FIR Compiler

User Guide

101 Innovation Drive
San Jose, CA 95134
www.altera.com

Software Version: 9.0
Document Date: March 2009

Copyright © 2009 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

I.S. EN ISO 9001

Contents

Chapter 1. About This Compiler

Release Information . . . 1–1
Device Family Support . . . 1–1
Features . . . 1–2
General Description . . . 1–3
MegaCore Verification . . . 1–5
Performance and Resource Utilization . . . 1–5
Installation and Licensing . . . 1–7
OpenCore Plus Evaluation . . . 1–8
OpenCore Plus Time-Out Behavior . . . 1–8

Chapter 2. Getting Started

Design Flows . . . 2–1
DSP Builder Flow . . . 2–1
MegaWizard Plug-In Manager Flow . . . 2–2
Parameterize the MegaCore Function . . . 2–3
Generate the MegaCore Function . . . 2–6
Simulate the Design . . . 2–8
Simulating in Model*Sim* . . . *2–8*
Simulating in MATLAB . . . 2–9
Simulating in Third-Party Simulation Tools Using NativeLink . . . 2–9
Compile the Design and Program a Device . . . 2–9

Chapter 3. Parameter Settings

Specifying the Coefficients . . . 3–1
Using the FIR Compiler Coefficient Generator . . . 3–2
Loading Coefficients from a File . . . 3–6
Analyzing the Coefficients . . . 3–8
Specify the Input and Output Specifications . . . 3–9
Specify the Architecture Specification . . . 3–11
Resource Estimates . . . 3–16
Filter Design Tips . . . 3–16

Chapter 4. Functional Description

FIR Compiler . . . 4–1
Number Systems & Fixed-Point Precision . . . 4–1
Generating or Importing Coefficients . . . 4–1
Coefficient Scaling . . . 4–2
Symmetrical Architecture Selection . . . 4–3
Symmetrical Serial . . . 4–3
Coefficient Reloading and Reordering . . . 4–4
Structure Types . . . 4–6
Multicycle Variable Structures . . . 4–6
Parallel Structures . . . 4–6
Serial Structures . . . 4–7
Multibit Serial Structure . . . 4–7
Multichannel Structures . . . 4–8
Interpolation & Decimation . . . 4–8

Implementation Details for Interpolation & Decimation Structures 4–10
Availability of Interpolation & Decimation Filters 4–11
Half-Band Decimation Filters 4–12
Symmetric-Interpolation Filters 4–12
Pipelining 4–12
Simulation Output 4–13
Avalon Streaming Interface 4–13
Avalon-ST Data Transfer Timing 4–14
Packet Data Transfers 4–15
Signals 4–16
Timing Diagrams 4–17
Reset and Global Clock Enable Operations 4–18
Single Rate Filter Timing Diagram 4–18
Interpolation Filter Timing Diagrams 4–20
Decimation Filter Timing Diagrams 4–21
Coefficient Reloading Timing Diagrams 4–22
References 4–26

Appendix A. FIR Compiler Supported Device Structures
Supported Device Structures A–1
HardCopy II Support A–3
Compiling HardCopy II Designs A–3

Additional Information Info–1
Revision History Info–1
How to Contact Altera Info–2
Typographic Conventions Info–2

1. About This Compiler

Release Information

Table 1–1 provides information about this release of the Altera® FIR Compiler.

Table 1–1. FIR Compiler Release Information

Item	Description
Version	9.0
Release Date	March 2009
Ordering Code	IP-FIR
Product ID	0012
Vendor ID	6AF7

For more information about this release, refer to the *MegaCore IP Library Release Notes and Errata*.

Altera verifies that the current version of the Quartus® II software compiles the previous version of each MegaCore® function. The *MegaCore IP Library Release Notes and Errata* report any exceptions to this verification. Altera does not verify compilation with MegaCore function versions older than one release.

Device Family Support

MegaCore® functions provide either full or preliminary support for target Altera device families, as described below:

- *Full support* means the MegaCore function meets all functional and timing requirements for the device family and may be used in production designs
- *Preliminary* support means the MegaCore function meets all functional requirements, but may still be undergoing timing analysis for the device family; it may be used in production designs with caution.

Table 1–2 shows the level of support offered by the FIR Compiler to each Altera device family.

Table 1–2. Device Family Support (Part 1 of 2)

Device Family	Support
Arria® II GX	Preliminary
Arria GX	Full
Cyclone®	Full
Cyclone II	Full
Cyclone III	Full
HardCopy® II	Full
Stratix®	Full

Table 1–2. Device Family Support (Continued) (Part 2 of 2)

Device Family	Support
Stratix II	Full
Stratix II GX	Full
Stratix III	Full
Stratix IV	Preliminary
Stratix GX	Full
Other device families	No support *(Note 1)*

Notes to Table 1–2:

(1) If you want to use Hardcopy Stratix devices, select the **Stratix** family and then browse through the available devices for **<*device*>_HARDCOPY_FPGA_PROTOTYPE**.

Features

The Altera® FIR Compiler implements a finite impulse response (FIR) filter MegaCore function and supports the following features:

- Multiple hardware architectures to enable optimal trade- offs between logic, memory, DSP blocks, and performance:
  - Fully parallel distributed arithmetic
  - Fully serial distributed arithmetic
  - Multibit serial distributed arithmetic
  - Multicycle variable structures
- Exploit maximal efficiency designs as a result of FIR Compiler hardware optimizations such as:
  - Interpolation
  - Decimation
  - Symmetry
  - Decimation half-band
  - Time sharing
- Easy system integration using Avalon® Streaming (Avalon-ST) interfaces.
- Precision control of chip resource utilization:
  - Logic cells, M512, M4K, M-RAM, MLAB, M9K, or M144K for data storage.
  - M512, M4K, M9K, MLAB or logic cells for coefficient storage.
  - Includes a resource estimator.
- Support for run-time coefficient reloading capability and multiple coefficient sets.
- Includes a built-in coefficient generator to enable efficient design space exploration.
- User-selectable output precision via rounding and saturation.
- DSP Builder ready.

General Description

The Altera FIR Compiler provides a fully integrated finite impulse response (FIR) filter development environment optimized for use with Altera FPGA devices.

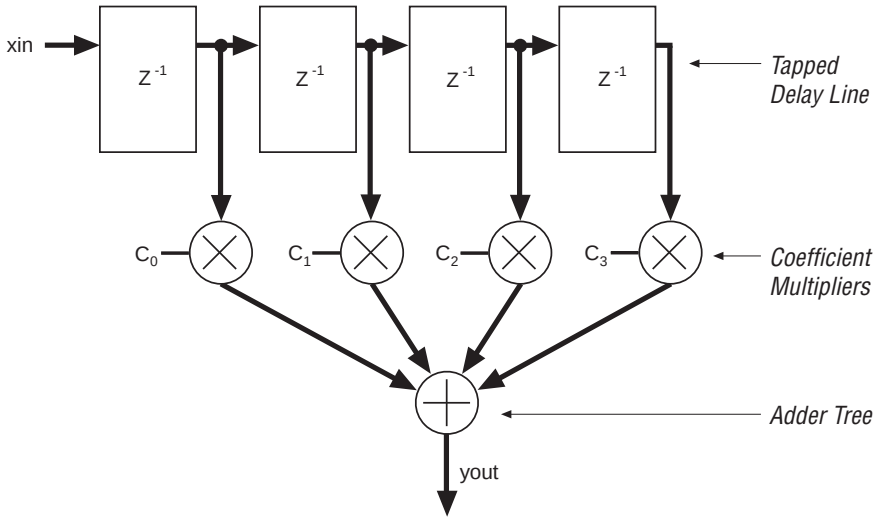
You can use the IP Toolbench interface to implement a variety of filter architectures, including fully parallel, serial, or multibit serial distributed arithmetic, and multicycle fixed/variable filters. The FIR Compiler includes a coefficient generator.

Many digital systems use signal filtering to remove unwanted noise, to provide spectral shaping, or to perform signal detection or analysis. Two types of filters that provide these functions are finite impulse response (FIR) filters and infinite impulse response (IIR) filters. Typical filter applications include signal preconditioning, band selection, and low-pass filtering.

In contrast to IIR filters, FIR filters have a linear phase and inherent stability. This benefit makes FIR filters attractive enough to be designed into a large number of systems. However, for a given frequency response, FIR filters are a higher order than IIR filters, making FIR filters more computationally expensive.

The structure of a FIR filter is a weighted, tapped delay line as shown in Figure 1–1.

Figure 1–1. Basic FIR Filter



The filter design process involves identifying coefficients that match the frequency response specified for the system. These coefficients determine the response of the filter. You can change which signal frequencies pass through the filter by changing the coefficient values or adding more coefficients.

Traditionally, designers have been forced to make a trade-off between the flexibility of digital signal processors and the performance of ASICs and application-specific standard product (ASSPs) digital signal processing (DSP) solutions. The Altera DSP solution reduces the need for this trade-off by providing exceptional performance combined with the flexibility of FPGAs.

Figure 1–2 compares the design cycle using a FIR Compiler MegaCore function with a traditional implementation.

Figure 1–2. Design Cycle Comparison

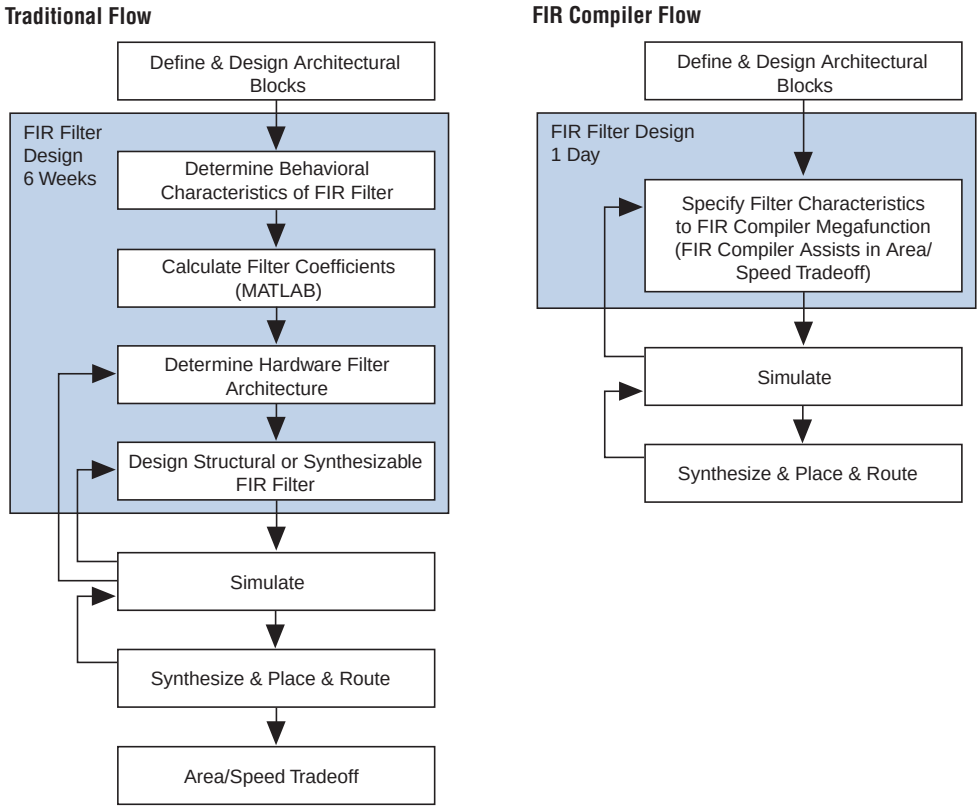
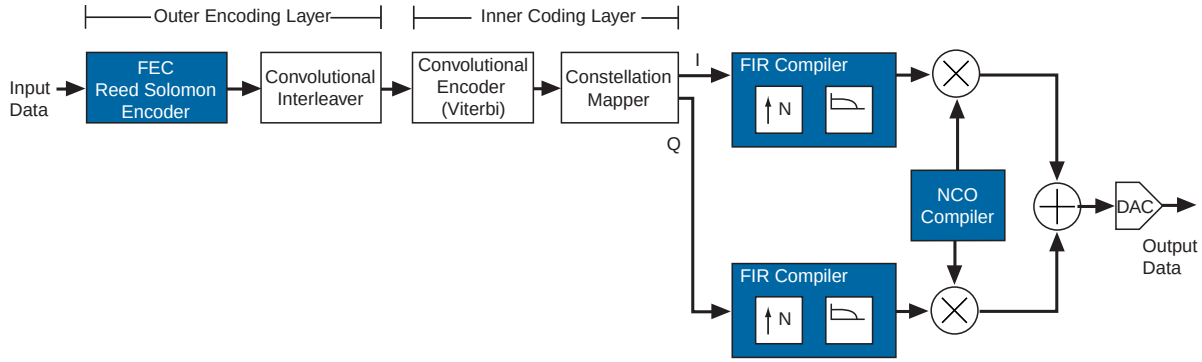


Figure 1–3 shows a typical DSP system that uses Altera MegaCore functions.

Figure 1–3. Typical Modulator System



DSP processors have a limited number of multiply accumulators (MACs), and require many clock cycles to compute each output value (the number of cycles is directly related to the order of the filter).

A dedicated hardware solution can achieve one output per clock cycle. A fully parallel, pipelined FIR filter implemented in an FPGA can operate at very high data rates, making FPGAs ideal for high-speed filtering applications.

Table 1–3 compares resource usage and performance for different implementations of a 120-tap FIR filter with a 12-bit data input bus.

Table 1–3. FIR Filter Implementation Comparison *(Note 1)*

Device	Implementation	Clock Cycles to Compute Result
DSP processor	1 MAC	120
FPGA	1 serial filter	12
	1 parallel filter	1

Note to Table 1–3:

(1) If you use the FIR Compiler to create a filter, you can also implement a variable filter in a FPGA that uses from 1 to 120 MACs, and 120 to 1 clock cycles.

The FIR Compiler speeds the design cycle by:

- Generating the coefficients needed to design custom FIR filters.
- Generating bit-accurate and clock-cycle-accurate FIR filter models (also known as bit-true models) in the Verilog HDL and VHDL languages and in the MATLAB environment.
- Automatically generating the code required for the Quartus II software to synthesize high-speed, area-efficient FIR filters of various architectures.
- Generating a VHDL testbench for all architectures.

MegaCore Verification

Before releasing an updated version of the FIR Compiler, Altera runs a comprehensive regression test to verify its quality and correctness.

All features and architectures are tested by sweeping all parameter options and verifying that the simulation matches a master functional model.

Performance and Resource Utilization

This section shows typical expected performance for a FIR Compiler MegaCore function using the Quartus II software, version 8.0 with Cyclone III, and Stratix IV devices. All figures are given for a FIR filter with 97 taps, 8-bit input data, 14-bit coefficients, and a target f_{MAX} set to 1GHz.

Cyclone III devices use combinational look-up tables (LUTs) and logic registers; Stratix IV devices use combinational adaptive look-up tables (ALUTs) and logic registers.

The resource and performance data was generated with the source ready signal (`ast_source_ready`) always driven high, as described in "Avalon Streaming Interface" on page 4–13.

Table 1–4 shows performance figures for Cyclone III devices:

Table 1–4. FIR Compiler Performance—Cyclone III Devices

Combinational LUTs	Logic Registers	Memory *(6)*		Multipliers (9x9)	f_{max} (MHz)	Throughput (MSPS)	Processing Equivalent (GMACs) *(1)*
		Bits	M9K				
Multibit Serial, pipeline level 1 *(2)*, *(3)*							
899	1,331	55,148	31	—	310	62	6
Multicycle variable (1 cycle) decimation by 4, pipeline level 1 *(2)*, *(3)*							
857	1,336	1,158	12	26	281	281	27
Multicycle variable (1 cycle) interpolation by 4, pipeline level 2 *(4)*							
1,528	2,657	66	1	50	290	290	28
Multicycle variable (1 cycle), pipeline level 2 *(2)*, *(4)*							
2,543	4,837	92	1	98	280	280	27
Multicycle variable (4 cycle), pipeline level 2 *(2)*, *(3)*							
1,182	1,715	578	9	26	283	71	7
Parallel (LE), pipeline level 1*(2)*, *(3)*							
3,416	3,715	208	3	—	288	288	28
Parallel (M9K), pipeline level 1 *(2)*, *(5)*							
1,948	2,155	120,030	48	—	283	283	27
Serial (M9K), pipeline level 1 *(2)*, *(3)*							
327	462	14,167	8	—	323	36	3

Notes to Table 1–4:

(1) GMAC = giga multiply accumulates per second (1 giga = 1,000 million).
(2) This FIR filter takes advantage of symmetric coefficients.
(3) Using EP3C10F256C6 devices.
(4) Using EP3C16F484C6 devices.
(5) Using EP3C40F780C6 devices.
(6) It may be possible to significantly reduce memory utilization by setting a lower target f_{MAX}.

Table 1–5 shows performance figures for Stratix IV devices:

Table 1–5. FIR Compiler Performance—Stratix IV Devices (Part 1 of 2)

Combinational ALUTs	Logic Registers	Memory			Multipliers (18x18)	f_{max} (MHz)	Throughput (MSPS)	Processing Equivalent (GMACS) *(1)*
		Bits	(M9K)	ALUTs				
Multibit Serial, pipeline level 1 *(2)*, *(3)*, *(4)*								
766	1,166	55,276	42	16	—	503	101	10
Multicycle variable (1 cycle) decimation by 4, pipeline level 1 *(2)*, *(3)*								
336	844	1,400	16	28	14	443	443	43
Multicycle variable (1 cycle) interpolation by 4, pipeline level 2 *(3)*								
200	1,274	64	—	8	24	372	372	36
Multicycle variable (1 cycle), pipeline level 2 *(2)*, *(3)*								
741	1,936	148	1	8	48	443	443	43

Table 1–5. FIR Compiler Performance—Stratix IV Devices (Part 2 of 2)

Combinational ALUTs	Logic Registers	Memory			Multipliers (18x18)	f_{max} (MHz)	Throughput (MSPS)	Processing Equivalent (GMACS) *(1)*
		Bits	(M9K)	ALUTs				
Multicycle variable (4 cycle), pipeline level 2 *(2)*, *(3)*								
717	1,398	796	6	36	14	323	81	8
Parallel (LE), pipeline level 1 *(2)*, *(3)*								
2,153	2,672	157	1	8	—	421	421	41
Parallel (M9K), pipeline level 1 *(3)*								
821	1,730	119,872	45	8	—	457	457	44
Serial (M9K), pipeline level 1 *(2)*, *(3)*								
245	415	14,231	11	8	—	523	58	6

Notes to Table 1–5:

(1) GMAC = giga multiply accumulates per second (1 giga = 1,000 million).
(2) This FIR filter takes advantage of symmetric coefficients.
(3) Using EP4SGX70DF29C2X devices.
(4) The data width is 16-bits and there are 4 serial units.

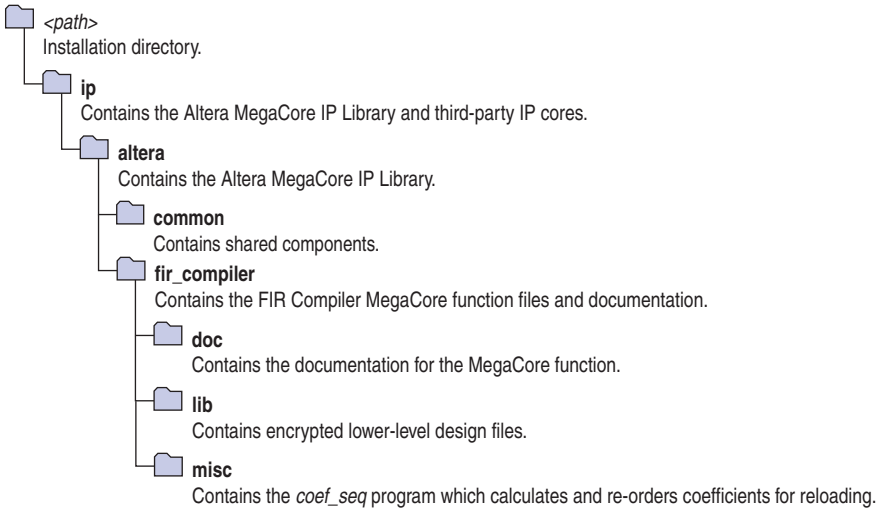
Installation and Licensing

The FIR Compiler MegaCore function is part of the MegaCore® IP Library, which is distributed with the Quartus® II software and downloadable from the Altera® website, www.altera.com.

For system requirements and installation instructions, refer to *Quartus II Installation & Licensing for Windows and Linux Workstations*.

Figure 1–4 shows the directory structure after you install the FIR Compiler, where <*path*> is the installation directory for the Quartus II software. The default installation directory on Windows is **c:\altera**<***version***>; or on Linux is **/opt/altera**<***version***>.

Figure 1–4. Directory Structure



OpenCore Plus Evaluation

With Altera's free OpenCore Plus evaluation feature, you can perform the following actions:

- Simulate the behavior of a megafunction (Altera MegaCore function or AMPP^{SM} megafunction) within your system.
- Verify the functionality of your design, as well as evaluate its size and speed quickly and easily.
- Generate time-limited device programming files for designs that include megafunctions.
- Program a device and verify your design in hardware.

You only need to purchase a license for the FIR Compiler when you are completely satisfied with its functionality and performance, and want to take your design to production.

After you purchase a license, you can request a license file from the Altera website at **www.altera.com/licensing** and install it on your computer. When you request a license file, Altera emails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.

For more information on OpenCore Plus hardware evaluation, refer to *AN 320: OpenCore Plus Evaluation of Megafunctions*.

OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation supports the following operation modes:

- *Untethered*—the design runs for a limited time.
- *Tethered*—requires a connection between your board and the host computer. If tethered mode is supported by all megafunctions in a design, the device can operate for a longer time or indefinitely.

All megafunctions in a device time-out simultaneously when the most restrictive evaluation time is reached. If there is more than one megafunction in a design, a specific megafunction's time-out behavior might be masked by the time-out behavior of the other megafunctions.

The untethered timeout for the FIR Compiler MegaCore function is one hour; the tethered timeout value is indefinite.

The data output signal is forced to zero when the hardware evaluation time expires.

Design Flows

The FIR Compiler MegaCore® function supports the following design flows:

- **DSP Builder**: Use this flow if you want to create a DSP Builder model that includes a FIR Compiler MegaCore function variation.
- **MegaWizard™ Plug-In Manager**: Use this flow if you would like to create a FIR Compiler MegaCore function variation that you can instantiate manually in your design.

This chapter describes how you can use a FIR Compiler MegaCore function in either of these flows. The parameterization is the same in each flow and is described in Chapter 3, Parameter Settings.

After parameterizing and simulating a design in either of these flows, you can compile the completed design in the Quartus II software.

