Distributed Arithmetic FIR Filter

Dec10 1999 Product Specification

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

1 Features

- Drop-in module for Virtex™, Virtex-E and Spartan™-II FPGAs
- High-performance finite impulse response (FIR), half-band, Hilbert transform and interpolated filters
- Highly parameterizable
- 2-to-256 tap symmetrical impulse response
- 2-to-128 tap non-symmetrical impulse response
- 1-to-32 bit input data precision
- signed or unsigned input data
- 1-to-32 bit coefficient precision
- 1-to-8 channels
- Coefficient symmetry exploited (symmetric/negative-symmetric) to produce compact implementations
- Data-flow style core interface and control
- High performance and density guaranteed through Relational Placed Macro (RPM) mapping and placement technology
- Incorporates Xilinx Smart-IP technology for maximum performance
- To be used with version 2.1i or later of the Xilinx CORE Generator System

2 General Description

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

The conventional single-rate FIR version of the core computes the convolution sum defined in Eq. (1)

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

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

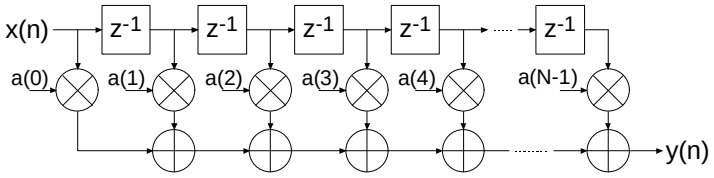

Figure 1: Conventional tapped-delay line FIR filter mechanization.

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

2.1 Filter Realization – *Distributed Arithmetic*

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

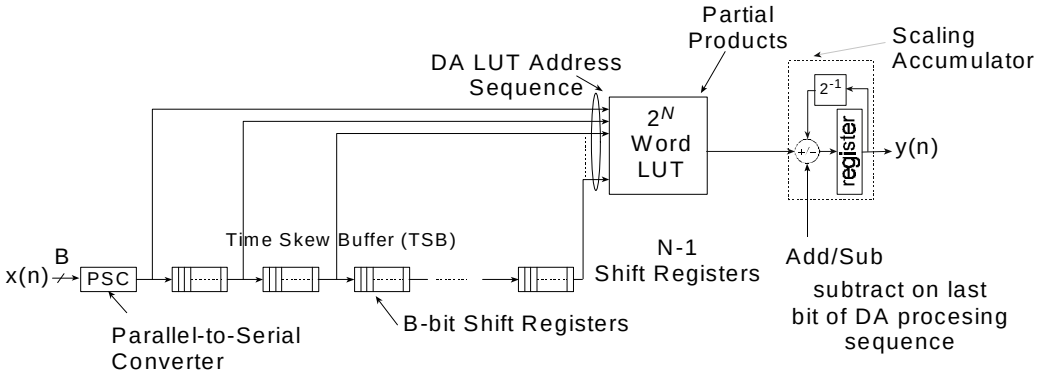

Figure 2: Serial distributed arithmetic FIR filter.

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

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

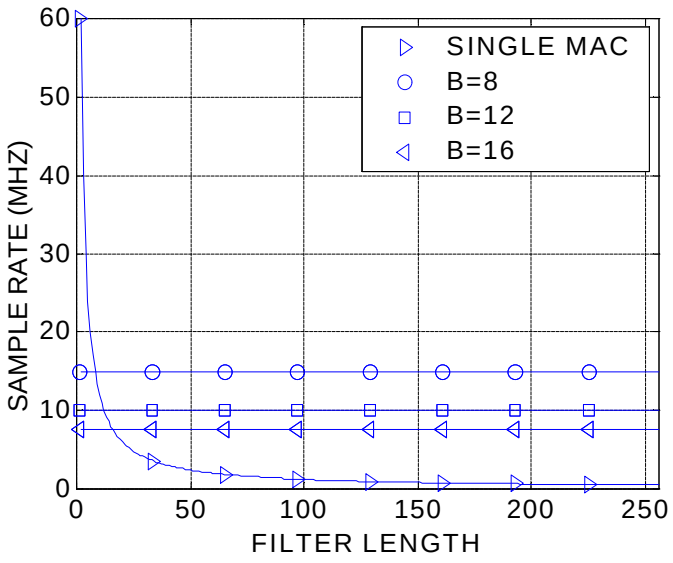

Figure 3: Throughput (sample rate) comparison of single-MAC based FIR and DA FIR as a function of filter length. B is the DA FIR input sample precision. The clock rate is 120 MHz.

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

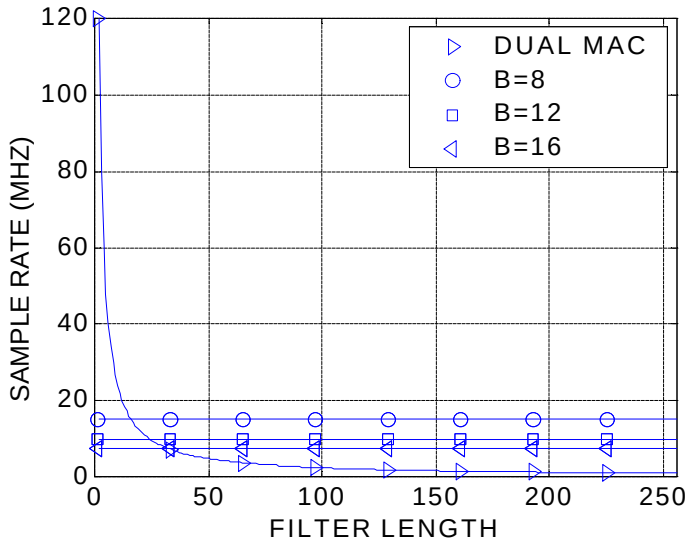

Figure 4: Throughput (sample rate) comparison of dual-MAC based FIR and DA FIR as a function of filter length. B is the DA FIR input sample precision. The clock rate is 120 MHz.

2.2 Exploiting Filter Symmetry

The impulse response for many filters possess significant symmetry. This symmetry can be exploited to minimize arithmetic requirements and produce area efficient filter realizations.
Figure 5 shows the impulse response for a 9-tap symmetric FIR filter.

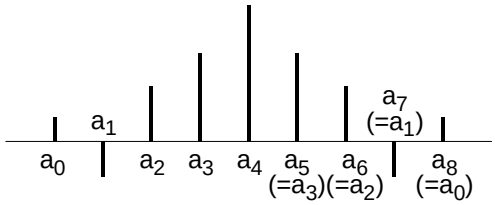

Figure 5: Symmetric FIR – odd number of terms.

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

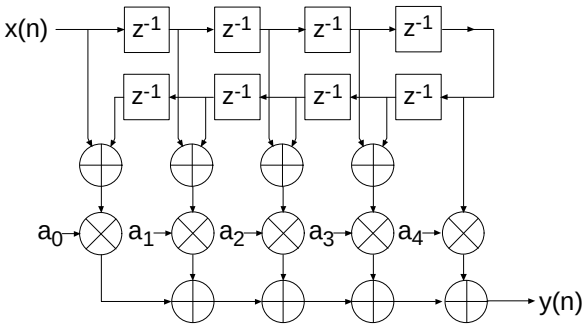

Figure 6: Exploiting coefficient symmetry – odd number of filter taps.

