended. If the user wishes to truncate the coefficients, the "Quantize Only" option must be used.

Quantize Only

Primarily for use when the filter coefficients have been specified using non-integer real numbers, this option quantizes the coefficients to the specified coefficient bit width. The coefficient values are rounded to the nearest quantum using a simple round towards zero algorithm. The coefficient word is split into integer and fractional bits. The integer width is determined by analyzing the filter coefficients to find the maximum integer value. The remaining bits are allocated to represent the fractional portion of the coefficient values. When the specified coefficient bit width is less than the required integer bit width, coefficients are appropriately rounded. The default value for the Coefficient Fractional Bits parameter is set to maximize the precision of the coefficients, but it can be reduced by the user. In this circumstance, more bits are allocated to the integer portion of the word, and the coefficient values are sign extended appropriately. When all the specified coefficients are between 1 and -1, only a single integer bit is required (to convey sign information), with the remainder of the coefficient word being used for fractional bits. When the coefficient range reduces further, the fractional bit width can be specified to a value greater than or equal to the coefficient width. See the Best Precision Fractional Length section for further explanation.

The frequency response of the quantized filter coefficients are compared to the ideal response on the Frequency Response Tab. This enables the user to explore the trade-off between filter performance and resources by varying the coefficient width parameter.

Maximize Dynamic Range

The user can also choose to scale the coefficients to utilize the full dynamic range provided by the coefficient bit width by selecting the Maximize Dynamic Range option. If selected, this results in the filter coefficients being scaled up by a common factor such that the largest coefficient (usually the center tap) is equal to the maximum representable value using the chosen bit width, then quantized. The overall scale factor is calculated as the ratio of the sum of the scaled and quantized coefficients to the sum of the original (ideal) coefficients. This value is calculated by the FIR Compiler and is presented (in dB) as part of the legend text on the filter response graph, or on the Summary page in the CORE Generator GUI.

The filter response plot for the quantized coefficients is scaled down by the scale factor for easy comparison against the ideal coefficients.

Scaling the coefficients introduces a gain which should be taken into account in the user's design.

Example 1

For this example the coefficients are signed with a coefficient width of 10 bits and a coefficient fractional width of 5 bits (using the System Generator Fix format notation Fix10_5). The specified coefficients range between -12.34 and +13.88.

Considering the coefficient bit width as integer only 10 bits give a maximum positive value of 511 and a maximum negative value of -512. The fractional bit width is 5 bits; this gives a maximum representable positive number of 511/(2^5)=15.96875 and a maximum representation negative number of -512/(2^5)=-16. All coefficients are scaled by the factor 15.96875/13.88=1.1504863 (=+1.2176dB) prior to quantization. The overall scaling factor is calculated as defined previously and displayed in the core GUI.

Example 2

For this example the coefficients are signed with a coefficient width of 18 bits and a coefficient fractional width of 19 bits, or Fix18_19. The specified coefficients range between -0.000256022 and +0.182865845.

An integer coefficient width of 18 bits gives a maximum positive value of 131071 and a maximum negative number of -131072. Considering the fractional bits, this gives a maximum representable positive number of 131071/(2^19)=0.249998092 and a maximum representable negative number of 131072/(2^19)=0.25. The scaling factor is determined by dividing the maximum value that can be represented (for the specified number of coefficient bits) by the maximum coefficient value. In this case 0.249998092/0.182865845=1.367112009 (=+2.716081962dB).

Note: While an appreciable improvement in performance can be achieved by making use of the full dynamic range of the coefficient bit width, this is not always the case, and the user must be satisfied that any changes are acceptable via the frequency response plot. The user must also account appropriately for any additional gain introduced by coefficient scaling elsewhere in the application system. In many systems, signal scaling may be arbitrary and no gain compensation is required; where scaling is necessary, it is often desirable to amalgamate gains inherent in a signal processing chain and compensate or adjust for these gains either at the front end (for example, in an Automatic Gain Control circuit) or the back end (for example, in a Constellation Decoder unit) of the chain. If the user wishes not to introduce any additional scaling into the design, “Quantize Only” should be chosen.

Best Precision Fractional Length

When the “Best Precision Fractional Length” option is selected, the coefficient fractional width is set to maximize the precision of the specified filter coefficients. As discussed in the Quantize Only section, the FIR Compiler analyzes the filter coefficients to determine how many bits are required to represent the integer portion of the coefficient values. All the remaining coefficient bits are then allocated to represent the fractional portion of the coefficients. When all the specified coefficients are between 1 and -1, only a single integer bit is required. The reminder of the coefficient word is then used for fractional bits. When the coefficient range reduces further, the fractional bit width is specified to a value greater than or equal to the coefficient width; otherwise the coefficient values contain redundant information that does not need to be explicitly stored. The available coefficient bits can then be better used to increase the precision of the coefficient values. This section goes on to illustrate this concept further. The System Generator Fix Format notation is used, Fix**word length_fractional length**. The word length is specified by the Coefficient Width parameter, and the fractional length is specified by the Coefficient Fractional Bits parameter.

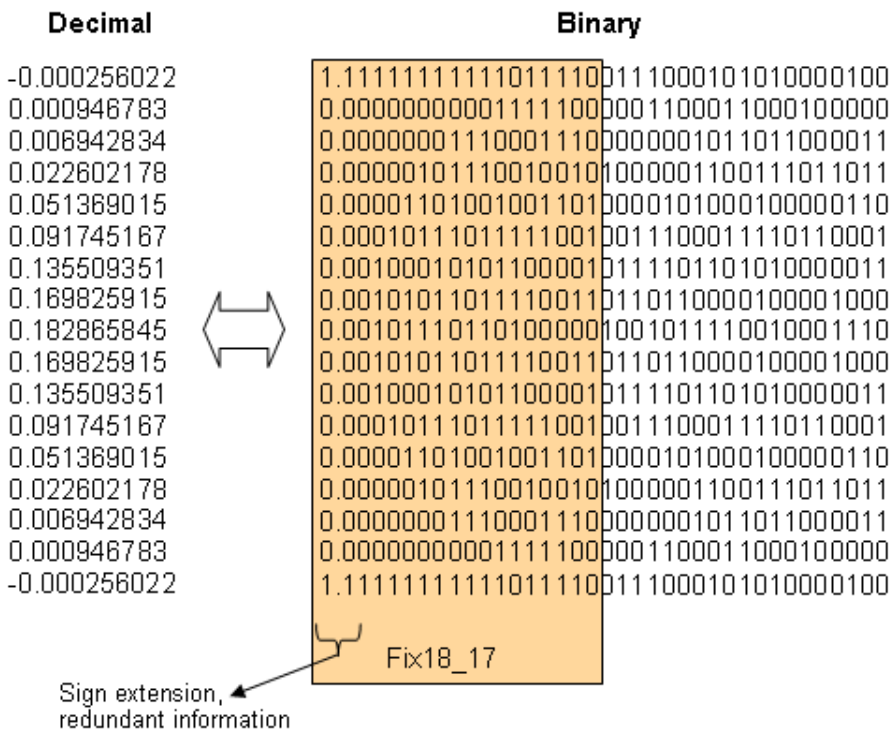


Figure 61: **Coefficient Quantization Fix18_17**

In Figure 61 the coefficient values are represented using 18 bits. The binary point is positioned such that 17 bits are used to represent the fractional portion of the number. An analysis of the coefficients reveals that no value has a magnitude greater than 0.25; therefore, the upper two MSBs are a sign extension and contain redundant information.

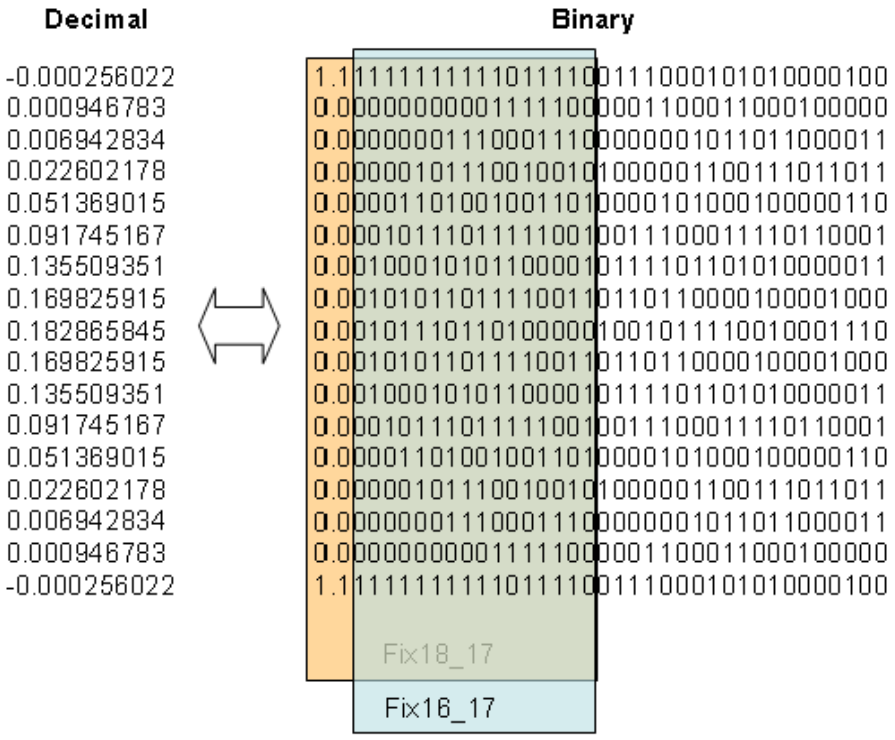


Figure 62: **Coefficient Quantization Fix16_17**

In Figure 62, 16 bits are used to represent the same coefficient values to the same precision. The redundant information has been removed, reducing the resources required to store the filter coefficients. The binary point position has not moved. 17 bits are still effectively used to represent the fractional portion of the number, but one of them does not need to be explicitly stored, as it is a sign extension of the stored MSB.

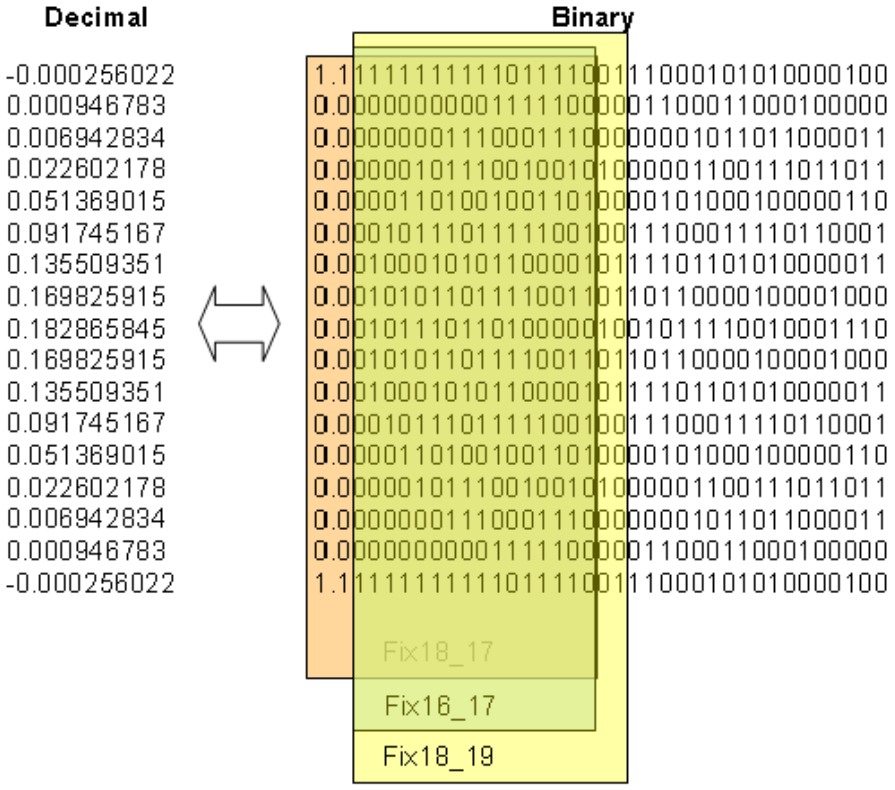


Figure 63: **Coefficient Quantization Fix18_19**

In Figure 63 18 bits are specified for the coefficient width. The two additional bits can be used to increase the precision. The binary point position has still not moved, but now, 19 bits are effectively used to represent the fractional portion of the number, which results in an increase of the filter precision.

Parallel Data Paths

The FIR Compiler provides support for processing multiple parallel data paths with the same filter coefficients. This feature differs from a multiple-channel implementation when it is necessary to time division multiplex (TDM) the individual channels onto a single data stream. When processing parallel data paths, the FIR Compiler allocates an input and output port (`DIN_1` to `DIN_16` and `DOUT_1` to `DOUT_16`) to each individual data path. In this configuration, the FIR Compiler can share control logic and coefficient memory resources between the parallel data paths. This offers significant resource savings over using one FIR Compiler instance per parallel data path.

Output Width and Bit Growth

The full precision output width can be defined as the input data width plus the bit growth due the application of the filter coefficients. Bit growth from the original sample width occurs as a result of the many multiplications and additions that form the basic function of the filter. Therefore, the accumulator result width is significantly larger than the original input sample width. Limiting the accumulator width is desirable to save resources, both in the filter output path (such as output buffer memory, if present) and in any subsequent blocks in the signal processing chain. The worst case bit growth can be obtained by adding the coefficient width to the base 2 logarithm of the number of non-zero multiplications required (rounded up); however, this does not take into account the actual coefficient values. Taking the base 2 logarithm of the sum of the absolute value of all filter coefficients reveals the true maximum bit growth for a fixed coefficient filter, and this can be used to limit the required accumulator width. The

following equation demonstrates this calculation, where B is the calculated bit growth, N is the number for filter coefficients, and a_n is n^{th} filter coefficient.

$$B = ceil\left[\log_2\left(\sum_{n=0}^{(N-1)} |a_n|\right)\right]$$

For MAC implementations the FIR Compiler automatically calculates the bit growth based on the actual coefficient values. For reloadable filters, or any DA-based filter, the worst case bit growth is used.