DSP Builder Flow

Altera's DSP Builder product shortens digital signal processing (DSP) design cycles by helping you create the hardware representation of a DSP design in an algorithm-friendly development environment.

DSP Builder integrates the algorithm development, simulation, and verification capabilities of The MathWorks MATLAB® and Simulink® system-level design tools with Altera Quartus® II software and third-party synthesis and simulation tools. You can combine existing Simulink blocks with Altera DSP Builder blocks and MegaCore function variation blocks to verify system level specifications and perform simulation.

After installing the FIR Compiler, a Simulink symbol for the FIR Compiler appears in the **MegaCore Functions** library of the **Altera DSP Builder Blockset** in the Simulink library browser.

You can use the FIR Compiler in the MATLAB/Simulink environment by performing the following steps:

1. Create a new Simulink model.
2. Select the `FIR Compiler` block from the **MegaCore Functions** library in the Simulink Library Browser, add it to your model, and give the block a unique name.
3. Double-click on the `FIR Compiler` block in your model to display IP Toolbench and click **Step 1: Parameterize** to parameterize a FIR Compiler MegaCore function variation. For an example of how to set parameters for the `FIR Compiler` block, refer to Chapter 3, Parameter Settings.
4. Click **Step 2: Generate** in IP Toolbench to generate your FIR Compiler MegaCore function variation. For information about the generated files, refer to Table 2–1 on page 2–6.

5. Connect your `FIR Compiler` MegaCore function variation block to the other blocks in your model.
6. Simulate the FIR Compiler MegaCore function variation in your DSP Builder model.

For more information about the DSP Builder flow, refer to the *Using MegaCore Functions* chapter in the *DSP Builder User Guide*.

When you are using the DSP Builder flow, device selection, simulation, Quartus II compilation and device programming are all controlled within the DSP Builder environment.

DSP Builder supports integration with SOPC Builder using Avalon® Memory-Mapped (Avalon-MM) master or slave, and Avalon Streaming (Avalon-ST) source or sink interfaces.

For more information about these interface types, refer to the *Avalon Interface Specifications*.

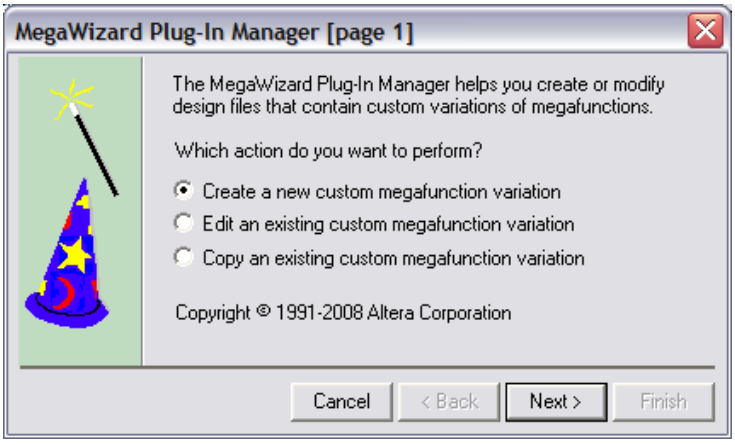
MegaWizard Plug-In Manager Flow

The MegaWizard Plug-in Manager flow allows you to customize a FIR Compiler MegaCore function, and manually integrate the MegaCore function variation into a Quartus II design.

To launch the MegaWizard Plug-in Manager, perform the following steps:

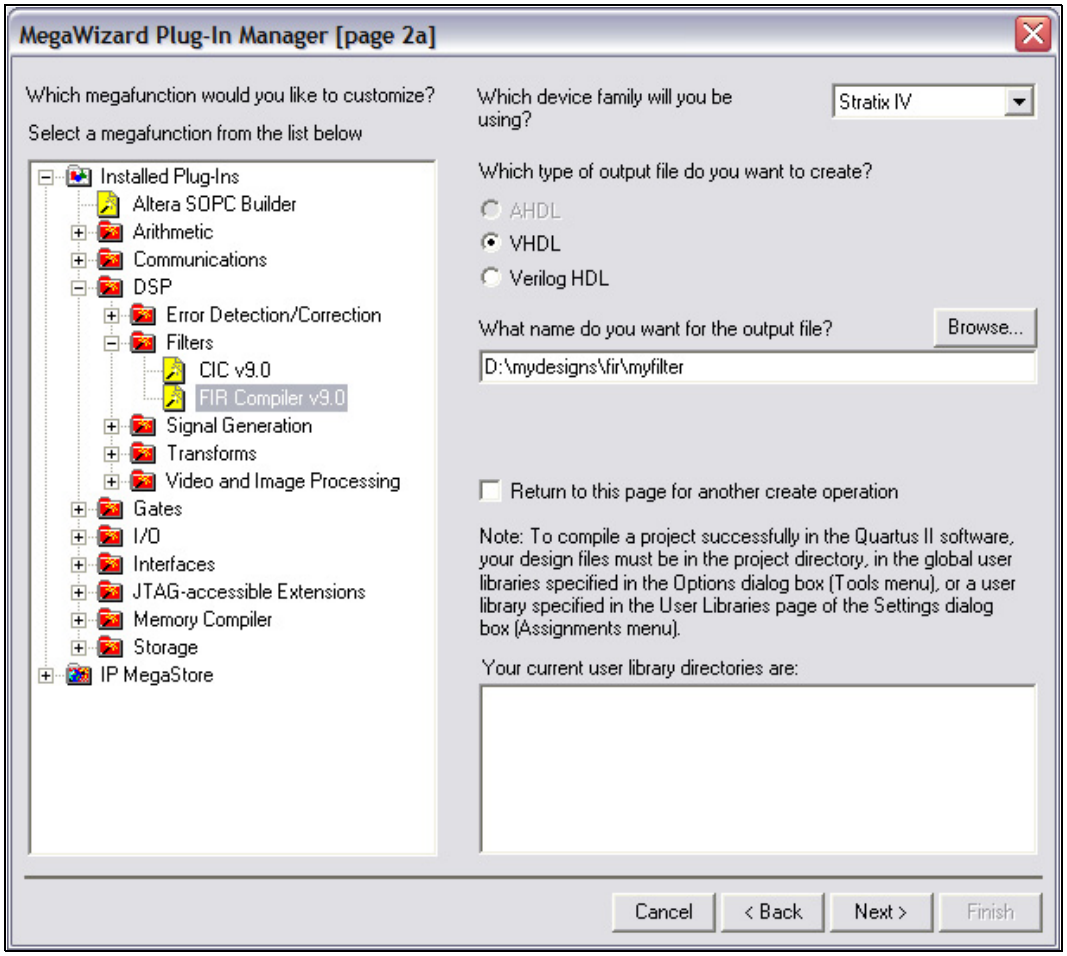
1. Create a new project using the **New Project Wizard** available from the File menu in the Quartus II software.
2. Launch **MegaWizard Plug-in Manager** from the Tools menu, and select the option to create a new custom megafunction variation (Figure 2–1).

Figure 2–1. MegaWizard Plug-In Manager



3. Click **Next** and select **FIR Compiler v9.0** from the **Filters** section in the **Installed Plug-Ins** tab. (Figure 2–2).

Figure 2–2. Selecting the MegaCore Function



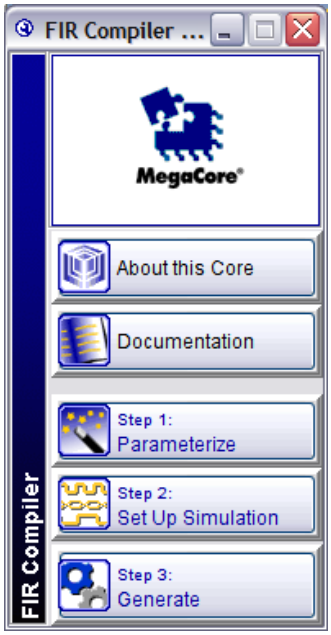
4. Verify that the device family is the same as you specified in the **New Project Wizard**.
5. Select the top-level output file type for your design; the wizard supports VHDL and Verilog HDL.
6. Specify the top level output file name for your MegaCore function variation and click **Next** to launch IP Toolbench (Figure 2–3 on page 2–4).

Parameterize the MegaCore Function

To parameterize your MegaCore function variation, perform the following steps:

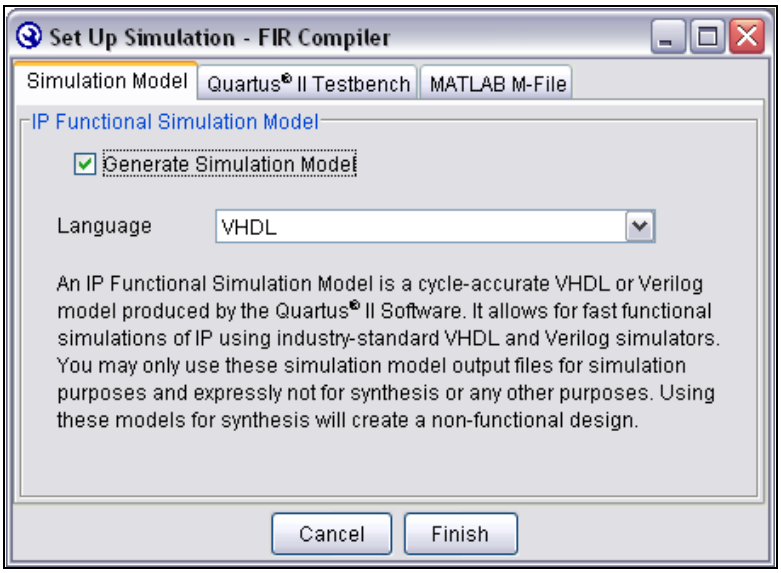
1. Click **Step 1: Parameterize** in IP Toolbench to display the **Parameterize - FIR Compiler** window. Use this interface to specify the required parameters for the MegaCore function variation. For an example of how to set parameters for the FIR Compiler MegaCore function, refer to Chapter 3, Parameter Settings.

Figure 2–3. IP Toolbench—Parameterize



2. Click **Step 2: Setup Simulation** in IP Toolbench to display the **Set Up Simulation - FIR Compiler** page (Figure 2–4).

Figure 2–4. Set Up Simulation



3. Turn on **Generate Simulation Model** to create an IP functional model.

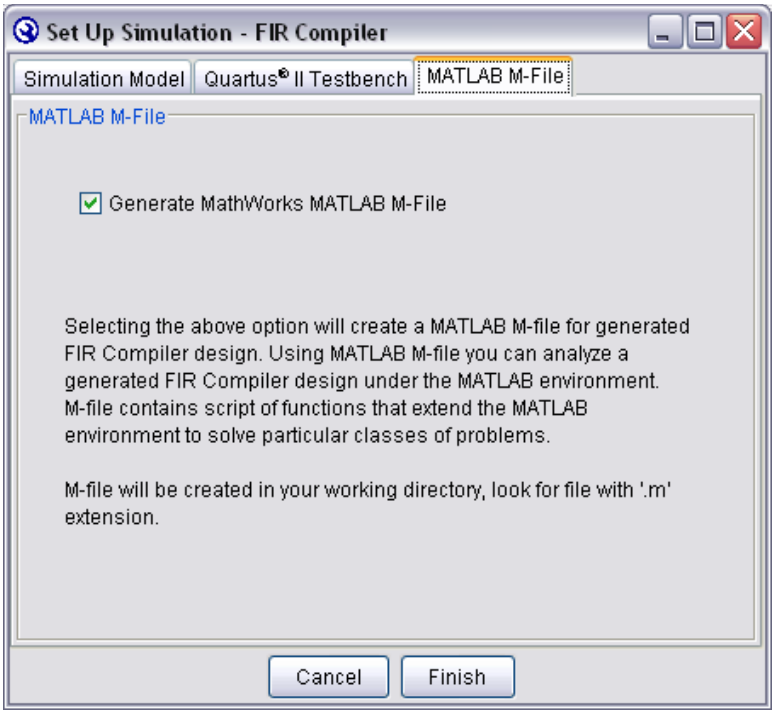
☞ An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus II software.

CAUTION Use the simulation models only for simulation and not for synthesis or any other purposes. Using these models for synthesis creates a non-functional design.

4. Select the required language from the **Language** list.
5. Click the **MATLAB M-File** tab on the **Set Up Simulation** page (Figure 2–5).
6. Turn on the **Generate MathWorks MATLAB M-File** option.

   This option generates a MATLAB m-file script that contains functions you can use to analyze a FIR Compiler design in the MATLAB environment. A testbench is also generated.

Figure 2–5. Create a MATLAB M-File



7. Click **Finish**.

The **Quartus II Testbench** tab contains an option that is not used in this version of the FIR Compiler and should be ignored.

Generate the MegaCore Function

To generate your MegaCore function variation, perform the following steps:

1. Click **Step 3: Generate** in IP Toolbench to generate your MegaCore function variation and supporting files. The generation phase may take several minutes to complete. The generation progress and status is displayed in a report window.

   Figure 2–6 shows the generation report.

Figure 2–6. Generation Report - FIR Compiler MegaCore Function

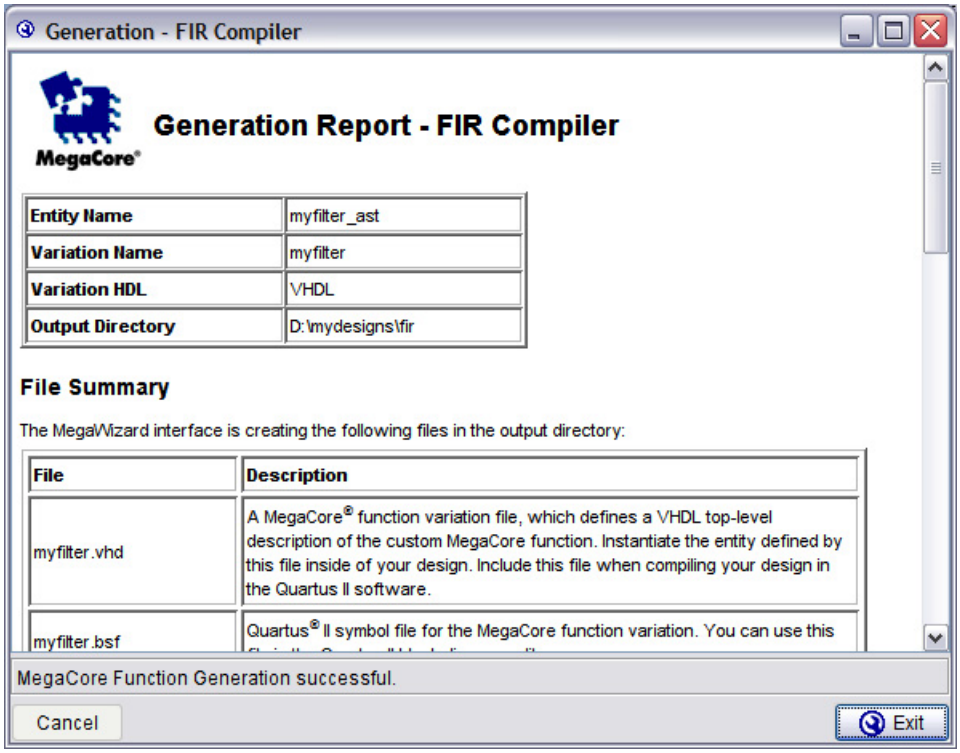


Table 2–1 describes the IP Toolbench-generated files and other files that may be in your project directory. The names and types of files specified in the report vary based on whether you created your design with VHDL or Verilog HDL.

Table 2–1. *Generated Files (Part 1 of 2) (Note 1) & (2)*

Filename	Description
<*entity name*>.vhd	A VHDL wrapper file for the Avalon-ST interface.
<*variation name*>.bsf	A Quartus II block symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.
<*variation name*>.cmp	A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.
<*variation name*>.html	A MegaCore function report file in hypertext markup language format.
<*variation name*>.qip	A single Quartus II IP file is generated that contains all of the assignments and other information required to process your MegaCore function variation in the Quartus II compiler. You are prompted to add this file to the current Quartus II project when you exit from IP Toolbench.

Table 2–1. *Generated Files (Part 2 of 2) (Note 1) & (2)*

Filename	Description
<*variation name*>.vec	Quartus II vector file. This file provides simulation test vectors to be used for simulating the customized FIR MegaCore function variation with the Quartus II software.
<*variation name*>.vhd or .v	A VHDL or Verilog HDL file that defines a VHDL or Verilog HDL top-level description of the custom MegaCore function variation. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
<*variation name*>.vho or **.vo**	A VHDL or Verilog HDL output file that defines the IP functional simulation model.
<*variation name*>_bb.v	A Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
<*variation name*>_coef_in_mlab.txt **<*variation name*>_coef_int.txt**	Text files that provides coefficient inputs for the MATLAB testbench model.
<*variation name*>_coef_n_inv.hex **<*variation name*>_coef_n.hex** **<*variation name*>_zero.hex**	Memory initialization files in INTEL Hex format. These files are required both for simulation with IP functional simulation models and synthesis using the Quartus II software.
<*variation name*>_constraints.tcl	Constraints setting Tcl file for Quartus II synthesis. This file contains the necessary constraints to achieve FIR Filter size and speed.
<*variation name*>_input.txt	This text file provides simulation data for the MATLAB model and the simulation testbench.
<*variation name*>_mlab.m	This MATLAB M-File provides the kernel of the MATLAB simulation model for the customized FIR MegaCore function variation.
<*variation name*>_model.m	This MATLAB M-File provides a MATLAB simulation model for the customized FIR MegaCore function variation.
<*variation name*>_msim.tcl	This Tcl script can be used to simulate the VHDL testbench together with the simulation model of the customized FIR MegaCore function variation.
<*variation name*>_nativelink.tcl	A Tcl script that can be used to assign NativeLink simulation testbench settings to the Quartus II project.
<*variation name*>_param.txt	This text file records all output parameters for customized FIR MegaCore function variation.
<*variation name*>_silent_param.txt	This text file records all input parameters for customized FIR MegaCore function variation.
<*variation name*>_core.vhd **<*variation name*>_st.v** **<*variation name*>_st_s.v** **<*variation name*>_st_u.v** **<*variation name*>_st_wr.v**	Generated FIR Filter netlists. These files are required for Quartus II synthesis and are added to your current Quartus II project.
tb_<*variation name*>.vhd	This VHDL file provides a testbench for the customized FIR MegaCore function variation.

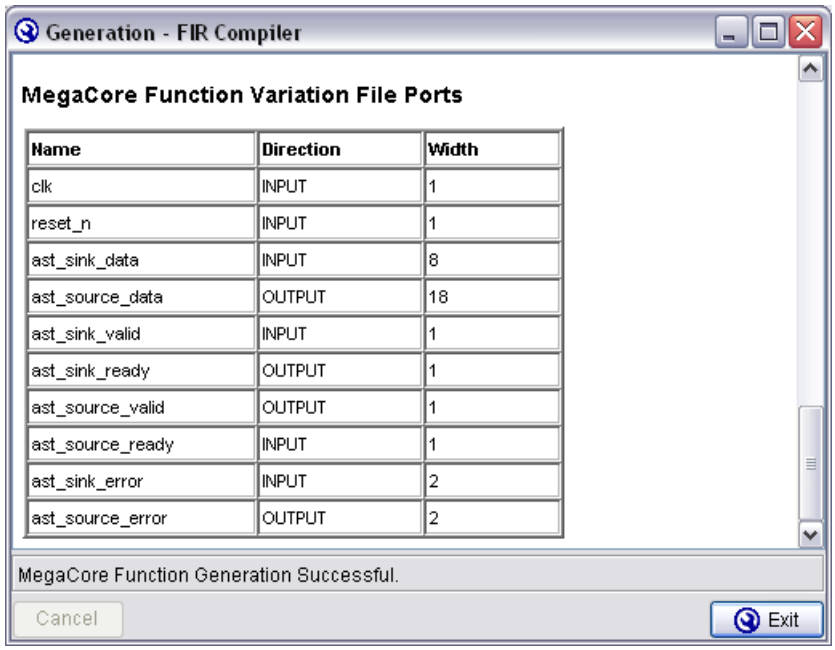
Notes to Table 2–1

(1) <*variation name*> is a prefix variation name supplied automatically by IP Toolbench.

(2) The <*entity name*> prefix is added automatically. The VHDL code for each MegaCore instance is generated dynamically when you click **Finish** so that the <*entity name*> is different for every instance. It is generated from the <*variation name*> by appending **_ast**.

The generation report also lists the ports defined in the MegaCore function variation file (Figure 2–7). For a full description of the signals supported on external ports for your MegaCore function variation, refer to Table 4–3 on page 4–16.

Figure 2–7. Port Lists in the Generation Report



2. After you review the generation report, click **Exit** to close IP Toolbench. Then click **Yes** on the **Quartus II IP Files** prompt to add the .**qip** file describing your custom MegaCore function variation to the current Quartus II project.

Simulate the Design

To simulate your design in Verilog HDL or VHDL, use the IP functional simulation models generated by IP Toolbench.

The IP functional simulation model is the .**vo** or .**vho** file (located in your design directory) generated as specified in Step 1 on page 2–3.

For more information on IP functional simulation models, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook*.

Simulating in Model*Sim*

A Tcl script (**<*variation name*>_msim.tcl**) is also generated which can be used to load the VHDL testbench into the ModelSim simulator.

This script uses the file **<*variation name*>_input.txt** to provide input data to the FIR filter. The output from the simulation is stored in a file **<*variation name*>_output.txt**.

Simulating in MATLAB

To simulate in a MATLAB environment, run the **<*variation_name*>_model.m** testbench m-script, which also is located in your design directory. This script also uses the file **<*variation name*>_input.txt** to provide input data. The output from the MATLAB simulation is stored in the file **<*variation name*>_model_output.txt**.

For MCV decimation filters, the **<*variation name*>_model_output_full.txt** file is generated to display all the phases of the filter. You can compare this file with the **<*variation name*>_output.txt** file to understand which phase the output belongs. For all other architectures, decimation filters provide the *N*th phase where *N* is the decimation factor.

Simulating in Third-Party Simulation Tools Using NativeLink

You can perform a simulation in a third-party simulation tool from within the Quartus II software, using NativeLink.

The Tcl script file **<*variation name*>_nativelink.tcl** can be used to assign default NativeLink testbench settings to the Quartus II project.

To perform a simulation in the Quartus II software using NativeLink, perform the following steps:

1. Create a custom MegaCore function variation as described earlier in this chapter but ensure you specify your variation name to match the Quartus II project name.
2. Verify that the absolute path to your third-party EDA tool is set in the **Options** page under the Tools menu in the Quartus II software.
3. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
4. On the Tools menu, click **Tcl scripts**. In the **Tcl Scripts** dialog box, select **<*variation name*>_nativelink.tcl** and click **Run**. Check for a message confirming that the Tcl script was successfully loaded.
5. On the Assignments menu, click **Settings**, expand **EDA Tool Settings**, and select **Simulation**. Select a simulator under **Tool name** then in **NativeLink Settings**, select **Compile test bench** and click **Test Benches**.
6. On the Tools menu, point to **EDA Simulation Tool** and click **Run EDA RTL Simulation**.

