

FIR Compiler

MegaCore Function User Guide



101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
www.altera.com

MegaCore Function Version: 3.3.1
Document Version: 3.3.1 rev 2
Document Date: April 2006

Copyright © 2006 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.



Part Number: UG-FIRCOMPILER-3.6

About This User Guide

Revision History	v
How to Contact Altera	v
Typographic Conventions	vi

Chapter 1. About This Compiler

Release Information	1-1
Device Family Support	1-1
Features	1-2
General Description	1-4
DSP Builder Support	1-7
OpenCore Plus Evaluation	1-8
Performance	1-9

Chapter 2. Getting Started

Design Flow	2-1
FIR Compiler Tutorial	2-2
Create a New Quartus II Project	2-3
Launch IP Toolbench	2-4
Step 1: Parameterize	2-5
Specify the Coefficients	2-8
Using the FIR Compiler Coefficient Generator	2-8
Loading Coefficients from a File	2-11
Creating a Blank Set of Coefficients	2-12
Analyzing the Coefficients	2-13
Specify the I/O Number Formats & Bit Widths	2-15
Choose the Architecture	2-18
Step 2: Set Up Simulation	2-24
Create MATLAB Simulation Model & Quartus II Vector Testbench	2-25
Step 3: Generate	2-26
Simulate the Design	2-29
Compile the Design	2-30
Program a Device	2-30
Set Up Licensing	2-30
Filter Design Tips	2-31

Chapter 3. Specifications

Functional Description	3-1
Number Systems & Fixed-Point Precision	3-1
Generating or Importing Coefficients	3-1

Coefficient Scaling	3-3
Symmetrical Architecture Selection	3-3
Symmetrical Serial	3-4
Coefficient Reordering	3-5
Structure Types	3-7
Multicycle Variable Structures	3-7
Parallel Structures	3-7
Serial Structures	3-8
Multibit Serial Structure	3-9
Multi-Channel Structures	3-11
Interpolation & Decimation	3-11
Implementation Details for Interpolation & Decimation Structures	3-12
Multiple Channels for Interpolation & Decimation Filters	3-14
Availability of Interpolation & Decimation Filters	3-15
Pipelining	3-15
OpenCore Plus Time-Out Behavior	3-16
Atlantic I/O Protocol	3-16
FIR Compiler Input Protocol	3-17
FIR Compiler Output Protocol	3-18
Simulation Output Files	3-21
Signals	3-21
Timing Diagrams	3-23
Parallel Timing Diagrams	3-24
Serial & Multibit Serial Timing Diagrams	3-25
Serial Timing Diagrams	3-25
Multibit Serial Timing Diagrams	3-27
Multicycle Variable Timing Diagrams	3-28
Coefficient Reloading Timing Diagrams	3-29
MegaCore Verification	3-33

FIR Compiler Supported Device Structures

Supported Device Structures	A-1
HardCopy Stratix & HardCopy II Support	A-3
Compiling HardCopy Stratix & HardCopy II Designs	A-4



About This User Guide

Revision History The table below displays the revision history for the chapters in this User Guide.

Chapter	Date	Version	Changes Made
All	April 2006	3.3.1	<ul style="list-style-type: none"> Updated the performance information Minor updates throughout the user guide
All	October 2005	3.3.0	<ul style="list-style-type: none"> Updated features and release information Added information to support new features Added HardCopy® II and Stratix® II GX Support Updated screenshots Updated many timing diagrams in Chapter 3
	December 2004	3.2.0	<ul style="list-style-type: none"> Updated screenshots Updated system requirements
	June 2004	3.1.0	<ul style="list-style-type: none"> 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, go to the Altera web site at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

Information Type	USA & Canada	All Other Locations
Technical support	www.altera.com/mysupport/	www.altera.com/mysupport/
	800-800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	+1 408-544-8767 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
Product literature	www.altera.com	www.altera.com
Altera literature services	literature@altera.com	literature@altera.com
Non-technical customer service	800-767-3753	+ 1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
FTP site	ftp.altera.com	ftp.altera.com

Typographic Conventions

This document uses the typographic conventions shown below.

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: Save As dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: f_{MAX} , iqdesigns directory, d: drive, chiptrip.gdf file.
<i>Italic Type with Initial Capital Letters</i>	Document titles are shown in italic type with initial capital letters. Example: <i>AN 75: High-Speed Board Design</i> .
<i>Italic type</i>	Internal timing parameters and variables are shown in italic type. Examples: <i>t_{PIA}</i> , <i>n + 1</i> . Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <file name>, <project name>.pdf file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of on-line help topics are shown in quotation marks. Example: “Typographic Conventions.”
Courier type	Signal and port names are shown in lowercase Courier type. Examples: data1, tdi, input. Active-low signals are denoted by suffix n, e.g., resetn. Anything that must be typed exactly as it appears is shown in Courier type. For example: c:\qdesigns\tutorial\chiptrip.gdf. Also, sections of an actual file, such as a Report File, references to parts of files (for example, the VHDL keyword BEGIN), as well as logic function names (for example, TRI) are shown in Courier.
1., 2., 3., and a., b., c., etc.	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
	Bullets are used in a list of items when the sequence of the items is not important.
	The checkmark indicates a procedure that consists of one step only.
	The hand points to information that requires special attention.
	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
	The warning indicates information that should be read prior to starting or continuing the procedure or processes
	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.

Release Information

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

Table 1–1. FIR Compiler MegaCore Function Release Information

Item	Description
Version	3.3.1
Release Date	April 2006
Ordering Code	IP-FIR
Product ID	0012
Vendor ID	6AF7

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 MegaCore Function to each Altera device family.

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

Device Family	Support
Stratix® II	Full
Stratix II GX	Preliminary
Stratix GX	Full
Stratix	Full
Hardcopy® II	Preliminary
HardCopy Stratix	Preliminary
Cyclone™ II	Full

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

Device Family	Support
Cyclone	Full
Other device families	No support

Features

- Interpolation and decimation filters are available for all architectures:
 - Fully parallel distributed arithmetic
 - Fully serial distributed arithmetic
 - Multibit serial distributed arithmetic
 - Multicycle variable structures (Stratix and Stratix II DSP blocks, Cyclone device hard multipliers, and Cyclone II embedded multiplier)
 - Full support for memory implementations (M512 and M4K used for data/coefficient storage)
- Stratix II filters implement ternary adder structures in all architectures:
 - Fully parallel distributed arithmetic
 - Fully serial distributed arithmetic
 - Multibit serial distributed arithmetic
 - Multicycle variable
- Multiple coefficient sets available in all structures
- Support for clock enable in all architectures, or Atlantic™-based flow control in single-rate serial, parallel, and multibit serial architectures
- Fully integrated FIR filter development environment
- Highly optimized for Altera devices
- DSP Builder ready
- Over 400-MHz performance in Stratix II devices
- Uses Stratix Tri-Matrix™ memory (M512, M4K, and M-RAM) for all filters, resulting in smaller FIR filters
- Precision control of chip resource utilization
 - Utilizes logic cells, M512, M4K or M-RAM for data storage
 - Utilizes M512, M4K, or logic cells for coefficient storage
 - Includes a resource estimator
- Supports a variety of filter architectures up to 2,047 taps:
 - Distributed arithmetic filters
 - Reloadable coefficients
 - Fully parallel
 - Serial
 - Multibit serial
 - Provides symmetry selection and extended pipelining

- Multiplier-based filters
 - Users choose number of clocks to compute the result
 - Supports preloading, reloading, and multiple coefficient sets
- Includes symmetric optimization to reduce the number of multipliers used in multichannel variable decimating FIR filters
- Includes a channel indicator for multichannel FIR filters
- All fixed FIR filters have reload capability when coefficients are stored in memory blocks
 - Coefficients can be updated during processing
 - Additional clock port allows updating coefficients on a separate clock cycle
- For FIR filter structures that support multiple coefficient sets:
 - For fixed coefficients, all sets can be preloaded and can switch during calculation
 - For reloaded coefficients, one set of coefficients can be reloaded while another set is in calculation
- Includes a built-in coefficient generator:
 - Supports coefficient widths from 2 to 32 bits of precision
 - Imports floating-point or integer coefficients from third-party tools
 - Supports multiple coefficient sets up to a total of 16 sets
 - Provides several coefficient scaling algorithms
 - Provides floating-point to fixed-point coefficient analysis
- Supports signed or unsigned input data widths, from 2 to 32 bits wide
- User-selectable output precision via rounding and saturation
- Simulation support:
- Verilog testbench generator
 - Includes a Verilog HDL testbench generator
 - Creates IP functional simulation models for use with Altera-supported VHDL and Verilog HDL simulators
 - Generates MATLAB simulation models and testbench
 - Generates Quartus® II vector files
 - Generates a Verilog HDL testbench for all architectures

General Description

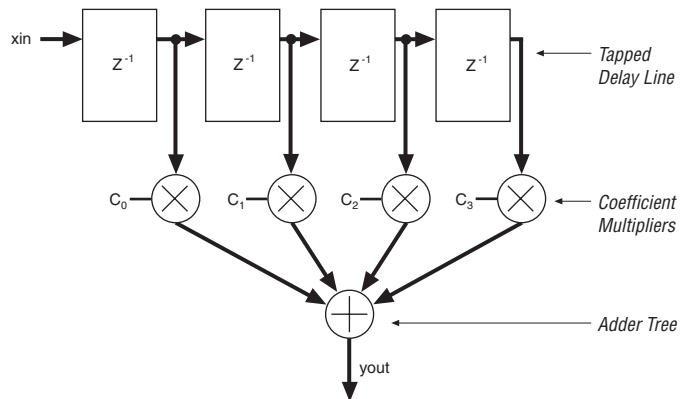
The Altera FIR Compiler MegaCore function generates finite impulse response (FIR) filters customized for Altera devices. You can use the IP Toolbench wizard interface to implement a variety of filter architectures, including fully parallel, serial, or multibit serial distributed arithmetic, and multicycle 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 (see [Figure 1-1](#)). The filter design process involves identifying coefficients that match the frequency response specified for the system. The 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.

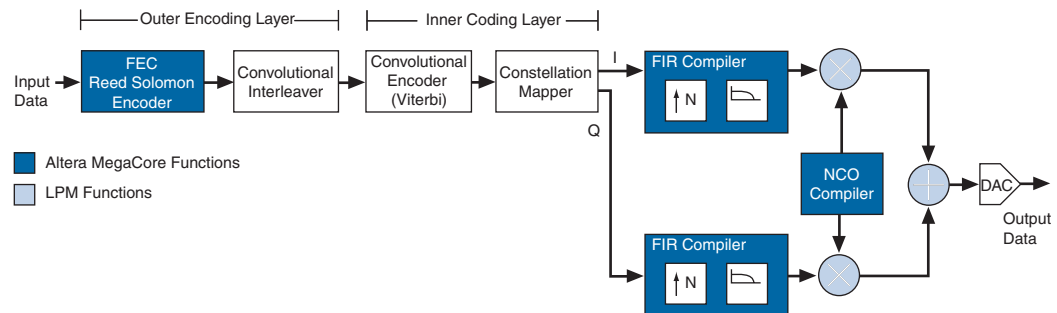
Figure 1-1. Basic FIR Filter



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. See “[DSP Builder Support](#)” on page 1–7.

Altera DSP solutions include MegaCore functions developed and supported by Altera, and Altera Megafunction Partners Program (AMPPSM) MegaCore functions. Additionally, many commonly used functions, such as adders and multipliers, are available from the industry-standard library of parameterized modules (LPM). [Figure 1–2](#) shows a typical DSP system that uses Altera MegaCore functions and LPM functions.

Figure 1–2. 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 the resource usage and performance for different implementations of a 120-tap FIR filter with a 12-bit data input bus.

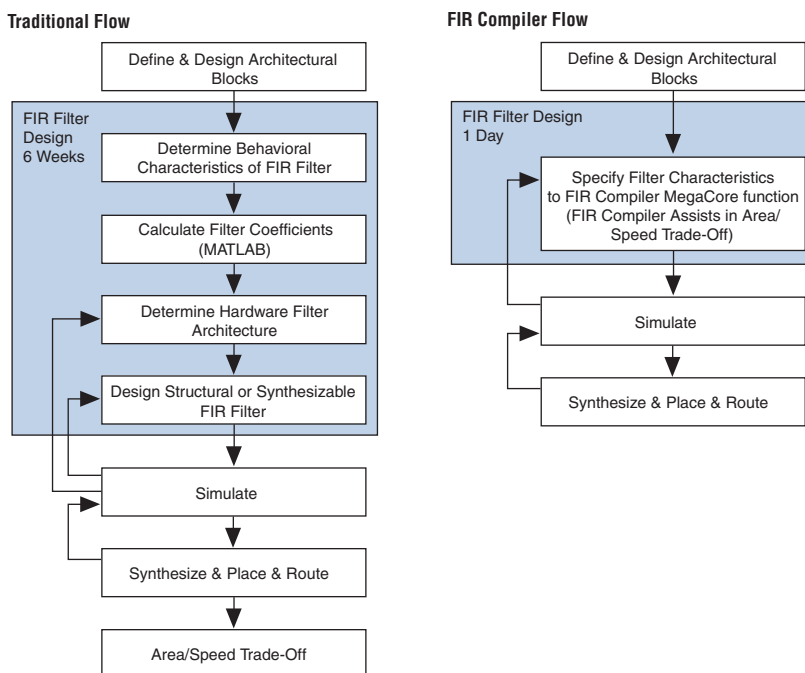
<i>Table 1–3. FIR Filter Implementation Comparison</i> <i>Note (1)</i>		
Device	Implementation	Clock Cycles to Compute Result
DSP processor	1 MAC	120
FPGA	1 serial filter	12
	1 parallel filter	1

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 function 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.
- Creating Quartus II test vectors to test the FIR filter’s impulse response.
- Generating a Verilog HDL testbench for all architectures.

Figure 1–3 compares the design cycle using the FIR Compiler MegaCore function versus a traditional implementation.

Figure 1–3. Design Cycle Comparison

DSP Builder Support

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

You can combine existing MATLAB/Simulink blocks with Altera DSP Builder/MegaCore blocks to verify system level specifications and generate hardware implementations. After installing this MegaCore function, a Simulink symbol of this MegaCore function appears in the Simulink library browser in the MegaCore library from the Altera DSP Builder blockset. To use this MegaCore function with DSP Builder, you require DSP Builder v6.0 or higher and the Quartus II software version 6.0 or higher.



For more information on DSP Builder, refer to the *DSP Builder User Guide* and the *DSP Builder Reference Manual*.

OpenCore Plus Evaluation

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

- Simulate the behavior of a MegaCore function 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 MegaCore function when you are completely satisfied with its functionality and performance, and want to take your design to production.



For more information on OpenCore Plus hardware evaluation using the FIR Compiler, see [“OpenCore Plus Time-Out Behavior” on page 3–16](#) and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Performance

Table 1–4 shows the FIR Compiler function’s performance using the Quartus II software, version 6.0, a FIR filter with 97 taps, 8-bit input data, and 14-bit coefficients.

