FIR Compiler

MegaCore Function User Guide

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

Core Version: 2.6.1
Document Version: 2.6 rev. 2
Document Date: November 2002

I.S. EN ISO 9001

UG-FIRCOMPILER-2.10

About this User Guide

This user guide provides comprehensive information about the Altera® FIR Compiler MegaCore® function.

Table 1 shows the user guide revision history.

Go to the following sources for more information:

- See "New Features in Version 2.6.1" on page 9 for a complete list of the core features, including new features in this release.
- Refer to the FIR Compiler readme file for late-breaking information that is not available in this user guide.

Table 1. User Guide Revision History

Date	Description
November 2002, v2.6.1	This version is a minor update to the core, and a minor update to the FIR Compiler Megacore Function User Guide, v2.6.0.
October 2002, v2.6.0 rev. 1	This version is an update to the core. Enhancements include support for the Cyclone™ device family ; this core uses Stratix Tri-Matrix Memory for all single-rate filters.
August 2002, v2.5.2 rev. 1	This version is a minor update to the core. Enhancements: disabled the Signed Binary Fraction Function for DSP Builder.
June 2002, v2.5.1 rev. 2	Updated the document for version 2.5.1 of the core. This version is a minor update, which is compatible with DSP Builder 2.0 Modular IP installation.
May 2002, v2.5.0 rev. 2	Updated the document for version 2.5.0 of the core. Renamed the Waveforms section as Timing Diagrams. Added information about the multi-cycle variable structure, modular support for DSP Builder, and OpenCore Plus.
December 2001	Updated the features and made significant changes to the walkthrough. Updated the timing diagrams and added them to a new Waveforms section. Updated parallel, serial, and multi-bit serial signals.
August 2001, v2.4	Updated the symbol figure. Added a note about not adding extra carriage returns in the coefficient text file. Updated the information for simulating using VHDL and Verilog HDL models.
August 2001	Updated the document for core version 2.3.0. Described new pipelining support and the full resolution calculation option. Updated the signal information and broke it into separate tables for parallel, serial, and variable. Updated the organization of the document.
April 2001	Updated document for core version 2.2.0. Added information on the multi-bit serial option. Updated signal names.
February 2001	Updated document for core version 2.1.0. Added multi-MAC architecture figure. Updated signal names.

How to Find Information

- The Adobe Acrobat Find feature allows you to search the contents of a PDF file. Click on the binoculars icon in the top toolbar to open the Find dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages.
- Numerous links, shown in green text, allow you to jump to related information.

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at http://www.altera.com.

For technical support on this product, go to http://www.altera.com/mysupport. For additional information about Altera products, consult the sources shown in Table 2.

Table 2. How to Contact Altera

Information Type	USA & Canada	All Other Locations
Technical support	http://www.altera.com/mysupport/	http://www.altera.com/mysupport/
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	(408) 544-7000 *(1)* (7:00 a.m. to 5:00 p.m. Pacific Time)
Product literature	http://www.altera.com	http://www.altera.com
Altera literature services	lit_req@altera.com *(1)*	lit_req@altera.com *(1)*
Non-technical customer service	(800) 767-3753	(408) 544-7000 (7:30 a.m. to 5:30 p.m. Pacific Time)
FTP site	ftp.altera.com	ftp.altera.com

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

Typographic Conventions

The ***FIR Compiler MegaCore Function User Guide*** uses the typographic conventions shown in Table 3.

Table 3. Conventions

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}, \qdesigns** directory, **d:** drive, **chiptrip.gdf** file.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <*file name*>, <*project name*>**.pof** 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 online 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 (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c.,...	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 angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

Notes:

About this Core

Release Information

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

Table 1. FIR Compiler Release Information

Item	Description
Version	2.6.1
Release Date	November 15, 2002
Ordering Code	IP-FIR
Product ID(s)	0012
Vendor ID(s)	6AF8 (Standard) 6AF9 (Time-Limited)

Introduction

The Altera FIR Compiler MegaCore function generates finite impulse response (FIR) filters customized for Altera devices. You can use the FIR Compiler wizard interface to implement a variety of filter architectures, including fully parallel, serial, multi-bit serial fixed-coefficient, and multi-cycle variable filters. The wizard also includes a coefficient generator to help you create filter coefficients.

New Features in Version 2.6.1

- Uses Stratix Tri-Matrix Memory for all single-rate filters
  - Takes advantage of M512, M4K, and M-RAM in Stratix families, 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 DSP Block for coefficient storage
  - Choose among several pipeline levels
  - Includes a resource estimator
- Support for Cyclone™ device family
- Information Box to assist in filter design techniques
  - Shows tips and techniques, when appropriate
  - Shows filter performance
- Symmetry selection for fixed-coefficient FIR filters
- Visual IP simulation models are provided
- Support for ModelSim 5.6

Features

- Over 250-MHz performance in Stratix™ devices
- Symmetric variable FIR filter support
- Supports OpenCore® and OpenCore Plus hardware evaluation
- Has the DSP Builder Ready certification
- Fully integrated FIR filter development environment
- First system-level, programmable logic solution for DSP designs, including automatic interpolation and decimation for all fixed FIR filters
- Highly optimized for Altera device architectures, including Cyclone, Stratix, APEX, APEX II, Mercury, FLEX®, and ACEX® devices
- Extended pipelining for all fixed FIR filters
- Supports a variety of architectures:
  - Fixed-coefficient filters
    - Fully parallel
    - Serial
    - Multi-bit serial
    - Supports interpolation and decimation
  - Variable filters
    - Multi-cycle (the user chooses the number of cycles)
    - Supports loading, reloading, and multiple coefficient sets
- Supports up to 2,047-tap filters
- Coefficient generator:
  - Includes a built-in coefficient generator
  - Supports coefficient widths from 4 to 32 bits of precision
  - Imports floating-point or integer coefficients from third-party tools
  - Supports multiple coefficient sets up to a total of 32 sets
  - Provides several coefficient scaling algorithms
  - Provides floating-point to fixed-point coefficient analysis
- Includes impulse, step function, and random input testbeds
- Supports signed or unsigned input data widths, from 4 to 32 bits wide
- User-selectable output precision via rounding and saturation
- Generates MATLAB simulation models
- Creates VHDL and Verilog HDL simulation files for all structures
- Generates Quartus® II and MAX+PLUS® II vector files
- Generates a report file in HTML format containing information about the filter created, and the generated synthesis and simulation files

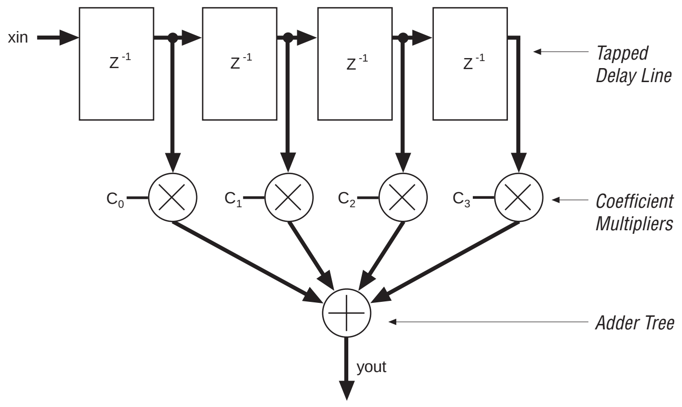
General Description

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. FIR filters are used in systems that require linear phase and have an inherently stable structure. IIR filters are used in systems that can tolerate phase distortion. 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.

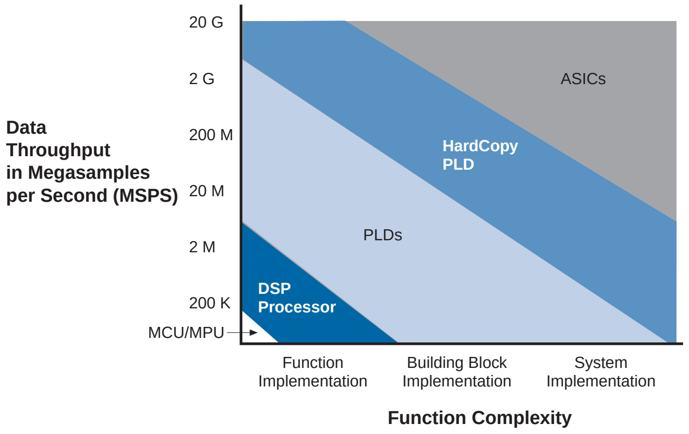
The structure of a FIR filter is a weighted, tapped delay line (see Figure 1). The filter design process involves identifying coefficients that match the frequency response specified for the system. The coefficients determine the response of the filter. You can change which signal frequencies pass through the filter by changing the coefficient values or adding more coefficients.

Figure 1. Basic FIR Filter



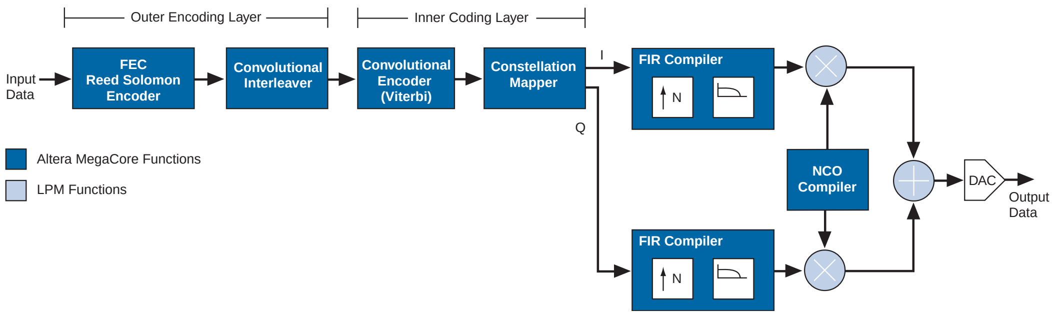
Traditionally, designers have been forced to make a trade-off between the flexibility of digital signal processors and the performance of ASICs and application-specific standard product (ASSPs) digital signal processing (DSP) solutions. The Altera DSP solution eliminates the need for this trade-off by providing exceptional performance combined with the flexibility of PLDs. See Figure 2.

Figure 2. Comparison of DSP Throughput



Altera DSP solutions include MegaCore® functions developed and supported by Altera, and Altera Megafunction Partners Program (AMPP^{SM}) functions. Additionally, many commonly used functions, such as adders and multipliers, are available from the industry-standard library of parameterized modules (LPM). Figure 3 shows a hypothetical DSP system and highlights the functions that are available from Altera and the LPM.

Figure 3. Hypothetical 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 a programmable logic device (PLD) can operate at very high data rates, making PLDs ideal for high-speed filtering applications.

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

Table 2. FIR Filter Implementation Comparison *Note (1)*

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

Note:

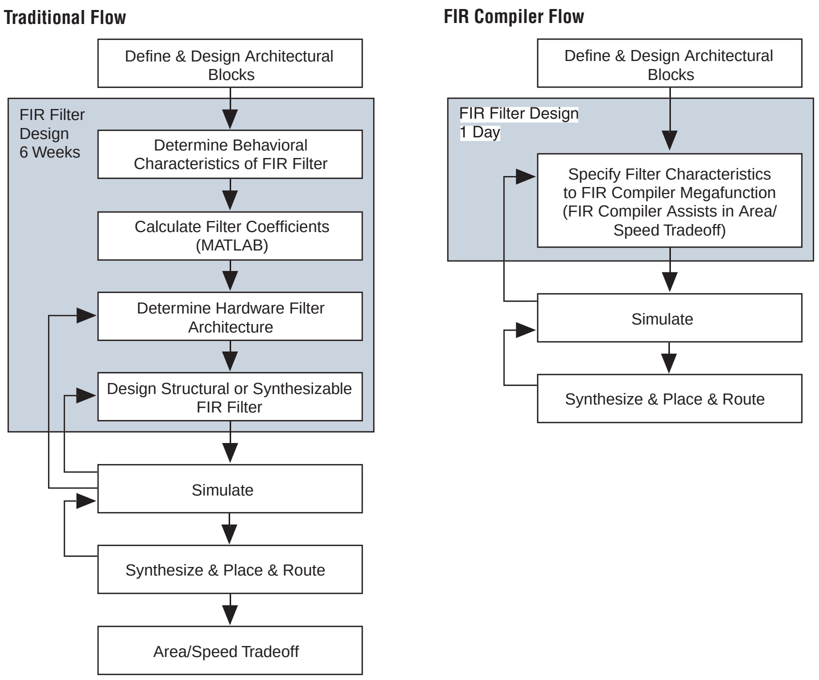
(1) If you use the FIR Compiler to create a filter, you can also implement a variable filter in a PLD that uses from 1 to 120 MACs, and 120 to 1 clock cycles.

The FIR Compiler function speeds the design cycle by:

- Finding 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 for the MATLAB environment (Simulink Model Files and M-Files).
- 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.

Figure 4 compares the design cycle using the FIR Compiler MegaCore function versus a traditional implementation.

Figure 4. Design Cycle Comparison

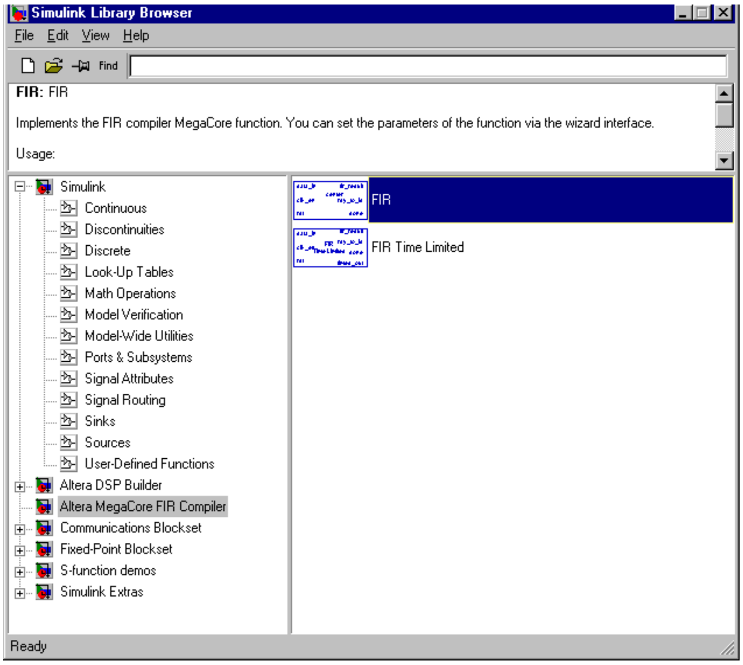

Traditional Flow
Define & Design Architectural Blocks
FIR Filter Design 6 Weeks
Determine Behavioral Characteristics of FIR Filter
Calculate Filter Coefficients (MATLAB)
Determine Hardware Filter Architecture
Design Structural or Synthesizable FIR Filter
Simulate
Synthesize & Place & Route
Area/Speed Tradeoff
FIR Compiler Flow
Define & Design Architectural Blocks
FIR Filter Design 1 Day
Specify Filter Characteristics to FIR Compiler Megafunction (FIR Compiler Assists in Area/ Speed Tradeoff)
Simulate
Synthesize & Place & Route

DSP Builder Support

DSP system design in Altera programmable logic devices requires both high-level algorithms and HDL development tools. The Altera DSP Builder, which you can purchase as a separate product, integrates the algorithm development, simulation, and verification capabilities of The MathWorks MATLAB and Simulink system-level design tools with VHDL synthesis and simulation of Altera development tools.

DSP Builder allows system, algorithm, and hardware engineers to share a common development platform. The 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 functions and Simulink blocks with Altera DSP Builder blocks to link system-level design and implementation with DSP algorithm development. The DSP Builder consists of libraries of blocks as shown in Figure 5.