   The Quartus II software selects the simulator, and compiles the Altera libraries, design files, and testbenches. The testbench runs and the waveform window shows the design signals for analysis.

For more information, refer to the *Simulating Altera IP in Third-Party Simulation Tools* chapter in volume 3 of the *Quartus II Handbook*.

Compile the Design and Program a Device

You can use the Quartus II software to compile your design.

After you have compiled your design, program your targeted Altera device and verify your design in hardware.

For instructions on compiling and programming your design, and more information about the MegaWizard Plug-In Manager flow, refer to the Quartus II Help.

3. Parameter Settings

This chapter gives an example of how to parameterize a FIR Compiler MegaCore® function and describes the available parameters.

The **Parameterize - FIR Compiler** pages provide the same options whether they have been opened from the DSP Builder or MegaWizard Plug-In Manager flow.

For information about opening the parameterization pages, refer to "Design Flows" on page 2–1.

The user interface only allows you to select legal combinations of parameters, and warns you of any invalid configurations.

Specifying the Coefficients

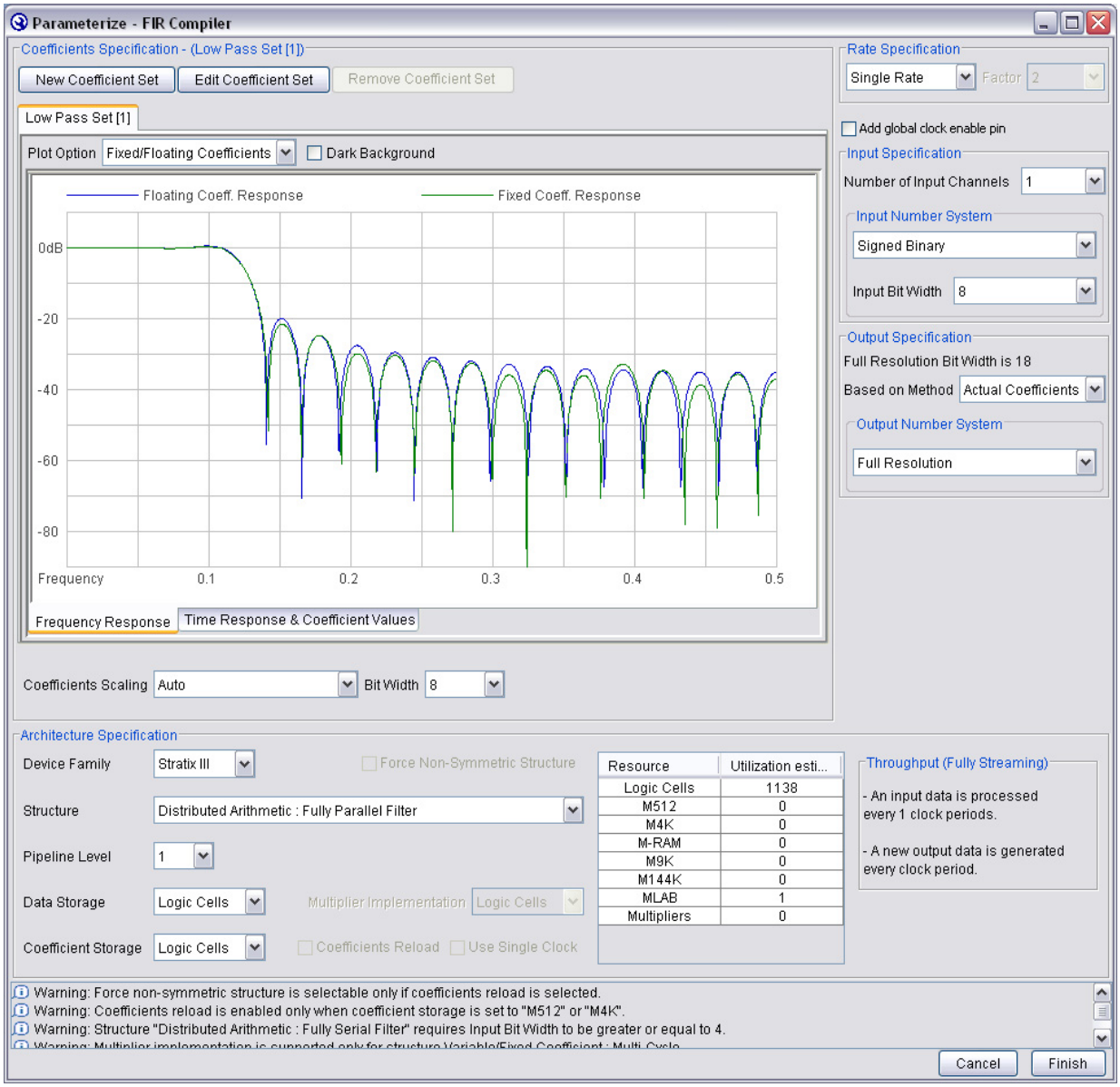
A FIR filter is defined by its coefficients. The FIR Compiler provides the following options for obtaining coefficients:

- You can use the FIR Compiler to generate coefficients. The coefficient generator supports single rate, interpolation, and decimation rate specification filter types. For information about generating coefficients for these filter types, refer to "Using the FIR Compiler Coefficient Generator" on page 3–2.
- You can load coefficients from a file. For example, you can create the coefficients in another application such as MATLAB, SPW, or a user-created program, save them to a file, and import them into the FIR Compiler. For more information, refer to "Loading Coefficients from a File" on page 3–6.)

Figure 3–1 on page 3–2 shows the **Parameterize - FIR Compiler** page.

You can click **New Coefficient Set** on this page to define or load new coefficients. Alternatively, or you can click **Edit Coefficient Set** to edit the default coefficient set or **Remove Coefficient Set** to clear the currently loaded coefficients.

Figure 3–1. IP Toolbench Parameterize Page



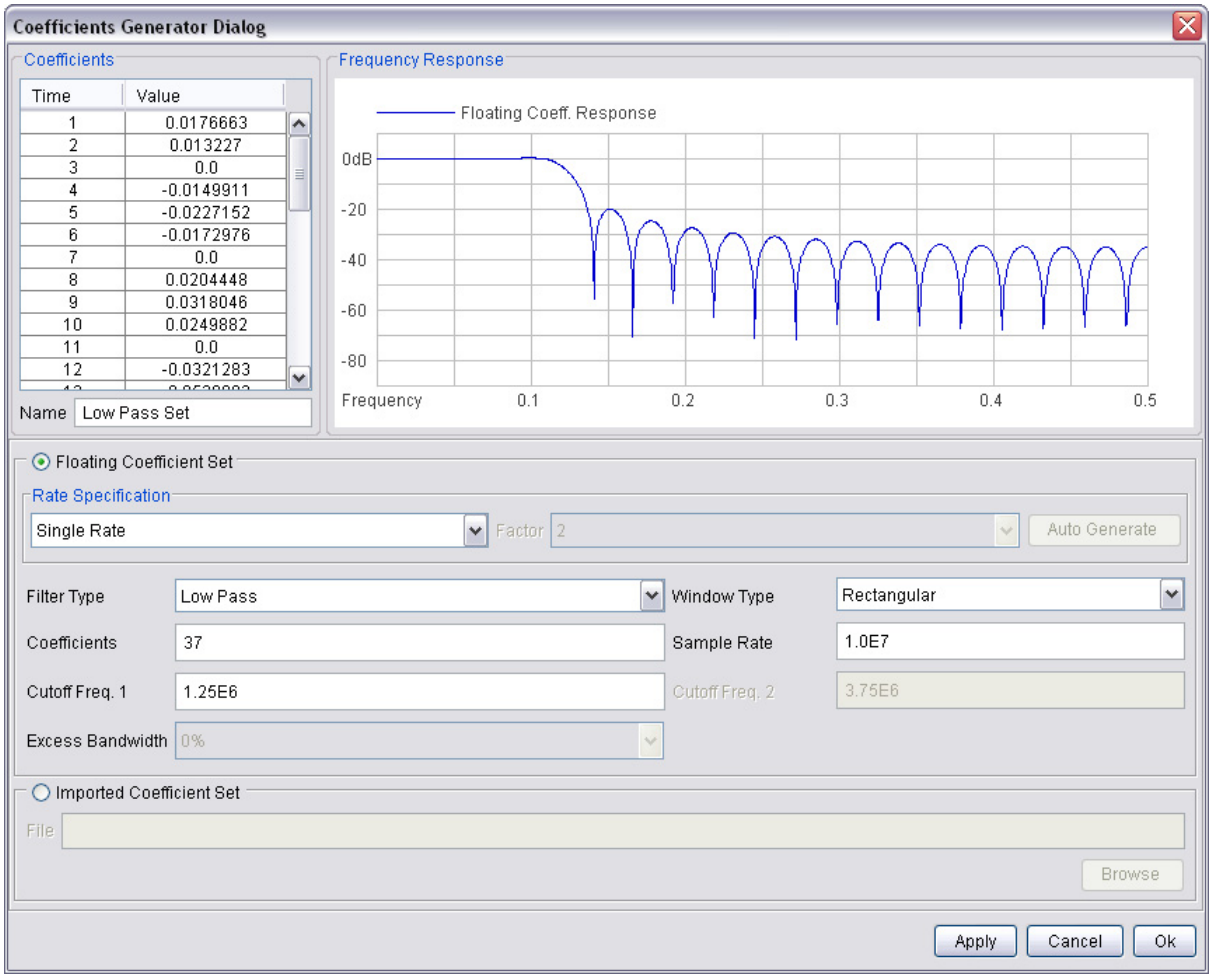
Using the FIR Compiler Coefficient Generator

1. Click **New Coefficient Set** in the **Parameterize - FIR Compiler** page to open the **Coefficients Generator** dialog box.

   You can use this dialog box to specify parameters for the coefficients, including the filter type, window type, sample rate, and excess bandwidth (for use with cosine filters).

   Figure 3–2 on page 3–3 shows the default values for a low pass filter.

Figure 3–2. Coefficients Generator Dialog Box Showing Default Low Pass Filter Parameters



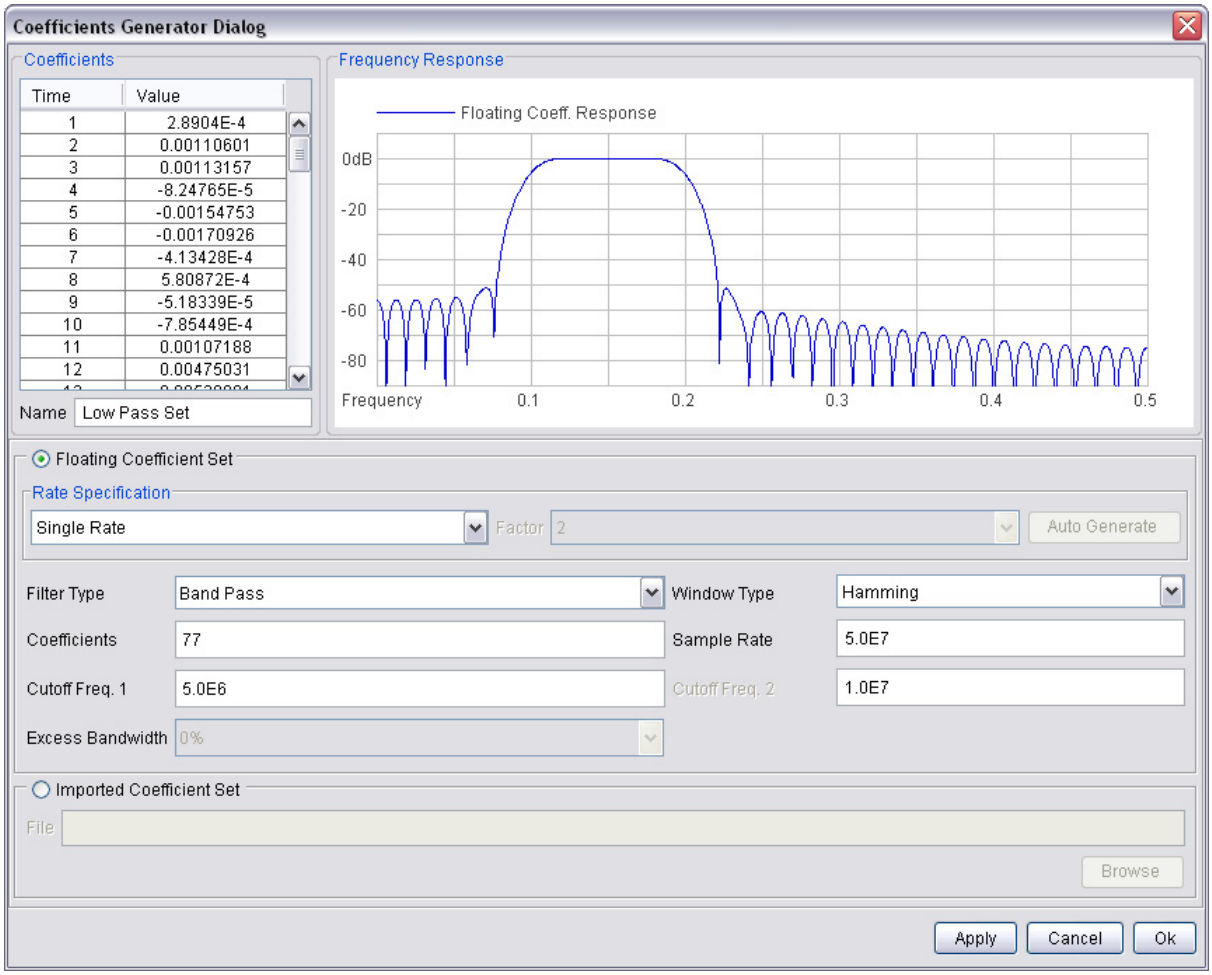
2. To generate the coefficients for a simple parallel filter, use the **Coefficients Generator** dialog box to make the settings listed in Table 3–1.

Table 3–1. Coefficients Generator Parameter Settings for a Simple Parallel Filter

Parameter	Value
Rate Specification	Single Rate
Filter Type	Band Pass
Coefficients	77
Cutoff Freq. 1	5e+006
Window Type	Hamming
Sample Rate:	50e+006
Cutoff Freq. 2	10e+006

3. After making your settings, click **Apply**. The dialog box displays the frequency response of the filter in blue and also displays a list of the actual coefficient values. (Figure 3–3).

Figure 3–3. Parallel FIR Filter Coefficient Parameters for Band Pass Filter



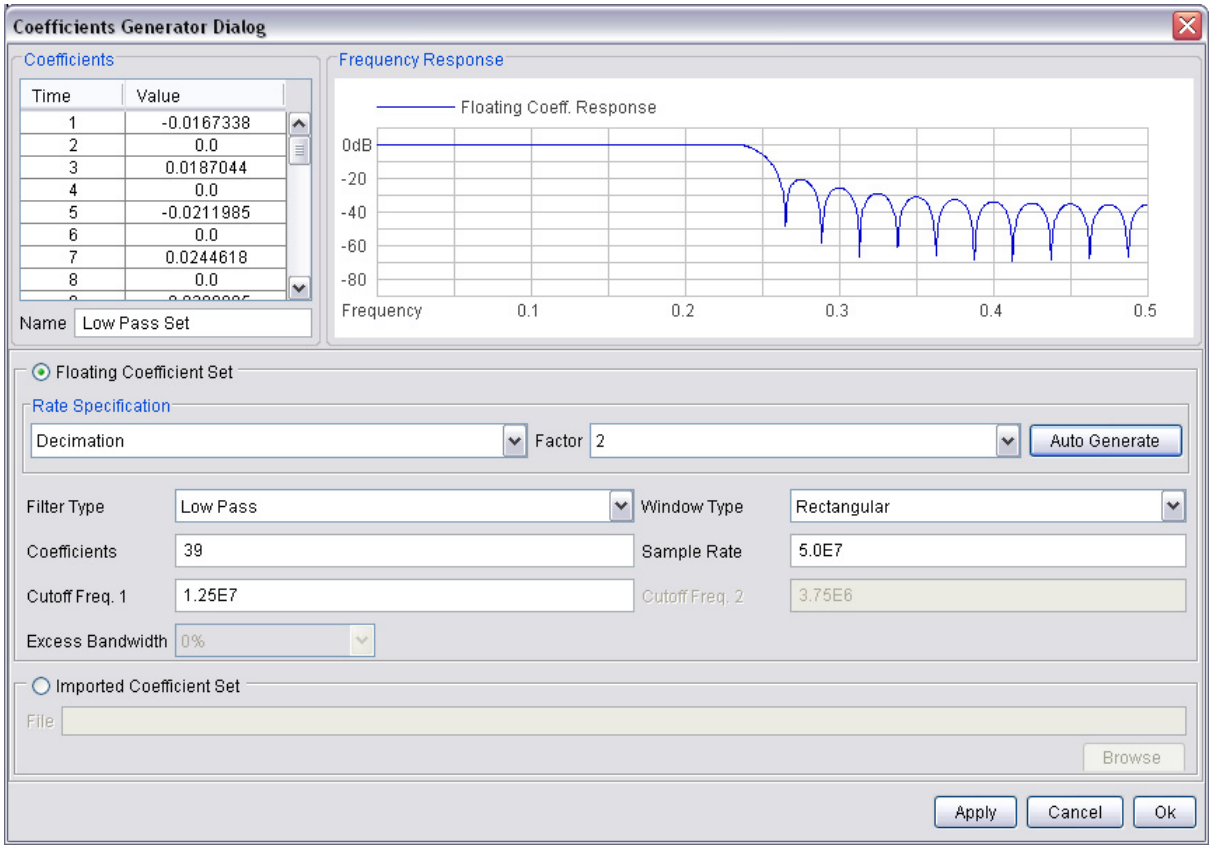
4. To generate floating-point coefficients for interpolation or decimation rate filters, select **Interpolation** or **Decimation** and the required **Factor** from the **Rate Specification** drop-down boxes.

   When you click **Auto Generate**, IP Toolbench generates coefficients for a low-pass filter with a cutoff frequency based on the specified rate.

   Figure 3–4 on page 3–5 shows a decimation filter. The cut-off frequency is ¼ of the sampling rate and results in a half-band coefficient set.

   For an explanation of interpolation and decimation, refer to "Interpolation & Decimation" on page 4–8.

Figure 3–4. Low-Pass Filter Results for an Interpolation Filter

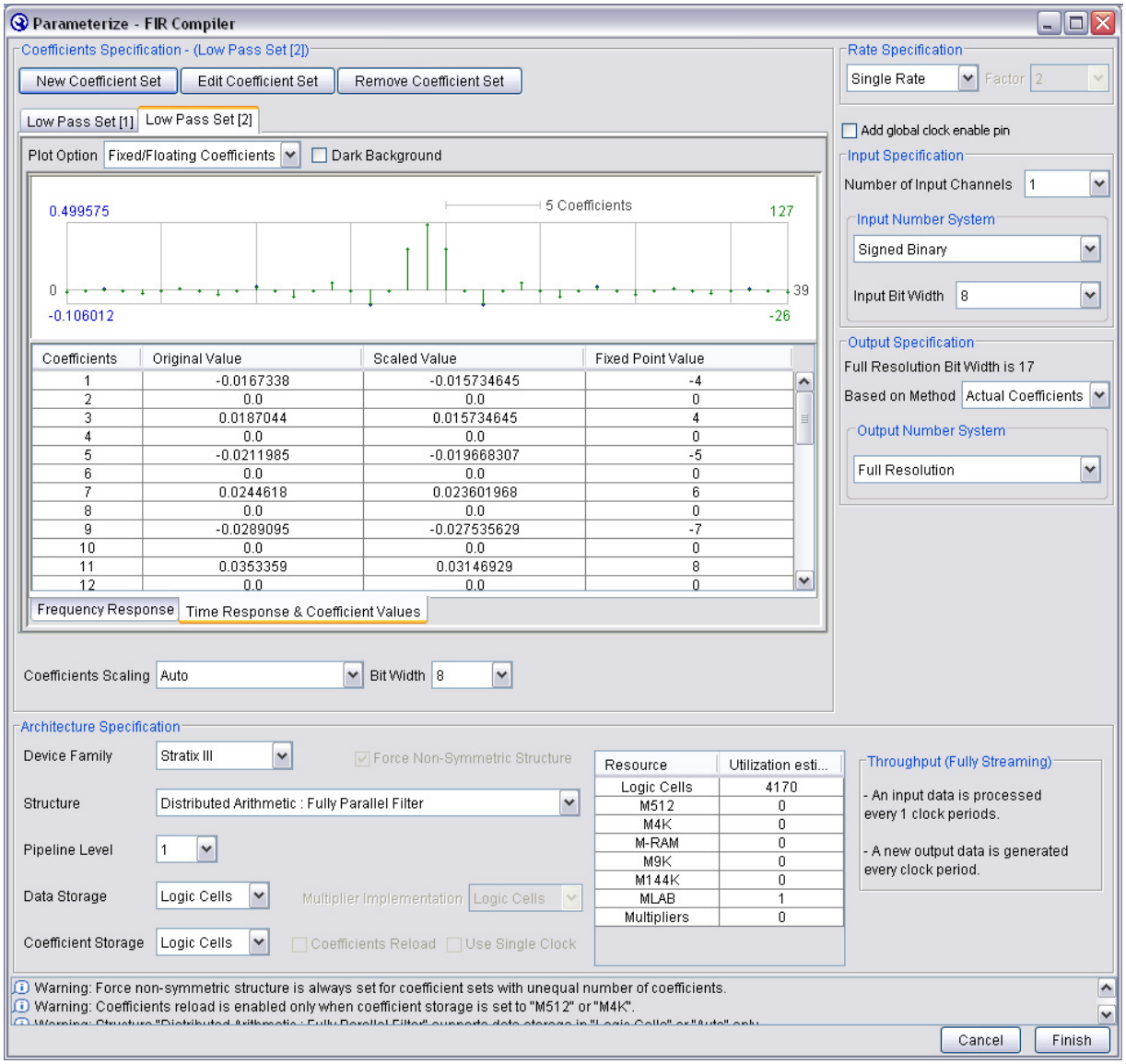


5. Click **OK** when you have finished making the parameter settings.

   The **Parameterize - FIR Compiler** page (Figure 3–1 on page 3–2) is updated to display the frequency response of the floating coefficients in blue and the frequency response of the fixed coefficients in green.

   You can click the **Time Response & Coefficient Values** tab to list the coefficients as shown in Figure 3–5 on page 3–6.