Table 1–4. Stratix II, Stratix, & Cyclone II Device Performance (Part 1 of 2) *Note (1)*

Device	Filter Type	Pipeline Level	LEs or ALUTs (2)	M512	M4K	18 x 18 Multipliers	f_{max}	Throughput (MSPS)	Processing Equivalent (GMACs) (3)
Stratix II	Serial (M4K, M512) (4)	1	295	0	8	0	480	53	5.46
	Parallel (M4K) (4)	1	2,176	0	45	0	424	424	43.7
	Parallel (LE) (4)	1	3,935	0	0	0	383	383	39.5
	Parallel (M512) (4)	1	1,791	72	0	0	361	361	37.2
	Multicycle variable (1 cycle) (4)	2	3,639	0	0	49	356	356	36.7
	Multicycle variable (4 cycle) (4)	2	1,083	0	12	16	392	98	10.1
	Multicycle variable (1 cycle) interpolation by 4	2	2,032	0	0	25	393	393	40.5
	Multicycle variable (1 cycle) decimation by 4 (4)	2	1,512	2	5	13	383	383	39.5

Table 1–4. Stratix II, Stratix, & Cyclone II Device Performance (Part 2 of 2) *Note (1)*

Device	Filter Type	Pipeline Level	LEs or ALUTs (2)	M512	M4K	18 x 18 Multipliers	f_{max}	Throughput (MSPS)	Processing Equivalent (GMACs) (3)
Cyclone II	Serial (M4K) (4)	1	304	N/A	8	0	309	34.3	3.54
	Parallel (M4K) (4)	1	2,181	N/A	45	0	271	271	27.9
	Parallel (LE) (4)	1	3,990	N/A	0	0	253	253	26.1
	Multicycle variable (1 cycle) (4)	2	4,767	N/A	0	49	222	222	22.9
	Multicycle variable (4 cycle) (4)	2	1,385	N/A	12	13	256	64	6.6
	Multicycle variable (1-cycle) interpolation by 4	2	2,579	N/A	0	50	263	263	27.1
	Multicycle variable (1-cycle) decimation by 4 (4)	2	1,830	N/A	4	13	257	257	26.5

Notes to Table 1–4:

- (1) The performance results were obtained via Quartus II, version 6.0 compilation, with a target f_{MAX} set to 400 MHz.
- (2) Stratix II devices use adaptive look-up tables (ALUTs); other devices use logic elements (LEs).
- (3) GMAC = giga multiply accumulates per second; 1 GMAC = 1,000 million multiply accumulates per second (MMACs).
- (4) This FIR filter takes advantage of symmetric coefficients.

Design Flow

To evaluate the FIR Compiler MegaCore Function using the OpenCore® Plus feature include these steps in your design flow:

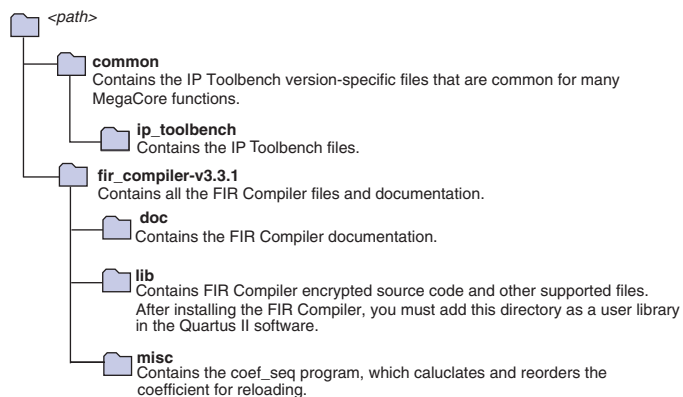
1. Obtain and install the FIR Compiler MegaCore Function.



For installation instructions, refer to the *FIR Compiler MegaCore Function vCompiler Version 3.3.1 Release Notes*.

Figure 2–1 shows the directory structure after you install the FIR Compiler MegaCore Function, where *<path>* is the installation directory.

Figure 2–1. Directory Structure



2. Create a custom variation of the FIR Compiler MegaCore Function using IP Toolbench.



IP Toolbench is a toolbar from which you quickly and easily view documentation, specify parameters, and generate all of the files necessary for integrating the parameterized MegaCore Function into your design.

3. Implement the rest of your design using the design entry method of your choice.
4. Use the IP Toolbench-generated IP functional simulation model to verify the operation of your design.



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

5. Use the Quartus II software to compile your design.



You can also generate an OpenCore Plus time-limited programming file, which you can use to verify the operation of your design in hardware.

6. Purchase a license for the FIR Compiler MegaCore Function.

After you have purchased a license for the FIR Compiler MegaCore Function, the design flow requires these additional steps:

1. Set up licensing.
2. Generate a programming file for the Altera® device(s) on your board.
3. Program the Altera device(s) with the completed design.
4. Perform design verification.

FIR Compiler Tutorial

This tutorial explains how to create a FIR Compiler MegaCore function using the Altera FIR Compiler IP Toolbench and the Quartus II software. When you finish generating a custom variation of the FIR Compiler MegaCore function, you can incorporate it into your overall project.

This tutorial involves the following steps:

- Create a New Quartus II Project
- Launch IP Toolbench
- Step 1: Parameterize
- Step 2: Set Up Simulation
- Step 3: Generate

Create a New Quartus II Project

You need to create a new Quartus II project with the **New Project Wizard**, which specifies the working directory for the project, assigns the project name, and designates the name of the top-level design entity. To create a new project follow these steps:

1. Choose **Programs > Altera > Quartus II <version>** (Windows Start menu) to run the Quartus II software. You can also use the Quartus II Web Edition software.
2. Choose **New Project Wizard** (File menu).
3. Click **Next** in the **New Project Wizard Introduction** (the introduction does not display if you turned it off previously).
4. In the **New Project Wizard: Directory, Name, Top-Level Entity** page, enter the following information:
 - a. Specify the working directory for your project. For example, this tutorial uses the `c:\altera\temp\fir` directory.
 - b. Specify the name of the project. This tutorial uses `fir_example` for the project name.



The Quartus II software automatically specifies a top-level design entity that has the same name as the project.

5. Click **Next** to close this page and display the **New Project Wizard: Add Files** page.



When you specify a directory that does not already exist, a message asks if the specified directory should be created. Click **Yes** to create the directory.

6. For Linux and Solaris operating systems, add user libraries by following these steps on the **New Project Wizard: Add Files** page:
 - a. Click **User Library Pathnames**.
 - b. Type `<path>\fir_compiler-<version>\lib` into the **Library name** box, where `<path>` is the directory in which you installed the FIR Compiler.
 - c. Click **Add to** add the path to the Quartus II project.
 - d. Click **OK** to save the library path in the project.

7. Click **Next** to close this page and display the **New Project Wizard: Family & Device Settings** page.
8. On the **New Project Wizard: Family & Device Settings** page, choose the target device family in the **Family** list.
9. The remaining pages in the **New Project Wizard** are optional. Click **Finish** to complete the Quartus II project.

You have finished creating your new Quartus II project.

Launch IP Toolbench

To launch IP Toolbench in the Quartus II software, follow these steps:

1. Start the MegaWizard® Plug-In Manager by choosing **MegaWizard Plug-In Manager** (Tools menu). The **MegaWizard Plug-In Manager** dialog box displays.

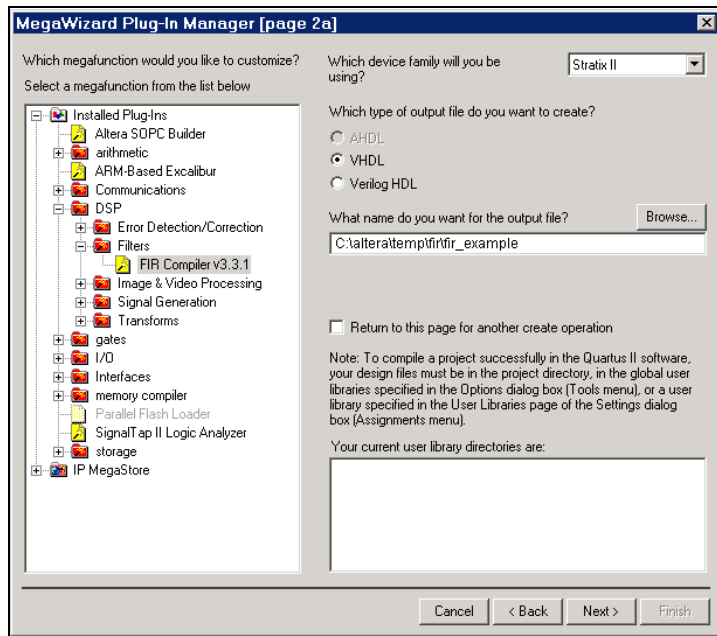


Refer to Quartus II Help for more information on how to use the MegaWizard Plug-In Manager.

2. Specify that you want to create a new custom megafunction variation and click **Next**.
3. Expand the **DSP > Filters** directory then click **FIR Compiler vCompiler Version 3.3.1**.
4. Select the output file type for your design; the wizard supports VHDL and Verilog HDL. This tutorial uses a **VHDL** file called **fir_example**.
5. The MegaWizard Plug-In Manager shows the project path that you specified in the **New Project Wizard**. Append a variation name for the MegaCore function output files `<project path>\<variation name>`. [Figure 2-2](#) shows the wizard after you have made these settings.



For designs using HardCopy devices, see [“Compiling HardCopy Stratix & HardCopy II Designs”](#) on page A-4

Figure 2–2. Select the MegaCore Function

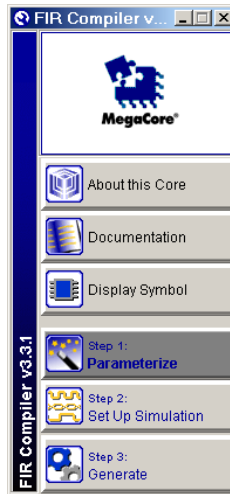
6. Click **Next** to launch IP Toolbench.

Step 1: Parameterize

To parameterize your MegaCore function, follow these steps:

1. Click **Step 1: Parameterize** in IP Toolbench (see [Figure 2–3](#)).

Figure 2-3. IP Toolbench—Parameterize



2. The **Parameterize** window is displayed (see [Figure](#)).

Figure 2-4. Parameterize Window

Parameterize - FIR Compiler MegaCore Function

Coefficients Specification - (Low Pass Set [1])

New Coefficient Set Edit Coefficient Set Remove Coefficient Set

Low Pass Set [1]

Plot Option: Fixed/Floating Coefficients Dark Background

Frequency Response Time Response & Coefficient Values

Coefficients Scaling: Auto Bit Width: 8

Architecture Specification

Device Family: Stratix II Force Non-Symmetric Structure

Structure: Distributed Arithmetic : Fully Parallel Filter

Pipeline Level: 1

Data Storage: Logic Cells Multiplier Implementation: Logic Cells

Coefficient Storage: Logic Cells Coefficients Reload Use Single Clock

Resource Estimates

Resource	Utilization
Logic Cells	947
M512	0
M4K	0
M-RAM	0
Multipilers	0

Based on Quartus II 5.1

Throughput

Input data must be valid for 1 clock period
Output data will be valid for 1 clock period
Output data is updated every clock period

Info: Force non-symmetric structure is selectable only if coefficients reload is selected.
Info: Coefficients reload is enabled only when coefficient storage is set to "M512" or "M4K".
Info: Structure "Distributed Arithmetic : Fully Serial Filter" requires Input Bit Width to be greater or equal to 4.

Cancel Finish

Specify the Coefficients

A FIR filter is defined by its coefficients. The FIR Compiler has several options for obtaining coefficients, as follows:

- You can use the FIR Compiler to generate coefficients. The FIR Compiler coefficient generator supports a variety of filter types.
- 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.
- You can generate a blank set of coefficients (initialized to zero) if you want to use a variable filter that has dynamically generated coefficients. In this case, the coefficients are generated in another application and are loaded into the filter.

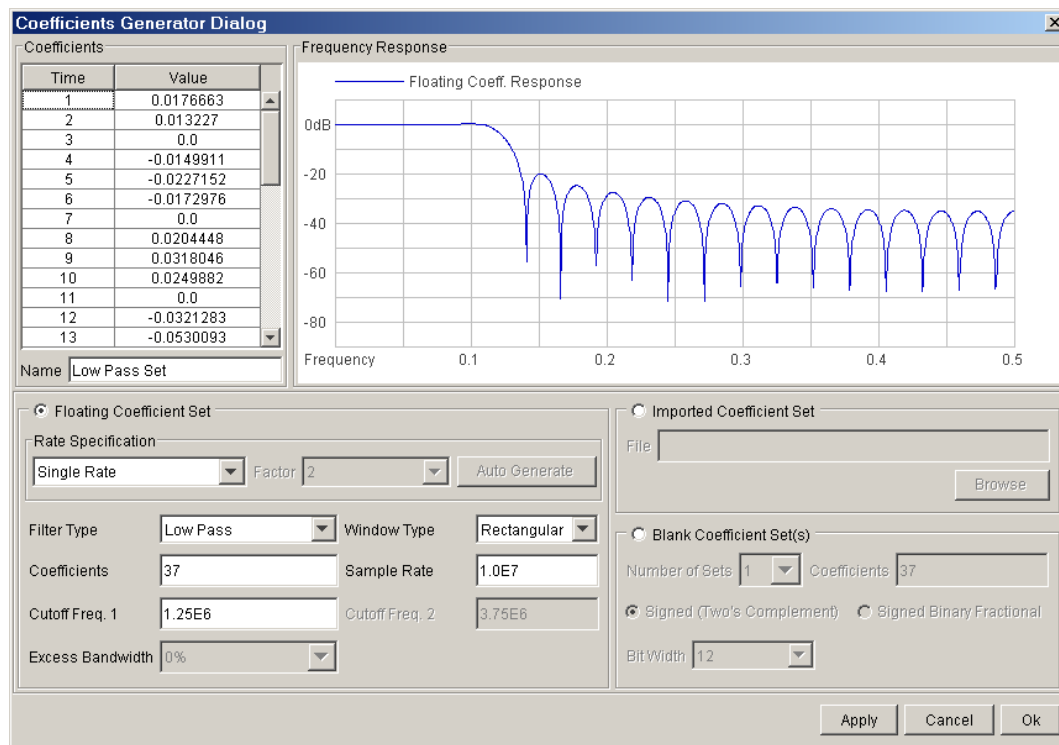
For this design tutorial you will generate coefficients using the FIR Compiler's coefficient generator. [Figure](#) shows the FIR Compiler's starting page in which you either specify a **New Coefficient Set** or you can edit the default coefficient set (**Edit Coefficient Set**).

The sections of this chapter titled [“Loading Coefficients from a File”](#) on [page 2-11](#) and [“Creating a Blank Set of Coefficients”](#) on [page 2-12](#) describe the other two options.

Using the FIR Compiler Coefficient Generator

Click **New Coefficient Set** to open the **Coefficients Generator Dialog** box. You can specify a number of parameters for the coefficients, including the filter type, window type, sample rate, excess bandwidth (for use with cosine filters), etc. See [Figure 2-5](#) on [page 2-9](#), which shows the default values for a low pass filter.