Figure 5. DSP Builder Blocks in Simulink Library Browser



FIR: FIR

Implements the FIR compiler MegaCore function. You can set the parameters of the function via the wizard interface.

Usage:

DSP Builder version 2.1 and higher provides modular support for Altera DSP cores, including the FIR Compiler. The MATLAB software automatically detects cores that support DSP Builder and the cores appear in the Simulink Library Browser.

For more information on using DSP Builder with FIR Compiler, see "DSP Builder Feature & Simulation Support" on page 60.

OpenCore & OpenCore Plus Hardware Evaluation

The OpenCore feature lets you test-drive Altera MegaCore functions for free using the Quartus® II software. You can verify the functionality of a MegaCore function quickly and easily, as well as evaluate its size and speed, before making a purchase decision. However, you cannot generate device programming files.

The OpenCore Plus feature set supplements the OpenCore evaluation flow by incorporating free hardware evaluation. The OpenCore Plus hardware evaluation feature allows you to generate time-limited programming files for designs that include Altera MegaCore functions. You can use the OpenCore Plus hardware evaluation feature to perform board-level design verification before deciding to purchase licenses for the MegaCore functions. You only need to purchase a license when you are completely satisfied with a core's functionality and performance, and would like to take your design to production.

If you are simulating a time-limited MegaCore function using the DSP Builder and Simulink, i.e., in software, the core operation does not time out and the done pin stays low.

For more information on OpenCore Plus hardware evaluation using FIR Compiler, see "OpenCore Plus Time-Out Behavior" on page 61 and *AN 176: OpenCore Plus Hardware Evaluation of MegaCore Functions.*

Performance

Table 3 shows the FIR Compiler function's performance using the Quartus II software, a FIR filter with 97 taps, 8-bit input data, and 14-bit coefficients.

Table 3. Performance

Device	Filter Type	Pipeline Level	DSP Blocks	M512	M4K	Logic Cells	Speed (MHz)	Throughput (MSPS)	Processing Equivalent (GMACs) *(1)*
Cyclone	Parallel (M4K)	1	n/a	n/a	45	2847	189	189	18.3
Stratix	Serial (M4K, M512)	1	0	6	9	348	291	36	3.5
	Parallel (M512)	1	0	63	0	3257	205	205	19.9
	Parallel (M4K)	1	0	0	45	2847	208	208	20.2
	Parallel (LC)	2	0	0	0	7130	237	237	23.0
	Multi-cycle variable (4 cycle)	1	25*(2)*	25	25	772	192	48	4.7

Notes:

(1) GMAC = giga multiply accumulates per second; 1 GMAC = 1,000 million multiply accumulates per second (MMACs).

(2) 18 × 18 bit multiplier.

Getting Started

Software Requirements

The FIR Compiler requires the following software:

- A PC running the Windows 98/NT/2000 operating system
- Quartus II version 2.1sp1 or higher
- DSP Builder version 2.1 or higher (optional)

☞ If you want to use the FIR Compiler function in the UNIX environment, use the RealPC application by FWB Software (http://www.fwb.com/) to emulate Windows and install the FIR Compiler function as you would on a PC.

FIR Compiler Design Flow

Once you have purchased a license for FIR Compiler, the design flow involves the following steps:

☞ If you have not purchased a license, you can test-drive the core for free using the OpenCore or OpenCore Plus feature. Refer to *AN 176: OpenCore Plus Hardware Evaluation of MegaCore Functions* for more information on the OpenCore Plus feature.

1. Download and install the FIR Compiler function.
2. Set up licensing. This step is not required if you are test-driving the core using the OpenCore feature, however, you do need to obtain and install an OpenCore Plus license to test-drive the core using this feature.
3. Generate the filter(s) for your system using the FIR Compiler wizard.
4. Implement the rest of your system using the Altera Hardware Description Language (AHDL), VHDL, Verilog HDL, or schematic entry.
5. Use the FIR Compiler wizard-generated VHDL or Verilog HDL simulation models to confirm your system's operation.
6. Compile your design and perform place-and-route.

7. Perform system verification.

8. License the FIR Compiler function to configure or program the devices.

Download & Install

Before you can start using Altera MegaCore functions, you must obtain the MegaCore files and install them on your PC. The following instructions describe this process.

Obtaining the FIR Compiler MegaCore Function

If you have Internet access, you can download MegaCore functions from Altera's web site at **http://www.altera.com**. Follow the instructions below to obtain the MegaCore functions via the Internet. If you do not have Internet access, you can obtain the MegaCore functions from your local Altera representative.

1. Point your web browser to http://www.altera.com/IPmegastore.

2. Choose **Megafunctions** from the **Product Type** drop-down list box.

3. Choose **Signal Processing (DSP)** from the **Technology** drop-down list box.

4. Type `FIR` in the **Keyword Search** box.

5. Click **Submit**.

6. Click the **Try** icon next to the Altera FIR Compiler MegaCore function in the search results table.

7. Follow the online instructions to download the function and save it to your hard disk.

Installing the FIR Compiler Files

For Windows, follow the instructions below:

1. Open the directory into which you downloaded the FIR Compiler MegaCore function (in Step 7, above).

2. Double-click on the application icon labeled "fir_compiler-v2.6.1."

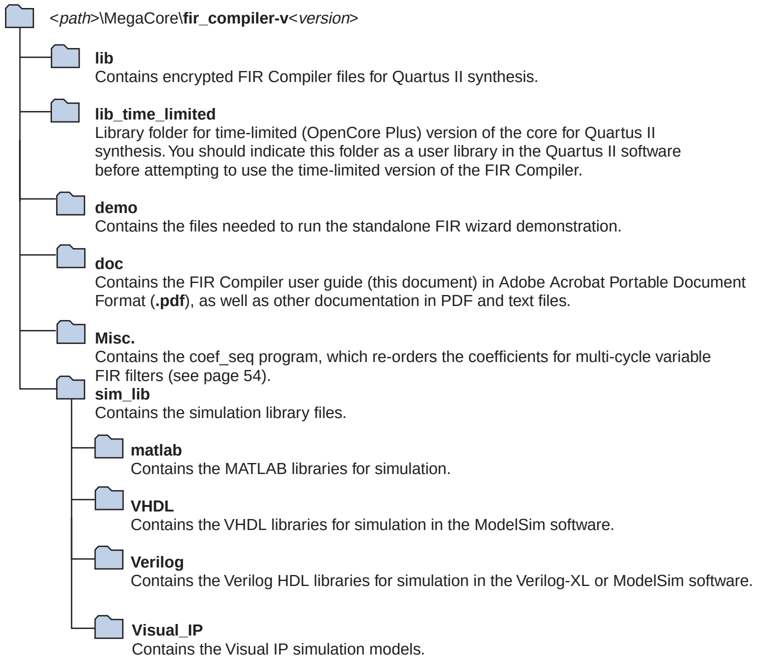
3. Click **OK**. The **FIR Compiler Installation** dialog box appears. Follow the online instructions to finish installation.

4. After you have finished installing the MegaCore files, you must specify the directory in which you installed them (i.e., <*path*>**fir_compiler-v**<*version*>**\lib**) as a user library in the Quartus II software. Search for "User Libraries" in Quartus II Help for instructions on how to add these libraries.

FIR Compiler Directory Structure

Figure 6 shows the directory structure for the FIR Compiler.

Figure 6. FIR Compiler Directory Structure



<*path*>\MegaCore\fir_compiler-v<*version*>

- **lib**
  Contains encrypted FIR Compiler files for Quartus II synthesis.
- **lib_time_limited**
  Library folder for time-limited (OpenCore Plus) version of the core for Quartus II synthesis. You should indicate this folder as a user library in the Quartus II software before attempting to use the time-limited version of the FIR Compiler.
- **demo**
  Contains the files needed to run the standalone FIR wizard demonstration.
- **doc**
  Contains the FIR Compiler user guide (this document) in Adobe Acrobat Portable Document Format (**.pdf**), as well as other documentation in PDF and text files.
- **Misc.**
  Contains the coef_seq program, which re-orders the coefficients for multi-cycle variable FIR filters (see page 54).
- **sim_lib**
  Contains the simulation library files.
  - **matlab**
    Contains the MATLAB libraries for simulation.
  - **VHDL**
    Contains the VHDL libraries for simulation in the ModelSim software.
  - **Verilog**
    Contains the Verilog HDL libraries for simulation in the Verilog-XL or ModelSim software.
  - **Visual_IP**
    Contains the Visual IP simulation models.

Set Up Licensing

You can use Altera's OpenCore feature to compile and simulate the FIR Compiler MegaCore function, allowing you to evaluate it before purchasing a license. You can simulate your design in the Quartus II software using the OpenCore feature. However, you must obtain a license from Altera before you can generate programming files or EDIF, VHDL, or Verilog HDL gate-level netlist files for simulation in third-party EDA tools.

After you purchase a license for the FIR Compiler, you can request a license file from the Altera web site at http://www.altera.com/licensing and install it on your PC. When you request a license file, Altera e-mails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.

☞ If you want to use the OpenCore Plus feature, you must request a license file from the licensing page of the Altera web site (http://www.altera.com/licensing) to enable it. Your license file is sent to you via e-mail; follow the instructions below to install the license file.

To install your license, you can either append the license to your **license.dat** file or you can specify the core's license.dat file in the Quartus II software.

☞ Before you set up licensing for the FIR Compiler, you must already have the Quartus II software installed on your PC with licensing set up.

Append the License to Your license.dat File

To append the license, perform the following steps:

1. Close the following software if it is running on your PC:

   – Quartus II
   – MAX+PLUS II
   – LeonardoSpectrum
   – Synplify
   – ModelSim

2. Open the FIR Compiler license file in a text editor. The file should contain one FEATURE line, spanning 2 lines.

3. Open your Quartus II **license.dat** file in a text editor.

4. Copy the FEATURE line from the FIR Compiler license file and paste it into the Quartus II license file.

   ☞ Do not delete any FEATURE lines from the Quartus II license file.

5. Save the Quartus II license file.

   ☞ When using editors such as Microsoft Word or Notepad, ensure that the file does not have extra extensions appended to it after you save (e.g., **license.dat.txt** or **license.dat.doc**). Verify the filename in a DOS box or at a command prompt.

Specify the Core's License File in the Quartus II Software

To specify the core's license file, perform the following steps:

1. Create a text file with the FEATURE line and save it to your hard disk.

   ☞ Altera recommends that you give the file a unique name, e.g., <*core name*>**_license.dat**.

2. Run the Quartus II software.

3. Choose **License Setup** (Tools menu). The **Options** dialog box opens to the **License Setup** page.

4. In the **License file** box, add a semicolon to the end of the existing license path and filename.

5. Type the path and filename of the core license file after the semicolon.

   ☞ Do not include any spaces either around the semicolon or in the path/filename.

6. Click **OK** to save your changes.

FIR Compiler Tutorial

This tutorial explains how to create a basic parallel FIR filter using the Altera FIR Compiler MegaWizard® Plug-In and the Quartus II software. As you go through the wizard, each page is described in detail. When you are finished generating the filter, you can incorporate it into your system design.

☞ You can also run the FIR Compiler wizard without installing the Quartus II software. Execute the file **demo_fir.bat**, which is located in the <*path*>**\fir_compiler-v**<*version*>**\demo** directory.

You can use Altera’s OpenCore evaluation feature to compile and simulate the MegaCore functions, allowing you to evaluate the FIR Compiler before deciding to purchase a license. However, you must purchase a license before you can generate programming files or EDIF, VHDL, or Verilog HDL gate-level netlist files for simulation in third-party EDA tools.

This walkthrough consists of the following steps:

- “Create a New Quartus II Project” on page 23
- “The MegaWizard Plug-In Manager” on page 24
- “Specify the Coefficients” on page 27
- “Analyze the Coefficients” on page 31
- “Specify the I/O Number Formats and Bit Widths” on page 33
- “Choose the Architecture” on page 36
- “Simulate the Filter” on page 39
- “View the Report File” on page 41

Create a New Quartus II Project

Before you begin creating a filter, you must create a new Quartus II project. With the New Project Wizard, you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. You will also specify the FIR Compiler user library. To create a new project, perform the following steps:

1. Choose **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 **File** > **New Project Wizard**.
3. Click **Next** in the introduction (the introduction will not display if you turned it off previously).
4. Specify the working directory for your project. This walkthrough uses the directory **c:\qdesigns\fir_compiler**.
5. Specify the name of the project. This walkthrough uses **fir_compiler**.
6. Click **Next**.
7. Click **User Library Pathnames**.
8. Type <*path*>`\fir_compiler-v<`*`version`*`>\lib\` (or <*path*>`\fir_compiler-v<`*`version`*`>\lib_time_limited`, to use the OpenCore Plus-capable version) into the **Library name** box, where <*path*> is the directory in which you installed the FIR Compiler. The default installation directory is **c:\MegaCore**.
9. Click **Add**.
10. Click **OK**.
11. Click **Finish**.

You have finished creating your new Quartus II project.

The MegaWizard Plug-In Manager

The MegaWizard Plug-In Manager allows you to run a wizard that helps you easily specify options for the FIR Compiler. The wizard lets you generate coefficients, make I/O settings, specify a filter architecture, etc.

You can launch the MegaWizard Plug-In Manager from within the Quartus II software, or you can run it from the command line. The FIR Compiler wizard generates an instance of the megafunction that you can instantiate in your design. Table 4 highlights the main features of the wizard.

Table 4. FIR Compiler Wizard Options (Part 1 of 2)

Option	Wizard Page(s)	Description
Coefficient Specification	Start, Coefficient Generator, Generate Blank Coefficient Set	The FIR Compiler can read filter coefficients that have been exported from a third-party system-level application, generate coefficients using a built-in coefficient generator. In both cases, you can scale the coefficients and indicate the bits of precision. The wizard detects the filter symmetry and displays it. The wizard can also generate a blank set of coefficients. The built-in coefficient generator lets you specify the sample rate (either in Hertz or in relation to the Nyquist rate), the number of taps, and cut-off frequencies. The function supports low-pass, high-pass, band-pass, and band-reject filters. The supported filter windows include rectangular, Hanning, Hamming, and Blackman. As you change the coefficient settings, you can view the frequency and response of the filter dynamically.
Multi-Rate Filters	Coefficient Generator, Coefficient Analysis	You can use the wizard to create multi-rate filters using interpolation and decimation. You can specify the interpolation and decimation factors, as well as a polyphase output enable.
Filter Architecture	Architecture	For variable filters, you can choose a multi-cycle filter, and specify the number of cycles. The variable architecture supports pipelining, can read coefficients stored off-chip (e.g., in a memory device), supports one or more channels, supports pipelining, and supports multiple coefficient sets. For fixed filters, you can indicate whether the filter is parallel, serial, or multi-bit serial, and the number of channels for the filter. For serial and multi-bit serial filters, you can use either memory blocks or logic cells to implement the filter data and/or coefficient storage. You can specify whether or not to use pipelining.

Table 4. FIR Compiler Wizard Options (Part 2 of 2)

Option	Wizard Page(s)	Description
Input and Output Number Format and Bit Widths	I/O Specification	You can specify the width of the input data bus and the number system used. The wizard supports unsigned binary data, two's complement data, and signed binary fractional data (you can choose how many bits to use on either side of the decimal point). The FIR Compiler determines the output bit width for full precision based on the actual coefficient values and the input bit width. These two parameters define the maximum positive and negative output values. The wizard extrapolates 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 reduce the precision of your filter by removing bits from the most significant bit (MSB) via truncation or saturation, or least significant bit (LSB) via truncation or rounding.
Simulator and Simulation File Specification	Simulation Files	The FIR Compiler generates several types of simulation files, including Quartus II Vector Files (**.vec**), MATLAB M-Files (**.m**), Simulink Model Files (**.mdl**), Verilog HDL models, and VHDL output files. You can specify the clock frequency for the output files.

