AN639: Inferring Stratix V DSP Blocks for FIR Filtering Applications

AN-639-2.0 Application Note

This application note describes how to craft your RTL code to control the Quartus® II software-inferred configuration of variable precision digital signal processing (DSP) blocks in finite impulse response (FIR) filtering applications on Altera® Stratix® V devices. The examples demonstrate DSP block inference for multiple FIR filter variations. In this application note, you learn how to write your HDL code to ensure the Quartus II Fitter utilizes the appropriate DSP block features for your FIR filter application.

Stratix V DSP Block Features

Different features of the Stratix V variable precision DSP blocks are useful for efficient configuration of different FIR filter variations. The DSP blocks can be programmed in different operational modes which use different combinations of these features. Use of the appropriate set of features minimizes resource utilization, reduces power consumption, and improves data throughput and performance for your DSP application.

The FIR filtering applications described in this document use the following features of the Stratix V variable precision DSP blocks:

- High performance, power-optimized, fully registered multiplication operations.
- Support for 18-bit and 27-bit word lengths.
- Built-in addition, subtraction, and 64-bit accumulation units that support efficient combination of multiplication results.
- Cascading 64-bit output bus to propagate output results from one block to another in a systolic array without external logic support.
- Hard pre-adders that support halving multiplier usage in symmetric filters in both 18-bit and 27-bit modes.
- Internal coefficient register banks for filter implementation.
- Output adder that is optionally incorporated in cascading output bus.

The DSP block scan-in registers can form a cascading 18-bit or 27-bit input bus. This DSP block feature is designed to be a latency- and resource-efficient solution for implementing the tap-delay line for single-channel single-rate FIR filters not optimized for symmetric coefficients. However, these scan-in registers are not available in the Quartus II software v10.1 release. Therefore, all the examples use a tap-delay line external to the Stratix V DSP blocks.

For more information about the Stratix V DSP blocks, refer to the *Variable Precision DSP Blocks in Stratix V Devices* chapter of the *Stratix V Device Handbook*.

Subscribe

FIR Filter Application Design Examples

The design files that accompany this application note contain HDL code for examples of the following different FIR filter variations:

1. Single-channel single-rate systolic FIR filter
2. Single-channel single-rate systolic FIR filter optimized for symmetric coefficients
3. Multi-channel single-rate systolic FIR filter
4. Multi-channel single-rate systolic FIR filter optimized for symmetric coefficients
5. Multi-channel interpolation systolic FIR filter
6. Multi-channel decimation systolic FIR filter
7. Multi-channel interpolation direct-form FIR filter
8. Multi-channel decimation direct-form FIR filter

The design files include Verilog HDL code to infer variations 1, 2, 3, 5, and 6, and VHDL code to infer variations 2, 4, 7, and 8.

Table 1 summarizes the FIR filter design examples and the DSP block features they demonstrate. In this document, the number of taps is the number of coefficients in the FIR filter equation before any optimization. All the applications are tested for the parameter values in Table 1.

Table 1. FIR Filter Design Example Features and Stratix V DSP Block Features

Example		Number of Taps	Clock Rate as Multiple of Input Sample Rate	Number of Channels	Input Data Width (Bits)	Coefficient Width (Bits)	Demonstrated DSP Block Features
Verilog HDL	1	128	1×	1	18	18	Sum of two 18×18 multipliers with systolic register, output adder chain.
	2	128	1×	1	17	18	Pre-adder, sum of two 18×18 multipliers with systolic register, coefficient storage, output adder chain.
	3	128	4×	12	18	18	Sum of two 18×18 multipliers with systolic register, output adder chain.
	5	128	16×	4	18	18	Sum of two 18×18 multipliers with systolic register, coefficient storage, output adder chain.
	6	128	16×	4	18	18	Sum of two 18×18 multipliers with systolic register, coefficient storage, output adder chain, output accumulator.
VHDL	2	64	1×	1	25	27	Pre-adder, 27×27 multiplier, coefficient storage, systolic register, output adder chain.
	4	128	4×	12	25	27	Pre-adder, 27×27 multiplier, coefficient storage, systolic register, output adder chain.
	7	128	16×	4	25	27	27×27 multiplier, coefficient storage, output adder chain.
	8	128	16×	4	25	27	27×27 multiplier, coefficient storage, output adder chain.

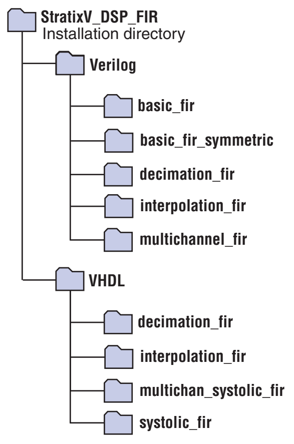
Each of the example applications demonstrates HDL code from which the Quartus II software infers the Stratix V DSP blocks and operational modes, parallel adders, and memory-based shift tap registers (using the altshift_taps megafunction in some cases). The HDL code does not instantiate the FIR compiler, the FIR II compiler, or any other Altera MegaCore function or megafunction explicitly.

Downloading the Example Applications

All of the files discussed in this application note are available for you to download from the Design Store.

After you unzip the file, your installation directory has the structure shown in Figure 1.

Figure 1. Design File Directory Structure



Example Application Structure

Each FIR filter application folder contains a Quartus II project and HDL files from which the Quartus II software can infer the target FIR filter structure. The project is a Quartus II software v10.1 project.

In addition, all the examples include input test data and ModelSim simulation scripts, the Verilog HDL examples include a **.do** file to display the waveform in ModelSim after simulation, and the VHDL interpolation and decimation direct form filter examples each include a MATLAB script and DSP Builder Advanced Blockset design for input and output signal visualization.

Example Application Usage

The HDL files are already included in the Quartus II project. When you compile the project in the Quartus II software v10.1, the Quartus II software infers your DSP block usage.

After you compile the Quartus II project, you can verify the inferred DSP block structure by any of the following methods:

- To confirm your module is mapped to Stratix V DSP blocks, examine the Fitter Resource Utilization by Entity report. The report should show no combinational ALUTs, ALMs, or dedicated logic registers in the implementation of the core DSP module.
- To confirm the operational mode of the Stratix V DSP blocks in which your design is configured, examine the Analysis & Synthesis DSP Block Usage Summary report. The report shows **Number Used** and number **Available per Block** for elements such as **Sum of two 18 × 18 with systolic register**, **DSP Block**, **DSP 18-bit Element**, **Signed Multiplier**, **Dedicated Pre-Adder**, **Dedicated Coefficient Storage**, **Dedicated Output Adder Chain**, and **Dedicated Output Accumulator**.
- To confirm both the mapping to DSP blocks and the operational mode of each DSP block, on the Quartus II Tools menu, point to **Netlist Viewers** and click **Technology Map Viewer**. Each DSP block is labeled <*block_instance_name*>**-mac**. If you double-click on a block, the viewer shows the configured components in that block.
- To confirm all details of the DSP block configuration, perform gate-level simulation and examine the netlist files. Altera recommends this approach only for advanced users.

Example Application Signals and Parameters

You can reuse the code from any of the HDL examples in your own design. All the FIR filter examples have the top-level signals shown in Table 2. The key parameters are shown in Table 3.

Table 2. FIR Filter Top Level Signals

Signal	Direction	Verilog		VHDL	
		Name	Width	Name	Width
Clock	In	`clk`	1	`clk`	1
Reset	In	`reset_n`	1	`reset`	1
Data in	In	`ast_sink_data`	`SINK_DATA_WIDTH`	`in_data`	`din_width_c`
Valid in	In	`ast_sink_valid`	1	`in_valid`	1
Channel in *(1)*	In	`ast_sink_chan`	`CHAN_WIDTH`	`in_chan`	`channel_width_c`
Data out	Out	`ast_source_data`	`SOURCE_DATA_WIDTH`	`out_data`	`dout_width_c`
Valid out	Out	`ast_source_valid`	1	`out_valid`	1
Channel out *(1)*	Out	`ast_source_chan`	`CHAN_WIDTH`	`out_chan`	`channel_width_c`

Note to Table 2:

(1) The channel in and channel out signals are available only for multi-channel FIR filters.

Table 3. Main FIR Filter Verilog HDL Parameters and VHDL Generics

HDL	Name	Description
Verilog HDL *(1)*	`NUM_TAPS`	Number of taps, defined as the number of coefficients in the basic equation.
	`INT`	Rate change factor for an interpolation FIR filter. Value is set to 4.
	`DEC`	Rate change factor for a decimation FIR filter. Value is set to 4.
	`SINK_DATA_WIDTH`	Input data width in bits. Only tested with value 17.
	`COEF_WIDTH`	Coefficient width in bits. Only tested with value 18.
	`SOURCE_DATA_WIDTH`	Output data width in bits. the value is derived from the input data width, coefficient data width, and NUM_TAPS.
VHDL	`engine_size_c`	Number of multipliers used in the FIR filter. The value depends on the number of taps and on the filter type. For interpolation and decimation filters, the value is the length of the polyphase subfilters. For single-rate symmetric filters, the value is one half the total number of basic equation summands
	`rate_c` *(2)*	Rate change factor for interpolation or decimation FIR filter.
	`din_width_c`	Input data width in bits. Only tested with value 25.
	`coeff_width_c`	Coefficient width in bits. Only tested with value 27.
	`num_chan_c`	Number of supported channels. Only available for multi-channel FIR filters.
	`dout_width_c`	Output data width in bits. The value is derived from the input data width, coefficient data width, and total number of basic equation coefficients.

Note to Table 3:

(1) In the Verilog examples, the number of channels cannot be modified without code changes.

(2) rate_c is a constant in the VHDL package rather than a VHDL generic.

FIR Filters Overview

The HDL code samples are intended to infer direct form and systolic FIR filters. The interpolation and decimation filter examples assume that polyphase decomposition has already been performed and that a set of coefficients is therefore available to be loaded in the DSP blocks' dedicated coefficient storage.

Direct Form and Systolic Filters

This section describes and contrasts direct form FIR filters and systolic FIR filters.

In the Quartus II software v10.1, both types of FIR filters require a tap-delay line external to the DSP blocks, as described in "Stratix V DSP Block Features" on page 1.

Direct Form FIR Filters

In a direct form implementation of a FIR filter, an adder tree sums the outputs of the individual multipliers. Each Stratix V DSP block implements two 18×18 multipliers or one 27×27 multiplier; a dedicated output adder in the DSP block performs the first level of addition. However, all of the other levels of the adder tree are implemented outside the DSP blocks, increasing resource utilization and lowering performance due to routing delay.

The number of cycles of latency is in the order of $\log_2$(<*number of taps*>), and the number of DSP blocks is <*number of taps*> for a FIR filter with 27-bit coefficients, or <*number of taps*>/2 for a FIR filter with 18-bit coefficients. Recall that the number of taps is the number of coefficients, not the physical number of taps in the tap-delay line.

Figure 2 shows a direct form FIR filter with 18-bit coefficients. In this implementation, chainout adders in the DSP blocks perform the first round of addition, halving the number of required adders outside the DSP blocks. Each blue block in the diagram is a distinct DSP block.

