



FFT MegaCore Function

User Guide



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

MegaCore Version:	6.1
Document Version:	6.1
Document Date:	December 2006

Copyright © 2006 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, maskwork rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.



UG-FFT-3.0



About This User Guide	v
Revision History	v
How to Contact Altera	v
Typographic Conventions	vi

Chapter 1. About This MegaCore Function

Release Information	1-1
Device Family Support	1-1
New in Version 6.1	1-2
Features	1-2
General Description	1-3
Fixed Transform Size Architecture	1-3
Variable Streaming Architecture	1-3
OpenCore Plus Evaluation	1-4
DSP Builder Support	1-4
Performance	1-5

Chapter 2. Getting Started

Design Flow	2-1
FFT Walkthrough	2-2
Create a New Quartus II Project	2-3
Launch IP Toolbench	2-4
Step 1: Parameterize	2-6
Step 2: Set Up Simulation	2-10
Step 3: Generate	2-11
Simulate the Design	2-15
Simulate in the MATLAB Software (Fixed Transform Size Architectures Only)	2-15
Simulate with IP Functional Simulation Models	2-16
Simulating in Third-Party Simulation Tools Using NativeLink	2-16
Compile the Design	2-17
Fixed Transform Architectures	2-17
Variable Streaming Architecture	2-18
Program a Device	2-19
Set Up Licensing	2-20

Chapter 3. Specifications

Functional Description	3-1
Buffered, Burst, & Streaming Architectures	3-2
Variable Streaming Architecture:	3-3
The Avalon Streaming Interface	3-3

OpenCore Plus Time-Out Behavior	3-4
FFT Processor Engine Architectures	3-4
Radix-2 ² Single Delay Feedback Architecture	3-5
Quad-Output FFT Engine Architecture	3-5
Single-Output FFT Engine Architecture	3-6
I/O Data Flow Architectures	3-8
Streaming	3-8
Variable Streaming	3-10
Buffered Burst	3-12
Burst	3-14
Parameters	3-15
Signals	3-17

Appendix A. FFT/IFFT Exponent Scaling Values

Calculating Possible Exponent Values	A-1
Implementing Scaling	A-2

Appendix B. Upgrading from v2.2.1 to v6.1



About This User Guide

Revision History

The table below displays the revision history for chapters in this user guide.

Chapter	Date	Version	Changes Made
All	December 2006	6.1	<ul style="list-style-type: none">• Changed interface information.• Added variable streaming information
All	April 2006	2.2.1	Updated format.
All	October 2005	2.2.0	<ul style="list-style-type: none">• Added HardCopy® II support• Miscellaneous updates to fix bugs
1	June 2004	2.1.0	<ul style="list-style-type: none">• Updated release information and device family support tables• Updated the features• Added OpenCore Plus description• Added DSP Builder support information• Updated the performance information• Enhancements include support for Cyclone® II devices; bit-accurate MATLAB models; DSP Builder ready
2	June 2004	2.1.0	<ul style="list-style-type: none">• Updated system requirements• Updated tutorial instructions and screenshots• Added MATLAB output files names to output file table• Added MATLAB simulation and IP functional simulation model information
3	June 2004	2.1.0	<ul style="list-style-type: none">• Updated the performance information

How to Contact Altera



For technical support or other information about Altera® products, go to the Altera world-wide web site at www.altera.com. You can also contact Altera through your local sales representative or any of the sources listed below..




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

Information Type	USA & Canada	All Other Locations
Non-technical customer service	800-767-3753	+ 1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
FTP site	ftp.altera.com	ftp.altera.com

Typographic Conventions

This document uses the typographic conventions shown below.

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

Visual Cue	Meaning
	A warning calls attention to a condition or possible situation that can cause injury to the user.
	The angled arrow indicates you should press the Enter key.
	The feet direct you to more information on a particular topic.



1. About This MegaCore Function

Release Information

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

<i>Table 1–1. Product Name Release Information</i>	
Item	Description
Version	6.1
Release Date	December 2006
Ordering Code	IP-FFT
Product ID	0034
Vendor ID	6AF7

Device Family Support

MegaCore functions provide either full or preliminary support for target Altera device families:

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

Table 1–2 shows the level of support offered by the FFT MegaCore function to each of the Altera device families.