The Coefficient (and Data) fractional width does not affect the output width calculation. The core determines the output width without considering fractional bits. The core determines the full precision output as previously described and then determines the output fractional width by summing the data and coefficient fractional bit width

Output Rounding

As mentioned in "Output Width and Bit Growth," it is desirable to limit the output sample width of the filter to minimize resource utilization in downstream blocks in a signal processing chain. For MAC implementations the FIR Compiler includes features to limit the output sample width and round the result to the nearest representable number within that bit width. Several rounding modes are provided to allow the user to select the preferred trade-off between resource utilization, rounding precision, and rounding bias:

- Full Precision
- Truncation (removal of LSBs)
- Non-symmetric rounding (towards positive or negative)
- Symmetric rounding (towards zero or infinity)
- Convergent rounding (towards odd or even)

In the following descriptions, the variable *x* is the fractional number to be rounded, with *n* representing the output width (that is, the integer bits of the accumulator result) and *m* representing the truncated LSBs (that is, the difference between the accumulator width and the output width). In Figure 64 through Figure 66, the direction of inflexion on the red midpoint markers indicates the direction of rounding.

Full Precision

In Full Precision mode, no output sample bit width reduction is performed (n=accumulator width, m=0). This is the default option and is also the only option for DA-based filters.

Truncation

In Truncation mode, the m LSBs are removed from the accumulator result to reduce it to the specified output width; the effect is the same as the MATLAB function *floor(x)*. This has the advantage that it can be implemented simply with zero resource cost, but has the disadvantage of being biased towards the negative by 0.5.

Non-symmetric Rounding to Positive

In this rounding mode, a binary value corresponding to 0.5 is added to the accumulator result and the m LSBs are removed; this is equivalent to the MATLAB function *floor(x+0.5)*. The addition can usually be done in most filter configurations with little or no resource cost in hardware using the DSP slice features. It has the disadvantage of being biased towards the positive by $2^{-(m+1)}$.

Non-symmetric Rounding to Negative

In a modification of the preceding technique, a binary value corresponding to 0.499... is added to the accumulator result and the m LSBs are removed; this is equivalent to the MATLAB function *ceil(x-0.5)*. The resource usage advantage is the same, but the bias in this case is towards the negative by $2^{-(m+1)}$.

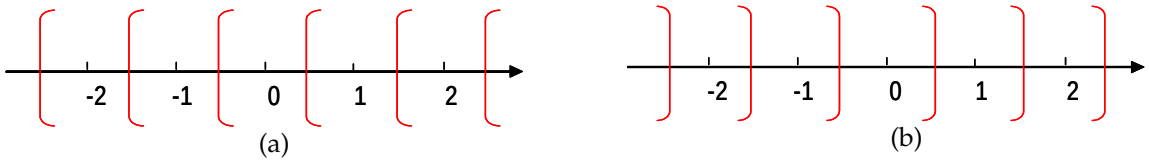


Figure 64: **Non-symmetric Rounding (a) to Positive (b) to Negative**

Symmetric Rounding to Highest Magnitude

The bias incurred during non-symmetric rounding occurs because rounding decisions at the midpoints always go in the same direction. In symmetric rounding, the decision on which direction to round is based on the sign of the number. For rounding towards highest magnitude, a binary value corresponding to 0.499 is added to the accumulator result, and the inverse of the accumulator sign bit is added as a carry-in before removal of the m LSBs. As is generally the case, there are as many positive as negative numbers; the result should not be biased in either direction. This rounding mode is commonly used in general applications, mainly due to the fact that it is equivalent to the MATLAB function *round(x)*.

Symmetric Rounding to Zero

The implementation difference for this mode from round to highest magnitude is that the sign bit is used directly as the carry-in (see Figure 65). There is no direct MATLAB software equivalent of this operation. One minor advantage of rounding towards zero is that it does not cause overflow situations.

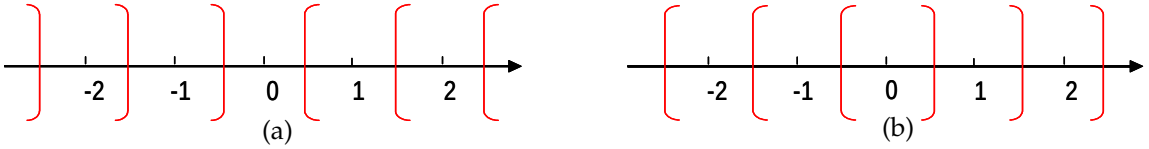


Figure 65: **Symmetric Rounding (a) to Highest Magnitude (b) to Zero**

Approximation of Symmetric Rounding

One important point to note about symmetric rounding mode is that to achieve the correct result, the sign of the accumulator must be known before the addition of the rounding constant to generate the correct carry-in. This requires an additional processing cycle to be available. When the additional cycle is not available and the user wishes to maintain full accuracy, a separate rounding unit must be used (FIR Compiler calculates whether or not this is required automatically).

An alternative technique is available to users who wish to employ symmetric rounding but do not have a spare cycle available, if some inaccuracies are acceptable. The rounding constant can be added on the initial loading of the accumulator, and the sign bit can be checked on the penultimate accumulation cycle and added on the final accumulation. This normally achieves the same result, but there is a small risk that the accumulated result changes sign between the penultimate and final accumulation cycles, which causes the midpoint decision to go in the wrong direction occasionally.

It is important to note that while some implementations of this approximation technique rearrange the calculation order of coefficients and data such that the smallest coefficient is used last, the FIR Compiler does not perform any rearrangement of coefficients and data. This is significant for symmetric filters, as the centre coefficient is the final

coefficient calculated. For non-symmetric filters, the final coefficient is often very small and would be unlikely to affect the sign of the final result. It is also important to note that the risk of the sign changing between the penultimate and final accumulation cycles increases as the level of parallelism employed in the core increases. This is due to the contribution added to the accumulation on each cycle increasing as the number of cycles per output decreases. Therefore, it is important that users consider carefully the coefficient structure and level of parallelism they intend to use before deciding on whether to employ approximation of symmetric rounding.

Convergent Rounding

Convergent rounding chooses the rounding direction for midpoints as either toward odd or even numbers, rather than toward positive or negative (Figure 66). This can be advantageous, as the balance of rounding direction decisions for midpoints is based on the probability of occurrence of odd or even numbers, which are generally equal in most scenarios, even when the mean of the input signal moves away from zero. The function is achieved by adding a rounding constant, as in other modes, but then checking for a particular pattern on the LSBs to detect a midpoint and forcing the LSB to be either zero (for round to even) or one (for round to odd) when a midpoint occurs.

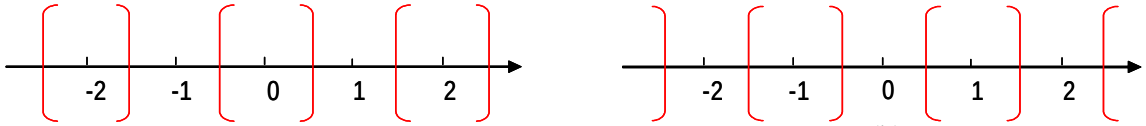


Figure 66: **Convergent Rounding (a) to Even (b) to Odd**

Resource Implications of Rounding

The implications with regard to resource utilization of selecting a particular rounding mode should be considered by users. Generally, the FIR Compiler IP core attempts to integrate rounding functions with existing functions, which usually means the accumulator portion of the circuit. However, this is not always possible. In certain combinations of rounding mode, filter type and device family, an additional DSP slice must be used to implement the rounding function. The most important factor to consider is the inherent hardware support for each mode in each of the device families, but filter type and configuration also play a role. Convergent rounding requires pattern detection support, and, therefore, this mode is only available in Virtex-5 and Virtex-6 devices.

Table 7 indicates the combinations of filter type and rounding type for which no extra DSP slice is likely to be required. Where all three DSP slice enabled device families are likely to support that combination of rounding mode and filter type without an additional DSP slice, a tick mark is entered; where none of the three is likely to support the combination without the additional DSP slice, a check mark is entered; where there is a list of families provided, the list refers to those families that support the combination without an extra DSP slice. The device families are abbreviated to: V4 for Virtex-4; V5 for Virtex-5; and S3D for Spartan-3A DSP. Support for symmetric rounding assumes that either there is a spare cycle available, or approximation is allowed. If this is not the case, an additional DSP slice is always required for symmetric rounding modes, regardless of filter type or family.

It is important to note that the table is indicative only, and certain combinations for which hardware support is indicated will actually require the extra DSP, and vice versa. Notable exceptions to the table include parallel multi-channel decimation with symmetric rounding (approximated), which requires an additional DSP slice.

Table 7: **Indicative Table of Hardware Support for Rounding Modes for Particular Filter Types**

Filter Type	Non-symmetric	Symmetric (Infinity)	Symmetric (Zero)	Convergent
Single Rate, Interpolated, Hilbert	✔	V4,V5,V6	V5,V6	V5,V6
Half-band	✔	V4,V5,V6	V5,V6	V5,V6

Table 7: **Indicative Table of Hardware Support for Rounding Modes for Particular Filter Types *(Cont'd)***

Filter Type	Non-symmetric	Symmetric (Infinity)	Symmetric (Zero)	Convergent
Interpolating without Symmetry	✔	V4,V5,V6	V5,V6	V5,V6
Interpolate by 2, Odd Symmetry	✔	V4,V5,V6	V5,V6	V5,V6
Interpolating with Symmetry (others)	✖	✖	✖	✖
Interpolating Half-band	✔	V4,V5,V6	✖	V5,V6
Decimating, Single-channel	✔	V4,V5,V6	V5,V6	V5,V6
Decimating, Multi-channel	✔	V4,V5,V6	V5,V6	V5,V6
Decimating Half-band	✔	V4,V5,V6	V5,V6	V5,V6
Fractional Interpolation	✔	V4,V5,V6	V5,V6	V5,V6
Fractional Decimation, Single-channel	✔	V4,V5,V6	V5,V6	V5,V6
Fractional Decimation, Multi-channel	✔	V4,V5,V6	V5,V6	V5,V6

Multiple Column Filter implementation

The FIR Compiler can build filter implementations that span multiple DSP slice columns. The multi-column implementation is only required when the filter parameters, specifically the number of filter coefficients and the hardware oversampling rate (Sample Frequency to Clock Frequency ratio), result in an implementation that requires more DSP slices than are available in a single column of the select device. Figure 67 illustrates the structures implemented.

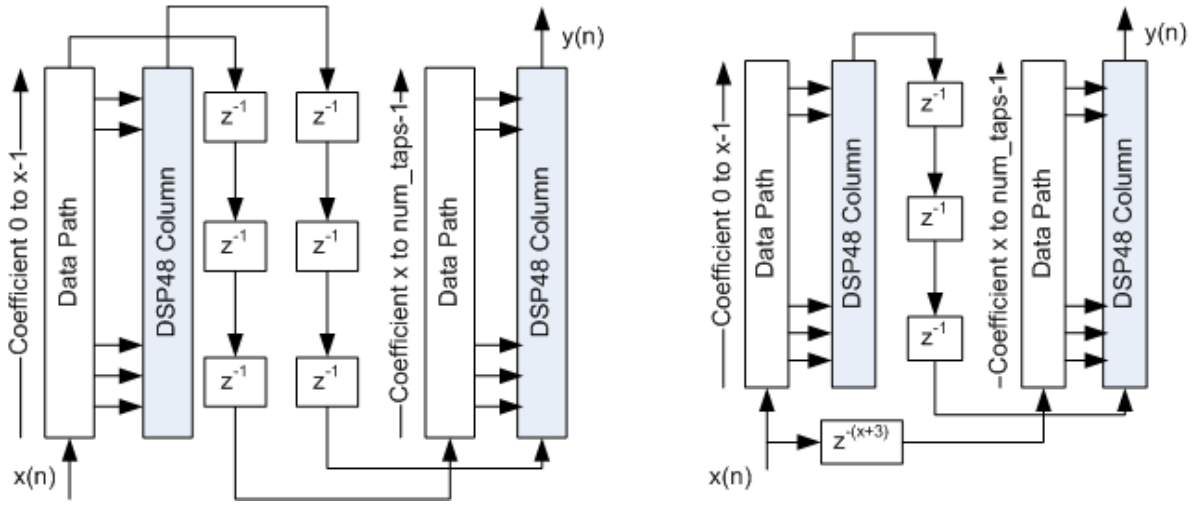


Figure 67: **Multi-column Implementations: Standard Implementation, Left; DSP Slice Data Cascade Port Implementation, Right**

This feature is only available when the Multiply-Accumulate filter architectures are selected on device families with more than one DSP slice column. Currently this feature is not supported for symmetric coefficient structures. To ensure this feature is available, set the Coefficient Structure parameter to "Non Symmetric." See the Multiple Chan-

nels and Symmetric Filters section of Resource Considerations for further details on how to implement large symmetric filters.

The DSP column lengths are displayed on the Details Implementation Options page of the CORE Generator GUI. The implemented column lengths can be determined automatically, Multi-column Support: Automatic, or specified by the user, Multi-column Support Automatic. The length of each implemented DSP column can be specified using the Column Configuration parameter. See the Detailed Implementation Options Screen section for more details.