When you finish going through the wizard, it generates the following files based on your chosen options.

- Text Design File (**.tdf**) used to instantiate an instance of the FIR filter in your design
- Quartus II Vector File (**.vec**) used for simulation within the Quartus II environment
- Symbol File (**.sym**) used to instantiate the filter into a schematic design
- MATLAB and Simulink files used for simulation in MATLAB Simulink (**.m** and **.mdl**)
- VHDL and Verilog HDL models used for simulation in other EDA tools

Visual IP models are not generated, but are included in the installation directory, as shown in Figure 6 on page 19.

Launch the MegaWizard Plug-In Manager

Perform the following steps to launch the wizard and begin generating a filter.

1. Choose **Tools > MegaWizard Plug-In Manager**.

2. In Page 1 of the MegaWizard Plug-in Manager, select **Create a new custom megafunction variation** (default).

3. Click **Next**.

4. In Page 2a of the MegaWizard Plug-in Manager, expand the **Signal Processing** folder, under **Installed Plug-Ins,** by clicking the '+' next to the name.

5. In the same manner as Step 4, expand the **Filters** folder under **Signal Processing**.

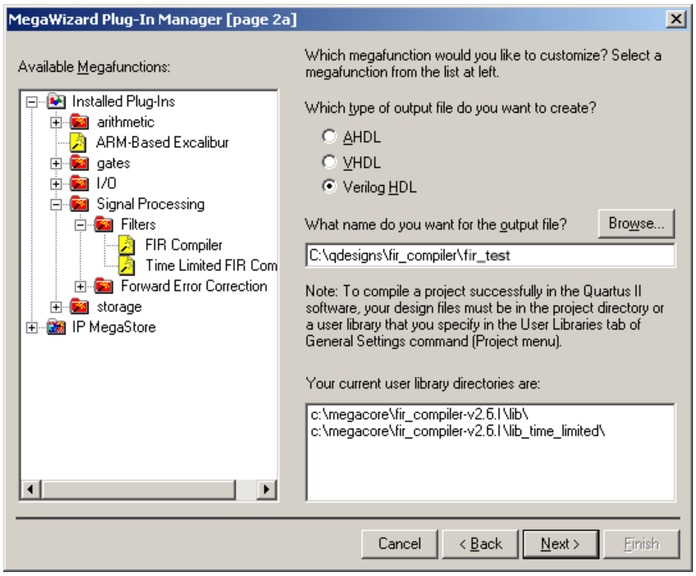
6. Select **FIR Compiler** or **Time-Limited FIR Compiler**.

7. Choose the output file type for your design; the wizard supports AHDL, VHDL, and Verilog HDL. This tutorial uses Verilog HDL, however, you can use any of the 3 languages.

8. In the "What name do you want for the output file?" field, type the name of the output file. This tutorial uses **fir_test**. Figure 7 on page 27 shows the wizard after you have made these settings.

9. Click **Next**.

Figure 7. Choose FIR Compiler in the MegaWizard Plug-In Manager


MegaWizard Plug-In Manager [page 2a]

Available Megafunctions:

- Installed Plug-Ins
  - arithmetic
  - ARM-Based Excalibur
  - gates
  - I/O
  - Signal Processing
    - Filters
      - FIR Compiler
      - Time Limited FIR Com
    - Forward Error Correction
  - storage
- IP MegaStore

Which megafunction would you like to customize? Select a megafunction from the list at left.

Which type of output file do you want to create?

AHDL
VHDL
[✓ Verilog HDL]

What name do you want for the output file? Browse...

C:\qdesigns\fir_compiler\fir_test

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory or a user library that you specify in the User Libraries tab of General Settings command (Project menu).

Your current user library directories are:

c:\megacore\fir_compiler-v2.6.1\lib\
c:\megacore\fir_compiler-v2.6.1\lib_time_limited\

You are now ready to set the options for your custom FIR filter.

Specify the Coefficients

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

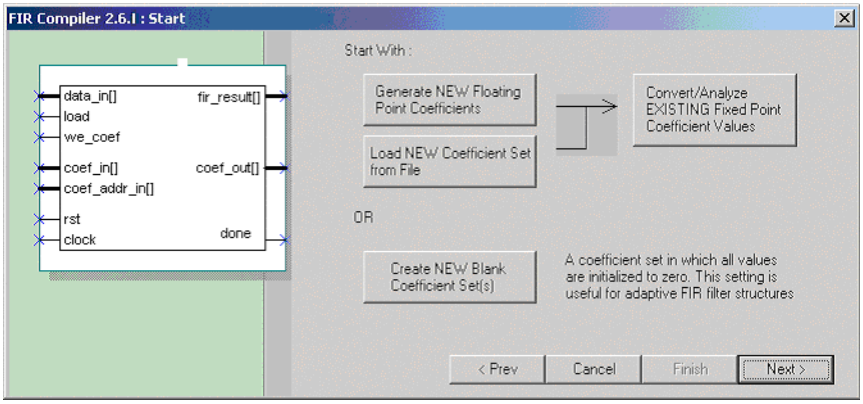
- You can use the FIR Compiler wizard to generate coefficients. The FIR Compiler coefficient generator supports a variety of filter types.
- You can load coefficients from a file. For example, you can create the coefficients in another application such as MATLAB, SPW, or a user-created program, save them to a file, and import them into the FIR Compiler wizard.
- 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.

Figure 8 shows the MegaWizard Plug-in Manager Start page (for the type of compiler we have specified in this sample walktrough), in which you choose how to obtain the coefficients. For this design walkthrough you will generate coefficients using the wizard's coefficient generator.

The sections of this chapter titled "Loading Coefficients from a File" and "Creating a Blank Set of Coefficients", both on page 30, describe the other two options.

☞ After you create a set of coefficients, if you click the MegaWizard Plug-in Manager's **Back** button, then navigate back to the page shown in Figure 8 and make another selection, the wizard creates a *new* set of coefficients. It *does not overwrite* the set you made previously. To change a set of coefficients, select the coefficient set and click **RECREATE THIS Coefficient Set** in the coefficient analysis tool. See "Analyze the Coefficients" on page 31, and Figure 11 on page 33, for more details.

Figure 8. Choose How to Obtain Coefficients


FIR Compiler 2.6.1 : Start
data_in[]
load
we_coef
coef_in[]
coef_addr_in[]
rst
clock
fir_result[]
coef_out[]
done
Start With :
Generate NEW Floating Point Coefficients
Load NEW Coefficient Set from File
Convert/Analyze EXISTING Fixed Point Coefficient Values
OR
Create NEW Blank Coefficient Set(s)
A coefficient set in which all values are initialized to zero. This setting is useful for adaptive FIR filter structures
< Prev
Cancel
Finish
Next >

Using the FIR Compiler Coefficient Generator

Click **Generate NEW Floating Point Coefficients** to launch the coefficient generator. 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.

To generate the coefficients for the simple parallel filter in this walkthrough, make the following settings in the coefficient generator, as shown on the completed wizard page in Figure 9.

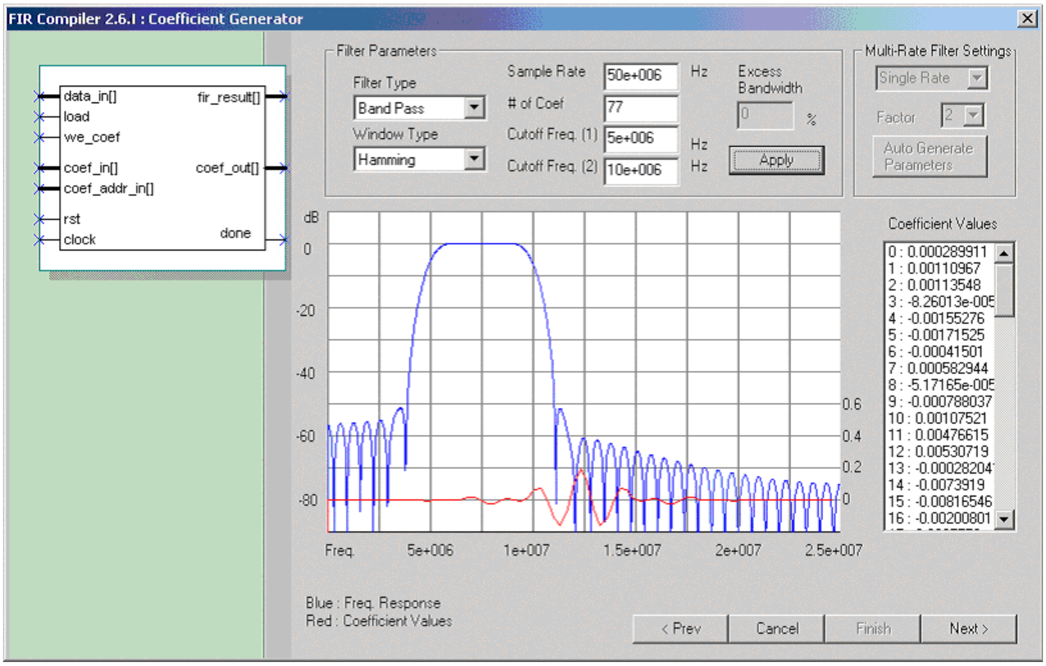
- **Filter Type:** Band Pass
- **Window Type:** Hamming
- **Sample Rate:** 50e+006
- **# of Coef:** 77
- **Cutoff Freq (1):** 5e+006
- **Cutoff Freq (2):** 10e+006

After you choose your settings, click **Apply** (in the **Filter Parameters** area). The wizard graphically displays the frequency response of the filter in blue and the coefficient values in red. The wizard also lists the actual coefficient values.

To quickly generate floating-point coefficients for multi-rate filters, use the options under **Multi-Rate Filter Settings**. You can choose **Interpolation** or **Decimation** and the **Factor**. When you click **Auto Generate Parameters**, the wizard generates coefficients for a low-pass filter with a cutoff frequency based on the same rate.

Refer to "Interpolation & Decimation" on page 57 for an explanation of interpolation and decimation.

Figure 9. Specify Parallel FIR Filter Coefficient Parameters



Filter Parameters
Filter Type: Band Pass
Window Type: Hamming
Sample Rate: 50e+006 Hz
# of Coef: 77
Cutoff Freq. (1): 5e+006 Hz
Cutoff Freq. (2): 10e+006 Hz
Excess Bandwidth: 0 %

Multi-Rate Filter Settings
Single Rate
Factor: 2

Coefficient Values
0 : 0.000289911
1 : 0.00110967
2 : 0.00113548
3 : -8.26013e-005
4 : -0.00155276
5 : -0.00171525
6 : -0.00041501
7 : 0.000582944
8 : -5.17165e-005
9 : -0.000788037
10 : 0.00107521
11 : 0.00476615
12 : 0.00530719
13 : -0.00028204
14 : -0.0073919
15 : -0.00816546
16 : -0.00200801

Blue : Freq. Response
Red : Coefficient Values

Click **Next** when you are finished making the parameter settings.

The Coefficient Analysis page is displayed, as shown in Figure 11.

Loading Coefficients from a File

To load a coefficient set from a file, click the **Load NEW Coefficient Set from File** button (refer back to Figure 8 on page 28). 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.

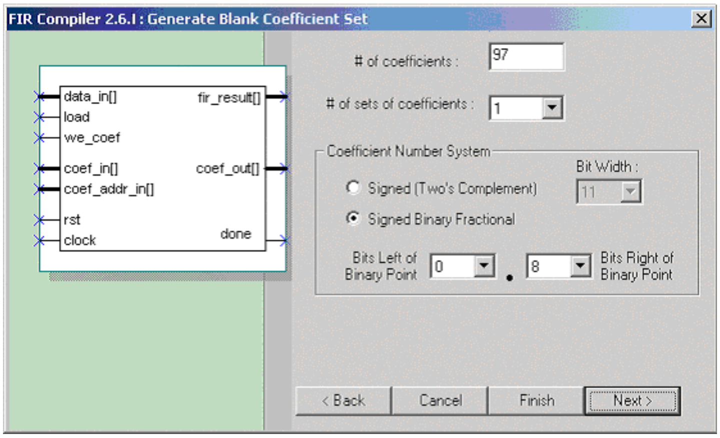
You cannot use the FIR Compiler coefficient analysis tool on blank sets of coefficients.

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

1. On the Start page, shown in Figure 8, click **Create NEW Blank Coefficient Set(s)**. The Generate Blank Coefficient Set window is displayed, as shown in Figure 10.

2. Specify information about the blank set, such as the number of coefficients, how many sets you want, and which number system to use. See Figure 10 for an example.

Figure 10. Generate a Blank Coefficient Set



# of coefficients : 97

# of sets of coefficients : 1

Coefficient Number System

Bit Width : 11

Signed (Two's Complement) [✓ Signed Binary Fractional]

Bits Left of Binary Point 0 . 8 Bits Right of Binary Point

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

The I/O Specifications window (Figure 12) is displayed.

Analyze the Coefficients

The FIR Compiler wizard contains a coefficient analysis tool, which you can use to create sets of coefficients and perform actions on each set. Some actions, such as scaling, apply to all sets. Other actions, such as recreating, reloading, or deleting, apply to the set you are currently viewing. The Coefficient Analysis tool is shown in Figure 11.

The FIR Compiler supports up to 32 sets of coefficients (multi-cycle architecture only). You can toggle between sets using the **Current Coef. Set** drop-down list box (the coefficient sets are numbered). When you select a set, the wizard displays the frequency response of the fixed-point coefficients in blue and the frequency response of the floating-point coefficients in green. It also displays the actual coefficient values.

The FIR Compiler supports signed binary fractional notation, which allows you to monitor which bits are preserved and which bits are removed during filtering. A signed binary fractional number has the format *<sign> <integer bits>***.***<fractional bits>*. A signed binary fractional number is interpreted as shown below, and in the following equation.

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

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

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

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

☞ If you specify signed binary fractional in the coefficient analysis tool (or in other wizard pages), Altera recommends that you specify signed binary fractional in all wizard pages; however, you are not required to do so.

When analyzing the coefficients, follow this design tip.

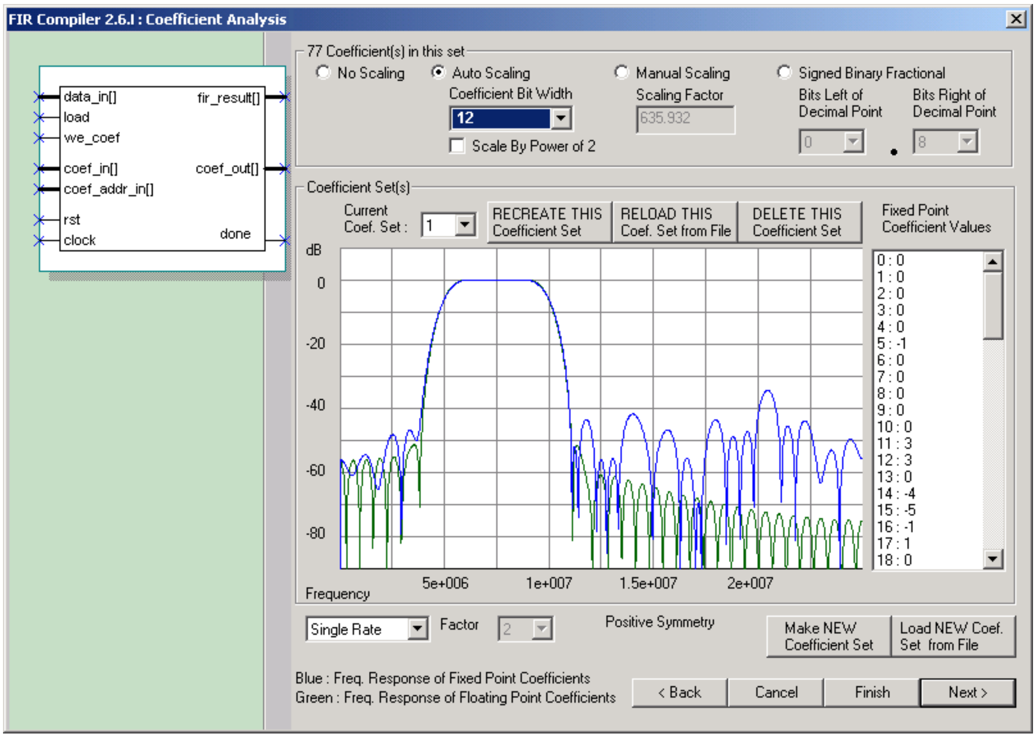
- The coefficient analysis tool shows the filter's symmetry. Symmetric filters tend to use fewer resources than asymmetric ones.