Figure 2. Direct Form FIR Filter with 18-Bit Coefficients

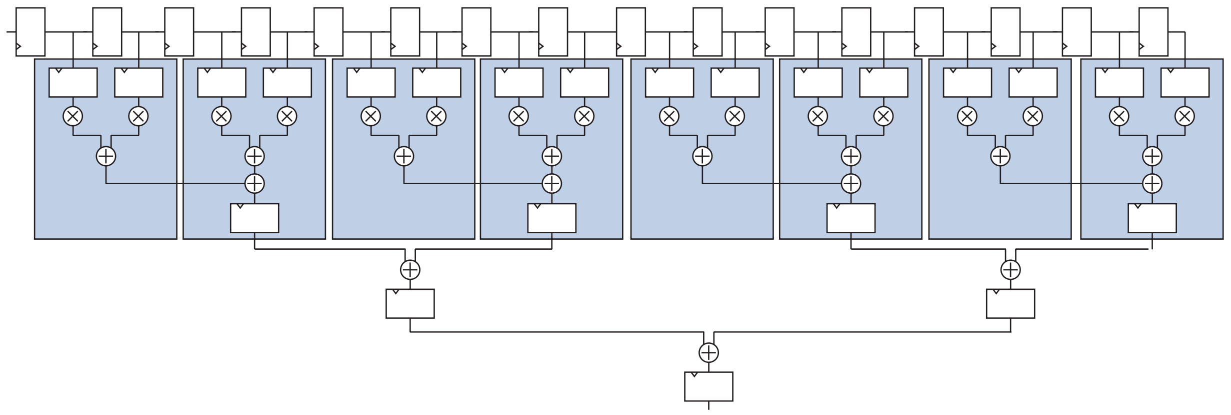
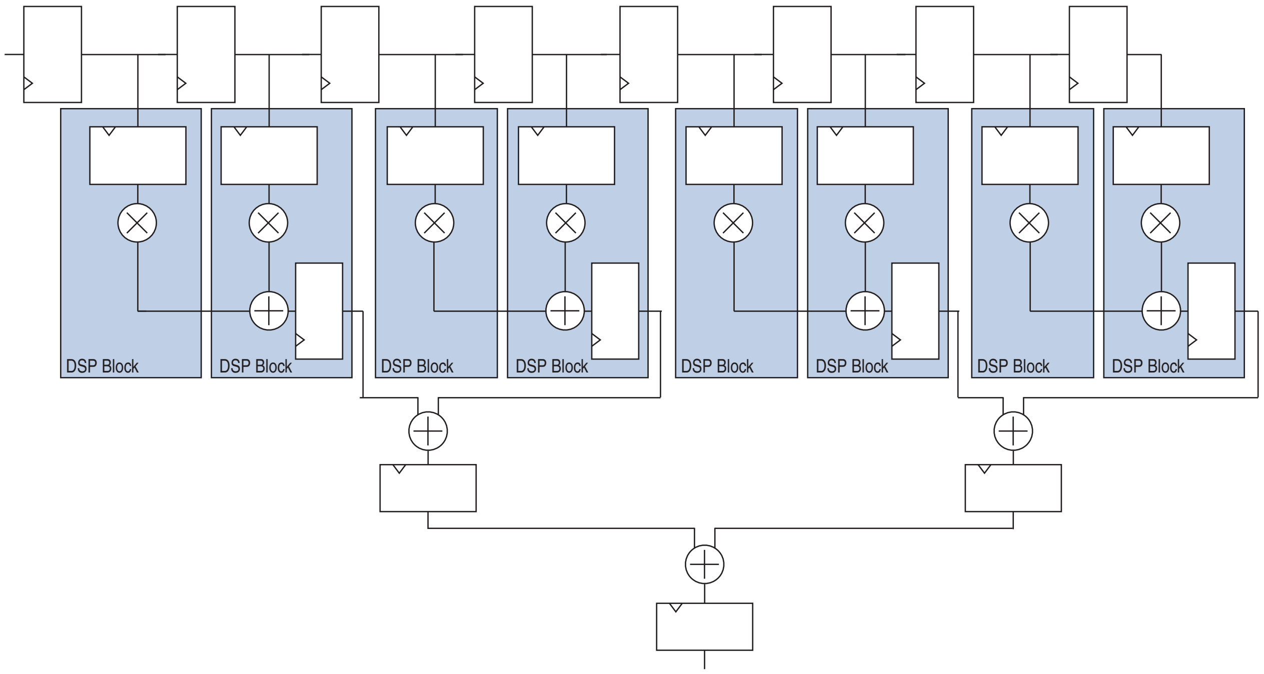


Figure 3 shows a direct form FIR filter with 27-bit coefficients. In this implementation, the chainout adders of the DSP blocks perform the first round of addition, halving the number of required adders outside the DSP blocks.

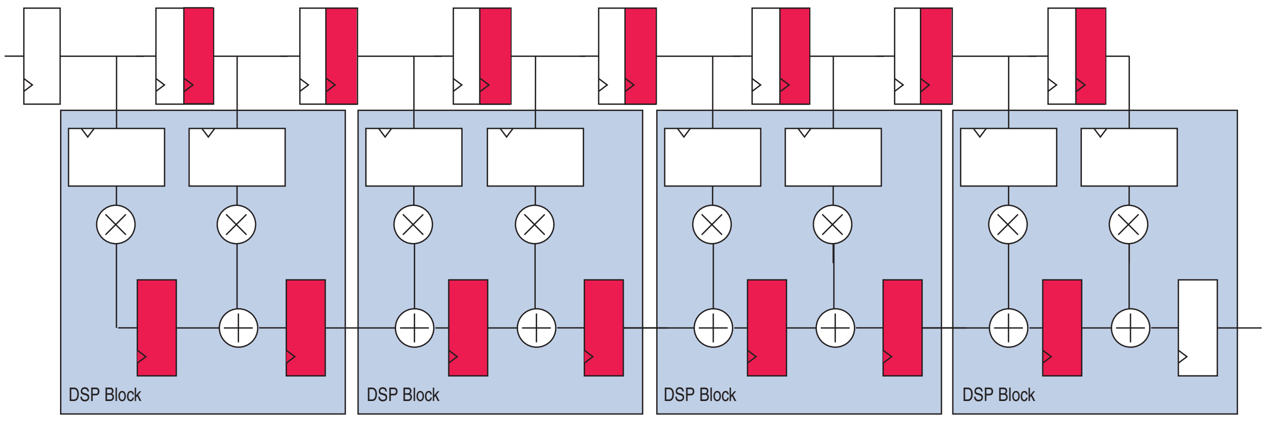
Figure 3. Direct Form FIR Filter with 27-Bit Coefficients



Systolic FIR Filters

In a systolic implementation of a FIR filter, the addition of the multiplier results is implemented in an output adder chain. More generally, a systolic FIR filter is an array of multiplier-adders in a pipeline structure. The registers at the outputs of all the multiplier-adders ensure the result incorporates all the input data correctly. Additional registers in the input tap-delay line synchronize the input values from the tap-delay line with the accumulating sum in the output adder chain. The synchronizing registers appear in red in Figure 4, which shows a single-rate systolic FIR filter with 18-bit coefficients.

Figure 4. Single-Rate 18-Bit Coefficient Systolic FIR Filter with Eight Taps



The number of cycles of latency through the systolic FIR filter is in the order of <*number of taps*>, and the number of DSP blocks is <*number of taps*> for a FIR filter with 27-bit coefficients, or <*number of taps*>/2 for a FIR filter with 18-bit coefficients. This type of filter uses fewer resources than the direct form FIR filter, because it does not require the adder tree. However, it has slightly longer latency.

Symmetric Systolic Filters

If you know that the coefficient sequence is symmetric, you can realize additional resource savings in your FIR filter implementation. You can halve the number of multipliers by adding the data before multiplying. If you have N coefficients, and coefficient h_0 and coefficient h_{N-1} have the same value, then you can replace $x_0h_0 + x_{N-1}h_{N-1}$ in the FIR filter computation with $h_0(x_0 + x_{N-1})$, and so on for $h_1,...,h_{\mathrm{floor}(N/2)}$, adjusting for odd N as required.

Figure 5 and Figure 6 show the target DSP block configuration for a symmetric systolic FIR filter.

Figure 5. 18×18 DSP Block with Pre-Adder and Chainout Adder for Systolic Array

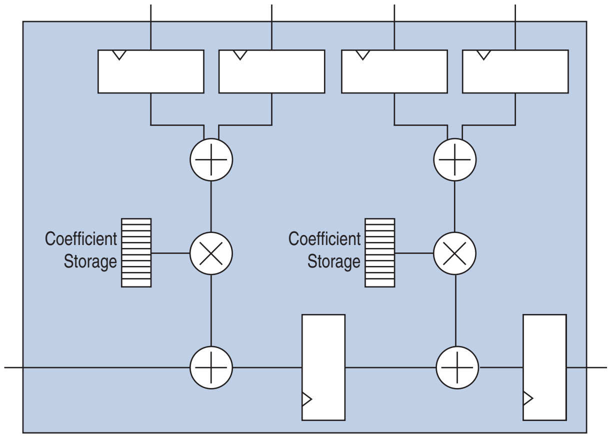


Figure 6. 27×27 DSP Block with Pre-Adder and Chainout Adder for Systolic Array

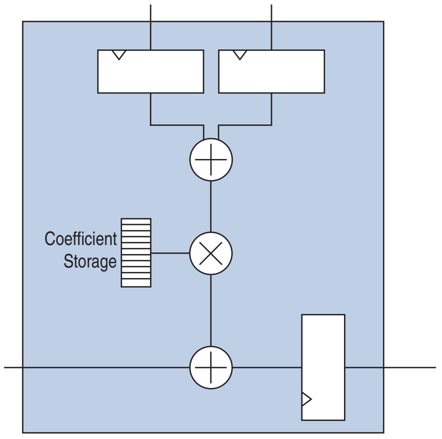
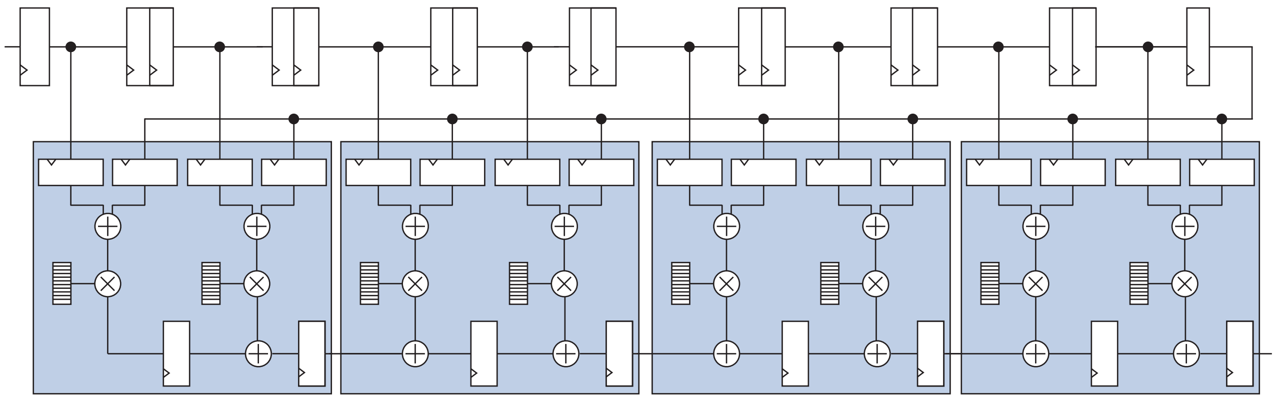


Figure 7 shows a possible implementation of a full symmetric single-channel systolic FIR filter with sixteen taps and 18-bit coefficients implemented in Stratix V DSP blocks. Figure 7 illustrates the structure of the Verilog HDL example single-channel single-rate systolic FIR filter optimized for symmetric coefficients. Other implementations of a single-channel single-rate symmetric systolic FIR filter are possible.

Figure 7. Sixteen-Tap Single-Channel Symmetric Systolic FIR Filter with 18-Bit Coefficients



In this structure, the latency is <*number of taps*>/2, and the number of DSP blocks is <*number of taps*>/4. In the equivalent structure for 27-bit coefficients, the latency is also <*number of taps*>/2, but the number of DSP blocks is <*number of taps*>/2. Recall that the number of taps is the number of coefficients, not the physical number of taps in the tap-delay line.

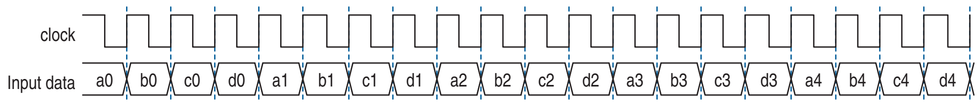
Multi-Channel Filters

Your DSP system may require two FIR filters, with the same or different coefficients. If your system requirements do not imply a high-speed solution, the design can use a single set of FIR filter hardware resources to implement both FIR filter applications.

A multi-channel FIR filter implements multiple FIR filter applications using the same hardware resources. The input data are interleaved, and the output data are interleaved. If you implement a two-channel FIR filter, you must clock the filter at twice the data rate of the incoming data on each channel; if you implement a FIR filter with Y channels, you must clock the filter at least at Y times the data rate of a single channel. This clocking scheme enables the filter to calculate the output data for each channel at the expected data rate.

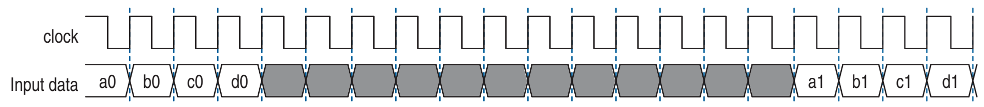
Figure 8 shows the input data format for all the multi-channel examples if the FIR filter operating clock rate is exactly the number of channels C multiplied by the input data sample rate on each channel, for C=4. This format is the required input data format for the Verilog HDL examples. Data sample a0 is the first data sample on channel a, and data sample d1 is the second data sample on channel d.

Figure 8. Input Data Format When FIR Filter Clock Rate is C=4 Times Channel Input Sample Rate



The VHDL examples also support a FIR filter clock rate greater than the number of channels C multiplied by the input data sample rate on each channel. For example, if a four-channel FIR filter runs at clock frequency 245.76 MHz and each input channel has sample rate 15.36 MHz, then each input channel provides one new valid data value every 16 clock cycles. The VHDL examples described in this application note assume the data format shown in Figure 9 at the input to a four-channel FIR filter with these example clock rates.