<i>Table 1–2. Device Family Support</i>	
Device Family	Support
Cyclone®	Full
Cyclone II	Full
HardCopy® II	Preliminary
HardCopy Stratix	Full
Stratix®	Full
Stratix II	Full

Table 1–2. Device Family Support

Device Family	Support
Stratix II GX	Preliminary
Stratix III	Preliminary
Stratix GX	Full
Other device families	No support

New in Version 6.1

- Fixed point variable streaming which allows runtime variable transform sizes and reduction in memory requirements
- Support for Stratix III devices
- Avalon® Streaming (ST) interfaces



For more information on Avalon Streaming interfaces, refer to the *Avalon Streaming Interface Specification*.

Features

- Bit-accurate MATLAB models for fixed-transform size FFTs
- DSP Builder ready
- Radix-4 and mixed radix-4/2 implementations
- Block floating-point architecture—maintains the maximum dynamic range of data during processing (not for variable streaming)
- Uses embedded memory
- Maximum system clock frequency >300 MHz
- Optimized to use Stratix II, Stratix GX, and Stratix DSP blocks and TriMatrix™ memory architecture
- High throughput quad-output radix 4 FFT engine
- Support for multiple single-output and quad-output engines in parallel
- Multiple I/O data flow modes: streaming, buffered burst, and burst
- Avalon-ST compliant input and output interfaces
- Parameterization-specific VHDL and Verilog HDL testbench generation
- Transform direction (FFT/IFFT) specifiable on a per-block basis
- Easy-to-use IP Toolbench interface
- IP functional simulation models for use in Altera-supported VHDL and Verilog HDL simulators

General Description

The FFT MegaCore function is a high performance, highly-parameterizable Fast Fourier transform (FFT) processor. The FFT MegaCore function implements a complex FFT or inverse FFT (IFFT) for high-performance applications.

The FFT MegaCore function implements one of the following architectures:

- Fixed transform size architecture
- Variable streaming architecture

Fixed Transform Size Architecture

The fixed transform architecture FFT implements a radix-2/4 decimation-in-frequency (DIF) FFT fixed-transform size algorithm for transform lengths of 2^m where $6 \leq m \leq 14$. This architecture uses block-floating point representations to achieve the best trade-off between maximum SNR and minimum size requirements.

The fixed transform architecture accepts as an input, a two's complement format complex data vector of length N , where N is the desired transform length in natural order; the function outputs the transform-domain complex vector in natural order. An accumulated block exponent is output to indicate any data scaling that has occurred during the transform to maintain precision and maximize the internal signal-to-noise ratio. Transform direction is specifiable on a per-block basis via an input port.

Variable Streaming Architecture

The variable streaming architecture FFT implements a radix-2² single delay feedback architecture, which you can configure during runtime to perform FFT algorithm for transform lengths of 2^m where $6 \leq m \leq 14$. This architecture uses a fixed-point representation that grows the data widths naturally from input through to output thereby maintaining a high SNR at the output.



For more information on radix-2² single delay feedback architecture, refer to *S. He and M. Torkelson, A New Approach to Pipeline FFT Processor, Department of Applied Electronics, Lund University, IPPS 1996.*

The variable streaming architecture accepts as an input, a two's complement format complex data vector N where N is also an input to the architecture, in natural order and outputs the transform-domain complex vector in bit-reversed or natural order. The transform direction is specifiable on a per-block basis via an input port.

OpenCore Plus Evaluation

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

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

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



For more information on OpenCore Plus hardware evaluation using the FFT MegaCore function, see ["OpenCore Plus Time-Out Behavior" on page 3–4](#) and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

DSP Builder Support

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

You can combine existing MATLAB/Simulink blocks with Altera DSP Builder/MegaCore blocks to verify system level specifications and generate hardware implementations. After installing this MegaCore function, a Simulink symbol of this MegaCore function appears in the Simulink library browser in the MegaCore library from the Altera DSP Builder blockset. To use this MegaCore function with DSP Builder, you require DSP Builder v6.1 or higher and the Quartus® II software v6.1 or higher.