Figure 3–5. IP Toolbench Parameterize Page, Time Response and Coefficient Values Tab

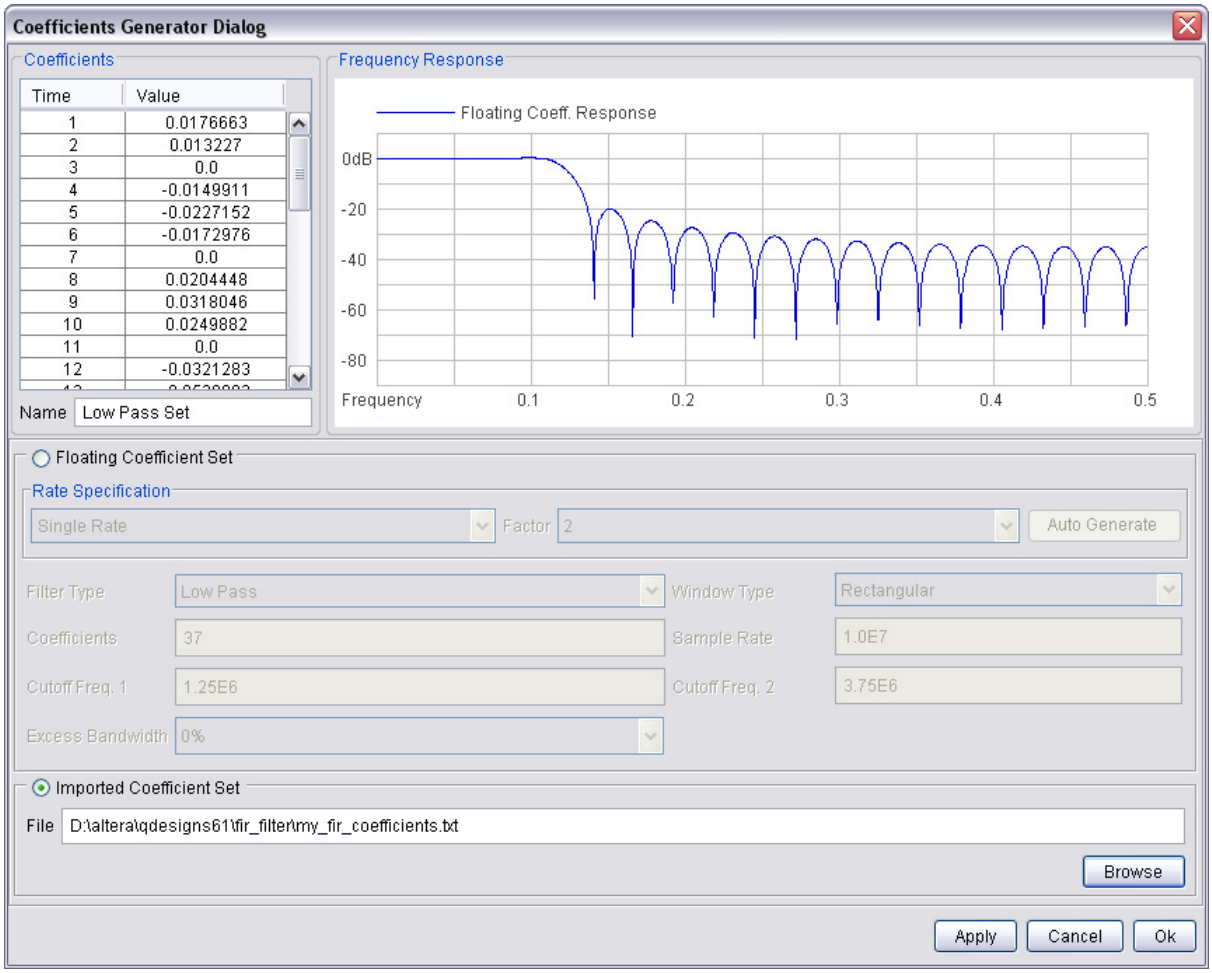


Loading Coefficients from a File

To load a coefficient set from a file, perform the following steps:

1. Click **New Coefficient Set** in the **Parameterize - FIR Compiler** page (Figure 3–1 on page 3–2); then select **Imported Coefficient Set** in the **Coefficients Generator** dialog box (Figure 3–6 on page 3–7).

Figure 3–6. Importing a Coefficient Set



Note to Figure 3–6:

(1) The radio buttons for the Floating Coefficient Set and Imported Coefficient Set parameters are linked together; selecting one disables the other.

2. Browse in the file system for the file you want to use, and click **Open**.

   Your coefficient file should have each coefficient on a separate line and no carriage returns at the end of the file. You can use floating-point or fixed-point numbers, as well as scientific notation.

   Do not insert additional carriage returns at the end of the file. The FIR Compiler interprets each carriage return as an extra coefficient with the value of the most recent past coefficient. The file should have a minimum of five non-zero coefficients.

3. Click **OK** to import your coefficient set.

Analyzing the Coefficients

The FIR Compiler contains a coefficient analysis tool, which you can use to create sets of coefficients and perform actions on each set.

Some actions, such as scaling, apply to all sets. Other actions, such as recreating, reloading, or deleting, apply to the set you are currently viewing.

The FIR Compiler supports up to 16 sets of coefficients. You can switch between sets using the coefficient tabs in the **Parameterize - FIR Compiler** page. (The coefficient sets are numbered, for example, **Low Pass Set 1**, **Low Pass Set 2** and so on.)

When you select a set, the frequency response of the floating-point coefficients is displayed in blue, and the frequency response of the fixed-point coefficients in green. You can also view the actual coefficient values. by clicking the **Time Response & Coefficient Values** tab.

The FIR Compiler supports two's complement, signed binary fractional notation, which allows you to monitor which bits are preserved and which bits are removed during filtering. A signed binary fractional number has the format:

<*sign*> <*integer bits*>.<*fractional bits*>

A signed binary fractional number is interpreted as shown below:

<*sign*> <x_1 *integer bits*>.<y_1 *fractional bits*>	Original input data
<*sign*> <x_2 *integer bits*>.<y_2 *fractional bits*>	Original coefficient data
<*sign*> <*i integer bits*>.<$y_1 + y_2$ *fractional bits*>	Full precision after FIR calculation
<*sign*> <x_3 *integer bits*>.<y_3 *fractional bits*>	Output data after limiting precision

where $i = \text{ceil}(\log_2(\textit{number of coefficients})) + x_1 + x_2$

If, for example, the number has 3 fractional bits and 4 integer bits plus a sign bit, the entire 8-bit integer number is divided by 8, which yields a number with a binary fractional component.

DSP Builder incorporates the sign bit as part of the integer bits. Thus, if you are using the FIR filter in a DSP Builder design, DSP builder will recognize the sign bit as an additional integer bit.

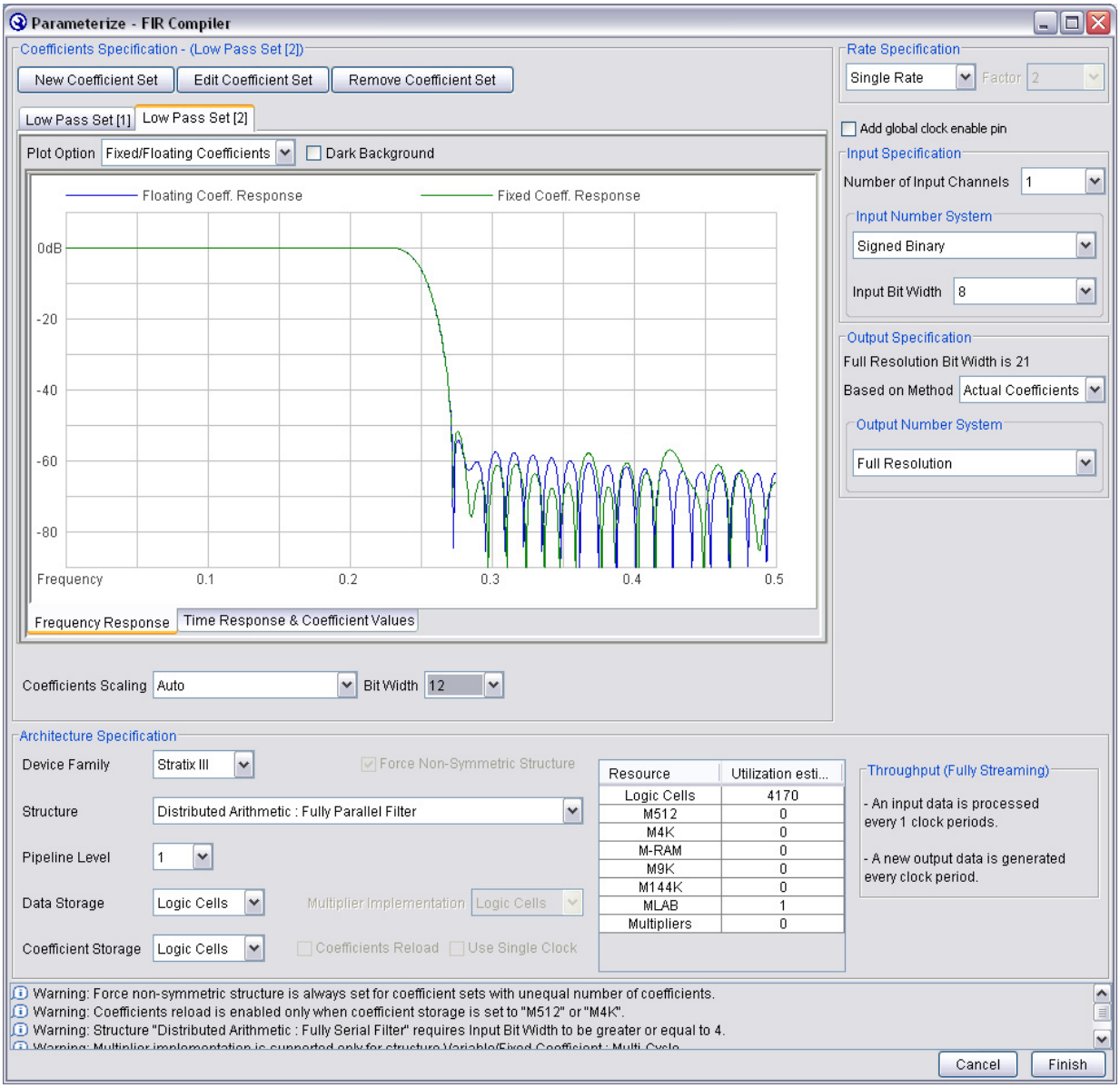
When converted to decimal numbers, certain fractions have an infinite number of binary bits. For example, converting 1/3 to a decimal number yields 0.333*n* with *n* representing an infinite number of 3s. Similarly, numbers such as 1/10 cannot be represented in a finite number of binary digits with full precision. If you use signed binary fractional notation, the FIR Compiler uses the fractional number that most closely matches the original number for the number of bits of precision you select.

For this tutorial, select **Auto** for **Coefficients Scaling** and **12** for the **Coefficient Bit Width**.

Auto scaling (without the power of two option) provides the maximum signal-to-noise ratio. All other scaling factors such as Signed Binary Fractional can result in a loss of effective bits (that is, where each effective bit provides 6dB of SNR).

Figure 3–7 shows the result after you have made the selections. Note that the side lobes of the fixed-point frequency response decrease when you change the bit width from 8 (the default) to 12.

Figure 3–7. Analyzing the Coefficients



Specify the Input and Output Specifications

You can specify the **Number of Input Channels** (that is, the number of data streams that generate an output for each stream) and the **Input Number System** in the **Parameterize - FIR Compiler** page (Figure 3–7).

The FIR Compiler calculates how many bits your filter requires for full resolution using two methods: actual coefficients or the coefficient bit widths. These parameters define the maximum positive and negative output values. Select either **Bit Width Only** or **Actual Coefficients** in the **Output Specification** drop-down box. The FIR Compiler will extrapolate the number of bits required to represent that range of values. For full precision, you must use this number of bits in your system.

If your filter has coefficient reloading or multiple sets of coefficients, you must select **Bit Width Only**.

You can use full or limited precision for the filtered output (`out`). To use full precision, leave the **Output Number System** set to Full Resolution (default). To limit the precision, select **Custom Resolution** or **Signed Binary Fractional** from the drop down box.

When the **Output Number System** is set to **Custom Resolution**, you can choose to truncate or saturate the most significant bit (MSB) and to truncate or round the least significant bit (LSB). Saturation, truncation, and rounding are non-linear operations.

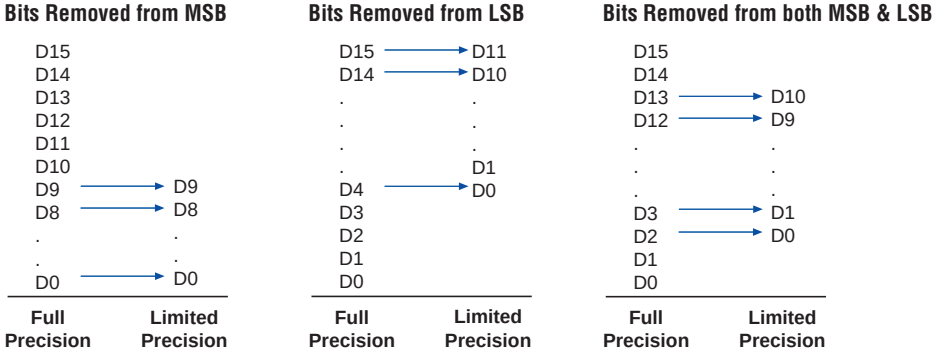
Table 3–2 shows the options for limiting the precision of your filter.

Table 3–2. Options for Limiting Precision

Bit Range	Option	Result
MSB	Truncate	In truncation, the filter disregards specified bits. (Figure 3–8).
	Saturate	In saturation, if the filtered output is greater than the maximum positive or negative value that can be represented, the output is forced (or saturated) to the maximum positive or negative value.
LSB	Truncate	Same process as for MSB.
	Round	The output is rounded away from zero.

Figure 3–8 shows an example of removing bits from the MSB and LSB.

Figure 3–8. Removing Bits from the MSB and LSB



Alternatively, you can select **Signed Binary Fractional** notation and specify the number of bits to keep. The FIR Compiler displays how many bits are removed.

When adjusting the input and output specification, follow these tips:

- Truncating from the MSB reduces logic resources more than saturation.
- The Number of Input Channels option is useful for designs such as modulators and demodulators, which have I and Q channels. If you are designing this type of application, select 2 input channels. This tutorial uses the default settings.

Specify the Architecture Specification

You are now ready to select the architecture parameters from the lower half of the **Parameterize - FIR Compiler** page.

The FIR Compiler supports several filter structures, including:

- Variable/Fixed coefficient: Multicycle
- Distributed arithmetic: Fully Parallel Filter
- Distributed arithmetic: Fully Serial Filter
- Distributed arithmetic: Multibit Serial Filter

For maximum clock speed, select the Distributed Arithmetic: Fully Serial Filter structure. (For Stratix, Stratix II, Stratix III, or Stratix IV devices, using smaller memory resources for coefficient and data storage is faster than using larger memory resources.) For maximum throughput, select the Distributed Arithmetic: Fully Parallel structure.

When reloading coefficients, a multicycle variable FIR filter structure has a short reloading time compared to a fixed FIR filter. Additionally, smaller memory blocks have a shorter reloading time than larger memory blocks.

Table 3–3 describes the relative trade-offs for the different architecture options.

Table 3–3. Architecture Trade-Offs

Technology	Option	Area	Speed (Data Throughput)
Distributed arithmetic	Fully parallel	Large area	Creates a fast filter: 140 to over 300 MSPS throughput with pipelining in Stratix II devices.
Distributed arithmetic	Fully serial	Small area	Requires multiple clock cycles for a single computation.
Distributed arithmetic	Multibit serial	Medium area	Uses several serial units to increase throughput.This results in greater throughput than fully serial, but less throughput than fully parallel.
DSP block multiplier	Multicycle	Area depends on the number of calculation cycles selected (area increases as the number of calculation cycles increases)	Data throughput increases as the number of calculation cycles decreases. This architecture takes advantage of Stratix, Stratix II, Stratix III, or Stratix IV DSP Blocks, and Cyclone II Multipliers.
Available option for all architectures	Pipelining	Creates a higher performance filter with an area increase.	Increases throughput with additional latency and size increase.

For more information about the filter architectures and how they operate, refer to "FIR Compiler" on page 4–1.

Table 3–4, Table 3–5, Table 3–6, and Table 3–7 describe the FIR Compiler options that are available for each architecture.

Table 3–4. Multicycle Filter Architecture *(Note 1)*

Parameter	Description
Clocks to Compute	Specifies the number of clock cycles required to compute a result. Using more clock cycles to compute a result reduces the filter resource usage. The number of multipliers the filter uses is equal to the number of taps divided by the number of clock cycles to compute the result.
Data Storage	Specifies the device resources used for data storage. You can select **Logic Cells**, **M512**, **M4K**, **M-RAM**, **MLAB**, **M9K**, **M144K**, or **Auto**. If you select **Auto**, the Quartus II software may store data in logic cells or memory, depending on the resources in the selected device, the size of the data storage, the number of clock cycles to compute a result, and the number of input channels. The option list changes depending on which device you select and the number of clock cycles to compute a result. Choosing embedded memory reduces logic cell usage and may increase the speed of the filter.
Coefficient Storage	Specifies the device resources used for coefficient storage. You can select **Logic Cells**, **M512**, **M4K**, **MLAB**, **M9K**, or **Auto**. If you select **Auto**, the Quartus II software automatically selects the most appropriate memory block size for the selected device. The option list changes depending on which device you select and the number of clock cycles to compute a result. Choosing embedded memory reduces logic cell usage and may increase the speed of the filter.
Multiplier Implementation	Specify the device resources used to implement the multiplier. You can select **Logic Cells**, **DSP Blocks**, or **Auto**. If you select **Auto**, the Quartus II software turns on the **DSP Block Balancing** logic option. Using embedded DSP blocks results in a smaller and faster design in a device with enough DSP blocks for all multipliers. The most efficient use of DSP block is for 9×9 (in groups of 8) or 18×18 (in groups of 4) multipliers.
Force Non-Symmetric Structure	If you want to create a design that uses both symmetric and non-symmetric coefficients, turn on this option. Non-symmetric architectures may use more resources.
Coefficients Reload	Turn on this option to allow coefficient reloading.
Pipeline Level	When you turn on this option, FIR Compiler creates a higher performance filter that uses more device resources.
Use Single Clock	Use this option when creating designs with DSP Builder. This option is only available when **Coefficients Reload** is on and **M512**, **M4K**, **MLAB** or **M9K** is specified in **Coefficient Storage**. This option ties the `coef_clk_in` and `clk` signals together.

Note to Table 3–4:

(1) When the input data is unsigned, the input data bit width should be greater than or equal to one. When the input data is signed, the input data bit width should be greater than or equal to two.

Table 3–5. Fully Serial Filter Architecture *(Note 1)*

Parameter	Description
Data Storage	Specifies the device resources used for data storage. You can select **Logic Cells**, **M512**, **M4K**, **M-RAM**, **MLAB**, **M9K**, **M144K**, or **Auto**. If you select **Auto**, the Quartus II software may store data in logic cells or memory, depending on the resources in the selected device, the size of the data storage, and the number of input channels.
Coefficient Storage	Specifies the device resources used for coefficient storage. You can select **Logic Cells**, **M512**, **M4K**, **MLAB**, **M9K**, or **Auto**. If you select **Auto**, the Quartus II software automatically selects the most appropriate memory block size for the selected device. The option list changes depending on which device you select. Selecting embedded memory reduces logic cell usage and may increase the speed of the filter.
Force Non-Symmetric Structure	If you want to create a design that uses both symmetric and non-symmetric coefficients, turn on this option. Symmetric algorithms require an extra clock cycle per calculation cycle, which leads to lower throughput.
Coefficients Reload	If you want to change coefficients, turn on this option. This option is available when you choose to store coefficients in embedded memory. Selecting this option increases resource usage, turns off several optimization schemes, and adds additional input ports to the filter.
Pipeline Level	Creates a higher performance filter with a resource usage increase.
Use Single Clock	Use this option when creating designs with DSP Builder. This option is only available when **Coefficients Reload** is selected and **M512**, **M4K**, **MLAB** or **M9K** is specified in **Coefficient Storage**. This option ties the `coef_in_clk` and `clk` signals together.

Note to Table 3–5:

(1) The input data bit width should be greater than or equal to four.

Table 3–6. Multibit Serial Filter Architecture (Part 1 of 2) *(Note 1)*

Parameters	Description
Number of Serial Units	Specifies the number of serial units needed to make the filter. You can select **2**, **3**, or **4**. The calculation cycles of each result are reduced to one nth of the corresponding serial filter, where n is the number of serial units. Correspondingly, there is an increase in resource utilization.
Data Storage	Specifies the device resources used for data storage. You can select **Logic Cells**, **M512**, **M4K**, **M-RAM**, **MLAB**, **M9K**, **M144K**, or **Auto. If** you select **Auto**, the Quartus II software selects the type of embedded memory blocks, depending on the resources in the selected device, the size of the data storage, the number of clock cycles to compute a result, and the number of input channels. The option list changes depending on which device you select and whether you select multirate (interpolation or decimation). Choosing embedded memory reduces logic cell usage and may increase the speed of the filter.
Coefficient Storage	Specifies the device resources used for coefficient storage. You can select **Logic Cells**, **M512**, **M4K**, **MLAB**, **M9K**, or **Auto**. If you select **Auto**, the Quartus II software automatically selects the most appropriate memory block size for the selected device. The option list changes depending on which device you select. Selecting embedded memory reduces logic cell usage and may increase the speed of the filter.
Force Non-Symmetric Structure	If you want to create a design that uses both symmetric and non-symmetric coefficients, turn on this option. Symmetric algorithms require an extra clock cycle per calculation cycle, which leads to lower throughput.

Table 3–6. Multibit Serial Filter Architecture (Part 2 of 2) *(Note 1)*

Parameters	Description
Coefficient Reload	If you want to change coefficients, turn on this option. This option is available when you choose to store coefficients in embedded memory. Selecting this option increases resource usage, turns off several optimization schemes, and adds additional input ports to the filter.
Pipeline Level	Creates a higher performance filter with a resource usage increase.
Use Single Clock	Use this option when creating designs with DSP Builder. This option is only available when **Coefficients Reload** is selected and **M512**, **M4K**, **MLAB** or **M9K** is specified in **Coefficient Storage**. This option ties the `coef_clk_in` and `clk` signals together.

Note to Table 3–6:

(1) The bit width of input data should divide evenly by the number of serial units and result of division must be greater than or equal to four.