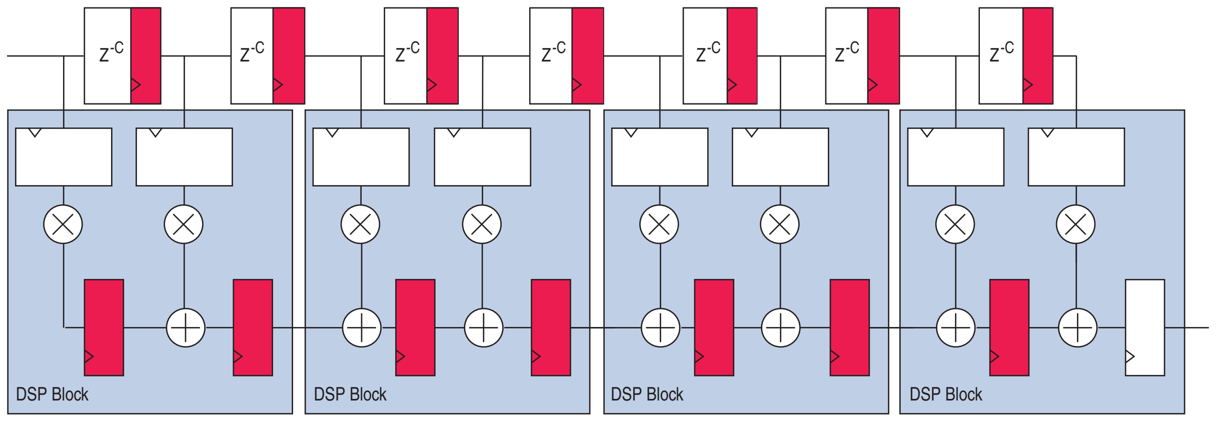
Figure 9. VHDL Example Input Data Format for Multi-Channel FIR Filters



The user tells the FIR filter to ignore the Don’t Care inputs by pulling the Valid in bit low during the Don’t Care cycles. The FIR filter tells the user to ignore the output from the Don’t Care input data by pulling the Valid out bit low when appropriate. Refer to Table 2 on page 5 for signal names for the Valid in and Valid out bits.

Figure 10 shows a multi-channel single-rate systolic FIR filter with eight taps and 18-bit coefficients. The number of channels C determines the number of delay elements in Z^{-C}.

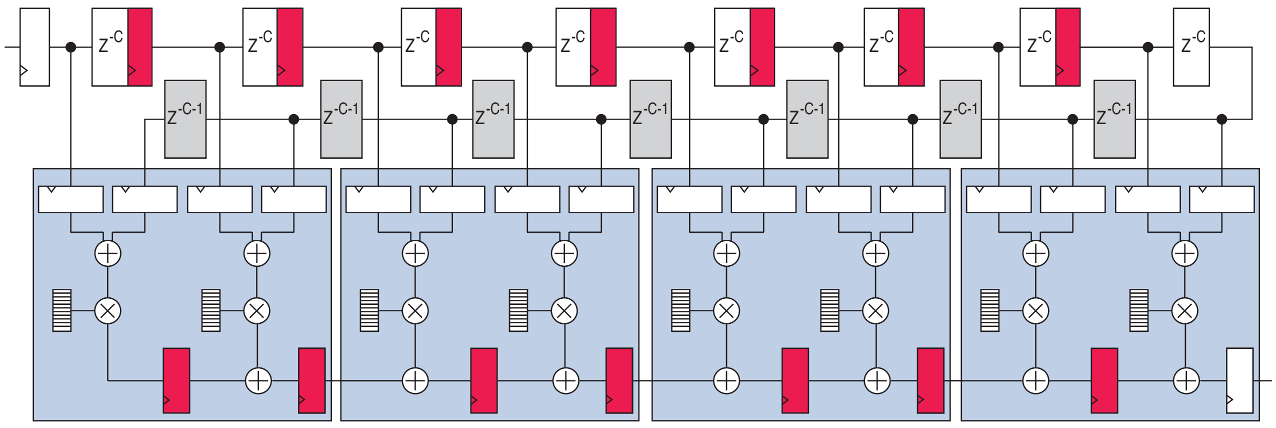
Figure 10. Eight-Tap Multi-Channel Systolic FIR Filter with C Channels and 18-Bit Coefficients



Symmetric Multi-Channel Filters

If you know that the coefficient sequence for each of your multiple input channels is symmetric, you can combine the advantages of the symmetric systolic FIR filter and the multi-channel FIR filter. Individual DSP blocks are configured as for a single-channel symmetric systolic filter, as shown in Figure 5 and Figure 6 on page 9. However, the tap-delay line reflects the multi-channel sampling with the addition of delays on both the forward and backward parts of the line. Figure 11 shows the resulting FIR filter structure.

Figure 11. Sixteen-Tap Multi-Channel Symmetric Systolic FIR Filter with 18-Bit Coefficients



C, the number of delay elements in Z^{-C}, must be greater than or equal to the number of channels. When C is 1, the structure is a single-channel FIR filter. Each Z^{-C-1} has (C–1) delay elements.

Interpolation Filters

An interpolation filter produces M×N output samples from N input samples for rate change factor M. An efficient implementation of such a filter processes the data as polyphase decomposed subsequences: the FIR filter output samples the output of M parallel subfilter paths. Figure 12 shows a high-level block diagram of polyphase decomposition for interpolation by M.

Figure 12. Polyphase Decomposition for Interpolation

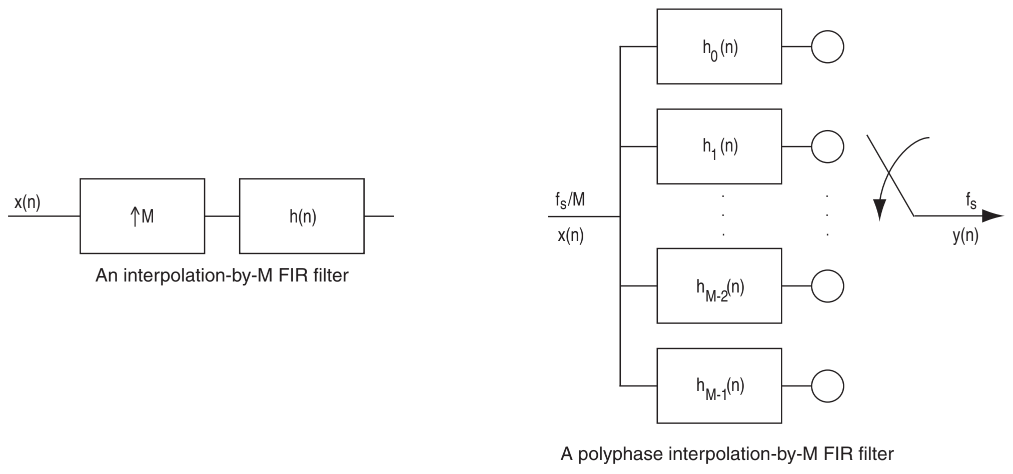

x(n)
↑M
h(n)
An interpolation-by-M FIR filter
f_S/M
x(n)
$h_0(n)$
$h_1(n)$
$h_{M-2}(n)$
$h_{M-1}(n)$
f_S
y(n)
A polyphase interpolation-by-M FIR filter

Figure 13 and Figure 14 show the effects of an interpolation FIR filter. Figure 13 shows sine waves before interpolation is performed, and Figure 14 shows the same sine waves after interpolation is performed.

Figure 13. Multi-Channel Input Data to Interpolation FIR Filter

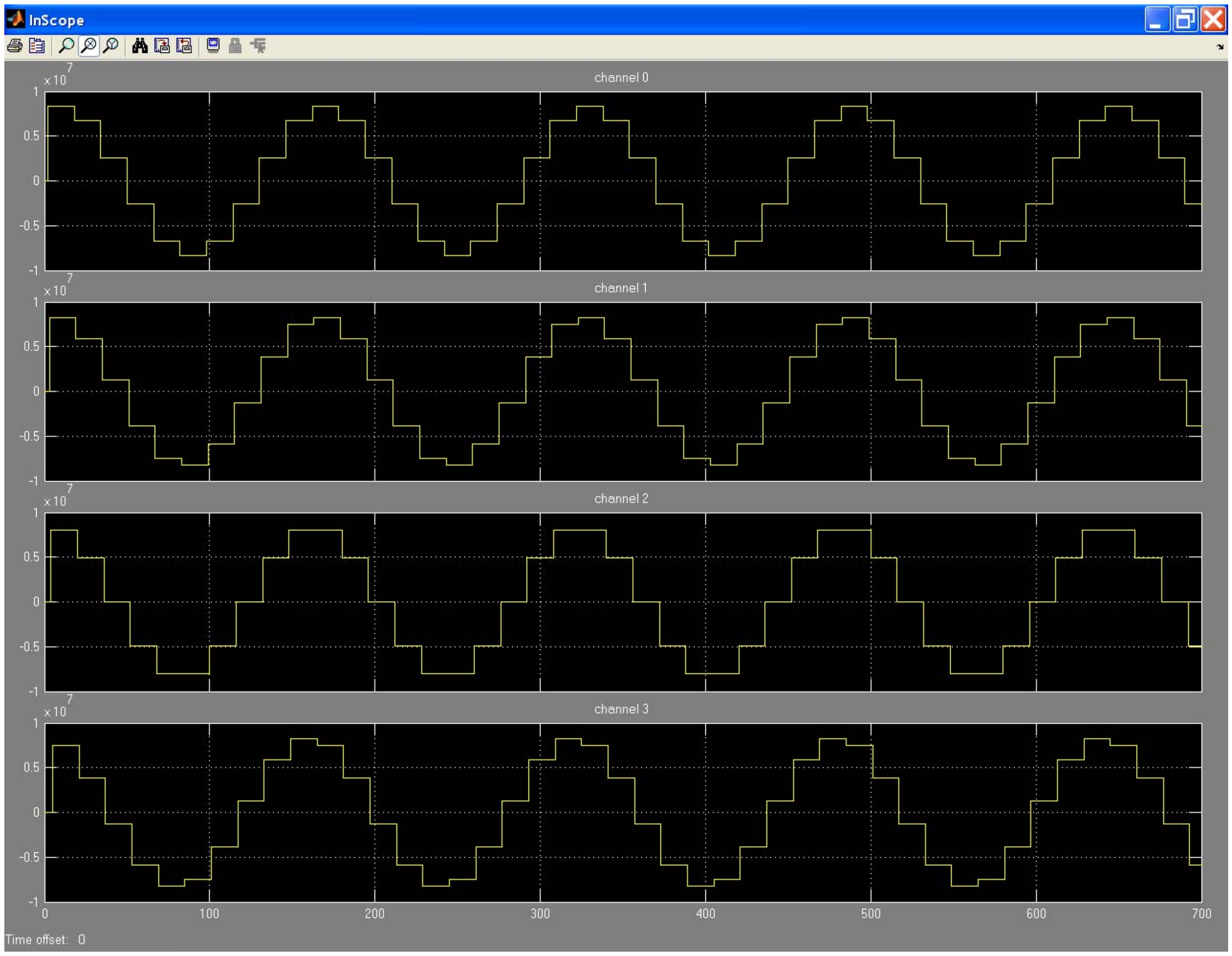
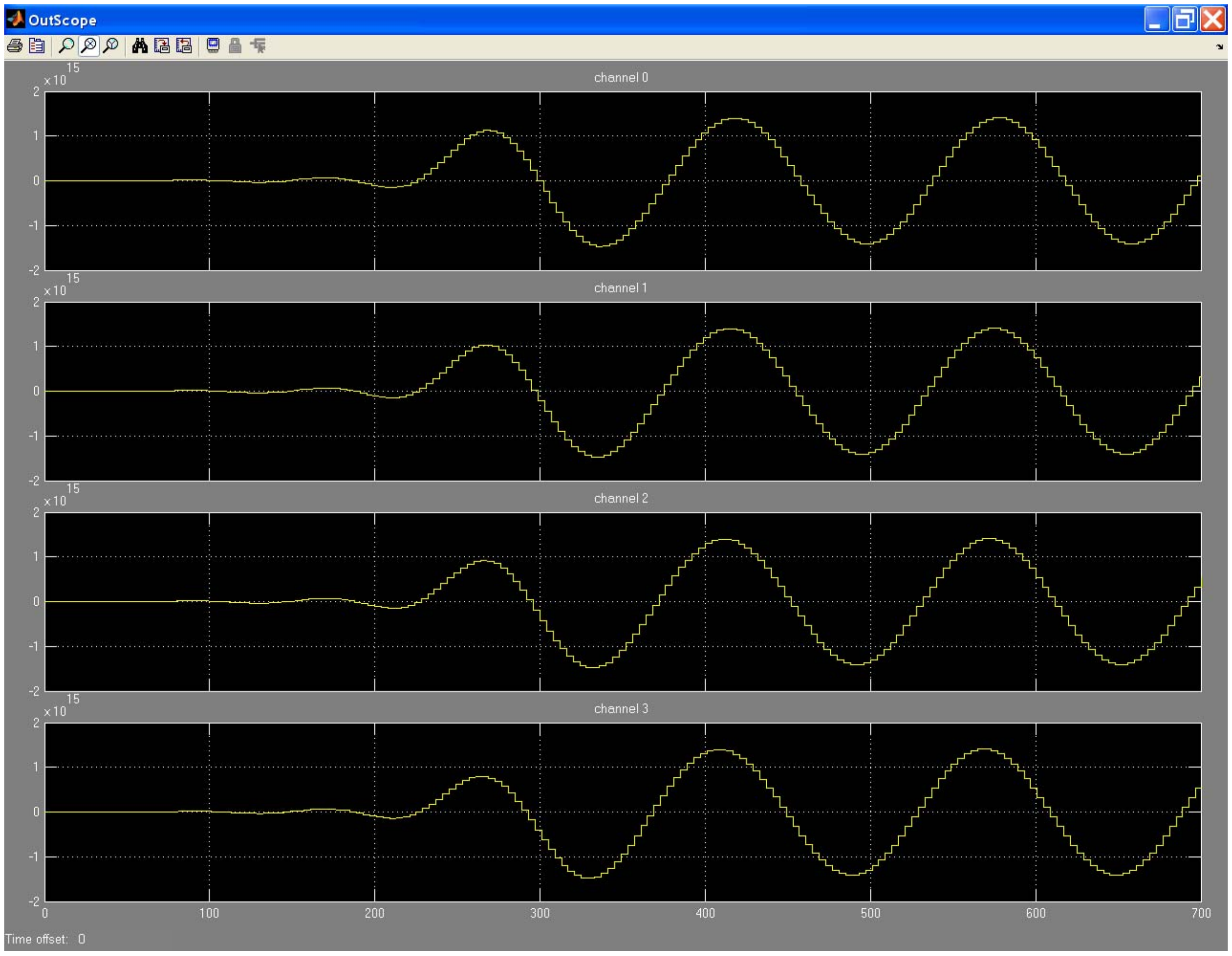
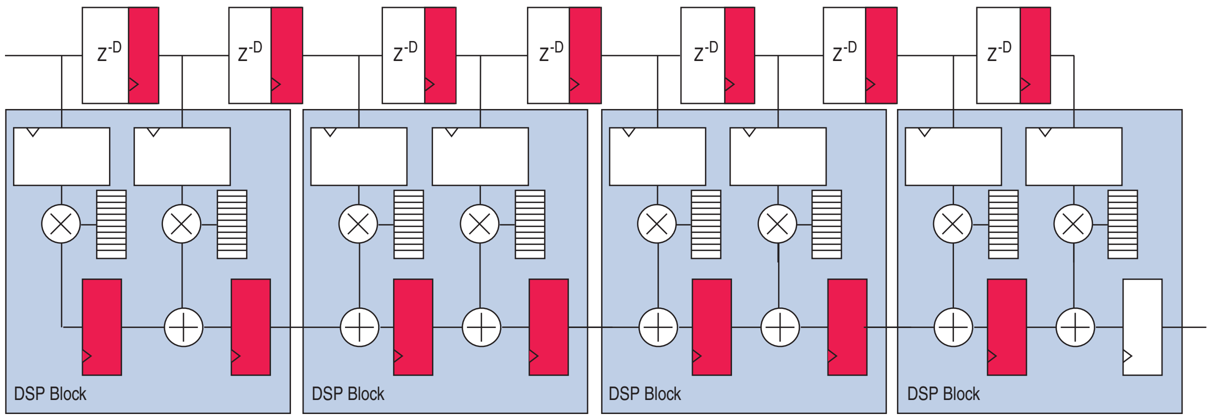