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

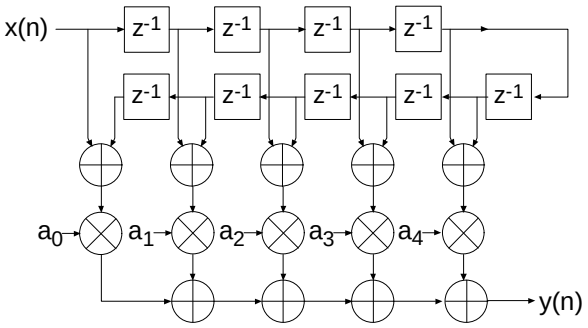

Figure 7: Exploiting coefficient symmetry – even number of filter taps.

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

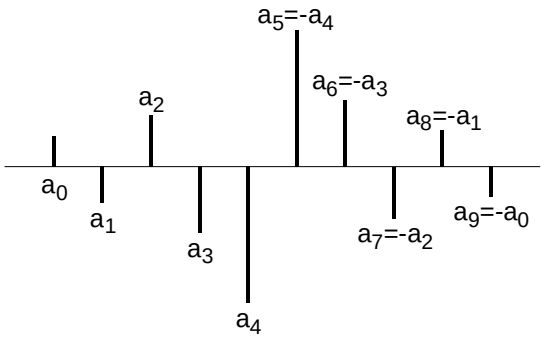

Figure 8: Negative Symmetric impulse response.

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

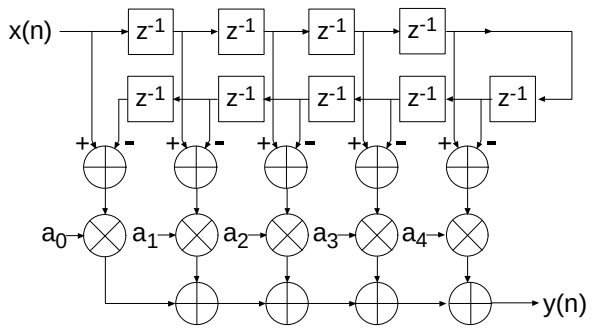

Figure 9: FIR architecture exploiting negative symmetry.

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

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

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

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

3 Filter Throughput

The signal sample rate for a filter is a function of the core bit clock frequency, fclk Hz, the input data sample precision *B,* the number of channels and the coefficient symmetry. For a single channel non-symmetrical FIR filter, the filter sample frequency, or sample throughput, is fclk/B Hz. If the filter is symmetrical, then the sample rate is fclk/(*B*+1) Hz. The coefficient symmetry must be explicitly specified in the GUI, it is not automatically detected by inspection of the coefficient set. A filter core can be generated using a symmetrical coefficient set, but specifying a non-symmetrical implementation. In this case the sample rate is the same as for a non-symmetrical filter – fclk/*B* Hz.

As an example, consider a filter with a core clock frequency equal to 100 MHz ,10-bit input samples, 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.
If the input sample precision is changed to 8 bits, the filter sample rate for a non-symmetrical filter would be 100/8 = 12.5 MHz.

4 Processing Multiple Channels

In many applications the same filter must be applied to several data streams. A common example is the simple digital down converter shown in Figure 10. Here a complex base-band signal

$x(n) = x_I(n) + jx_Q(n)$ is applied to a matched filter M(z). The in-phase and quadrature components are each processed by the same filter.

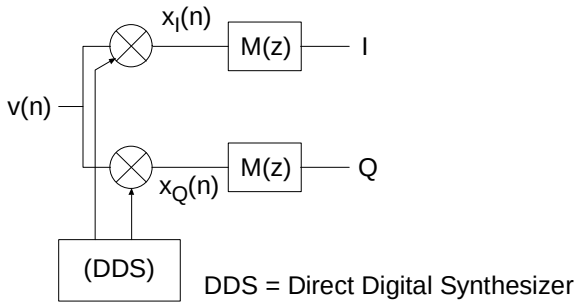

Figure 10: Digital down converter.

One candidate solution to this problem is to simply employ two separate filters. However, this can be wasteful of logic resources. A more efficient design can be realized using a filter architecture that shares logic resources between multiple sample streams. All of the filter classes supported by the filter core provide in-built support for multi-channel processing and can accommodate up to 8 independent data streams. As more channels are processed by a filter core, the sample throughput is commensurately reduced. For example, if the sample rate (not the core bit clock *CLK*) for a single channel filter is f_S, a two-channel version of the same filter will process two sample streams, each with a sample rate of $f_S/2$. A three channel version of the filter will process three data streams, and support a sample rate of $f_S/3$ for each of the streams.

A multi-channel filter implementation is very efficient in terms of the amount of logic resources utilized. A filter with two or more channels can be realized using virtually the same amount of logic resources as a single channel version of the same filter. The tradeoff that needs to be addressed when employing multi-channel filters is one of sample rate versus logic requirements. As the number of channels is increased, the logic area remains approximately constant, but the sample rate for an individual input stream will decrease.

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

5 Filter Configurations

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

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

5.1 Single Rate FIR

The FIR filter core is a single-rate (input sample rate = output sample rate) finite impulse response filter. Figure 11 shows the schematic symbol for a single channel instance of this

module. Filter input data is supplied on the *DIN* port and filter output samples are presented on the *DOUT* port. The *CLK* signal is the bit-rate clock for the core, and is recognized as being different (higher frequency) to the input signal sample frequency. The *ND, RDY* and *RFD* signals are filter interface/control signals that permit a simple and efficient data-flow style interface for supplying input samples and reading output samples from the filter. The core interface signals are discussed in detail in the *Interface and Control* section of the product guide.

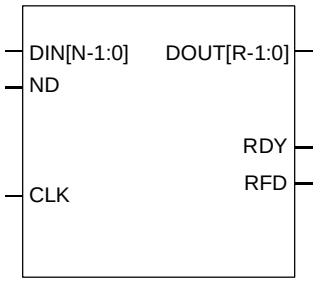

Figure 11: Single channel FIR symbol.

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

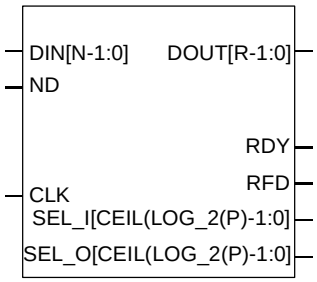

Figure 12: Multi-channel FIR symbol.

The pin definitions are provided in Table 1.