Table 3–7. Fully Parallel Filter Architecture *(Note 1)*

Parameters	Description
Data Storage	Specifies the device resources used for data storage. You can select **Logic Cells** or **Auto**. If you select **Auto**, the Quartus II software may store data in logic cells or memory, depending on the resources in the selected device, the size of the data storage, and the number of input channels.
Coefficient Storage	Specifies the device resources used for coefficient storage. You can select **Logic Cells**, **M512**, **M4K**, **MLAB**, **M9K**, or **Auto**. If you select **Auto**, the Quartus II software automatically selects the most appropriate memory block size for the selected device. The option list changes depending on which device you select. Selecting embedded memory reduces logic cell usage and may increase the speed of the filter.
Force Non-Symmetric Structure	If you want to create a design that uses both symmetric and non-symmetric coefficients, turn on this option. Non-symmetric architectures may use more resources. This option is available when coefficients are stored in the embedded memory.
Coefficient Reload	If you want to change coefficients, turn on this option. This option is available when you choose to store coefficients in embedded memory. Selecting this option increases resource usage, turns off several optimization schemes, and adds additional input ports to the filter.
Pipeline Level	Creates a higher performance filter with a resource usage increase.
Use Single Clock	Use this option when creating designs with DSP Builder. This option is only available when **Coefficients Reload** is selected and **M512**, **M4K**, **MLAB** or **M9K** is specified in **Coefficient Storage**. This option ties the `coef_clk_in` and `clk` signals together.

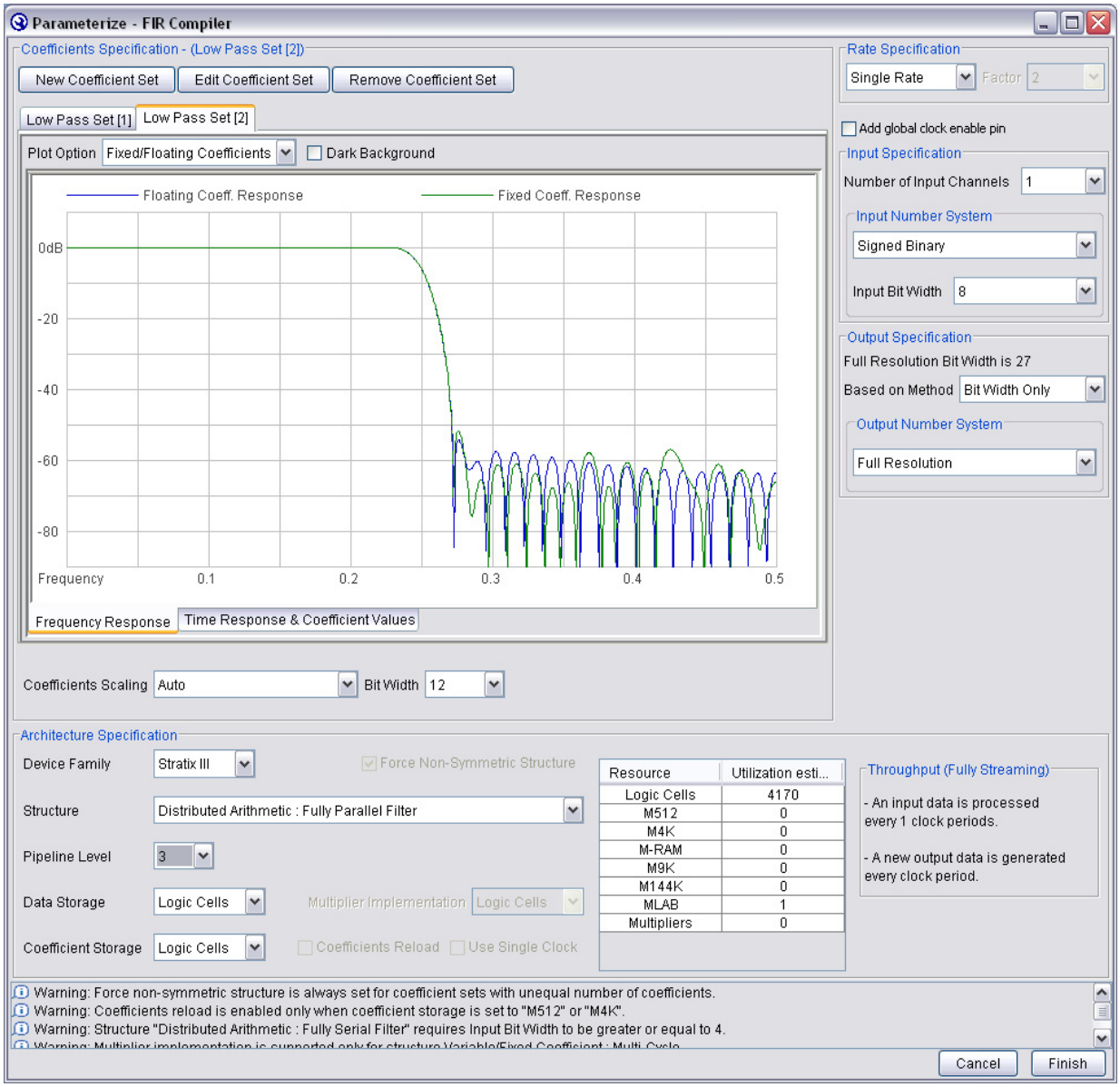
Note to Figure 3–6:

(1) When input data is unsigned, the input data bit width should be greater than or equal to one. When input data is signed, the input data bit width should be greater than or equal to two.

1. For this tutorial, select **Distributed Arithmetic: Fully Parallel Filter** structure with a pipeline level of **3**.

   Although these settings create a filter that uses a large number of logic cells, increasing the pipeline level to 3 decreases the number of clock cycles to one, thereby greatly increasing system performance. These settings are shown in Figure 3–9.

Figure 3–9. Specify the Filter Architecture



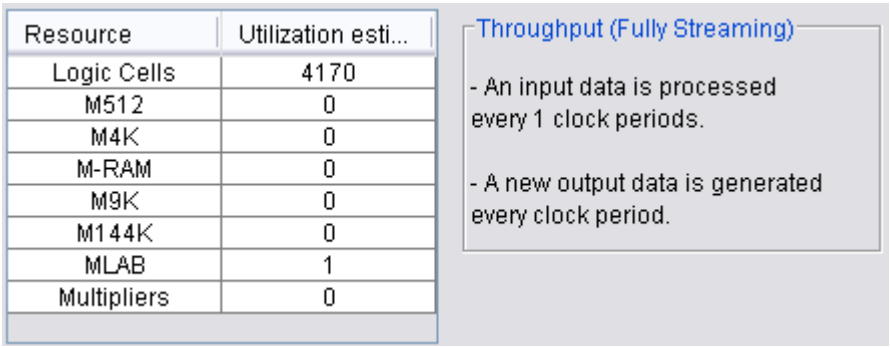
2. Click **Finish** when you have set the architecture parameters.

Resource Estimates

The FIR Compiler automatically calculates and displays the estimated resources that the filter will use in the **Resource Estimates** box of the **Architecture Specification** section (**Parameterize FIR Compiler** page).

The FIR Compiler provides the estimated size in embedded memory blocks, DSP blocks, and logic cells. The **Throughput** box displays the number of clock cycles required to compute the result (Figure 3–10).

Figure 3–10. Resource Estimates



The resource usage estimate may differ from Quartus II resource usage by +/- 30%, depending on which optimization method you use in the Quartus II software. Additionally, the resource estimator is less accurate for small filters (500 logic cells or less). For small filters, compile the design in the Quartus II software to obtain the resource usage.

Filter Design Tips

This section provides some additional tips for using the FIR Compiler:

- To prevent high-pass filters from rolling off near Nyquist, select an odd number of taps.
- You can import coefficients from the MATLAB software into the FIR Compiler via a text file. Simply save your coefficients as fixed or floating-point numbers to an ASCII file, one coefficient per line.
- To make a quadrature phase shift keying (QPSK), quadrature amplitude modulation (QAM), or phase shift keying (PSK) modulator or demodulator using the FIR Compiler, create a multichannel filter by indicating two or more channels on the input specification area.
- A comb filter is a filter that has repetitive notches. You can make a comb filter by first making a single-notch filter, and then using sub-sampling. The process of sub-sampling reflects or mirrors the notches in the frequency domain at all frequencies above Nyquist.

- When importing floating-point coefficients, you should apply a scaling factor to generate fixed-point integer numbers. Because coefficients are rounded to the nearest integer, the scaling (or gain) factor can be set to zero—i.e., if it is too small. If you do not scale the coefficients appropriately, you may have a filter with many zeros.
- The highest throughput filters are parallel filters with extended pipelining that generate an output for every clock cycle.
- Altera recommends that you use memory blocks to reduce the area.
- The FIR filter typically runs at a higher f_{max} if the following constraints are set:

```
set_global_assignment -name "PHYSICAL_SYNTHESIS_COMBO_LOGIC" "ON"
set_global_assignment -name "PHYSICAL_SYNTHESIS_REGISTER_RETIMING" "ON"
```

- Standard Fit (highest effort) is recommended for the fitter settings in the Quartus II software to achieve optimum synthesis results.
- To enable the decimation half-band optimized architecture, data storage and coefficient storage should be set to either **Auto** or one of the available block memories. Then select the filter tap value to be an odd number. The coefficient set should be symmetric and every other coefficient value should be 0.
- To enable the symmetric-interpolation optimized architecture, data storage and coefficient storage should be set to either `Auto` or one of the available block memories. The number of taps should be an odd value. Currently only even symmetry is supported.

FIR Compiler

The FIR Compiler has an interactive wizard-driven interface that allows you to easily create custom FIR filters. The wizard outputs IP functional simulation model files for use with Verilog HDL and VHDL simulators.

Number Systems & Fixed-Point Precision

The FIR Compiler function supports signed or unsigned fixed-point numbers from 4- to 32-bits-wide in two's complement and signed binary fractional formats.

The entire filter operates in a single number system. The coefficient precision is independent of input data width; you can specify the output precision.

Generating or Importing Coefficients

You can use the FIR Compiler function to create coefficients, or you can create them using another application such as MATLAB, save them as an ASCII file, and read them into the FIR Compiler.

Coefficients can be expressed as floating-point or integer numbers; each one must be listed on a separate line.

If you specify negative values for the coefficients, the FIR Compiler generates a two's complement signed number.

Figure 4–1 shows the contents of a sample coefficient text file.

Figure 4–1. Sample Filter Coefficients

```
-3.09453e-005
-0.000772299
-0.00104106
-0.000257845
 0.00150377
.
.
.
 0.00163125
 0.00278506
 0.00150377
-0.000257845
-0.00104106
-0.000772299
-3.09453e-005
```

The FIR Compiler automatically creates coefficients (with a user-specified number of taps) for the following filters:

- Low Pass
- High Pass
- Band Pass

- Band Reject
- Raised Cosine
- Root Raised Cosine
- Half Band (low pass)

You can adjust the number of taps, cut-off frequencies, sample rate, filter type, and window method to build a custom frequency response. Each time you apply the settings, the FIR Compiler calculates the coefficient values and displays the frequency response on a logarithmic scale. The coefficients are floating-point numbers and must be scaled.

The values are displayed in the Coefficients scroll-box, of the **Coefficients Generator Dialog** box, refer to Figure 3–2 on page 3–3.

When the FIR Compiler reads in the coefficients, it automatically detects any symmetry. The filter gives you several scaling options, for example, scaling to a specified number of bits or scaling by a user-specified factor.

The scaled coefficients are displayed in the **Time Response & Coefficient Values** tab of the **Parameterize FIR Compiler** page, refer to Figure 3–5 on page 3–6.

Coefficient Scaling

Coefficient values are often represented as floating-point numbers. To convert these numbers to a fixed-point system, the coefficients must be multiplied by a scaling factor and rounded. The FIR Compiler provides five scaling options:

- *Auto scale to a specified number of precision bits*—Because the coefficients are represented by a certain number of bits, it is possible to apply whatever gain factor is required such that the maximum coefficient value equals the maximum possible value for a given number of bits. This approach produces coefficient values with the maximum signal-to-noise ratio.
- *Auto with a power of 2*—With this approach, the FIR Compiler selects the largest power of two scaling factor that can represent the largest number within a particular number of bits of resolution. Multiplying all of the coefficients by a particular gain factor is the same as adding a gain factor before the FIR filter. In this case, applying a power of two scaling factor makes it relatively easy to remove the gain factor by shifting a binary decimal point.
- *Manual*—The FIR Compiler lets you manually scale the coefficient values by a specified gain factor.
- *Signed binary fractional*—You can specify how many digits to use on either side of the decimal point (supported in the variable architecture only).
- *None*—The FIR Compiler can read in pre-scaled integer values for the coefficients and not apply scaling factors.

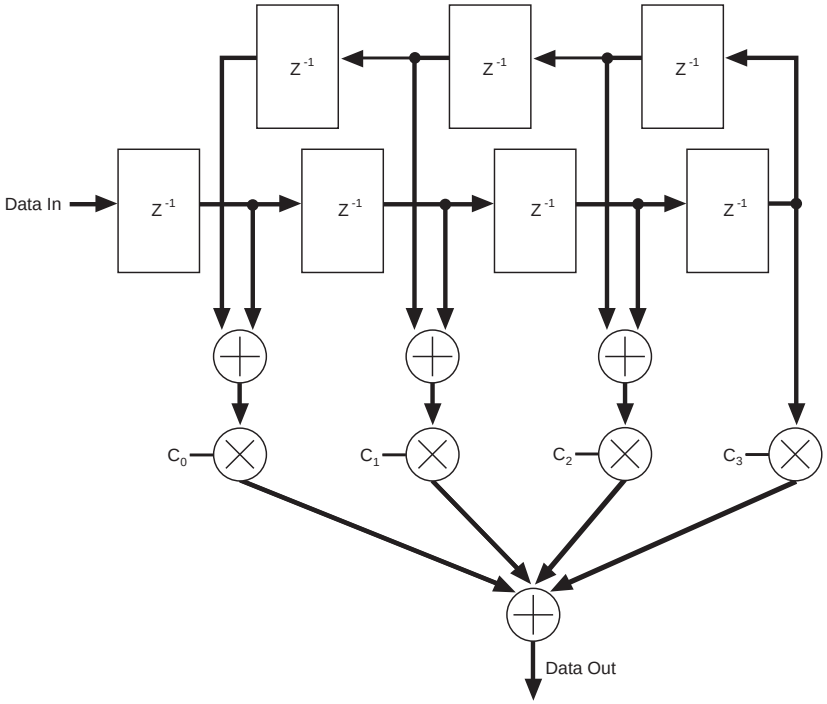
Symmetrical Architecture Selection

Many FIR filters have symmetrical coefficient values. The FIR Compiler examines the coefficients and automatically determines the filter's symmetry: even, odd, or none. After detecting symmetry, the wizard selects an optimum algorithm to minimize the amount of computation needed. The FIR compiler determines coefficient symmetry after the coefficients are rounded. If symmetry is present, two data points are added prior to the multiplication step, saving a multiplication operation (taking advantage of filter symmetry reduces the number of multipliers by about half).

The wizard gives you the option to force non-symmetrical structures. If the symmetry-optimized architecture is not available, this option is disabled.

Odd and even filter structures are shown in Figure 4–1 and Figure 4–2.

Figure 4–1. Seven-Tap Symmetrical FIR Filter



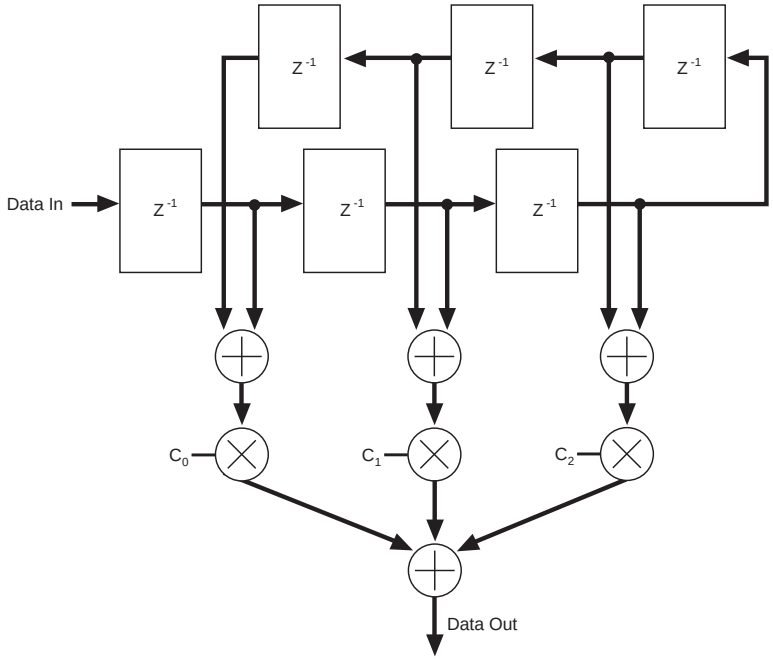
Symmetrical Serial

Symmetrical serial filters take an additional clock cycle to perform the FIR computation (so the filter can compute the carry). Additional logic cells are required for the symmetrical adder resources.

Because non-symmetrical serial FIR filters do not require this resource, non-symmetrical filters may be smaller and/or faster.

You can use the Resource Estimator in the **Architecture Specification** area of the **Parameterize FIR Compiler** page to determine the best solution available. Refer to Figure 3–9 on page 3–15).

Figure 4–2. Six-Tap Symmetrical FIR Filter



Coefficient Reloading and Reordering

All of the FIR Compiler structures allow multiple coefficient sets, and the filter can switch between coefficient sets dynamically. Additionally, while the filter uses one coefficient set, you can update other sets. Therefore, your filter can switch between an infinite number of coefficient sets.

To maximize silicon efficiency, coefficients are not stored in their natural order. Reordering is performed automatically during the initial design. However, if the filter coefficients are reloadable, any new coefficient set that you want to reload during the filter operation must be reordered before the reload. A C++ program that can be used to reorder coefficients is provided. A precompiled executable for Windows is also provided.

The program can be found in **<*install path*>\fir_compiler\misc**. The C++ source code file is named **coef_seq.cpp** and the executable program (for the Windows operating system) is **coef_seq.exe**. You can add the source code to your coefficient generation program, or use the executable file to re-order the coefficients.

The command to use `coef_seq.exe` is:

`coef_seq.exe` *<path>*`/input.txt` *<path>*`/output.txt` *<FIR structure> <coefficient store> <allow or disallow symmetry> <number of calculations for MCV | coefficient bit width for others> <number of coefficient sets> <filter rate> <filter factor> <coefficient bit width>*

☞ You should include the directory path with the input and output coefficient file names, as indicated above.

where:

- <*FIR structure*> is:
  - MCV—multicycle variable
  - SER—fully serial
  - MBS—multibit serial
  - PAR—fully parallel
- <*coefficient store*> is:
  - LC—logic cells
  - M512—M512 and MLAB blocks
  - M4K—M4K and M9K blocks
  - AUTO—Automatic memory block selection
- <*allow or disallow symmetry*> is:
  - MSYM—Take advantage of symmetric coefficients
  - NOSYM—Use nonsymmetric coefficients
- <*number of calculations for MCV | coefficient bit width for others*> is:
  - for multicycle variable filters, the number of clock cycles to calculate the result
  - for all other filters, use the coefficient bit width
- <*number of coefficient sets*> is the user-specified number of coefficient sets
- <*filter rate*> is be specified as one of the following (SGL, INT, DEC)
  - SGL—Single Rate FIR Filter
  - INT—Interpolating FIR Filter
  - DEC—Decimating FIR Filter
- <*filter factor*> is an integer value representing the rate-changing factor.
  - For single-rate filters, this argument should be set to 1
  - For multirate FIR filters, this argument should be an integer between 1 and 16
- <*coefficient bit width*> is the integer value representing the user-specified coefficient bit width, which ranges from 2-32

For example:

```
coef_seq.exe D:/FIR/coef_log.txt D:/FIR/coef_in.txt MCV M4K MSYM 4 1 SGL 1 8
```

The program checks for symmetry automatically, but you can force it to disallow symmetry. Your specification should be consistent with the setting in the FIR Compiler wizard.

The reloading capability allows you to change coefficient values. These filters may contain optimizations for symmetrical filters. If you want a filter that may need both symmetrical and non-symmetrical filters, turn on **Force Non-Symmetrical Structures** in the **Architecture Specification** section of the **Parameterize FIR Compiler** page.

If you select multiple-set coefficients, the filter can update one coefficient set while another set is being used for a calculation.

Structure Types

The FIR Compiler wizard generates multicycle variable, parallel, serial, multibit serial, and multichannel structures. All of these structures support coefficient reloading.

For information on reordering the coefficients before reloading them, refer to "Coefficient Reloading and Reordering" on page 4–4.

Multicycle Variable Structures

Multicycle variable (MCV) filters are optimized for high throughput. In a multicycle variable structure, the designer specifies that the filter uses 1 to 1,024 clock cycles to compute a result (for any filter that fits into a single device).

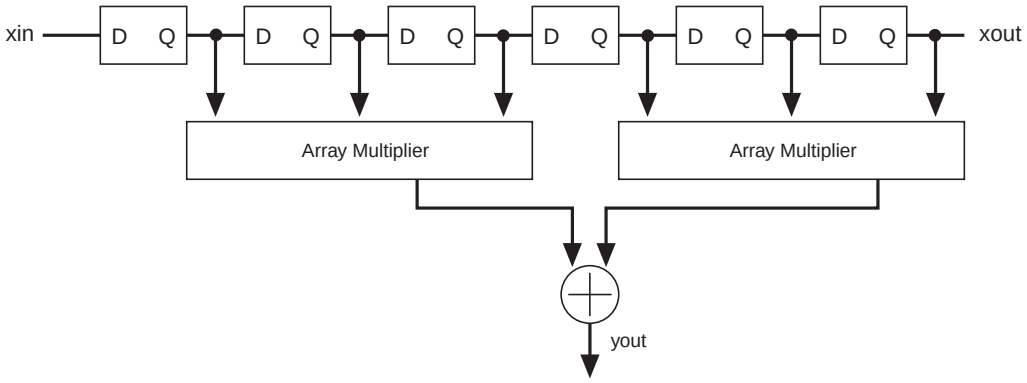
For Stratix, Stratix II, Stratix III, or Stratix IV devices, if you select the multicycle variable structure, selecting **DSP Blocks** in the **Multiplier** list box allows the FIR Compiler to use embedded DSP blocks for multipliers. This implementation results in a smaller and faster design.

Parallel Structures

A parallel structure calculates the filter output in a single clock cycle. Parallel filters provide the highest performance and consume the largest area. Pipelining a parallel filter allows you to generate filters that run between 120 and 300 MHz at the cost of pipeline latency.

Figure 4–3 shows the parallel filter block diagram.

