FIR Compiler MegaCore Function User Guide

September 1999

Printed on Recycled Paper.

About this User Guide

September 1999

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

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

How to Contact Altera

For additional information about Altera products, consult the sources shown in Table 1.

Table 1. How to Contact Altera

Information Type	Access	USA & Canada	All Other Locations
Altera Literature Services	Telephone hotline	(888) 3-ALTERA *(1)*	(888) 3-ALTERA *(1)*
	Electronic mail	**lit_req@altera.com** *(1)*	**lit_req@altera.com** *(1)*
Non-technical customer service	Telephone hotline	(800) SOS-EPLD	(408) 544-7000
	Fax	(408) 544-7606	(408) 544-7606
Technical support	Telephone hotline (6:00 a.m. to 6:00 p.m. Pacific Time)	(800) 800-EPLD	(408) 544-7000 *(1)*
	Fax	(408) 544-6401	(408) 544-6401 *(1)*
	Electronic mail	**sos@altera.com**	**sos@altera.com**
	FTP site	**ftp.altera.com**	**ftp.altera.com**
General product information	Telephone	(408) 544-7104	(408) 544-7104 *(1)*
	World-wide web site	**http://www.altera.com**	**http://www.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 2.

Table 2. Conventions

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names and dialog box titles 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: $\mathbf{f_{MAX}}$, **\maxplus2** directory, **d:** drive, **chiptrip.gdf** file.
Bold italic type	Book titles are shown in bold italic type with initial capital letters. Example: ***1999 Data Book***.
Italic Type with Initial Capital Letters	Document titles, checkbox options, and options in dialog boxes are shown in italic type with initial capital letters. Examples: *AN 75 (High-Speed Board Design)*, the *Check Outputs* option, the *Directories* box in the **Open** dialog box.
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 MAX+PLUS® II Help topics are shown in quotation marks. Example: "Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable."
`Courier type`	Reserved signal and port names are shown in uppercase Courier type. Examples: `DATA1`, `TDI`, `INPUT`. User-defined signal and port names are shown in lowercase Courier type. Examples: `my_data`, `ram_input`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\max2work\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.

Introduction

September 1999, ver. 1.10 User Guide

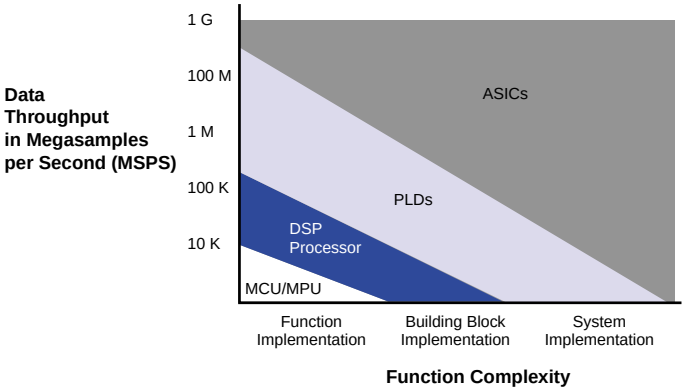
Altera MegaCore Functions

As programmable logic device (PLD) densities grow to over 1 million gates, design flows must be as efficient and productive as possible. Altera provides ready-made, pre-tested, and optimized megafunctions that let you rapidly implement the functions you need, instead of building them from the ground up. Altera® MegaCore™ functions, which are reusable blocks of pre-designed intellectual property, improve your productivity by allowing you to concentrate on adding proprietary value to your design. When you use MegaCore functions, you can focus on your high-level design and spend more time and energy on improving and differentiating your product.

Digital Signal Processing Functions

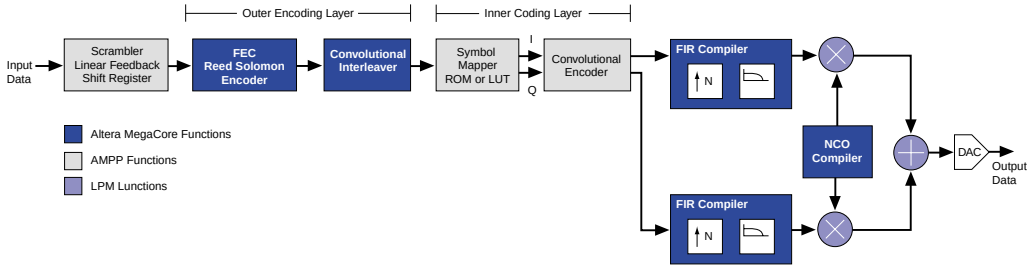
Traditionally, designers have been forced to make a tradeoff between the flexibility of digital signal processors, and the performance of application-specific integrated circuits (ASICs) and application-specific standard products (ASSPs) digital signal processing (DSP) solutions. The Altera FLEX® and APEX™ DSP solution eliminates the need for this tradeoff by providing exceptional performance combined with the flexibility of PLDs. See Figure 1.

Figure 1. 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). Altera devices easily implement DSP applications, while leaving ample room for your custom logic. The devices are supported by Altera's MAX+PLUS® II and Quartus™ development systems, which allow you to perform a complete design cycle including design entry, synthesis, place-and-route, simulation, timing analysis, and device programming. Altera devices, software, and DSP MegaCore functions provide you with a complete design solution. Figure 2 shows a hypothetical DSP system and highlights the functions that are available from Altera, the LPM, and AMPP partners.

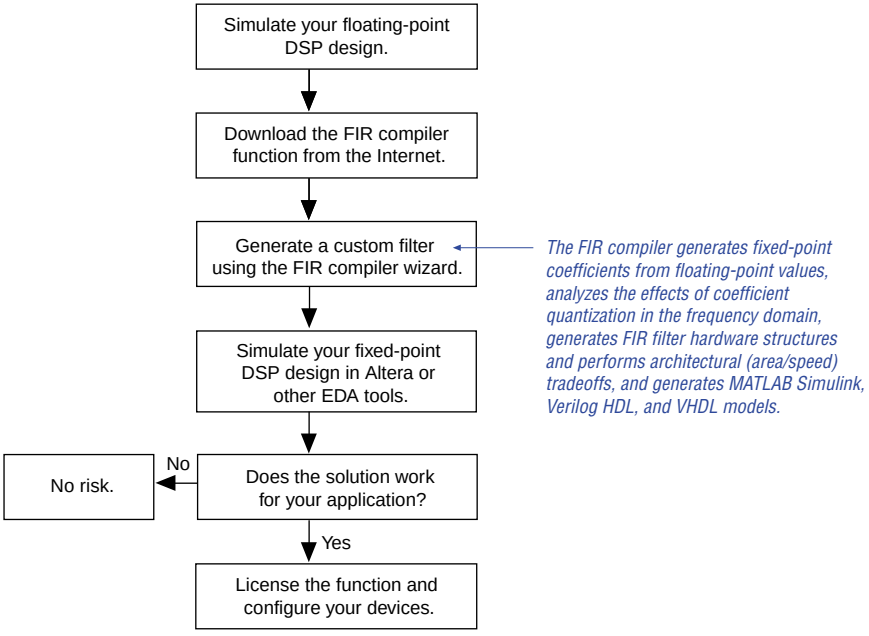
Figure 2. Hypothetical Modulator System



OpenCore Feature

Altera's exclusive OpenCore™ feature allows you to evaluate MegaCore functions before deciding to license them. You can instantiate a MegaCore function in your design, compile and simulate the design, and then verify the MegaCore function's size and performance. This evaluation provides first-hand functional, timing, and other technical data that allows you to make an informed decision on whether to license the MegaCore function. The finite impulse response (FIR) compiler function generates register transfer level (RTL) VHDL and Verilog HDL simulation models, MATLAB and Simulink models, and vector files. Once you license the function, you can use the MAX+PLUS II or Quartus software to generate programming files as well as EDIF, VHDL, or Verilog HDL output netlist files for simulation in third-party EDA tools. Figure 3 shows a typical design flow using MegaCore functions and the OpenCore feature.

Figure 3. OpenCore Design Flow



Altera Devices

The FIR compiler MegaCore function has been optimized and targeted for Altera APEX, FLEX 10K, and FLEX 6000 devices. The FLEX 10K embedded PLD family delivers the flexibility of traditional programmable logic with the efficiency and density of gate arrays with embedded memory. The new 2.5-V FLEX 10KE devices support efficient implementation of dual-port RAM, and further enhance the performance of the FLEX 10K family. Using EABs to implement FIR filters helps reduce the overall resource usage of the filter.

Altera's 5.0-V and 3.3-V FLEX 6000 devices deliver the flexibility and time-to-market of programmable logic at prices that are competitive with gate arrays. Featuring the OptiFLEX™ architecture, FLEX 6000 devices provide a flexible, high-performance, and cost-effective alternative to ASICs for high-volume production.

APEX 20K devices offer complete system-level integration on a single device. The APEX MultiCore™ architecture delivers the ultimate in design flexibility and efficiency for high-performance System-on-a-Programmable Chip™ applications. With densities ranging from 100,000 to 1,000,000 gates, the APEX 20K architecture integrates look-up-table (LUT) logic, product-term logic, and memory into a single architecture, eliminating the need for multiple devices, saving board space, and simplifying the implementation of complex designs.

In the APEX MultiCore architecture, embedded system blocks (ESBs) and logic array blocks (LABs) are combined into MegaLAB™ structures. Each APEX 20K ESB can be configured as product-term logic, enabling APEX 20K devices to achieve unmatched integration efficiency, as LUT logic or as memory. The ESB can be configured as dual-port RAM, with a wide range of RAM widths and depths as ROM in APEX 20K devices, and as content-addressable memory (CAM), a memory technology that accelerates applications requiring fast searches, in APEX 20KE devices.

For more information on APEX 20K, FLEX 10K, and FLEX 6000 devices, refer to the following documents:

- *APEX 20K Programmable Logic Device Family Data Sheet*
- *FLEX 10K Embedded Programmable Logic Family Data Sheet*
- *FLEX 10KE Embedded Programmable Logic Family Data Sheet*
- *FLEX 6000 Programmable Logic Device Family Data Sheet*

Software Tools

Altera offers the fastest, most powerful, and most flexible programmable logic development software in the industry. The MAX+PLUS II and Quartus software offer a completely integrated development flow and an intuitive, graphical user interface, making them easy to learn and use.