Resource Considerations

The number of DSP slices utilized by the FIR Compiler is primarily determined by the number of coefficients, modified by any rate change, and the hardware oversampling rate per channel (defined by the Sample Period or the Sample frequency to Clock frequency ratio divided by the number of channels). Users should also be aware that Data and Coefficient Bit Width and Output Rounding Selection can also affect the DSP slice usage and are discussed in the following sections.

Tab 3: Implementation Details of the CORE Generator GUI displays the core DSP slice usage given all the core parameters.

Data and Coefficient Bit Width

When the FIR Compiler is configured to implement the Multiply-Accumulate filter architectures, the DSP slice resource usage is influenced by the data and coefficient width specified. When the data and coefficient widths are specified to be greater than the input width of the DSP slice for the given device family, the core uses multiple DSP slice columns to implement the filter. Table 8 provides a guide to the number of DSP columns that are required for various combinations of data and coefficient widths.

Table 8: **DSP Slice Column Usage for Given Data and Coefficient Widths**

Family	Data Width		Coefficient Width		Number of DSP Slice Columns
	Unsigned	Signed	Unsigned	Signed	
Spartan 3ADSP, Spartan-6, and Virtex-4	<=17	<=18	<=17	<=18	1
	>17	>18	<=17	<=18	2
	<=17	<=18	>17	>18	2
	>17	>18	>17	>18	4
Virtex-5 and Virtex-6(1)	<=24	<=25	<=17	<=18	1
	<=17	<=18	<=24	<=25	1
	>24	>25	<=17	<=18	2
	<=17	<=18	>24	>25	2
	>17	>18	<=24	<=25	2
	<=24	<=25	>17	>18	2
	>24	>25	>17	>18	4
	>17	>18	>24	>25	4

1. **Note:** The data/coefficient widths at which Virtex-5/6 implementation transition to multi-column implementations may lower given the number of filter coefficients to ensure the accumulator width does not exceed 48 bits, thereby avoiding overflow.

Output Rounding Selection

The selected output rounding mode may cause additional DSP slice resources to be used. See the Output Rounding section for more details.

Multiple Channels and Symmetric Filters

When a filter is configured to use multiple channels, the number of clock cycles available to process the filter is reduced compared to implementing the same filter for a single channel. For example, a single-channel filter with a sample frequency of 1 MHz and a clock frequency of 4 MHz gives four clock cycles per input sample to generate an output. If the same sample and clock frequency is retained but two channels are processed, the core input is time-division multiplexed between the two channels, reducing the clock cycles per input sample to two. This results in a proportional increase in the number of DSP slices utilized.

A problem can arise when the FIR Compiler has detected that the specified filter coefficients have a symmetric coefficient structure (see the Filter Symmetry section for more details on utilizing coefficient symmetry), but the implementation still requires more DSP slices than are available in a single DSP slice column of the selected device. As symmetry is not supported by the multi-column implementation, it is not possible to generate this filter configuration. To enable this feature, the coefficient structure would have to be set to "Non Symmetric," but the DSP slice resources required will be doubled. When the filter has been configured to support multiple channels, an alternative implementation is possible. Splitting the channels to be implemented across multiple parallel data paths results in each data path having more cycles available to process the filter coefficients, reducing the number of DSP slices required in a single column. It may then be possible to implement the filter configuration.

For example, a filter with 96 symmetric coefficients implementing four channels with a sample frequency of 1 MHz and a clock frequency of 4 MHz requires 48 DSP slices. If the selected device only has 32 DSP slices per column, this filter configuration cannot be generated. If the coefficient structure is set to "Non Symmetric," the implementation requires 96 DSP slices, but they can be split over three DSP slice columns. If the configuration is changed to implement two parallel data paths with two time-division multiplexed channels per path, the core uses 25 DSP slices per parallel data path (24 multiply-adds plus an accumulator), giving a total of 50 DSP slices. As the DSP slice column requirement is reduced from 48 to 25, the filter configuration can be generated.

Multiple Channel vs. Parallel Data Paths

The Multiple Channel Filters and Parallel Data Paths features both offer the facility to process multiple input sample streams but using different interfaces. A multi-channel interface requires the multiple input streams to be time division multiplexed (TDM) into a single core input, whereas the Parallel Data Paths interface provides an individual core input for each input stream. The choice of interface can influence the resources used by the core. In general, the multi-channel implementation uses less DSP slice resources, but under some circumstances this is not the case. The following example demonstrates such a situation. It may also be desirable to consider the Parallel Data Paths implementation when implementing filter where a large number of DSP slices is required. This is discussed in the Multiple Channels and Symmetric Filters section.

Example 1

Consider an 8-tap single rate filter that is to process four 12.5 MHz input streams with a clock frequency of 100 MHz.

Multi-channel implementation:

100 MHz/12.5 MHz=8 clock cycles per input sample. Shared between the four input streams, 8/4=2, gives a hardware oversampling rate of 2. The 8 filter coefficients must be processed in 2 clock cycles. This gives 8/2=4 DSP slices, where the filter processes the first 4 coefficients on the first clock cycle and the remaining 4 coefficients on the

second clock cycle. The two partial products must be summed together, so an additional accumulator DSP slice is required. This gives a total of **5** DSP slices.

Parallel Data Paths:

100 MHz/12.5 MHz=8 clock cycles per input sample. Each input stream can use the full 8 clock cycles to process the 8 filter coefficients. This gives 8/8=1 multiply-accumulate DSP slice. The core provides four input streams, each using 1 DSP slice. This gives a total of **4** DSP slices.

This demonstrates that the Parallel Data Path implementation offers a more efficient implementation.

If the input sample frequency was increased to 25 MHz per channel, this would not be the case, illustrated as follows.

Multi-channel implementation:

8 taps/(100 MHz/25 MHz/4)=**8** DSP slices, no accumulator required.

Parallel Data Paths:

8 taps/(100 MHz/25 MHz)=2 DSP slices, plus 1 accumulator DSP slice gives 3 DSP slices per path. A total of **12** DSP slices are required.

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 (Ready For Data), and RDY (Ready) are used to coordinate I/O operations. In addition, for multi-channel filters, CHAN_IN and CHAN_OUT indicate the active input and output stream respectively; and for multiple coefficient sets, the current set to be used is specified using FILT_SEL. Generally, these flow control signals are compulsory; however, for MAC-based FIR filter implementations, ND is optional and a Clock Enable (CE) pin is provided to allow core processing operations to be halted.

Handshake Control Signals

ND is an active high input signal which, when asserted, indicates to the core that there is a valid input sample on the DIN port. ND is internally qualified with the active high output status signal RFD. When both RFD and ND are asserted, the DIN port is sampled on the rising clock edge. The active high output signal RDY indicates that a valid filter output is available on the DOUT port. For Multiply-Accumulate architectures, ND is optional, in which case the filter always takes data from the input port on the first cycle that RFD is asserted, or continuously for parallel filter structures. For parallel symmetric filters, use of CE without ND can lead to an appreciably more efficient implementation.

The handshake signals provide a simple and efficient interface to control the flow of sample data and results. Similar to a clock enable signal, the ND signal is used to enable the input of samples into the filter. The difference between ND and a clock enable is that the ND signal starts the processing operation that continues to completion. By not asserting the ND signal further, processing is halted, whereas a clock enable provides an immediate start and stop of the processing operation. A clock enable pin is provided for Multiply-Accumulate architectures, and its use is compatible with the ND function; it can be used with or without ND being present.

RFD provides a status signal for upstream data flow control, and when asserted indicates that the core can accept more input samples. The RDY signal is often used as a clock enable for the next stage of processing or as the ND signal when filters are cascaded.

The optional output signal DATA_VALID can be used in conjunction with or in place of RDY. As with RDY, it is active high and indicates a valid filter output is available on the DOUT port, but it is only asserted when the output sample on DOUT has been generated from a complete data vector. Following a reset, the core data memories are not cleared, which can result in a corrupt data vector. The data vector is not complete until the core has received the same number of input data samples as the filter has coefficients, or in the case of interpolating filters, the number coefficients per polyphase sub-filter. The use of DATA_VALID results in additional core resources – a small amount of control logic and a counter used to determine when the data vector is complete.

Resetting the Core

SCLR (Synchronous Clear) is an active high input port which, when asserted, forces the internal control logic to the initialized condition. No internal data is cleared from the filter memories during the reset process. Following a reset operation, the filter output remains dependent on the prior input samples until the filter data memory is completely flushed. When CE is selected, SCLR has priority.

The "Use Deterministic SCLR Behavior" option available for Multiply-Accumulate architectures (following a reset) forces the filter output to zero until the data memories are flushed. When the core is generated with a behavioral simulation file (rather than a structural simulation file), the reset behavior may not be identical to the generated netlist until the data vector is complete. The data vector is complete when the core has received the same number of input data samples as the filter has coefficients, or in the case of interpolating filters, the number of coefficients per polyphase sub-filter. The use of this feature results in additional core resources – a small amount of control logic (a counter used to determine when the data vector is complete), and the core output is always registered. The optional output signal DATA_VALID offers an alternative by adding additional qualification to the filter output indicating when the data vector is complete.

Input/Output Channel Decoding

When configured for a multiple-channel operation, two channel indicator status output ports are provided: CHAN_IN and CHAN_OUT. The CHAN_IN port identifies the input channel number; CHAN_OUT provides the mapping between the current sample on the filter output port DOUT and the sample stream number. These signals are often used as select controls for multiplexing input streams or de-multiplexing the time division multiplexed result bus. The CHAN_OUT signal is valid when RDY is asserted and changes after the falling edge of RDY.

The channel value presented on the CHAN_IN output port can be generated a number of input samples in advance when the "Generate CHAN_IN Value in Advance" option has been selected. The number of samples can then be specified in the GUI. This enables the CHAN_IN value to drive a registered process to select the input data sample for a given channel. For example, the CHAN_IN value could be used to drive the select input of a registered multiplexer or as the address for a memory block. This feature is only available for Multiply-Accumulate architectures

Coefficient Set Selection

When configured for multiple-coefficient operation, an additional input port FILTER_SEL is provided. The FILTER_SEL port identifies which set of coefficients are used to process the current set of data. This port is latched along with the input data DIN is sampled. The value on this port is used to address the portion of the coefficient buffer containing the desired coefficients.

Cycle Latency

The cycle latency of the filter is a function of the number of taps, filter type, number of channels, and coefficient symmetry. Cycle latency specifies the number of clock cycles from RFD being deasserted (indicating an input sam-

ple has been accepted from the DIN port) to the assertion of RDY, indicating a valid output has been generated by the filter.

When the input data rate of the filter is equal to the core clock rate, the RFD signal is not deasserted as the core accepts input data every clock cycle. Similarly for single rate filters, RDY remains asserted. In this circumstance, cycle latency specifies the number of clock cycles from a given input sample to the first output value it has contributed to.

For multiple channel decimation or interpolation configurations, the core contains some sample buffering. The buffer introduces a sample latency relating to the size of the buffer implemented. The cycle latency specifies the number of clock cycles from RFD being deasserted for the final input sample to the assertion of RDY.

The cycle latency is displayed on Tab 3: Implementation Details of the CORE Generator GUI for a given filter configuration.

The cycle latency is indicated on the timing diagrams contained in the MAC-based FIR Filter Timing section as a black arrow connecting corresponding RFD and RDY pulses.

Nomenclature

In the timing diagrams supplied in this section, the notations $x(n)$ 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 annotated only with the index value n.

MAC-based FIR Filter Timing

Single Rate, Multi-channel and Multiple Filter Sets

Figure 68 illustrates the timing for a single-rate, single-channel, N-tap MAC-based filter. ND is asserted while valid input is available on the DIN port. At the rising edge of the clock, the data is sampled and processing begins. RFD is deasserted to reflect that the MAC-based FIR core is processing the data and unable to accept further input samples for the period of the input data rate. After a number of clock cycles equal to the "Cycle Latency," RDY is asserted and the valid filter output is presented on the DOUT port. In this example, the DOUT value is held in the optional output register. In this configuration, core operation can be halted by holding ND low for the required idle note. However, the core continues to process any input data sampled so far and to produce outputs based on those input samples.

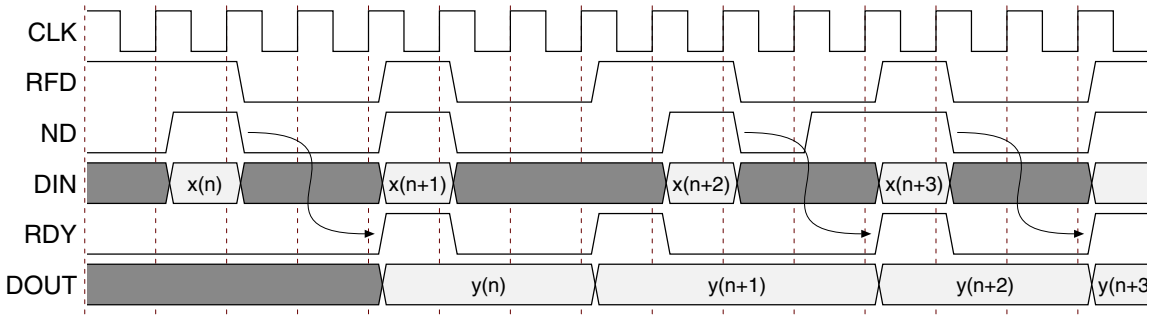


Figure 68: **Timing Diagram for a Single-channel Filter Using ND, with Registered Output**

Figure 69 illustrates the timing for the same filter without the ND port. When RFD is asserted, the input data is sampled at the rising edge of the clock and processing begins. RFD is deasserted as normal to indicate that the core cannot accept further input samples for period of the input data rate. When processing has completed, RFD is asserted once more for a single cycle and the next input data is processed. Note that in this configuration, it is required that the system or circuit that is driving the input data continues to feed data to the filter at the specified input rate; otherwise invalid data is sampled. Similarly, data samples should be held until the RFD signal is asserted; otherwise that sample is missed. After a number of clock cycles equal to the "Cycle Latency," RDY is asserted and the valid filter output is presented on the DOUT port. In this example, the DOUT value is held in the optional output register. If halting of core operation is required in this configuration, a clock enable pin is required on the core to halt all core operation. This is fundamentally different than halting the filter using ND – the clock enable halts all core operation and no outputs are produced during the period for which CE is deasserted. Core outputs continue only after CE is asserted once more.