Figure 14. Output Data from Interpolation FIR Filter



The polyphase decomposition filter can be implemented in hardware as a multi-channel FIR filter that accesses a multiple coefficient set. Figure 15 shows an efficient implementation of an 8×M-tap, C-channel interpolation systolic FIR filter with rate change factor M in Stratix V DSP blocks. D, the number of delay elements in Z^{-D}, is the product of the number of channels C and the rate change factor M. The FIR filter runs at frequency C × M × <*input data rate*>. Therefore, the same data value is captured M times, and the coefficient value is switched M times for the same data value. Figure 15 shows the interpolation FIR filter structure that the Quartus II software infers from the Verilog HDL example code.

Figure 15. 8×M-Tap Multi-Channel Interpolation Systolic FIR Filter with 18-Bit Coefficients



This application note presents examples of HDL code that use two different data formats for multi-channel interpolation filters. Figure 16 and Figure 18 show the two different input data formats that the example four-channel interpolation FIR filters are designed to accept. In these examples, the number of channels is four, the rate change factor is four, and the operating clock rate is four times the input sample rate on each channel. Data sample a0 is the first data sample on channel a, data sample b0 is the first data sample on channel b, and so on.

The output data format from the design example interpolation FIR filters depends on the input data format. For the Verilog HDL example, Figure 16 shows the input data format and Figure 17 shows the output data format. In Figure 17, *xo<n>* are the output data points related to input channel *x*; *xo*1, *xo*2, and *xo*3 are the interpolation data points on channel *x*.

Figure 16. Four-Channel Interpolation FIR Filter Input Format I For Rate Change Factor 4

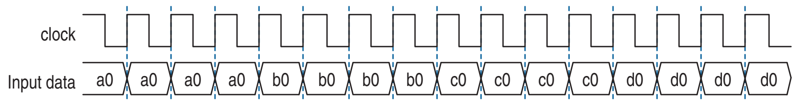
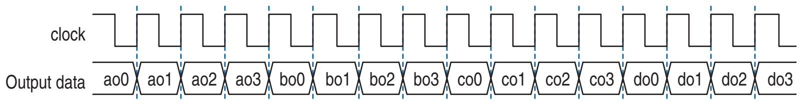


Figure 17. Four-Channel Interpolation FIR Filter Output For Rate Change Factor 4 and Input Data Format I



For the VHDL example, Figure 18 shows the input data format and Figure 19 shows the output data format. Output data points *xo<3n+i>* are computed from data point *x<n>* on input channel *x*, for i = 0 to M. In this case M=4.

Figure 18. Four-Channel Interpolation FIR Filter Input Format II For Rate Change Factor 4

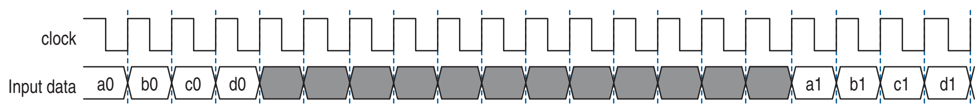
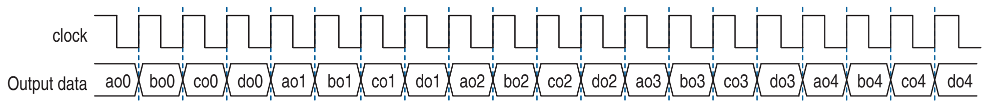
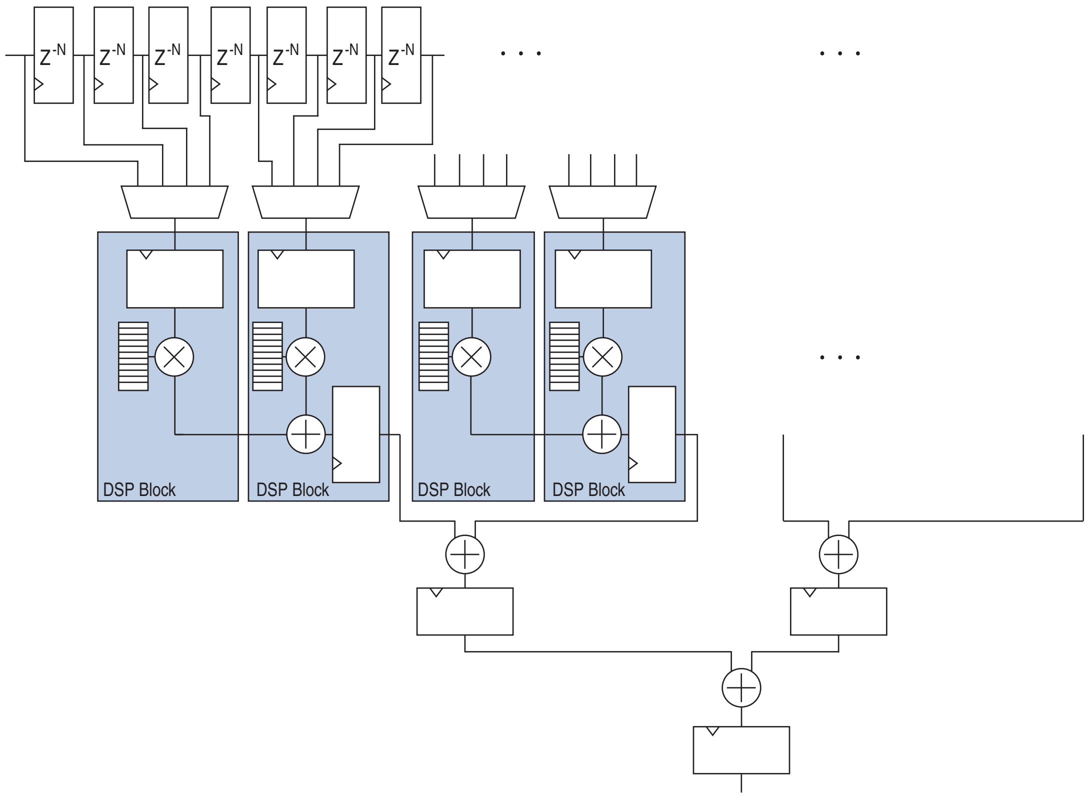


Figure 19. Four-Channel Interpolation FIR Filter Output For Rate Change Factor 4 and Input Data Format II



The VHDL example input format requires additional complexity in the hardware implementation outside the DSP blocks. Figure 20 shows the structure that the Quartus II software infers from the VHDL example code.

Figure 20. Multi-Channel Interpolation Direct Form FIR Filter with 27-Bit Coefficients and Rate Change Factor 4



In Figure 20, N, the number of delay elements in Z^{-N}, is defined by

N = `floor`(<*clk_rate*>/<*sample_rate*>/M where

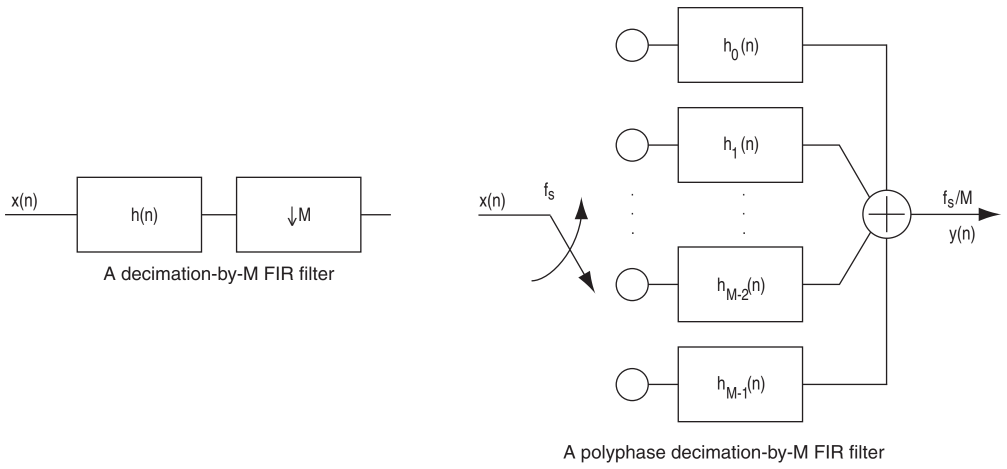
- <*clk_rate*> is the FIR filter operating clock frequency
- <*sample_rate*> is the input sample rate
- M is the rate change factor

If the FIR filter operating clock runs at exactly <*sample_rate*> times M, then N=1.

Decimation Filters

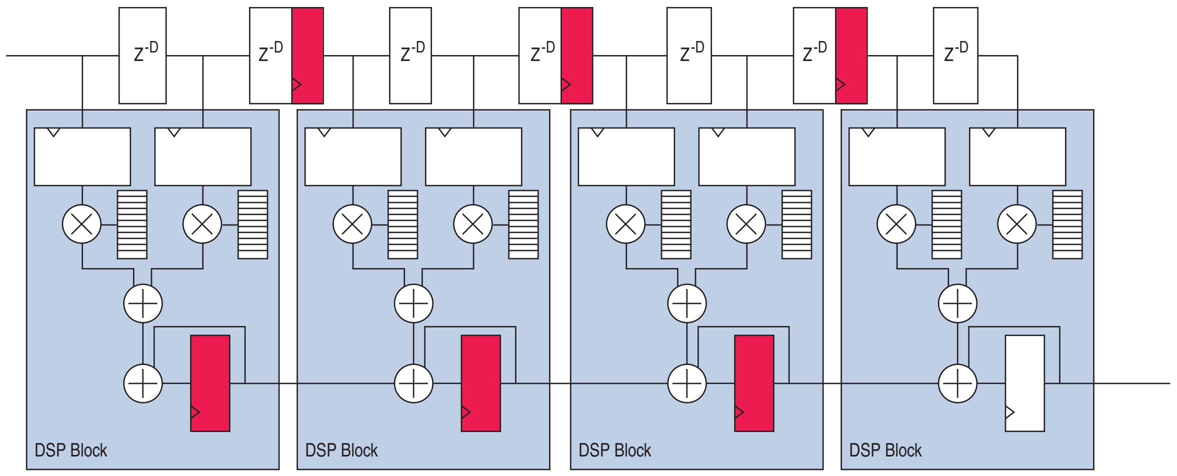
A decimation filter removes input samples from the output stream, leaving 1/M of the samples for rate change factor M. An efficient implementation of such a filter processes the data as polyphase decomposed subsequences: the FIR filter output samples the output of M parallel subfilter paths. Figure 21 shows a high-level block diagram of polyphase decomposition for decimation by M.

Figure 21. Polyphase Decomposition for Decimation



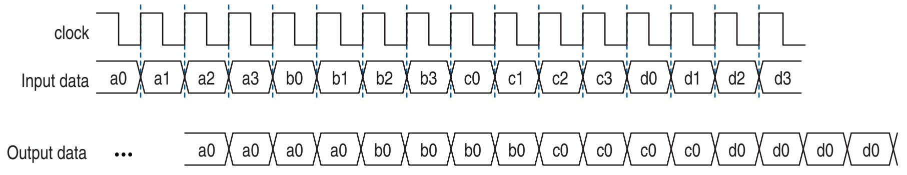
The polyphase decomposition filter can be implemented in hardware as a multi-channel FIR filter that accesses a multiple coefficient set. Figure 22 shows an efficient implementation of an 8×M-tap, C-channel decimation systolic FIR filter with rate change factor M in Stratix V DSP blocks. D, the number of delay elements in Z^{-D}, is the product of the number of channels C and the rate change factor M.