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

Performance

Performance varies depending on the FFT engine architecture and I/O data flow. All data represents the geometric mean of a three seed Quartus II synthesis sweep. Table 1–3 shows the streaming data flow performance, using the 4 mults/2 adders complex multiplier structure, for Stratix III (EP3SL70F484C2) devices.

Table 1–3. Performance with the Streaming Data Flow Engine Architecture

Points	Width	Combinational ALUTs	Logic Registers	18 × 18 Mults	Memory (M9K)	f _{MAX} (MHz)	Clock Cycle Count	Transform Time (μs)
256	16	2,144	3,758	12	19	370	256	0.69
1,024	16	2,453	4,457	12	19	370	1,024	2.77
4,096	16	3,747	6,001	24	75	338	4,096	12.12

Table 1–4 shows the variable streaming data flow performance for Stratix III (EP3SL70F484C2) devices.



The variable streaming uses fixed-point number representation and natural word growth, therefore the multiplier requirement is larger compared with the equivalent streaming FFT with the same number of points.

Table 1–4. Performance with the Variable Streaming Data Flow Engine Architecture

Points	Width	Combinational ALUTs	Logic Registers	18 × 18 Mults	Memory (M9K)	f _{MAX} (MHz)	Clock Cycle Count	Transform Time (μs)
256	16	2,061	4,324	24	4	291	256	0.88
1,024	16	2,917	5,630	32	12	286	1,024	3.58
4,096	16	3,801	6,962	40	43	263	4,096	15.57

Table 1–5 lists resource usage with buffered burst data flow architecture, using the 4 mults/2 adders complex multiplier structure, for Stratix III (EP3SL70F484C2) devices.

Table 1–5. Resource Usage with Buffered Burst Data Flow Architecture

Points	Width (1)	Number of Engines (2)	Combinational ALUTs	Logic Registers	18 × 18 Mults	Memory M9K	f _{MAX} (MHz)
256	16	1	1,991	3,614	12	15	355
256	16	2	3,337	5,579	24	30	363
256	16	4	5,878	9,939	48	59	364
1,024	16	1	2,034	3,808	12	15	371
1,024	16	2	3,378	5,782	24	30	364
1,024	16	4	5,933	10,155	48	59	338
4,096	16	1	2,076	3,985	12	59	353
4,096	16	2	3,422	5,968	24	59	354
4,096	16	4	6,011	10,344	48	59	348

Notes to Table 1–5:

- (1) Represents data and twiddle factor precision.
- (2) When using the buffered burst architecture, you can specify the number of quad-output FFT engines in the FFT wizard.

Table 1–6 lists performance with buffered burst data flow architecture, using the 4 mults/2 adders complex multiplier structure, for Stratix III (EP3SL70F484C2) devices.

Table 1–6. Performance with the Buffered Burst Data Flow Architecture (Part 1 of 2)

Points	Width (1)	Number of Engines (2)	f _{MAX} (MHz)	Transform Calculation Time (3)		Data Load & Transform Calculation		Block Throughput (4)	
				Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
256	16	1	355	235	0.66	491	1.38	331	0.93
256	16	2	363	162	0.45	397	1.09	299	0.82
256	16	4	364	118	0.32	347	0.95	283	0.78
1,024	16	1	371	1,069	2.88	2,093	5.64	1,291	3.48
1,024	16	2	364	557	1.53	1,581	4.34	1,163	3.2
1,024	16	4	338	340	1.01	1,364	4.04	1,099	3.25

Table 1–6. Performance with the Buffered Burst Data Flow Architecture (Part 2 of 2)

Points	Width ⁽¹⁾	Number of Engines ⁽²⁾	f _{MAX} (MHz)	Transform Calculation Time ⁽³⁾		Data Load & Transform Calculation		Block Throughput ⁽⁴⁾	
				Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
4,096	16	1	353	5,167	14.64	9,263	26.24	6,157	17.44
4,096	16	2	354	2,607	7.36	6,703	18.94	5,133	14.5
4,096	16	4	348	1,378	3.96	5,474	15.73	4,633	13.31