For this walkthrough, make the following selections in the Coefficient Analysis tool:

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

Figure 11 shows the Coefficient Analysis tool after you have made these 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 11. Analyze the Coefficients


FIR Compiler 2.6.1 : Coefficient Analysis

77 Coefficient(s) in this set

No Scaling [✓ Auto Scaling] Manual Scaling Signed Binary Fractional

Coefficient Bit Width: 12

Scale By Power of 2

Scaling Factor: 635.932

Bits Left of Decimal Point: 0 . Bits Right of Decimal Point: 8

Coefficient Set(s)

Current Coef. Set : 1

Fixed Point Coefficient Values

0 : 0
1 : 0
2 : 0
3 : 0
4 : 0
5 : -1
6 : 0
7 : 0
8 : 0
9 : 0
10 : 0
11 : 3
12 : 3
13 : 0
14 : -4
15 : -5
16 : -1
17 : 1
18 : 0

Single Rate

Factor: 2

Positive Symmetry

Blue : Freq. Response of Fixed Point Coefficients
Green : Freq. Response of Floating Point Coefficients

1. Click **Next** when you are finished making the parameter settings.

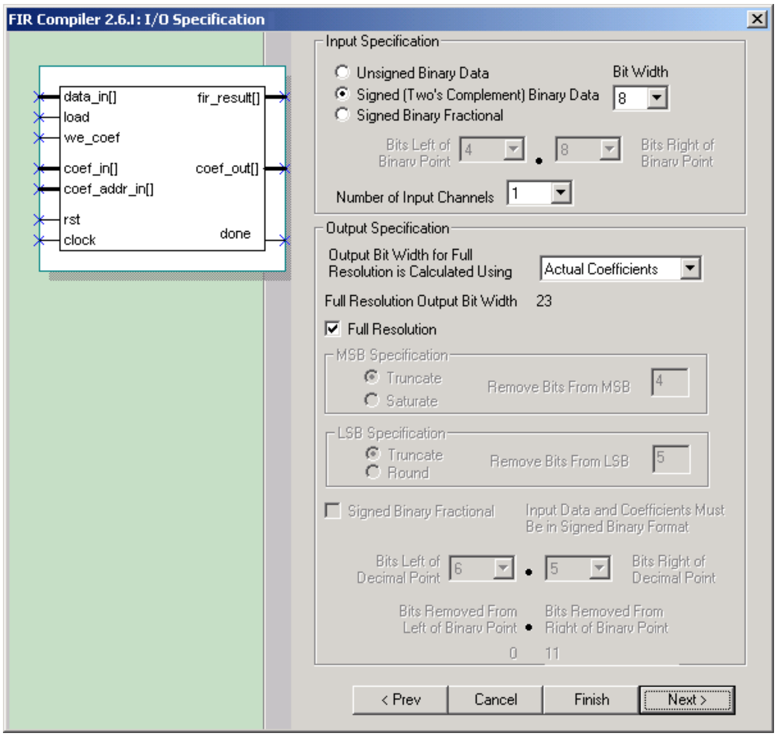
The I/O Specifications window (Figure 12) is displayed.

Specify the I/O Number Formats and Bit Widths

You can specify the number format for the inputs and the number of input channels (i.e., how many data streams will generate an output for each stream) in the I/O Specification window, shown in Figure 12.

☞ If you specify signed binary fractional in this window, or in other wizard pages, Altera recommends that you specify signed binary fractional in all wizard pages. However, you are not required to do so.

Figure 12. The I/O Specification window



Input Specification

Unsigned Binary Data
[✓ Signed (Two's Complement) Binary Data]
Signed Binary Fractional

Bit Width: 8

Bits Left of Binary Point: 4 • Bits Right of Binary Point: 8

Number of Input Channels: 1

Output Specification

Output Bit Width for Full Resolution is Calculated Using: Actual Coefficients

Full Resolution Output Bit Width: 23

[✓ Full Resolution]

MSB Specification

[✓ Truncate] Saturate

Remove Bits From MSB: 4

LSB Specification

[✓ Truncate] Round

Remove Bits From LSB: 5

Signed Binary Fractional

Input Data and Coefficients Must Be in Signed Binary Format

Bits Left of Decimal Point: 6 • Bits Right of Decimal Point: 5

Bits Removed From Left of Binary Point: 0 • Bits Removed From Right of Binary Point: 11

The wizard 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 which method you want in the **Output Bit Width for Full Resolution is Calculated Using** drop-down list box. The wizard 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 (`yout`). To use full precision, leave the **Full Resolution** option turned on (default). Turn it off to limit the precision.

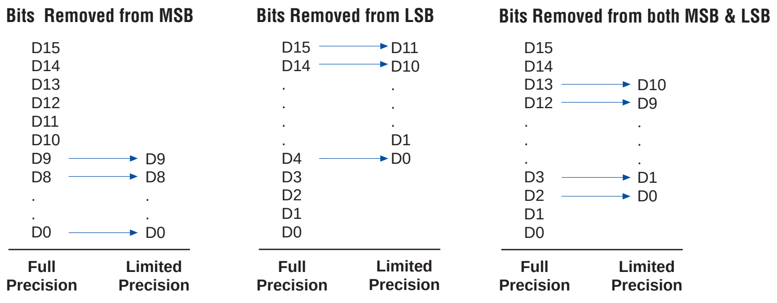
The wizard gives you the option of truncating or saturating the most significant bit (MSB) and / or rounding or truncating the least significant bit (LSB). Saturation, truncation, and rounding are non-linear operations. Table 5 shows the options for limiting the precision of your filter.

Table 5. Options for Limiting Precision

Bit Range	Option	Result
MSB	Truncate	In truncation, the filter disregards specified bits. See Figure 13.
	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 13 shows an example of removing bits from the MSB and LSB.

Figure 13. Removing Bits from the MSB and LSB



Instead of using the option shown in Table 5, you can use signed binary fractional notation to limit the number of bits. The wizard displays how many bits are removed.

When adjusting the I/O number formats and bit widths, follow these design 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 walkthrough uses the following default settings, shown in the I/O Specification window in Figure 12.

- 8-bit signed binary inputs
- Full resolution outputs

Click **Next** when you are finished making the parameter settings, and the Architecture window is displayed.

Choose the Architecture

The FIR Compiler supports several filter structures, including:

- Variable/fixed coefficient, multi-cycle
- Fixed coefficient, fully serial
- Fixed coefficient, multi-bit serial
- Fixed coefficient, fully parallel

Only the multi-cycle architecture supports multiple coefficient sets.

Table 6 describes three of the relative "trade-offs" for the different architecture options.

Table 6. Architecture Trade-Offs

Option	Area	Speed (data throughput)
Parallel	Large area	Creates a fast filter: 140 to 250 MSPS throughput with pipelining.
Serial	Small area	Requires multiple clock cycles for a single computation.
Pipelining	Creates a high-performance filter with only an area increase.	Increases throughput with additional latency.

Refer to "Functional Description" on page 49 for a detailed explanation about the filter architectures and how they operate. Also, see Figure 14.

The wizard automatically calculates and displays the resources that the filter will use in the **Resource** box at the lower-left corner of this window. It provides the estimated size in embedded memory blocks, DSP blocks, and logic cells. The **Information** box displays the number of clock cycles required to compute the result, along with numerous design “tips.” The latency (i.e., the number of clock cycles before the output is available) is shown in the FIR Compiler Report File (**<design name.htm>**), as shown in Figure 16 on page 41.

☞ 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 (e.g., 500 logic cells or less). For small filters, compile the design in the Quartus II software to obtain the resource usage.

Refer to the following tips when you are choosing a structure. Also, see Figure 14.

- Choosing embedded memory blocks (M512, M4K/EAB/ESB, M-RAM) for data storage will reduce the logic cell usage. Choosing “Auto” in data storage will allow Quartus II to use memory blocks instead of logic cells, when possible. This reduction in logic cell usage may increase the speed of the filter.

- Choosing M512 or M4K for coefficient storage, as compared to data storage, will result in a smaller reduction in logic cell usage.

- In the Stratix family, when you select the Multi-Cycle Variable structure, selecting DSP Blocks in the Multiplier pull-down menu will let the FIR Compiler use embedded DSP blocks for multipliers. This will result in a smaller and faster design in a device with enough DSP blocks for all multipliers.

- If you need to switch between multiple coefficient sets, select the Multi-Cycle Variable structure. If coefficients are stored in logic cells, the compiler can update one coefficient set at the same time as another set is being used for a calculation.

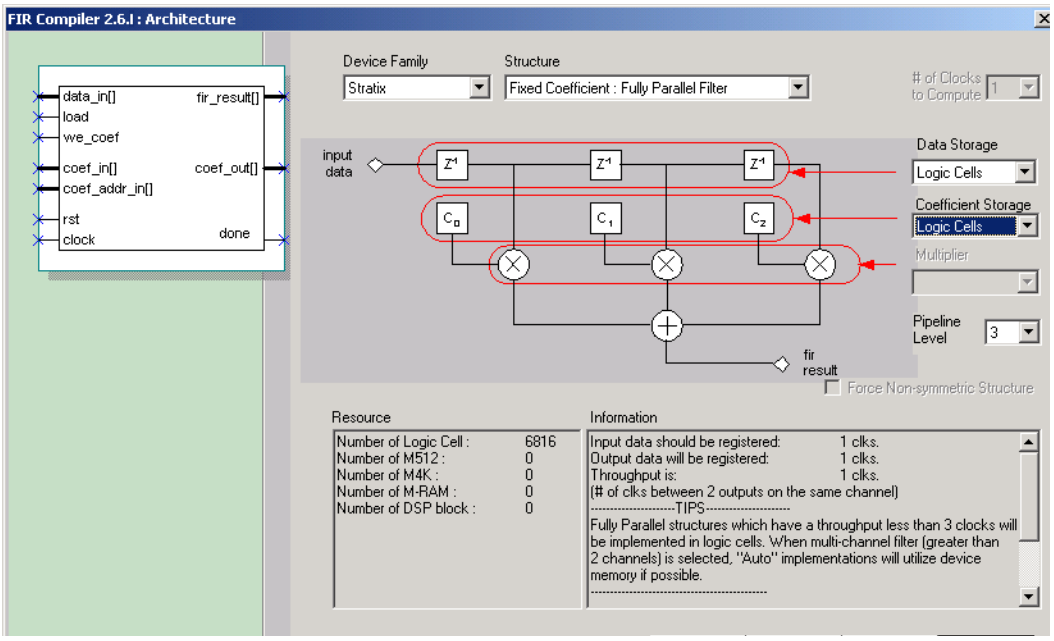
- For maximum clock speed, choose the Fully Serial Filter structure. In the Stratix family, M512 is faster than M4K and M-RAM. For maximum throughput, choose a fully parallel filter.

- In fully serial and multi-bit serial architectures, the structure symmetry default selection in the Stratix and Cyclone families is Force Non-symmetric Structure, even if your coefficients are symmetrical. The reason for this is symmetrical algorithms require an extra clock cycle per calculation cycle, which leads to lower throughput.

- Multi-Cycle Variable filters allow users to change coefficient values. These filters may contain optimizations for symmetrical filters. If you desire a filter which may need both symmetrical and non-symmetrical filters, select Force Non-symmetrical Structures in the Architecture page.

For this walkthrough, select a fully parallel structure with a pipeline level of 3. These settings create a filter that uses a large amount of logic cells, but only requires one clock cycle to compute the result. See Figure 14.

Figure 14. Specify the Filter Architecture



Device Family: Stratix
Structure: Fixed Coefficient : Fully Parallel Filter
# of Clocks to Compute: 1
Data Storage: Logic Cells
Coefficient Storage: Logic Cells
Pipeline Level: 3
Force Non-symmetric Structure

Resource

```
Number of Logic Cell :     6816
Number of M512 :           0
Number of M4K :            0
Number of M-RAM :          0
Number of DSP block :      0
```

Information

```
Input data should be registered:      1 clks.
Output data will be registered:       1 clks.
Throughput is:                        1 clks.
(# of clks between 2 outputs on the same channel)
--------------------TIPS--------------------
Fully Parallel structures which have a throughput less than 3 clocks will
be implemented in logic cells. When multi-channel filter (greater than
2 channels) is selected, "Auto" implementations will utilize device
memory if possible.
---------------------------------------------
```

Click **Next** when you are finished making the parameter settings.

Simulate the Filter

The FIR Compiler wizard includes a built-in simulation tool that has controls like those of an oscilloscope. The simulation display shows a plot of the bit-and cycle-accurate simulation of the filter response—the input data is shown in red and the output data is shown in yellow. The coefficient analysis tool only shows the effect of coefficient quantization. The simulation display shows the effects of limiting the I/O bit widths with truncation, saturation, and/or rounding.

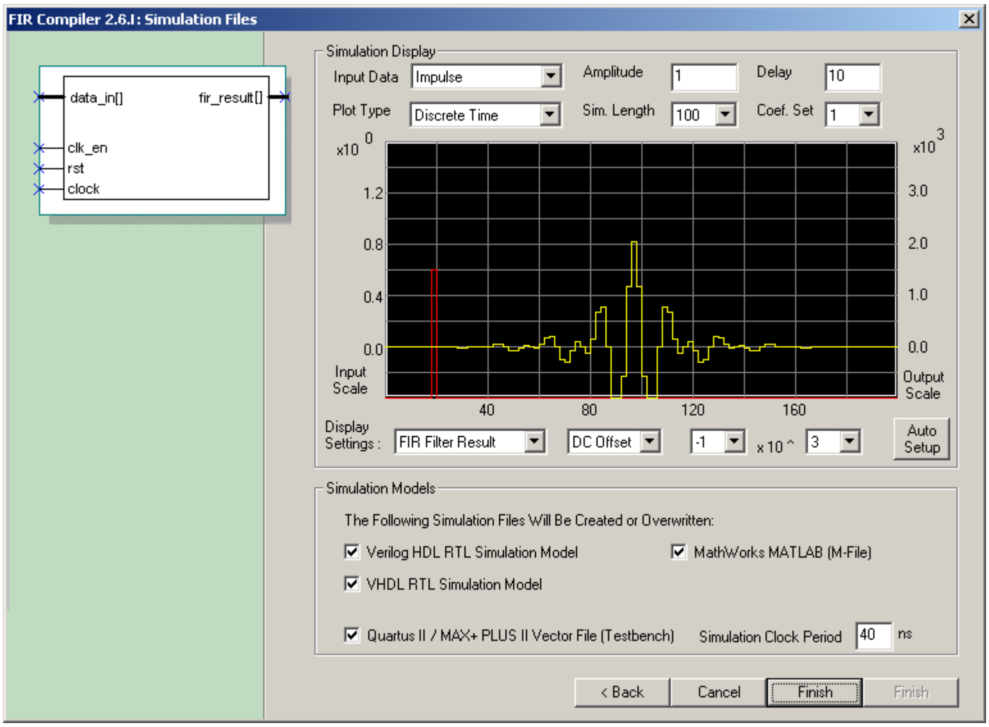
The simulation display is like an oscilloscope. You can adjust the simulator display's Y gain to zoom in or out and use the up and down arrows on your keyboard to adjust the display. Click the **Auto Setup** button to go back to the default view.