Figure 22. 8×M-Tap Multi-Channel Decimation Systolic FIR Filter with 18-Bit Coefficients



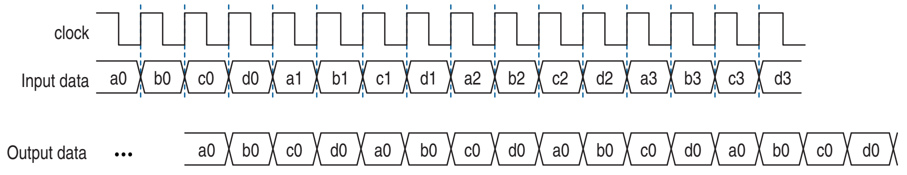
In the Verilog HDL example decimation FIR filter, the input data format is the format shown in Figure 17 on page 16 and the output data format is the format shown in Figure 16 on page 16. Figure 23 summarizes the input and output data formats for the Verilog HDL example decimation FIR filter.

Figure 23. Verilog HDL Example Decimation FIR Filter Input and Output Data Formats


clock
Input data
a0 a1 a2 a3 b0 b1 b2 b3 c0 c1 c2 c3 d0 d1 d2 d3
Output data
...
a0 a0 a0 a0 b0 b0 b0 b0 c0 c0 c0 c0 d0 d0 d0 d0

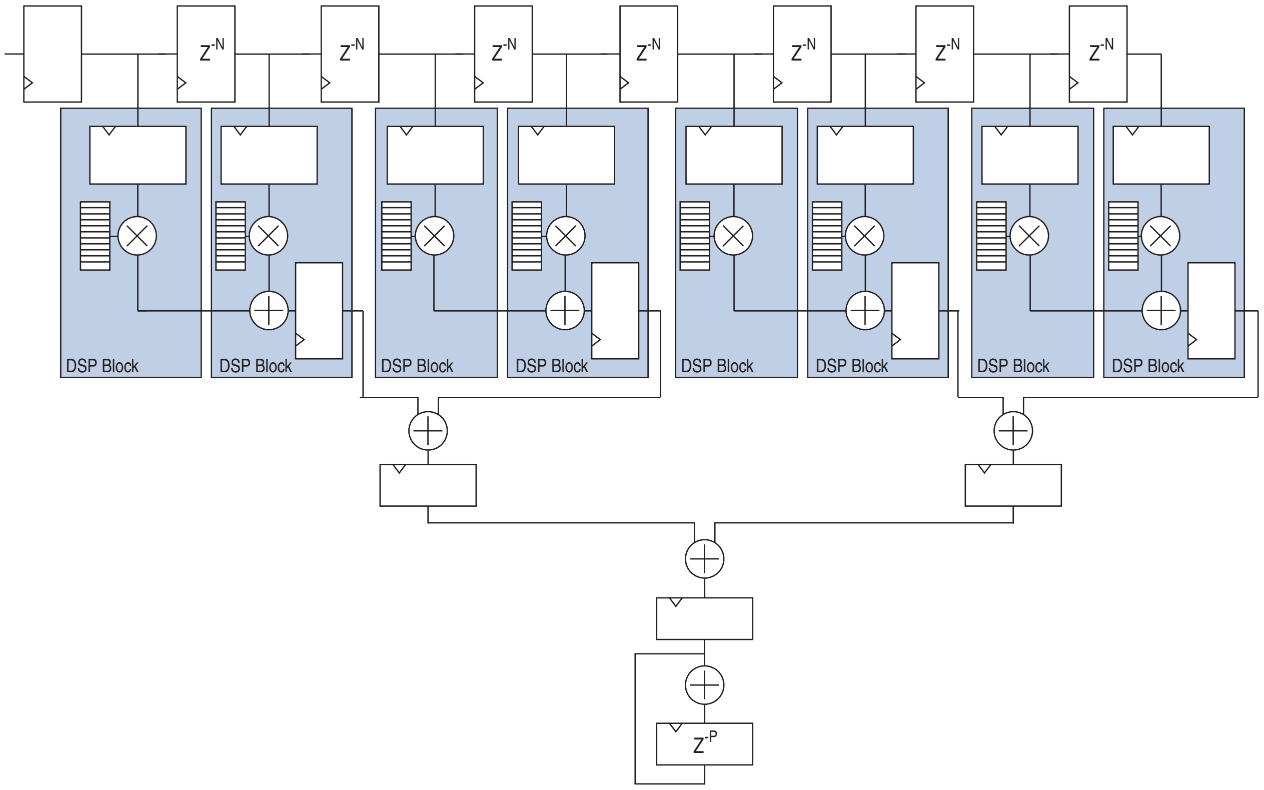
In the VHDL example decimation FIR filter, the input data format is the format shown in Figure 19 on page 17 and the output data format is the format shown in Figure 18 on page 17. Figure 24 summarizes the input and output data formats for the VHDL example decimation FIR filter.

Figure 24. VHDL Example Decimation FIR Filter Input and Output Data Formats


clock
Input data
a0 b0 c0 d0 a1 b1 c1 d1 a2 b2 c2 d2 a3 b3 c3 d3
Output data
...
a0 b0 c0 d0 a0 b0 c0 d0 a0 b0 c0 d0 a0 b0 c0 d0

Handling the data format for the VHDL example decimation FIR filter complicates the hardware implementation. Because the output data are interleaved, the accumulation function cannot be implemented in the DSP blocks. Figure 25 shows the structure that the Quartus II software infers from the VHDL example code. N is defined as for Figure 20 on page 17. P, the final accumulator delay, is the FIR filter operating clock frequency divided by the input sample rate.

Figure 25. Multi-Channel Decimation Direct Form FIR Filter with 27-Bit Coefficients



Tap-Delay Line

In the Quartus II software v10.1, all FIR filters require a tap-delay line external to the DSP blocks, as described in "Stratix V DSP Block Features" on page 1.

The altshift_taps megafunction can be inferred to implement a shift register chain with equally spaced taps in RAM. In the design examples, the VHDL code from which the Quartus II software infers the multi-channel single-rate, interpolation, and decimation filters include code to direct the software to infer an altshift_taps megafunction. For the multi-channel single rate systolic FIR filter that is optimized for symmetric coefficients, the forward, backward, and folding-point taps all have different depths. Therefore, the code instantiates three separate altshift_taps instances and concatenates them together. For the other examples, the tap-delay line is coded as sequences of delay registers.

For more information about the altshift_taps megafunction, refer to the *altshift_taps Megafunction User Guide*.

Verilog HDL Examples

The Verilog HDL examples demonstrate Verilog HDL code that guides the Quartus II Fitter to infer Stratix V DSP blocks with specific operational modes. All of the Verilog HDL examples require 18-bit by 18-bit multiplication. Coefficients and input data are 18-bits wide. Therefore, the Quartus II software can configure two multiplier paths in a DSP block.

Inferring a Single-Channel Single-Rate Systolic FIR Filter

The design example in the **Verilog/basic_fir/18x18/systolic_chainout_adder** directory contains Verilog HDL code from which the Quartus II software infers a 128-tap single-rate systolic FIR filter with 18-bit coefficients. The structure of this filter is shown in Figure 4 on page 8. During the mapping stage of compilation, when the Quartus II software identifies a multiplier-adder with register, it maps the function to the appropriate device atoms. If the design targets a Stratix V device with DSP blocks, the Quartus II software can implement this function in a DSP block instead of in custom logic, depending on the device resources already committed.

Example 1 shows the Verilog HDL code in the **dsp_block.v** file. The section that specifies the multiplier-adder with register is marked in red. Refer to Figure 4 on page 8 for the DSP block implementation that the Quartus II software infers from this Verilog HDL code. Each multiplier-adder plus register is mapped to an 18 × 18 DSP block element.

Example 1. Verilog HDL Code for Inferring 18×18 DSP Block with Chainout Adder for Systolic Array

```
module dsp_block (clk, aclr, ena, ay, by, chainin, chainout);

parameter coef_a0 = 0;
parameter coef_b0 = 0;
parameter DATA_WIDTH = 18;
parameter COEF_WIDTH = 18;
parameter CHAININ_WIDTH = 44;
parameter CHAINOUT_WIDTH = 44;

input clk;
input aclr;
input ena;
input signed [17:0]ay, by;
input signed [CHAININ_WIDTH-1:0]chainin;
output signed [CHAINOUT_WIDTH-1:0]chainout;

reg signed [CHAINOUT_WIDTH-1:0]  chainout;

reg signed [COEF_WIDTH-1:0] coefa, coefb;
initial begin
   coefa <= coef_a0;
   coefb <= coef_b0;
end

reg signed [DATA_WIDTH-1:0] ay_reg, by_reg;
reg signed [CHAINOUT_WIDTH-1:0] sa;
always @(posedge clk  or posedge aclr) begin
   if(aclr) begin
      ay_reg   <=  0;
      by_reg   <=  0;
      sa       <=  0;
      chainout <=  0;
   end
   else if(ena) begin
      ay_reg   <=  ay;
      by_reg   <=  by;
      sa       <=  chainin + ay_reg*coefa; // Form systolic structure
      chainout <=  sa + by_reg*coefb;        // from individual multiplier-adders
   end
end

endmodule
```

To enable inference of the cascaded output bus in the systolic structure, the Quartus II software v10.1 requires that the incoming and outgoing adder chain widths be specified for each DSP block. The System Verilog HDL file **single_fir.sv**, where the `dsp_block` module instances are generated, includes the HDL code to meet this requirement. Example 2 shows this HDL code.

Example 2. Specification of Incoming and Outgoing Adder Chain Widths for Each DSP Block

```
parameter SINK_DATA_WIDTH = 18;
parameter COEF_WIDTH = 18;
parameter NUM_TAPS = 128;
localparam NUM_DSP_Block = NUM_TAPS/2;

...

genvar n;
generate
    for (n=1; n < NUM_TAPS/2; n = n+1) begin : dsp_block

       dsp_block u (
               .clk(clk),
               .aclr(~reset_n),
               .ena(ast_sink_valid),
               .ay(sink_data_array[2*n]),
               .by(sink_data_array[2*n+1]),
               .chainin(chainout_array[n-1]),
               .chainout(chainout_array[n])
             );
        defparam u.DATA_WIDTH = 18;
        defparam u.coef_a0 = coef_array[2*n];
        defparam u.coef_b0 = coef_array[2*n+1];
        defparam u.CHAININ_WIDTH = (n<2  )? SINK_DATA_WIDTH+COEF_WIDTH+1:
                                   (n<3  )? SINK_DATA_WIDTH+COEF_WIDTH+2:
                                   (n<5  )? SINK_DATA_WIDTH+COEF_WIDTH+3:
                                   (n<9  )? SINK_DATA_WIDTH+COEF_WIDTH+4:
                                   (n<17 )? SINK_DATA_WIDTH+COEF_WIDTH+5:
                                   (n<33 )? SINK_DATA_WIDTH+COEF_WIDTH+6:
                                   (n<65 )? SINK_DATA_WIDTH+COEF_WIDTH+7:
                                   (n<129)? SINK_DATA_WIDTH+COEF_WIDTH+8:
                                   (n<257)? SINK_DATA_WIDTH+COEF_WIDTH+9:
                                            SINK_DATA_WIDTH+COEF_WIDTH+10;

        defparam u.CHAINOUT_WIDTH = (n<2  )? SINK_DATA_WIDTH+COEF_WIDTH+2:
                                    (n<4  )? SINK_DATA_WIDTH+COEF_WIDTH+3:
                                    (n<8  )? SINK_DATA_WIDTH+COEF_WIDTH+4:
                                    (n<16 )? SINK_DATA_WIDTH+COEF_WIDTH+5:
                                    (n<32 )? SINK_DATA_WIDTH+COEF_WIDTH+6:
                                    (n<64 )? SINK_DATA_WIDTH+COEF_WIDTH+7:
                                    (n<128)? SINK_DATA_WIDTH+COEF_WIDTH+8:
                                    (n<256)? SINK_DATA_WIDTH+COEF_WIDTH+9:
                                             SINK_DATA_WIDTH+COEF_WIDTH+10;
    end
endgenerate
```

The final outgoing adder chain width from the full systolic FIR filter and the outgoing adder chain width from the initial DSP block are also specified in the **single_fir.sv** file.