5.2 Half-Band FIR

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

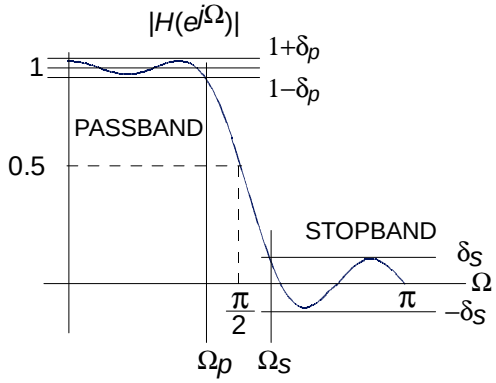

Figure 13: Half-band filter – magnitude frequency response.

Observe from the figure that the magnitude frequency response is symmetrical about quarter sample frequency $\pi/2$ radians. The sample rate is normalized to 2π radians/sec. The passband and stopband frequencies are positioned such that

$$\Omega_p = p - \Omega_s$$

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

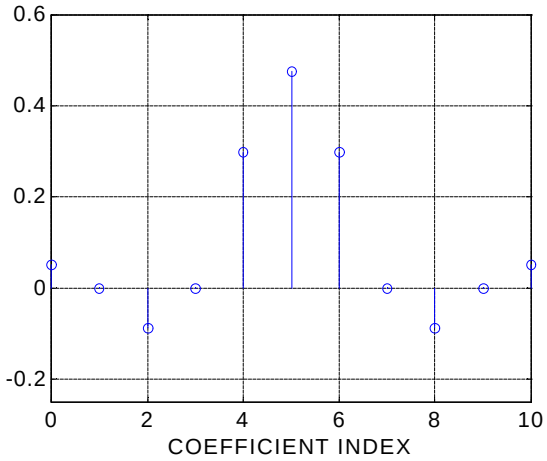

Figure 14: Half-band filter impulse response.

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

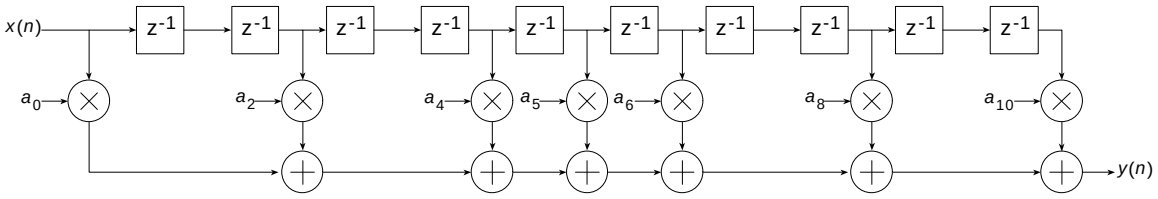

Figure 15: Half-band filter architecture.

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

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

5.3 Hilbert Transform

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

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

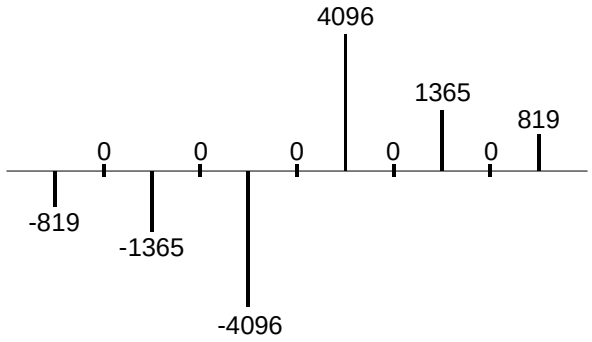

Figure 16: Impulse response of a Hilbert transformer.

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

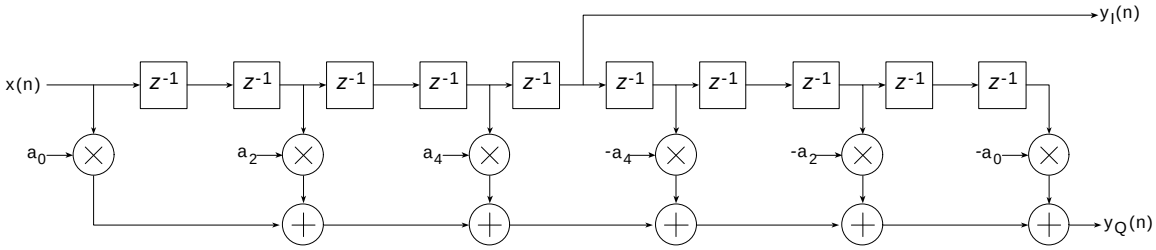

Figure 17: FIR filter realization of a Hilbert transformer.

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

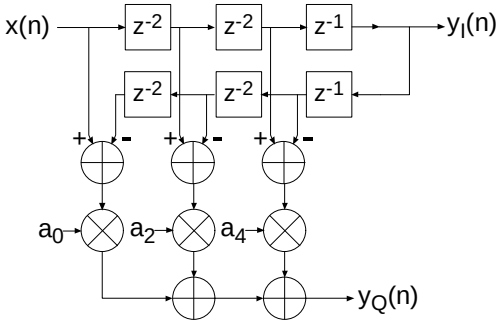

Figure 18: Hilbert transformer exploiting zero-valued filter coefficients and negative symmetry.

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

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

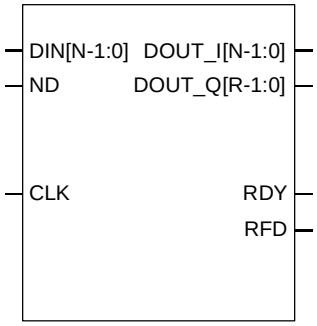

Figure 19: Hilbert transform symbol.

The Hilbert transform core has the same data-flow interface and control signals (*ND, RDY,RFD*) as the single-rate FIR filter core.

The Hilbert transform core also supports multiple channels as shown in Figure 20.

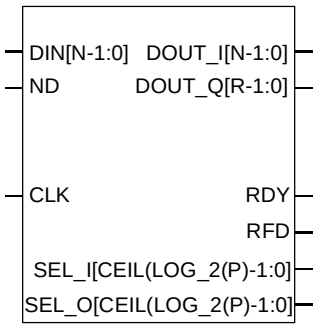

Figure 20: Multi-channel Hilbert transform core.

5.4 Interpolated FIR

An *interpolated FIR* (IFIR) [4] [6] has a similar architecture to a conventional FIR filter, but with the unit delay operator replaced by *k*-1 units of delay. *k* is referred to as the *zero-packing factor.* An *N*-tap IFIR filter is shown in Figure 21.

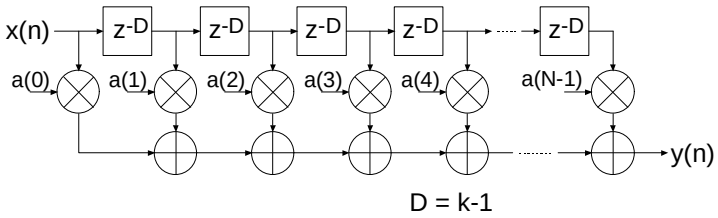

Figure 21:Interpolated FIR (IFIR). The zero-packing factor is *k.*

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

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

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

Table 1: FIR core signal pinout.