You can simulate using impulse, step, or random data, and you can specify the amplitude and simulation length. The simulation length should be greater than the number of coefficients to show a response. Using a longer simulation length gives you a more accurate display but takes more time to compute. You can also use a delay to shift the display for better viewing.

If your filter has multiple coefficient sets, you can view a plot for each set by toggling between sets using the **Coef. Set** drop-down list box.

Figure 15 shows the simulation display.

Figure 15. Simulation Display


FIR Compiler 2.6.1: Simulation Files

Simulation Display

Input Data: Impulse Amplitude: 1 Delay: 10

Plot Type: Discrete Time Sim. Length: 100 Coef. Set: 1

Display Settings: FIR Filter Result DC Offset -1 x 10 ^ 3

Simulation Models

The Following Simulation Files Will Be Created or Overwritten:

☑ Verilog HDL RTL Simulation Model ☑ MathWorks MATLAB (M-File)

☑ VHDL RTL Simulation Model

☑ Quartus II / MAX+ PLUS II Vector File (Testbench) Simulation Clock Period 40 ns

Under **Simulation Models**, you can specify which simulation files you want to output. The wizard generates a variety of simulation files for use with simulators such as MATLAB or ModelSim, and for simulation in the Quartus II software.

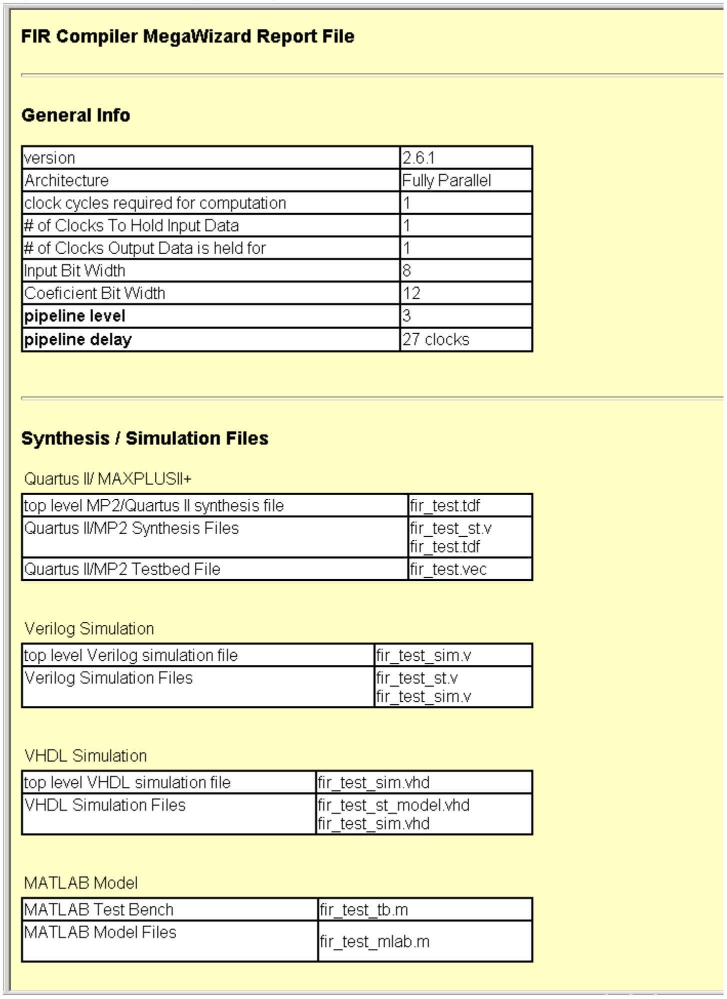
Click **Finish** when you are finished making settings. The wizard generates output files in your working project directory.

You are finished creating a FIR filter using the FIR Compiler wizard. Next, you can view the FIR Compiler report file, simulate a model of the filter in MATLAB/Simulink, perform VHDL or Verilog HDL simulation, or simulation in the Quartus II software. The Quartus II vector file provides a unit impulse response.

View the Report File

When it has finished generating output files, the FIR Compiler generates a report file in HTML format that contains information about the filter you created and a listing of simulation output files that were generated. This tutorial uses the output file name `fir_test`, as shown in Figure 7, therefore the name of the report file will be `fir_test.htm`. Figure 16 shows an example of a report file.

Figure 16. FIR Compiler Report File



FIR Compiler MegaWizard Report File

General Info

version	2.6.1
Architecture	Fully Parallel
clock cycles required for computation	1
# of Clocks To Hold Input Data	1
# of Clocks Output Data is held for	1
Input Bit Width	8
Coeficient Bit Width	12
pipeline level	3
pipeline delay	27 clocks

Synthesis / Simulation Files

Quartus II/ MAXPLUSII+

top level MP2/Quartus II synthesis file	fir_test.tdf
Quartus II/MP2 Synthesis Files	fir_test_st.v fir_test.tdf
Quartus II/MP2 Testbed File	fir_test.vec

Verilog Simulation

top level Verilog simulation file	fir_test_sim.v
Verilog Simulation Files	fir_test_st.v fir_test_sim.v

VHDL Simulation

top level VHDL simulation file	fir_test_sim.vhd
VHDL Simulation Files	fir_test_st_model.vhd fir_test_sim.vhd

MATLAB Model

MATLAB Test Bench	fir_test_tb.m
MATLAB Model Files	fir_test_mlab.m

Simulate Using Various Models

The FIR Compiler supports the ModelSim simulator with precompiled VHDL and Verilog models. The FIR compiler also supports the VerilogXL and NCVerilog simulators with protected simulation models. All other simulators are supported using pre-compiled Visual IP simulation models.

If you use the simulation library in the ModelSim PE, SE, or EE simulator, you must reinstall the FIR Compiler MegaCore function before you can use the library in the ModelSim-Altera simulator.

Compiling the VHDL Simulation Model in ModelSim

The FIR Compiler ships with a precompiled library for simulation in ModelSim. To simulate the core, complete the following steps.

1. At the ModelSim command prompt in your working directory, create a new library, `leaf_lib`, and map it to the FIR Compiler simulation library using the following commands:

   ```
   vmap -c
   ```

   `vmap leaf_lib` <*path*>`/fir_compiler-v`<*version*>`/sim_lib /vhdl/modelsim/wfir`

2. Create a new library, `work,` as follows:

   ```
   vlib work
   ```

3. Update the precompiled libraries to be compatible with your version of ModelSim using the **refresh** command:

   ```
   vcom -work leaf_lib -refresh
   ```

4. Compile your design.

   a. Open the FIR Compiler report file.

   b. Note the names and location of the VHDL simulation files.

   c. Compile all of the files using the `vcom` command in the order that they are listed in the report file.

Compiling the Verilog HDL Simulation Model in ModelSim

The FIR Compiler ships with a precompiled library for simulation in the ModelSim software. To simulate the core, you must complete the following steps.

1. At the ModelSim command prompt in your working directory, create a new library `work`, and map it to the FIR Compiler simulation library using the following commands:

   ```
   vmap -c

   vmap work <path>/fir_compiler-v<version>/sim_lib
   /verilog/modelsim/wfir
   ```

2. Update the precompiled libraries to be compatible with your version of ModelSim using the **refresh** command:

   ```
   vlog work -refresh
   ```

3. Compile your design.

   a. Open the FIR Compiler report file.

   b. Note the names and location of the Verilog HDL simulation files.

   c. Compile all of the files using the `vlog` command in the order that they are listed in the report file.

Simulating in Verilog-XL

You do not need to compile the models in Verilog-XL before simulating, however, you do need to:

- Copy the simulation files to your UNIX workstation (the files are located in the following directory.

  ```
  <path>/fir_compiler-v<version>/sim_lib
  /Verilog/VerilogXL
  ```

- Indicate the location of the FIR Compiler simulation library

☞ You can use the RealPC application by FWB software (http://www.fwb.com/) to emulate Windows and install the FIR Compiler function as you would on a PC.

After you have copied the files to your UNIX workstation, use the following command to specify the library and start simulation:

`verilog -v` *<path to FIR Compiler Verilog-XL simulation files> <FIR Compiler-generated files>*

Simulating in NCVerilog

You do not need to compile the models in NCVerilog before simulating, however, you do need to:

- Copy the simulation files to your UNIX workstation (the files are located in the *<path>*`/fir_compiler-v`*<version>*`/sim_lib /Verilog/VerilogXL` directory and are the same ones used for the Verilog-XL simulator).
- Indicate the location of the FIR Compiler simulation library.

☞ You can use the RealPC application by FWB software (http://www.fwb.com/) to emulate Windows and install the FIR Compiler function as you would on a PC.

After you have copied the files to your UNIX workstation, use the following command to specify the library and start simulation:

`ncverilog -v` *<path to FIR Compiler NCVerilog simulation files> <FIR Compiler-generated files>*

Simulating Using the Visual IP Model

Follow the instructions below to obtain the Visual IP software via the Internet. If you do not have Internet access, you can obtain the Visual IP software from your local Altera representative, as follows.

1. Point your web browser to: http://www.altera.com/products/ip/altera/visual_ip.html..

2. Follow the online instructions to download the function and save it to your hard disk.

Follow the instructions below to use the Visual IP software.

1. Set up your system to use the Visual IP software, as detailed in the Visual IP documentation (*Simulating Visual IP Models with the ModelSim Simulator for PCs White Paper, Simulating Visual IP Models with theNC-Verilog, Verilog-XL, VCS, or ModelSim [UNIX] Simulators White Paper*).

2. Ensure the appropriate ModelSim and Visual IP bin directories are in the following path.

   ```
   c:\modeltech\win32pe;c:\progra~1\visualIP\bin;
   ```

3. Set the VIP_MODELS_DIR environment variable to point to the directory containing the Visual IP models, i.e.:

   `set VIP_MODELS_DIR =`
   <path>`\fir_compiler-v`*<version>*`\sim_lib\visualIP\`

4. Start the ModelSim simulation tool.

5. Select **File > Change Directory**, and change the directory to your working directory for the simulator.

6. Create a new working library in this directory by selecting **Design > Create a New Library**.

7. Select a new library and a logical mapping to it and type `work` in the **Library** field.

8. Click **OK**.

9. The ModelSim software creates a settings file, **modelsim.ini,** in the working directory. Open this file in a text editor and search for the string `veriuser`. You should find the following line.

   ```
   ; Veriuser = veriuser.sl
   ```

   Remove the semi-colon (otherwise the line is treated as a comment and ignored), and change the directory name to where Visual IP is installed, i.e.:

   ```
   Veriuser = c:\progra~1\visualIP\bin\libplimtivip
   ```

   Save the **modelsim.ini** file and return to the ModelSim software.

10. Restore the leaf nodes of the FIR compiler in the following directory. *<path>***/fir_compiler-v***<version>***/sim_lib/VIP/fir_vip_files.tar.**

11. Compile VIP wrappers of all leaf nodes.

    The Verilog version of the wrapper is found in the **`$VIP_MODELS_DIR\`***<leaf_node>***`\interface\pli`** directory. The corresponding VHDL version is the **`$VIP_MODELS_DIR`** **`\<`***leaf_node***`>\interface\mti`** directory. For example, to compile

the Verilog wrapper from the ModelSim command line, enter the following command.
`vlog {$VIP_MODELS_DIR/`*<leaf_node>*`/interface/pli/`*<leaf_node>*`.v}`

where *<leaf_node>* are the leaf nodes restored from Step 10.

For the Visual IP VHDL simulation, compile the **LEAF_NODES_PACK.VHD** file, found at *<path>*`/fir_compiler-v`*<version>*`/sim_lib/VIP` before you compile any other leaf nodes.

12. Compile all of the design files generated by the FIR compiler. For the full set of file names, refer to the report file shown in Figure 16 on page 41.

Compiling and Simulating in the Quartus II Software

The following steps explain how to compile and simulate your design in the Quartus II software, and how to use the test vector configuration file.

1. Click **Start Compilation** (Processing Menu) to compile your design.

2. Click **Simulation Mode** (Processing menu). Choose **Simulator Settings** (Processing menu) and select the **Time/Vectors** tab. Turn off **Automatically Add Pin to Simulation Output Waveforms**. In the **Source of Vector Stimuli** box, select *<output name>*.**vec**, where *<output name>* is the name you specified in the MegaWizard Plug-In. Click **OK**.

3. Click **Run Simulation** (Processing menu) to begin simulation.

Synthesis, Compilation and Post-Place-&-Route Simulation

The Quartus II software works seamlessly with tools from all EDA vendors, including Cadence, Exemplar Logic, Mentor Graphics, Synopsys, Synplicity, and Viewlogic. After you have licensed the MegaCore function, you can generate EDIF, VHDL, Verilog HDL, and Standard Delay Output Files from the Quartus II software and use them with your existing EDA tools to perform functional modeling and post-place-and-route simulation of your design.

The following sections describe the design flow to compile and simulate your custom MegaCore design with a third-party EDA tool. To synthesize your design in a third-party EDA tool, and perform post-place-and-route simulation, perform the following steps:

1. Create your custom design instantiating a FIR Compiler MegaCore function.

2. Synthesize the design using your third-party EDA tool. Your EDA tool should treat the MegaCore instantiation as a black box by either setting attributes or ignoring the instantiation.

   ☞ For more information on setting compiler options in your third-party EDA tool, refer to the Quartus II Nativelink Guidelines.

3. After compilation, generate a hierarchical netlist file in your third-party EDA tool.

4. Open your netlist file in the Quartus II software.

5. Select **Compile mode** (Processing Menu).

6. Specify the Compiler settings in the **Compiler Settings** dialog box (Processing menu) or use the Compiler Settings wizard.

7. Specify the user libraries for the project and the order in which the compiler searches the libraries.

8. Specify the input settings for the project. Choose **EDA Tool Settings** (Project menu). Select **Custom EDIF** in the Design entry/synthesis tool list. Click **Settings**. In the **EDA Tool Input Settings** dialog box, make sure that the relevant tool name or option is selected in the **Design Entry/Synthesis Tool** list.

9. Depending on the type of output file you want, specify Verilog HDL output settings or VHDL output settings in the **General Settings** dialog box (Project Menu). Use the **1993 VHDL language** option.

10. Compile your design. The Quartus II Compiler synthesizes and performs place-and-route on your design, and generates output and programming files.

11. Import your Quartus II-generated output files (**.edo**, **.vho**, **.vo**, or **.sdo**) into your third-party EDA tool for post-route, device-level, and system-level simulation.

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. See Figure 17 on page 50 for a sample text file.
- 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 multi-channel filter by indicating two or more channels on page 7 of the wizard.
- 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. If the scaling (or gain) factor is too small, since coefficients are rounded towards the nearest integer, they may be set to zero. Therefore, 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 and Cyclone families, we recommend that you use memory blocks to reduce area.
- In the APEX, Mercury, or FLEX families, it is recommended that you use the *Fast* logic synthesis style to utilize the built-in “carry-and-cascade” chain. Following these recommendations will result in a smaller and faster filter.

Functional Description

The FIR Compiler has an interactive wizard-driven interface that allows you to create custom FIR filters easily. The wizard outputs simulation files for use with third-party tools, including MATLAB. The FIR Compiler supports up to 2047 taps.

Number Systems and Fixed-Point Precision

The FIR Compiler function supports signed or unsigned fixed-point numbers from 4 to 32 bits wide using unsigned binary, two's complement, or signed binary fractional numbers (for the variable architecture), or two's complement numbers (fixed-coefficient architecture). 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 17 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 17. 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. These coefficients are floating-point numbers and must be scaled. These values are displayed in the Coefficient Values scroll-box, at the right side of the Coefficient Generator page, as shown in Figure 9 on page 29.