Figure 2–5. Coefficients Generator Dialog Box Shows Default Low Pass Filter Parameters

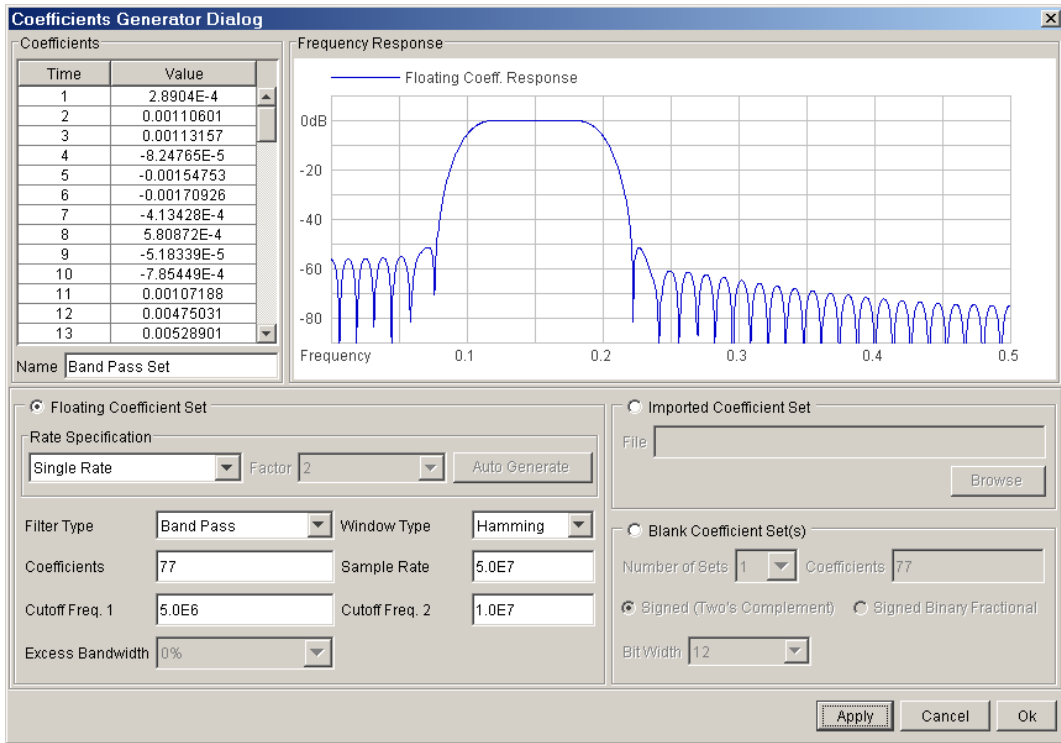


To generate the coefficients for the simple parallel filter in this tutorial, make the following settings in the **Coefficients Generator Dialog** box, as shown in [Figure 2–6 on page 2–10](#).

- **Filter Type:** Band Pass
- **Coefficients:** 77
- **Cutoff Freq (1):** 5e+006
- **Window Type:** Hamming
- **Sample Rate:** 50e+006
- **Cutoff Freq (2):** 10e+006

After you choose your settings, click **Apply**. The FIR Compiler graphically displays the frequency response of the filter in blue. The IP Toolbench also lists the actual coefficient values. See [Figure 2–6 on page 2–10](#).

Figure 2–6. Parallel FIR Filter Coefficient Parameters for Band Pass Filter

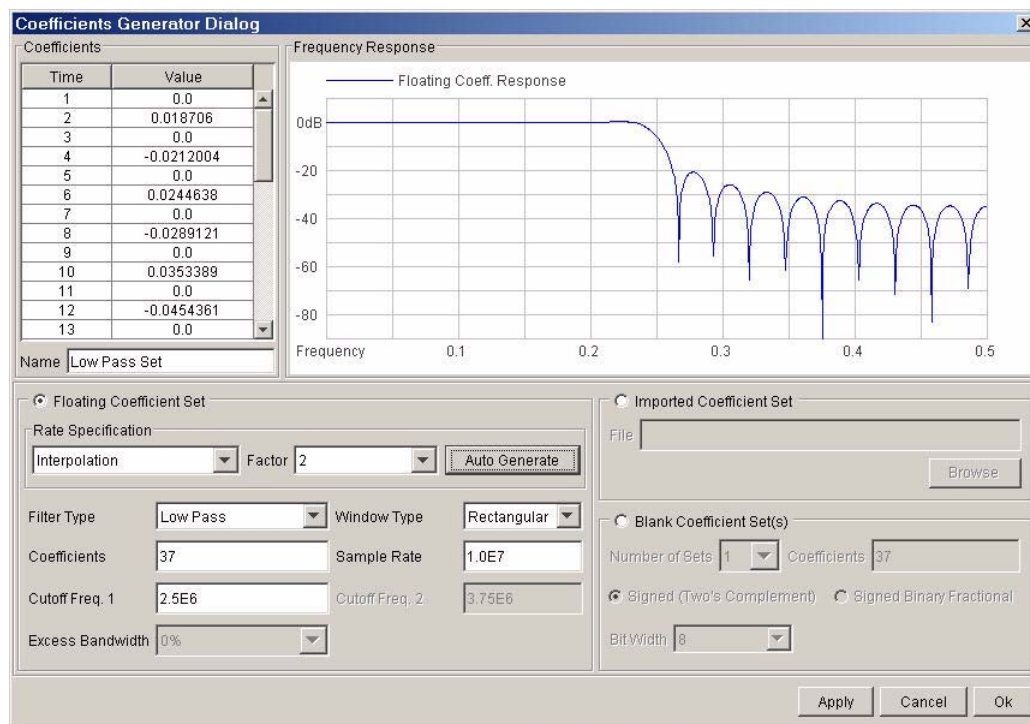


To quickly generate floating-point coefficients for interpolation or decimation rate filters, choose **Interpolation** or **Decimation** and the **Factor** from the **Rate Specification** drop-down boxes. When you click **Auto Generate**, IP Toolbench generates coefficients for a low-pass filter with a cutoff frequency based on the same rate. See [Figure 2–7 on page 2–11](#)



Refer to “[Interpolation & Decimation](#)” on page 3–11 for an explanation of interpolation and decimation.

Figure 2–7. Low-Pass Filter Results for an Interpolation Filter

**Note to Figure :**

- (1) The radio buttons for the Floating Coefficient Set, Imported Coefficient Set, and Blank Coefficient Set(s) parameters are linked together, i.e., selecting one disables the others.

3. Click **Ok** when you are finished making the parameter settings.

The floating point coefficients are displayed in the **Coefficient Specification** section of the **Parameterize FIR Compiler** page (see [Figure 2–7](#)). The FIR Compiler displays the frequency response of the floating coefficient in blue and the frequency response of the fixed coefficient in green.

Loading Coefficients from a File

To load a coefficient set from a file, click the **New Coefficient Set** (refer to [Figure on page 2–7](#)); then select **Imported Coefficient Set** from the **Coefficients Generator Dialog** box. 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.

Creating a Blank Set of Coefficients

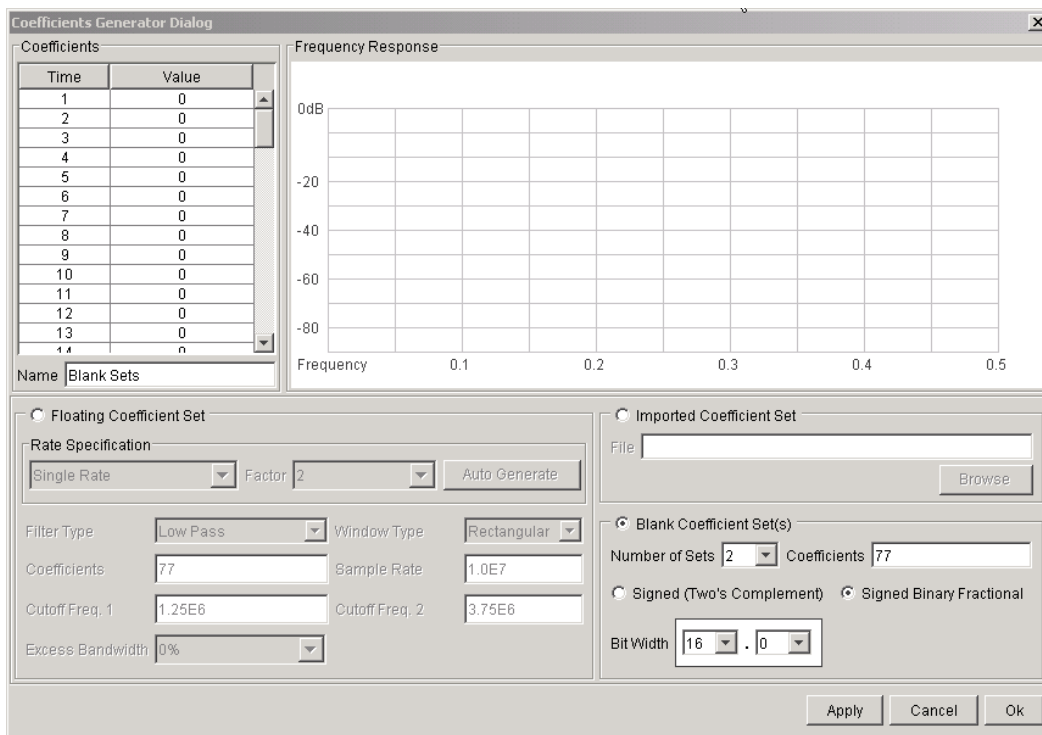
When you create a blank coefficient set, you specify the rough details about the coefficients, such as how many coefficients and how many sets. The FIR Compiler will generate a structure that supports the coefficients.



You cannot use the FIR Compiler coefficient analysis tool on blank sets of coefficients.

To create a blank coefficient set, perform the following steps:

1. From the FIR Compiler's starting page (see [Figure on page 2-7](#)), click **New Coefficient Set**, and then choose **Blank Coefficient Set(s)** from the **Coefficients Generator Dialog** box.
2. Specify information about the blank set, such as the number of sets and coefficients, which number system to use, and bit width. See [Figure 2-8](#).

Figure 2–8. Generate a Blank Coefficient Set

3. Click **Ok** after you make your selections.

The **Coefficients Specification** window (Figure 2–10 on page 2–16) is displayed.

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. See Figure 2–9 on page 2–15.

The FIR Compiler supports up to 16 sets of coefficients. You can switch between sets using the coefficient tabs in the **Coefficients Specification** window (the coefficient sets are numbered, e.g., **Low Pass Set 1**, etc.). When you select a set, the FIR Compiler displays the frequency response of the floating-point coefficients in blue, and the frequency response of

the fixed-point coefficients in green. The FIR Compiler also displays the actual coefficient values. To view the coefficient values, click 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 $\langle \text{sign} \rangle \langle \text{integer bits} \rangle . \langle \text{fractional bits} \rangle$. A signed binary fractional number is interpreted as shown below, and in the following equation.

$\langle \text{sign} \rangle \langle x_1 \text{ integer bits} \rangle$	$\cdot \langle y_1 \text{ fractional bits} \rangle$	Original input data
$\langle \text{sign} \rangle \langle x_2 \text{ integer bits} \rangle$	$\cdot \langle y_2 \text{ fractional bits} \rangle$	Original coefficient data
$\langle \text{sign} \rangle \langle i \text{ integer bits} \rangle$	$\cdot \langle y_1 + y_2 \text{ fractional bits} \rangle$	Full precision (after FIR calculations)
$\langle \text{sign} \rangle \langle x_3 \text{ integer bits} \rangle$	$\cdot \langle y_3 \text{ fractional bits} \rangle$	Output data (after limiting precision)

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

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 $0.333n$ 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 choose.

- For this tutorial, make the following selections in the **Coefficients Specification** area:

- **Coefficients Scaling:** Auto
- **Coefficient Bit Width:** 12



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* (i.e., where each effective bit provides 6dB of SNR).

Figure 2–9 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 2–9. Analyzing the Coefficients

Parameterize - FIR Compiler MegaCore Function

Coefficients Specification - (Low Pass Set [1])

New Coefficient Set Edit Coefficient Set Remove Coefficient Set

Low Pass Set [1] Low Pass Set [2] Low Pass Set [3] Blank Sets [4..5] Blank Set-6

Plot Option Fixed/Floating Coefficients Dark Background

— Floating Coeff. Response — Fixed Coeff. Response

0dB

-20

-40

-60

-80

Frequency 0.1 0.2 0.3 0.4 0.5

Frequency Response Time Response & Coefficient Values

Coefficients Scaling Auto Bit Width 12

Architecture Specification

Device Family Stratix II Force Non-Symmetric Structure

Structure Variable/Fixed Coefficient: Multi-Cycle

Pipeline Level 3 Clocks to Compute 1

Data Storage Logic Cells Multiplier Implementation Logic Cells

Coefficient Storage Logic Cells Coefficients Reload Use Single Clock

Resource Estimates

Resource	Utilization
Logic Cells	41496
M512	0
M4K	0
M-RAM	0
Multipliers	0

Based on Quartus II 5.1

Throughput

Input data must be valid for 1 clock period

Output data will be valid for 1 clock period

Output data is updated every clock period

Info: Force non-symmetric structure is always set for coefficient sets with unequal number of coefficients.

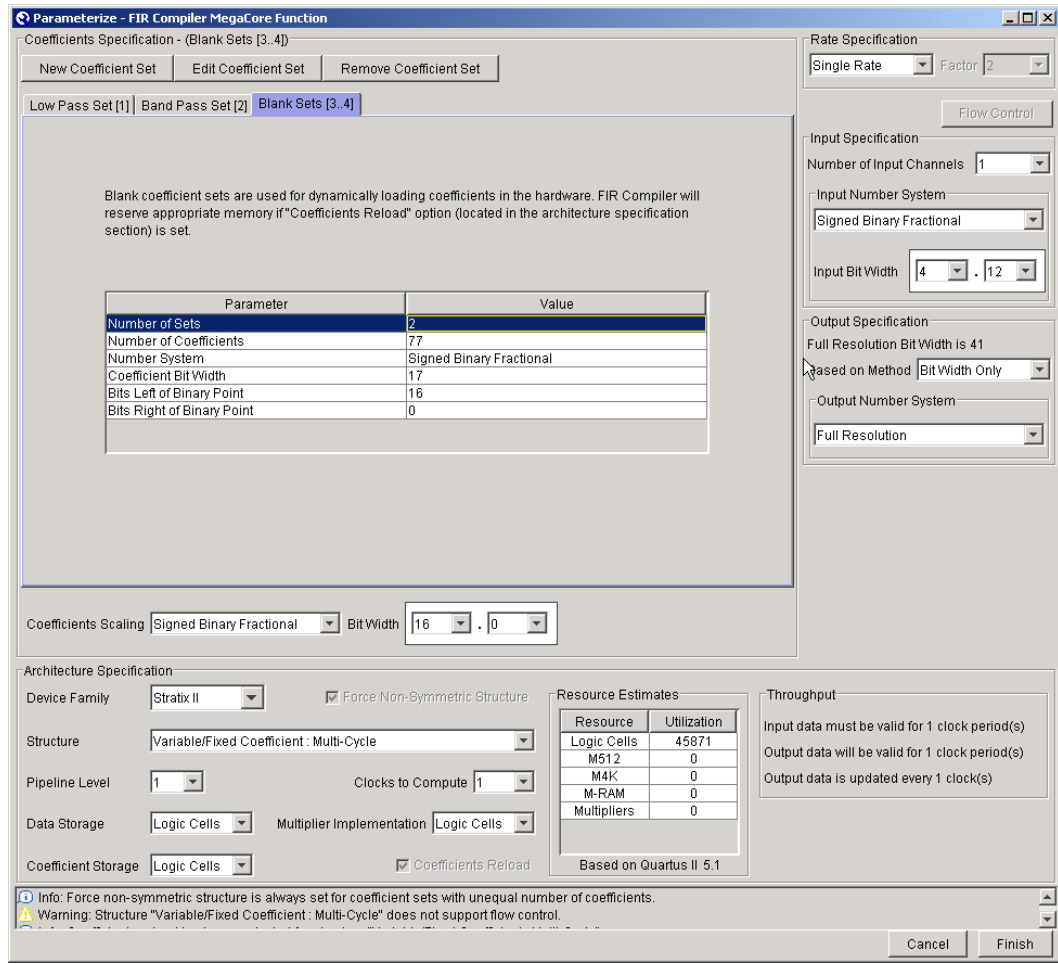
Info: Single clock coefficient reload is always selected for structure "Variable/Fixed Coefficient: Multi-Cycle" and coefficient storage set to "Logic Cells".