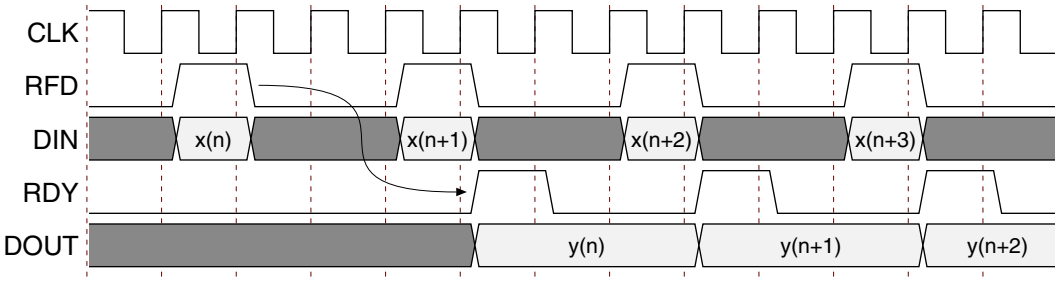


Figure 69: **Timing Diagram for a Single-channel Filter without ND Port, with Registered Output**

Figure 70 illustrates the timing for a multi-channel filter. The core accepts inputs for each channel sequentially (Time Domain Multiplexed or TDM format). Outputs are also presented as TDM format. A channel indicator is provided to track the currently active input and output channel.

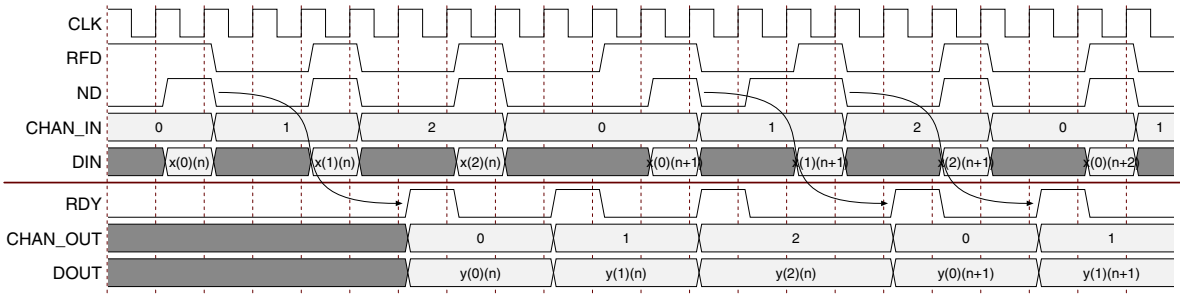


Figure 70: **Timing Diagram for a 3-channel Filter with ND Port and Registered Output**

Figure 71 illustrates the timing for a multi-channel filter which also has multiple filter sets. The filter interface operation is as described previously for multi-channel mode, but in this case there is a switch to an alternative filter set during the third data input cycle shown in the diagram. The filter set switch-over can occur on any data input cycle, and the filter immediately moves to that set of coefficients for processing that data sample (and all subsequent data samples while the filter select port value remains the same) through the filter.

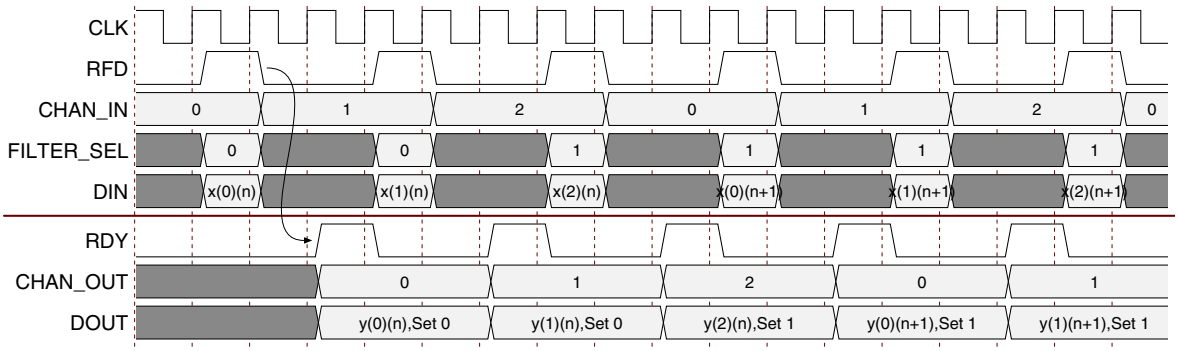


Figure 71: **Timing Diagram for a Multi-channel Filter with Multiple Filter Sets**

Multi-rate Filters

Multi-rate filters involve an increase or decrease in rate from input to output. Figure 72 shows a multi-channel (three channels) decimation filter with a rate decrease of two. Input data is taken in TDM format with two input samples for each channel being required before an output can be produced. Output data is also presented in TDM format at the lower rate.

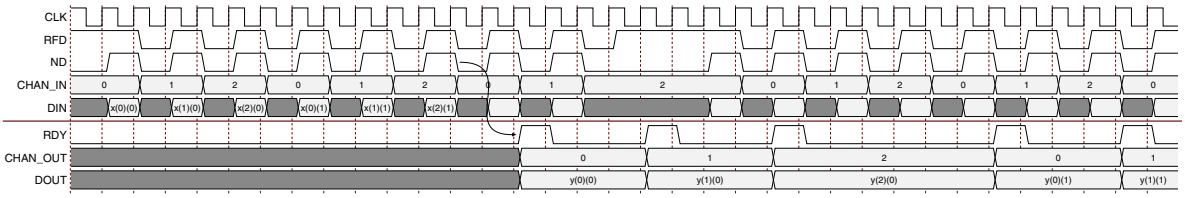


Figure 72: **Timing Diagram for a Multi-channel Decimation Filter**

Figure 73 shows a multi-channel (three channels) interpolation filter with a rate increase of two. Note that input data is taken in TDM format. Output data is then presented in TDM format at the higher rate.

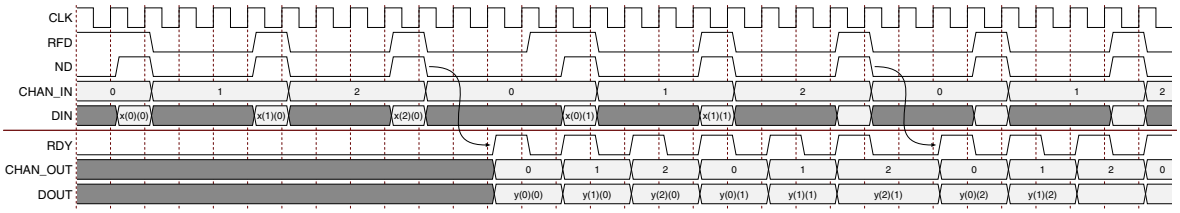


Figure 73: **Timing Diagram for a Multi-channel Interpolation Filter**

Figure 74 shows a multi-channel (two channels) fixed fractional interpolation filter with a rate increase of 5/3. The input sample period is four clock cycles per input, or eight clock cycles per channel.

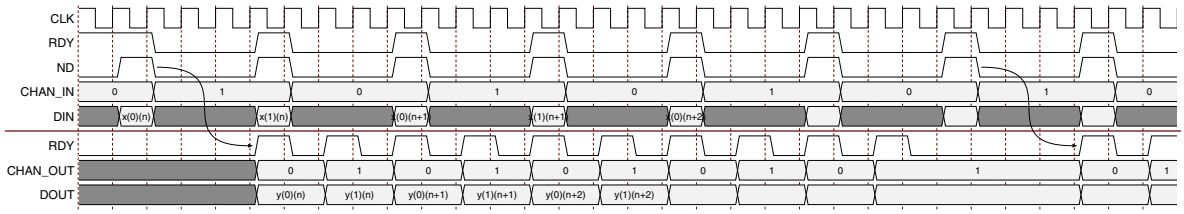


Figure 74: **Timing Diagram for a Multi-channel Fixed Fractional Interpolation Filter**

Figure 75 shows a multi-channel (two channels) fixed fractional decimation filter with a rate decrease of 3/5. The input sample period is three clock cycles per input, or six clock cycles per channel. This gives an output sample period of five clock cycles per output, or ten per channel output.

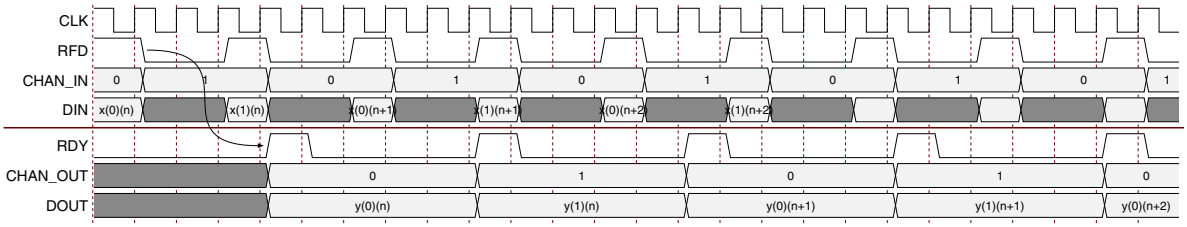


Figure 75: **Timing Diagram 1 for a Multi-channel Fixed Fraction Decimation Filter**

Figure 76 shows the timing diagram for the same filter configuration as the previous example, but the output sample period has been reduced to three clock cycles per output. This corresponds to an input sample period of 1.8 clock cycles. The input samples must therefore be provided in a non-periodic manner. The RFD output pin indicates then that the core is able to accept input data, which is at a rate to maintain the full output sample rate.

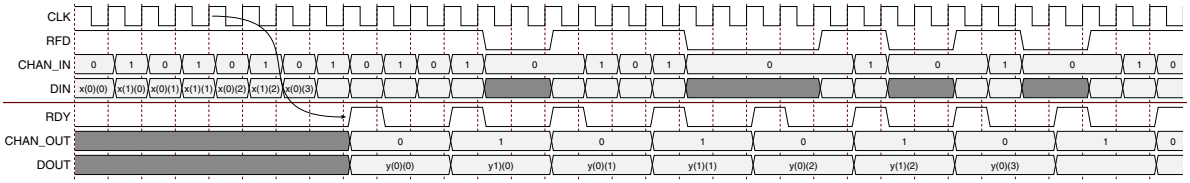


Figure 76: **Timing Diagram 2 for a Multi-channel Fixed Fraction Decimation Filter**

DATA_VALID and SCLR Deterministic Control Signals

Figure 77 shows a 5-tap multi-channel single-rate filter with the optional DATA_VALID port and "Use Deterministic SCLR Behavior" selected.

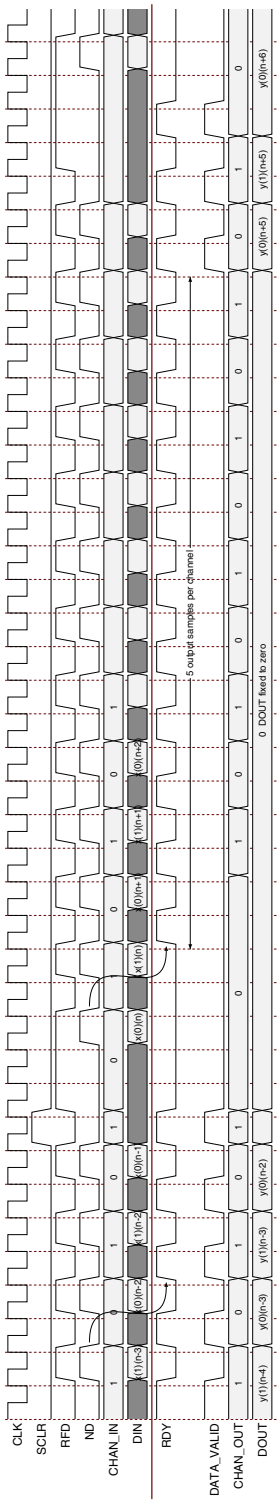


Figure 77: **DATA_VALID and Use Deterministic SCLR Behavior**

Polyphase Filter Bank

Figure 78 shows the timing diagrams for both an 8-channel transmitter (left) and receiver (right) Polyphase Filter Bank

The transmitter timing diagram illustrates a case where the individual channels have an input sample period of 16, and the time division multiplexed (TDM) input stream has a sample period of 2. For this configuration, the Input Sample Period parameter should be set to 2. Or, in terms of frequency, each channel Input Sampling Frequency is 1/16 of the clock rate. For example, if the Clock Frequency is 100 MHz, the Input Sampling Frequency would be 6.25 MHz.