Figure 4–3. Parallel Filter Block Diagram

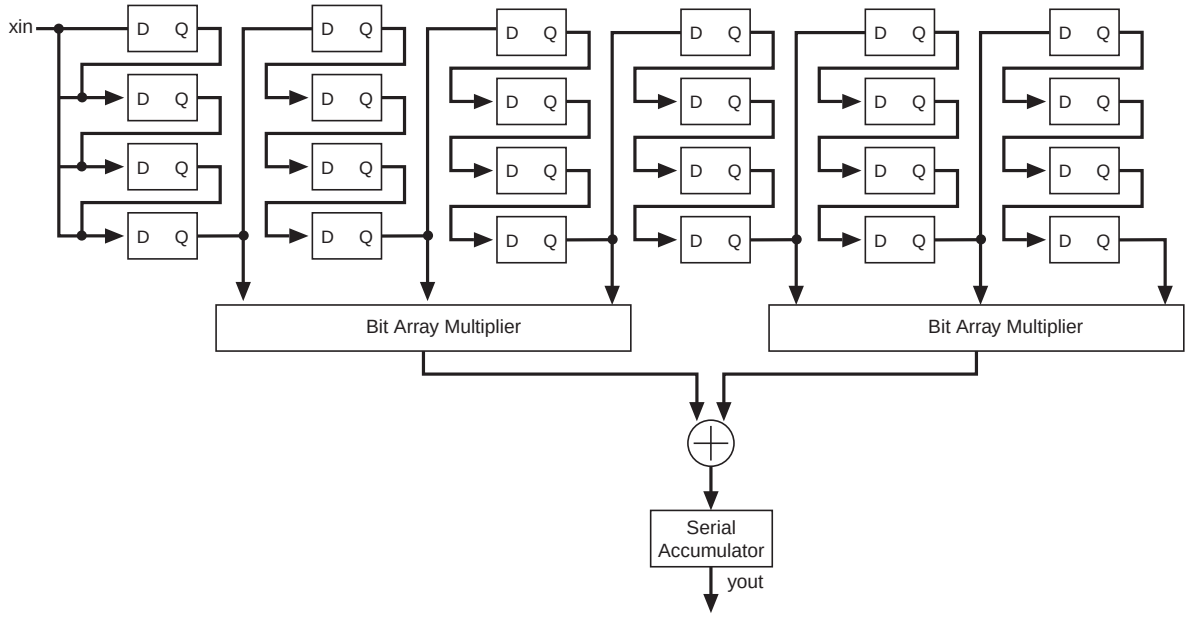


Serial Structures

A serial structure trades off area for speed. The filter processes input data one bit at-a-time per clock cycle. Therefore, serial structures require *N* clock cycles (where *N* is the input data width) to calculate an output. In the Stratix IV, Stratix III, Stratix II, Stratix, Cyclone III, Cyclone II, and Cyclone device families, using memory blocks for data storage will result in a significant reduction in area.

Figure 4–4 shows the serial filter block diagram.

Figure 4–4. Serial Filter Block Diagram



Multibit Serial Structure

A multibit serial structure combines several small serial FIR filters in parallel to generate the FIR result. This structure provides greater throughput than a standard serial structure while using less area than a fully parallel structure, allowing you to trade off device area for speed.

Figure 4–5 shows the multibit serial structure.

Figure 4–5. Multibit Serial Structure

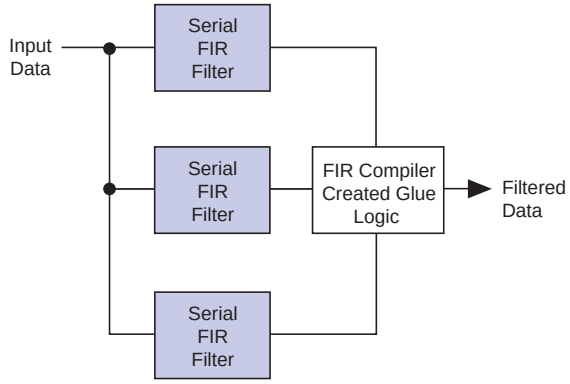
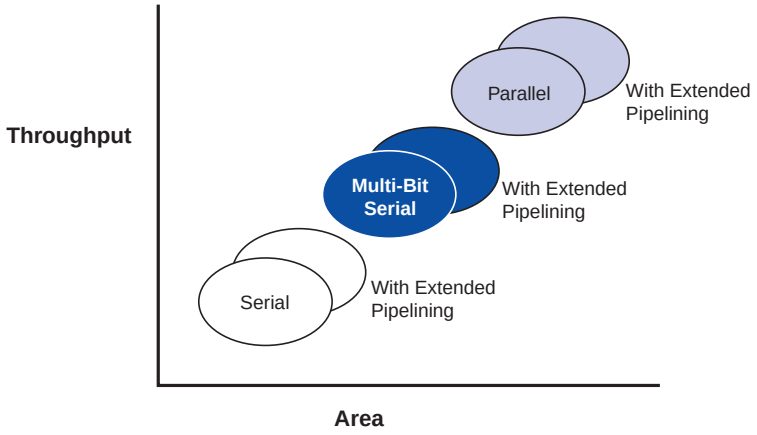


Figure 4–6 shows the area/speed "trade-off" of fixed FIR filters.

Figure 4–6. Fixed FIR Filters: Area Vs. Throughput



Two serial filters operating in parallel compute the result at twice the rate of a single serial filter. Three serial filters operate at triple the speed; four operate at four times the speed. For example, a 16-bit serial FIR filter requires 16 clock cycles to complete a single FIR calculation. A multibit serial FIR filter with two serial structures takes only eight clock cycles to compute the result. Using four serial structures, only four clock cycles are required to perform the computation. Three serial structures cannot be used for a 16-bit serial structure, however, because 16 does not divide evenly by three.

Multichannel Structures

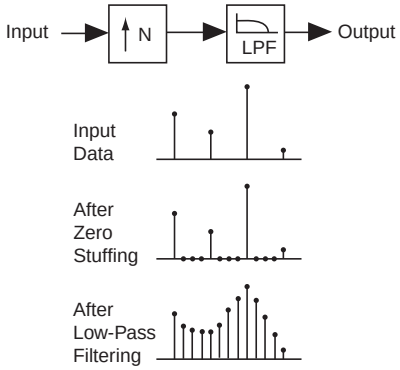
When designing DSP systems, you may need to generate two FIR filters that have the same coefficients. If high speed is not required, your design can share one filter, which uses fewer resources than two individual filters. For example, a two-channel parallel filter requires two clock cycles to calculate two outputs. The resulting hardware would need to run at twice the data rate of an individual filter.

To minimize the number of logic elements, use a distributed serial arithmetic architecture, multiple channels, and memory blocks for data and coefficient storage.

Interpolation & Decimation

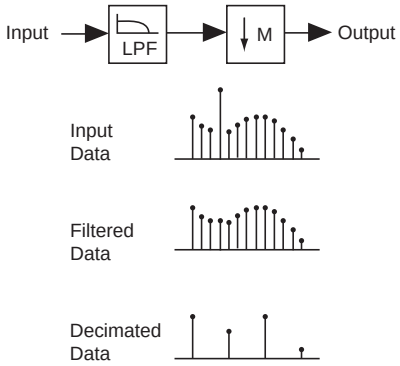
You can use the FIR Compiler to interpolate or decimate a signal. Interpolation generates extra points in between the original samples; decimation removes redundant data points. Both operations change the effective sample rate of a signal. Mathematically, when a signal is interpolated, zeros are inserted between data points and the data is then filtered to remove spectral components that were not present in the original signal (Figure 4–7 on page 4–9).

Figure 4–7. Signal Interpolation



To decimate a signal, a low-pass filter is applied, which removes spectral components that will not be present at the low sample rate. After filtering, appropriate sample values are taken (Figure 4–8).

Figure 4–8. Signal Decimation



The FIR Compiler generates interpolation and decimation filters by combining high- and low-level optimization techniques.

Using the high-level optimization technique, the FIR Compiler processes the data from a polyphase decomposed filter. The polyphase decomposition breaks a single filter into several smaller filters, which results in the following:

- When using an interpolation filter, zero-stuffed data does not need to be computed; potentially saving resources (Figure 4–9 on page 4–10).
- When using a decimation filter, output data—which is discarded during downsampling—is never computed, again potentially saving resources (Figure 4–10 on page 4–11).

Using the low-level optimization technique, the polyphase decomposed filter is implemented using a multichannel, multiple coefficient set structure with an appropriate wrapper. Because the FIR Compiler is an automated design tool, it is possible to implement a multichannel, multiple coefficient set interpolation or decimation filter (which is further implemented as a multichannel, multiple coefficient set structure).

The net result of these optimization techniques is a general savings in resources.

Implementation Details for Interpolation & Decimation Structures

Figure 4–9 and 4–14 illustrate the results when applying polyphase decomposition to interpolation and decimation filters.

Figure 4–9. Interpolation Filter Structure

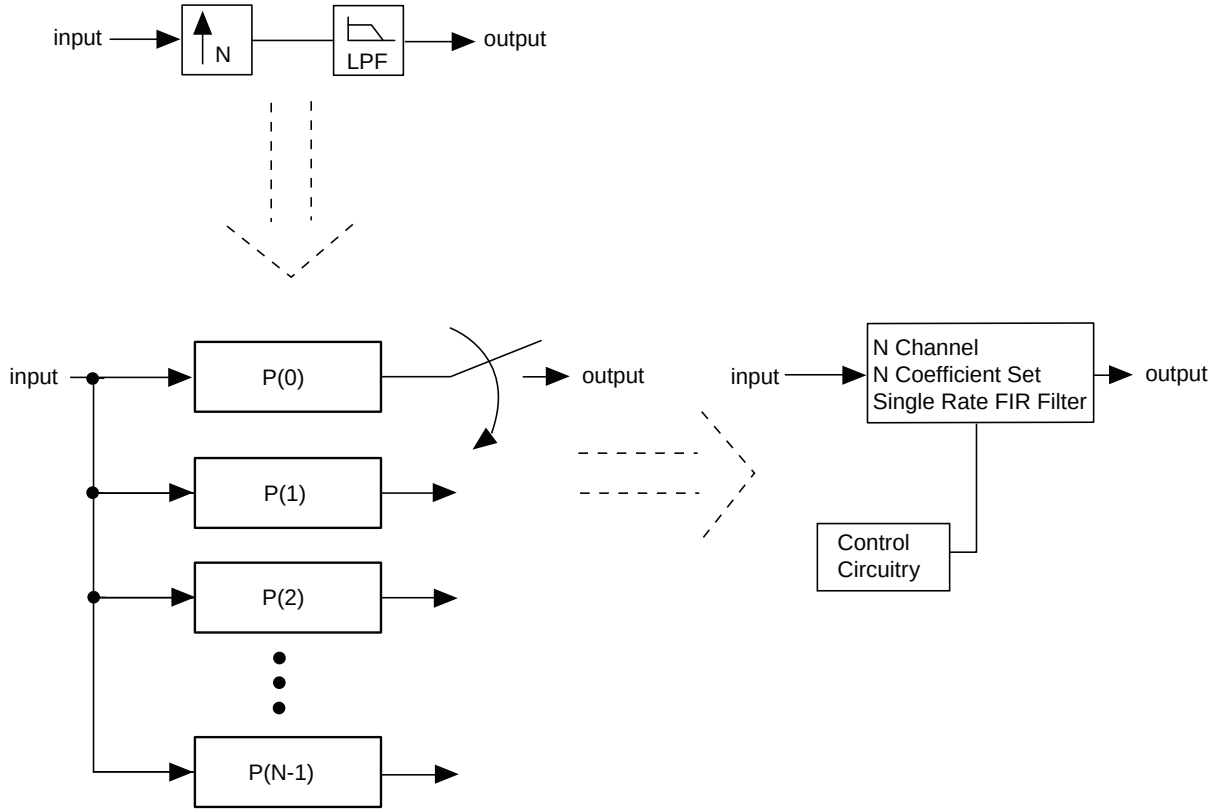
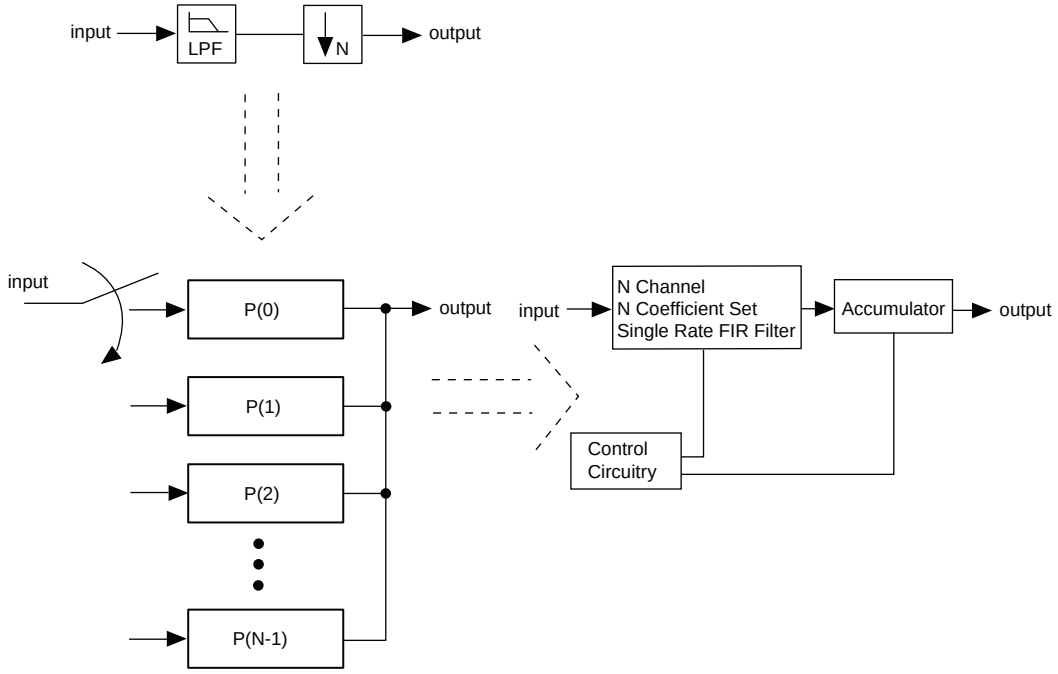


Figure 4–9 illustrates an interpolation structure. It takes a constant number of clocks to compute each polyphase output. The input data must be held for the number of clocks to compute each polyphase output multiplied by the number of polyphase units (which is the same as the interpolation factor).

Figure 4–10 on page 4–11 shows a decimation filter (with polyphase decomposition). Each polyphase filter must be computed prior to computing the final results. Because there are several polyphase results that must be accumulated, it is clear that the output will update every N clocks, where N = number of polyphase filters × number of clocks to compute each polyphase result.

The number of polyphase filters is equal to the decimation factor. The input data must be held for the time it takes to compute a single polyphase filter.

Figure 4–10. Decimation Filter Structure



Availability of Interpolation & Decimation Filters

Interpolation and decimation filters are available for all architectures:

- Parallel distributed arithmetic
- Serial distributed arithmetic
- Multibit serial distributed arithmetic
- Multicycle variable structures

All architecture configuration options are available for interpolation and decimation filters, including:

- User configuration of data storage type (memory or logic cells)
- User configuration of coefficient storage type (memory or logic cells)
- Multichannel capability
- Multiple coefficient set capability

Family-Specific Features

Stratix IV, Stratix III and Stratix II filters implement ternary adder structures in all architectures:

- Fully parallel distributed arithmetic
- Fully serial distributed arithmetic
- Multibit serial distributed arithmetic
- Multicycle variable

All multicycle variable structures allow the use of hard multipliers in Stratix IV, Stratix III, Stratix II, Stratix, Cyclone III, and Cyclone II structures. In addition, Stratix IV, Stratix III, Stratix II, and Stratix multicycle variable implementations take advantage of the built-in adder structures in the DSP block.

Stratix series devices allow the most flexibility for data and coefficient storage. You can choose between M512, M4K, and MRAM (when appropriate) for Stratix and Stratix II devices. Stratix III and Stratix IV devices support MLAB, M9K, and M144K.

Half-Band Decimation Filters

A decimation half-band optimized architecture is available for multicycle variable structures. This architecture uses half the number of multipliers compared to the decimation-symmetric architecture when a half-band coefficient set is selected. A halfband coefficient set has an odd number of symmetric coefficients and every other coefficient value is 0.

Currently only a single fixed-coefficient set is supported with this optimized architecture. The data storage and coefficient storage should be set to either `Auto` or one of the available block memories. Any value for the decimation factor and the number of channels can be selected. The number of clocks to compute should be greater than 1. The FIR Compiler automatically picks the decimation half-band optimized architecture when these conditions are met.

Symmetric-Interpolation Filters

A new symmetric-interpolation optimized architecture is available for multicycle variable structures. This architecture requires half the number of multipliers compared to the standard interpolation filter when a symmetric coefficient set is selected.

The number of filter taps should be an odd value. Currently only a single fixed-coefficient set is supported with the optimized architecture. The data storage and coefficient storage should be set to either `Auto` or one of the available block memories. Any value for interpolation factor and number of channels can be selected. The number of clocks to compute should be greater than 1. The FIR compiler automatically picks the optimized architecture when these conditions are met.

Pipelining

Pipelining is most effective for producing high-performance filters at the cost of increased latency: the more pipeline stages you add, the faster the filter becomes.

Pipelining breaks long carry chains into shorter lengths. Therefore, if the carry chains in your design are already short, adding pipelining may not speed your design.

The FIR Compiler lets you select whether to add one, two, or three pipeline levels.

Simulation Output

The FIR Compiler generates a number of output files for design simulation. After you have created a custom FIR filter, you can use the output files with MATLAB or VHDL simulation tools. You can use the test vectors and MATLAB software to simulate your design.

IP functional simulation models will output correct data only when data storage is clear. When data storage is not clear, functional simulation models will output non-relevant data. The number of clock cycles it takes before relevant samples are available is N; where N = (number of channels) × (number of coefficients) × (number of clock cycles to calculate an output).

For a full list of files generated by the FIR Compiler, refer to Table 2–1 on page 2–6 .

Avalon Streaming Interface

The Avalon® Streaming (Avalon-ST) interface defines a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface and simplifies the process of controlling the flow of data in a datapath.

Avalon-ST interface signals can describe traditional streaming interfaces supporting a single stream of data without knowledge of channels or packet boundaries.

Such interfaces typically contain data, ready, and valid signals. The Avalon-ST interface can also support more complex protocols for burst and packet transfers with packets interleaved across multiple channels.

The Avalon-ST interface inherently synchronizes multi-channel designs, which allows you to achieve efficient, time-multiplexed implementations without having to implement complex control logic.

The Avalon-ST interface supports backpressure, which is a flow control mechanism where a sink can signal to a source to stop sending data. The sink typically uses backpressure to stop the flow of data when its FIFO buffers are full or when there is congestion on its output.

When designing a datapath which includes the FIR Compiler MegaCore function, you may not need backpressure if you know the downstream components can always receive data. You may achieve a higher clock rate by driving the `ast_source_ready` signal of the FIR Compiler high, and not connecting the `ast_sink_ready` signal.

The coefficient reload related ports and coefficient set selection ports in multi-set filters are not Avalon Streaming compliant.

The *Avalon Interface Specifications* define parameters which can be used to specify any type of Avalon-ST interface. Table 4–1 on page 4–14 lists the values of these parameters that are defined for the Avalon-ST interfaces used by the FIR Compiler. All parameters not explicitly listed in the table have undefined values.

Table 4–1. Avalon-ST Interface Parameters

Parameter Name	Value
READY_LATENCY	0
BITS_PER_SYMBOL	data width
SYMBOLS_PER_BEAT	1
SYMBOL_TYPE	signed/unsigned
ERROR_DESCRIPTION	00: No error 01: Missing `startofpacket` (SOP) 10: Missing `endofpacket` (EOP) 11: Unexpected EOP or any other error

The *Avalon Interface Specifications* define many signal types many of which are optional. Table 4–2 lists the signal types used by the Avalon-ST interfaces for the FIR Compiler MegaCore function. Any signal type not explicitly listed in the table is not included.

Table 4–2. Avalon-ST Interface Signal Types

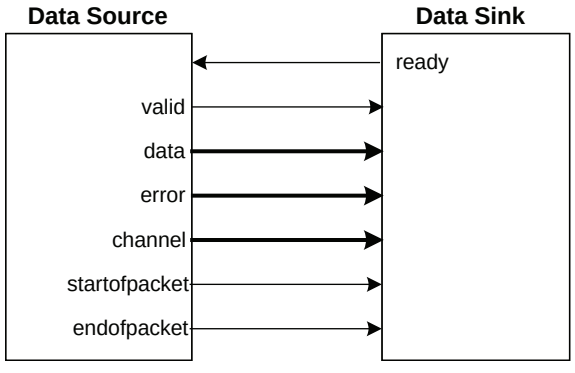
Signal Type	Width	Direction
ready	1	Sink to Source
valid	1	Source to Sink
data	data width	Source to Sink
channel	$\log_2$(number of channels)	Source to Sink
error	2	Source to Sink
startofpacket	1	Source to Sink
endofpacket	1	Source to Sink

For a full description of the Avalon-ST interface protocol, refer to the *Avalon Interface Specifications*.

Avalon-ST Data Transfer Timing

Figure 4–11 shows the Avalon-ST interface signals.

Figure 4–11. Avalon-ST Interface

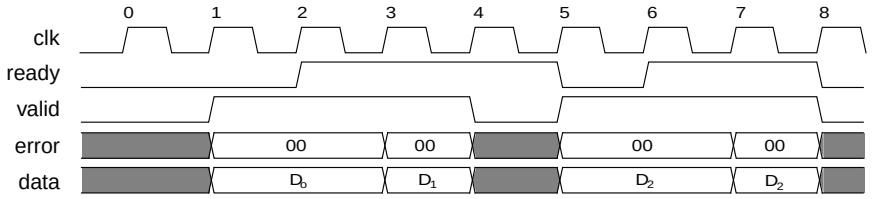


The sink indicates to the source that it is ready for an active cycle by asserting the `ready` signal for a single clock cycle. Cycles during which the sink is ready for data are called *ready cycles*. During a ready cycle, the source may assert `valid` and provide data to the sink. If it has no data to send, it deasserts `valid` and can drive data to any value.

When READY_LATENCY=0, data is transferred only when `ready` and `valid` are asserted on the same cycle. In this mode of operation, the source data does not need to receive the sink's `ready` signal before it begins sending valid data. The source provides the data and asserts `valid` whenever it can and waits for the sink to capture the data and assert `ready`. The sink only captures input data from the source when `ready` and `valid` are both asserted.

Figure 4–12 illustrates the data transfer timing.