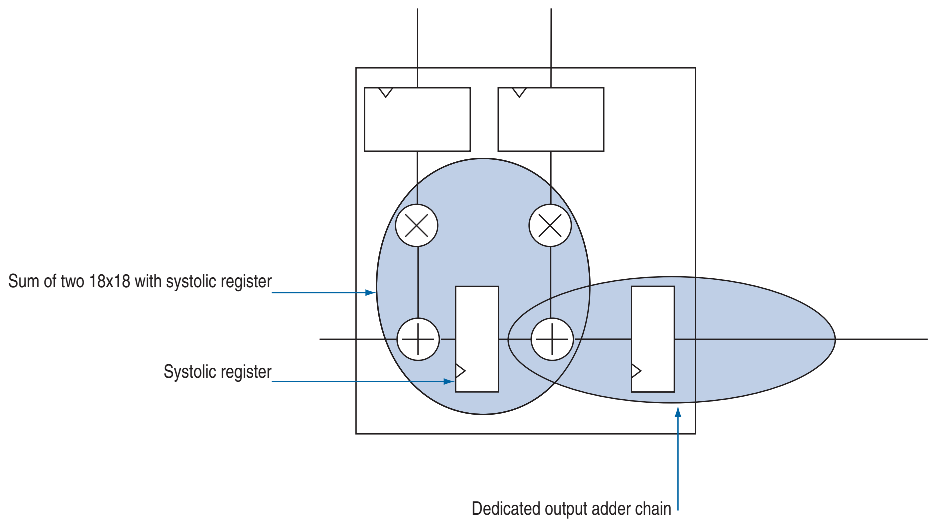
This Verilog HDL example generates a 128-tap single-channel single-rate systolic FIR filter with 18-bit coefficients that uses 64 DSP blocks.

After compilation, the Analysis & Synthesis DSP Block Usage Summary report shows the features of the DSP blocks that are configured for use. Table 4 summarizes the reported usage and Figure 26 shows the names of the relevant DSP block elements.

Table 4. DSP Block Usage for Verilog HDL Example Single-Rate Systolic FIR Filter

Statistic	Number Used	Available per Block
Sum of two 18×18 with systolic register	64	1.00
DSP Block	64	—
DSP 18-bit Element	128	2.00
Signed Multiplier	128	—
Dedicated Output Adder Chain	63	—

Figure 26. DSP Block Element Names


Sum of two 18x18 with systolic register
Systolic register
Dedicated output adder chain

Inferring a Single-Channel Single-Rate Symmetric Systolic FIR Filter

The design example in the **Verilog/basic_fir_symmetric/18x18/systolic_chainout_adder** directory contains Verilog HDL code from which the Quartus II software infers a 128-tap single-channel single-rate systolic FIR filter with 18-bit symmetric coefficients. The structure of this filter is shown in Figure 7 on page 10. During the mapping stage of compilation, when the Quartus II software identifies a two-input adder or a multiplier-adder with register, it maps the function to the appropriate device atoms. If the design targets a Stratix V device with DSP blocks, the Quartus II software can implement this function in a DSP block instead of in custom logic, depending on the device resources already committed.

Example 3 shows the Verilog HDL code from which the Quartus II software infers the structure shown in Figure 5 on page 9. You can find this HDL code in the Verilog HDL **Verilog/basic_fir_symmetric/18x18/systolic_chainout_adder/dsp_block.v** file. The sections that specify the two-input adder and the multiplier-adder with register are marked in red. Each two-input adder is mapped to a dedicated pre-adder and each multiplier-adder plus register is mapped to an 18 × 18 DSP block element.

Example 3. Verilog HDL Code for Inferring 18×18 DSP Blocks in Systolic Array with Pre-Adder and Chainout Adder

```
module dsp_block (clk, aclr, ena, ay, ax, by, bx, chainin, chainout);

parameter coef_a0 = 0;
parameter coef_b0 = 0;
parameter DATA_WIDTH = 17;
parameter COEF_WIDTH = 18;
parameter CHAININ_WIDTH = 44;
parameter CHAINOUT_WIDTH = 44;

input  clk;
input  aclr;
input  ena;
input  signed [DATA_WIDTH-1:0] ay, ax, by, bx;
input  signed [CHAININ_WIDTH-1:0] chainin;
output signed [CHAINOUT_WIDTH-1:0] chainout;

reg signed [CHAINOUT_WIDTH-1:0]  chainout;

reg signed [COEF_WIDTH-1:0] coefa, coefb;
initial begin
   coefa <= coef_a0;
   coefb <= coef_b0;
end

reg signed [DATA_WIDTH-1:0] ay_reg, ax_reg, by_reg, bx_reg;
wire signed [DATA_WIDTH:0] a_sum = ax_reg + ay_reg; // Two input addition
wire signed [DATA_WIDTH:0] b_sum = bx_reg + by_reg; // Two input addition
reg signed [CHAINOUT_WIDTH-1:0] sa;
always @(posedge clk  or posedge aclr) begin
   if(aclr) begin
      ay_reg   <=  0;
      ax_reg   <=  0;
      by_reg   <=  0;
      bx_reg   <=  0;
      sa        <=  0;
      chainout <=  0;
   end
   else if(ena) begin
      ay_reg   <=  ay;
      ax_reg   <=  ax;
      by_reg   <=  by;
      bx_reg   <=  bx;
      sa       <=  chainin + a_sum*coefa; // Multiplier-adder chain
      chainout <=  sa + b_sum*coefb;      // Multiplier-adder chain
//    sa       <=  chainin + (ay_reg+ax_reg)*coefa; // This code would lead the
//    chainout <=  sa + (by_reg+bx_reg)*coefb;      // Quartus II software to infer
                                                    // custom logic rather than
   end                                              // the desired DSP block structure,
end                                                 // because the bit width of the
                                                    // pre-adder output is not
                                                    // clear to the synthesis tool.
endmodule
```

The separation of the two-input addition code from the multiplier-adder chain code is necessary to enable the Quartus II software to infer the DSP block implementation. If the two functions are combined in the same line of code, as shown in blue in the commented code in Example 3, the Quartus II software instead infers custom logic, because it cannot deduce the pre-adder output width from the HDL code.

To enable inference of the cascaded output bus in the systolic structure, the Quartus II software v10.1 requires that the incoming and outgoing adder chain widths be specified for each DSP block. Refer to Example 2 on page 23 in "Inferring a Single-Channel Single-Rate Systolic FIR Filter" on page 21.

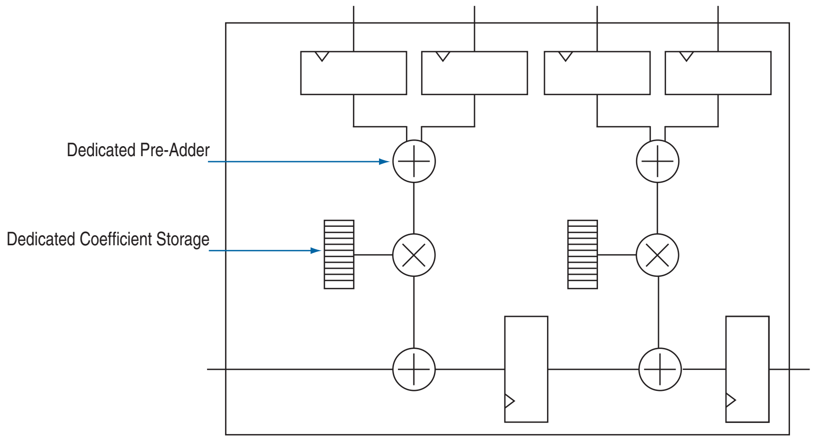
This Verilog HDL example generates a 128-tap single-channel single-rate systolic FIR filter with 18-bit symmetric coefficients that uses 32 DSP blocks.

After compilation, the Analysis & Synthesis DSP Block Usage Summary report shows the features of the DSP blocks that are configured for use. Table 5 summarizes the reported usage and Figure 27 shows the names of the relevant DSP block elements not already introduced in Figure 26 on page 24.

Table 5. DSP Block Usage for Verilog HDL Example Single-Rate Symmetric Systolic FIR Filter

Statistic	Number Used	Available per Block
Sum of two 18×18 with systolic register	32	1.00
DSP Block	32	—
DSP 18-bit Element	64	2.00
Signed Multiplier	64	—
Dedicated Pre-Adder	64	
Dedicated Coefficient Storage	64	
Dedicated Output Adder Chain	31	—

Figure 27. DSP Block Element Names Introduced in Table 5



Inferring a Multi-Channel Single-Rate Systolic FIR Filter

The design example in the **Verilog/multichannel_fir/18x18/systolic_chainout_adder** directory contains Verilog HDL code from which the Quartus II software infers a 128-tap 4-channel single-rate systolic FIR filter with 18-bit coefficients. The structure of this filter is shown in Figure 10 on page 11. The FIR filter must be clocked at the input data sample rate per channel times the number of channels (four in this case). The input data format understood by the FIR filter is the format shown in Figure 8 on page 10. If the FIR filter is clocked at four times the input data sample rate, the input data format is a simple interleaving of the data samples from channels a, b, c, and d, as shown in Figure 8. The output data format is identical.

The DSP blocks in this FIR filter are identical to those described in "Inferring a Single-Channel Single-Rate Systolic FIR Filter" on page 21 and shown in Figure 4 on page 8. The DSP block structure is inferred from code identical to the `dsp_block` module code shown in Example 1. The **chan4_fir.sv** file includes the following features:

- Definition of the chainout adder bit width per DSP block (as shown in Example 2 on page 23).
- Call to instantiate the additional delay registers in the tap delay line. These delay registers are required to enable the multi-channel implementation.

This Verilog HDL example generates a 128-tap four-channel single-rate systolic FIR filter with 18-bit coefficients that uses 64 DSP blocks.

After compilation, the Analysis & Synthesis DSP Block Usage Summary report shows the features of the DSP blocks that are configured for use. Table 6 summarizes the reported usage.

Table 6. DSP Block Usage for Verilog HDL Example Multi-Channel Single-Rate Systolic FIR Filter

Statistic	Number Used	Available per Block
Sum of two 18×18 with systolic register	64	1.00
DSP Block	64	—
DSP 18-bit Element	128	2.00
Signed Multiplier	128	—
Dedicated Output Adder Chain	63	—

Inferring a Multi-Channel Interpolation Systolic FIR Filter

The design example in the **Verilog/interpolation_fir/18x18/systolic_chainout_adder** directory contains Verilog HDL code from which the Quartus II software infers a 128-tap four-channel systolic interpolation FIR filter with 18-bit coefficients and rate change factor 4. The structure of this filter is shown in Figure 15 on page 16. The input data format understood by the FIR filter is the format shown in Figure 16 on page 16. The output data format is shown in Figure 17 on page 16.

During the mapping stage of compilation, when the Quartus II software identifies a coefficient select or a multiplier-adder with register, it maps the function to the appropriate device atoms. If the design targets a Stratix V device with DSP blocks, the Quartus II software can implement the function in a DSP block instead of in custom logic, depending on the device resources already committed.

Example 4 shows the Verilog HDL code in the **dsp_block.v** file. The sections that specify the coefficient select and the multiplier-adder with register are marked in red. Refer to Figure 15 on page 16 for the DSP block implementation that the Quartus II software infers from this Verilog HDL code. Each multiplier-adder plus register is mapped to an 18 × 18 DSP block element and each coefficient select is mapped to a dedicated coefficient storage element in the DSP block.

To ensure inference of the dedicated coefficient storage in the DSP blocks for multiple coefficient sets, the Verilog HDL code must specify that the coefficients are stored in ROM storage.

Example 4. Verilog HDL Code for Inferring 18×18 DSP Blocks in Systolic Array, Coefficient Storage, and Chainout Adder

```
module dsp_block (clk, aclr, ena, ay, by, coefsela, coefselb, chainin, chainout);

parameter coef_a0 = 0;
parameter coef_a1 = 0;
parameter coef_a2 = 0;
parameter coef_a3 = 0;
parameter coef_b0 = 0;
parameter coef_b1 = 0;
parameter coef_b2 = 0;
parameter coef_b3 = 0;
parameter DATA_WIDTH = 18;
parameter COEF_WIDTH = 18;
parameter CHAININ_WIDTH = 44;
parameter CHAINOUT_WIDTH = 44;

input clk, aclkr, ena;
input [1:0] coefsela, coefselb;
input signed [17:0] ay, by;
input signed [CHAININ_WIDTH-1:0] chainin;
output signed [CHAINOUT_WIDTH-1:0] chainout;

reg signed [CHAINOUT_WIDTH-1:0]  chainout;
reg signed [COEF_WIDTH-1:0] coefa, coefb;
reg signed [COEF_WIDTH-1:0] coefa_array[3:0], coefb_array[3:0];

initial begin
   coefa_array[0] <= coef_a0;    // Coefficients passed into the module
   coefa_array[1] <= coef_a1;    // are assigned to the dedicated coefficient storage
   coefa_array[2] <= coef_a2;    // in the DSP block.
   coefa_array[3] <= coef_a3;
   coefb_array[0] <= coef_b0;
   coefb_array[1] <= coef_b1;
   coefb_array[2] <= coef_b2;
   coefb_array[3] <= coef_b3;
end
always @ (coefsela)
begin
   coefa = coefa_array[coefsela];   // Coefficient select signals point to different
end
always @ (coefselb)
begin
   coefb = coefb_array[coefselb];   // addresses in the coefficient bank.
end
reg signed [DATA_WIDTH-1:0] ay_reg, by_reg;
reg signed [CHAINOUT_WIDTH-1:0] sa;
always @(posedge clk  or posedge aclr) begin
   if(aclr) begin
      ay_reg   <=  0;
      by_reg   <=  0;
      sa       <=  0;
      chainout <=  0;
   end
   else if(ena) begin
      ay_reg   <=  ay;
      by_reg   <=  by;
      sa       <=  chainin + ay_reg*coefa; // Multiplier-adder chain
      chainout <=  sa + by_reg*coefb;      // Multiplier-adder chain
   end
end
endmodule
```

To enable inference of the cascaded output bus in the systolic structure, the Quartus II software v10.1 requires that the incoming and outgoing adder chain widths be specified for each DSP block. Refer to Example 2 on page 23 in "Inferring a Single-Channel Single-Rate Systolic FIR Filter" on page 21.