Cancel Finish

Specify the I/O Number Formats & Bit Widths


You can specify the number of input channels (i.e., how many data streams will generate an output for each stream) and the input number system in the **Coefficients Specification** window, see [Figure 2–10](#).

Figure 2–10. Coefficients Specification



The FIR Compiler calculates how many bits your filter requires for full resolution using two methods: actual coefficients or the coefficient bit widths. These two 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.

You can use full or limited precision for the filtered output (out). To use full precision, leave the **Full Resolution** option (Output Number System) turned on (default). To limit the precision, pull down the drop box and select another resolution—other than **Full Resolution**.

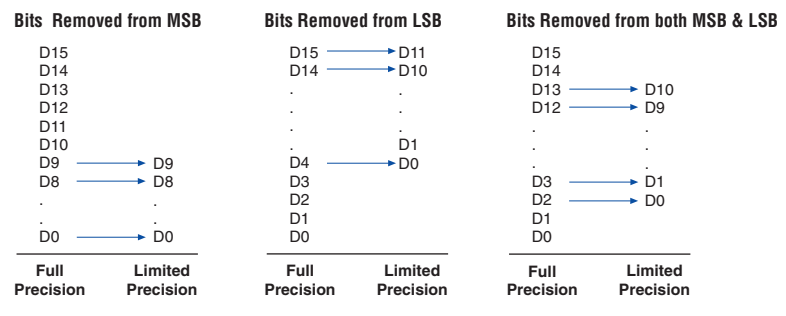
 If your filter has coefficient reloading or multiple sets of coefficients, you must select **Bit Widths Only**.

The FIR Compiler gives you the option of truncating or saturating the most significant bit (MSB) and/or rounding or truncating the least significant bit (LSB). Saturation, truncation, and rounding are non-linear operations. Table 2-1 shows the options for limiting the precision of your filter.

Bit Range	Option	Result
MSB	Truncate	In truncation, the filter disregards specified bits. See Figure 2-11.
	Saturate	In saturation, if the filtered output is greater than the maximum positive or negative value able to 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 2-11 shows an example of removing bits from the MSB and LSB.

Figure 2-11. Removing Bits from the MSB and LSB



Instead of using the options shown in [Table 2-1](#), you can use signed binary fractional notation to limit the number of bits. 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, shown in the **Coefficients Specification** window in [Figure 2-10](#).

You are now ready to choose the architecture parameters from the lower half of the **Coefficients Specification** window.

Choose the Architecture

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

[Table 2-2](#) describes the relative trade-offs for the different architecture options.

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 Cyclone II 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 and Stratix II 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.



Refer to “[Functional Description](#)” on page 3–1 for more information about the filter architectures and how they operate. Also, see [Figure 2–12](#) on page 2–23.

[Tables 2–3](#) through [2–6](#) describe the FIR Compiler options that are available for each architecture.

Parameter	Description
Clocks to Compute	Use this option to indicate 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	Use this option to indicate which device resources to use for data storage. You can choose Logic Cells , M512 , M4K , MRAM , or Auto . If you choose 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 choose 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	Use this option to indicate which device resources to use for coefficient storage. You can choose Logic Cells , M512 , or M4K . The option list changes depending on which device you choose 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	Use this option to indicate which device resources to use to implement the multiplier. You can choose Logic Cells , DSP Blocks , and Auto . If you choose Auto , the Quartus II software turns on the DSP Block Balancing logic option. Using embedded DSP blocks for multipliers results in a smaller and faster design in a device with enough DSP blocks for all 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 models and designs with DSP Builder. This option is only available when the Coefficients Reload option is selected and M512 or M4K is specified in the Coefficient Storage option. This option ties the <code>coef_clk_in</code> and <code>clk</code> signals together.

Note to Table 2–3:

- (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 2–4. Fully Serial Filter Architecture *Note (1)*

Parameter	Description
Data Storage	Use this option to indicate which device resources to use for data storage. You can choose Logic Cell or Auto . If you choose 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	Use this option to indicate which device resources to use for coefficient storage. You can choose Logic Cell , M512 , or M4K . If you choose the M512 or M4K option, you can also choose to reload coefficients. The option list changes depending on which device you choose. Choosing 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. 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.

Note to Table 2–4:

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

Table 2–5. Multibit Serial Filter Architecture (Part 1 of 2) *Note (1)*

Parameters	Description
Number of Serial Units	Use this option to indicate the number of serial units needed to make the filter. You can choose 2 , 3 , or 4 . The calculation cycles of each result are reduced to one n th of the corresponding serial filter, where n is the number of serial units. Correspondingly, there is an increase in resource utilization.
Data Storage	Use this option to indicate which device resources to use for data storage. You can choose Logic Cell , M512 , M4K , MRAM , or Auto . If you choose Auto , the Quartus II software chooses 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 choose and whether you choose multi-rate (interpolation or decimation). Choosing embedded memory reduces logic cell usage and may increase the speed of the filter.
Coefficient Storage	Use this option to indicate which device resources to use for coefficient storage. You can choose Logic Cell , M512 , or M4K . If you choose the M512 or M4K option, you can also choose to reload coefficients. The option list changes depending on which device you choose. Choosing 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 2–5. 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.

Note to Table 2–5:

- (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 2–6. Fully Parallel Filter Architecture *Note (1)*

Parameters	Description
Data Storage	Use this option to indicate which device resources to use for data storage. You can choose Logic Cells or Auto . If you choose 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	Use this option to indicate which device resources to use for coefficient storage. You can choose Logic Cells , M512 , or M4K . If you choose the M512 or M4K option, you can also choose to reload coefficients. The option list changes depending on which device you choose. Choosing 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.

Note to Table 2–6:

- (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.

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



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.

When you are choosing the architectural structure, refer to the following tips:

- For Stratix II devices, when you use the **Variable/Fixed Coefficient: Multicycle** structure, selecting **DSP Blocks** in the **Multiplier Implementation** list box lets the FIR Compiler use embedded DSP blocks for multipliers. This setting results in a smaller and faster design in a device with enough DSP blocks for all multipliers.
- When reloading coefficients, a multicycle variable FIR filter structure has a short reloading time compared to a fixed FIR filter. Additionally, M512 blocks have a shorter reloading time than M4K blocks.
- For maximum clock speed, choose the **Distributed Arithmetic: Fully Serial Filter** structure. For Stratix II, Stratix GX, and Stratix devices, using M512 memory for coefficient and data storage is faster than using M4K or M-RAM memory. For maximum throughput, choose the **Distributed Arithmetic: Fully Parallel** structure.
- The most efficient use of the Stratix and Stratix II DSP block is for 9×9 (in groups of 8) or 18×18 (in groups of 4) multipliers.

For this tutorial, select **Distributed Arithmetic: Fully Parallel Filter** structure with a pipeline level of three. Although these settings create a filter that uses a large number of logic cells, increasing the pipeline level to three decreases the number of clock cycles to one. Thereby, greatly increasing system performance. See [Figure 2–12](#).

Figure 2–12. Specify the Filter Architecture

The screenshot shows the 'Parameterize - FIR Compiler MegaCore Function' dialog box. The 'Coefficients Specification' section includes tabs for 'New Coefficient Set', 'Edit Coefficient Set', and 'Remove Coefficient Set'. Below these are tabs for 'Low Pass Set [1]', 'Low Pass Set [2]', 'Low Pass Set [3]', 'Blank Sets [4..5]', and 'Blank Set 6'. The 'Plot Option' is set to 'Fixed/Floating Coefficients' and 'Dark Background' is unchecked. The plot shows 'Floating Coeff. Response' (blue) and 'Fixed Coeff. Response' (green) with frequency on the x-axis (0.1 to 0.5) and magnitude in dB on the y-axis (-80 to 0). The 'Coefficients Scaling' is set to 'Auto' and 'Bit Width' is 8. The 'Architecture Specification' section shows 'Device Family' as 'Stratix II', 'Structure' as 'Distributed Arithmetic: Fully Parallel Filter', 'Pipeline Level' as 3, 'Data Storage' as 'Logic Cells', and 'Multiplier Implementation' as 'Logic Cells'. The 'Resource Estimates' table is as follows:

Resource	Utilization
Logic Cells	6041
M512	0
M4K	0
M-RAM	0
Multipliers	0

The 'Throughput' section indicates: 'Input data must be valid for 1 clock period', 'Output data will be valid for 1 clock period', and 'Output data is updated every clock period'. The 'Cancel' and 'Finish' buttons are at the bottom right.

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

Step 2: Set Up Simulation

An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model file produced by the Quartus II software. The model allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

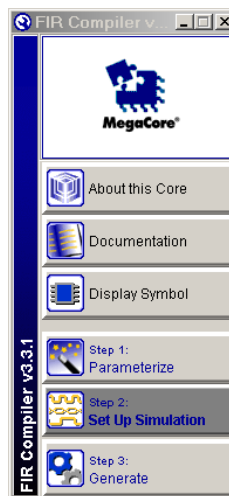


Only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis creates a nonfunctional design.

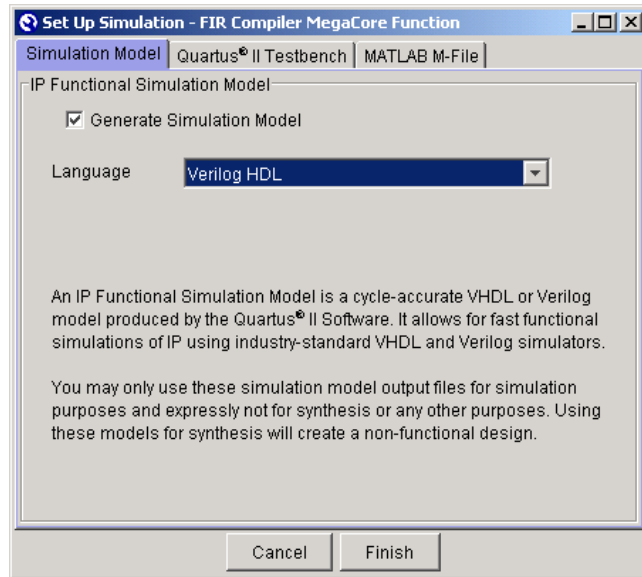
To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Click **Step 2: Set Up Simulation** in IP Toolbench (see [Figure 2-13](#)).

Figure 2-13. IP Toolbench—Set Up Simulation



2. Select the IP Functional **Simulation Model** tab (**Set Up Simulation** window).
3. Turn on **Generate Simulation Model** (see [Figure 2-14](#)).

Figure 2–14. Generate Simulation Model

4. Choose the language in the **Language** list (from the **IP Functional Simulation Model** tab).
5. Click **OK**.

Create MATLAB Simulation Model & Quartus II Vector Testbench

This section provides the steps to create MATLAB simulation models or Quartus II software vector testbench files.

To create a MATLAB simulation model and testbench, follow these steps:

1. Click the **MathWorks MATLAB** tab on the **Set Up Simulation** page.
2. Specify **Generate Mathworks MATLAB simulation model**. A testbench also is generated.

To create a Quartus II vector file testbench, follow these steps:

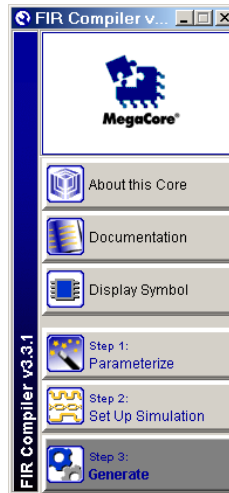
1. Click the **Quartus II Testbench** tab on the **Set Up Simulation** page.
2. Specify **Generate Quartus II Vector File**.
3. Click **Finish**.

Step 3: Generate

To generate your MegaCore function, follow these steps:

1. Click **Step 3: Generate** in IP Toolbench (see [Figure 2-15](#)).

Figure 2-15. IP Toolbench—Generate



2. The generation report lists the design files that IP Toolbench creates (see [Figure 2-16](#)). Your project directory might contain more files than listed in this report. For a comprehensive list of files that your project directory might contain, refer to [Figure 2-15 on page 2-26](#) and [Table 2-7 on page 2-27](#).

Figure 2–16. Generation Report

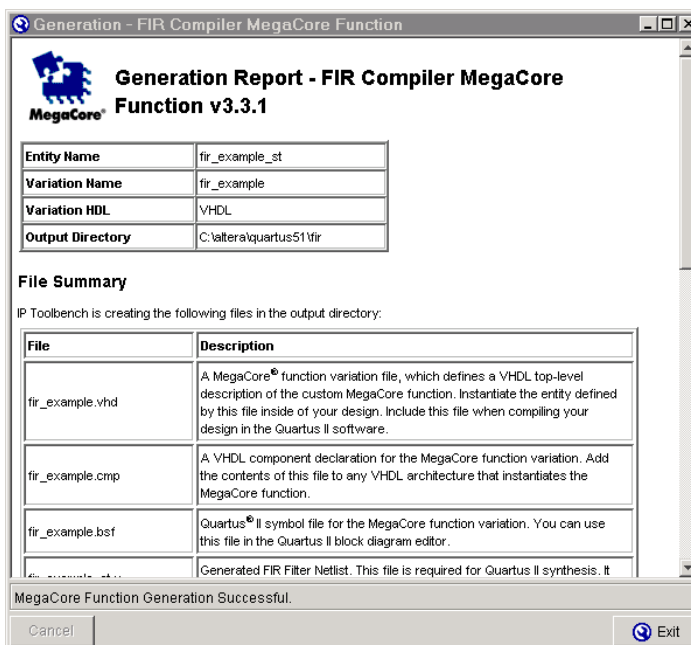


Table 2–7 describes the IP Toolbench-generated files and additional files contained in your project directory. The files in your project directory vary based on the language you choose (VHDL or Verilog HDL) and your design.

Filename	Description
altera_vhdl_support.vhd	A VHDL package that contains functions for the generated entities. This file may be shared between MegaCore functions.
<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>.bsf	A Quartus II software 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>_coef_in.txt	A text file that provides coefficient input file for Verilog HDL testbench
<variation name>_coef_in_mlab.txt	A text file that provides coefficient input file for Matlab testbench.

Table 2–7. Generated Files (Part 2 of 3) Notes (1) & (2)

Filename	Description
<variation name>_coef_int.tx	A text file that records fixed point coefficients.
<variation name>_coef_n.hex	Memory initialization files. These files are required for both simulation with IP Functional Simulation Models and Synthesis using the Quartus II software.
<variation name>_constraints.tcl	Constraints setting file for Quartus II synthesis. This file contains the necessary constraints to achieve FIR Filter size and speed
<variation name>_data_in.txt	A text file. This file provides an impulse simulation data file for Matlab or Verilog HDL testbench.
<variation name>.html	A MegaCore function report file.
<variation name>_mlab.m	A MATLAB M-File. This file provides a MATLAB simulation model for the customized FIR Variant
<variation name>_param.txt	A text file. This file records all output parameters for customized FIR variant.
<variation name>_silent_param.txt	A text file. This file records all input parameters for customized FIR variant.
<variation name>_simgen_vho.tcl	A Modelsim Tcl file. This file contains a Tcl script that will allow you to simulate the FIR Compiler generated Verilog HDL testbench using co-simulation with VHDL
<variation name>_simgen_vo.tcl	A Modelsim Tcl file. This file contains a Tcl script that will allow you to simulate the FIR Compiler- generated Verilog HDL testbench within a Verilog HDL design.
<variation name>_st_s.v	Generated FIR Filter Netlist. This file is required for Quartus II synthesis. It will be added to your current Quartus II project
<variation name>_st_u.v	Generated FIR Filter Netlist. This file is required for Quartus II synthesis. It will be added to your current Quartus II project
<variation name>_st.v	Generated FIR Filter Netlist. This file is required for Quartus II synthesis. It will be added to your current Quartus II project
<variation name>.vhd or .v	A MegaCore function variation file that defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. 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>.vo or .vho	A VHDL or Verilog HDL IP functional simulation model.
<variation name>.vec	Quartus II Vector file. This file provides simulation test vectors to be used simulating the customized FIR Variant with the Quartus II software
<variation name>_wr.v	Generated FIR Filter Netlist. This file is required for Quartus II synthesis. It will be added to your current Quartus II project
<variation name>_zero.hex	A memory initialization file. This file is required both for simulation with IP functional simulation models and for synthesis using the Quartus II software.