The MAX+PLUS II software offers a seamless development flow, allowing you to enter, compile, and simulate your design and program devices using a single, integrated tool, regardless of the Altera device you choose. Altera's fourth-generation Quartus software offers designers the ideal platform for processing multi-million gate designs, with state-of-the-art features that shorten design cycles, streamline the development flow, and reduce verification time. The Quartus software shortens design cycles with the revolutionary nSTEP™ Compiler, which permits incremental recompilation, and multiple processor support that can operate locally or across networks and even platforms.

MegaWizard Plug-Ins

MegaWizard™ Plug-Ins are parameterization tools that help you integrate megafunctions into your designs without requiring the use of third-party tools. You can use this feature in the MAX+PLUS II and Quartus software or as a stand-alone tool with third-party EDA design interfaces. MegaWizard Plug-Ins provide maximum flexibility, allowing you to customize megafunctions without changing your design's source code. You can integrate a parameterized megafunction in any hardware description language (HDL) or netlist file using any EDA tool.

The MAX+PLUS II MegaWizard Plug-In Manager allows you to bring up the megafunction's wizard so that you can set the parameters of the megafunction to fit your design. The wizard then generates a customized instance of the megafunction, which you can instantiate in your design file.

EDA Interfaces

Altera has worked closely with major EDA vendors to develop the best interface any PLD software has to offer.

As a standard feature, the MAX+PLUS II software interfaces with all major EDA design tools, including tools for ASIC designers. Once a design is captured and simulated using the tool of your choice, you can transfer your EDIF file directly into the MAX+PLUS II software. After synthesis and fitting, you can transfer your file back into your tool of choice for simulation. The MAX+PLUS II system outputs the full-timing VHDL, Verilog HDL, Standard Delay Format (SDF), and EDIF netlists that can be used for post-route device- and system-level simulation. Altera opened the Quartus interface to various EDA partners to enable them to provide unmatched levels of integration. NativeLink integration provides a truly seamless interface to major EDA software tools to support existing design flows, eliminating the need to learn new design tools.

To support Altera's DSP functions, the DSP MegaWizard Plug-Ins work together with the MATLAB and Simulink software. The wizard imports MATLAB coefficients and outputs MATLAB/Simulink-compatible models for system verification.

To simplify the design flow between the Altera software and other EDA tools, Altera has developed the MAX+PLUS II Altera Commitment to Cooperative Engineering Solutions (ACCESS^{SM}) Key Guidelines and the Quartus NativeLink Guidelines. These guidelines provide complete instructions on how to create, compile, and simulate your design with tools from leading EDA vendors. The guidelines are part of Altera's ongoing efforts to give you state-of-the-art tools that fit into your design flow, and to enhance your productivity for even the highest-density devices. The MAX+PLUS II ACCESS Key Guidelines are available on the Altera web site (**http://www.altera.com**) and the MAX+PLUS II CD-ROM. The NativeLink guidelines are integrated into Quartus Help.

Specifications

September 1999, ver. 1.10 User Guide

Features

- First system-level, programmable logic solution for digital signal processing (DSP) designs, including:
  - Automatic interpolation filters
  - Automatic decimation filters
- Fully integrated finite impulse response (FIR) filter development environment
- Highly optimized for Altera® device architectures
- Fully parallel or serial arithmetic architectures
- Supports any number of taps
- Includes a built-in coefficient generator
- Imports floating-point or integer coefficients from third-party tools
- Provides multiple coefficient scaling algorithms
- Provides floating-point to fixed-point coefficient analysis
- Supports coefficient widths from 4 to 32 bits of precision
- Supports signed or unsigned input data widths, from 4 to 32 bits wide
- User-selectable output precision via parameterizable rounding and saturation
- Generates MATLAB Simulink, VHDL, and Verilog HDL simulation models
- Generates MAX+PLUS® II and Quartus™ vector files
- Includes an impulse, step function, and random input testbed
- Provides resource estimates dynamically

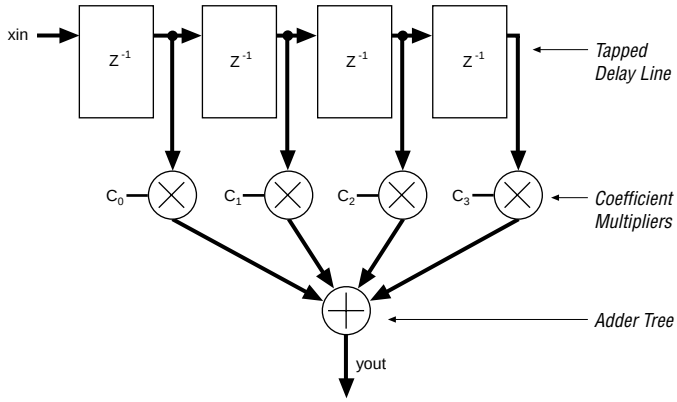
General Description

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 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 that they are 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



DSP processors have a limited number of multiplier-accumulators (MACs), which 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 data rates above 100 million samples per second (MSPS), making PLDs ideal for high-speed filtering applications.

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

Table 1. FIR Filter Implementation Comparison

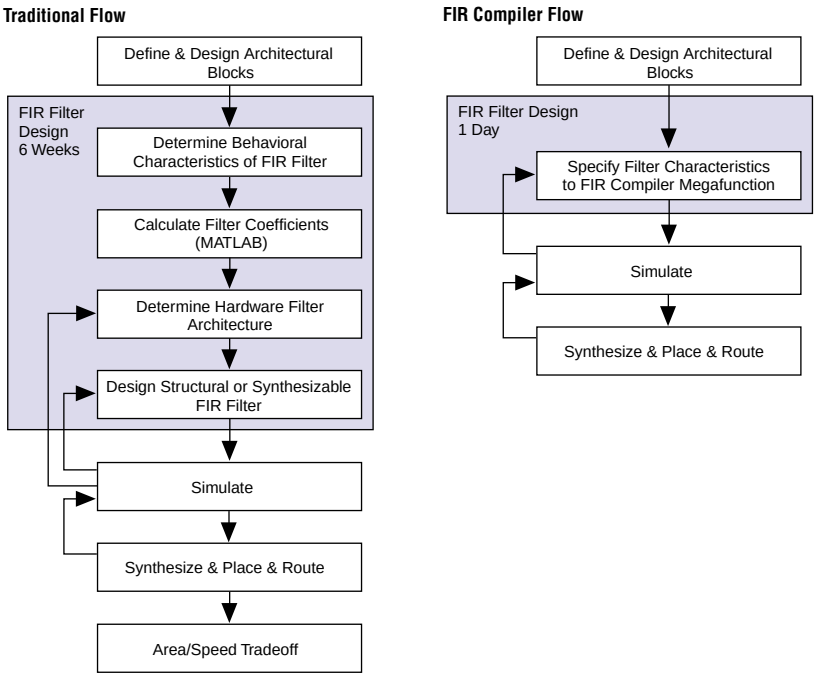
Device	Implementation	Clock Cycles to Compute Result
DSP processor	1 MACs	120
PLD	1 serial filter	12
	1 parallel filter	1

The FIR compiler speeds up the design cycle by:

- Finding the coefficients needed to design custom FIR filters.
- Generating 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 MAX+PLUS II or Quartus software to synthesize high-speed, area-efficient FIR filters of various architectures.
- Creating standard test vectors (i.e., impulse, step, and random input) to test the FIR filter's response.

Figure 2 compares the design cycle using the FIR compiler MegaCore function versus a traditional implementation.

Figure 2. Design Cycle Comparison



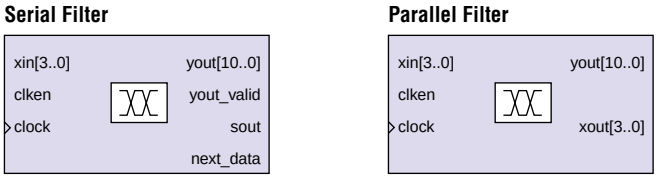
Signals

The FIR compiler can generate two 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.

Figure 3 shows the symbols for serial and parallel FIR compiler functions.

Figure 3. Serial & Parallel Symbols



The FIR compiler function has the signals shown in Table 2.

Table 2. FIR Compiler Signals

Signal	Structure	Type	Description
`xin[width_xin-1..0]`	Parallel and serial	Input	Input data to be filtered.
`yout[width_yout-1..0]`	Parallel and serial	Output	Result of filtering operation performed in `xin`.
`clock`	Parallel and serial	Input	Input clock signal.
`clken`	Parallel and serial	Input	Active-high clock enable.
`yout_valid`	Serial	Output	Goes high when the output result is valid.
`next_data`	Serial	Output	Goes high when it is ready to load the next data word.
`xout[width_xin-1..0]`	Parallel	Output	Output data from the tapped delay line.
`sin[x..0]`	Serial	Input	Input data to be filtered. The data is shifted in serially one bit at a time (LSB first).
`sout`	Serial	Output	Output from the tapped delay line (LSB first).
`phase_o[]`	Parallel and serial	Output	Phase outbut buses for the polyphase filter with an interpolation or decimation factor greater than 1.

To clock in data, the `clken` signal must be high for at least one clock cycle. Serial filters require at least *N* clock cycles to complete the calculation before the `clken` signal goes high again. When the `next_data` signal goes high, the next data word can be clocked. Figure 4 shows the input requirements for a parallel filter with no pipelining. Parallel filters generate an output every clock cycle.