This Verilog HDL example generates a 128-tap four-channel interpolation systolic FIR filter with 18-bit coefficients and rate change factor 4 that uses 16 DSP blocks.

After compilation, the Analysis & Synthesis DSP Block Usage Summary report shows the features of the DSP blocks that are configured for use. Table 7 summarizes the reported usage.

Table 7. DSP Block Usage for Verilog HDL Example Interpolation FIR Filter

Statistic	Number Used	Available per Block
Sum of two 18×18 with systolic register	16	1.00
DSP Block	16	—
DSP 18-bit Element	32	2.00
Signed Multiplier	32	—
Dedicated Coefficient Storage	32	—
Dedicated Output Adder Chain	15	—

Inferring a Multi-Channel Decimation Systolic FIR Filter

The design example in the **Verilog/decimation_fir/18x18/sum_of_2_chainout_adder** directory contains Verilog HDL code from which the Quartus II software infers a 128-tap 4-channel systolic decimation FIR filter with 18-bit coefficients and rate change factor 4. The structure of this filter is shown in Figure 22 on page 19. The input and output data formats are shown in Figure 23 on page 19.

During the mapping stage of compilation, when the Quartus II software identifies an accumulator, a coefficient select, a multiplier-adder with register, or a chainout adder function, it maps the function to the appropriate device atoms. If the design targets a Stratix V device with DSP blocks, the Quartus II software can implement the function in a DSP block instead of in custom logic, depending on the device resources already committed.

Example 5 shows the Verilog HDL code in the **dsp_block.v** file. The section that specifies the sum-of-two and accumulator and output adder chain is marked in red. Refer to Figure 22 on page 19 for the DSP block implementation that the Quartus II software infers from this Verilog HDL code. Each multiplier-adder plus register is mapped to an 18 × 18 DSP block element, each coefficient select is mapped to a dedicated coefficient storage element in the DSP block, and the sum-of-two plus accumulator plus output adder chain are mapped to the appropriate elements in the DSP block.

To ensure inference of the dedicated coefficient storage in the DSP blocks for multiple coefficient sets, the Verilog HDL code must specify that the coefficients are stored in ROM storage.

Example 5. Verilog HDL Code for Inferring 18×18 DSP Blocks in Systolic Array, Coefficient Storage, Chainout Adder, and Sum-of-Two in Accumulator Structure on Output Adder Chain

```
module dsp_block (clk, aclr, ena, ay, by, accum, coefsela, coefselb, chainin, chainout);

parameter coef_a0 = 0;
parameter coef_a1 = 0;
parameter coef_a2 = 0;
parameter coef_a3 = 0;
parameter coef_b0 = 0;
parameter coef_b1 = 0;
parameter coef_b2 = 0;
parameter coef_b3 = 0;
parameter DATA_WIDTH = 18;
parameter COEF_WIDTH = 18;
parameter CHAININ_WIDTH = 44;
parameter CHAINOUT_WIDTH = 44;

input clk, aclr, ena;
input accum;
input [1:0] coefsela, coefselb;
input signed [17:0] ay, by;
input signed [CHAININ_WIDTH-1:0] chainin;
output signed [CHAINOUT_WIDTH-1:0] chainout;

reg signed [CHAINOUT_WIDTH-1:0]  chainout;
reg signed [COEF_WIDTH-1:0] coefa, coefb;
reg signed [COEF_WIDTH-1:0] coefa_array[3:0], coefb_array[3:0];
initial begin
   coefa_array[0] <= coef_a0;
   coefa_array[1] <= coef_a1;
   coefa_array[2] <= coef_a2;
   coefa_array[3] <= coef_a3;
   coefb_array[0] <= coef_b0;
   coefb_array[1] <= coef_b1;
   coefb_array[2] <= coef_b2;
   coefb_array[3] <= coef_b3;
end
always @ (coefsela)
begin
   coefa = coefa_array[coefsela];
end
always @ (coefselb)
begin
   coefb = coefb_array[coefselb];
end

wire signed [CHAINOUT_WIDTH-1:0] acc_sel;
assign acc_sel = accum ? chainout : 0;
reg signed [DATA_WIDTH-1:0] ay_reg, by_reg;
always @(posedge clk  or posedge aclr) begin
   if(aclr) begin
      ay_reg   <=  0;
      by_reg   <=  0;
      chainout <=  0;
   end
   else if(ena) begin
      ay_reg   <=  ay;
      by_reg   <=  by;
      chainout <= acc_sel + (chainin + (ay_reg*coefa + by_reg*coefb));
   end
end
endmodule
```

To enable inference of the cascaded output bus in the systolic structure, the Quartus II software v10.1 requires that the incoming and outgoing adder chain widths be specified for each DSP block. Refer to Example 2 on page 23 in "Inferring a Single-Channel Single-Rate Systolic FIR Filter" on page 21.

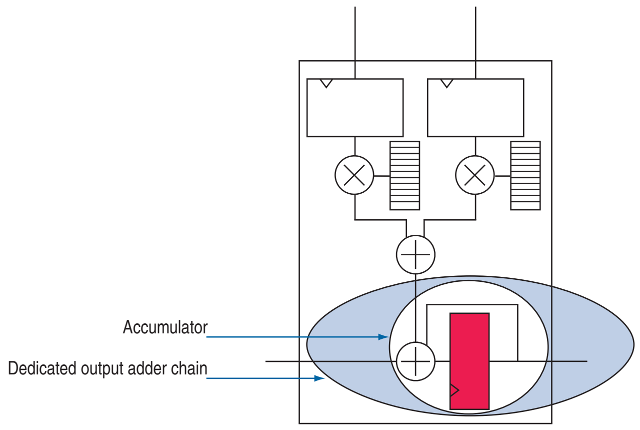
This Verilog HDL example generates a 128-tap four-channel decimation systolic FIR filter with 18-bit coefficients and rate change factor 4 that uses 16 DSP blocks.

After compilation, the Analysis & Synthesis DSP Block Usage Summary report shows the features of the DSP blocks that are configured for use. Table 8 summarizes the reported usage and Figure 28 shows the names of the relevant DSP block elements not already introduced in Figure 26 on page 24 and Figure 27 on page 26.

Table 8. DSP Block Usage for Verilog HDL Example Decimation FIR Filter

Statistic	Number Used	Available per Block
Sum of two 18×18 with systolic register	16	1.00
DSP Block	16	—
DSP 18-bit Element	32	2.00
Signed Multiplier	32	—
Dedicated Coefficient Storage	32	—
Dedicated Output Adder Chain	15	—
Dedicated Output Accumulator	16	—

Figure 28. DSP Block Element Names Introduced in Table 8



VHDL Examples

The VHDL examples demonstrate VHDL code that guides the Quartus II Fitter to infer Stratix V DSP blocks with specific operational modes. All of the VHDL examples require 27-bit by 27-bit multiplication. Coefficients are 27 bits wide and input data is 25 bits wide. Therefore, the Quartus II software configures only a single multiplier path in each DSP block.

The VHDL coding style in these examples includes the use of multidimensional arrays to manage signals to multipliers. This coding style creates compact code and enhances readability. However, it requires custom data types, which may in turn require type conversion in the code. The VHDL code examples include a package definition file that specifies the necessary data types. The dimensions of all array types are fully parameterizable.

Inferring a Single-Channel Single-Rate Symmetric Systolic FIR Filter

The design example in the **VHDL/systolic_fir** directory contains VHDL code from which the Quartus II software infers a 64-tap single-channel single-rate systolic FIR filter with 27-bit symmetric coefficients. The structure of this filter is shown in Figure 7 on page 10. During the mapping stage of compilation, when the Quartus II software identifies a two-input adder or a multiplier-adder with register, it maps the function to the appropriate device atoms. If the design targets a Stratix V device with DSP blocks, the Quartus II software can implement this function in a DSP block instead of in custom logic, depending on the device resources already committed.

Example 6 shows the VHDL code from which the Quartus II software infers the DSP block structure shown in Figure 6 on page 9. You can find this HDL code in the **VHDL/systolic_fir/systolic_fir.vhd** file. The sections that specify the two-input adder and the multiplier-adder with register are shown. Each two-input adder is mapped to a dedicated pre-adder and each multiplier-adder plus register is mapped to a 27 × 27 DSP block element.

Example 6. VHDL Code for Inferring 27×27 DSP Blocks in Systolic Array with Pre-Adder and Chainout Adder

```
preadder_map : for j in 0 to engine_size_c-1 generate
   preadder_out(j) <= resize(taps(j), din_width_c + 1)
                    + resize(taps(engine_size_c), din_width_c + 1);
 end generate preadder_map;

 first_mult : process (reset, clk) is
   variable current_adder_out : signed(chainout_bitwidth_c(0)-1 downto 0);
 begin  -- process mult
   if reset = '1' then
     current_adder_out := (others => '0');
   elsif rising_edge(clk) and (reg_in_valid = '1') then
     current_adder_out := preadder_out(0)*coeff(0);
   end if;
   chainout(0) <= resize(current_adder_out, dout_width_c);
 end process first_mult;

 mult_map : for n in 1 to engine_size_c -1 generate
   mult : process (reset, clk) is
     variable current_adder_out : signed(chainout_bitwidth_c(n)-1 downto 0);
   begin  -- process mult
     if reset = '1' then
       current_adder_out := (others => '0');
     elsif rising_edge(clk) and (reg_in_valid = '1') then
       -- modelsim complains width mismatch. had to use resize
       current_adder_out := resize(chainout(n-1), chainout_bitwidth_c(n))
                          + resize(preadder_out(n)*coeff(n), chainout_bitwidth_c(n));
     end if;
     chainout(n) <= resize(current_adder_out, dout_width_c);
              // Value of dout_width_c is declared at start of file to be
              //58 = din_width_c+1+coeff_width_c+ ceiling(log2(engine_size_c))
   end process mult;
 end generate mult_map;
```

A single coefficient set is stored in each DSP block's dedicated coefficient buffer. The Quartus II software infers internal storage for a single coefficient set if the HDL code declares the coefficients as constants, as is done in **systolic_fir.vhd**.

To enable inference of the pre-adder inside the DSP block, the summation result must be specified to be one bit wider than the incoming data. The VHDL resize function is marked in red in Example 6.

To enable inference of the cascaded output bus in the systolic structure, the Quartus II software v10.1 requires that the incoming and outgoing adder chain widths be specified for each DSP block. Example 6 shows the HDL code to meet this requirement, based on the VHDL package definition of a constant array `chainout_bitwidth_c`. The array values ensure that the output of each multiplication operation has declared width exactly the sum of the multiplicand data width, the coefficient data width, and the required bit growth width. The bit growth equation for adder n on the chain is ceiling($\log_2(n)$) for n in 1,..., `engine_size_c`. The value of `dout_width_c` is also declared as a constant in the VHDL package. The VHDL package definition is in the **systolic_type_pkg.vhd** file.

This VHDL example generates a 64-tap single-channel single-rate systolic FIR filter with 27-bit symmetric coefficients that uses 32 DSP blocks. The number of delay elements in the tap-delay line is 62.

After compilation, the Analysis & Synthesis DSP Block Usage Summary report shows the features of the DSP blocks that are configured for use. Table 9 summarizes the reported usage.

Table 9. DSP Block Usage for VHDL Example Single-Channel Single-Rate FIR Filter

Statistic	Number Used	Available per Block
Independent 27×27 (one)	32	1.00
DSP Block	32	—
DSP 27-bit Element	32	1.00
Signed Multiplier	32	—
Dedicated Output Adder Chain	31	—

Inferring a Multi-Channel Single-Rate Symmetric Systolic FIR Filter

The design example in the **VHDL/multichan_systolic_fir** directory contains VHDL code from which the Quartus II software infers a 128-tap 12-channel single-rate systolic FIR filter with 27-bit symmetric coefficients. The structure of this filter is shown in Figure 11 on page 12. During the mapping stage of compilation, when the Quartus II software identifies a two-input adder or a multiplier-adder with register, it maps the function to the appropriate device atoms. If the design targets a Stratix V device with DSP blocks, the Quartus II software can implement this function in a DSP block instead of in custom logic, depending on the device resources already committed.

The DSP blocks that implement this FIR filter are identical to the DSP blocks inferred for the single-channel symmetric FIR filter, and are inferred from the same code. Refer to "Inferring a Single-Channel Single-Rate Symmetric Systolic FIR Filter" on page 33. However, the tap-delay line is different.

Example 7 and Example 8 show the VHDL code from which the Quartus II software infers the tap-delay line shown in Figure 11 on page 12. The value of C in Figure 11 is 12 in this example. Because the forward, backward, and folding-point taps all have different depths, the code is written to instantiate three separate altshift_taps megafunction instances and concatenate them together. Example 7 shows the VHDL code from which an altshift_taps megafunction is inferred, and Example 8 shows the VHDL code for instantiating the three instances and concatenating them together. This code is located in the VHDL **tap_delay_chain.vhd** file.