Table 2–7. Generated Files (Part 3 of 3) Notes (1) & (2)

Filename	Description
tb_<variation name>.m	A MATLAB M-File. This file provides a MATLAB testbench for the customized FIR Variant.
tb_<variation name>.v	A Verilog HDL file. This file provides a Verilog HDL testbench for the customized FIR variant.

Notes to Table 2–7

(1) These files are variation dependent, some may be absent or their names may change.

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

- After you review the generation report, click **Exit** to close IP Toolbench.

You can now integrate your custom MegaCore function variation into your design and simulate and compile.



If your design uses HardCopy Stratix or HardCopy II devices, refer to [“Compiling HardCopy Stratix & HardCopy II Designs” on page A-4.](#)

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 2: Set Up Simulation” on page 2–24.](#) Compile the VO or VHO file in your simulator environment to perform functional simulation of your custom variation of the MegaCore function. A Verilog HDL testbench named **tb_<variation name>.v** is also generated.

If your simulation environment is Modelsim, a related TCL script also is generated. To use this script to simulate a VO file, use the file `<variation name>_simgen_vo.tcl`. To simulate a VHO file, use the file `<variation name>_simgen_vho.tcl`.



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



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 $N = (\text{number of channels}) \times (\text{number of coefficients}) \times (\text{number of clock cycles to calculate an output})$.

To simulate in a MATLAB environment run the testbench, `tb_<variation_name>.m`, which also is located in your design directory. This testbench will give you an impulse response of the FIR.

Compile the Design

You can use the Quartus II software to compile your design. Refer to Quartus II Help for instructions on compiling your design.

Program a Device

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

With Altera's free OpenCore Plus evaluation feature, you can evaluate the FIR Compiler MegaCore function before you purchase a license. OpenCore Plus evaluation allows you to generate an IP functional simulation model and produce a time-limited programming file.



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

You can simulate the FIR Compiler MegaCore function in your design and perform a time-limited evaluation of your design in hardware.



For more information on OpenCore Plus hardware evaluation using the FIR Compiler MegaCore function, see “[OpenCore Plus Time-Out Behavior](#)” on page 3–16 and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Set Up Licensing

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

After you purchase a license for FIR Compiler MegaCore function, 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.

Filter Design Tips

This section provides some tips for using the FIR Compiler:

- To prevent high-pass filters from rolling off near Nyquist, choose 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 fastest filters are parallel filters with extended pipelining that generate an output for every clock cycle.
- In the Stratix II, Stratix, Cyclone II, and Cyclone families, Altera recommends that you use memory blocks to reduce area.

Functional Description

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.

This section discusses:

- Number Systems & Fixed-Point Precision
- Generating or Importing Coefficients
- Structure Types
- Interpolation & Decimation
- Pipelining

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. [Figure 3-1](#) shows the contents of a sample coefficient text file.



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

Figure 3–1. 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 and high-pass
- Band-pass and band-reject
- Raised cosine and root raised cosine

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, in the **Coefficients Generator Dialog** page, see [Figure on page 2–9](#).

When the FIR Compiler reads in the coefficients, it automatically detects any symmetry. The filter gives you several scaling options, e.g., 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 **Coefficients Specification** page, see [Figure 2–9 on page 2–15](#).

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:

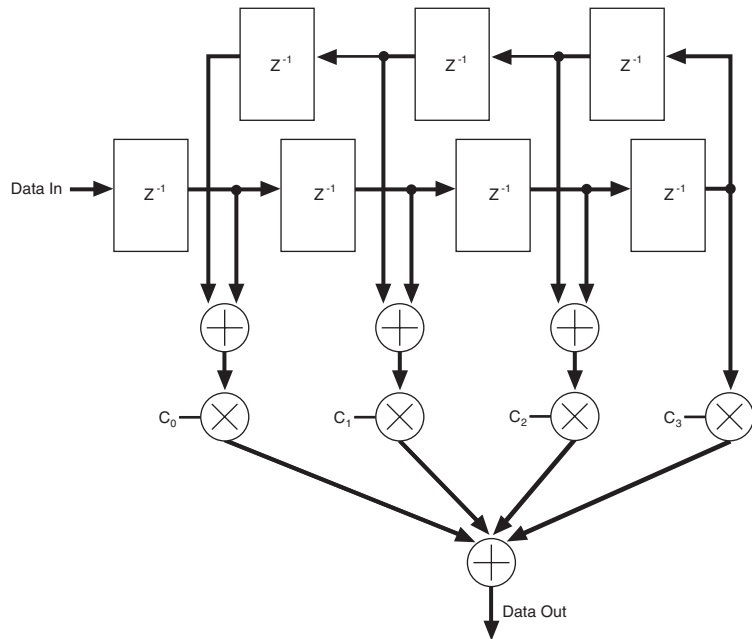
- *Auto scale to a specified number of precision bits*—Because the coefficients are represented by a certain number of bits, it is possible to apply whatever gain factor is required such that the maximum coefficient value equals the maximum possible value for a given number of bits. This approach produces coefficient values with a maximum signal-to-noise ratio.
- *Auto with a power of 2*—With this approach, the FIR Compiler chooses 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 chooses 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). Odd and even filter structures are shown in [Figures 3-2](#) and [3-3](#).



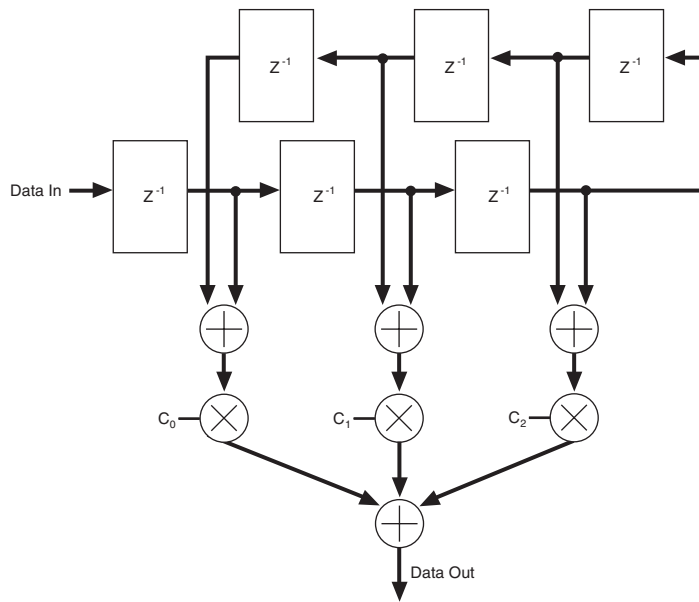
The wizard gives you the option to force non-symmetrical structures.

Figure 3–2. 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. Use the Resource Estimator to determine the best solution available (**Architecture Specification** area of the **Parameterize** page, see [Figure 2–12 on page 2–23](#)).

Figure 3–3. Six-Tap Symmetrical FIR Filter


Coefficient 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, the coefficients must be reordered during reloading. Altera® provides source code for a C++ program with FIR Compiler that reorders the coefficients. Additionally, Altera provides a precompiled Windows executable that reorders the coefficients.