When the FIR Compiler reads in the coefficients, it automatically determines any symmetry. The filter gives you several scaling options, e.g., scaling to a specified number of bits of precision or scaling by a user-specified factor. The scaled coefficients are displayed in the Fixed Point Coefficient Values scroll-box, at the right side of the Coefficient Analysis page, as shown in Figure 11 on page 33.

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 four scaling options:

- *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.

- *Limit scaling factors to powers 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.

- *Scale manually*—The FIR Compiler lets you manually scale the coefficient values by a specified gain factor.

- *Scale to a specified number of fractional bits*—You can specify how many digits to use on either side of the decimal point (supported in the variable architecture only).

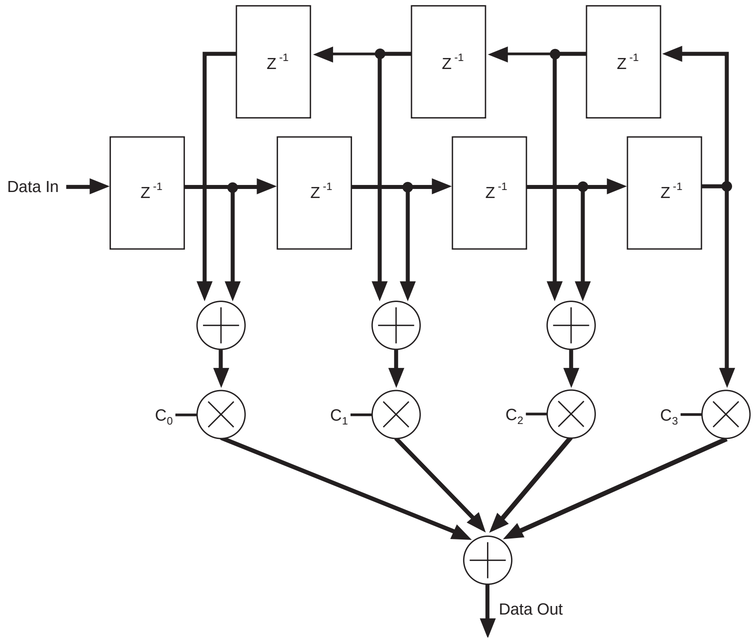
- *No scaling*—The FIR Compiler can read in pre-scaled integer values for the coefficients and not apply scaling factors.

Symmetrical Architecture Selection

Many FIR filters have symmetrical coefficient values. The FIR Compiler examines the coefficients and automatically determines the filter 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 even 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). If the filter has odd symmetry, two data points are subtracted prior to the multiplication step (again eliminating half of the multipliers). Odd and even filter structures are shown in Figures 18 and 19.

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

Figure 18. 7-Tap Symmetrical FIR Filter

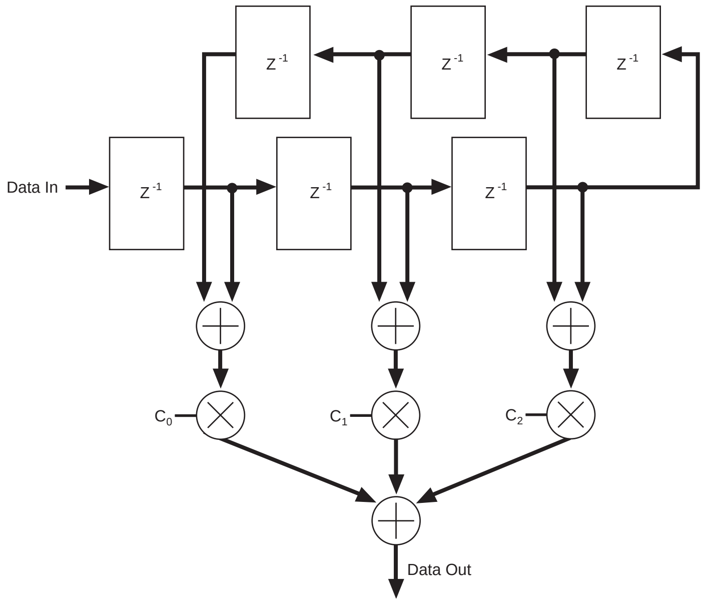


Symmetrical Serial

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.

Since non-symmetrical serial FIR filters do not require this resource, non-symmetrical filters may be smaller and/or faster. Use the Resource window on the Architecture page, shown in Figure 14, to determine the best solution.

Figure 19. 6-Tap Symmetrical FIR Filter



Structure Types

The FIR Compiler wizard generates variable, parallel, serial, multi-channel, and single and multi-cycle structures.

Multi-Cycle Variable Structures

Multi-cycle variable filters are optimized for high throughput. In a multi-cycle 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).

The multi-cycle variable structure allows 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. With the FIR Compiler, Altera provides source code for a C++ program that reorders the coefficients. Additionally, Altera provides a precompiled Windows executable that reorders the coefficients.

This 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**. You can add source code to your coefficient generation program, or use the executable file to re-order the coefficients.

The `coef_seq.exe` command is as follows.

`coef_seq.exe` <*input coefficient file,* with full path>
<*output coefficient file,* with full path> <*number of cycles to compute*>

Multi-Cycle Variable filters allow users to change coefficient values. These filters may contain optimizations for symmetrical filters. If you desire a filter which may need both symmetrical and non-symmetrical filters, select Force Non-symmetrical Structures in the Architecture page.

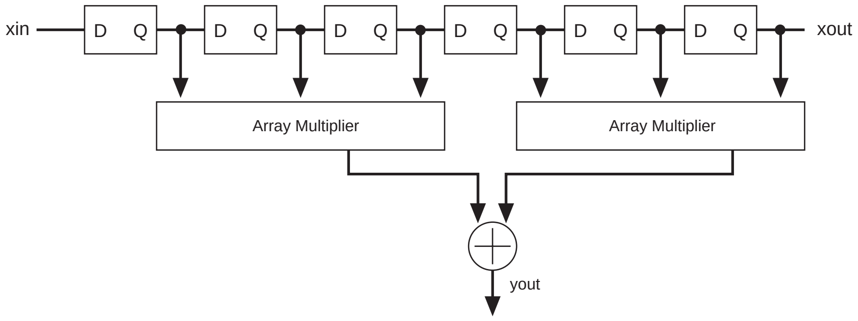
If you need to switch between multiple-set coefficients, select the Multi-Cycle Variable structure. If coefficients are stored in a logic cell, the compiler can update other coefficient sets at the same time as one set is being used for a calculation. If coefficients are stored in memory blocks, the calculation must be halted while it is updating any set of coefficients.

In the Stratix family, when you select the Multi-Cycle Variable structure, selecting DSP Blocks in the Multiplier pull-down menu will let the FIR Compiler use embedded DSP blocks for multipliers. This will result in a smaller and faster design in a device which contains enough DSP blocks for all multipliers.

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. Refer to Figure 27 on page 64 for a timing diagram of the parallel structure. Figure 20 shows the parallel filter block diagram.

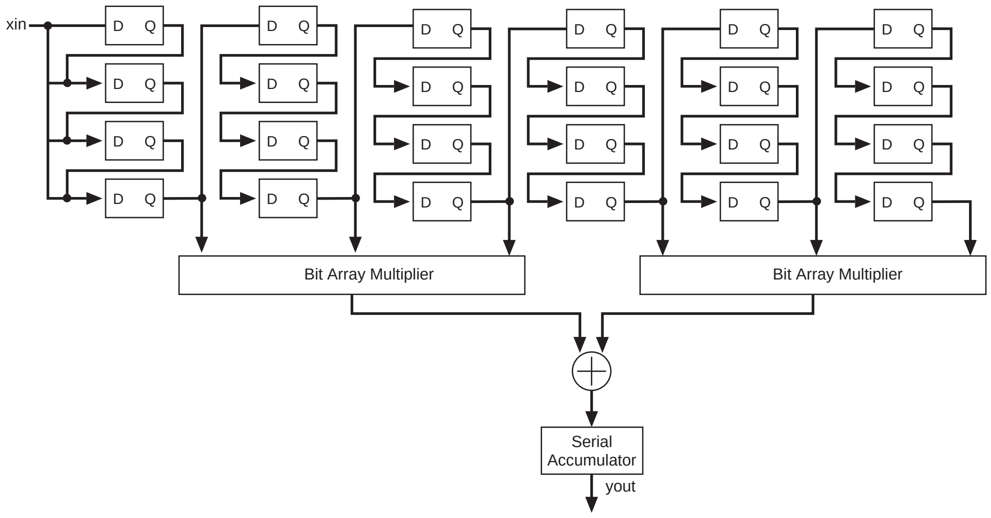
Figure 20. Parallel Filter Block Diagram



Serial Structures

A serial structure trades off area for speed. The filter processes input data one bit at-a-time per clock cycle. Therefore, serial structures require *N* clock cycles (where *N* is the input data width) to calculate an output. In the Stratix and Cyclone families, using memory blocks for data storage will result in a significant reduction in area. Refer to Figure 33 on page 67 for a timing diagram of the serial structure. Figure 21 shows the serial filter block diagram.

Figure 21. Serial Filter Block Diagram



Multi-Bit Serial Structure

A multi-bit 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 the designer to trade off area vs. speed. Figure 22 shows the multi-bit serial structure.

Figure 22. Multi-Bit Serial Structure

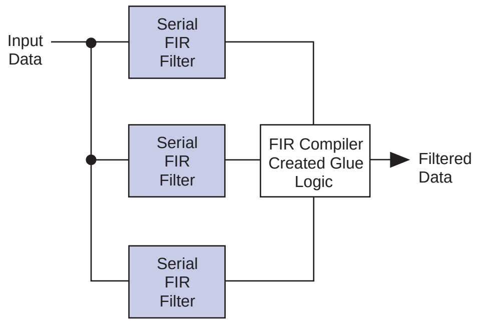
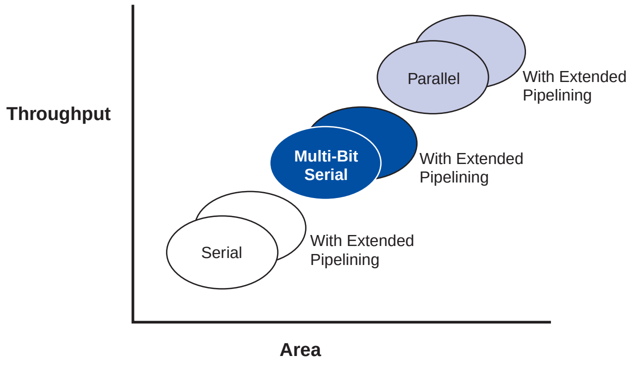


Figure 23 shows the area / speed "trade-off" of fixed FIR filters.

Figure 23. 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; 4 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 multi-bit serial FIR filter with 2 serial structures takes only 8 clock cycles to compute the result. Using 4 serial structures, only 4 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 3.

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.

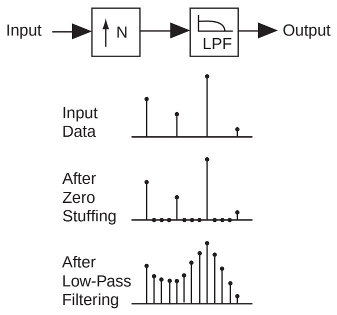
☞ For maximum area efficiency, use a distributed serial arithmetic architecture, multiple channels, and memory blocks for data and coefficient storage.

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.

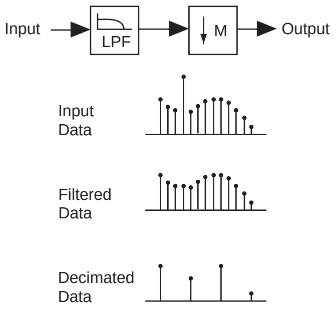
When a signal is interpolated, zeros are inserted between data points and the data is filtered to remove spectral components that were not present in the original signal. See Figure 24.

Figure 24. Signal Interpolation



To decimate a signal, a low-pass filter is applied, which removes spectral components that are not present at the low sample rate. After filtering, appropriate sample values are taken. See Figure 25.

Figure 25. Signal Decimation

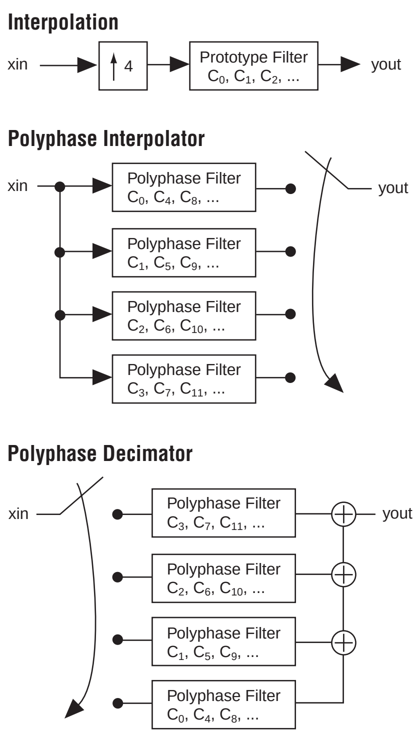


The FIR Compiler automatically creates interpolation and decimation filters using a polyphase decomposition. Polyphase decimation filters provide speed optimization because each filter runs at the output data rate. Polyphase interpolation filters provide the following benefits:

- *Speed optimization*—Each of the polyphase filters runs at the input data rate for maximum throughput.
- *Area optimization*—The polyphase interpolator shares resources.

Figure 26 shows block diagrams for polyphase interpolation and decimation.

Figure 26. Polyphase Interpolation & Decimation Block Diagrams



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 let you select whether to add 1, 2, or 3 pipeline levels.

Simulation Output Files

The FIR Compiler generates several types of output files for use in system simulation. After you have created a custom FIR filter, you can use the output files with MATLAB, VHDL, or Visual IP simulation tools. You can use the test vectors and MATLAB software to simulate your design. Visual IP models can be used with the Visual IP software, and are supported by other simulators. When you compile your FIR filter design, the FIR Compiler wizard generates MATLAB Simulink-compatible models for system verification.

The FIR Compiler includes a quick built-in simulator with impulse, step, and random inputs. You can view the results in time and frequency. The FIR wizard creates MATLAB M-Files, MATLAB Model Files, Altera Vector Files, VHDL, and Visual IP simulation models.

DSP Builder Feature & Simulation Support

You can create Simulink Model Files (**.mdl**) using FIR Compiler and DSP Builder blocks. DSP Builder supports the following FIR Compiler options:

- Fully parallel filters
- Fully serial filters

DSP Builder does not support the following FIR Compiler options:

- Multi-bit serial filters
- Multi-cycle variable filters

After you create your model, you can perform simulation. DSP Builder supports the simulation types shown in Table 7 for FIR Compiler.

Table 7. FIR Compiler Simulation File Support in DSP Builder

Simulation Type	Simulation Flow
Precompiled ModelSim model for RTL functional simulation	The DSP Builder SignalCompiler block generates a ModelSIm Tcl script and a VHDL testbench on-the-fly.
VHDL Output File (**.vho**) models for timing simulation	You can generate a **.vho** after you have purchased a license for your MegaCore function. Refer to the “VHDL Output File (.vho)“ topic in Quartus II Help for more information.
Visual IP(VIP) Models	The DSP Builder does not support generating scripts which use VIP models. Simulations with these models may be run manually.
Quartus II simulation	The DSP Builder SignalCompiler block generates a Quartus II simulation vector file on-the-fly.

If you are using the time-limited version of the FIR Compiler in your Model File, simulation does not time out in the Simulink simulation environment. The core only times out if you are performing hardware evaluation as described in "OpenCore Plus Time-Out Behavior" on page 61.