Notes to Table 1–6:

- (1) Represents data and twiddle factor precision.
- (2) When using the buffered burst architecture, you can specify the number of quad-output engines in the FFT wizard. You may choose from one, two, or four quad-output engines in parallel.
- (3) In a buffered burst data flow architecture, transform time is defined as the time from when the *N*-sample input block is loaded until the first output sample is ready for output. Transform time does not include the additional *N*-1 clock cycle to unload the full output data block.
- (4) Block throughput is the minimum number of cycles between two successive start-of-packet (sink_sop) pulses.

Table 1–7 lists resource usage with burst data flow architecture, using the 4 mults/2 adders complex multiplier structure, for Stratix III (EP3SL70F484C2) devices.

Table 1–7. Resource Usage with the Burst Data Flow Architecture (Part 1 of 2)

Points	Width ⁽¹⁾	Engine Architecture	Number of Engines ⁽²⁾	Combinational ALUTs	Logic Registers	18 × 18 Mults	Memory (M9K)
256	16	Quad Output	1	1,834	3,550	12	7
256	16	Quad Output	2	3,053	5,515	24	14
256	16	Quad Output	4	5,337	9,869	48	27
1,024	16	Quad Output	1	1,874	3,736	12	7
1,024	16	Quad Output	2	3,091	5,709	24	14
1,024	16	Quad Output	4	5,397	10,077	48	27
4,096	16	Quad Output	1	1,913	3,903	12	27
4,096	16	Quad Output	2	3,132	5,882	24	27
4,096	16	Quad Output	4	5,470	10,266	48	27
256	16	Single Output	1	706	1,497	4	2
256	16	Single Output	2	1,021	2,345	8	8
1,024	16	Single Output	1	750	1,544	4	5
1,024	16	Single Output	2	1,037	2,422	8	10
4,096	16	Single Output	1	811	1,589	4	18

Table 1–7. Resource Usage with the Burst Data Flow Architecture (Part 2 of 2)

Points	Width (1)	Engine Architecture	Number of Engines (2)	Combinational ALUTs	Logic Registers	18 × 18 Mults	Memory (M9K)
4,096	16	Single Output	2	1,068	2,498	8	27

Notes to Table 1–7:

- (1) Represents data and twiddle factor precision.
- (2) When using the burst data flow architecture, you can specify the number of engines in the FFT wizard. You may choose from one to two single-output engines in parallel, or from one, two, or four quad-output engines in parallel.

Table 1–8 lists performance with burst data flow architecture, using the 4 mults/2 adders complex multiplier structure, for Stratix III (EP3SL70F484C2) devices.

Table 1–8. Performance with the Burst Data Flow Architecture (Part 1 of 2)

Points	Width (1)	Engine Architecture	Number of Engines (2)	f_{MAX} (MHz)	Transform Calculation Time (3)		Data Load & Transform Calculation		Block Throughput (4)	
					Cycles	Time (μ s)	Cycles	Time (μ s)	Cycles	Time (μ s)
256	16	Quad Output	1	363	235	0.65	491	1.35	331	0.91
256	16	Quad Output	2	349	162	0.46	397	1.14	299	0.86
256	16	Quad Output	4	353	118	0.33	374	1.06	283	0.8
1,024	16	Quad Output	1	357	1,069	2.99	2,093	5.86	1,291	3.62
1,024	16	Quad Output	2	351	557	1.59	1,581	4.5	1,163	3.31
1,024	16	Quad Output	4	336	340	1.01	1,364	4.06	1,099	3.27
4,096	16	Quad Output	1	366	5,167	14.12	9,263	25.31	6,157	16.82
4,096	16	Quad Output	2	358	2,607	7.28	6,703	18.72	5,133	14.34
4,096	16	Quad Output	4	349	1,378	3.95	5,474	15.68	4,633	13.28
256	16	Single Output	1	364	1,115	3.06	1,371	3.77	1,628	4.47
256	16	Single Output	2	357	585	1.64	841	2.36	1,098	3.08
1,024	16	Single Output	1	357	5,230	14.65	6,344	17.77	7,279	20.39
1,024	16	Single Output	2	371	2,652	7.15	3,676	9.91	4,701	12.67
4,096	16	Single Output	1	357	24,705	69.2	28,801	80.68	32,898	92.15