Figure 4–12. Avalon-ST Interface Timing with READY_LATENCY=0



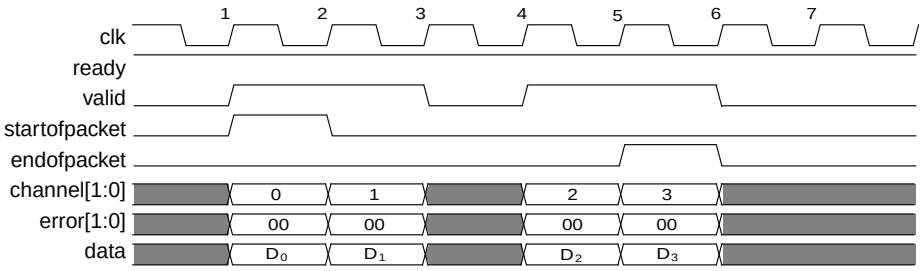
The source provides data and asserts `valid` on cycle 1, even though the sink is not ready. The source waits until cycle 2, when the sink does assert `ready`, before moving onto the next data cycle. In cycle 3, the source drives data on the same cycle and because the sink is ready to receive it, the transfer occurs immediately. In cycle 4, the sink asserts `ready`, but the source does not drive valid data.

Packet Data Transfers

A beat is defined as the transfer of one unit of data between a source and sink interface. This unit of data may consist of one or more symbols and makes it is possible to support modules that convey more than one piece of information on each valid cycle. Packet data transfers are used for multichannel transfers. Two additional signals (`startofpacket` and `endofpacket`) are defined to implement the packet transfer.

Figure 4–13 shows an example where the `channel` signal shows to which channel the data sample belongs.

Figure 4–13. Packet Data Transfer



The `channel` input signal is not used in the FIR Compiler interface.

The data transfer in Figure 4–13 occurs on cycles 1, 2, 4, and five, when both `ready` and `valid` are asserted. During cycle 1, `startofpacket` is asserted, and the first data is transferred. During cycle 5, `endofpacket` is asserted indicating that this is the end of the packet.

The `channel` signal indicates the channel index associated with the data. For example, on cycle 1, the data D_0 associated with channel 0 is available.

The `error` signal stays at value `00` during a normal operation. Whenever a value other than `00` is received from the data source (as in Figure 4–11), or a packet error is detected by the Avalon-ST controller of the FIR filter, the controller is reset and waits for the next valid `startofpacket` signal. It also transmits the received error signal from its data source module error output.

The `error` signal only resets the Avalon-ST controller and not the design. Therefore, the output data produced after an error condition may contain invalid data for several cycles. It is recommended that a global reset is applied whenever an error message is present in the system.

Signals

Table 4–3 lists the input and output signals for the FIR Compiler MegaCore function.

Table 4–3. *FIR Compiler Signals (Part 1 of 2)*

Signal	Direction	Description
`clk`	Input	Clock signal used to clock all internal FIR filter registers.
`enable`	Input	Active high clock enable signal. This pin appears when the **Add global clock enable pin** option is selected on the **Parameterize FIR Compiler** page. (The Avalon-ST registers are NOT connected to this clock enable.)
`reset_n`	Input	Synchronous active low reset signal. Resets the FIR filter control circuit on the rising edge of `clk`. This signal should last longer than one clock cycle.
`ast_sink_ready`	Output	Asserted by the FIR filter when it is able to accept data in the current clock cycle.
`ast_sink_valid`	Input	Asserted when input data is valid. When `ast_sink_valid` is not asserted, the FIR processing is stopped if new data is required and no data is left in the Avalon-ST input FIFO. Otherwise, the FIR processing continues.
`ast_sink_data`	Input	Sample input data.
`ast_sink_sop`	Input	Marks the start of the incoming sample group. The start of packet (SOP) is interpreted as a sample from channel 0.
`ast_sink_eop`	Input	Marks the end of the incoming sample group. If there is data associated with N channels, the end of packet (EOP) must be high when the sample belonging to the last channel (that is, channel *N*-1), is presented at the data input.
`ast_sink_error`	Input	Error signal indicating Avalon-ST protocol violations on the sink side: ■ 00: No error ■ 01: Missing SOP ■ 10: Missing EOP ■ 11: Unexpected EOP Other types of errors are also marked as 11.
`ast_source_ready`	Input	Asserted by the downstream module if it is able to accept data.
`ast_source_valid`	Output	Asserted by the FIR filter when there is valid data to output.

Table 4–3. *FIR Compiler Signals (Part 2 of 2)*

ast_source_channel	Output	Indicates the index of the channel whose result is presented at the data output. The width of this signal = $\log_2$(number of channels).
ast_source_data	Output	Filter output. The data width depends on the parameter settings.
ast_source_sop	Output	Marks the start of the outgoing FIR filter result group. If '1', a result corresponding to channel 0 is output.
ast_source_eop	Output	Marks the end of the outgoing FIR filter result group. If '1', a result corresponding to channel *N*-1 is output, where *N* is the number of channels.
ast_source_error	Output	Error signal indicating Avalon-ST protocol violations on the source side: ■ 00: No error ■ 01: Missing SOP ■ 10: Missing EOP ■ 11: Unexpected EOP Other types of errors are also marked as 11.
coef_set	Input	Selects which coefficient set the FIR filter uses for the calculation. (Appears when multiple coefficient sets are used.) The width of this signal = $\log_2$(number of coefficient sets).
coef_in_clk	Input	Clock to reload coefficients when coefficients are stored in memory. (Appears when the **Coefficient Reload** option is selected and the **Use Single Clock** option is not selected) This clock can be different than clk.
coef_set_in	Input	Selects which coefficient set to be reloaded. (Appears when multiple coefficient sets are used and the **Coefficient Reload** option is selected.) The width of this signal = $\log_2$(number of coefficient sets)
coef_in	Input	Input coefficient value when reloading coefficient. (Appears when the **Coefficient Reload** option is selected)
coef_we	Input	Active high write enable signal. Enables coefficient overwriting when coefficients are reloadable.
coef_ld	Output	Coefficient reload control port. This port is created only when multicycle filters are selected and the coefficient storage is logic cells.

Timing Diagrams

The reset_n signal resets the control logic and state machines that control the FIR Compiler (not including data storage elements that hold previous inputs used to calculate the result).

The previous data is not cleared when the reset_n signal is applied. To clear the data, set the ast_sink_data port to 0 for n clock cycles, where *n* = (*number of coefficients*) × (*number of input channels*) × (*number of clock cycles needed to compute a FIR result*).

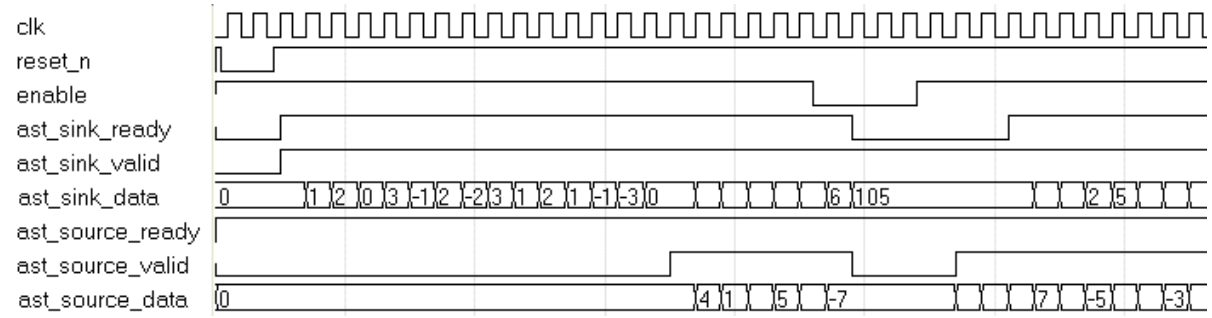
The FIR output value depends on the coefficient values in the design. Therefore, the timing diagrams of your own design may be different than those shown in the following figures. However, you can use the testbench generated by the FIR compiler to get the correct timing relation between signals for a specific parameterized case.

All timing diagrams assume a full streaming operation where ast_source_ready and ast_sink_ready are always 1 (unless otherwise stated).

Reset and Global Clock Enable Operations

Figure 4–14 shows the reset and clock enable operations.

Figure 4–14. Reset & Clock Enable Protocol



When the reset (`reset_n`) is applied to the filter, the `ast_sink_ready` and `ast_source_valid` signals go low. At the next rising edge of the clock (`clk`) after the reset is released, `ast_sink_ready` goes high indicating that the design is ready to accept new data. This behavior is independent of the filter type and architecture because there is a small FIFO in the Avalon-ST controller.

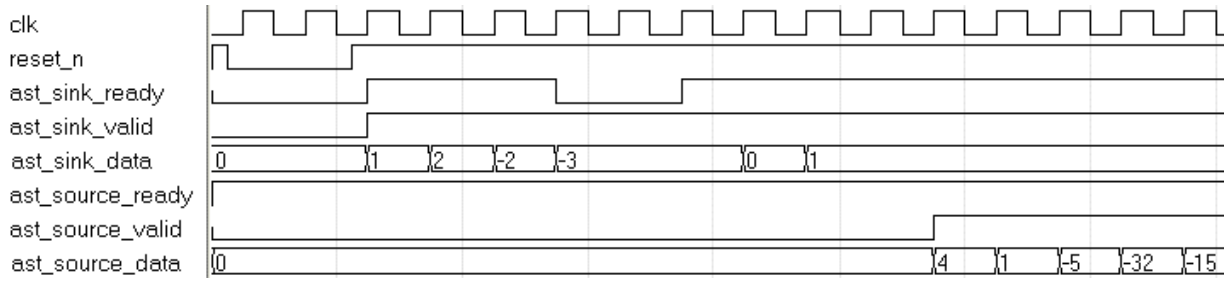
The global clock enable signal (where it exists) can also control when the FIR MegaCore function is stalled. The Avalon-ST controller operates independent of the global clock enable. The FIR is stalled as soon as the global clock enable goes low. However, because of the internal buffering in the Avalon-ST controller, the `ast_sink_ready` signal can go low in the following cycles. For the same reason, when the global enable is high, `ast_sink_ready` may go high at a later cycle.

The `ast_source_valid` signal is produced by the Avalon-ST controller is therefore independent of the global clock enable. When the available valid data is transferred, and no more output data is available, it goes low until there is valid data to transfer.

Single Rate Filter Timing Diagram

Figure 4–15 shows the timing diagram of a single channel single rate FIR filter implemented either in MCV architecture with a Clocks to Compute value of 1, or in Parallel architecture.

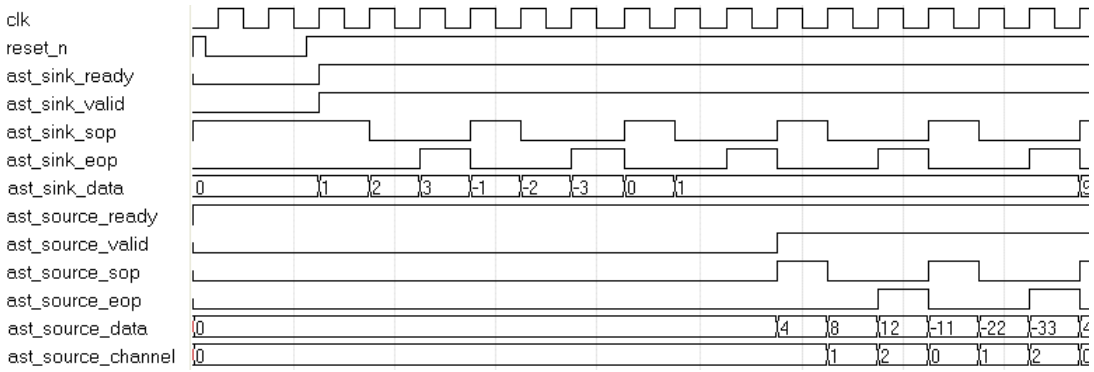
Figure 4–15. Single Channel, Single Rate (Parallel or MCV Single Cycle)



This filter accepts an input every clock cycle and produces an output every clock cycle. Because `ast_source_ready` and `ast_sink_valid` are kept at high, the filter can internally run fully streaming. An input is transferred when `ast_sink_ready` and `ast_sink_valid` are both high during the rising edge of the clock.

Figure 4–16 shows a three channel filter with the same specification as the single channel filter in Figure 4–15.

Figure 4–16. Three Channel, Single Rate (Parallel or MCV Single Cycle)



The FIR filter now has start of packet (`sop`) and end of packet (`eop`) signals for both the sink (input) and source (output) modules. The first input data to the FIR filter is accompanied by the high value of the `ast_sink_sop` port, which means it belongs to the first channel.

The third input data is marked as an end of packet by the high value of the `ast_sink_eop` port. This sequence repeats itself continuously at each cycle.

When the filter output is ready, `ast_source_valid` goes high, and for the first data output `ast_source_sop` goes high to mark the start of the packet. The `ast_source_channel` output shows to which channel that particular output belongs. The last channel data is marked with the high value of the `ast_source_eop` port.

Figure 4–17 and Figure 4–18 on page 4–20 demonstrate another single channel, single rate filter timing diagram. In these diagrams, the FIR filter requires input data every three clock cycles and produces one output data every three clock cycles. In general, MCV multicycle filters (when the Clocks to Compute value is greater than one), Multibit Serial filters, and Serial filters require a new input data every *N* clock cycles where *N* represents the following:

- For an MCV multicycle filter, *N* is the clocks to compute value
- For a Multibit Serial filter, *N* = (input data bit width)/(number of serial units)
- For a Serial filter, *N* = (input data bit width +1)

Figure 4–17. Single Channel, Single Rate (Serial, Multibit Serial, MCV Multicycle), ast_sink_valid Control

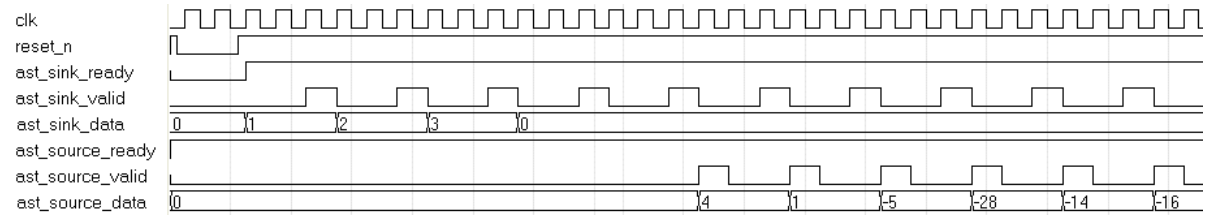
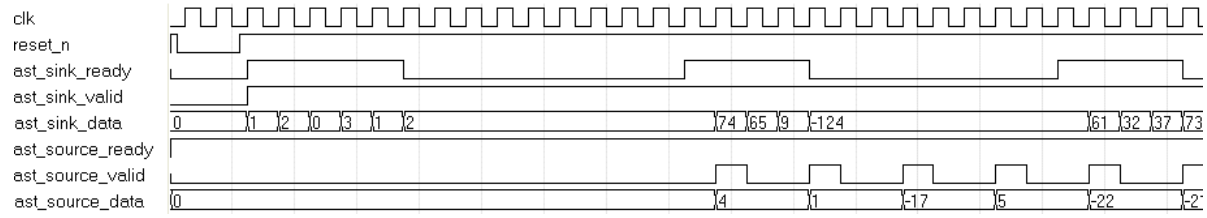


Figure 4–18. Single Channel, Single Rate (Serial, Multibit Serial, MCV Multicycle) ast_sink_ready Control



In Figure 4–17, the flow is controlled by the data provider asserting `ast_sink_valid` every three clock cycles.

In Figure 4–18, `ast_sink_valid` is always held high and the data provider can feed new data in every clock cycle, but the filter accepts new data every three clock cycles by asserting `ast_sink_ready`.

In this scenario, a number of data samples are fetched at once and then `ast_sink_ready` is de-asserted for a longer period. This behavior is due to the internal buffering of the Avalon-ST controller.

Interpolation Filter Timing Diagrams

Figure 4–19 and Figure 4–20 on page 4–21 show a single channel interpolation-by-2 filter with a parallel architecture.

Figure 4–19. Single Channel, Interpolation-by-2 (Parallel, MCV Single Cycle), ast_sink_valid Control

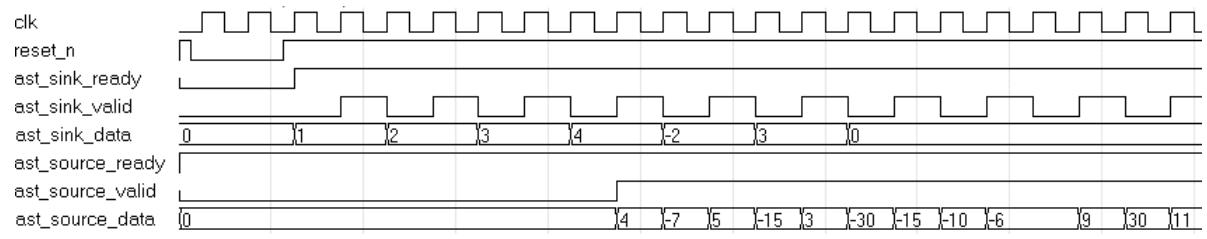
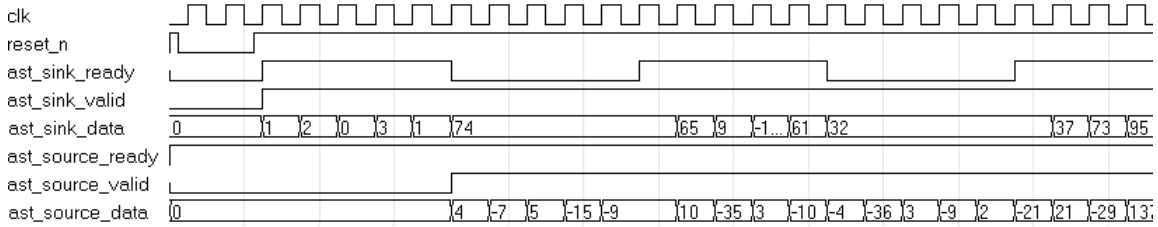


Figure 4–20. Single Channel, Interpolation-by-2 (Parallel, MCV Single Cycle), ast_sink_ready Control

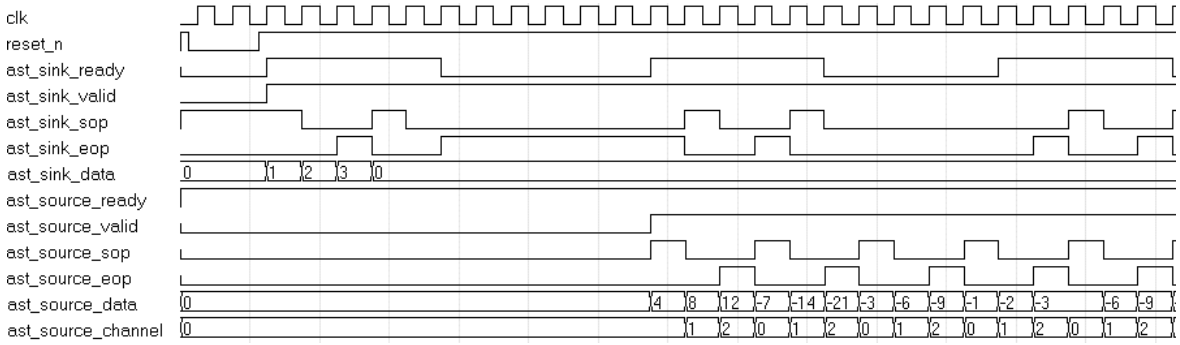


These timing diagrams also apply to an MCV single cycle filter. An interpolation-by-2 filter produces two output data for each input data it receives. As seen from the figures, a new output is produced every cycle. This means that new input data is required every other cycle.

This behavior can be observed easily when the flow of data is controlled by the `ast_sink_valid` signal as in Figure 4–19.

Figure 4–21 shows the timing diagram for a three channel, interpolation-by-2 filter with a Parallel or MCV single cycle architecture illustrating the additional start of packet and end of packet signals.

Figure 4–21. Three Channel, Interpolation-by-2 (Parallel, MCV Single Cycle)

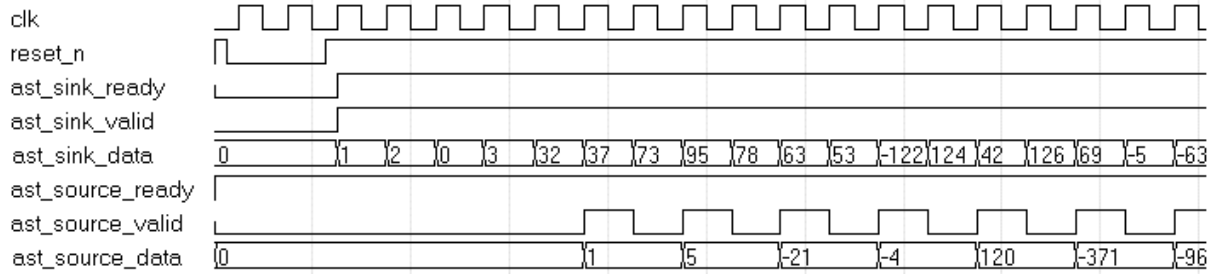


The timing diagrams for Serial, Multibit serial, and MCV multicycle filters would be similar to Figure 4–17 and Figure 4–18. For an MCV filter with a Clocks to Compute value of *N*, and interpolation factor *M*, new input data is required every *N*×*M* clock cycles and a new output would be produced every *M* clock cycles.

Decimation Filter Timing Diagrams

In a decimation-by-M filter, for every *M* input data, one output will be produced. Figure 4–22 on page 4–22 shows that for a Parallel or MCV (single cycle) decimation-by-2 filter, new input data is taken each clock cycle and new output data is produced every other clock cycle.

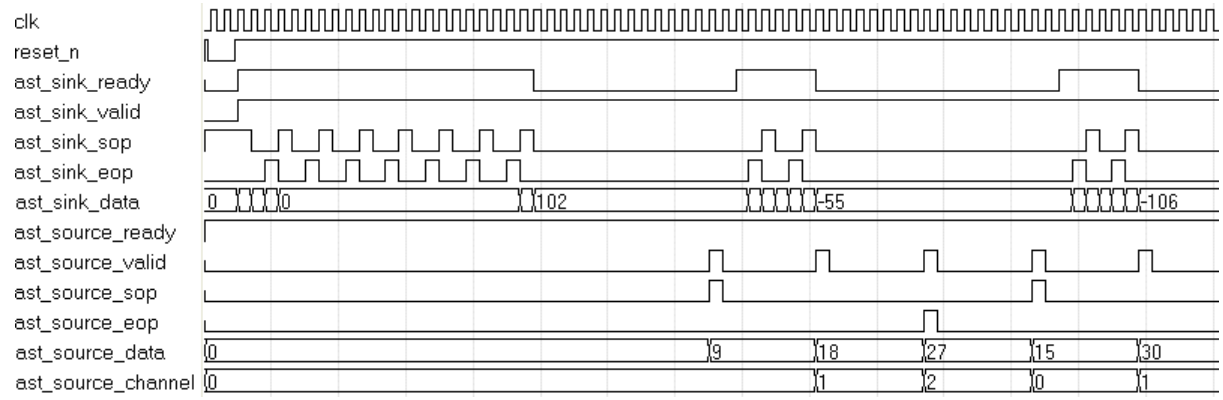
Figure 4–22. Single Channel, Decimation-by-2 (Parallel, MCV Single Cycle)



If the filter is time-shared by *N*, (that is the Clocks to compute value is *N*), then new input data is required every *N* clock cycles and new output data is produced every *N*×*M* cycles.