Figure 4. Parallel Filter (No Pipelining)

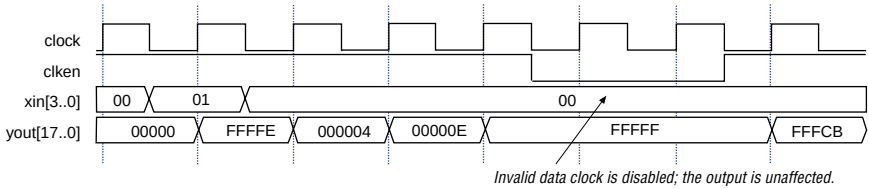
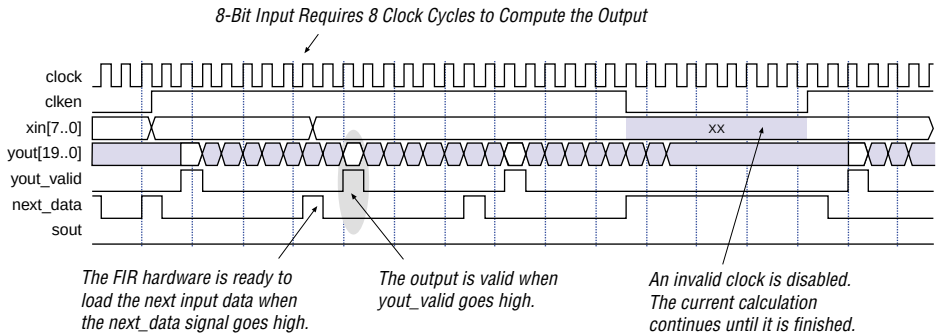


Figure 5 shows the input waveform for an 8-bit serial filter.

Figure 5. 8-Bit Serial Filter



MegaWizard Plug-In

You can launch the MegaWizard™ Plug-In Manager from within the Quartus or MAX+PLUS 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 3 highlights the main features of the wizard.

Table 3. FIR Compiler Wizard Options

Option	Description
Architecture	You can indicate whether the filter is parallel or serial, and the number of channels for the filter. For serial filters, you can use either EABs or logic cells to implement the filter. For both parallel and serial filters, you can specify whether or not to use pipelining.
Coefficients	The FIR compiler can read filter coefficients that have been exported from a third-party system-level application or 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 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.
Input Data Specification	You can specify the width of the input data bus (from 4 to 32 bits wide) and whether the bus is signed or unsigned.
Limiting Precision	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 LSB via truncation or rounding.
Multi-Rate Filters	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.
Simulation Output Files	The FIR compiler generates several types of simulation files, including MAX+PLUS 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:

- Text Design File (**.tdf**) used to instantiate an instance of the FIR filter in your design
- MAX+PLUS II Vector File (**.vec**) used for simulation within the MAX+PLUS II environment
- Symbol File (**.sym**) used to instantiate the filter into a schematic design
- ASCII text file **coef.txt** that contains the rounded and scaled coefficients
- MATLAB and Simulink files used for simulation in MATLAB Simulink
- VHDL and Verilog HDL models used for simulation in other EDA tools

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 any number of taps.

Number Systems & Fixed-Point Precision

The FIR compiler function supports signed or unsigned fixed-point numbers from 4 to 32 bits wide using two's complement numbers. The entire filter operates in a single number system. The coefficient precision is independent of input data width; the user 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 6 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 output.

Figure 6. 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. The FIR compiler saves the coefficients into the **coef_float.txt** file.

When the FIR compiler reads in the coefficients, it automatically determines any symmetry and selects the appropriate architecture. 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. After scaling, the coefficients are output to the **coef.txt** file.

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.

- *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 filter 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 7 and 8.

Figure 7. 7-Tap Symmetrical FIR Filter

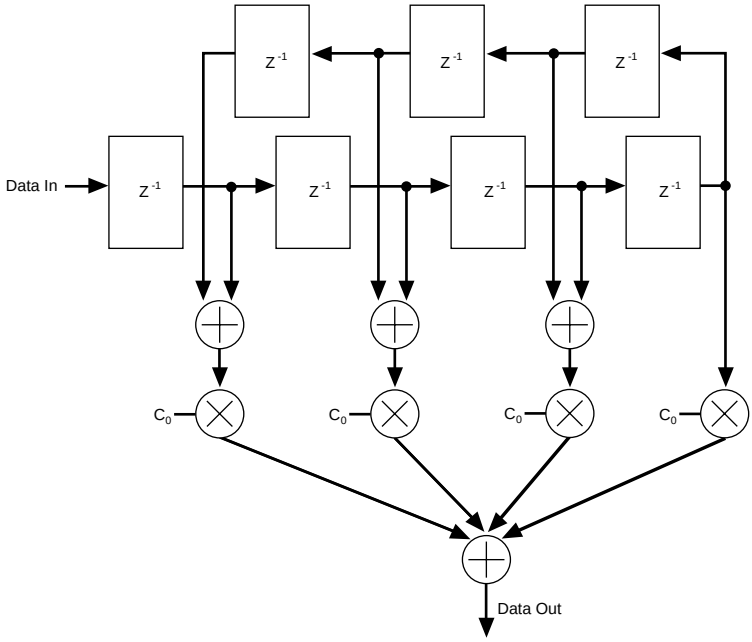
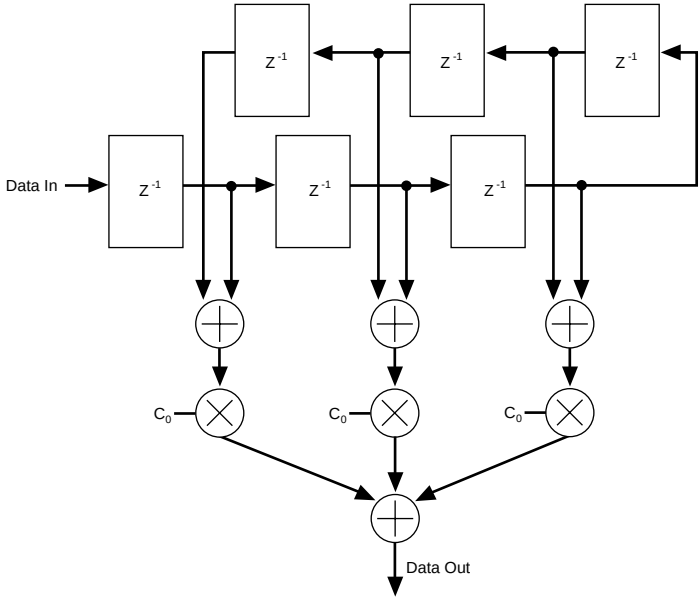


Figure 8. 6-Tap Symmetrical FIR Filter

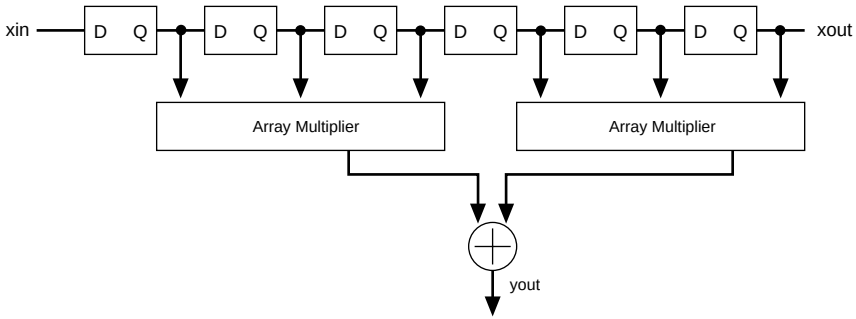


The FIR compiler wizard generates parallel, serial, multi-channel, and single and multiple MAC arithmetic structures.

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 70 and 120 MHz at the cost of pipeline latency. Refer to Figure 4 on page 11 for a waveform of the parallel structure. Figure 9 shows the parallel filter block diagram.

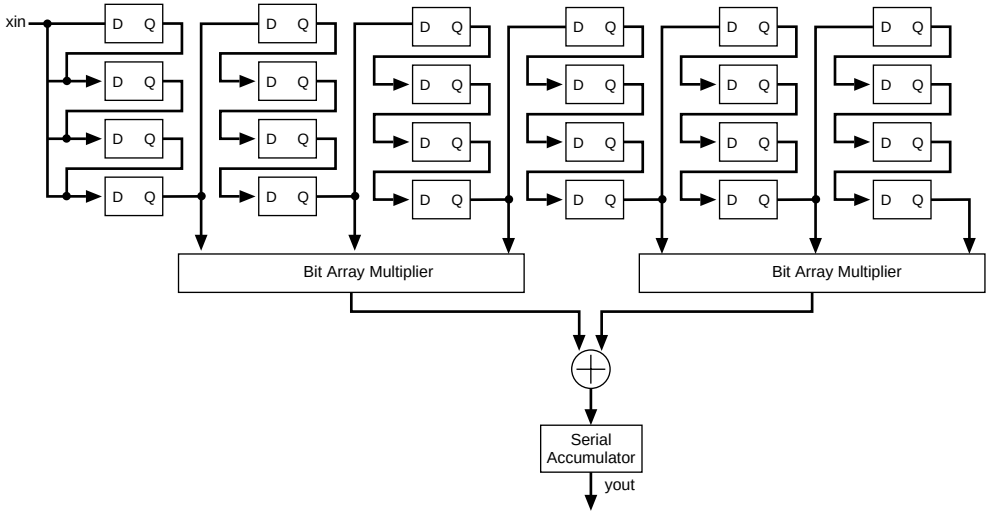
Figure 9. Parallel Filter Block Diagram



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. You can use FLEX 10K EABs or APEX 20K ESBs to store data for the tapped delay line (shift register). Refer to Figure 5 on page 11 for a waveform of the serial structure. Figure 10 shows the serial filter block diagram.

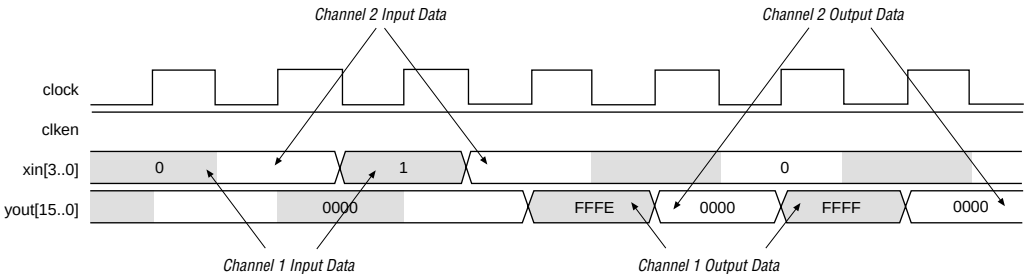
Figure 10. Serial Filter Block Diagram



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. Figure 11 shows the inputs and outputs of a multi-channel parallel filter.