Signal Name	Direction	Description
CLK	Input	BIT CLOCK (active rising edge)
ND	Input	NEW DATA (active high) – When this signal is

		asserted the data sample presented on the *DIN* port is loaded into the PSC shown in Figure 2 and an inner-product computation is started. *ND* should not be asserted while *RFD* is low. Doing so will corrupt the calculation.
DIN[N-1:0]	Input	FILTER INPUT DATA SAMPLE – N-bit wide filter input sample.
RDY	Output	FILTER OUTPUT SAMPLE READY (active high) Indicates that a new filter output sample is available on the *DOUT* port.
RFD	Output	READY FOR DATA – (active high) Indicates when the final bit of the current data sample is about to be processed and new data may be supplied to the filter.
SEL_I[ceil(log_2(P))-1:0]	Output	INPUT CHANNEL SELECT This is a standard binary count generated by the core that indicates the current filter input channel number.
SEL_O[ceil(log_2(P))-1:0]	Output	OUTPUT CHANNEL SELECT This is a standard binary count generated by the core that indicates the current filter output channel number.
DOUT[R-1:0]	Output	FILTER OUTPUT SAMPLE R-bit wide output sample bus for the FIR, half-band and interpolated filters. R depends of the filter parameters (data precision, coefficient precision, number of taps and coefficient optimization selection) and is always supplied as a full-precision output port to avoid any potential for overflow.
DOUT_I[N-1:0]	Output	FILTER OUTPUT SAMPLE, Hilbert transform – In-phase (I) component. A Hilbert transform accepts real valued input data and produces a complex result. This port is the real or in-phase component of the result. Since this output port is simply an access point to the center of the filter memory buffer, it carries the same precision as the input sample data stream, i.e., N bits.
DOUT_Q[R-1:0]	Output	FILTER OUTPUT SAMPLE, Hilbert transform – quadrature (Q) component. A Hilbert transform accepts real valued input data and produces a complex result. This port is the imaginary or quadrature component of the result.

6 Interface and Control

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

The timing for a single channel, non-symmetric FIR with *B*-bit precision input samples and a registered output port is shown in Figure 22. The *ND* input signal is used for loading a new input sample into the filter. It is effectively used internally as a clock enable, and the actual sample load operation occurs on the rising of the clock (*CLK*). When the core is ready to accept a new input sample the *RFD* signal is asserted. When a new output sample is available *RDY* is asserted for a single clock period. When the registered output option is selected the output sample will remain valid between successive assertions of *RDY.*

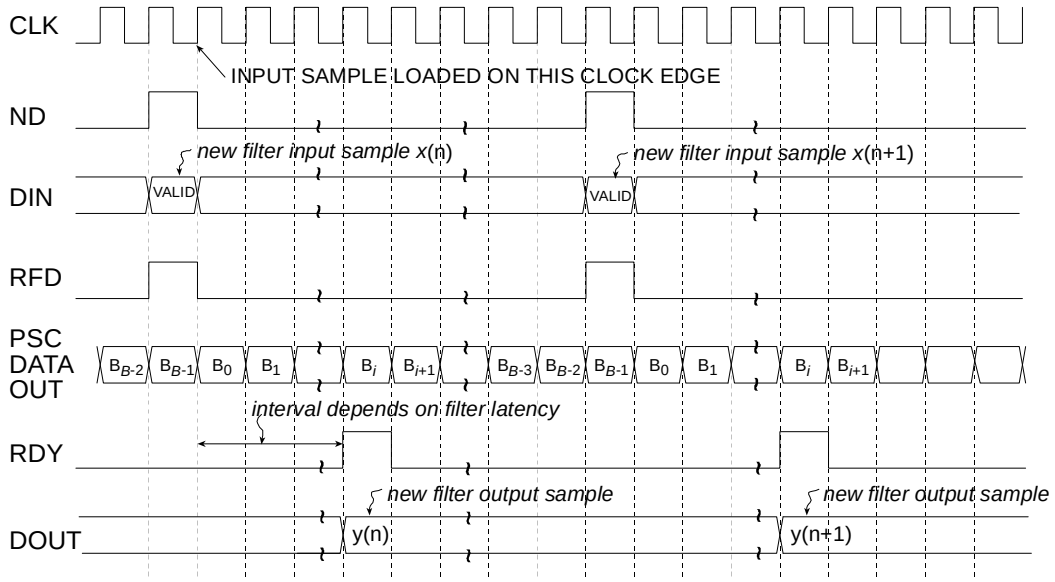

Figure 22: Single channel FIR filter timing. *B*-bit input samples, non-symmetric, registered output.

Figure 23 shows the timing for a single channel non-symmetric filter employing *B*-bit precision input samples, but in this example the output result is not registered. The input timing is the same as for the registered output example, but now the filter result is valid for only a single clock period and is framed by *RDY*.

For the two cases described so far, the host system is supplying input samples at the highest frequency possible, that is, every *B* clock ticks. This does not have to be the case, data samples can be supplied at a lower rate without disturbing the operation of the filter as shown in Figure 24. In this example, even though *B*-bit precision input samples are employed, new data is supplied to the filter every *B*+2 clock periods. Observe that *RFD* is still asserted on the *B*th clock cycle of a data sample epoch, but the host system only supplies a new input sample 2 clock cycles later. *RFD* remains active until the new input sample has been accepted by the filter core. This occurs synchronously with the positive going edge of the clock and with *ND* effectively acting as an active high clock enable.

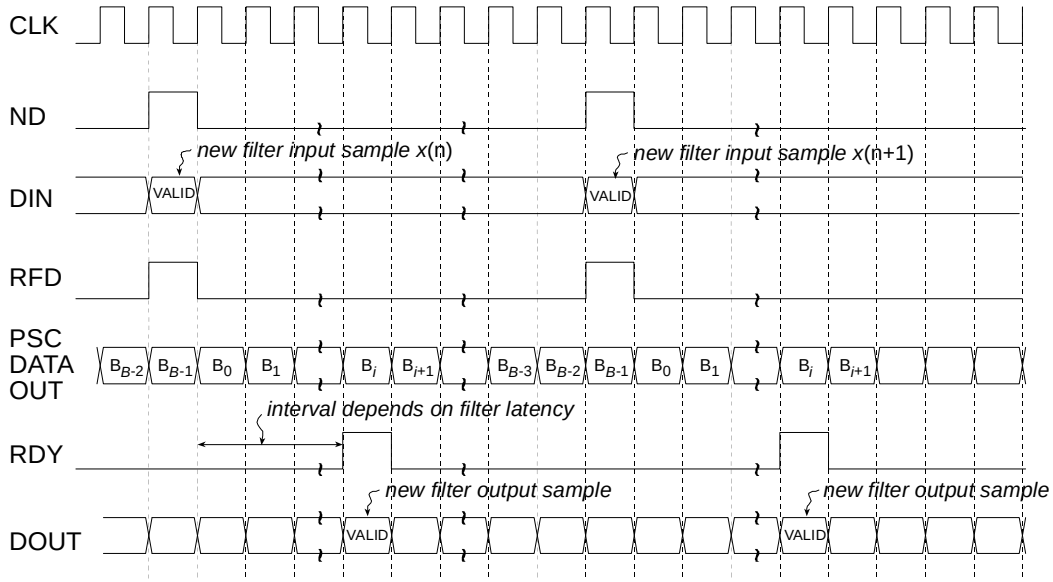

Figure 23: Single channel FIR filter timing. *B*-bit input samples, non-symmetrical impulse response, unregistered output.