Example 7. VHDL Code for Inferring an altshift_taps Megafunction

```
component shift_taps is
   generic (
     din_width_c         : natural;
     entire_tap_length_c : natural;
     tap_depth_c         : natural;
     num_taps_c          : natural);
   port (
     clk       : in  std_logic;
     reset     : in  std_logic;
     en        : in  std_logic;
     data_in   : in  signed(din_width_c-1 downto 0);
     data_taps : out tap_array_type(0 to num_taps_c-1);
     data_out  : out signed(din_width_c-1 downto 0)
     );
 end component shift_taps;
```

Table 10 shows the parameters that support inference of the altshift_taps megafunction. For successful inference, in addition, the component ports must match the megafunction ports.

Table 10. VHDL Generics for Inferring an altshift_taps Megafunction

VHDL Generic	Description
`din_width_c`	Width of incoming data to the tap-delay line.
`entire_tap_length_c`	Size of the tap-delay line. Value is `tap_depth_c` × `num_taps_c`.
`tap_depth_c`	Number of delay elements between two adjacent taps in the tap-delay line.
`num_taps_c`	Number of output taps off the tap-delay line.

Using the altshift_taps megafunction ensures the delay taps are implemented in on-chip memory.

For more information about the altshift_taps megafunction, refer to the *altshift_taps Megafunction User Guide*.

For more information about writing HDL code to infer an altshift_taps megafunction instance, refer to *Recommended HDL Coding Styles* in volume 1 of the *Quartus II Handbook*.

Example 8 shows the VHDL code for instantiating the three instances and concatenating them together.

Example 8. VHDL Code for Inferring Three altshift_taps Instances for a Multi-Channel Symmetric FIR Filter

```
architecture arch of tap_delay_chain is

  signal tap_forward  : tap_array_type(0 to engine_size_c-1);
  signal tap_fold     : tap_array_type(0 to 0);
  signal tap_backward : tap_array_type(0 to engine_size_c-2);

....

begin
  -- forward path each tap is period_c + 1 deep.  Number of taps excludes the
  -- first tap.
  forward_tap : component shift_taps
    generic map (
      din_width_c          => din_width_c,
      entire_tap_length_c => (engine_size_c-1)*(period_c + 1),
      tap_depth_c          => period_c+1,
      num_taps_c           => engine_size_c-1)
    port map (
      clk       => clk,
      reset     => reset,
      en        => en,
      data_in   => data_in,
      data_taps => tap_forward(1 to engine_size_c -1));
      tap_forward(0) <= data_in;

  -- the folding point has one tap, but it is period_c deep
  mid_tap : component shift_taps
    generic map (
      din_width_c          => din_width_c,
      entire_tap_length_c => period_c,
      tap_depth_c          => period_c,
      num_taps_c           => 1)

port map (
      clk       => clk,
      reset     => reset,
      en        => en,
      data_in   => tap_forward(engine_size_c-1),
      data_taps => tap_fold);

  --backward taps are period_c-1 deep
  backward_tap : component shift_taps
    generic map (
      din_width_c          => din_width_c,
      entire_tap_length_c => (engine_size_c-1)*(period_c - 1),
      tap_depth_c          => period_c -1,
      num_taps_c           => engine_size_c-1)
    port map (
      clk       => clk,
      reset     => reset,
      en        => en,
      data_in   => tap_fold(0),
      data_taps => tap_backward);

  data_taps(0 to engine_size_c-1)                   <= tap_forward;
  data_taps(engine_size_c)                          <= tap_fold(0);
  data_taps(engine_size_c + 1 to 2*engine_size_c-1) <= tap_backward;
end architecture arch;
```

This VHDL example generates a 128-tap 12-channel single-rate systolic FIR filter with 27-bit symmetric coefficients that uses 64 DSP blocks. The number of delay elements in the tap-delay line is 1461.

After compilation, the Analysis & Synthesis DSP Block Usage Summary report shows the features of the DSP blocks that are configured for use. Table 11 summarizes the reported usage.

Table 11. DSP Block Usage for VHDL Example Multi-Channel Single-Rate FIR Filter

Statistic	Number Used	Available per Block
Independent 27×27 (one)	64	1.00
DSP Block	64	—
DSP 27-bit Element	64	1.00
Signed Multiplier	64	—
Dedicated Output Adder Chain	63	—

Inferring a Multi-Channel Interpolation Direct Form FIR Filter

The design example in the **VHDL/interpolation_fir** directory contains VHDL code from which the Quartus II software infers a 128-tap four-channel interpolation direct-form FIR filter with 27-bit coefficients and rate change factor 4. The structure of this filter is shown in Figure 20 on page 17. The MUX at the input to each multiplier selects the current polyphase filter data from the <*rate change factor*> taps that feed the current multiplier. The MUXes have a common select line. The input data format understood by the FIR filter is the format shown in Figure 18 on page 17. The output data format is shown in Figure 19 on page 17.

During the mapping stage of compilation, when the Quartus II software identifies a coefficient select, a multiplier, or a multiplier-adder, it maps the function to the appropriate device atoms. If the design targets a Stratix V device with DSP blocks, the Quartus II software can implement this function in a DSP block instead of in custom logic, depending on the device resources already committed.

In Figure 3, consecutive DSP blocks alternate in their configuration. Pairs of blocks are configured in two 27×27 multiplier adder operational mode. To ensure the Quartus II software infers the multipliers in DSP blocks, you must resize the multiplication result to be exactly the sum of the multiplicand data width and the coefficient data width. To ensure the Quartus II software infers the chainout adder, you must declare its addition output bit width to be exactly one more than the multiplier output signal width in each DSP block.

Example 9 shows the VHDL code that enforces the width of the multiplication result. This code is located in the VHDL **interpolation_fir.vhd** file.

Example 9. VHDL Code for Inferring the Width of the Multiplication Result

```
mult_map : for n in 0 to engine_size_c -1 generate
    mult : process (reset, clk) is
    begin  -- process mult
      if reset = '1' then
        multout(n) <= (others => '0');
      elsif rising_edge(clk) and (reg_in_valid = '1') then
        multout(n) <= resize(taps(n)*coeff_row(n), din_width_c+coeff_width_c);
      end if;
    end process mult;
  end generate mult_map;
```

Example 10 shows the VHDL code that enforces the width of the chainout adder in every second DSP block. This code is located in the VHDL **interpolation_fir.vhd** file.

Example 10. VHDL Code for Inferring the Width of the Chainout Adder

```
sumof2_gen : for k in 0 to engine_size_c/2-1 generate
    sumof2_add : process (multout(2*k), multout(2*k+1)) is
    begin  -- process sumof2_add
      sumof2(k) <= resize(multout(2*k), sumof2_bitwidth_c)
                     + resize(multout(2*k+1), sumof2_bitwidth_c);
    end process sumof2_add;
  end generate sumof2_gen;
```

The multi-channel interpolation FIR filter requires four coefficient sets, to match the number of polyphase decomposed subfilters—the rate change factor. For multiple coefficient sets, the Quartus II software infers the dedicated coefficient storage in DSP blocks only if the coefficients are stored in a ROM structure.

Example 11 and Example 12 show the VHDL code that meets this requirement. The code in Example 11 creates ROM initialization values for the coefficients in a DSP block. The Quartus II software infers a ROM whose depth is the rate change factor for the FIR filter. This code is located in the VHDL **coeff_rom.vhd** file.

Example 11. VHDL Code to Infer Dedicated Coefficient Storage Per DSP Block

```
entity coeff_rom is
generic
    (coeff_width_c : natural := 27;
     contents_c    : memory_t;          -- coefficients for 1 multiplier across
                                        -- rate_c polyphases
     rate_c        : natural := 4);
  port
    (addr : in  natural range 0 to rate_c - 1;
     q    : out word_t);
end entity;

architecture rtl of coeff_rom is
  -- Build a 2-D array type for the RoM
  -- initialize ROM contents to the input coefficients vector
  function init_rom
    return memory_t is
    variable tmp : memory_t := (others => (others => '0'));
  begin
    for addr_pos in 0 to rate_c - 1 loop
      -- Initialize each address with the coeff vector
      tmp(addr_pos) := contents_c(addr_pos);
    end loop;
    return tmp;
  end init_rom;

  -- Declare the ROM signal and specify a default value. The Quartus II
  -- software will create a memory initialization file (.mif) based on the
  -- default value.
  signal rom : memory_t := init_rom;
-- this is combinational logic
begin
  process(addr)
  begin
    q <= rom(addr);
  end process;
end rtl;
```

The code in Example 12 instantiates one of these ROM blocks for each DSP block and collects the coefficients for the current polyphase for all the blocks in an array `coeff_row`. This code is located in the VHDL **interpolation_fir.vhd** file.

Example 12. VHDL Code to Use Coefficient Storage Correctly Across DSP Blocks

```
-- an engine_size_c by rate_c matrix; each row is used to initialize a coeff ROM
  constant coeff_matrix_signed_c : memory_array_t := coeff_type_conv;
-- an engine_size_c by 1 vector, each element feeds a DSP block
  signal coeff_row               : coeff_row_signed_t;

rom_gen : for i in 0 to engine_size_c-1 generate
    rom_inst : component coeff_rom
      generic map (
        coeff_width_c => coeff_width_c,
        contents_c    => coeff_matrix_signed_c(i),
        rate_c        => rate_c)
      port map (
        addr => sel,
        q    => coeff_row(i));
  end generate rom_gen;
```

The select signal controls a data MUX and coefficient selection, to ensure the input data and coefficients from the same polyphase filter are aligned properly.

The multi-channel interpolation FIR filter tap-delay line is implemented with the altshift_taps megafunction. Because this FIR filter does not assume symmetric coefficients, only a single altshift_taps instance is required. The VHDL code to ensure inference of the altshift_taps megafunction appears in the VHDL **shift_taps.vhd** file. This code includes the entity definition shown in Example 7 on page 35 and a simplified version of the instantiations in Example 8 on page 36. The distance between taps is 4, the ratio of the device clock frequency to the input data channel rate, divided by the number of channels.

This VHDL example generates a 128-tap four-channel interpolation direct-form FIR filter with 27-bit symmetric coefficients and rate change factor 4 that uses 32 DSP blocks. The number of delay elements in the tap-delay line is 496.

After compilation, the Analysis & Synthesis DSP Block Usage Summary report shows the features of the DSP blocks that are configured for use. Table 12 summarizes the reported usage.

Table 12. DSP Block Usage for VHDL Example Decimation FIR Filter

Statistic	Number Used	Available per Block
Sum of two 27×27	16	0.5
DSP Block	32	—
DSP 27-bit Element	32	1.00
Signed Multiplier	32	—
Dedicated Coefficient Storage	32	—

Figure 13 on page 14 shows the input data provided for this design example, and Figure 14 on page 15 shows the result of simulating the design example testbench on this input data.

Inferring a Multi-Channel Decimation Direct Form FIR Filter

The design example in the **VHDL/decimation_fir** directory contains VHDL code from which the Quartus II software infers a 128-tap four-channel decimation direct-form FIR filter with 27-bit coefficients and rate change factor 4. The structure of this filter is shown in Figure 25 on page 20. The accumulator at the final stage supports multiple channels. The input and output data formats are shown in Figure 24 on page 19.

The multi-channel decimation direct form FIR filter does not support inference of the DSP block internal accumulator, because of the choice of data format. With the interleaved output data format in Figure 24, accumulating requires multiple cycles of delay. The output data format used by the Verilog HDL multi-channel interpolation FIR filter example supports a single-cycle delay between data words from the same channel. Therefore, it supports the inference of the DSP block internal accumulator. However, the output data format used by the VHDL multi-channel interpolation FIR filter example does not support single-cycle delay between data words from the same channel. If the number of channels is 1, you might be able to use the internal accumulator for this example to improve latency and resource utilization, depending on your operating clock rate.

During the mapping stage of compilation, when the Quartus II software identifies a coefficient select, a multiplier, or a multiplier-adder, it maps the function to the appropriate device atoms. If the design targets a Stratix V device with DSP blocks, the Quartus II software can implement this function in a DSP block instead of in custom logic, depending on the device resources already committed.

The VHDL code from which the Quartus II software infers the tap-delay line, and from which it infers the multipliers, coefficient storage, and chainout adder configuration in the DSP blocks, is the same as the code for the interpolation direct-form FIR filter described in "Inferring a Multi-Channel Interpolation Direct Form FIR Filter" on page 37.

The distance between taps is 16, which is the ratio of the device clock frequency to the input data channel rate. The final accumulator delay is 4.

This VHDL example generates a 128-tap four-channel decimation direct-form FIR filter with 27-bit symmetric coefficients and rate change factor 4 that uses 32 DSP blocks. The number of delay elements in the tap-delay line is 496.

After compilation, the Analysis & Synthesis DSP Block Usage Summary report shows the features of the DSP blocks that are configured for use. Table 13 summarizes the reported usage.

Table 13. DSP Block Usage for VHDL Example Interpolation FIR Filter

Statistic	Number Used	Available per Block
Sum of two 27×27	16	0.5
DSP Block	32	—
DSP 27-bit Element	32	1.00
Signed Multiplier	32	—
Dedicated Coefficient Storage	32	—

Document Revision History

Table 14 shows the revision history for this application note.

Table 14. Document Revision History

Date	Version	Changes
March 2017	2.0	Updated the URL to the example designs in the “Downloading the Example Applications” section.
January 2011	1.0	Initial release.