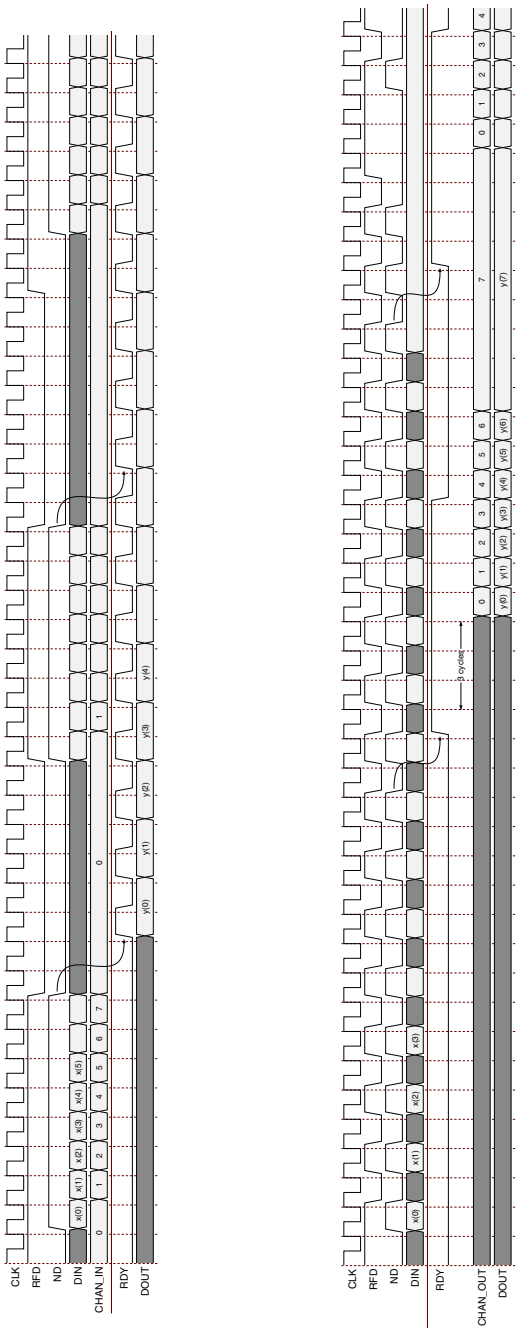
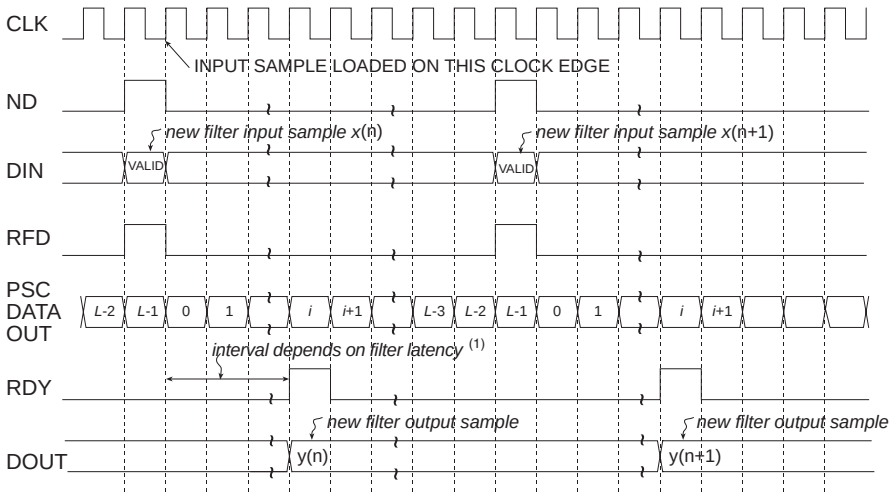


Figure 78: **Transmitter (Left) and Receiver (Right) Polyphase Filter Bank**

Distributed Arithmetic Filter Timing

Single-channel and Multi-channel DA FIR Filters

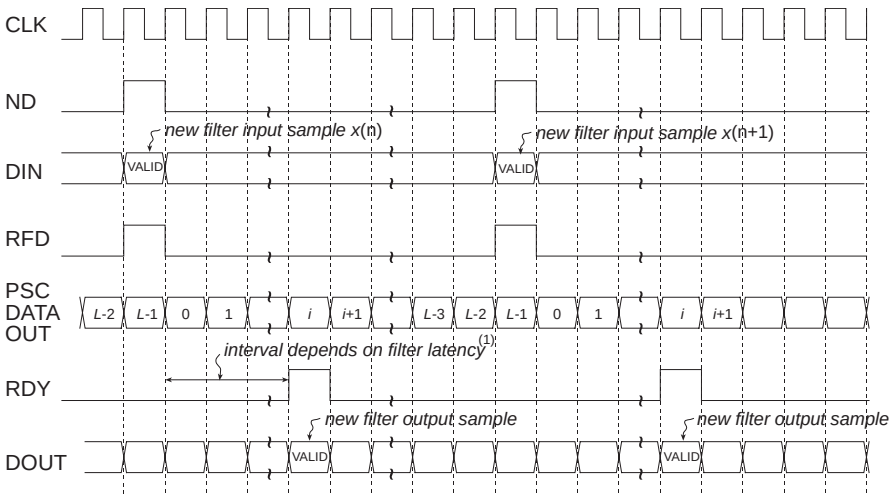
Figure 79 illustrates the timing for a single-channel filter, with *L* clock cycles per output sample and a registered output port. The ND input signal is used for loading a new input sample into the filter. It is effectively used internally as a clock enable, and the actual sample load operation occurs 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 79: **Single-channel FIR Filter Timing, *L*-Clock Cycles per Output Sample, Registered Output**

Figure 80 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.

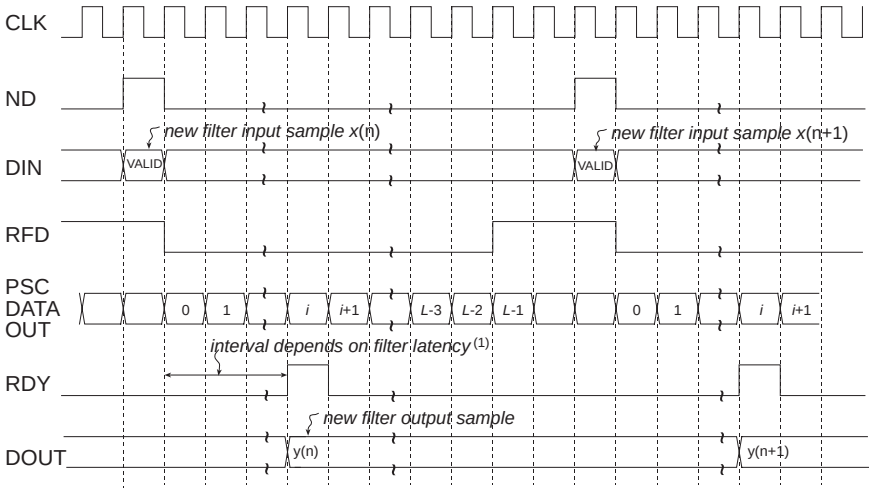


1. The latency is reported on the filter GUI

Figure 80: **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 81.

In this example, despite the filter being designed to specify *L* clock cycles per output sample, new data (input samples) 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 81: ***L*-Clock Cycles per Output Sample, Registered Output**

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 82. Ten clock cycles are needed to process each new input sample.

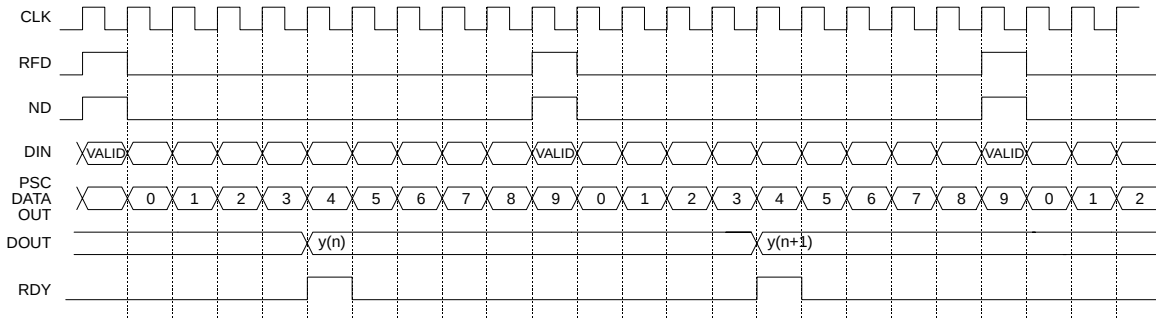


Figure 82: **Full Serial Implementation, 10-bit Input Samples, Registered Output**

A symmetrical filter with *B*-bit precision input samples requires, in general, *B*+1 clock periods for a full serial (SDA) implementation. Figure 83 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.

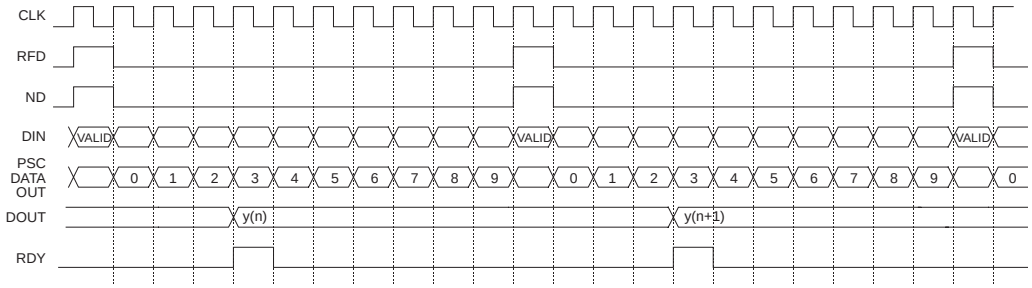


Figure 83: **Full Serial Implementation, 10-Bit Input Samples, Symmetrical Impulse Response, Registered Output**

Figure 82 and Figure 83 illustrate the timing for full serial or SDA filter implementations with symmetrical and non-symmetrical coefficient data. The FIR Compiler 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 automatically generated from the Input Sampling Frequency and Clock Frequency specified in the core GUI.

Figure 84, Figure 85, and Figure 86 illustrate the timing diagrams for a filter with *B*=10 bit precision input samples, registered output, with *L*=2, 4, and, 6, respectively.

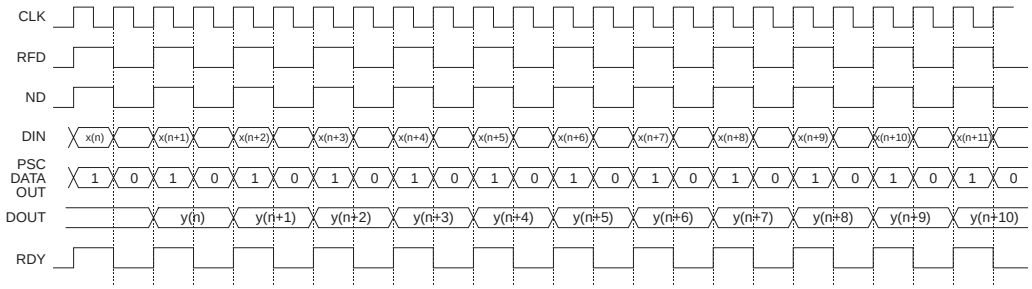


Figure 84: **PDA FIR with *B*=10-Bit Input Samples, *L*=2 Clock Cycles per Output Sample**

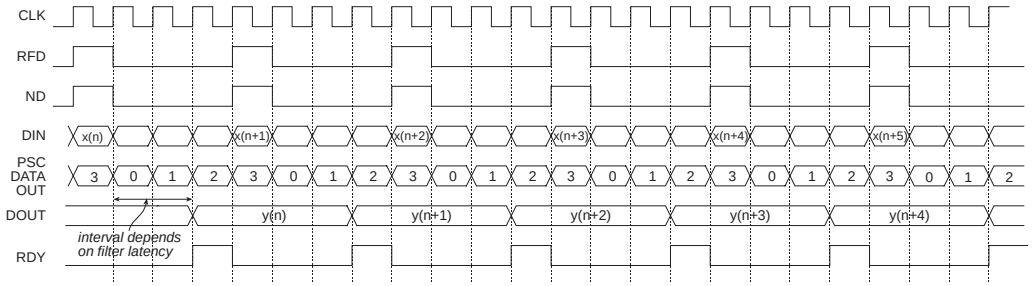


Figure 85: **PDA FIR with *B*=10-Bit Input Samples, *L*=4 Clock Cycles per Output Sample**

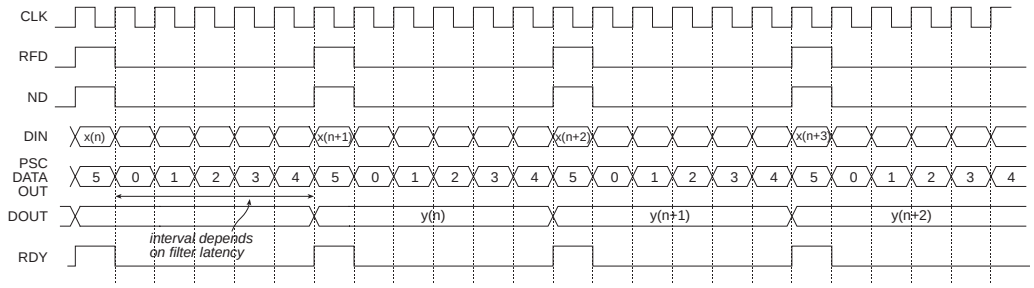


Figure 86: **PDA FIR with *B*=10-Bit Input Samples, *L*=6 Clock Cycles per Output Sample**

Figure 87 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.

Figure 87 shows ND valid on every clock edge, so a new input sample is delivered to the filter on each clock edge. Of course, ND can be removed for an arbitrary number of clock cycles 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).