Table 1–8. Performance with the Burst Data Flow Architecture (Part 2 of 2)

Points	Width (1)	Engine Architecture	Number of Engines (2)	f_{MAX} (MHz)	Transform Calculation Time (3)		Data Load & Transform Calculation		Block Throughput (4)	
					Cycles	Time (μs)	Cycles	Time (μs)	Cycles	Time (μs)
4,096	16	Single Output	2	358	12,329	34.44	16,495	46.08	20,602	57.55

Notes to Table 1–8:

- (1) Represents data and twiddle factor precision
- (2) In the burst I/O data flow architecture, you can specify the number of engines in the FFT wizard. You may choose from one to two single-output engines in parallel, or from one, two, or four quad-output engines in parallel.
- (3) Transform time is the time frame when the input block is loaded until the first output sample (corresponding to the input block) is output. Transform time does not include the time to unload the full output data block.
- (4) Block throughput is defined as the minimum number of cycles between two successive start-of-packet (`sink_sop`) pulses.

Design Flow

To evaluate the FFT MegaCore® function using the OpenCore Plus feature, include these steps in your design flow:

1. Obtain and install the FFT MegaCore function.

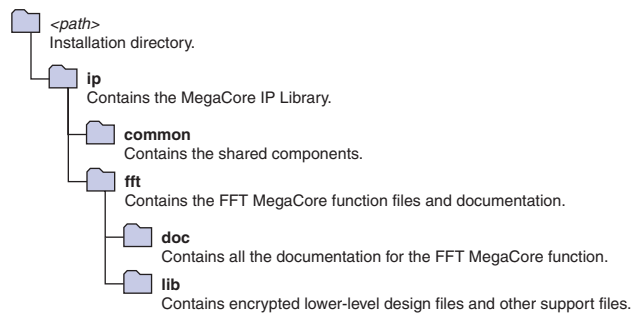
The FFT MegaCore function is part of the MegaCore IP Library, which is distributed with the Quartus® II software and downloadable from the Altera® website, www.altera.com.



For system requirements and installation instructions, refer to *Quartus II Installation & Licensing for Windows* or *Quartus II Installation & Licensing for UNIX & Linux* on the Altera website at www.altera.com/literature/lit-qts.jsp.

Figure 2-1 shows the directory structure after you install the FFT MegaCore function, where *<path>* is the installation directory. The default installation directory on Windows is `c:\altera\61`; on UNIX and Solaris it is `/opt/altera/61`.

Figure 2-1. Directory Structure



2. Create a custom variation of the FFT MegaCore function using IP Toolbench.



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

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



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

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



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



For more information on OpenCore Plus hardware evaluation using the FFT MegaCore functions, see [“OpenCore Plus Time-Out Behavior” on page 3–4](#), and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

6. Purchase a license for the FFT MegaCore function.

After you have purchased a license for the FFT, follow these additional steps:

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

FFT Walkthrough

This walkthrough explains how to create a custom variation of the FFT MegaCore function using IP Toolbench and the Quartus II software. As you go through the wizard, each step is described in detail. When you finish generating a custom variation of the FFT MegaCore function, you can incorporate it into your overall project.

This walkthrough requires the following steps:

- [Create a New Quartus II Project](#)
- [Launch IP Toolbench](#)

- Step 1: Parameterize
- Step 2: Set Up Simulation
- Step 3: Generate

Create a New Quartus II Project

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

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



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

- b. Specify the name of the project. This walkthrough uses **example** for the project name.
5. Click **Next** to close this page and display the **New Project Wizard: Add Files** page.



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

6. If you installed the MegaCore IP Library in a different directory from where you installed the Quartus II software, you must add the user libraries:
 - a. Click **User Libraries**.