Figure 11. Parallel Multi-Channel Filter



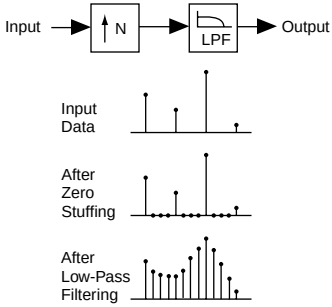
For maximum area efficiency, use a distributed serial arithmetic architecture, multiple channels, and dual-port RAM in FLEX 10KE EABs or APEX ESBs.

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.

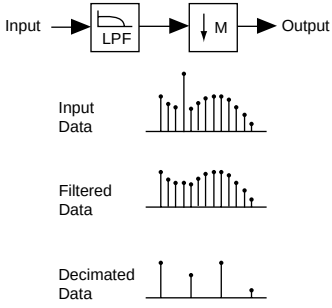
When a signal is interpolated, zeros are inserted between data points and the data is filtered to remove spectral components that were not present in the original signal. See Figure 12.

Figure 12. 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 13.

Figure 13. Signal Decimation

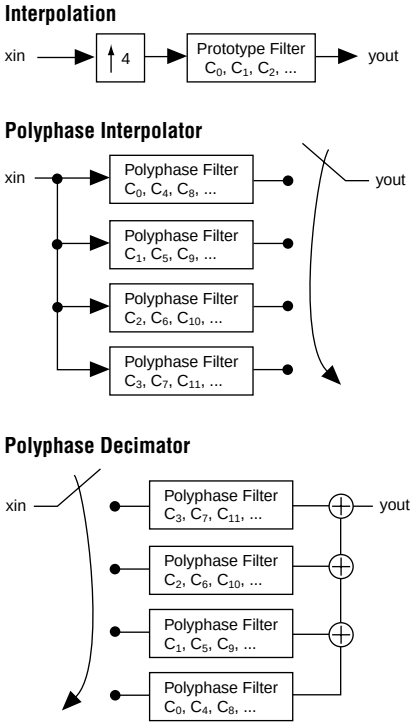


The FIR compiler automatically creates interpolation and decimation filters using a polyphase decomposition. Polyphase decimation filters provide speed optimization because each filter runs at the output data rate. Polyphase interpolation filters provide the following benefits:

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

Figure 14 shows block diagrams for polyphase interpolation and decimation.

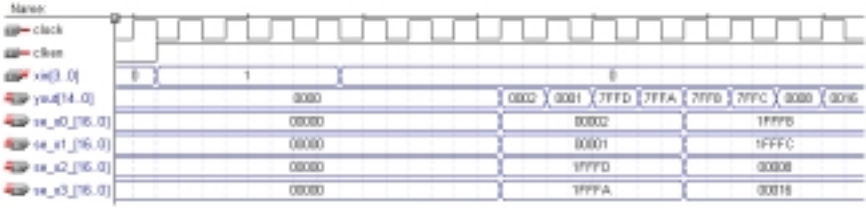
Figure 14. Polyphase Interpolation & Decimation Block Diagrams



Serial Interpolation Waveforms

Figure 15 shows the waveform 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.

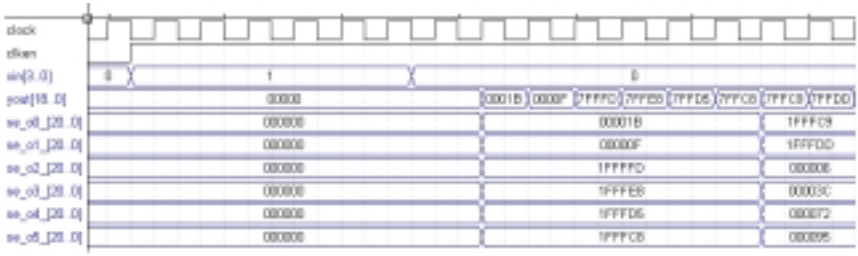
Figure 15. Interpolation Factor = Input Data Width



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 `yout` (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 16 shows the waveform for a filter in which the interpolation factor (six) is greater than the input data width (four). The filter has six polyphase outputs.

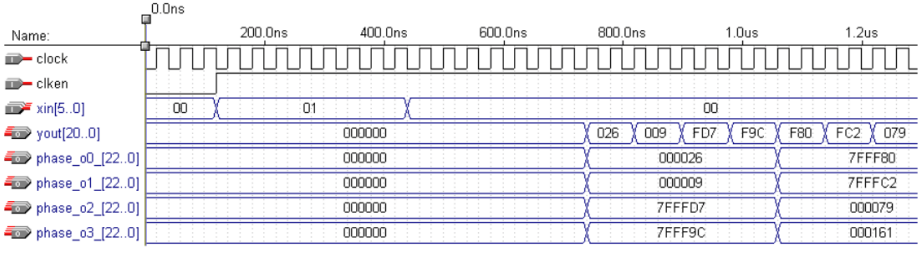
Figure 16. Interpolation Factor > Input Data Width



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

Figure 17. Interpolation Factor < Input Data Width

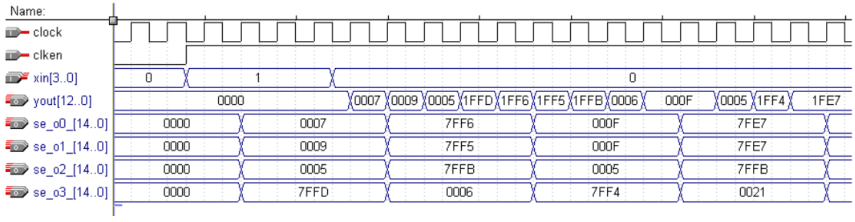


For this filter, a 4× clock does not provide enough cycles to calculate an individual polyphase output. To ensure a constant output data rate, the FIR compiler uses an 8× clock (or a clock rate of two times the interpolation factor), switching between every polyphase output every two clocks. The 8× clock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

Parallel Interpolation Waveforms

Figure 18 shows the waveform 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.

Figure 18. Parallel Interpolation Waveform

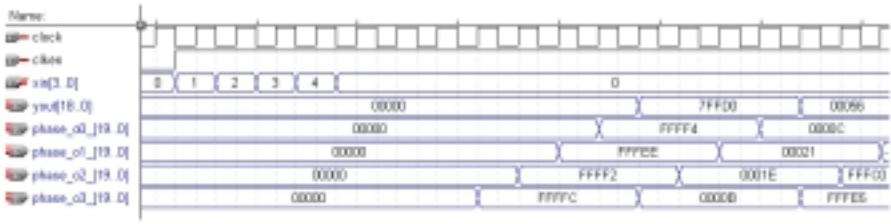


The parallel case, which is the simplest for timing, illustrates the benefit of a polyphase decomposition. This technique relaxes the timing requirements on the FIR filter that is generated. If the input data rate is 50 MHz, and the interpolation factor is four, the polyphase filters must run at the 50 MHz data rate. The multiplexer which switches through all the filters need to run at 200 MHz (Altera FLEX and APEX devices support multiplexers that run at these speeds). 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.

Serial Decimation Waveforms

Figure 19 shows a serial decimation filter in which the decimation factor (four) equals the input bit width (four).

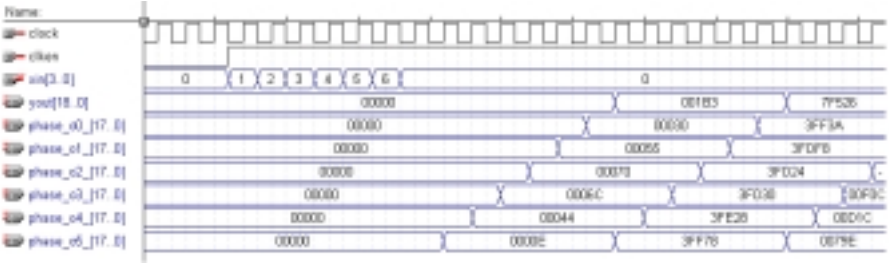
Figure 19. Decimation Factor = Input Bit Width



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. In addition, outputs from each of the four individual filters are available separately.

Figure 20 shows a serial decimation filter in which the decimation factor (six) is greater than the input data width (four).

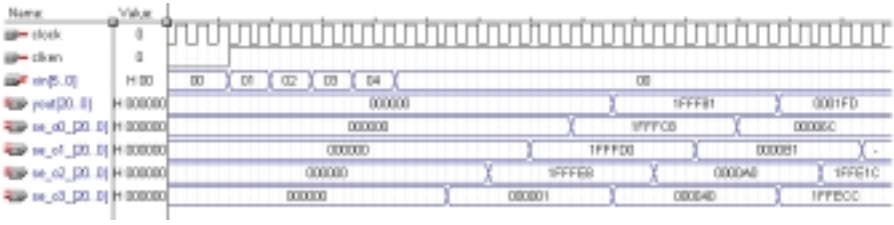
Figure 20. Decimation Factor > Input Data Width



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

Figure 21. Decimation Factor < Input Data Width

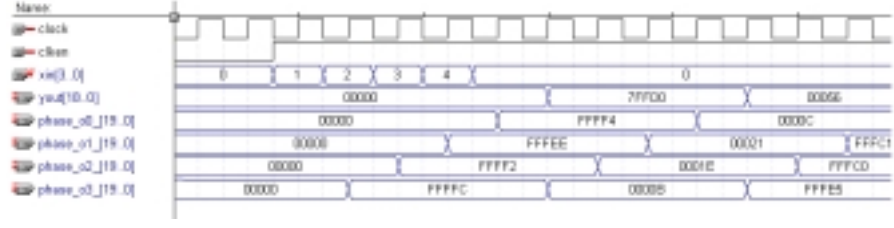


The FIR compiler generates four polyphase filters. Each filter requires at least 6 clock cycles to generate an output because they are serial filters with input data widths of six bits. Therefore, a single 4× clock is not sufficient to create the structure. By providing twice the clock rate (8×) there 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 are available for use.