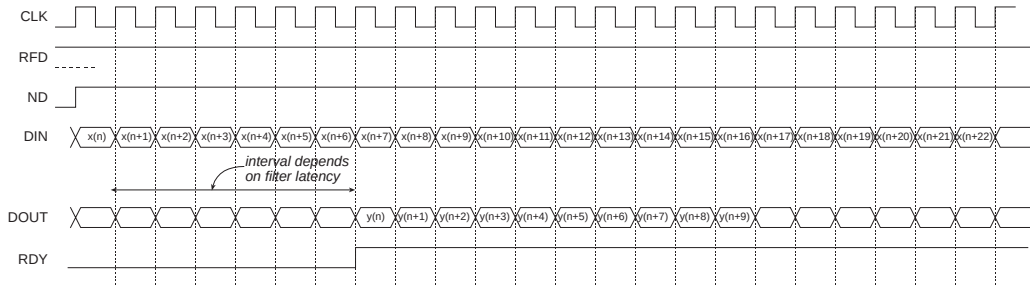


Figure 87: **Fully Parallel Implementation, Single-channel Filter**

Figure 88 and Figure 89 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 88 illustrates a filter with an unregistered output. 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.

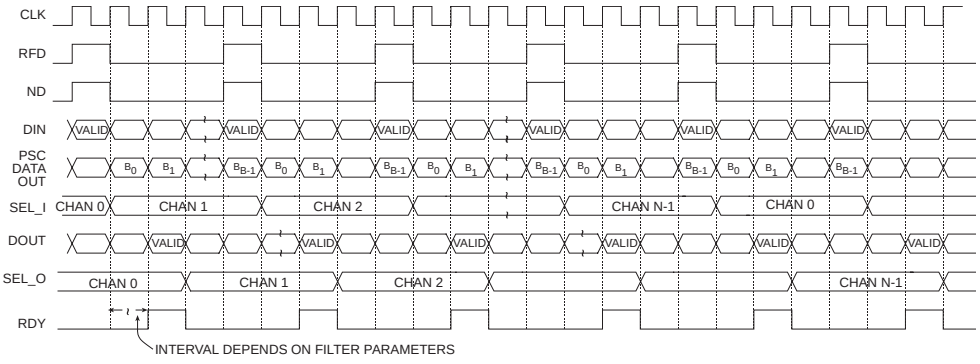


Figure 88: **Multi-channel FIR Filter Timing (Direct Output)**

Figure 89 shows the Multi-channel FIR filter timing for registered output samples.

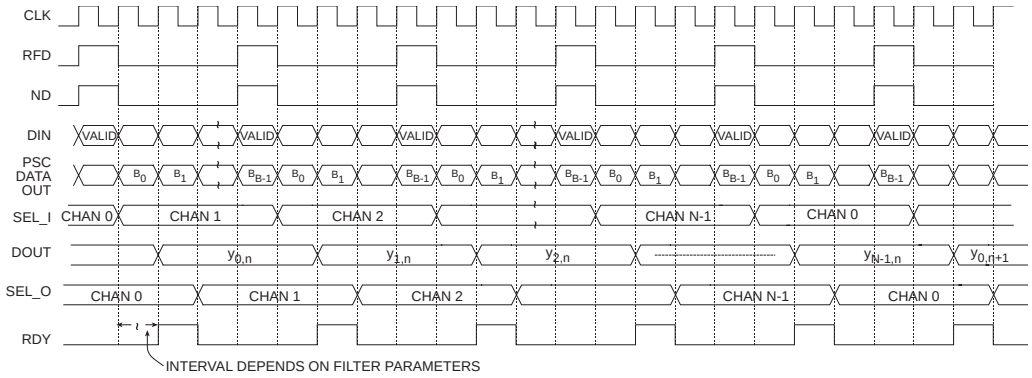


Figure 89: **Multi-channel FIR Filter Timing (Registered Output)**

Figure 90 demonstrates the timing for a polyphase decimator with $M = 4, B = 8$ and eight clock cycles per output point (*Clock Cycles/Output Sample=8*). As previously stated, all of the multi-rate filter structures – the number of clock cycles per output point specification (*Clock Cycles/Output Sample*) – see the individual filter segments that comprise the filter, and are not directly associated with the filter output port DOUT.

The filter is always able to accept input samples, as indicated by RFD=1. New output samples become available after *M* (in this case four) input samples have been delivered to the filter. New output samples are produced in response to each new block of four input values. Delivering the final value in each *M*-tuple begins a new inner product calculation. 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, and 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.

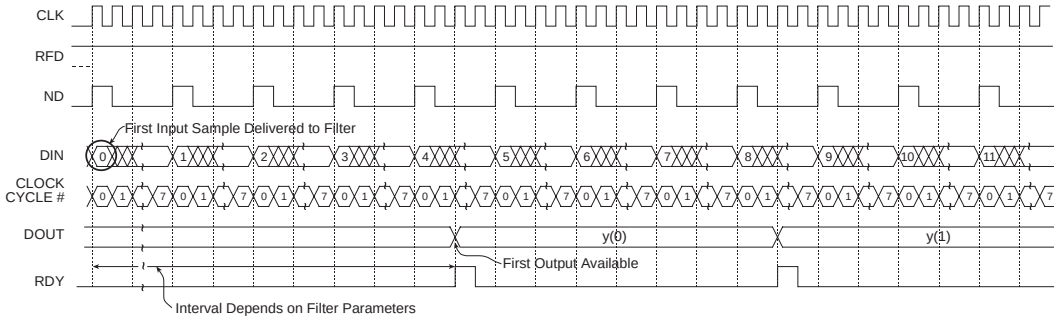


Figure 90: **8-bit Precision Input Samples, Down-sampling Factor *M*=4, *L*=8**

Figure 90 illustrates the timing for a 4-to-1 polyphase decimator with similar parameters to the filter considered in Figure 88, 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 88, samples can be presented to filter every four clock cycles, in contrast to every eight clock periods in the previous example. The filter supports double the input sample rate and, therefore, twice the bandwidth, of the filter with *L*=8.

Polyphase Decimator DA FIR Filter Timing: Burst Input Mode

Internal buffering in the polyphase decimator allows the user to burst samples into the DIN port. This is illustrated in Figure 91 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 eight samples can be written to the filter before the device removes RFD.

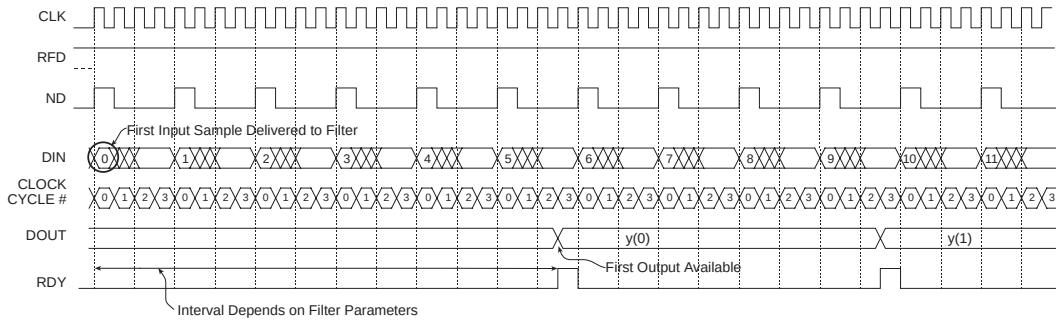


Figure 91: **8-bit Precision Input Samples, Down-sampling Factor *M*=4, *L*=4**

After the filter has moved out of this start-up state, input samples must obey the timing diagram shown in Figure 92. Only four samples can be supplied in each data burst.

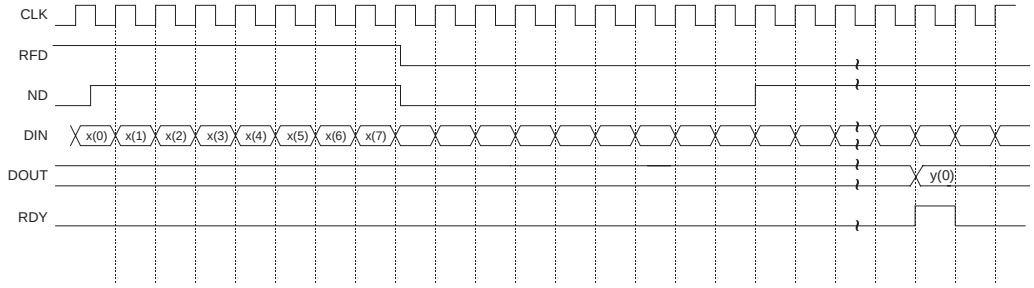


Figure 92: **Polyphase Decimator Timing, Filter out of Start-up State**

As with the *Clock Cycles/Output Sample* parameter for the single-rate filters, this parameter can be used with all the multi-rate filters to trade off performance with silicon area. Figure 93 shows the polyphase decimator timing with 12-bit precision input samples, down-sampling factor *M*=4, *L*=12, and burst input data operation. This diagram shows timing after the filter has moved out of the start-up timing.

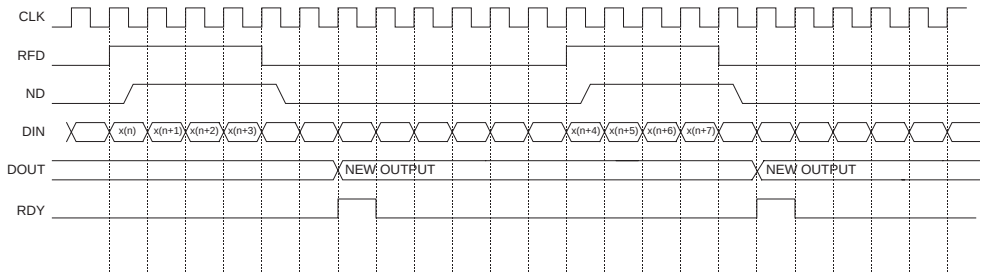


Figure 93: **Polyphase Decimator Timing, 12-Bit Precision Samples**

Polyphase Interpolator DA FIR Filter Timing

Figure 94 shows the timing for a polyphase interpolator that supports a sample rate change of *P*=4, 8-bit precision input samples (*B*=8) and eight 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 sub-filters in the polyphase structure. In this example, each sub-filter produces a new output sample every eight clock cycles. The four polyphase segments are actually operating concurrently so, in fact, internal to the filter, four new output samples are available every eight 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 multiplexer select signal is referenced to the filter master clock signal CLK. As shown in Figure 94, the vector of *P* output samples is validated by the core output control signal RDY.

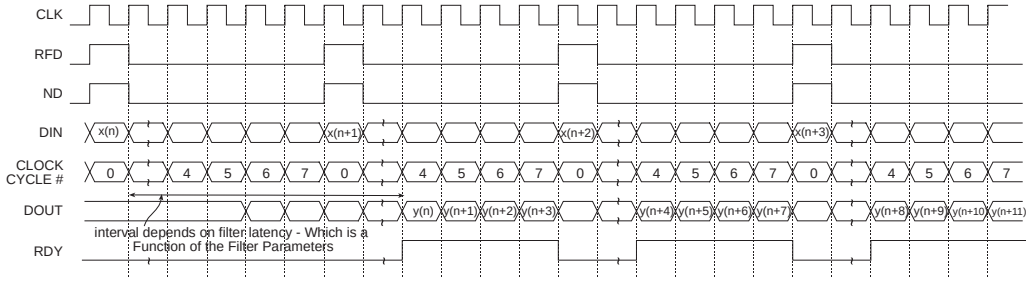


Figure 94: **Polyphase Interpolator Timing, 8-Bit Precision Input Samples, Up-sampling Factor *P*=4, *L*=8**

Figure 95 shows the timing for an interpolator with similar parameters to the previous example, but in this case a value of *L*=4 has been used. This means that each polyphase segment produces a new output sample every four clock cycles. In addition, all four 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.

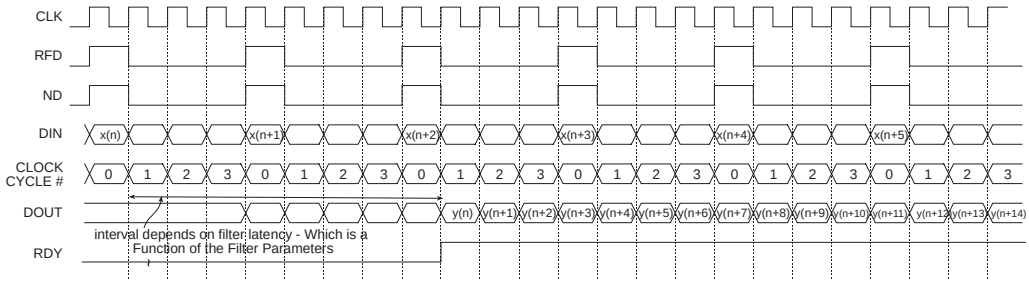


Figure 95: **Polyphase Interpolator Timing, 8-Bit Precision Input Samples, Up-sampling Factor *P*=4, *L*=4**

Performance and Resource Utilization

This section provides indicative resource utilization figures, for example, filters in various families and using both MAC- and DA-based architectures. To be concise, codes are used in these tables to indicate particular configuration options; these are detailed in the following sections.

The maximum clock frequency results were obtained by double-registering input and output ports (using IOB flip-flops) to reduce dependence on I/O placement. The inner level of registers used a separate clock signal to measure the path from the input registers to the first output register through the core.

The resource usage results do not include the preceding "characterization" registers and represent the true logic used by the core. LUT counts include SRL16s or SRL32s (according to device family).

The map options used were: `"map -ol high"`

The par options used were: `"par -ol high"`

Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification.