The program is in `<path>\fir_compiler-v<version>\misc`. The C++ source code is named `coef_seq.cpp` and the executable program is `coef_seq.exe` (Windows operating system). You can add 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 to the input coefficient file>


```



You should include the directory path with the file name, as indicated above with the forward slash marks.

where:

- *<FIR structure>* is:
 - MCV—multicycle variable
 - SER—fully serial
 - MBS—multibit serial
 - PAR—fully parallel
- *<coefficient store>* is:
 - LC—logic cells
 - M512—M512 blocks
 - M4K—M4K blocks
- *<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 multi-rate 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** 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, see [“Coefficient Reordering” on page 3–5](#).

Multicycle Variable Structures

Multicycle variable filters are optimized for high throughput. In a multicycle variable structure, the designer specifies that the filter uses 1 to 16 clock cycles to compute a result (for any filter that fits into a single device).

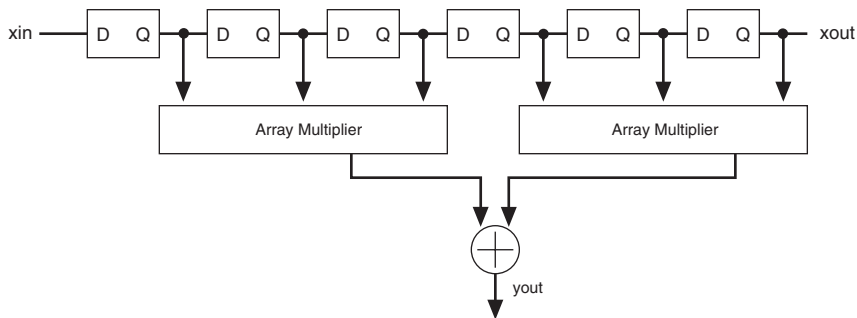
For Stratix® II and Stratix devices, if you select the multicycle variable structure, selecting **DSP Blocks** in the **Multiplier** list box lets the FIR Compiler use embedded DSP blocks for multipliers. This implementation results in a smaller and faster design in a device that contains enough DSP blocks for all multipliers.



For information on the multicycle variable signals, see [“Multicycle Variable Timing Diagrams” on page 3–28](#).

Parallel Structures

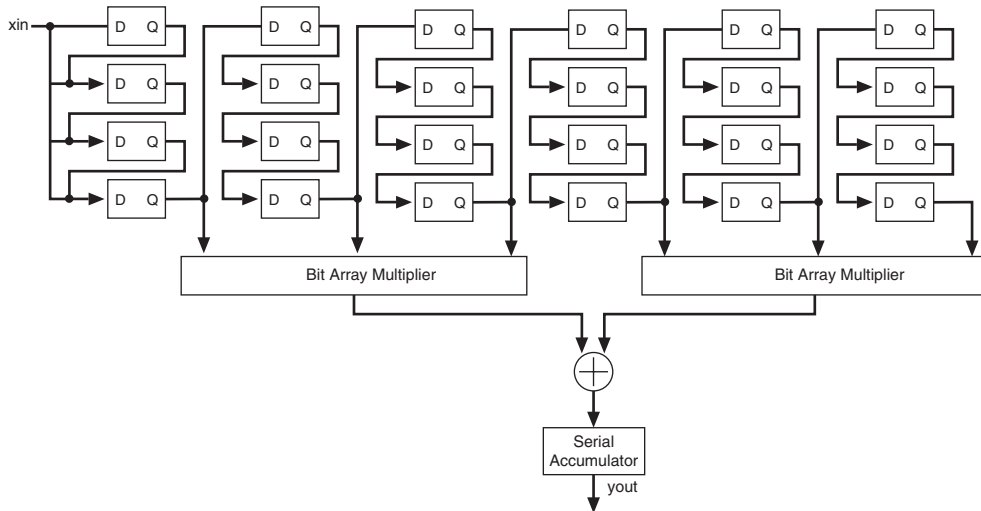
A parallel structure calculates the filter output in a single clock cycle. Parallel filters provide the highest performance and consume the largest area. Pipelining a parallel filter allows you to generate filters that run between 120 and 300 MHz at the cost of pipeline latency. [Figure 3–4](#) shows the parallel filter block diagram.

Figure 3–4. Parallel Filter Block Diagram

For information on the parallel signals, see [“Parallel Timing Diagrams”](#) on page 3–24.

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 II, Stratix GX, Stratix, Cyclone™ II, and Cyclone device families, using memory blocks for data storage will result in a significant reduction in area. [Figure 3–5](#) shows the serial filter block diagram.

Figure 3–5. Serial Filter Block Diagram

For information on the serial signals, see “Figure 3–18 shows the parallel interpolation waveform.” on page 3–24.

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 3–6 shows the multibit serial structure.

Figure 3–6. Multibit Serial Structure

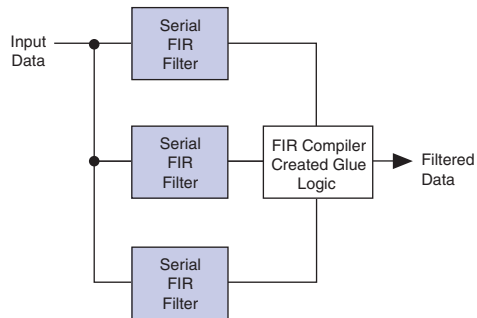
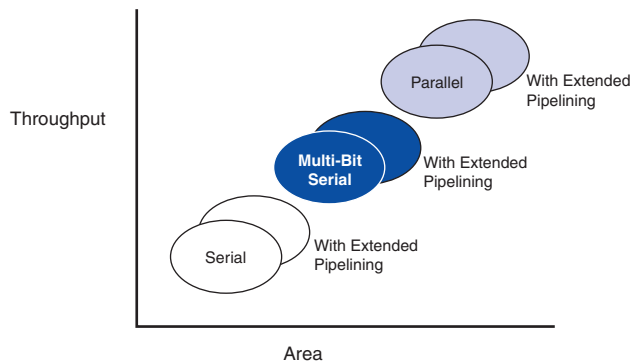


Figure 3–7 shows the area/speed “trade-off” of fixed FIR filters.

Figure 3–7. 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.

Multi-Channel 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.



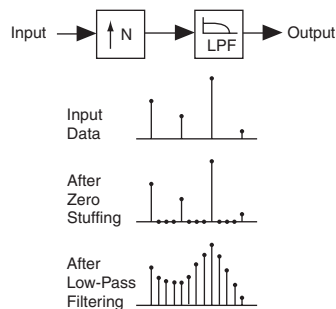
To minimize the number of logic elements, use a distributed serial arithmetic architecture, multiple channels, and memory blocks for data and coefficient storage.

Interpolation & Decimation

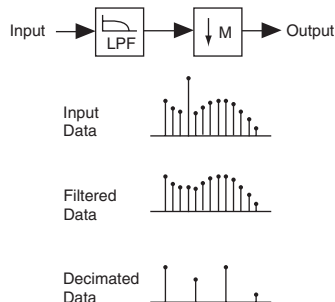
You can use the FIR Compiler to interpolate or decimate a signal. Interpolation generates extra points in between the original samples; decimation removes redundant data points. Both operations change the effective sample rate of a signal.

Mathematically, when a signal is interpolated, zeros are inserted between data points and the data is filtered to remove spectral components that were not present in the original signal. See [Figure 3-8](#).

Figure 3-8. 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. See [Figure 3-9](#).

Figure 3–9. 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. See [Figure 3–10](#).
- When using a decimation filter, output data—which is discarded during downsampling—is never computed, again potentially saving resources. See [Figure 3–11](#).

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

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

Implementation Details for Interpolation & Decimation Structures

[Figures 3–10](#) and [3–11](#) illustrate the results when applying polyphase decomposition to interpolation and decimation filters.

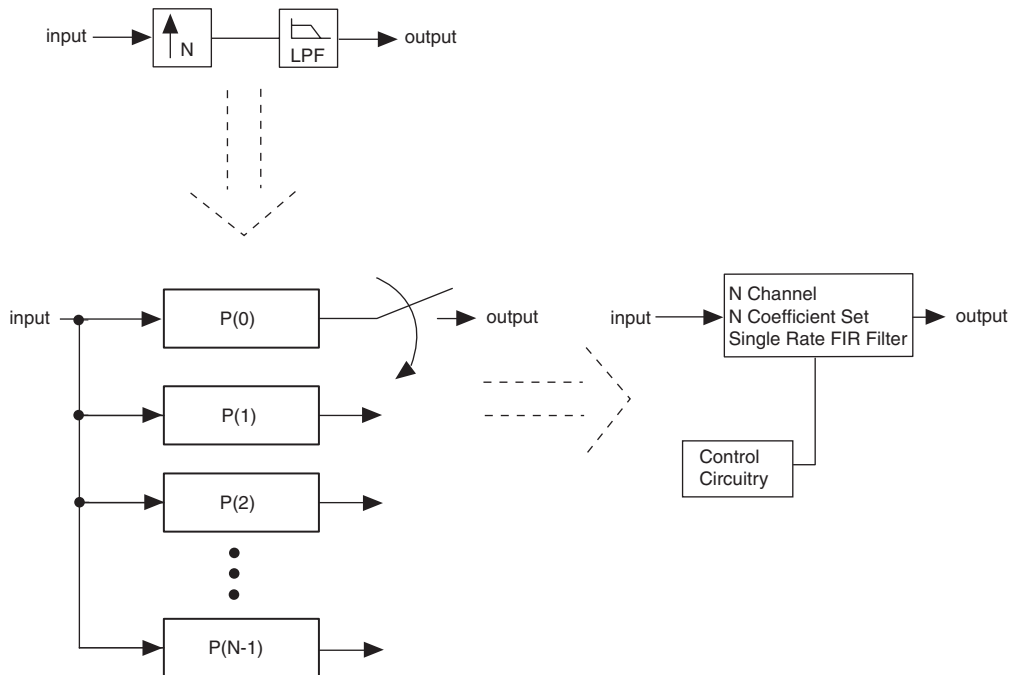
Figure 3–10. Interpolation Filter Structure

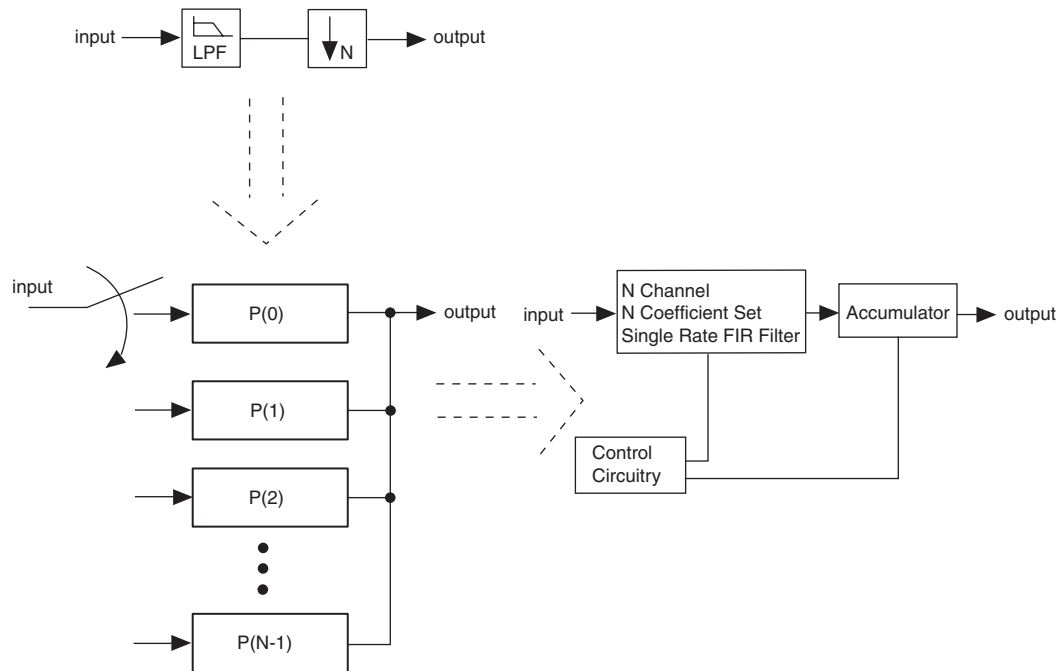
Figure 3–10 illustrates an interpolation structure. It takes a constant number of clocks to compute each polyphase output. The input data must be held for the number of clocks to compute each polyphase output multiplied by the number of polyphase units (which is the same as the interpolation factor).

Figure 3–11 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 $N = \text{number of polyphase filters} \times \text{number of clocks to compute each polyphase result}$.



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 3–11. Decimation Filter Structure



Multiple Channels for Interpolation & Decimation Filters

Consider the results when applying a multi-channel scheme to a multirate filter.

Applying a multi-rate interpolation filter to a multi-channel structure is similar to applying a single-channel interpolation filter (refer to [“Implementation Details for Interpolation & Decimation Structures”](#) on page 3–12), in that the input is held until all the phases are computed. However, in a multichannel application, the input for *each channel* is held until all phases of *that channel* are computed; then, the next channel input is applied to the interpolation filter, etc. Thus, for each channel in a multichannel scheme, an input is held constant and all phases are computed before the next channel’s input may be applied.

When applying a multi-rate decimation filter to a multi-channel structure, it is necessary to keep applying inputs until the FIR Compiler computes the results for a single channel, and then inputs from the next channel are applied, etc.

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
- Multi-coefficient set capability

Family-Specific Features

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, Stratix II, and Cyclone II structures. In addition, Stratix and Stratix II multicycle variable implementations take advantage of the built-in adder structures in the DSP block.



Stratix and Stratix II filters allow the most flexibility for data and coefficient storage; users can choose between M512, M4K, and MRAM (when appropriate).

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.

OpenCore Plus Time-Out Behavior

OpenCore® Plus hardware evaluation supports the following two 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 MegaCore functions 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 may be masked by the time-out behavior of the other megafunctions.

 For MegaCore functions, the untethered timeout is 1 hour; the tethered timeout value is indefinite.

The output `fir_result` is forced low when the evaluation time expires. Your design stops working after the hardware evaluation time expires.



For more information on OpenCore Plus hardware evaluation, see [“OpenCore Plus Evaluation” on page 1–8](#) and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

Atlantic I/O Protocol

Atlantic™ I/O protocol is available for the following architectures:

- Fully parallel
- Fully serial
- Multibit serial

By default, the FIR Compiler implements a master sink and master source, and will:

- Interface with a slave source to obtain data
- Interface with a slave sink to output data

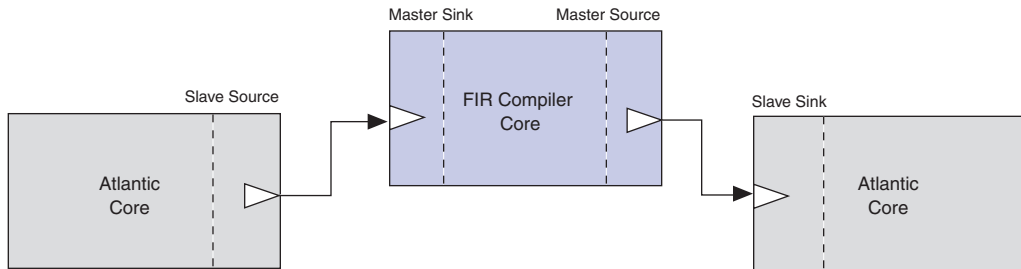
The FIR Compiler allows you to convert the master sink into a slave sink with a slave-to-slave adaptor. To make these selections, click the **Flow Control** button in the **Parameterize** page, see [Figure on page 2–7](#).

Implementing the Altera Atlantic I/O protocol with the FIR Compiler is optional; however, implementing the FIR Compiler's master sink and master source modes does requires additional memory resources. See [Figure 3–12](#).



For more information on the Atlantic interface, see *FS 13: Atlantic Interface*.

Figure 3–12. FIR Compiler Communicating with the Atlantic Interface

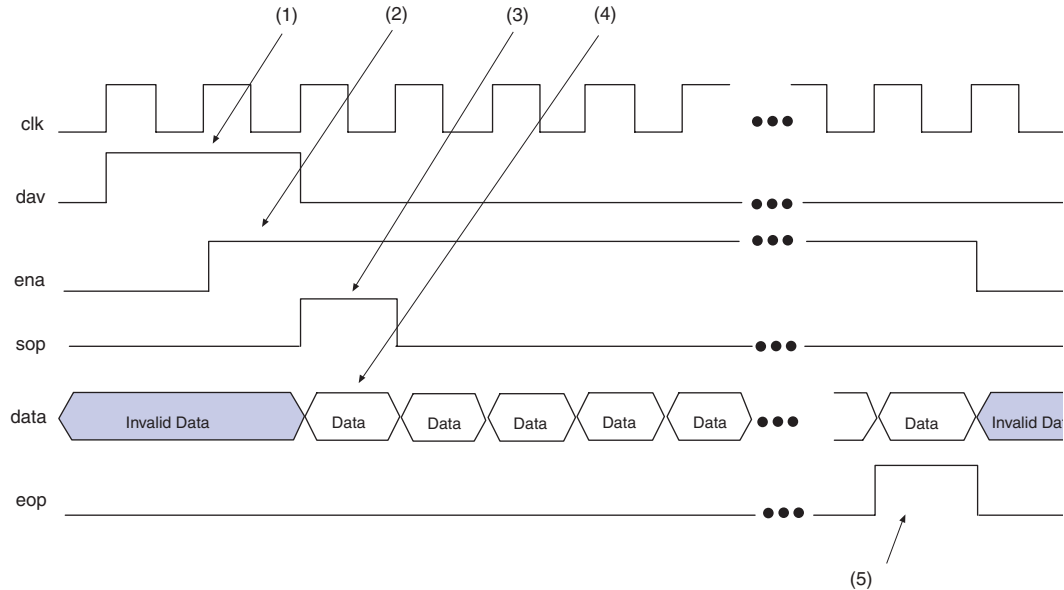


FIR Compiler Input Protocol

A FIR Compiler master sink interface operation generates read commands to a slave source interface. When the master sink is ready for data, it asserts `ena`. On the following rising `clk` edge, the slave source observes that `ena` is asserted. In response, the slave source immediately drives new data on `dat`. See [Figure 3–13](#).

If `ena` is asserted on a previous clock edge and `val` is asserted on the current clock edge, then `dat`, `sop`, and `eop` are also valid on the current clock edge, and `dat` contains new data.

The `dav` signal indicates that the slave can supply a significant amount of data—the amount of data is application dependent. If the master continues to assert `ena` for an extended period of time after `dav` is deasserted, the slave may underflow.

Figure 3–13. FIR Compiler Input Protocol: Atlantic Interface Timing with a Master Sink**Notes to Figure 3–13:**

- (1) Slave asserts `dav` (indicating that THRESHOLD words are available).
- (2) Master asserts `ena`.
- (3) Slave responds with `sop`.
- (4) Slave presents data to master.
- (5) Slave presents `eop` (indicating the last data in packet).

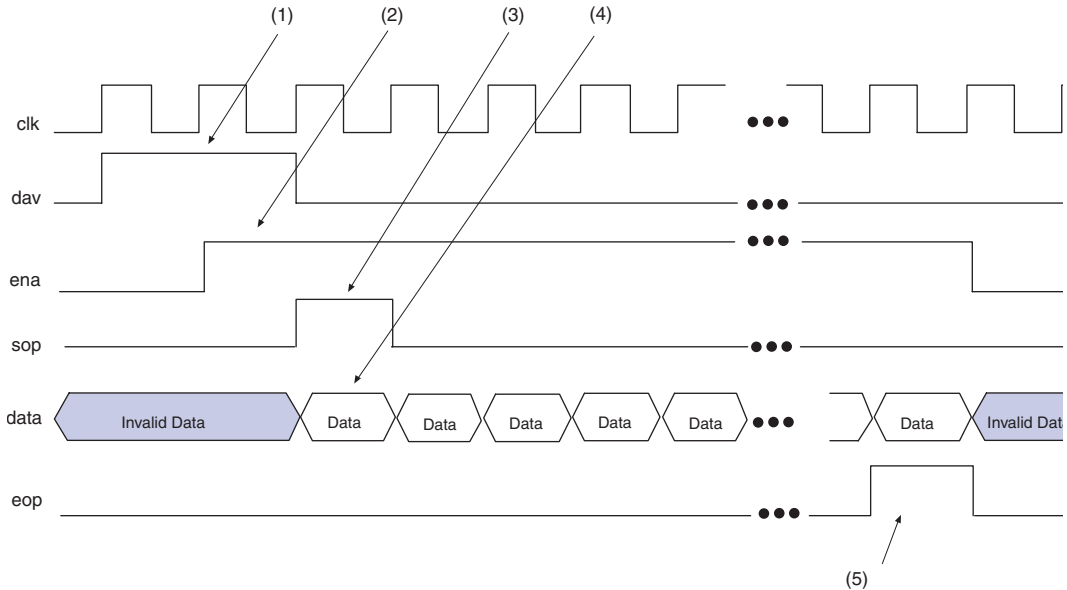
FIR Compiler Output Protocol