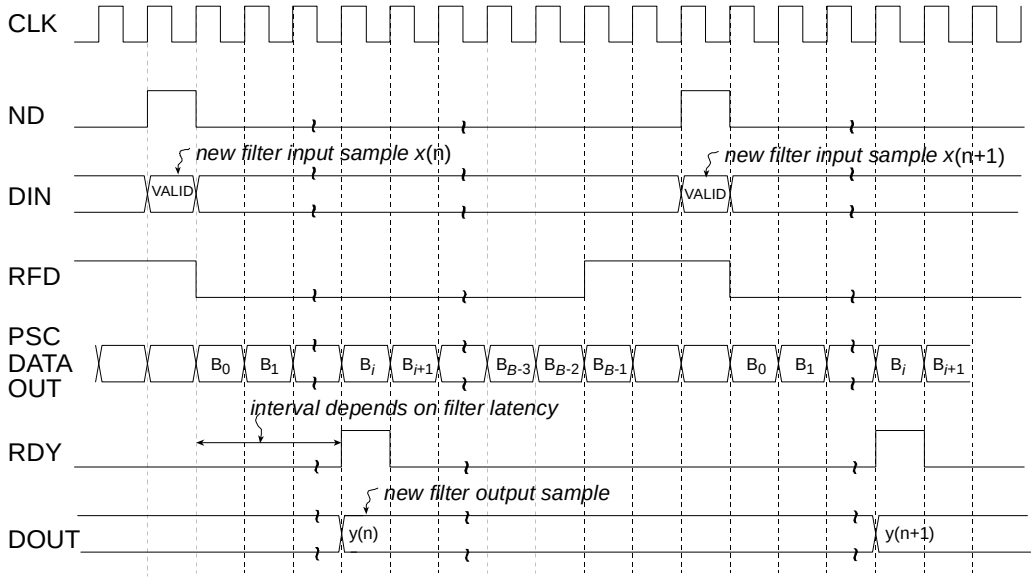

Figure 24: Single channel FIR filter timing. *B*-bit input samples, non-symmetric, registered output, input samples supplied every *B*+2 clock periods.

The timing for a symmetrical filter is slightly different to a non-symmetrical filter. In this case, for *B*-bit input samples, *B*+1 clock cycles are required to perform the computation. As a result, the maximum rate at which new samples can be written to the filter is one sample every *B*+1 clock periods. The timing of *RFD* of course reflects this additional clock period of processing. Figure 25 shows the timing for a symmetrical FIR filter.

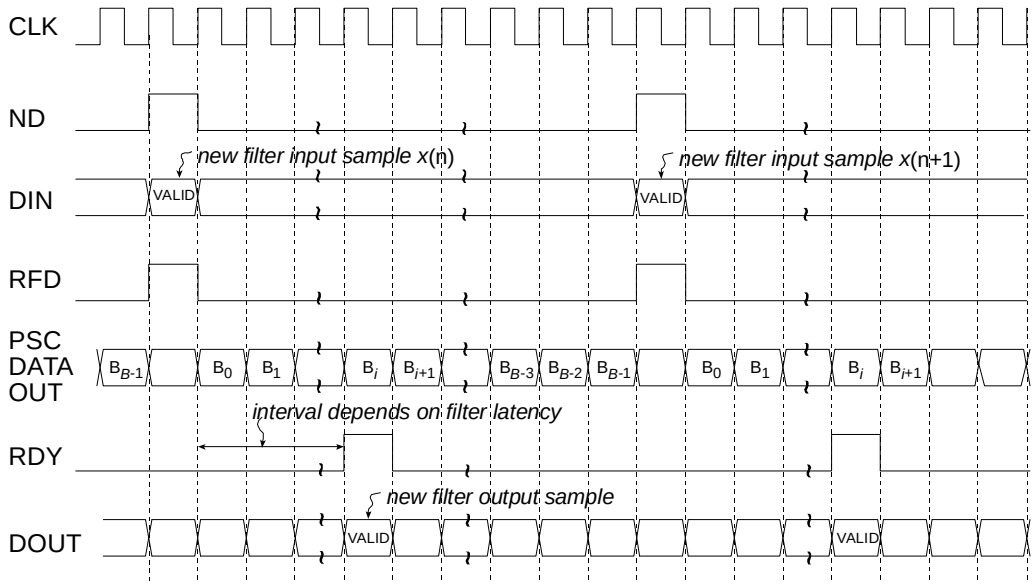

Figure 25: Single channel FIR filter timing. *B*-bit input samples, symmetrical impulse response, unregistered output. *B*+1 clock periods are required to process each new input sample.

As a specific example of the filter interface timing, consider a single-channel, non-symmetric FIR filter with 10-bit precision input samples. The timing diagram is shown in Figure 26. Ten clock cycles are needed to process each new input sample.

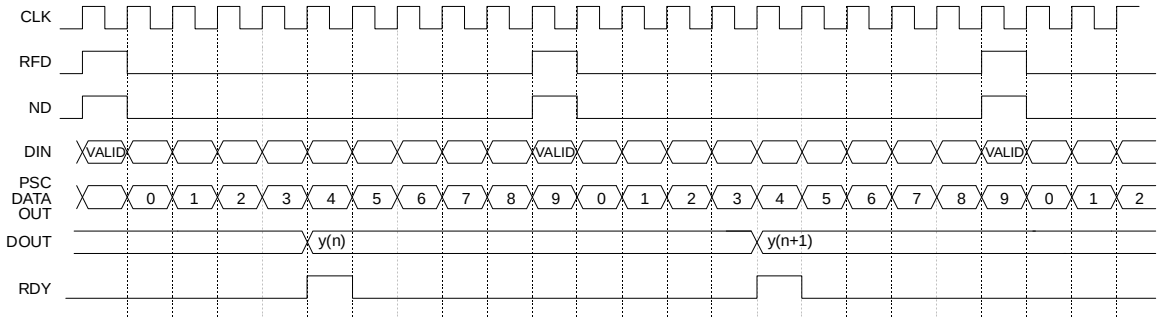

Figure 26: Single channel FIR filter timing. 10-bit input samples, non-symmetrical impulse response, registered output. 10 clock periods are required to process each new input sample.

Figure 27 shows the timing for a single channel symmetrical FIR employing 10-bit input samples. In this case, eleven clock cycles are required to process each new piece of data.