- b. Type `<path>\ip` into the **Library name** box, where `<path>` is the directory in which you installed the FFT.
 - c. Click **Add** to add the path to the Quartus II project.
 - d. Click **OK** to save the library path in the project.
7. Click **Next** to close this page and display the **New Project Wizard: Family & Device Settings** page.
8. On the **New Project Wizard: Family & Device Settings** page, choose the target device family in the **Family** list.
9. The remaining pages in the **New Project Wizard** are optional. Click **Finish** to complete the Quartus II project.

You have finished creating your new Quartus II project.

Launch IP Toolbench

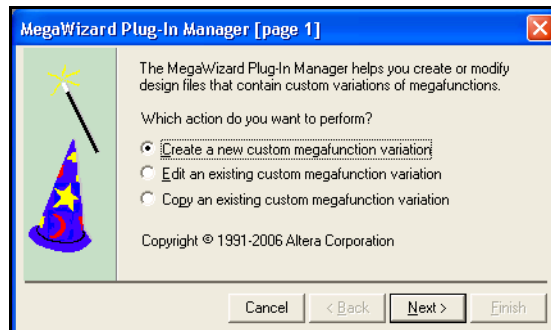
To launch IP Toolbench in the Quartus II software, follow these steps:

1. Start the MegaWizard® Plug-In Manager by choosing **MegaWizard Plug-In Manager** (Tools menu). The **MegaWizard Plug-In Manager** dialog box displays (see [Figure 2–2](#)).



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

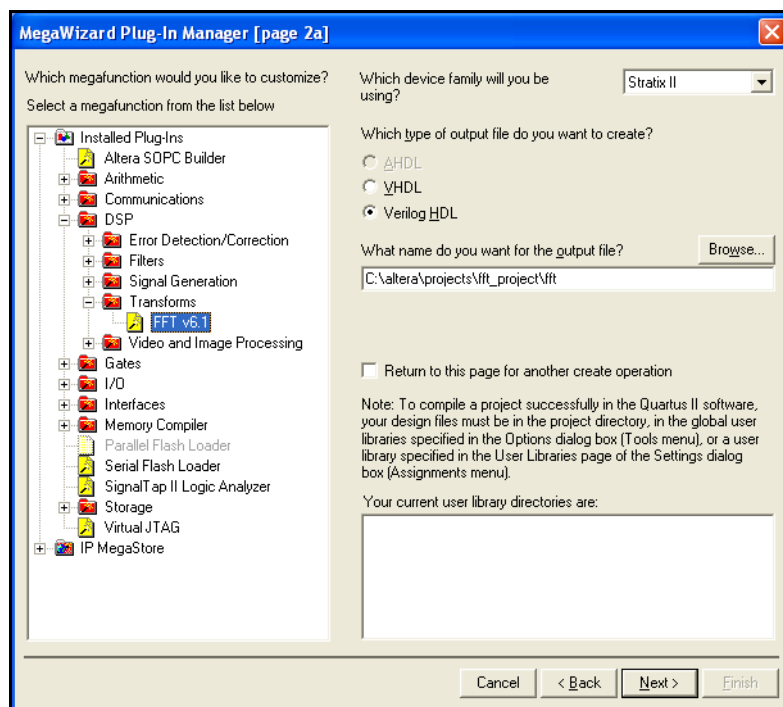
Figure 2–2. MegaWizard Plug-In Manager



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

3. Expand the **DSP > Transforms** directory and click **FFT v6.1**.
4. Choose the output file type for your design; the wizard supports, VHDL and Verilog HDL.
5. The MegaWizard Plug-In Manager shows the project path that you specified in the **New Project Wizard**. Append a variation name for the MegaCore function output files `<project path>\<variation name>`. [Figure 2-3](#) shows the wizard after you have made these settings.