For more information on DSP Builder, see "DSP Builder Support" on page 14. Also see the DSP Builder User Guide at http://www.altera.com/products/software/system/products/dsp/dsp-builder.html

OpenCore Plus Time-Out Behavior

The following events occur when the OpenCore Plus hardware evaluation times out:

- `fir_result` is driven low
- `timed_out` is driven from low to high

A time-limited FIR Compiler runs for approximately 30 minutes for a 150 MHz clock (exactly 270,000,000,000 clock cycles).

For more information on OpenCore Plus hardware evaluation, see "OpenCore & OpenCore Plus Hardware Evaluation" on page 15 and *AN 176: OpenCore Plus Hardware Evaluation of MegaCore Functions.*

Core 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 testbenches are created and the results are compared to the MATLAB software using NC-Verilog and ModelSim simulators to exercise the Verilog HDL and VHDL models.

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 three 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.
- *Variable*—Designed for flexibility; the user specifies the number of cycles (multi-cycle) that the filter uses.

The FIR Compiler has the signals shown in Tables 8 through 11.

Table 8. Parallel, Serial & Multi-Bit Serial Signals

Signal	Direction	Description
`clk_en`	Input	Active-high clock enable.
`clock`	Input	Input clock signal.
`rst`	Input	Active-high signal that resets the FIR filter.
`data_in[data width-1..0]`	Input	Input data to be filtered.
`done`	Output	Indicates that the FIR calculation is complete and that the output is available.
`rdy_to_ld`	Output	Active-high signal that indicates the filter is ready to load new data on the data input pin.
`fir_result[FIR width-1..0]`	Output	Result of filtering operation performed by `data_in`.
`timed_out` *(1)*	Output	Signal used for OpenCore Plus hardware evaluation.

Note to Table 8:
(1) The `timed_out` signal is only generated if you are using the Open Core Plus version of the FIR Compiler.

Table 9. Multi-Cycle Variable Signals (One Coefficient Set)

Signal	Direction	Description
`clock`	Input	Input clock signal.
`clk_en`	Input	Clock enable.
`coef_in[]`	Input	New coefficients.
`coef_we`	Input	Active-high to enable coefficient updates.
`data_in[]`	Input	Input data to be filtered.
`rst`	Input	Active-high signal that resets the FIR filter.
`done`	Output	Indicates that the FIR calculation is complete and that the output is available.
`fir_result[FIR width-1..0]`	Output	Result of filtering operation.
`rdy_to_ld`	Output	Active-high signal that indicates the filter is ready to load new data on the data input pin.
`timed_out` *(1)*	Output	Signal used for OpenCore Plus hardware evaluation.

Note to Table 9:
(1) The `timed_out` signal is only generated if you are using the Open Core Plus version of the FIR Compiler.

Table 10. Multi-Cycle Variable Signals (Multiple Coefficient Sets; coefficients are stored in logic cells)

Signal	Direction	Description
`clock`	Input	Input clock signal.
`clk_en`	Input	Clock enable.
`coef_set_n_in[]`	Input	New coefficient value to overwrite coefficient set n. `0<=n<=q`, where q is the number of coefficient sets specified in the Wizard.
`coef_set_n_we[]`	Input	Coefficient set n write enable (active high). `0<=n<=q`, where q is the number of coefficient sets specified in the Wizard.
`coef_set[]`	Input	Selects which coefficient set the filter uses for the calculation.
`data_in[]`	Input	Input data to be filtered.
`rst`	Input	Active-high signal that resets the FIR filter.
`done`	Output	Indicates that the FIR calculation is complete and that the output is available.
`fir_result[FIR width-1..0]`	Output	Result of filtering operation.
`rdy_to_ld`	Output	Active-high signal that indicates the filter is ready to load new data on the data input pin.
`timed_out` *(1)*	Output	Signal used for OpenCore Plus hardware evaluation.

Note to Table 10:
(1) The `timed_out` signal is only generated if you are using the Open Core Plus version of the FIR Compiler.

.

Table 11. Multi-Cycle Variable Signals (Multiple Coefficient Sets; coefficients are stored in memory blocks)

Signal	Direction	Description
`clock`	Input	Input clock signal.
`rst`	Input	Active-high signal that resets the FIR filter.
`data_in[]`	Input	Input data to be filtered.
`clk_en`	Input	Clock enable.
`coef_set_in[]`	Input	Selects which coefficient set to overwrite.
`coef_set[]`	Input	Selects which coefficient set the filter uses for the calculation.
`coef_we[]`	Input	Enable coefficient overwrite.
`coef_in`		New coefficient value to overwrite the old coefficient.
`timed_out` *(1)*	Output	Signal used for OpenCore Plus hardware evaluation.

Note to Table 11:
(1) The `timed_out` signal is only generated if you are using the Open Core Plus version of the FIR Compiler.

Timing Diagrams

This section provides the timing diagrams for various types of filters.

Parallel Timing Diagrams

Figure 27 shows the input requirements for a parallel filter with no pipelining. Parallel filters generate an output every clock cycle.

Figure 27. Parallel Filter

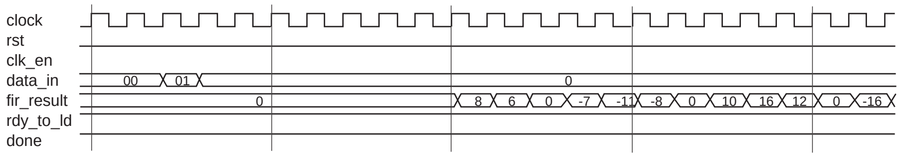
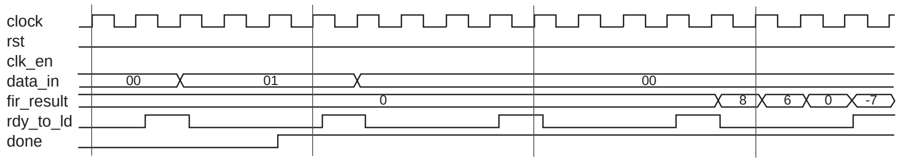


Figure 28 shows the timing diagram for a parallel interpolation filter with an interpolation factor of four. The filter has four polyphase outputs, each running at the input data rate. There is a final multiplexer that switches through all the filters. The input should be held until all output phases are shifted out.

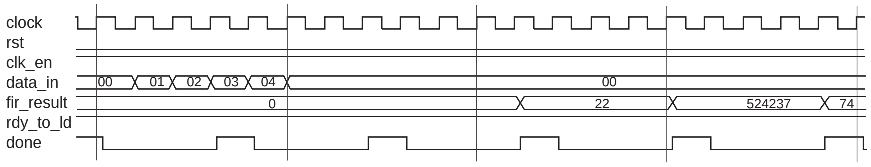
Figure 28. Parallel Interpolation Timing Diagram: Interpolation Factor = 4



The parallel case, which is the simplest for timing, illustrates the benefit of a polyphase decomposition. This technique relaxes the timing requirements on the FIR filter that is generated. If the input data rate is 50 MHz, and the interpolation factor is four, the polyphase filters must run at the 50 MHz data rate. The multiplexer, which switches through all the filters, will need to run at 200 MHz. Because the filter has fewer gates toggling at a slower rate, the design also saves power. Finally, a polyphase interpolation filter uses fewer resources than zero insertion followed by filtering.

Figure 29 shows a parallel decimation timing diagram with a decimation factor of four. The polyphase technique for decimation generates four filters, each of which operates at the output rate. At every clock cycle, the input data goes to the next polyphase. After four clock cycles, the outputs from each polyphase are added together.

Figure 29. Parallel Decimation Filter: Decimation Factor = 4



The benefits of a polyphase decomposition for decimation are twofold. Because the individual polyphase filters operate at the output clock rate, the timing requirements for the polyphase filter are relaxed. For example, a 4-to-1 decimation filter with an input data rate of 200 MSPS, would require 4 polyphase filters, each of which operate at a data rate of 50 MSPS. Additionally, the input data is time division multiplexed across 4 different filters with a switch rate of 200 MHz. The total system throughput is 200 MSPS (generated from 4 filters operating in parallel at a 50 MSPS rate).

Figure 30 shows a multi-channel parallel timing diagram with 2 channels.

Figure 30. Multi-Channel Parallel Timing Diagram

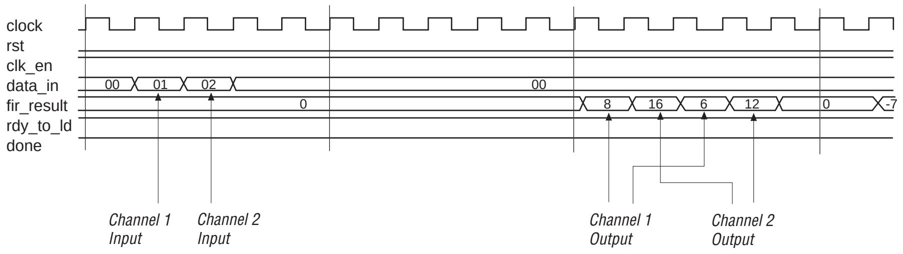


Figure 31 shows a multi-channel parallel interpolation timing diagram.

Figure 31. Multi-Channel Parallel Interpolation Timing Diagram

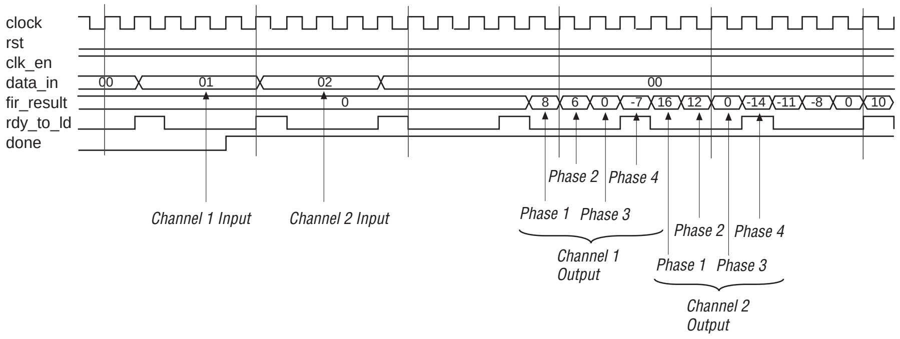
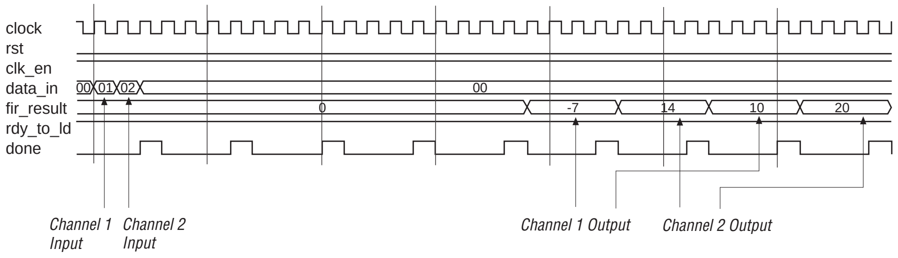


Figure 32 shows a multi-channel parallel decimation timing diagram.

Figure 32. Multi-Channel Parallel Decimation Timing Diagram



Serial & Multi-Bit Serial Timing Diagrams

This section provides timing diagrams and information on controlling the rate of serial and multi-bit serial filters.

Serial Timing Diagrams

Figure 33 shows the input timing diagram for an 8-bit serial filter.

Figure 33. 8-Bit Serial Filter

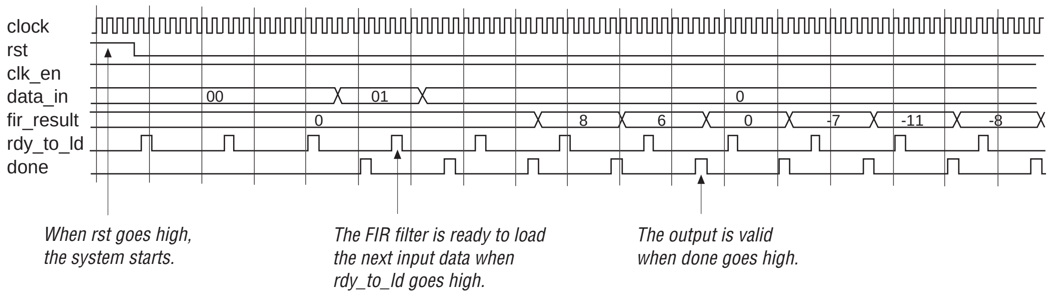
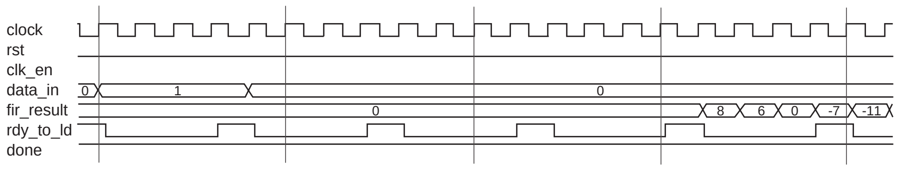

clock
rst
clk_en
data_in
fir_result
rdy_to_ld
done
00
01
0
0
8
6
0
-7
-11
-8
When rst goes high, the system starts.
The FIR filter is ready to load the next input data when rdy_to_ld goes high.
The output is valid when done goes high.

Figure 34 shows the timing diagram for a serial interpolation filter in which the interpolation factor is equal to the input data width (both have a value of four). The filter has four polyphase outputs.

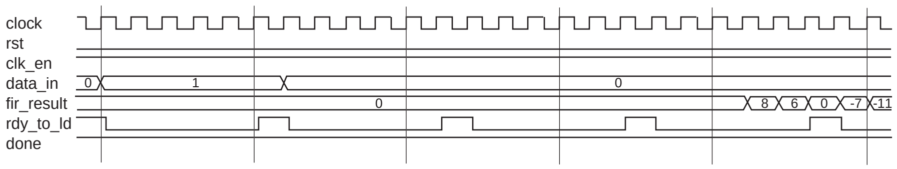
Figure 34. Interpolation Factor = Input Data Width



The structure runs at a 4× clock. The input data is held for 4 clock cycles, and each polyphase is computed every 4 clocks. The interpolation scheme switches through the four outputs every clock cycle to generate `fir_result` (the final output). The FIR Compiler provides access to the polyphase outputs, which allows you to multiplex through the outputs to suit the needs of your application.

Figure 35 shows the timing diagram for a filter in which the interpolation factor (six) is greater than the input data width (four). The filter has six polyphase outputs.

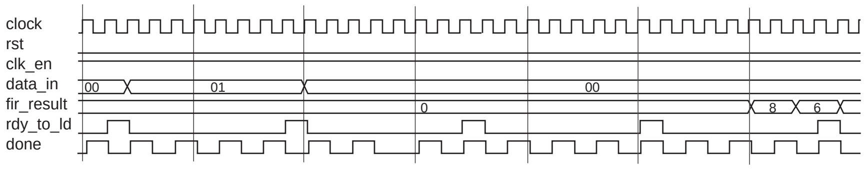
Figure 35. Interpolation Factor > Input Data Width



The entire structure runs at a 6× clock. The input data is held for 6 clock cycles. There are six serial filters, and each filter calculates a particular phase. Each of the six serial filters requires 4 clock cycles to compute a phase because there are 4 bits of input data. However, six clock cycles are needed to switch through all the filters, so the entire design requires a 6× clock.

Figure 36 shows the timing diagram for a filter in which the interpolation factor (four) is less than the input data width (six). The filter has four polyphase outputs.

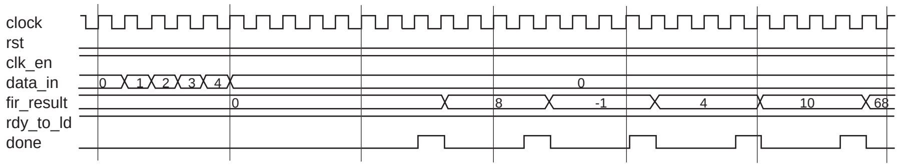
Figure 36. Interpolation Factor < Input Data Width



For this filter, a 4× clock does not provide enough cycles to calculate an individual polyphase output. To ensure a constant output data rate, the FIR Compiler uses an 8× clock (or a clock rate of two times the interpolation factor), switching between every polyphase output every two clocks. The 8× clock provides sufficient clock cycles to perform the serial calculation.