Figure 4–23 shows the timing diagram for a Multibit serial filter with two serial units and an input data width of 8-bits.

Figure 4–23. Three Channel, Decimation Filter (Serial, MBS, MCV Multicycle)



The filter needs new data every four clock cycles and produces an output every 8 clock cycles. Because the flow is controlled by `ast_sink_ready`, the input data fetching occurs in a groups, where `ast_sink_ready` goes high for five cycles and five new data inputs are taken at once. Then `ast_sink_ready` goes low and no data is accepted for 15 cycles. The `ast_source_sop` and `ast_source_eop` signals mark the start of packet and end of packet respectively.

Coefficient Reloading Timing Diagrams

The coefficient reload ports are not Avalon-ST compliant and work independently of the enable and reset signals, and of the Avalon-ST controller. You can load new coefficients even if the filter is under reset conditions or not enabled.

Serial, multibit serial, and parallel FIR filters use a distributed arithmetic algorithm and the coefficients stored in memory blocks are precalculated. When updating the coefficients, the new coefficients first go through a pre-calculating algorithm. The first data to reload in each memory block is always zero. The rising edge of the coef_we signal resets the internal data address counter for reloading.

For information about how to pre-calculate coefficients, refer to "Coefficient Reloading and Reordering" on page 4–4.

In serial and multibit serial filters, coef_we is effective two clock cycles ahead of the first coef_in data and lasts until the last `coef_in` data is transmitted. In parallel filters, `coef_we` only needs to be effective one clock cycle ahead of the first `coef_in` data. To reload another set of coefficients, `coef_we` must be low for at least one clock cycle. The reload clock does not have to be the same clock as the one used by the FIR calculation.

Figure 4–24 shows the serial and multibit serial coefficient reloading timing diagram.

Figure 4–24. Serial & Multibit Serial Coefficient Reloading Timing Diagram

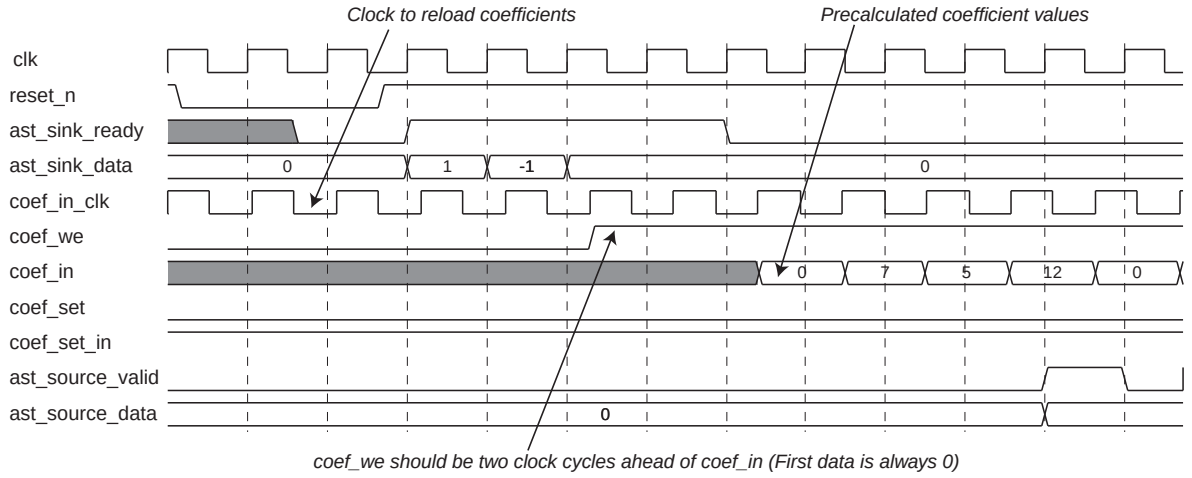
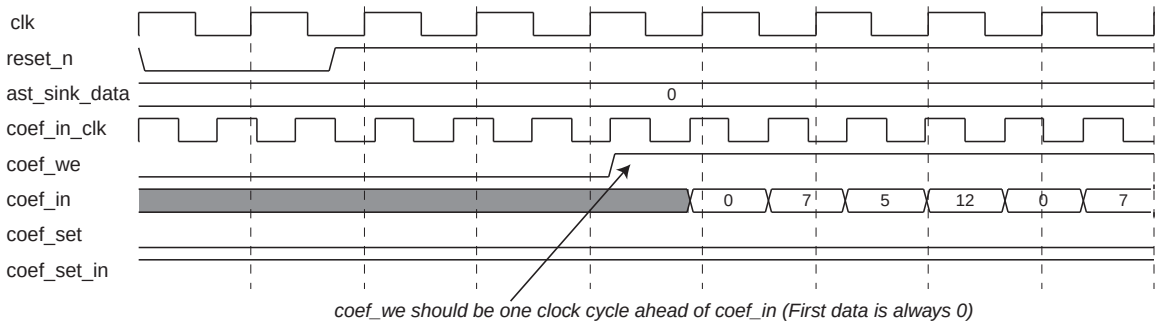


Figure 4–25 shows the parallel coefficient reloading timing diagram.

Figure 4–25. Parallel Coefficient Reloading Timing Diagram



Serial, multibit serial, and parallel FIR architectures use a distributed arithmetic algorithm. In the algorithm, look-up tables store partial products of the coefficient; the first data of the partial product is always 0. When reloading pre-calculated coefficients in serial, multibit serial, and parallel architectures, the first reloading coefficient is always 0.

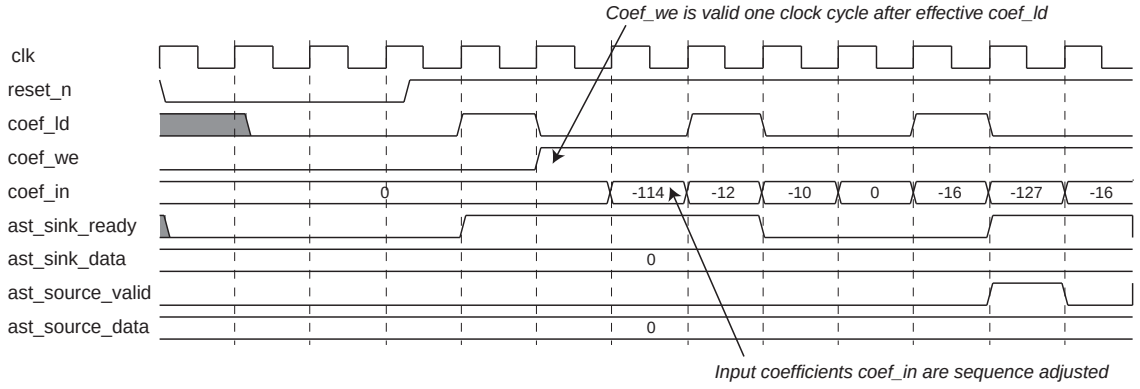
For information about how to pre-calculate coefficients, refer to "Coefficient Reloading and Reordering" on page 4–4.

Multicycle variable reloading is faster than the fixed FIR (with reloading capability). Coefficients need sequence adjustment using the same algorithm as fixed FIR filters for all types of coefficient storage. The reloading clock is the same as the FIR filter calculation clock; coef_we should be triggered by the `coef_ld` signal.

When the coefficients are stored in logic cells, a reloaded coefficient set reverts backs to the original set after a reset operation.

Figure 4–26 shows the Multicycle variable coefficient reloading timing diagram when the coefficients are stored in logic cells.

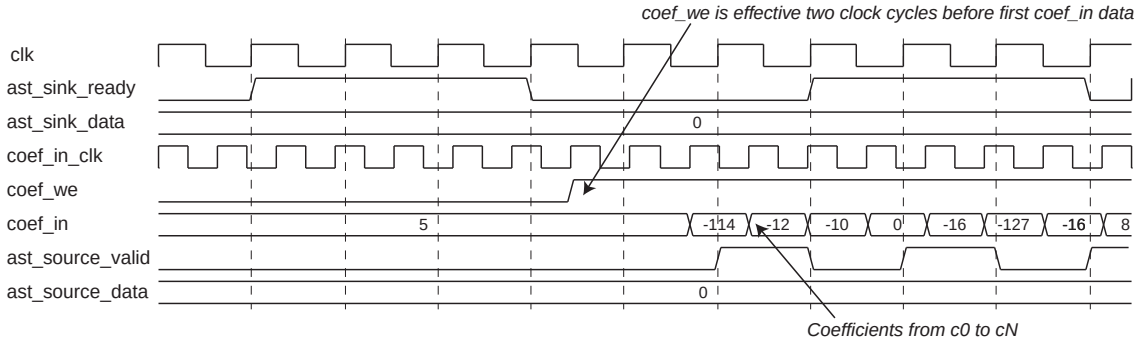
Figure 4–26. Multicycle Variable (Using Logic Cells) Coefficient Reloading Timing Diagram



For multicycle variable FIR filters, when coefficients are stored in memory blocks, `coef_we` should be effective two clock cycles before the first `coef_in` data, and should last until the last `coef_in data` is transmitted. In this case, coefficients do not need change sequence. Coefficients can be transmitted from c0 to cn by a different clock.

Figure 4–27 shows the Multicycle variable coefficient reloading timing diagram when the coefficients are stored in memory blocks.

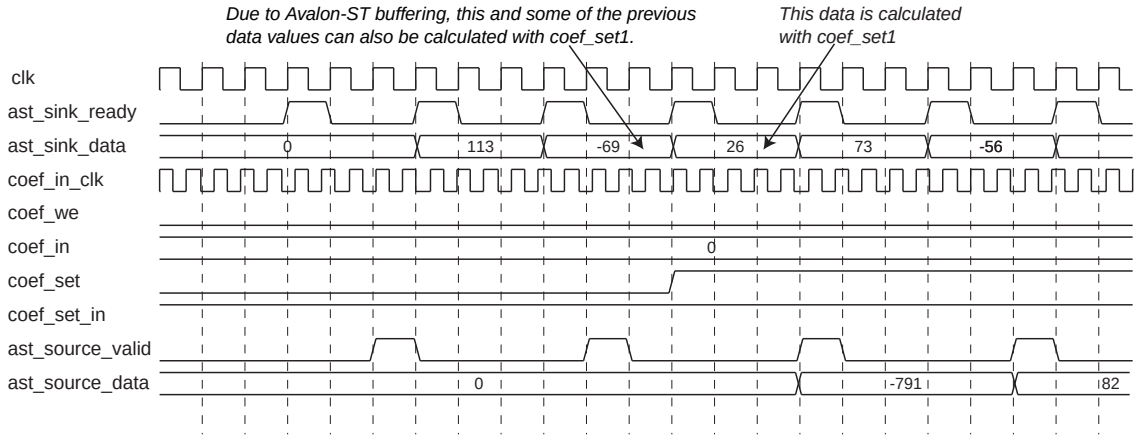
Figure 4–27. Multicycle Variable (Using Memory Blocks) Coefficient Reloading Timing Diagram



If you use multiple coefficient sets, you can update one set of coefficients while using another set for calculation. The signals `coef_set_in` and `coef_we` are not clocked in and pipelined synchronously. While you update the coefficient set, you need to set and hold the `coef_set_in` signal for several cycles before `coef_we` is asserted and after it is de-asserted.

The selection of the coefficient set for calculation is also not synchronous to the input data because of the Avalon-ST flow controller. Once the `coef_set` signal is set to a particular value, it immediately affects the operation of the filter. This means that some of the input data already received by the Avalon-ST controller will be calculated using the new coefficient set (Figure 4–28).

Figure 4–28. Multiple Coefficient Set Selection Timing Diagram

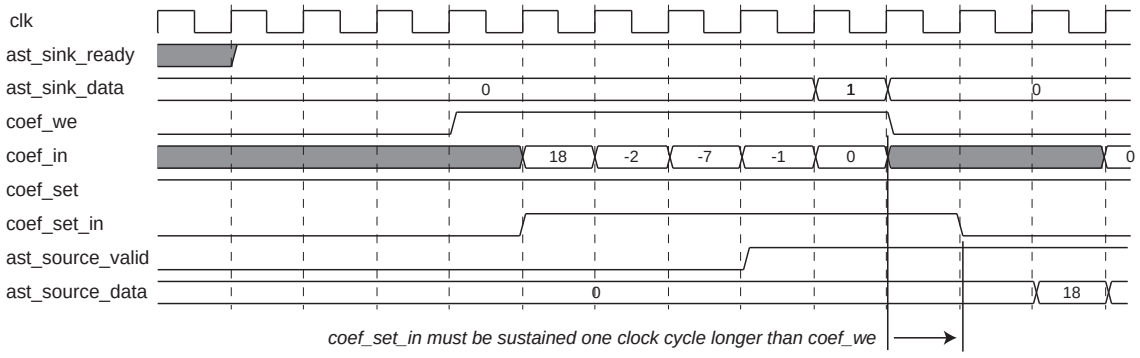


For DSP Builder users, a button is available that ties the `coef_in_clk` to the `clk` in the wrapper. DSP Builder will not work with more than one clock domain per MegaCore function.

When loading multiple coefficient sets, to identify the coefficient set being loaded, the duration of the clock cycle for coef_set_in must be one clock cycle longer than the duration of coef_we as shown in Figure 4–29.

These timing requirements affect all designs that load multiple coefficient sets. If the specified timing requirements are not met when loading multiple coefficient sets, a specific set of coefficients will not be identified.

Figure 4–29. Timing Requirements for Loading Multiple Coefficient Sets



References

Altera application notes, white papers, and user guides providing more detailed explanations of how to effectively design with MegaCore functions and the Quartus II software are available at the Altera web site (www.altera.com).

A. FIR Compiler Supported Device Structures

Supported Device Structures

Table A–1 lists the device structures supported by the FIR Compiler.

The data storage depends on the specified user settings. Cyclone® devices do not support M512 or MRAM memory. MLAB, M9K and M144K are supported by Stratix® III and Stratix IV devices only.

Table A–1. Device Structures Supported by FIR Compiler (Part 1 of 2)

Structure	Sub Structure	Input Bit Width	Flow Control	Data Storage	Coefficient Storage	Multiple Coefficient Sets	Symmetric Coefficient
Serial	**Fixed Coefficient**	4 to 32	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, M-RAM MLAB, M9K, M144K, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	Yes
	Reloadable Coefficient	4 to 32	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, M-RAM MLAB, M9K, M144K, Auto	M512, M4K, M512, M4K, MLAB, M9K, Auto	Yes	Yes
	Interpolation (Reloadable Coefficient must be in memory)	4 to 32	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, M-RAM MLAB, M9K, M144K, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	N/A
	Decimation (Reloadable Coefficient must be in memory)	4 to 32	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, M-RAM MLAB, M9K, M144K, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	N/A
Multibit Serial	**Fixed Coefficient**	≥ 4 (number of serial units)	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, M-RAM MLAB, M9K, M144K, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	Yes
	Reloadable Coefficient	≥ 4 (number of serial units)	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, M-RAM MLAB, M9K, M144K, Auto	M512, M4K, M512, M4K, MLAB, M9K, Auto	Yes	Yes
	Interpolation (Reloadable Coefficient must be in memory)	≥ 4 (number of serial units)	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, M-RAM MLAB, M9K, M144K, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	N/A
	Decimation (Reloadable Coefficient must be in memory)	≥ 4 (number of serial units)	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, M-RAM MLAB, M9K, M144K, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	N/A

Table A–1. Device Structures Supported by FIR Compiler (Part 2 of 2)

Structure	Sub Structure	Input Bit Width	Flow Control	Data Storage	Coefficient Storage	Multiple Coefficient Sets	Symmetric Coefficient
Parallel	**Fixed Coefficient**	2 to 32 for signed, 1 to 32 for unsigned	Avalon-ST interface and optional global clock enable	Logic cell, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	Yes
	Reloadable Coefficient	2 to 32 for signed, 1 to 32 for unsigned	Avalon-ST interface and optional global clock enable	Logic cell, Auto	M512, M4K, M512, M4K, MLAB, M9K, Auto	Yes	Yes
	Interpolation (Reloadable Coefficient must be in memory)	4 to 32	Avalon-ST interface and optional global clock enable	Logic cell, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	N/A
	Decimation (Reloadable Coefficient must be in memory)	4 to 32	Avalon-ST interface and optional global clock enable	Logic cell, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	N/A
Multicycle Variable	**Clocks to compute = 1**	2 to 32 for signed, 1 to 32 for unsigned	Avalon-ST interface and optional global clock enable	Logic cell, Auto	Logic cell, if (*number of coefficient set*) >1, M512, M4K, MLAB, M9K, Auto	Yes	Yes
	Clocks to compute = 2	2 to 32 for signed, 1 to 32 for unsigned	Avalon-ST interface and optional global clock enable	Logic cell, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	Yes
	Clocks to compute = 3	2 to 32 for signed, 1 to 32 for unsigned	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, MLAB, M9K, M144K, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	Yes
	Interpolation: Clocks per output = 1	2 to 32 for signed, 1 to 32 for unsigned	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, MLAB, M9K, M144K, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	N/A
	Decimation: Clocks per input = 1	2 to 32 for signed, 1 to 32 for unsigned	Avalon-ST interface and optional global clock enable	Logic cell, M512, M4K, MLAB, M9K, M144K, Auto	Logic cell, M512, M4K, MLAB, M9K, Auto	Yes	Yes

HardCopy II Support

Preloaded RAM can be used by the other supported device families in tapped delay line (for data storage) or for coefficient storage in reloadable FIR filters. However, HardCopy® II devices do not support preloaded RAM elements.

Preloaded RAM is used by the other devices if you select **M512** or **M4K** for data storage (or if you select **Auto**, the Quartus® II software selects M512 or M4K) and the memory is automatically preloaded with zeros. This cannot be done for HardCopy II devices.

For HardCopy II devices, you need to flush the memory by preceding your real data with *n* zeros when loading the data into memory, and then discard the corresponding *n* outputs.

The formula for doing this is as follows:

n = number of channels × number of coefficients

Preloaded RAM may also be used by the other device families if you select **M512** or **M4K** for the coefficient storage of reloadable FIR filters. This RAM cannot be preloaded in HardCopy II, and you must implement the logic to initialize or update the coefficients. This can be done on-chip or by using the coefficient reordering program (**coef_esq.exe**) included in the **<*install path*>\fir_compiler\misc** directory and described in the section "Coefficient Reloading and Reordering" on page 4–4.

Compiling HardCopy II Designs

When you store your data in memory or you store your coefficients in memory and reload them, you must delete the memory initialization files described below before you can successfully compile a HardCopy II design in the Quartus II software.

If you are storing data in memory, to successfully compile your design in the Quartus II software, you must first delete the following file from the project directory before compiling your design:

<*variation name*>_zero.hex

If you store your coefficients in memory and reload them, to successfully compile your design in the Quartus II software, you must first delete the following file from the project directory before compiling your design:

<*variation name*>_coef_X.hex (where *X* is an integer)

These files are created by the FIR Compiler and are stored in the project directory you specified when you ran the FIR Compiler.

Revision History

The following table displays the revision history for this user guide.

Date	Version	Changes Made
March 2009	9.0	■ Preliminary support for Arria® II GX
November 2008	8.1	■ Full support for Stratix® III ■ Applied new technical publications style ■ Withdrawn support for UNIX
May 2008	8.0	■ Full support for Cyclone® III ■ Preliminary support for Stratix IV ■ Option to automatically select memory block size for coefficient storage ■ Separate Getting Started and Parameter settings chapters
October 2007	7.2	■ Full support for Arria™ GX
May 2007	7.1	■ Added symmetric interpolation support ■ Preliminary Arria GX support ■ Full support for Stratix II GX and HardCopy® II devices.
December 2006	7.0	■ Preliminary support for Cyclone® III
December 2006	6.1	■ Preliminary support for Stratix III ■ Updated description for Avalon® Streaming interfaces ■ Added half-band decimator filter support ■ Minor updates throughout the user guide
April 2006	3.3.1	■ Updated the performance information ■ Minor updates throughout the user guide
October 2005	3.3.0	■ Updated features and release information ■ Added information to support new features ■ Preliminary support for Stratix II GX and HardCopy II ■ Updated screenshots ■ Updated many timing diagrams in Chapter 3
December 2004	3.2.0	■ Updated screenshots ■ Updated system requirements
June 2004	3.1.0	■ Updated release information and device family support tables ■ Updated the features ■ Added OpenCore Plus description ■ Added DSP Builder support information ■ Updated the performance information ■ Enhancements include support for Cyclone II devices; DSP Builder ready

How to Contact Altera

For the most up-to-date information about Altera® products, refer to the following table.

Contact *(1)*	Contact Method	Address
Technical support	Website	www.altera.com/support
Technical training	Website	www.altera.com/training
	Email	custrain@altera.com
Product literature	Website	www.altera.com/literature
Non-technical support (General)	Email	nacomp@altera.com
(Software Licensing)	Email	authorization@altera.com

Note to table:

(1) You can also contact your local Altera sales office or sales representative.

Typographic Conventions

This document uses the typographic conventions shown in the following table.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Indicates command names, dialog box titles, dialog box options, and other GUI labels. For example, **Save As** dialog box.
bold type	Indicates directory names, project names, disk drive names, file names, file name extensions, and software utility names. For example, **\qdesigns** directory, **d:** drive, and **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Indicates document titles. For example: *AN 519: Stratix IV Design Guidelines.*
Italic type	Indicates variables. For example, *n* + 1. Variable names are enclosed in angle brackets (< >). For example, ***<file name>*** and ***<project name>*.pof** file.
Initial Capital Letters	Indicates keyboard keys and menu names. For example, Delete key and the Options menu.
"Subheading Title"	Quotation marks indicate references to sections within a document and titles of Quartus II Help topics. For example, "Typographic Conventions."
`Courier type`	Indicates signal, port, register, bit, block, and primitive names. For example, `data1`, `tdi`, and `input`. Active-low signals are denoted by suffix n. Example: `resetn`. Indicates command line commands and anything that must be typed exactly as it appears. For example, `c:\qdesigns\tutorial\chiptrip.gdf`. Also indicates sections of an actual file, such as a Report File, references to parts of files (for example, the AHDL keyword `SUBDESIGN`), and logic function names (for example, `TRI`).
1., 2., 3., and a., b., c., and so on.	Numbered steps indicate a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■ ■	Bullets indicate a list of items when the sequence of the items is not important.
	The hand points to information that requires special attention.

Visual Cue	Meaning
	A caution calls attention to a condition or possible situation that can damage or destroy the product or your work.
	A warning calls attention to a condition or possible situation that can cause you injury.
	The angled arrow instructs you to press the enter key.
	The feet direct you to more information about a particular topic.