Parallel Decimation Waveform

Figure 22 shows a parallel decimation waveform with a decimation factor of four. The polyphase technique for decimation generates four filters, each of which operate at the output rate. At every clock cycle, the input data goes to the next polyphase. After four clock cycles, the outputs from the each polyphase are added together.

Figure 22. Parallel Decimation Filter



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. Altera FLEX and APEX devices can easily generate multiplexers that operate at these speeds. The total system throughput is 200 MSPS (generated from 4 filters operating in parallel at a 50 MSPS rate).

Pipelining

Pipelining is most effective for producing high-performance filters at the cost of increased latency. The FIR compiler provides the following pipelining options:

- *Automatic*—The wizard adds pipeline stages for maximum throughput.
- *None*—The wizard does not pipeline the design.

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 Verilog HDL simulation tools.

The FIR compiler automatically generates test vectors for the following types of stimulus:

- Impulse
- Step function
- Random input

You can use these test vectors and MATLAB software to simulate your design. When you compile your FIR filter design, the FIR compiler wizard generates MATLAB Simulink-compatible models for system verification.

Performance

Table 4 shows the FIR compiler function's performance using the MAX+PLUS II version 9.2 software.

Table 4. FIR Compiler Performance

Device	Parameters	LEs Used	EABs Used	f_{MAX} (MHz)
FLEX 10KE-1	17-tap, fully parallel	879	0	82
	19-tap, fully parallel	1,260	0	101
	79-tap, serial	761	5	69

Notes:

September 1999, ver. 1.10 User Guide

Altera digital signal processing (DSP) MegaCore™ functions provide solutions for integrating finite impulse response (FIR) filters into your digital communications system. The functions are optimized for Altera® APEX™ 20K, FLEX® 10K, and FLEX 6000 devices, greatly enhancing your productivity by allowing you to focus efforts on the custom logic in the system. This section describes how to obtain the FIR compiler MegaCore function, explains how to install it on your PC, and walks you through the process of implementing the function in a design. You can test-drive MegaCore functions using Altera's OpenCore™ feature to simulate the functions within your custom logic. When you are ready to license a function, contact your local Altera sales representative.

Design Walkthrough

This design walkthrough involves the following steps:

1. Build your system using MATLAB Simulink.
2. Download and install the FIR compiler function.
3. Generate the filter(s) for your system using the FIR compiler wizard.
4. Drag and drop the FIR compiler wizard-generated Simulink model files your system model.
5. Implement the rest of your system using the Altera Hardware Description Language (AHDL), VHDL, Verilog HDL, or schematic entry.
6. Use the FIR compiler wizard-generated VHDL or Verilog HDL simulation models to confirm your system's operation.
7. Compile your design and perform place-and-route.
8. Perform system verification.
9. License the FIR compiler function to configure or program the devices.

The instructions assume that:

- You are using a PC running Windows NT 3.51 or 4.0, or Windows 95/98.
- You are familiar with the MATLAB and MAX+PLUS II software.
- MAX+PLUS II version 9.2 or higher is installed in the default location (**c:\maxplus2**).
- You are using the OpenCore feature to test-drive the FIR compiler function or you have licensed the function.

Download & Install the FIR Compiler Function

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. Run your web browser (e.g., Netscape Navigator or Microsoft Internet Explorer).
2. Open the URL **http://www.altera.com**.
3. Click the Tools icon on the home page toolbar.
4. Click the MegaCore Functions link.
5. Click the link for the FIR compiler function.
6. Follow the on-line instructions to download the function and save it to your hard disk.

Installing the FIR Compiler Files

For Windows 95/98 and Windows NT 4.0, follow the instructions below:

1. Click **Run** (Start menu).
2. Type <*path name*>\<*filename*>`.exe`, where <*path name*> is the location of the downloaded MegaCore function and <*filename*> is the filename of the function.
3. Click **OK**. The **MegaCore Installer** dialog box appears. Follow the on-line instructions to finish installation.
4. After you have finished installing the MegaCore files, you must specify the directory in which you installed them (e.g., <*path*>/**fir_compiler/lib**) as a user library in the MAX+PLUS II software. Search for "User Libraries" in MAX+PLUS II Help for instructions on how to add these libraries.

Generate a FIR Filter

This section describes the design flow using the Altera FIR compiler MegaCore function and the MAX+PLUS II development system. The MegaWizard Plug-In Manager, which you can use within the MAX+PLUS II software or as a stand-alone application, lets you create or modify design files that contain megafunction variations. You can then instantiate the megafunction variation in your design file. The FIR compiler function is parameterized, and you can customize it with the MegaWizard Plug-In to meet the needs of your application.

You can use Altera's OpenCore feature to compile and simulate the MegaCore functions, allowing you to evaluate the functions before deciding to license them. 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.

You should use the MAX+PLUS II software version 9.2 or higher when designing with the FIR compiler function. The function relies on library files that only exist in these versions of software.

To create a FIR function using the wizard, follow these steps:

1. Start the MegaWizard Plug-in Manager by choosing the **MegaWizard Plug-In Manager** command (File menu) in any MAX+PLUS II application, or by starting the stand-alone version of the MegaWizard Plug-In Manager by typing the command `megawiz` ↵ at a command prompt. The **MegaWizard Plug-In Manager** dialog box is displayed.

Refer to MAX+PLUS II Help for detailed instructions on how to use the MegaWizard Plug-In Manager.

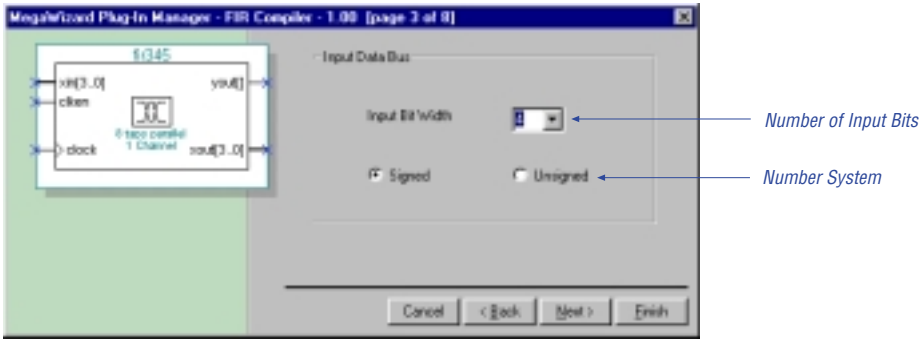
2. Specify that you want to create a new custom megafunction variation and click **Next**.

3. On the second page of the wizard, select the FIR compiler function from the *DSP MegaCore* drop-down list in the *Available Megafunctions* box, choose that you wish to create an AHDL TDF, specify the directory and name for the files the wizard creates, and click **Next**.

   If you do not specify the directory in which you installed the MegaCore files as a user library in the MAX+PLUS II software, the function will not appear in the *Available Megafunctions* box. Search for "User Libraries" in MAX+PLUS II Help for information on how to specify these libraries.

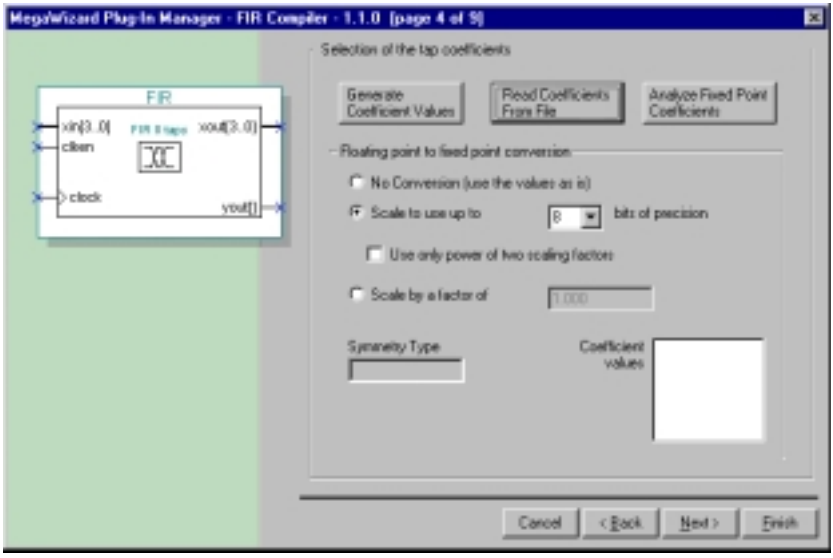
4. Specify the bit width of the filter input data bus. You can also specify whether the bus is signed or unsigned; the signed representation uses the two's complement numbering system. See Figure 1. Click **Next**.

Figure 1. Enter Data Bus Parameters



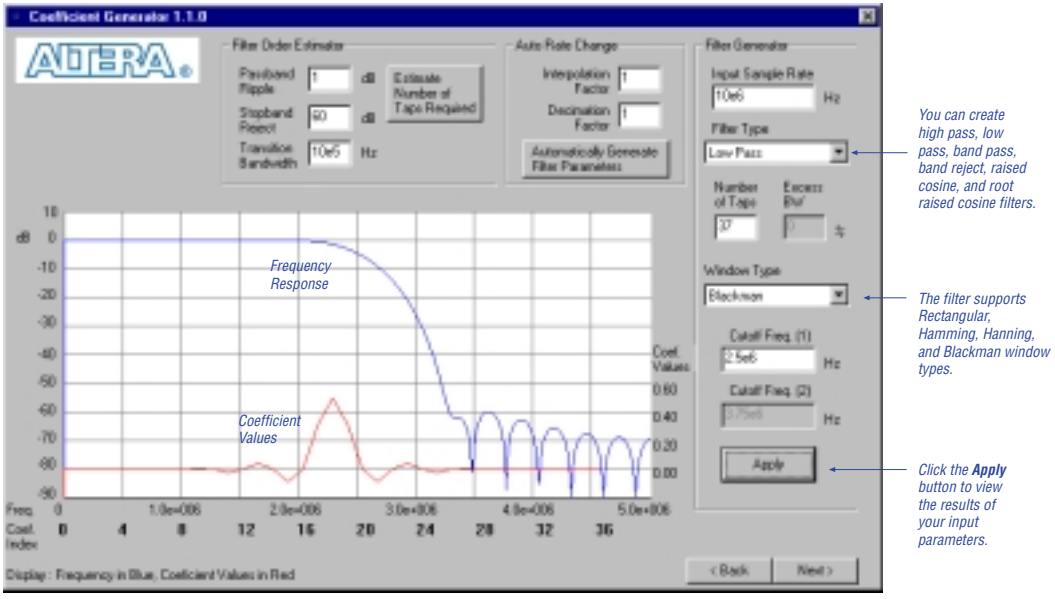
5. Specify coefficients for your filter. The wizard can generate coefficients or you can read the coefficients from a file. The wizard can read fixed-point or floating-point coefficients from a simple ASCII text file. Refer to Figure 6 on page 13 for a sample filter coefficient file. To use the wizard to generate coefficients, click the **Generate Coefficient Values** button on the fourth page of the wizard. See Figure 2.