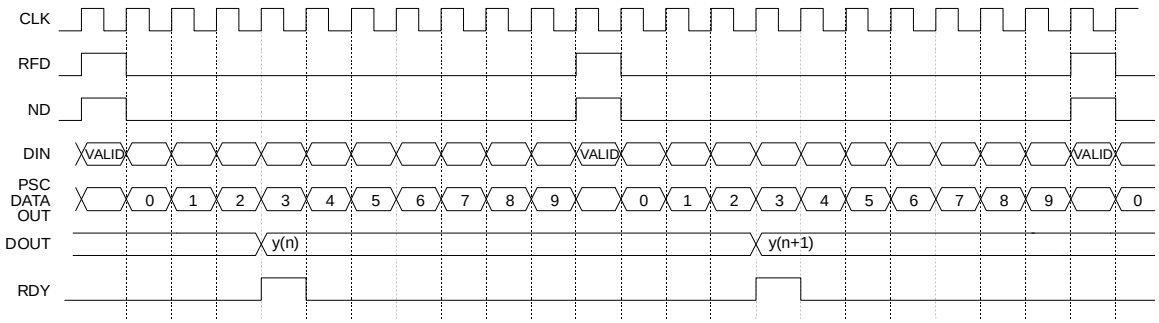

Figure 27: Single channel FIR filter timing. 10-bit input samples, symmetrical impulse response, registered output. 11 clock periods are required to process each new input sample.

Figure 28 and Figure 29 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 28 illustrates a filter with an unregistered output while Figure 29 shows the timing for registered output samples.

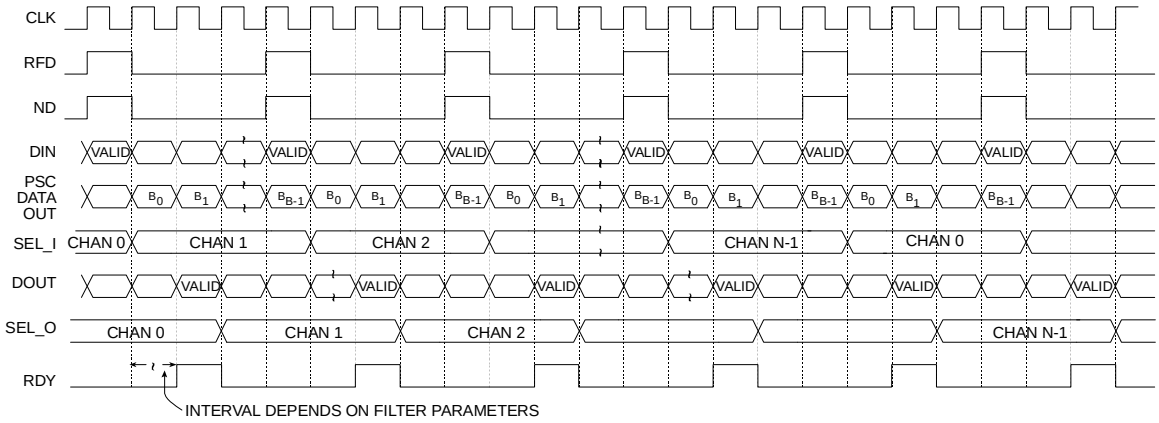

Figure 28: Multi-channel FIR filter timing. Non-symmetrical impulse response, *B*-bit input samples, unregistered output.

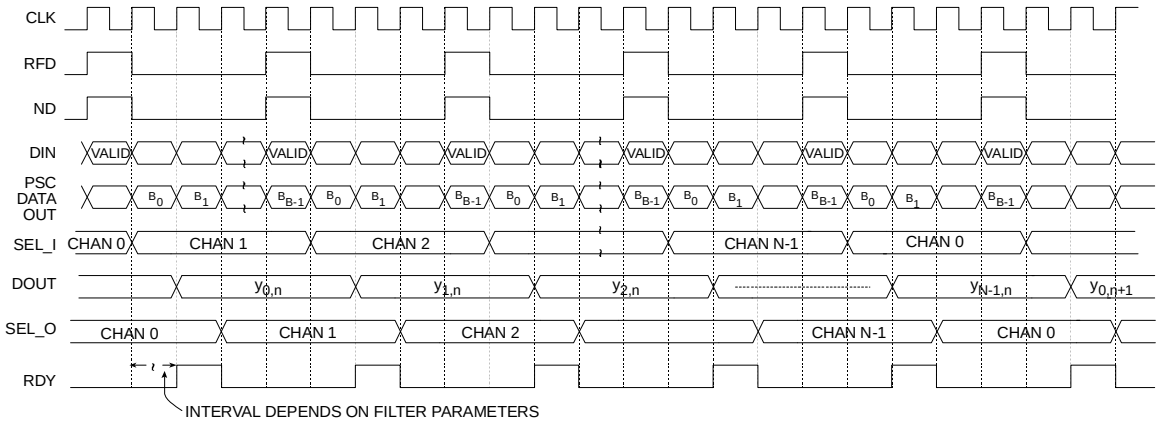

Figure 29: Multi-channel FIR filter timing. Non-symmetrical impulse response, *B*-bit input samples, registered output.

7 Filter Coefficient Data

The filter coefficients are supplied to the filter compiler using a coefficient file with a *.coe* extension. This is an ASCII text file with a single line header that defines the radix of the number representation used for the coefficient data, followed by the coefficient values themselves. This is shown in Figure 30 for an *N*-tap filter.

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

Figure 30: Filter coefficient file format.

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

The coefficient values may also be placed on a single line as shown in Figure 31.

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

Figure 31: Filter coefficient file format – coefficient data on a single line.

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

7.1 Fir

The coefficient file for the single-rate FIR filter is straightforward and consists of a one-line header followed by the filter coefficient data. For example, the filter coefficient file for an 8-tap filter using a base-10 representation for the coefficient values is shown in Figure 32.

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

Figure 32: 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 33 is presented in Figure 34.

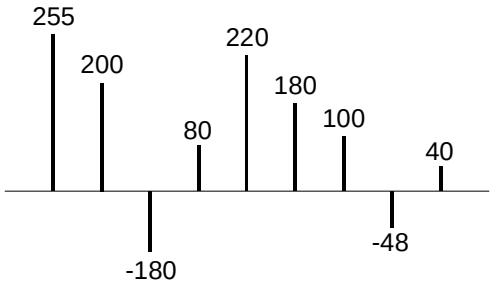

Figure 33: Non-symmetric impulse response.

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

Figure 34: Coefficient file for the non-symmetric impulse response in Figure 33.

The coefficient file for the negative-symmetric filter characterized by the impulse response in Figure 35 is shown in Figure 36.