The maximum achievable clock frequency and the resource counts may also be affected by other tool options, additional logic in the FPGA device, using a different version of Xilinx tools, and other factors.

Control Structure Options

`ND` indicates flow control based on the use of the New Data input pin to validate input samples.

`CE` indicates control based on Clock Enable only, with no `ND` input pin to validate input data samples.

The Interface, Control, and Timing section contains full details of the various control signals.

Rounding Style Options

The rounding option codes shown in Table 9 are used in the resource utilization tables. Note that these options are only applicable to MAC-based filter implementation on device families with XtremeDSP Slices. Only a limited number of rounding examples are provided. See Table 7 in the Resource Implications of Rounding section for a breakdown of the filter types and families that require an additional DSP slice for rounding.

Table 9: **Rounding Style Options in Resource Utilization Tables**

Table Entry	Rounding Style
None	Full Precision; no reduction in output sample width
a	Truncation to input data sample width + 2
b	Non-symmetric Rounding Down, reducing to input data sample width + 2
c	Non-symmetric Rounding Up, reducing to input data sample width + 2
d	Symmetric Rounding to Zero, reducing to input data sample width + 2
e	Approximated Symmetric Rounding to Zero, reducing to input data sample width + 2
f	Symmetric Rounding to Infinity, reducing to input data sample width + 2
g	Approximated Symmetric Rounding to Infinity, reducing to input data sample width + 2
h	Convergent Rounding to Even, reducing to input data sample width + 2
j	Convergent Rounding to Odd, reducing to input data sample width + 2

Resource Utilization for MAC-based FIR Filters (Virtex-6 FPGA)

Table 10 provides characterization data for Virtex-6 FPGAs using a XC6VLX75T-1FF784. Generally the overall filter performance is within 10% of the DSP slice clock rating for the given device speed grade (for example, 472 MHz in -1), and often reaches this clock rate (although the Speed setting may be required to achieve this in some cases). Some fully parallel cases can be slower due to routing congestion. Note that block RAM counts quoted are for 18k blocks, which are often amalgamated into pairs for mapping to 36k locations where possible; therefore customers should bear this in mind if comparing these values with map results for their particular configuration.

Table 10: **MAC-based FIR Resource Utilization in Virtex-6 FPGAs**

Filter Type	Rate	# Coefficients	Symmetric	Half-band	Reloadable	Rounding Style	Channels	Clocks/Sample / Channel	Input Width	Coefficient Width	Control Structure	Area/Speed	DSP48	Block RAM	LUT-FF pairs	Clock Fmax (MHz)
Single Rate	1	366					1	366	18	18	ND	A	1	1	127	450
Single Rate	1	4					4	1	18	18	ND	A	4	0	129	452
Single Rate	1	20					1	5	18	18	ND	A	5	0	194	472
Single Rate	1	20					3	5	18	18	ND	A	5	0	206	472
Single Rate	1	27					1	1	18	18	ND	A	27	0	86	472
Single Rate	1	21	✔				2	1	17	18	ND	A	11	0	432	472
Decimation	6	34	✔				1	3	16	16	ND	A	1	0	176	472
Decimation	2	69	✔				1	18	16	16	ND	A	1	0	202	472
Single Rate	1	19	✔				6	1	16	16	ND	A	10	0	368	472
Single Rate	1	32					1	32	16	16	ND	A	1	0	202	472
Single Rate	1	32					1	4	16	16	ND	A	9	0	236	472
Single Rate	1	32					1	1	16	16	ND	A	32	0	78	472
Single Rate	1	32	✔				1	32	16	16	ND	A	1	0	137	472
Single Rate	1	32	✔				1	4	16	16	ND	A	5	0	265	472
Single Rate	1	32	✔				1	1	16	16	ND	A	16	0	501	472
Single Rate	1	32					3	4	16	16	ND	A	9	0	256	472
Single Rate	1	32					3	1	16	16	ND	A	29	0	584	472
Single Rate	1	32	✔				3	4	16	16	ND	A	5	0	318	472
Single Rate	1	32	✔				3	1	16	16	ND	A	13	0	656	472
Single Rate	1	31	✔	✔			3	4	16	16	ND	A	3	0	234	472
Interpolation	5	32					1	20	16	16	ND	A	3	0	140	472
Interpolation	5	32					3	20	16	16	ND	A	3	0	205	472
Interpolation	5	61	✔				3	5	16	16	ND	A	8	0	414	472

Table 10: **MAC-based FIR Resource Utilization in Virtex-6 FPGAs *(Cont'd)***

Filter Type	Rate	# Coefficients	Symmetric	Half-band	Reloadable	Rounding Style	Channels	Clocks/Sample / Channel	Input Width	Coefficient Width	Control Structure	Area/Speed	DSP48	Block RAM	LUT-FF pairs	Clock Fmax (MHz)
Interpolation	5	61	✔				3	20	16	16	ND	A	3	0	337	472
Interpolation	2	31	✔	✔			1	8	16	16	ND	A	2	0	230	472
Interpolation	5/3	64					3	10	16	16	ND	A	4	0	240	472
Decimation	5	32					1	4	16	16	ND	A	3	0	143	472
Decimation	5	32					3	4	16	16	ND	A	3	0	292	472
Decimation	5	64	✔				3	1	16	16	ND	A	7	0	566	472
Decimation	5	64	✔				3	4	16	16	ND	A	3	0	416	450
Decimation	5	64	✔				3	13	16	16	ND	A	1	1	254	450
Decimation	2	31	✔	✔			1	3	16	16	ND	A	5	0	344	472
Decimation	3/5	64					3	10	16	16	ND	A	4	0	270	472
Interpolation	16	288			✔		16	16	18	18	CE	A	18	0	1210	450
Interpolation	8	144			✔		8	32	18	18	CE	A	13	5	549	450
Interpolation	36/25	144					2	6	18	18	ND	A	1	1	172	450
Interpolation	2	11	✔	✔			2	6	17	18	ND	A	1	0	201	472
Interpolation	2	15	✔	✔			2	12	16	18	ND	A	1	0	203	472
Interpolation	2	251	✔				2	24	16	18	ND	A	7	0	540	472
Single Rate	1	32				f	1	33	16	16	ND	A	1	0	92	472
Single Rate	1	32				f	1	32	16	16	ND	A	2	0	90	472
Single Rate	1	32				e	1	32	16	16	ND	A	1	0	90	472
Single Rate	1	32				h	1	4	16	16	ND	A	9	0	215	429
Single Rate[4]	1	32					1	4	16	16	ND	A	8	0	101	472
Interpolation[4]	5	32					1	20	16	16	ND	A	2	0	128	466
Decimation[4]	5	32					1	4	16	16	ND	A	2	0	121	452
Receiver Filter Bank[5]	1	256					16	64	16	16	ND	A	2	3	354	445

Notes:

1. Clock rates determined using a -1 speed grade.
2. Clocks per sample per channel uses the input sample rate as the basis for all filter types.
3. Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification.
4. Implemented using Transpose Multiply-Accumulate architecture.
5. Implements two parallel data paths (I and Q).

Resource Utilization for MAC-based FIR Filters (Virtex-5 FPGA)

Table 11 provides characterization data for Virtex-5 FPGAs using a XC5VSX35T-1FF665. Generally the overall filter performance is within 10% of the DSP slice clock rating for the given device speed grade (for example, 450 MHz in -1), and often reaches this clock rate (although the Speed setting may be required to achieve this in some cases). Some fully parallel cases can be slower due to routing congestion. Note that block RAM counts quoted are for 18k blocks, which are often amalgamated into pairs for mapping to 36k locations where possible; therefore customers should bear this in mind if comparing these values with map results for their particular configuration.

Table 11: **MAC-based FIR Resource Utilization in Virtex-5 FPGAs**

Filter Type	Rate	# Coefficients	Symmetric	Half-band	Reloadable	Rounding Style	Channels	Clocks/Sample / Channel	Input Width	Coefficient Width	Control Structure	Area/Speed	DSP48	Block RAM	LUT-FF pairs	Clock Fmax (MHz)
Single Rate	1	366					1	366	18	18	ND	A	1	1	130	450
Single Rate	1	4					4	1	18	18	ND	A	4	0	146	450
Single Rate	1	20					1	5	18	18	ND	A	5	0	207	450
Single Rate	1	20					3	5	18	18	ND	A	5	0	220	450
Single Rate	1	27					1	1	18	18	ND	A	27	0	118	450
Single Rate	1	21	✔				2	1	17	18	ND	A	11	0	669	450
Decimation	6	34	✔				1	3	16	16	ND	A	1	0	195	450
Decimation	2	69	✔				1	18	16	16	ND	A	1	0	292	450
Single Rate	1	19	✔				6	1	16	16	ND	A	10	0	515	450
Single Rate	1	32					1	32	16	16	ND	A	1	0	116	450
Single Rate	1	32					1	4	16	16	ND	A	9	0	773	450
Single Rate	1	32					1	1	16	16	ND	A	32	0	112	450
Single Rate	1	32	✔				1	32	16	16	ND	A	1	0	168	450
Single Rate	1	32	✔				1	4	16	16	ND	A	5	0	897	450
Single Rate	1	32	✔				1	1	16	16	ND	A	16	0	775	450
Single Rate	1	32					3	4	16	16	ND	A	9	0	288	450
Single Rate	1	32					3	1	16	16	ND	A	29	0	616	450
Single Rate	1	32	✔				3	4	16	16	ND	A	5	0	411	450
Single Rate	1	32	✔				3	1	16	16	ND	A	16	0	914	450
Single Rate	1	31	✔	✔			3	4	16	16	ND	A	3	0	287	450
Interpolation	5	32					1	20	16	16	ND	A	3	0	276	450
Interpolation	5	32					3	20	16	16	ND	A	3	0	224	450
Interpolation	5	61	✔				3	5	16	16	ND	A	8	0	573	443

Table 11: **MAC-based FIR Resource Utilization in Virtex-5 FPGAs *(Cont'd)***

Filter Type	Rate	# Coefficients	Symmetric	Half-band	Reloadable	Rounding Style	Channels	Clocks/Sample / Channel	Input Width	Coefficient Width	Control Structure	Area/Speed	DSP48	Block RAM	LUT-FF pairs	Clock Fmax (MHz)
Interpolation	5	61	✔				3	20	16	16	ND	A	3	0	402	447
Interpolation	2	31	✔	✔			1	8	16	16	ND	A	2	0	267	442
Interpolation	5/3	64					3	10	16	16	ND	A	4	0	262	449
Decimation	5	32					1	4	16	16	ND	A	3	0	150	450
Decimation	5	32					3	4	16	16	ND	A	3	0	339	447
Decimation	5	64	✔				3	1	16	16	ND	A	7	0	794	446
Decimation	5	64	✔				3	4	16	16	ND	A	3	0	501	442
Decimation	5	64	✔				3	13	16	16	ND	A	1	1	794	446
Decimation	2	31	✔	✔			1	3	16	16	ND	A	5	0	412	427
Decimation	3/5	64					3	10	16	16	ND	A	4	0	307	449
Interpolation	16	288			✔		16	16	18	18	CE	A	18	0	1576	450
Interpolation	8	144			✔		8	32	18	18	CE	A	6	5	533	424
Interpolation	36/25	144					2	6	18	18	ND	A	1	1	183	450
Interpolation	2	11	✔	✔			2	6	17	18	ND	A	1	0	241	432
Interpolation	2	15	✔	✔			2	12	16	18	ND	A	1	0	243	442
Interpolation	2	251	✔				2	24	16	18	ND	A	7	0	746	434
Single Rate	1	32				f	1	33	16	16	ND	A	1	0	102	450
Single Rate	1	32				f	1	32	16	16	ND	A	2	0	95	450
Single Rate	1	32				e	1	32	16	16	ND	A	1	0	98	450
Single Rate	1	32				h	1	4	16	16	ND	A	9	0	753	410
Single Rate[4]	1	32					1	4	16	16	ND	A	8	0	120	450
Interpolation[4]	5	32					1	20	16	16	ND	A	2	0	152	450
Decimation[4]	5	32					1	4	16	16	ND	A	2	0	135	450
Receiver Filter Bank[5]	1	256					16	64	16	16	ND	A	2	3	383	423

Notes:

1. Clock rates determined using a -1 speed grade.
2. Clocks per sample per channel uses the input sample rate as the basis for all filter types.
3. Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification.
4. Implemented using Transpose Multiply-Accumulate architecture.
5. Implements two parallel data paths (I and Q).

Resource Utilization for MAC-based FIR Filters (Spartan-6)

Table 12 provides characterization data for Spartan-6 FPGAs using a XC6SLX150-2FGG484. Generally the overall filter performance is within 10% of the DSP slice clock rating for the given device speed grade (for example, 250 MHz in -2), and often reaches this clock rate (although the Speed setting may be required to achieve this in some cases). Some fully parallel cases can be slower due to routing congestion.