Figure 2. Specify Coefficients



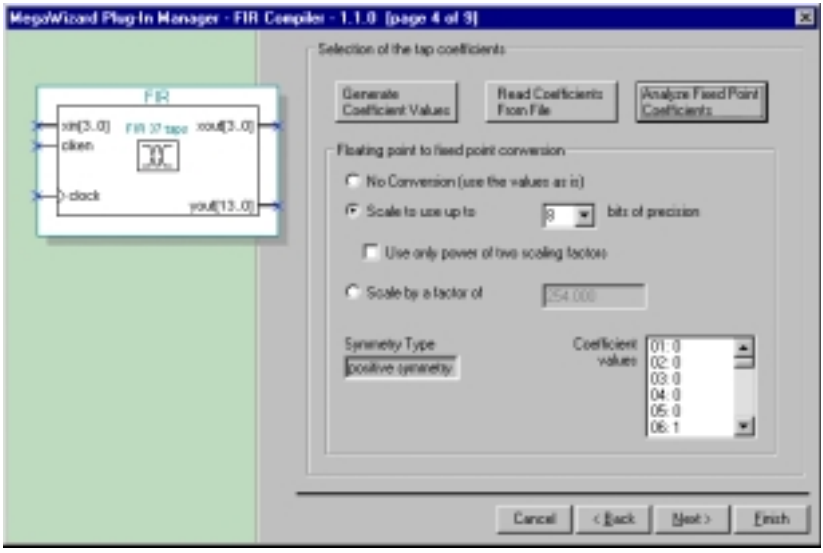
6. The wizard coefficient generator allows you to specify the sample rate, the number of taps, the cut-off frequencies, the filter type, and the window method. Specify the characteristics for your filter and click the **Apply** button to view the frequency response and the coefficient values. The wizard automatically generates symmetrical filters as needed. By default, the sample rate is in Hertz. To enter the filter parameters in relation to the Nyquist rate, enter a 1 for the sample rate. (The maximum cut-off frequency changes to 0.5.) You can also enter decimation and interpolation factors, or you can generate decimation or interpolation factors automatically. Figure 3 shows the coefficient generator. When you are satisfied with the settings, click **Next**. The wizard loads the coefficients automatically.

Figure 3. Generating Coefficients with the Wizard



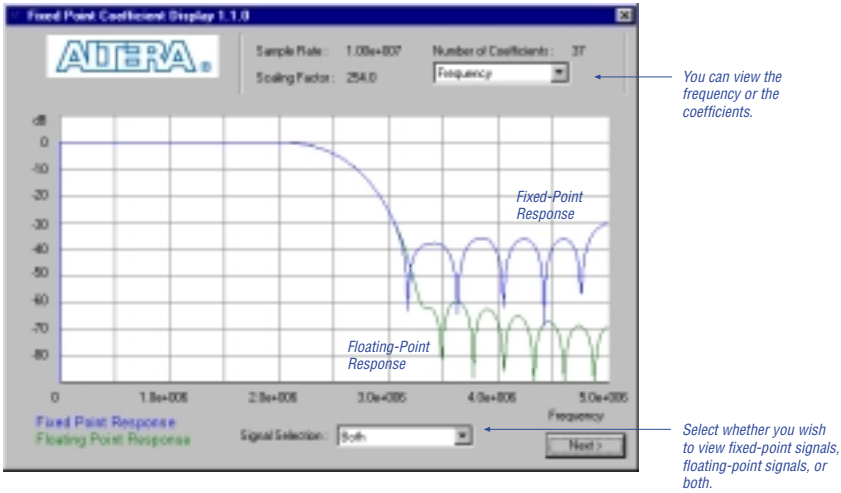
7. After you have specified the coefficients, either by using the wizard or reading them from a file, you can scale the coefficients or use them as is. However, if you specified the coefficients as floating point numbers, you must scale them. The coefficients are displayed in the *Scaled and Rounded Coefficients* box. The wizard detects any symmetry present, automatically selects an architecture to create a smaller filter, and displays the resulting symmetry in the *Symmetry Type* box (see Figure 4). Click **Next** to continue.

Figure 4. Scaled & Rounded Coefficients



8. After scaling and rounding the coefficients, you can use the wizard to view the resulting fixed-point coefficients and compare them to the floating-point values. The wizard displays both the frequency and coefficient values. The fixed-point coefficient analyzer lets you quickly determine the number of bits of preceision required to obtain a desired spectral response. See Figure 5.

Figure 5. Fixed-Point Coefficient Analyzer



9. You can specify whether to use full or limited precision for the filtered output (`yout`). The FIR compiler determines the output bit width 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.

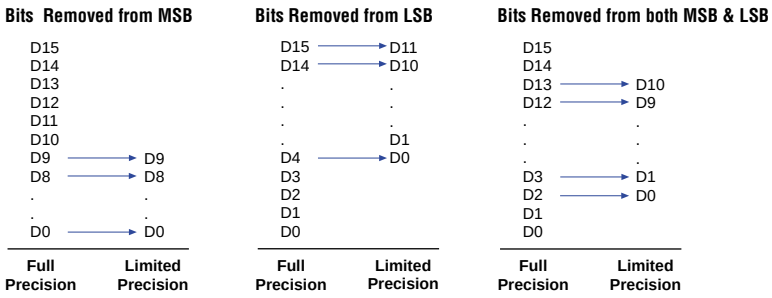
   If you choose limited precision, the wizard gives you the option of truncating or saturating the MSB and/or rounding or truncating the LSB. Saturation and rounding are non-linear operations. Table 1 shows the options for limiting the precision of your filter.

Table 1. Options for Limiting Precision

Bit Range	Option	Result
MSB	Truncate	In truncation, the filter disregards specified bits. See Figure 6.
	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.

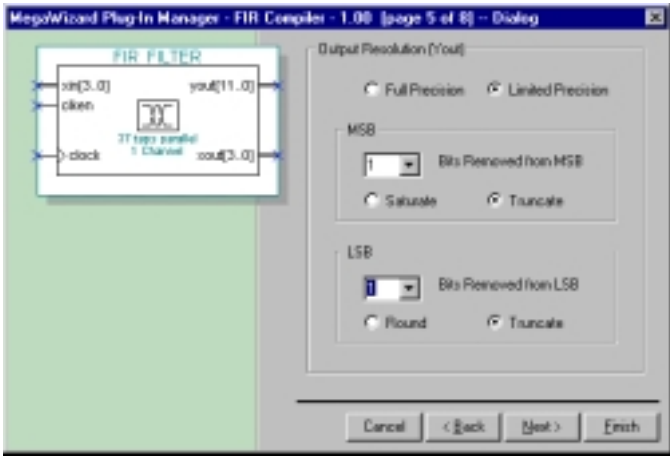
Figure 6 shows an example of removing bits from the MSB and LSB.

Figure 6. Removing Bits from the MSB & LSB



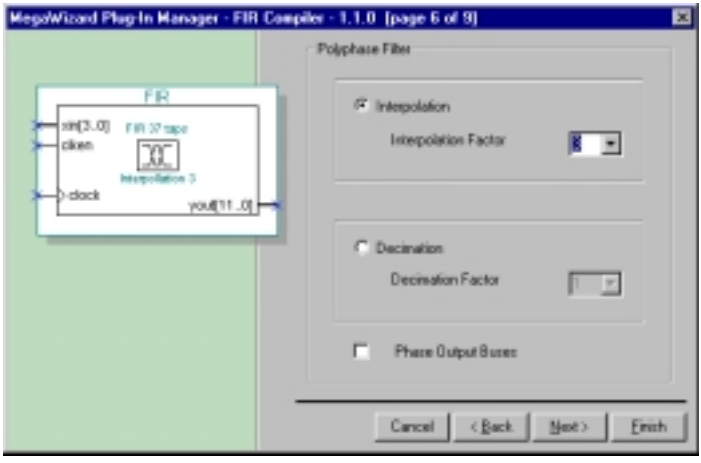
10. Make your selections and choose **Next**. See Figure 7.

Figure 7. Specify the Filter Precision



11. You can use the wizard for decimation and interpolation, which change the effective rate of the filter. You can choose whether to use interpolation or decimation, and indicate a factor for each. See Figure 8. Click **Next** after making your selections.

Figure 8. Decimation & Interpolation



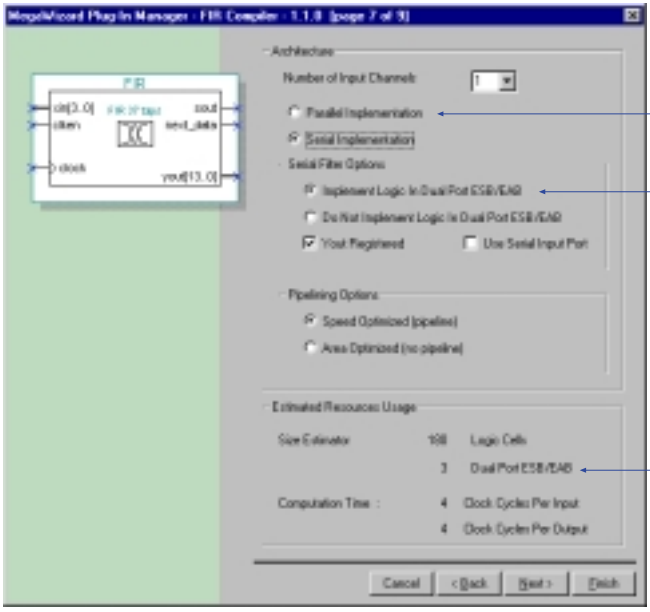
12. Indicate the architecture of the filter, for example, whether it is parallel or serial, any pipelining desired, and the number of input channels. The default value is parallel. Table 2 describes some of the trade-offs for the different architecture options.