Figure 37 shows a serial decimation filter in which the decimation factor (four) equals the input bit width (four).

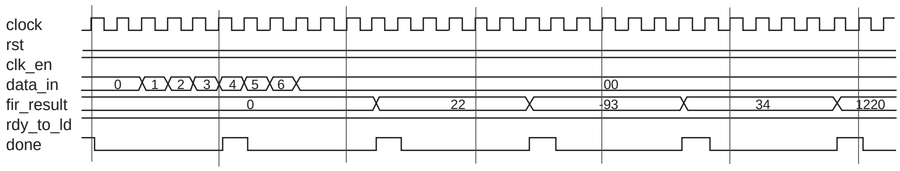
Figure 37. Decimation Factor = Input Bit Width



In this case, the FIR Compiler generates four serial filters because the decimation factor is four. Each of the decimation filters requires four clock cycles to generate an output. The decimation scheme switches through the four filters individually and adds the result of four filters together to generate a final decimated output.

Figure 38 shows a serial decimation filter in which the decimation factor (six) is greater than the input data width (four).

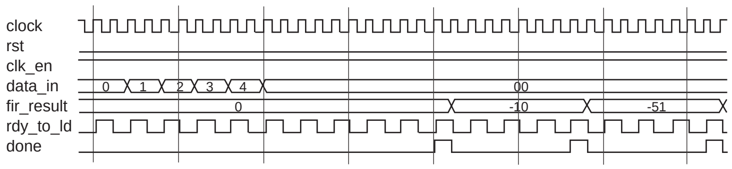
Figure 38. Decimation Factor > Input Data Width



The entire structure operates with a 6× clock. The input data is held constant while it is switched between the polyphase filter (in this case, for six clock cycles). The structure has six serial filters, and each filter calculates a particular phase. Each of the six serial filters requires four clock cycles to compute a phase (because there are four bits of input data). The entire computation requires the results from the six polyphase filters, so a 6× clock relative to the output rate is sufficient.

Figure 39 shows a filter in which the decimation factor (four) is less than the input data width (six).

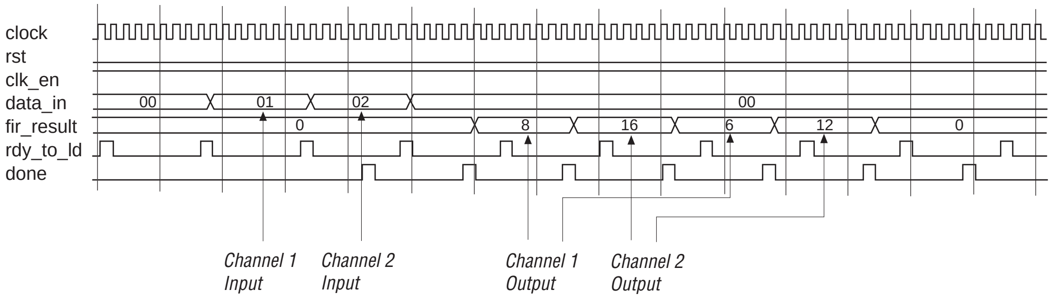
Figure 39. Decimation Factor < Input Data Width



The FIR Compiler generates four polyphase filters. Each filter requires at least 6 clock cycles to generate an output because they are serial filters with input data widths of six bits. Therefore, a single 4× clock is not sufficient to create the structure. By providing twice the clock rate (8×) there are enough clock cycles to compute the polyphase result; i.e., the input data is held for two clock cycles for each polyphase input. Eight clock cycles total are required for the structure to operate. Additionally, each of the polyphase outputs is available for use.

Figure 40 shows a multi-channel serial timing diagram with 2 channels and
8-bit input data.

Figure 40. Multi-Channel Serial Timing Diagram



Multi-Bit Serial Timing Diagrams

Figure 41 shows the timing diagram for a multi-bit serial filter. This multi-bit serial filter has 2 serial units and the input data is 8 bits. A standard serial structure requires 8 clock cycles to compute the result. In contrast, the multi-bit serial structure requires 4 clock cycles. The disadvantage is that the multi-bit serial implementation uses four times as many resources (LEs and EABs / ESBs) than a serial filter to achieve this 4x improvement. However, the multi-bit serial implementation still uses fewer resources than the parallel implementation.

Figure 41. Multi-Bit Serial Timing Diagram

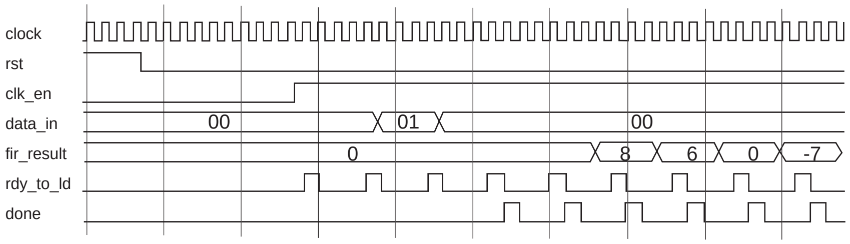


Figure 42 shows the same filter with 8 bits on input data and 2 serial channels implemented using 2 channels. One set of data feeds into the first channel, and another feeds into the second channel. A serial filter would take 8 clock cycles to compute a single channel. However, the multi-bit serial filter only takes 4 cycles to compute a single channel. The channels are time interleaved, and the filter generates a result for each channel 4 clock cycles apart.

Figure 42. Multi-Channel Multi-Bit Serial Timing Diagram

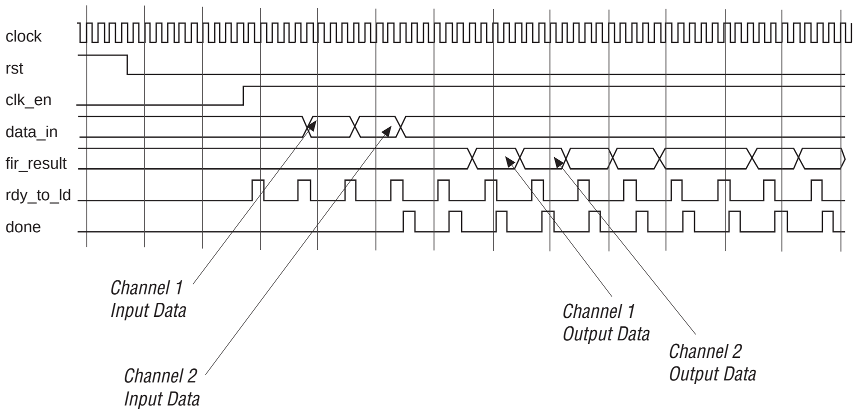


Figure 43 shows a multi-bit serial interpolation timing diagram with an interpolation factor of 4, an 8-bit input width, and 2 serial structures. The interpolation factor is equal to input data width divided by the number of serial structures. This timing diagram is similar to the serial interpolation timing diagram.

Figure 43. Multi-Bit Serial Interpolation Timing Diagram

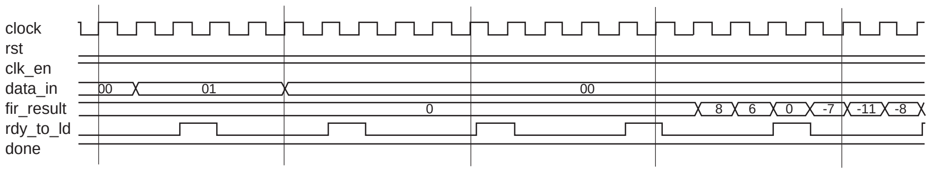
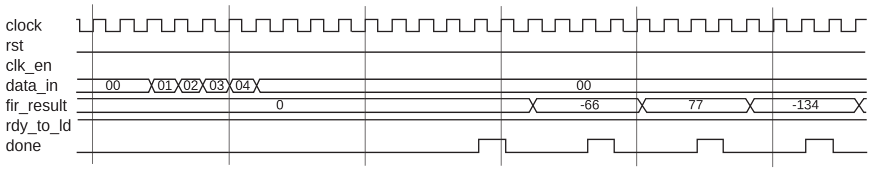


Figure 44 shows a multi-bit serial decimation timing diagram with a decimation factor of 4, an 8-bit input data width, and 2 serial structures. The entire structure runs ar 4x the clock speed because the input data width is broken down into the input data width divided by the number of serial structures. This timing diagram is similar to the serial decimation timing diagram.

Figure 44. Multi-Bit Serial Decimation Timing Diagram



Variable Timing Diagrams

This section provides timing diagrams of multi-cycle variable filters.

Multi-Cycle Variable Timing Diagrams

Figure 45 shows a multi-cycle FIR filter example with 4 cycles. The input and output data is held for 4 clock cycles.

Figure 45. Multi-Cycle Variable with 1 Channel & 4 Cycles Timing Diagram

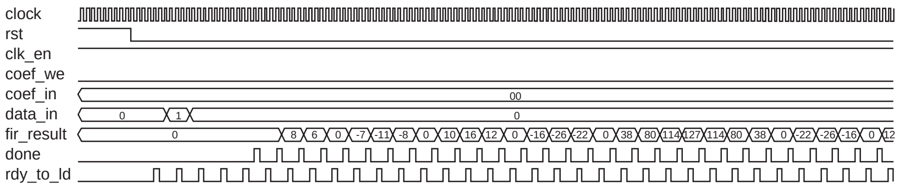


Figure 46 shows a multi-cycle variable filter with 4 channels.

Figure 46. Multi-Cycle Variable with 4 Channels TIming Diagram

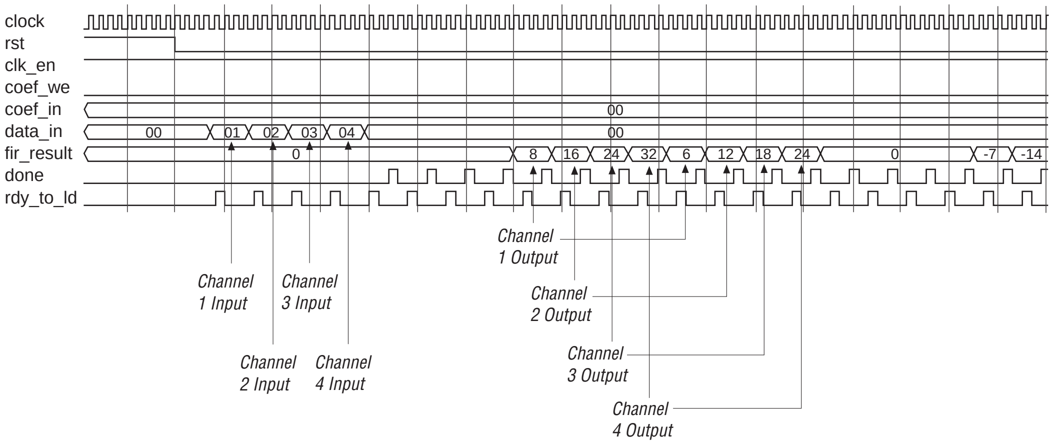


Figure 47 shows a multi-cycle variable timing diagram with 4 cycles and 2 coefficient sets. When the coefficient set switches from 0 to 1, the calculation switches from using coefficient set 0 to coefficient set 1 (there is a calculation latency).

Figure 47. Multi-Cycle Variable with 4 Cycles & 2 Coefficient Sets Timing Diagram

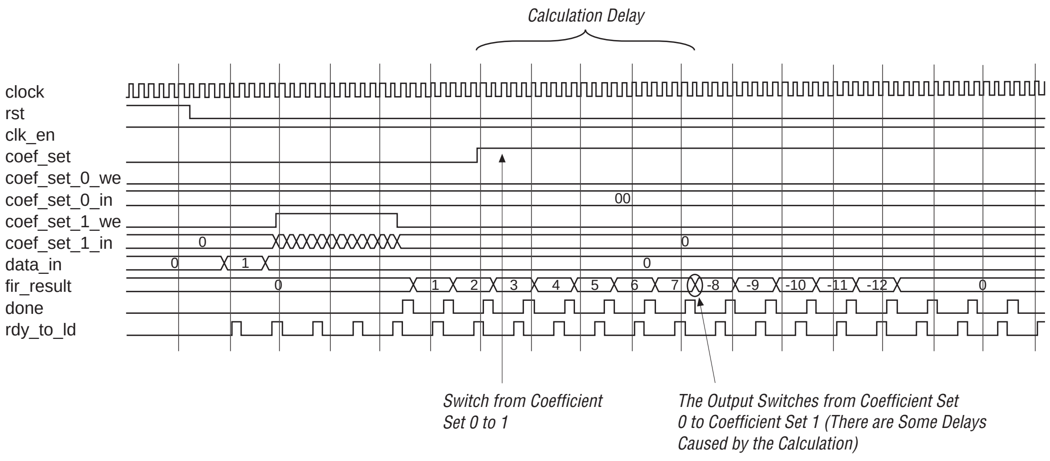
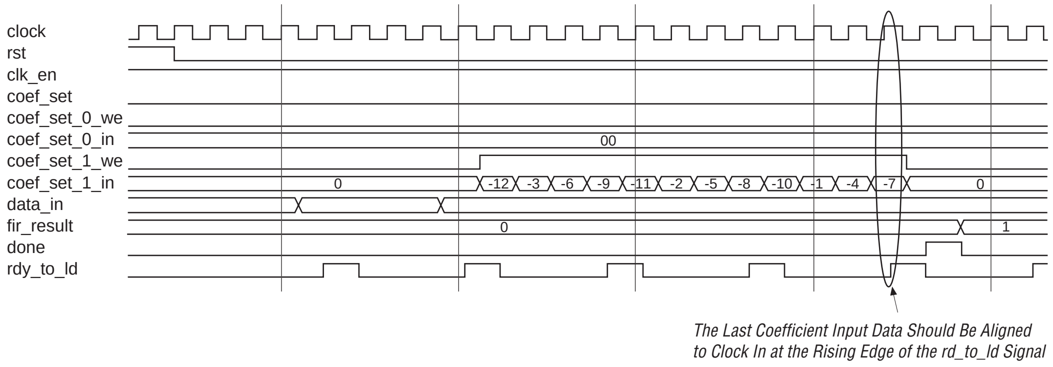

Calculation Delay
clock
rst
clk_en
coef_set
coef_set_0_we
coef_set_0_in
coef_set_1_we
coef_set_1_in
data_in
fir_result
done
rdy_to_ld
00
0
1
1
2
3
4
5
6
7
-8
-9
-10
-11
-12
Switch from Coefficient Set 0 to 1
The Output Switches from Coefficient Set 0 to Coefficient Set 1 (There are Some Delays Caused by the Calculation)

Figure 48 shows the timing for coefficient loading. When coefficient set 0 is in use, you can load coefficient set 1 by making `coef_set_1_we` go high. The filter switches to using coefficient set 1 when loading completes. Coefficient set 1 loads after sequence adjustment. The filter clocks in one coefficient on each clock cycle. If the coefficients are symmetric and the cycle is 1, the filter only needs to read in half of the coefficients.

Figure 48. Multi-Cycle Variable Coefficient Loading Timing Diagram


clock
rst
clk_en
coef_set
coef_set_0_we
coef_set_0_in
coef_set_1_we
coef_set_1_in
data_in
fir_result
done
rdy_to_ld
00
0
-12
-3
-6
-9
-11
-2
-5
-8
-10
-1
-4
-7
0
0
1
The Last Coefficient Input Data Should Be Aligned to Clock In at the Rising Edge of the rd_to_ld Signal