A FIR Compiler master source interface operation writes data to the slave sink, i.e., the master asserts `ena` followed by `sop` and `dat` on the following cycle. On the following rising `clk` edge, the slave sink observes `sop` and `dat`. If `sop` is asserted, the slave accepts and processes data; if `sop` is not asserted, the slave discards the contents of `dat`. No `val` is required for a master source to slave sink transaction because `ena` and `sop` indicate when `dat` contains new, valid data. The slave sink has no cycle-by-cycle flow control; it uses `dav` to request the master to stop data transfer. However, the master may take several clock cycles to stop transferring data, depending on the application.

The `dav` signal indicates that the slave can accept a significant amount of data—the amount of data is application dependent. Because THRESHOLD is defined, the slave must accept at least THRESHOLD words. If the master continues to assert `ena` for an extended period of time after `dav` is deasserted, the slave may overflow.

For a master source, there is no delay after `ena` is asserted or deasserted and dataflow on `dat` (and associated data interface signals) starts or stops. See [Figure 3-14](#).

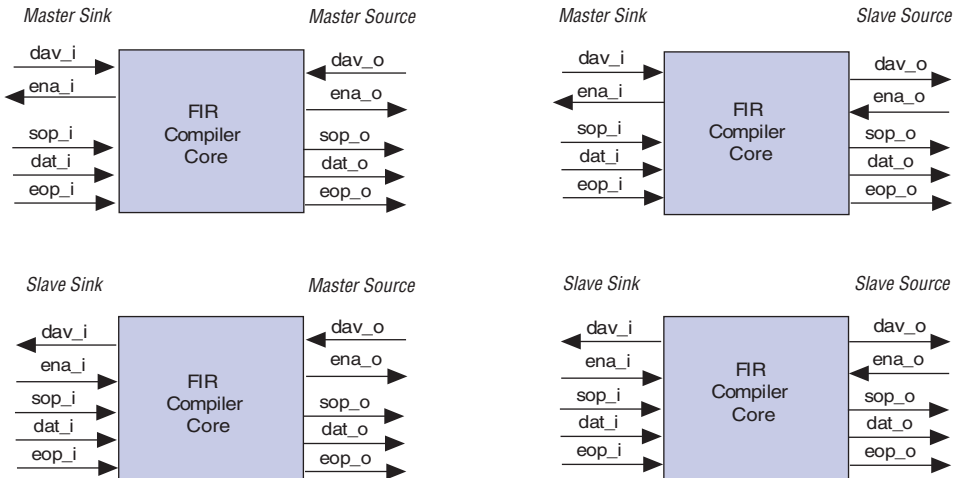
Figure 3-14. FIR Compiler Output Protocol: Atlantic Interface Timing with a Master Source



Notes to Figure 3-14:

- (1) Slave asserts `dav` (indicating that it has space for THRESHOLD words).
- (2) Master asserts `ena`.
- (3) Master responds with `sop`.
- (4) Master presents data to master.
- (5) Master presents `eop` (indicating the last data in packet).

The [Figure 3-15](#) system blocks show how the Atlantic I/O protocol is configured.

Figure 3–15. Atlantic I/O System Block Configuration

The FIR Compiler also contains built-in Atlantic adaptors to talk with master sources and master sinks. The adaptors are slave-slave adaptors that are attached to the inputs and/or outputs, and the user specifies the adaptor's threshold values.



Using slave-slave adaptors consumes additional device logic and memory. To accurately estimate the amount of required memory, use the Quartus® II **Resource Estimator** on the **Architecture** page, see [Figure 2–12 on page 2–23](#).

Simulation Output Files

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

The regression suite covers various parameters such as input and output bit widths, varying numbers of coefficients, and relevant architecture options.

Signals

The FIR Compiler can generate four different FIR structures:

- *Parallel*—Optimized for speed; provides one output per clock cycle.
- *Serial*—Optimized for area; uses a small number of clock cycles per output.
- *Multibit serial*—Designed for flexibility; the user specifies the number of serial units. The multibit serial structure uses several units for area and speed between parallel and serial.
- *Multicycle variable*—Designed for flexibility; the user specifies the number of cycles (multicycle) that the filter uses.



For more information on the FIR structures, refer to “Structure Types” on page 3–7.

Table 3–1 lists the FIR Compiler signals without Atlantic interface.

Signal	Direction	Description
clk	Input	Input clock signal.
clk_en	Input	Active high signal, enable clock. This pin appears when the Flow Control by clk_en pin option is selected (Flow Control button on the Coefficients Specification window of the Parameterize FIR Compiler page).
coef_in	Input	Input coefficient value when reloading coefficient.
coef_in_clk	Input	Clock to reload coefficients when coefficients are stored in memory. This clock can be different than clk.
rst	Input	Synchronous reset signal. Resets the FIR Compiler control circuit on the rising edge of clk. This signal should last longer than one clock cycle. See Figure 3–16 on page 3–24 .
data_in[]	Input	Input data to be filtered.
coef_set[]	Input	Selects which coefficient set the filter uses for the calculation. This is applicable only when multi-coefficient sets are used. The width of this signal = $\text{ceil}(\log(\text{num of the coefficient set})/\log(2.0))$

Signal	Direction	Description
coef_set_in[]	Input	Selects which coefficient set to overwrite when multiple sets are used. This is applicable only when multicoefficient sets are used. The width of this signal = $\text{ceil}(\log(\text{num of the coefficient set})/\log(2.0))$
coef_we	Input	Active high. Enables coefficient overwriting when multiple sets are used.
rdy_to_ld	Output	Active-high signal that indicates the filter is ready to load new data on the data input pin on the next clock cycle.
done	Output	Indicates the presence of new data.
fir_result[]	Output	Result of filtering operation performed by data_in. need input_ch_id and outup_ch_id.
coef_ld	Output	Indicates when to load a coefficient when a multicyle variable is selected and reloading coefficients are stored in logic cells.
input_ch_id[]	Output	Indicates the current channel number (from 0 to N-1) (for the input side) This signal only exists when multi_channel is used. The width of this signal = $\text{ceil}(\log(\text{num of_channel})/\log(2.0))$
output_ch_id[]	Output	Indicates the current channel number (from 0 to N-1) (for the output side) This signal only exists when multi_channel is used. The width of this signal = $\text{ceil}(\log(\text{num of_channel})/\log(2.0))$

Table 3–2 lists the FIR Compiler signals with Atlantic interface flow control.

Signal	Direction	Description
clk	Input	Input clock signal.
rst	Input	System reset.
dav_i	Input, when sink is Master; Output, when sink is slave.	Atlantic interface sink dav signal.
ena_i	Output, when sink is Master; Input, when sink is slave.	Atlantic sink ena signal.
sop_i	Input	Atlantic sink sop signal.
eop_i	Input	Atlantic sink eop signal.
dat_i[]	Input	Input of the data to be filtered. For multi-coefficient set case, dat_i[] is combined by coefficient set selection and input data to be filtered.
dav_o	Output, when sink is Master; Input, when sink is slave.	Atlantic source dav signal.
ena_o	Input, when sink is Master; Output, when sink is slave.	Atlantic source ena signal.
sop_o	Output	Atlantic source sop signal.

Table 3–2. FIR Compiler Signals With Atlantic Interface

Signal	Direction	Description
<code>eop_o</code>	Output	Atlantic source <code>eop</code> signal.
<code>dat_o[]</code>	Output	Result of filtering operation performed by <code>data_in</code> .
<code>coef_in[]</code>	Input	Input coefficient value when reloading coefficient.
<code>coef_in_clk</code>	Input	Input clock to reload coefficients when coefficients are stored in memory. This clock can be different than <code>clk</code> .
<code>coef_we</code>	Input	Write enable for coefficient reloading.
<code>coef_set_in[]</code>	Input	Selects which coefficient set to overwrite when multiple sets are used. This is applicable only when multicoefficient sets are used. The width of this signal = $\text{ceil}(\log(\text{num of the coefficient set})/\log(2.0))$

Timing Diagrams

This section describes the timing diagrams for different types of filters, specifically:

- Parallel Timing Diagrams
- Serial & Multibit Serial Timing Diagrams
- Multicycle Variable Timing Diagrams
- Coefficient Reloading Timing Diagrams



The reset 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 `rst` signal is applied. To clear the data, set the `data_in` port to 0 for n clock cycles, where $n = (\text{number of coefficients}) \times (\text{number of input channels}) \times (\text{number of clock cycles needed to compute a FIR result})$.

The `fir_result` value depends on the coefficient values in the design. Therefore, the timing diagrams of your own design may be different than what is 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.

Figure 3–18. Parallel Interpolation Waveform

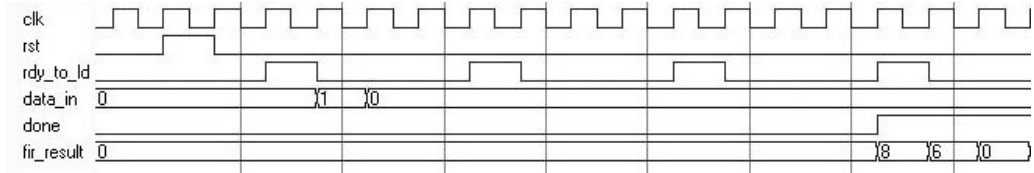
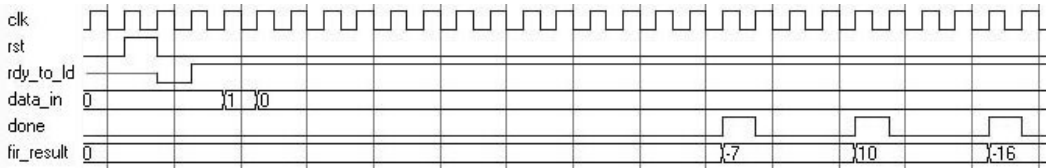


Figure 3–19 shows the parallel decimation waveform.

Figure 3–19. Parallel Decimation Waveform



Serial & Multibit Serial Timing Diagrams

This section provides information on controlling the rate of serial and multibit serial filters as well the associated timing diagrams.



Several of the diagrams in this section use an impulse that has leading zeros, one, and trailing zeros as input data. Because the FIR filter is causal, the response has pipeline delay followed by leading zeros and output data.

Serial Timing Diagrams

Figure 3–20 shows the input timing diagram for an 8-bit serial filter.

Figure 3–20. 8-Bit Serial Filter

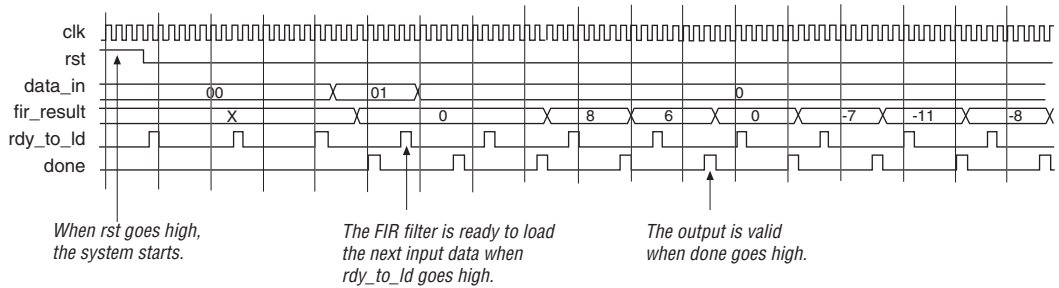


Figure 3–21 shows the serial interpolation waveform.

Figure 3–21. Serial Interpolation Waveform

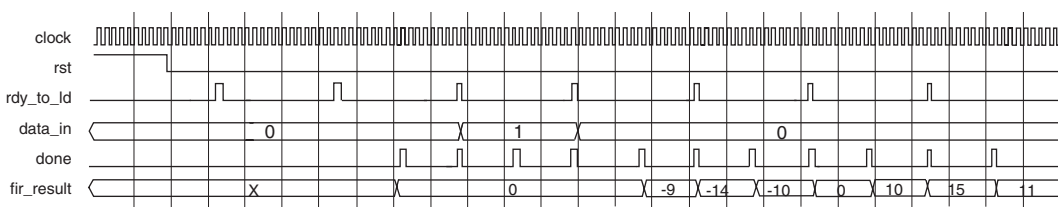


Figure 3–22 shows the serial decimation waveform.

Figure 3–22. Serial Decimation Waveform

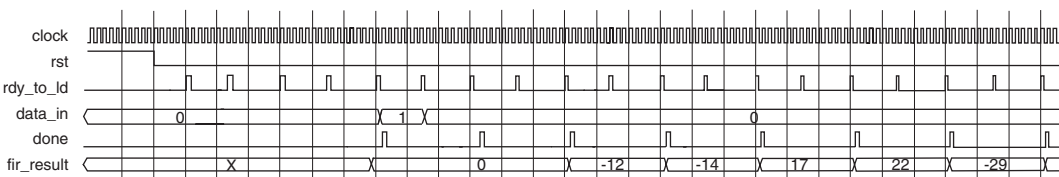
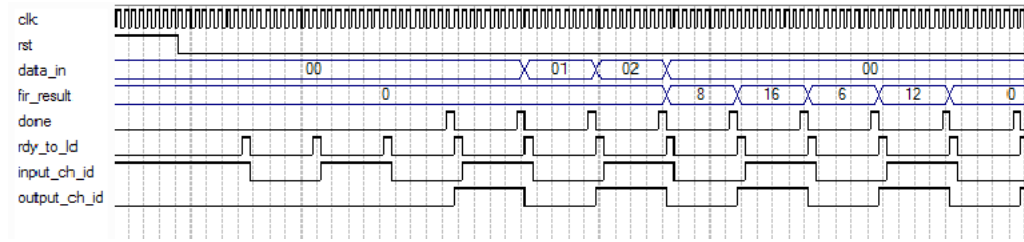


Figure 3–23 shows a multi-channel serial timing diagram with two channels and an eight-bit input data.

Figure 3–23. Multichannel Serial Timing Diagram



Multibit Serial Timing Diagrams

Figure 3–24 shows the timing diagram for a multibit serial filter which has two serial units and an eight-bit input data width. A standard serial structure requires eight clock cycles to compute the result. In contrast, the multibit serial structure requires four clock cycles. The disadvantage is that the multibit serial implementation uses four times as many resources (LEs and memory blocks) than a serial filter to achieve the 4X improvement. However, the multibit serial implementation still uses fewer resources than the parallel implementation.

Figure 3–24. Multibit Serial Timing Diagram

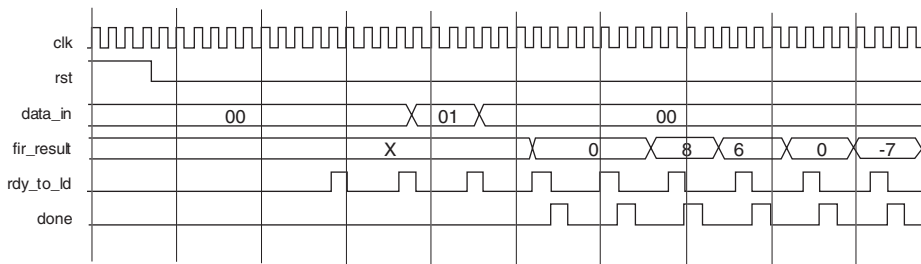
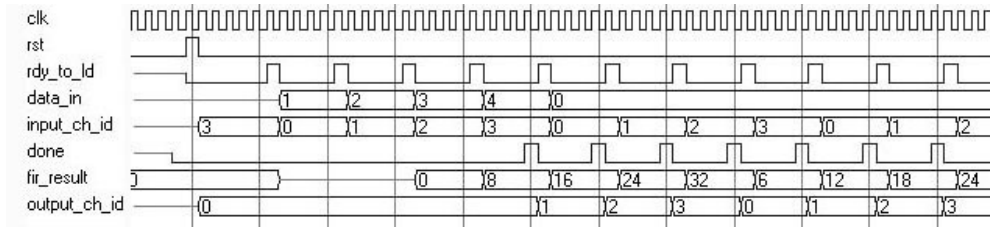