Table 2. Architecture Trade-Offs

Option	Area	Speed (Data Throughput)
Parallel	Uses a large area.	Creates a fast filter: 60 to 140 MSPS throughput with pipelining.
Serial	Uses a smaller area.	Requires multiple clock cycles for a single computation.
Pipelining	Creates a high-performance filter with only a small area increase.	Increases throughput by two to four times with additional latency.

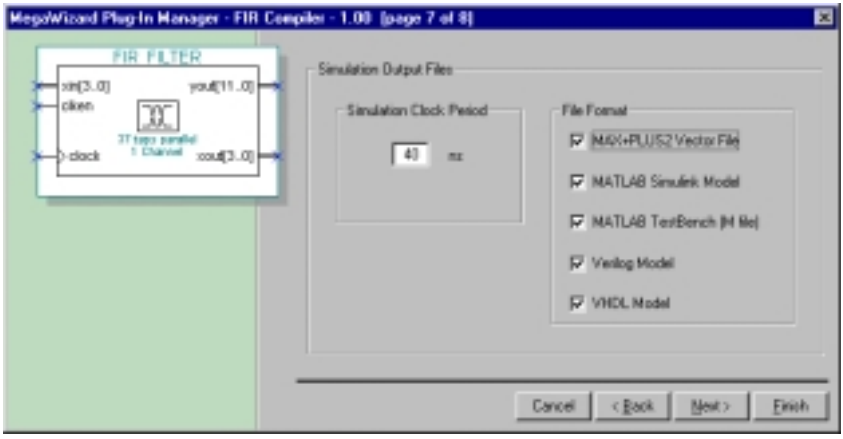
The wizard automatically calculates the resources the filter will use in the *Resource Usage* box. It provides the estimated size in EABs and/or logic cells and the number of clock cycles required to perform the FIR computation. The latency, i.e., the number of clock cycles before the output is available, is shown in the MAX+PLUS II Report File (**.rpt**) after design compilation. See Figure 9. Click **Next** when you are finished.

Figure 9. Specify the Filter Architecture



13. Specify the file formats and clock period for the simulation output files you wish to generate and click **Next**. The wizard generates all file formats by default. See Figure 10.

Figure 10. Choose the Output File Types



14. Click **Finish** to accept your selections and generate the new megafunction variation.

Once you have created a megafunction variation, you can integrate it into your custom design. After you have finished your design, you are ready to instantiate it in your system design and compile it.

For the best results, you should use the *Fast* logic synthesis style when compiling your design in the MAX+PLUS II software.

Model Your System Using MATLAB Simulink

The MATLAB software performs complex mathematical computations in an interactive environment. Many DSP designers use the MATLAB environment to generate FIR filter coefficients and to simulate their DSP systems. You can also use the software to evaluate the effects of channels and to analyze signals. The Simulink software is a MATLAB add-on product and provides a graphical interface that allows you to model, analyze, and simulate your system quickly.

To begin, perform the steps below.

1. Run the MATLAB software.
2. In the **MATLAB Command Window**, change to the working directory for your project.
3. Run Simulink by typing `Simulink` ↵ at the prompt. The **Simulink Library Browser** opens.

4. Create a new model by clicking the page icon in the **Simulink Library Browser** or by choosing **New** (File menu) in the **MATLAB Command Window**.

5. Drag and drop blocks into your new model according to your system requirements.

6. Simulate your model.

Drag & Drop the FIR Compiler Models into Your Simulink Model

System verification is an important part of filter design. The FIR compiler generates output files that work seamlessly with third-party system verification tools, including MATLAB/Simulink Model Files and S functions. After you create your FIR filter in the MAX+PLUS II software, you can use the models to test the function in-system.

You can use the FIR compiler megafunction together with the MATLAB and Simulink software to design, implement, and simulate your system. The megafunction imports coefficients generated using MATLAB and outputs MATLAB/Simulink-compatible models for system verification.

For more information on MATLAB and Simulink, refer to the Math Works web site at **http://www.mathworks.com**.

Altera provides sample Simulink models with the FIR compiler function that you can use to preview the operation of the function in-system. Additionally, the FIR compiler wizard generates output files for use with the MATLAB and Simulink software. See Table 3 for a description of the files provided with and generated by the FIR compiler.

Table 3. Wizard-Generated MATLAB/Simulink Files

File	Description
<*function name*>_**simulink.m**	An ASCII text file called an S-function. This file is a text-based description, or model, of the filter's functionality. The FIR compiler automatically generates S-functions that are clock-cycle accurate representations of the filter operation. You should not modify these files.
<*function name*>_**simulink.mdl**	Simulink Model File (**.mdl**) that provides a functional model of a system. Model Files include scope models that represent stimulus occuring in the system. You should create your own model to simulate a system using your custom version of the FIR compiler. The Simulink Model File incorporates S-functions in **.m** format. You should save both the **.mdl** and **.m** files together in your Simulink working directory. Altera provides a sample Simulink Model File with the FIR compiler function, which shows an example of how the FIR compiler operates in-system.
<*function name*>_**qplot.m**	A file that graphically plots the FIR compiler-generated scaled and rounded coefficients. You can use the plot to view a time display (in clock cycles), a frequency display, and the effects of random noise.

To model the wizard-generated files in MATLAB/Simulink, perform the steps below.

1. Copy the Simulink files (with the extension(s) **.mdl** and/or **.m**) into your MATLAB working directory.
2. Run the MATLAB software.
3. In the **MATLAB Command Window**, change to the directory in which you saved the wizard-generated files.
4. Open your system model.
5. Open the Model File for the filter (i.e., <*filename*>**_simulink.mdl**) and drag and drop the block into your system model.

6. Adjust your model as needed to work with the FIR compiler model. Altera provides the elements described in Table 4 with the FIR compiler that you can drop into your design.

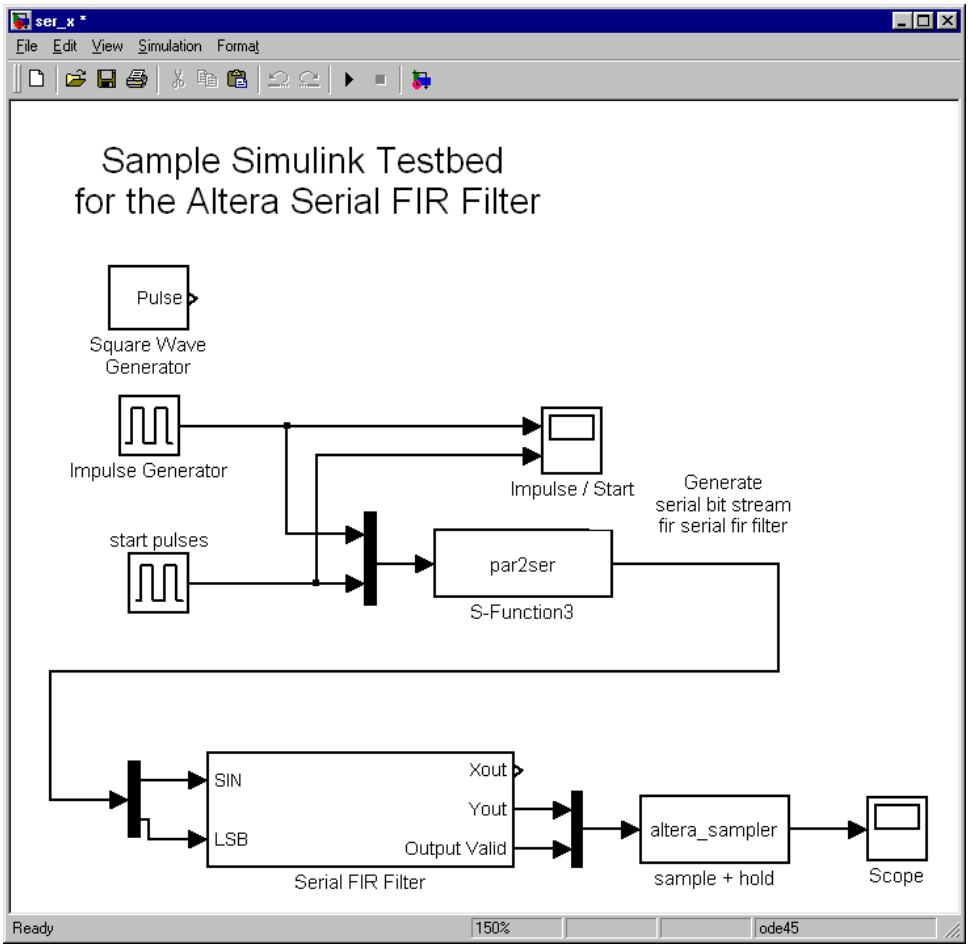
Table 4. Simulink Files Provided with the FIR Compiler

Element	Filename	Description
Serial to parallel converter	**ser2par.m**	Converts a serial data stream to a parallel stream.
Parallel to serial converter	**par2ser.m**	Converts a parallel data stream to a serial one.
Altera sampler	**altera_sampler.m**	This function holds the output of the filter for viewing. Output data is always valid.