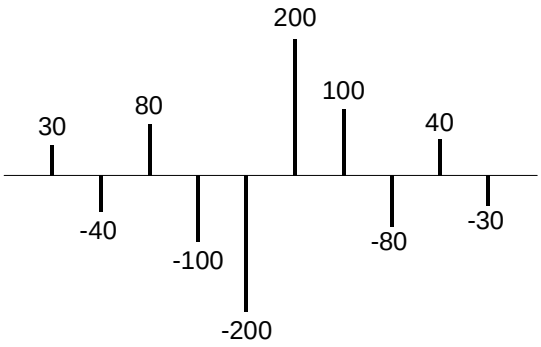

Figure 35: Symmetric impulse response.

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

Figure 36: Coefficient file for the symmetric impulse response in Figure 35.

7.2 Half-Band Filter

As described in a previous section, every second filter coefficient for a half-band filter with an odd number of terms will be zero. When specifying the filter coefficient data for this filter class, the zero value entries need to be included in the coefficient file. For example, the filter coefficient file that specifies the filter impulse response in Figure 37 is shown in Figure 38.

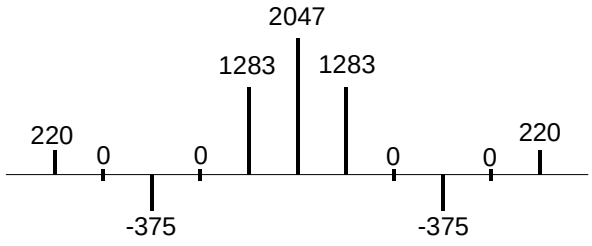

Figure 37: 11-tap half-band filter impulse response.

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

Figure 38: Coefficient file for the half-band filter impulse response shown in Figure 37.

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

Technically, the zero-valued entries for a half-band filter can occur at the filter impulse response extremities as shown in Figure 39. However, observe that these values do not contribute to the result.

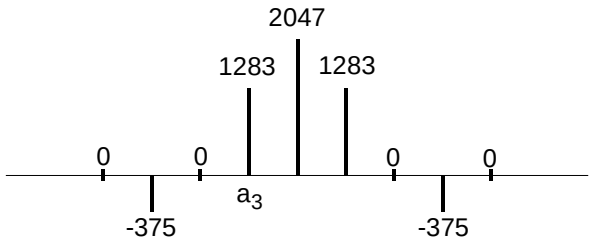

Figure 39: 9-tap half-band filter impulse response.

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

7.3 Hilbert Transform

The impulse response for a 10-term approximation to a Hilbert transformer is shown in Figure 40. 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 40 is shown in Figure 41.

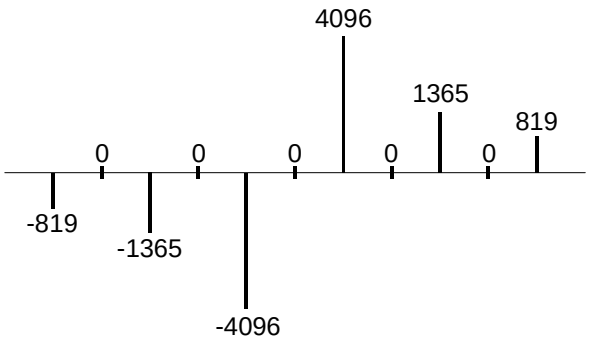

Figure 40: Hilbert transform – impulse response.

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

Figure 41: Coefficient file for the Hilbert transformer with the impulse response shown in Figure 40.

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

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

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

7.4 Interpolated Filter

In a previous section it was explained that an IFIR filter is similar to a conventional FIR, but with the unit delay operator replaced by *k*-1 units of delay. *k* is referred to as the *zero-packing factor.* One way to realize this substitution is by the insertion of *k*-1 zeros between the coefficient values of a prototype filter. When specifying an IFIR architecture, the full set of prototype coefficients are supplied in the coefficient file, without the zeros implied by the zero-packing factor. The zero-packing factor is defined through the filter user interface. For example, consider the filter coefficient data in the .coe file shown in Figure 42.

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

Figure 42: Prototype coefficient data for IFIR example.

If a zero-packing factor of *k*=2 is specified, the equivalent filter impulse response will be as shown in Figure 43.

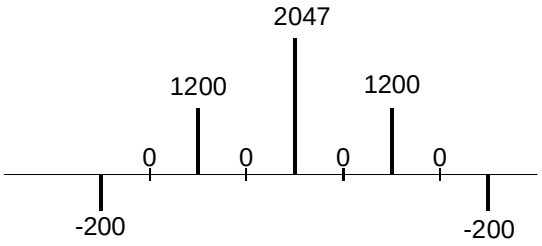

Figure 43: Equivalent IFIR impulse response for the coefficient data shown in Figure 42 with a zero-packing factor *k*=2.

If the zero-packing factor is changed to *k*=3, the impulse response will be as shown in Figure 44.

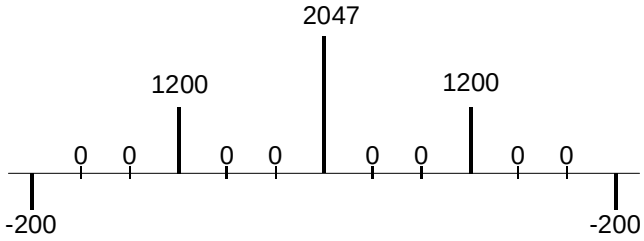

Figure 44: Equivalent IFIR impulse response for the coefficient data shown in Figure 42 with a zero-packing factor *k*=3.

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

8 CORE Generator Parameters

The CORE Generator parameterization screen for the filter is shown in Figure 45.