Figure 3–25 shows the same filter with eight bits of input data and two serial channels implemented using two channels. One set of data feeds into the first channel and another set feeds into the second. A serial filter will take eight clock cycles to compute a single channel. However, the multibit serial filter takes just four cycles to compute a single channel. The channels are time-interleaved, and the filter generates a result for each channel four clock cycles apart.

Figure 3–25. Multichannel Multibit Serial Timing Diagram



Multicycle Variable Timing Diagrams



Several of the diagrams in this section use an impulse that has leading zeros, one, and trailing zeros as input data. Because the FIR filter is causal, the response has pipeline delay followed by leading zeros and output data.

Figure 3–26 shows a multicycle FIR filter example with four cycles. The input and output data is held for four clock cycles.

Figure 3–26. Multicycle Variable with One Channel & Four Cycles Timing Diagram

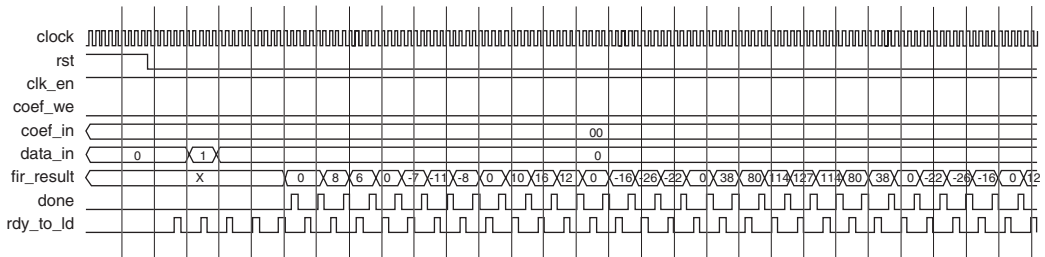
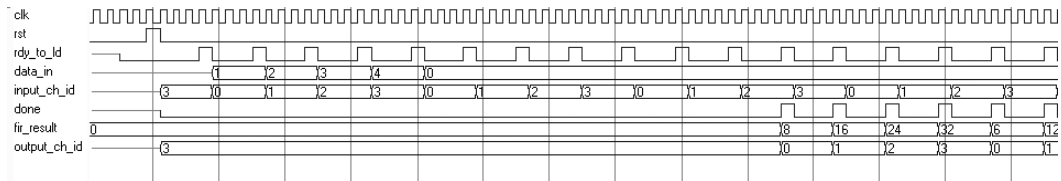


Figure 3–27 shows a multicycle variable filter with four channels.

Figure 3–27. Multicycle Variable with Four Channels Timing Diagram



Coefficient Reloading Timing Diagrams

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.

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 3–28](#) shows the serial and multibit serial coefficient reloading timing diagram. [Figure 3–29](#) shows the 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.

Figure 3–28. Serial & Multibit Serial Coefficient Reloading Timing Diagram

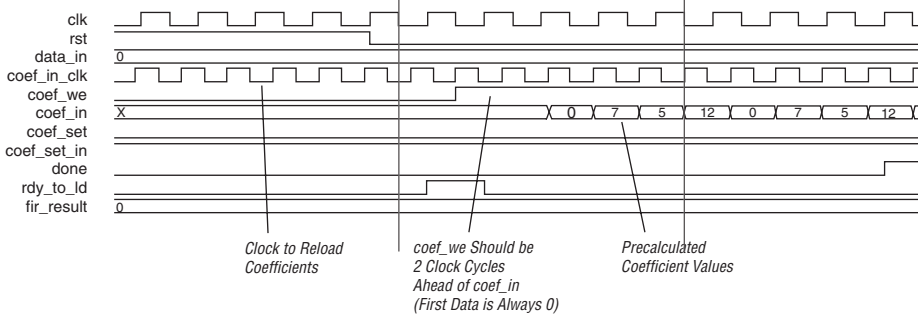
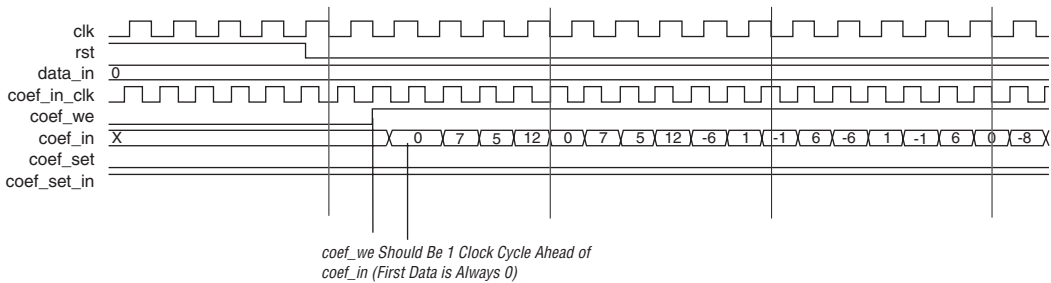
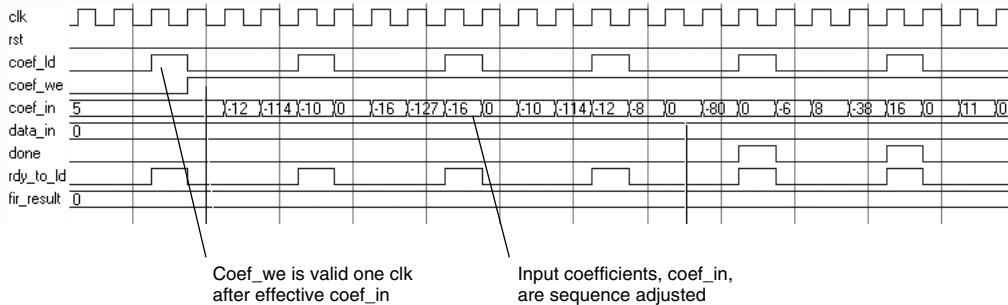


Figure 3–29. Parallel Coefficient Reloading Timing Diagram



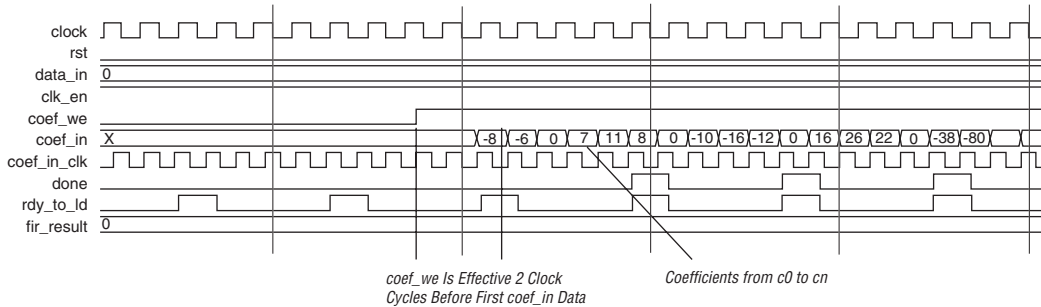
Multicycle variable reloading is faster than the fixed FIR (with reloading capability). If coefficients are stored in logic cells, coefficients need sequence adjustment using the same algorithm as fixed FIR filters. The reloading clock is the same as the FIR filter calculation clock; `coef_we` should be triggered by the `coef_ld` signal. See [Figure 3–30](#).

Figure 3–30. 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. See [Figure 3–31](#).

Figure 3–31. Multicycle Variable (Using Memory Blocks) Coefficient Reloading Timing Diagram

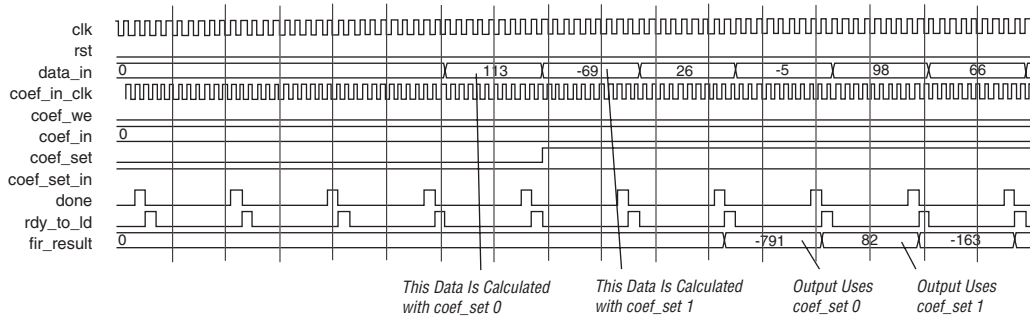


If you use multiple coefficient sets, you can update one set of coefficients while using another set of coefficients for calculation. The signals **coef_set_in** and **coef_we** are not clocked in and pipelined synchronously. While you update the coefficient set, you need to set and hold the **coef_set_in** signal for several cycles before **coef_we** is asserted and after it is de-asserted. While another set of coefficients are in calculation, the selection of the coefficient set aligns with the input data. See [Figure 3–32](#).



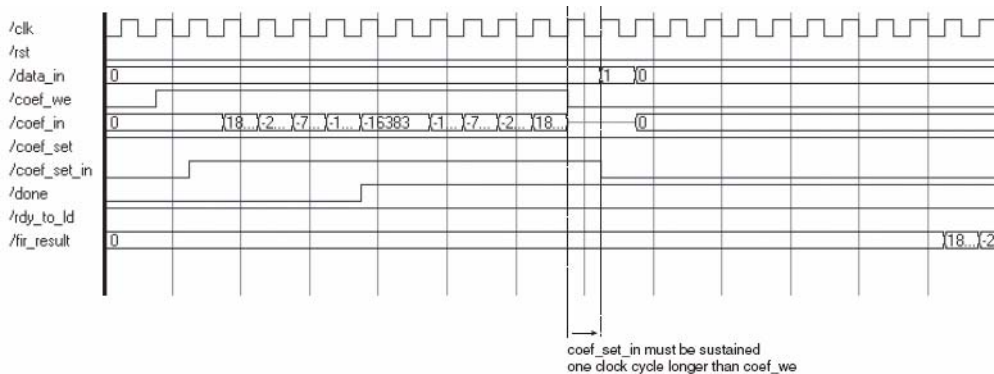
For DSP Builder users, a button is available that ties the `coef_in_clk` to the `clk` in the wrapper. DSP Builder will not work with more than one clock domain per MegaCore function.

Figure 3–32. Multiple Coefficient Set Selection Timing Diagram



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 [Figure 3–33](#) illustrates. 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 3–33. Timing Requirements for Loading Multiple Coefficient Sets



MegaCore Verification

Before releasing a version of the FIR Compiler, Altera runs a comprehensive regression test that executes the wizard to create the instance files. Next, Verilog HDL and VHDL test benches are created and the results are compared to the MATLAB software using ModelSim® simulators to exercise the Verilog HDL and VHDL models.



Appendix A. FIR Compiler Supported Device Structures

Supported Device Structures

Table A-1 lists Stratix® II, Stratix, Stratix GX, Cyclone™ II, and Cyclone device structures supported by FIR Compiler.

Table A-1. Device Structures Supported by FIR Compiler (Part 1 of 3)

Structure	Sub Structure	Input Data Bit Width	Flow Control	Data Storage (1) (2)	Coefficient Storage	Multi Coefficient Sets	Symmetric Coefficient	Adder Tree (3)
Serial	Fixed Coefficient	4 to 32	clk_en and Atlantic interface	Logic cell, M512, M4K, M-RAM Auto	Logic cell, M512, M4K	Yes	Yes	Ternary Binary
	Reloadable Coefficient	4 to 32	clk_en and Atlantic interface	Logic cell, M512, M4K, M-RAM Auto	M512, M4K	Yes	Yes	Ternary Binary
	Interpolation (Reloadable Coefficient must be in memory)	4 to 32	clk_en	Logic cell M512, M4K, M-RAM Auto	Logic cell M512, M4K	Yes	N/A	Ternary Binary
	Decimation (Reloadable Coefficient must be in memory)	4 to 32	clk_en	Logic cell M512, M4K, M-RAM Auto	Logic cell M512, M4K	Yes	N/A	Ternary Binary

Table A–1. Device Structures Supported by FIR Compiler (Part 2 of 3)

Structure	Sub Structure	Input Data Bit Width	Flow Control	Data Storage (1) (2)	Coefficient Storage	Multi Coefficient Sets	Symmetric Coefficient	Adder Tree (3)
Multibit Serial	Fixed Coefficient	≥ 4 (number of serial units) (1)	clk_en and Atlantic interface	Logic cell, M512, M4K, M-RAM Auto	Logic cell, M512, M4K	Yes	Yes	Ternary Binary
	Reloadable Coefficient	≥ 4 (number of serial units) (1)	clk_en and Atlantic Interface	Logic cell, M512, M4K, M-RAM Auto	M512, M4K	Yes	Yes	Ternary Binary
	Interpolation (Reloadable Coefficient must be in memory)	≥ 4 (number of serial units) (1)	clk_en	Logic cell M512, M4K, M-RAM Auto	Logic cell M512, M4K	Yes	N/A	Ternary Binary
	Decimation (Reloadable Coefficient must be in memory)	≥ 4 (number of serial units) (1)	clk_en	Logic cell M512, M4K, M-RAM Auto	Logic cell M512, M4K	Yes	N/A	Ternary Binary
Parallel	Fixed Coefficient	2 to 32 for signed, 1 to 32 for unsigned	clk_en and Atlantic interface	Logic cell, Auto	Logic cell, M512, M4K	Yes	Yes	Ternary Binary
	Reloadable Coefficient	2 to 32 for signed, 1 to 32 for unsigned	clk_en and Atlantic interface	Logic cell, Auto	M512, M4K	Yes	Yes	Ternary Binary
	Interpolation (Reloadable Coefficient must be in memory)	4 to 32	clk_en	Logic cell Auto	Logic cell M512, M4K	Yes	N/A	Ternary Binary
	Decimation (Reloadable Coefficient must be in memory)	4 to 32	clk_en	Logic cell Auto	Logic cell M512, M4K	Yes	N/A	Ternary Binary

Table A-1. Device Structures Supported by FIR Compiler (Part 3 of 3)

Structure	Sub Structure	Input Data Bit Width	Flow Control	Data Storage (1) (2)	Coefficient Storage	Multi Coefficient Sets	Symmetric Coefficient	Adder Tree (3)
Multicycle Variable	Clocks to compute = 1	2 to 32 for signed, 1 to 32 for unsigned	clk_en and Atlantic interface	Logic cell, Auto	Logic cell, when (number of coeff set) >1, M512, M4K	Yes	Yes	Ternary Binary
	Clocks to compute = 2	2 to 32 for signed, 1 to 32 for unsigned	clk_en and Atlantic interface	Logic cell, Auto	Logic cell, M512, M4K	Yes	Yes	Ternary Binary
	Clocks to compute = 3	2 to 32 for signed, 1 to 32 for unsigned	clk_en and Atlantic interface	Logic cell, M512, M4K, Auto	Logic cell, M512, M4K	Yes	Yes	Ternary Binary
	Interpolation: Clocks per output = 1	2 to 32 for signed, 1 to 32 for unsigned	clk_en	Logic cell, Auto, M512, M4K	Logic cell, M512, M4K	Yes	N/A	Ternary Binary
	Decimation: Clocks per input = 1	2 to 32 for signed, 1 to 32 for unsigned	clk_en	Logic cell, Auto, M512, M4K	Logic cell, M512, M4K	Yes	Yes	Ternary Binary

Notes to Table A-1:

- (1) Depends on the Quartus II software, version 5.1, user settings specified, i.e., logic cells or memory blocks.
- (2) Cyclone devices do not support M512 or MRAM memory.
- (3) Ternary adder trees are only available with Stratix II devices; all other devices use binary adder trees.



Vector files (“Step 2: Set Up Simulation” on page 2-24) are not provided for functions with the Atlantic™ interface. The Atlantic interface requires a Verilog HDL or VHDL simulator to properly simulate files. (The dav, ena, and sop protocols require receiving input to generate output from the test bench side).

HardCopy Stratix & HardCopy II Support

FIR Compiler provides full support for HardCopy Stratix and preliminary support for HardCopy II devices. Both HardCopy Stratix and HardCopy II devices do not support preloaded RAM elements. Therefore, these features cannot be used when targeting HardCopy Stratix and HardCopy II devices.

Preloaded RAM may be used in two places: in tapped delay line (for data storage) and/or for coefficient storage in reloadable FIR filters. This section outlines how to use preloaded RAM in these cases.

When used for data storage, preloaded RAM is used if the you choose M512 or M4K. In addition, if the you choose **Auto**, Quartus may choose to use M512 or M4K. For Stratix and Stratix II devices, the memory is preloaded with zeros automatically. This cannot be done for HardCopy Stratix and HardCopy II devices. For HardCopy Stratix and 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 discarding the corresponding n outputs. The formula for doing this is as follows:

$$n = \text{number of channels} * \text{number of coefficients}$$

Preloaded RAM may also be used if you choose M512 or M4K for the coefficient storage of reloadable FIR filters. Since this RAM cannot be preloaded in HardCopy Stratix or HardCopy II, you must implement the logic to initialize or update the coefficients. This can be done on-chip or by using the Coefficient Reordering software (coef_esq.exe) included in the `<path>\fir_compiler-v3.3.1\misc` directory and described in section [“Coefficient Reordering” on page 3–5](#).

Compiling HardCopy Stratix & 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 to successfully compile a HardCopy Stratix or 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:

`<module 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:

`<modulename>_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.