7. After you finish building your model, simulate it to see the effects of the filter in-system.

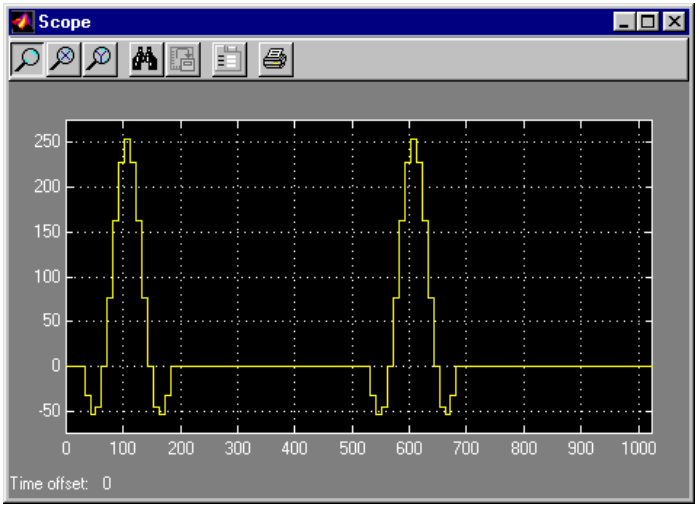
Altera provides a sample Simulink model with the FIR compiler. It shows a system with pulse generators, a parallel to serial converter, a serial FIR filter, and the Altera sampler connected to a scope. See Figure 11.

Figure 11. Simulink Testbed



During simulation, the scope outputs the plot shown in Figure 12.

Figure 12. Scope Plot of Testbed Results



Implement the System

When you are finished simulating your system in Simulink, you are ready to implement it. You can use the design files generated by the FIR compiler wizard in your design. The wizard outputs an AHDL Text Design File (**.tdf**) and a Symbol File (**.sym**). You can use the MAX+PLUS II software, Quartus software, or other EDA tools to create your design.

Simulate Using VHDL & Verilog HDL Models

The FIR compiler wizard generates VHDL and Verilog HDL simulation models. You can use these models to perform simulations of your system using third-party EDA tools.

Compile & Place & Route the Design

The following steps explain how to compile and simulate your design in the MAX+PLUS II software.

1. In the MAX+PLUS II Compiler, turn on **Functional SNF Extractor** (Processing menu).

2. Click **Start** to compile your design.

3. Run the MAX+PLUS II Simulator. The vector file created by the MegaWizard Plug-In Manager for your custom FIR compiler function is loaded automatically. Click **Start** to begin simulation.

4. Once simulation has completed, click the **Open SCF** button to view the waveform for the design.

After you have verified that your design is functionally correct, you are ready to perform system verification.

Perform Synthesis Compilation & Post-Routing Simulation

As a standard feature, Altera's MAX+PLUS 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 MAX+PLUS II software and use them with your existing EDA tools to perform functional modeling and post-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-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.

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

4. Open your EDIF file in the MAX+PLUS II software.

5. Set your EDIF file as the current project in the MAX+PLUS II software.

6. Choose **EDIF Netlist Reader Settings** (Interfaces menu).

7. In the **EDIF Netlist Reader Settings** dialog box, select the vendor for your EDIF netlist file in the *Vendor* drop-down list box and click **OK**.

8. Make logic option and/or place-and-route assignments for your custom logic using the commands in the Assign menu.

For best results, you should use the *Fast* logic synthesis style in the **Global Project Logic Synthesis** dialog box (Assign menu).

9. In the MAX+PLUS II Compiler, make sure **Functional SNF Extractor** (Processing menu) is turned off.

10. Turn on the **Verilog Netlist Writer** or **VHDL Netlist Writer** command (Interfaces menu), depending on the type of output file you want to use in your third-party simulator. Set the netlist writer settings as needed.

11. Compile your design. The MAX+PLUS II Compiler synthesizes and performs place-and-route on your design, and generates output and programming files.

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

For more information on setting compiler options in your third-party EDA tool, refer to the MAX+PLUS II ACCESS Key Guidelines.

Configuring a Device

After you have compiled and analyzed your design, you are ready to configure your targeted Altera FLEX device. If you are evaluating the MegaCore function with the OpenCore feature, you must license the function before you can generate configuration files.

Example 1: I/Q Modulator

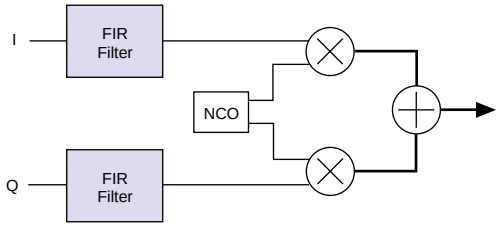
When you install the FIR compiler function, the installation program creates the directory **\reference_design\iq_modulation**. This directory contains the reference design files for an I/Q modulator application. The system parameters are shown in Table 5.

Table 5. I/Q Modulator Specifications

Function	Parameter	Value
Whole system	Data rate	1.25 MHz
	Type	16 quadrature amplitude modulation
FIR compiler specifications	Root-raised cosine – alpha	0.25
	Cut-off frequency	625 KHz
	Taps	67
Numerically controlled oscillator (NCO) specifications	Frequency resolution	1 Hz +/– 50 Hz
	Amplitude resolution	8 bits

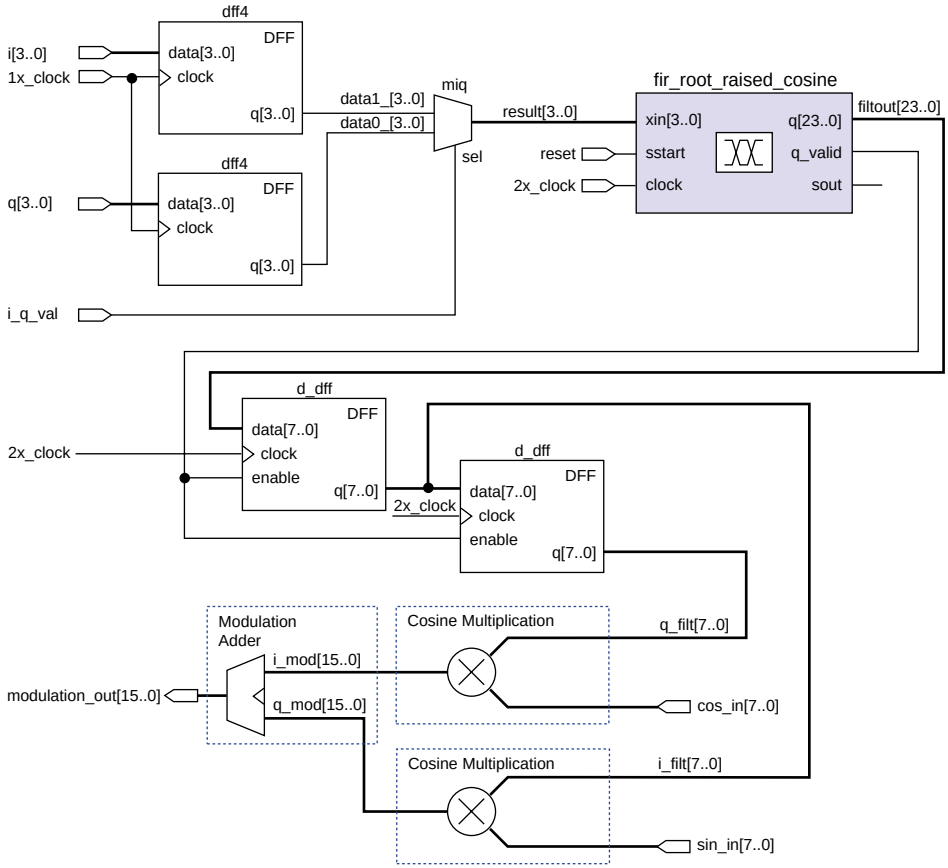
Figure 13 shows a block diagram of the system.

Figure 13. Block Diagram of QAM Example



The two FIR filters are implemented in a single instance of the FIR compiler megafunction. The function runs at twice the system clock speed. Figure 14 shows a schematic of the system. You can change the parameters of the reference design by editing the FIR compiler variation. For example, you can double-click on the symbols in the **top_modulation.gdf** file in the Graphic Editor to open the variation for editing.

Figure 14. QAM Example Schematic



Filter Design Tips

This section provides some tips for using the FIR compiler in your system.

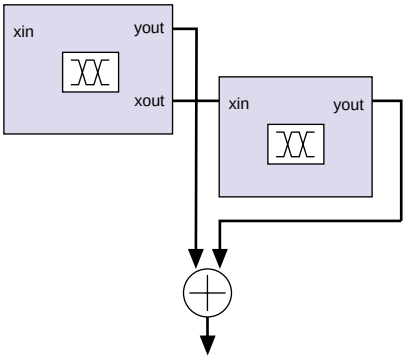
- To prevent high-pass filters from rolling off near Nyquist, choose an odd number of taps.
- You can import coefficients from the MATLAB software into the FIR compiler via a text file. Simply save your coefficients as fixed or floating-point numbers to an ASCII file, one coefficient per line. See Figure 6 on page 13 for a sample text file.
- To make a QPSK, QAM, or 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.

- If you have used all available EABs in your target FLEX 10K device and most of the device resources are unused, you can still make another filter by combining a logic cell implementation with an EAB/ESB implementation. Combining logic cells and EABs/ESBs allows you to create the maximum number of FIR filters on a single device.

- To make the smallest possible filter, use a serial implementation and FLEX 10KE EABs or APEX 20K ESBs. To make the fastest possible filter, use a parallel implementation. In both cases, you should choose the *Fast* project logic synthesis style when compiling in the MAX+PLUS II software.

- With FIR compiler, the only reason you need to cascade a filter is to make a filter across multiple devices. Figure 15 shows a cascaded filter.

Figure 15. Cascaded Filter



- 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 subsampling. The process of subsampling 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 insufficient, a coefficient is rounded towards the nearest integer, which will be zero. Therefore, if you do not scale the coefficients, you may have a filter with many zeros.

- To make an infinite impulse response (IIR) filter using the FIR compiler function, add the FIR compiler output to the IIR input and feed the result back into the FIR filter. The output for the resulting IIR filter should be taken right after the addition of the input and the output from the FIR filter.

- The fastest filters are fully parallel pipelined filters that generate an output for every clock cycle. Additionally, you should use the *Fast* logic synthesis style in the MAX+PLUS II software to use the carry and cascade chains built into the FLEX architecture, resulting in a smaller as well as a faster filter.

- To generate the most area efficient filter, create a serial multi-channel filter using FLEX 10K EABs or APEX ESBs. Additionally, for the best results you should use the *Fast* logic synthesis style in the **Global Project Logic Synthesis** dialog box (Assign menu) before compiling in the MAX+PLUS II software.