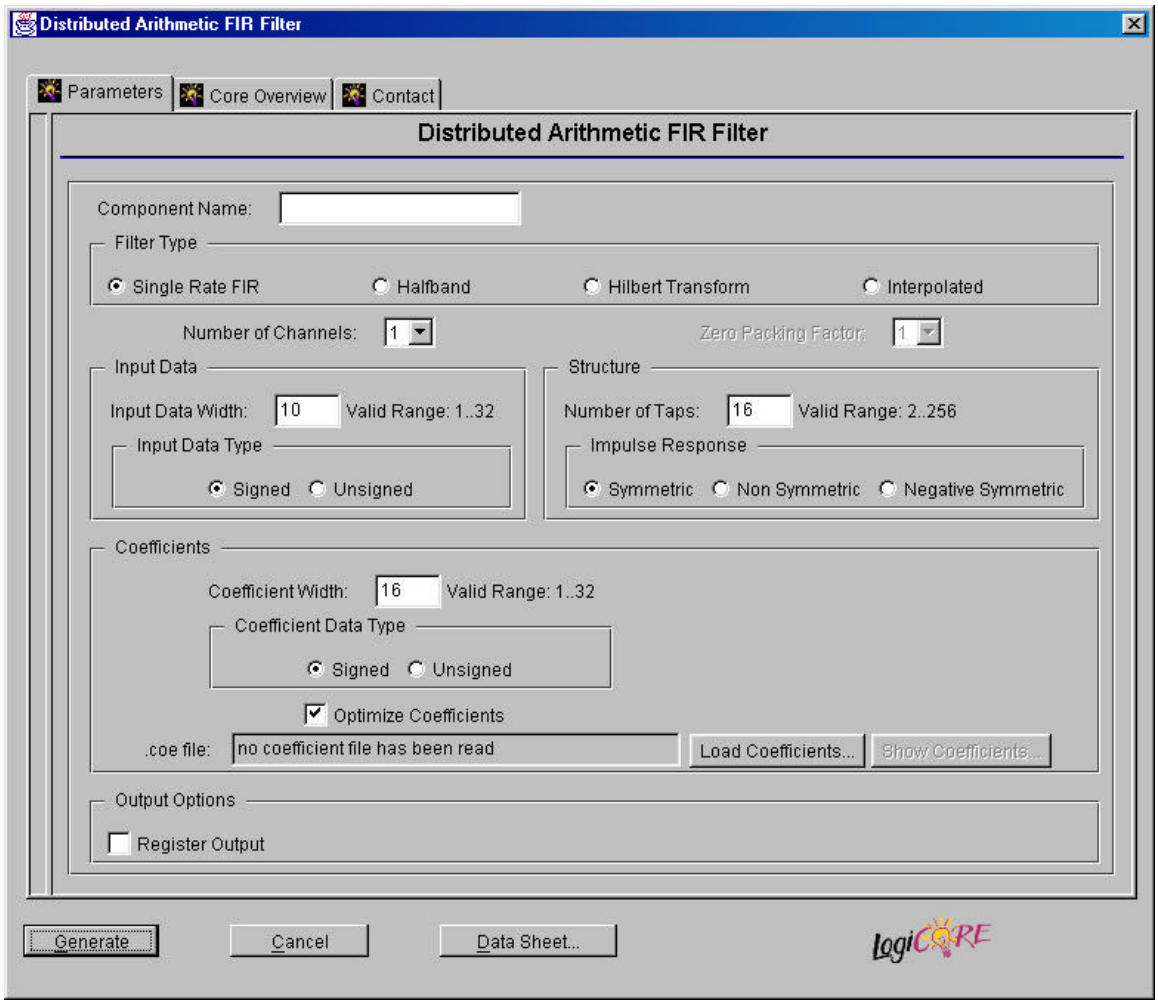

Figure 45: Parameterization screen.

The user supplied parameters are:

- **Component Name:** The user defined filter component name.
- **Filter Type:** Four filter types are supported 1. Single rate FIR, 2. Half-band FIR, 3. Hilbert transform, and 4. Interpolated FIR
- **Number of Channels:** The number of channels processed by the filter. One to a maximum of 8 channels can be accommodated by a single filter core.
- **Zero Packing Factor:** This field is applicable to the *interpolated* filter only. The zero packing factor specifies the number of 0's inserted between the coefficient data supplied by the user in the .coe (filter coefficient file). This is an integer value between 2 and 8 inclusive. A zero packing factor of *k* will insert *k*-1 0's between the supplied coefficient values.
- **Input Data Width:** The precision (in bits) of the filter input data samples. The input sample precision is an integer value between 1 and 32 inclusive.
- **Input Data Type:** The filter input data can be specified as either signed or unsigned. The signed option employs conventional two's complement arithmetic.
- **Number of Taps:** The number of filter taps. For a symmetric impulse response (either even or odd symmetric) the number of filters taps is between 2 and 256 inclusive. For a non-symmetrical coefficient set the range is 2 to 128 inclusive.

- **Impulse Response:** Indicates structure present in the coefficient set. The user may specify a symmetric, negative (odd)-symmetric or non-symmetric impulse response.
- **Coefficient Width:** The bit precision of the coefficient data. This is an integer value between 1 and 32 inclusive.
- **Coefficient Data Type:** The coefficient data can be specified as either signed or unsigned. When the signed option is selected conventional two's complement representation is assumed.
- **Optimize Coefficients:** The look-up tables employed in the filter mechanization can be optimized to minimize the amount of FPGA logic fabric employed by the core. The optimization is data (filter coefficient set) dependent.
- **Load Coefficients:** The filter coefficients are supplied in a coefficient or *coe* file. This is an ASCII file with a ".coe" extension. The file format is described in detail above. Activating this button brings-up a browser window that lets the user select a coefficient file.
- **Show Coefficients:** Selecting this button on the GUI activates a dialog box that displays the filter coefficient data.
- **Output Options:** The filter output bus can be registered or unregistered. When the registered output option is selected, the filter output bus *DOUT* is maintained at the core output between successive assertions of *RDY*. In the unregistered mode the output sample is only valid when *RDY* is active. At other times the port will change on successive clock cycles.

9 Core Resource Utilization

The logic utilization for a filter is a function of the filter length, coefficient precision, coefficient symmetry and input data precision. Tables 2 to 5 provide logic resource requirements for a number of filter configurations.

Table 2: Virtex logic slice utilization for several filter FIR filter configurations. 10-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output.

Filter Length	**Symmetry**	**Input Sample Precision**				
		4-bit	**8-bit**	**12-bit**	**16-bit**	**32-bit**
4	Symmetrical	31	34	41	43	66
	Non-symmetrical	29	33	36	43	67
8	Symmetrical	36	38	44	49	72
	Non-symmetrical	45	50	53	60	82
32	Symmetrical	103	108	113	117	157
	Non-symmetrical	141	146	151	154	196
80	Symmetrical	247	251	255	261	332
	Non-symmetrical	363	369	373	376	454
128	Symmetrical	370	377	380	385	493
	Non-symmetrical	532	536	537	543	646
256	Symmetrical	731	747	740	749	940

Table 3: Virtex logic slice utilization for several filter FIR filter configurations. 12-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output.

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

Table 4: Virtex logic slice utilization for several half-band filter configurations. 14-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output.

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

Table 5: Virtex logic slice utilization for several Hilbert transformer configurations. 14-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output.

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

Table 6: Virtex logic slice utilization for several interpolated filter configurations. 16-bit filter coefficients. Filter coefficient optimization is off. Single channel. Signed input, signed coefficients, unregistered output. Zero packing factor is 4.

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

10 References

[1] Peled and B. Liu, “A New Hardware Realization of Digital Filters”, *IEEE Trans. on Acoust., Speech, Signal Processing*, vol. ASSP-22, pp. 456-462, Dec. 1974.

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

[3] Xilinx Inc., *Xilinx Product Guide,* Xilinx Inc., San Jose California, 1999.

[4] P.P. Vaidyanathan, *Multirate Systems and Filter Banks,* Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[5] M. E. Frerking, *Digital Signal Processing in Communication Systems,* Van Nostrand Reinhold, New York, 1994.

[6] C. H. Dick, “Implementing Area Optimized Narrow-Band FIR Filters Using Xilinx FPGAs”, *SPIE International Symposium on Voice, Video and Data Communications – Configurable*

Computing: Technology an Applications Stream, Boston, Massachusetts USA, pp. 227-238, Nov 1-6, 1998. Also available at: http://www.xilinx.com/products/logicore/coredocs.htm