Table 12: **MAC-based FIR Resource Utilization in Spartan-6 FPGAs**

Filter Type	Rate	# Coefficients	Symmetric	Half-band	Reloadable	Rounding Style	Channels	Clocks/Sample / Channel	Input Width	Coefficient Width	Control Structure	Area/Speed	DSP48	Block RAM	LUT-FF pairs	Clock Fmax (MHz)
Single Rate	1	366					1	366	18	18	ND	A	1	1	118	244
Single Rate	1	4					4	1	18	18	ND	A	4	0	101	251
Single Rate	1	20					1	5	18	18	ND	A	5	0	152	251
Single Rate	1	20					3	5	18	18	ND	A	5	0	200	251
Single Rate	1	27					1	1	18	18	ND	A	27	0	86	236
Single Rate	1	21	✔				2	1	17	18	ND	A	11	0	320	251
Decimation	6	34	✔				1	3	16	16	ND	A	1	0	167	251
Decimation	2	69	✔				1	18	16	16	ND	A	1	0	180	251
Single Rate	1	19	✔				6	1	16	16	ND	A	10	0	276	251
Single Rate	1	32					1	32	16	16	ND	A	1	0	91	251
Single Rate	1	32					1	4	16	16	ND	A	9	0	156	251
Single Rate	1	32					1	1	16	16	ND	A	32	0	70	172
Single Rate	1	32	✔				1	32	16	16	ND	A	1	0	104	251
Single Rate	1	32	✔				1	4	16	16	ND	A	5	0	183	251
Single Rate	1	32	✔				1	1	16	16	ND	A	16	0	307	251
Single Rate	1	32					3	4	16	16	ND	A	9	0	171	251
Single Rate	1	32					3	1	16	16	ND	A	29	0	320	212
Single Rate	1	32	✔				3	4	16	16	ND	A	5	0	207	251
Single Rate	1	32	✔				3	1	16	16	ND	A	13	0	491	251
Single Rate	1	31	✔	✔			3	4	16	16	ND	A	3	0	197	251
Interpolation	5	32					1	20	16	16	ND	A	3	0	103	251
Interpolation	5	32					3	20	16	16	ND	A	3	0	175	251
Interpolation	5	61	✔				3	5	16	16	ND	A	8	0	358	251
Interpolation	5	61	✔				3	20	16	16	ND	A	3	0	277	236
Interpolation	2	31	✔	✔			1	8	16	16	ND	A	3	0	199	251
Interpolation	5/3	64					3	10	16	16	ND	A	4	0	229	244
Decimation	5	32					1	4	16	16	ND	A	3	0	130	251

Table 12: **MAC-based FIR Resource Utilization in Spartan-6 FPGAs *(Cont'd)***

Filter Type	Rate	# Coefficients	Symmetric	Half-band	Reloadable	Rounding Style	Channels	Clocks/Sample / Channel	Input Width	Coefficient Width	Control Structure	Area/Speed	DSP48	Block RAM	LUT-FF pairs	Clock Fmax (MHz)
Decimation	5	32					3	4	16	16	ND	A	3	0	274	251
Decimation	5	64	✔				3	1	16	16	ND	A	8	0	505	251
Decimation	5	64	✔				3	4	16	16	ND	A	3	0	398	244
Decimation	5	64	✔				3	13	16	16	ND	A	1	1	247	251
Decimation	2	31	✔	✔			1	3	16	16	ND	A	5	0	241	244
Decimation	3/5	64					3	10	16	16	ND	A	4	0	257	251
Interpolation	16	288			✔		8	24	16	16	CE	A	18	0	1679	172
Interpolation	4	16					2	4	16	16	CE	A	4	0	541	203
Decimation	6	31	✔		✔		2	1	16	16	CE	A	4	0	244	251
Interpolation	8	144			✔		8	24	16	16	CE	A	7	0	644	219
Single Rate	1	32				f	1	33	16	16	ND	A	2	0	83	251
Single Rate	1	32				f	1	32	16	16	ND	A	2	0	81	251
Single Rate	1	32				g	1	32	16	16	ND	A	2	0	81	251
Single Rate	1	32				b	1	4	16	16	ND	A	9	0	375	251
Single Rate[4]	1	32					1	4	16	16	ND	A	8	0	97	251
Interpolation[4]	5	32					1	20	16	16	ND	A	2	0	125	251
Decimation[4]	5	32					1	4	16	16	ND	A	2	0	114	244
Receiver Filter Bank[5]	1	256					16	64	16	16	ND	A	2	3	287	244

Notes:

1. Clock rates determined using a -2 speed grade.
2. Clocks per sample per channel uses the input sample rate as the basis for all filter types.
3. Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification.
4. Implemented using Transpose Multiply-Accumulate architecture.
5. Implements two parallel data paths (I and Q).

Resource Utilization for MAC-based FIR Filters (Spartan-3A DSP)

Table 13 provides characterization data for Spartan-3A DSP FPGAs using a XC3SD1800A-4FG676. Generally the overall filter performance is within 10% of the DSP slice clock rating for the given device speed grade (for example, 250 MHz in -4), and often reaches this clock rate (although the Speed setting may be required to achieve this in some cases). Some fully parallel cases can be slower due to routing congestion.

Table 13: **MAC-based FIR Resource Utilization in Spartan-3A DSP FPGAs**

Filter Type	Rate	# Coefficients	Symmetric	Half-band	Reloadable	Rounding Style	Channels	Clocks/Sample / Channel	Input Width	Coefficient Width	Control Structure	Area/Speed	DSP48	Block RAM	Slices	Clock Fmax (MHz)
Single Rate	1	366					1	366	18	18	ND	A	1	1	95	250
Single Rate	1	4					4	1	18	18	ND	A	4	0	77	250
Single Rate	1	20					1	5	18	18	ND	A	5	0	122	250
Single Rate	1	20					3	5	18	18	ND	A	5	0	158	250
Single Rate	1	27					1	1	18	18	ND	A	27	0	179	250
Single Rate	1	21	✔				2	1	17	18	ND	A	11	0	307	250
Decimation	6	34	✔				1	3	16	16	ND	A	1	2	124	250
Decimation	2	69	✔				1	18	16	16	ND	A	1	0	118	246
Single Rate	1	19	✔				6	1	16	16	ND	A	10	0	323	250
Single Rate	1	32					1	32	16	16	ND	A	1	0	79	250
Single Rate	1	32					1	4	16	16	ND	A	9	0	141	250
Single Rate	1	32					1	1	16	16	ND	A	32	0	170	250
Single Rate	1	32	✔				1	32	16	16	ND	A	1	0	92	250
Single Rate	1	32	✔				1	4	16	16	ND	A	5	0	169	250
Single Rate	1	32	✔				1	1	16	16	ND	A	16	0	302	250
Single Rate	1	32					3	4	16	16	ND	A	9	0	157	250
Single Rate	1	32					3	1	16	16	ND	A	29	0	428	250
Single Rate	1	32	✔				3	4	16	16	ND	A	5	0	185	250
Single Rate	1	32	✔				3	1	16	16	ND	A	16	0	489	250
Single Rate	1	31	✔	✔			3	4	16	16	ND	A	3	0	175	250
Interpolation	5	32					1	20	16	16	ND	A	3	0	94	250
Interpolation	5	32					3	20	16	16	ND	A	3	0	144	250
Interpolation	5	61	✔				3	5	16	16	ND	A	8	2	286	250
Interpolation	5	61	✔				3	20	16	16	ND	A	3	2	215	250
Interpolation	2	31	✔	✔			1	8	16	16	ND	A	3	0	175	250
Interpolation	5/3	64					3	10	16	16	ND	A	4	0	183	250
Decimation	5	32					1	4	16	16	ND	A	3	0	108	250

Table 13: **MAC-based FIR Resource Utilization in Spartan-3A DSP FPGAs *(Cont'd)***

Filter Type	Rate	# Coefficients	Symmetric	Half-band	Reloadable	Rounding Style	Channels	Clocks/Sample / Channel	Input Width	Coefficient Width	Control Structure	Area/Speed	DSP48	Block RAM	Slices	Clock Fmax (MHz)
Decimation	5	32					3	4	16	16	ND	A	3	2	210	250
Decimation	5	64	✔				3	1	16	16	ND	A	8	0	514	248
Decimation	5	64	✔				3	4	16	16	ND	A	3	2	260	250
Decimation	5	64	✔				3	13	16	16	ND	A	1	2	220	250
Decimation	2	31	✔	✔			1	3	16	16	ND	A	5	0	217	250
Decimation	3/5	64					3	10	16	16	ND	A	4	4	175	230
Interpolation	16	288			✔		8	24	16	16	CE	A	18	18	809	237
Interpolation	4	16					2	4	16	16	CE	A	4	0	308	250
Decimation	6	31	✔		✔		2	1	16	16	CE	A	4	0	199	250
Interpolation	8	144			✔		8	24	16	16	CE	A	7	6	439	238
Single Rate	1	32				f	1	33	16	16	ND	A	2	0	69	250
Single Rate	1	32				f	1	32	16	16	ND	A	2	0	69	250
Single Rate	1	32				g	1	32	16	16	ND	A	2	0	69	250
Single Rate	1	32				b	1	4	16	16	ND	A	9	0	131	250
Single Rate[4]	1	32					1	4	16	16	ND	A	8	0	75	250
Interpolation[4]	5	32					1	20	16	16	ND	A	2	1	88	249
Decimation[4]	5	32					1	4	16	16	ND	A	2	1	72	250
Receiver Filter Bank[5]	1	256					16	64	16	16	ND	A	2	5	218	240

Notes:

1. Clock rates determined using a -4 speed grade.
2. Clocks per sample per channel uses the input sample rate as the basis for all filter types.
3. Clock frequency does not take clock jitter into account and should be derated by an amount appropriate to the clock source jitter specification.
4. Implemented using Transpose Multiply-Accumulate architecture.
5. Implements two parallel data paths (I and Q).

Resource Utilization for DA-based FIR Filters

The logic utilization for a filter is a function of the filter length, coefficient precision, coefficient symmetry, and input data precision. Table 14 and Table 15 provide logic resource requirements for a number of serial (SDA) filter configurations, while Table 16 shows resources required by parallel (PDA) filters with several different levels of parallelism. A Virtex-5 XC5VSX35t-1FF665 has been used while generating all the results.

Table 14 shows the LUT-FF pairs utilization for several FIR Filter configurations: 10-bit Filter Coefficients, Single-channel, Signed Input; Signed Coefficients, and Unregistered Output.

Table 14: **DA-based Resource Utilization for SDA FIR Filter Configurations**

Filter Length	Symmetry	Input Sample Precision			
		4-bit	8-bit	16-bit	32-bit
4	Symmetric	24	28	38	57
	Non-symmetric	24	28	37	55
8	Symmetric	25	29	39	57
	Non-symmetric	38	42	52	70
32	Symmetric	67	71	81	100
	Non-symmetric	122	127	136	155
80	Symmetric	167	171	182	199
	Non-symmetric	306	310	321	341
128	Symmetric	236	240	251	268
	Non-symmetric	459	464	475	493
256	Symmetric	461	465	474	494

Table 15 shows the LUT-FF pairs utilization for several half-band filter configurations, including 14-bit Filter Coefficients, Single-channel, Signed Input, Signed Coefficients, and Unregistered Output.

Table 15: **DA-based Resource Utilization for Half-band SDA Filter Configurations**

Filter Length	Symmetry	Input Sample Precision		
		8-bit	16-bit	32-bit
7	Symmetric	33	43	61
31	Symmetric	119	138	188
79	Symmetric	373	391	490

Table 16 shows the LUT-FF pairs utilization for several PDA FIR filter configurations, including 12-bit Filter Coefficients and Input Data, 60-Taps, Filter Coefficient Optimization Of, Single-channel, Signed Input, Signed Coefficients, Unregistered Output, and Non-symmetrical Impulse Response. Filter master clock frequency is 150 MHz.

Table 16: **DA-based Resource Utilization for Several PDA FIR Filter Configurations**

Number of Clock Cycles per Output Sample	LUT-FF pairs	Filter Sample Rate[1] (MHz)
1	3382	150
2	1685	75
3	1102	50
4	844	37.5
6	558	25
12	261	12.5

Notes:

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

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. 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
4. P.P. Vaidyanathan, *Multi-Rate 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. Xilinx Inc., *XtremeDSP Design Manual,* Xilinx Inc., San Jose California, 2004.
7. Fred Harris, Chris Dick, and Michael Rice, *Digital Receivers and Transmitters Using Polyphase Filter Banks for Wireless Communications,* IEEE Trans. on Microwave Theory and Techniques, Vol. 51, No.4. 4 April 2003
8. Mou, Zhi-Jian, *Symmetry Exploitation in Digital Interpolators/Decimators*, IEEE Transactions on Signal Processing, Vol. 44 No. 10, Oct. 1996

Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled *DO NOT MODIFY*.

Refer to the IP Release Notes Guide (XTP025) for further information on this core. On the first page there is a link to "All DSP IP." The relevant core can then be selected from the displayed list.

For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Bug Fixes
- Known Issues

Ordering Information

This LogiCORE IP module is included at no additional cost with the Xilinx ISE Design Suite software and is provided under the terms of the Xilinx End User License Agreement. Use the CORE Generator software included with the ISE Design Suite to generate the core. For more information, please visit the core page.

Information about additional Xilinx LogiCORE modules is available at the Xilinx IP Center. For pricing and availability of other Xilinx LogiCORE modules and software, please contact your local Xilinx sales representative.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
01/18/06	1.0	Initial release.
09/28/06	2.0	Updated for v2.0 core, including Virtex-5 family support and additional features.
02/15/07	3.0	Updated for v3.0 core.
04/02/07	3.1	Added support for Spartan-3A DSP devices.
08/08/07	3.2	Added Spartan-3A DSP resource tables, Bit Growth, and Rounding Mode sections.
10/10/07	3.3	Added full feature support for Virtex and Spartan families with Embedded Multipliers.
06/27/08	4.0	Updated for v4.0 core.
06/24/09	5.0	Updated for v5.0 core including support for Virtex-6 and Spartan-6.
04/19/10	5.0.1	Updated for v5.0 core. 12.1 support and Simulator Support section added. Corrections to Interpolated FIR, Polyphase Decimator and Polyphase Interpolator sections.
03/01/11	5.1	Support added for Virtex-7 and Kintex-7. ISE Design Suite 13.1

Notice of Disclaimer