Figure 2-3. Select the MegaCore Function



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

Step 1: Parameterize

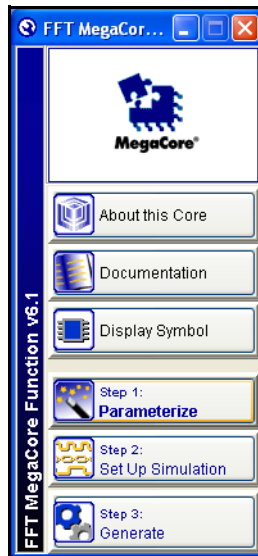
To create a custom variation of the FFT MegaCore function, follow these steps:



For more information on the parameters, see [“Parameters” on page 3–15](#).

1. Click the **Step 1: Parameterize** button in IP Toolbench (see [Figure 2–4](#)).

Figure 2–4. IP Toolbench—Parameterize

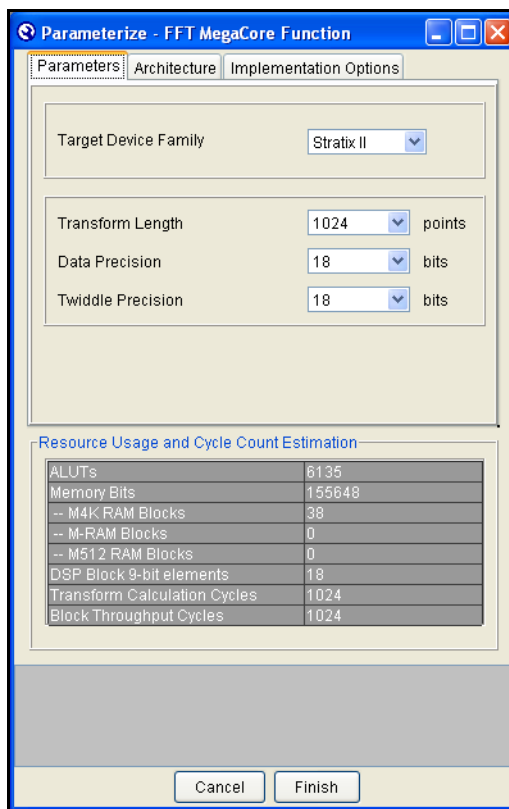


2. Choose the **Target Device Family** (see [Figure 2–5](#)).



For HardCopy® II devices, choose **Stratix II**.

Figure 2–5. Parameters Tab



- Choose the **Transform length**, **Data precision**, and **Twiddle precision**.



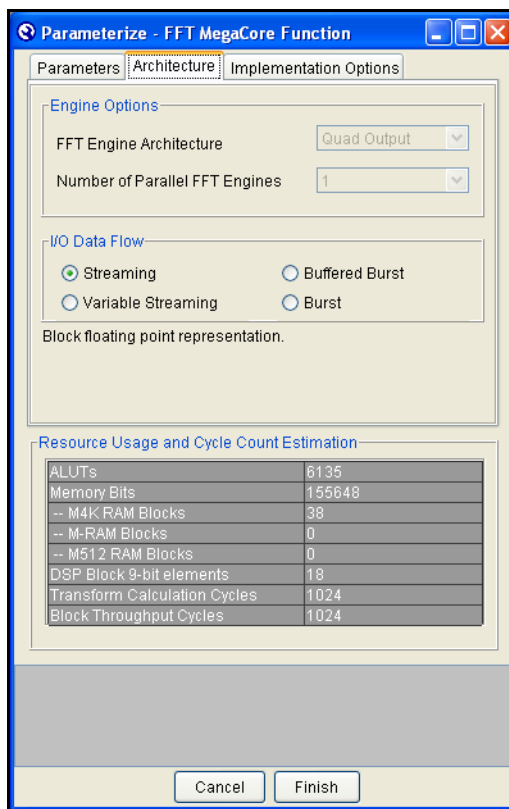
Twiddle factor precision must be less than or equal to data precision.



If the variable streaming option is chosen (see **Architecture** Tab), the **Transform length** represents the maximum transform length that can be performed. All transforms of length 2^m where $6 \leq m \leq \log_2(\text{transform length})$ can be performed at runtime.

- Click the **Architecture** tab (see Figure 2–6).

Figure 2–6. Architecture Tab



- Choose the **FFT Engine Architecture**, **Number of Parallel FFT Engines**, and select the **I/O Data Flow**.

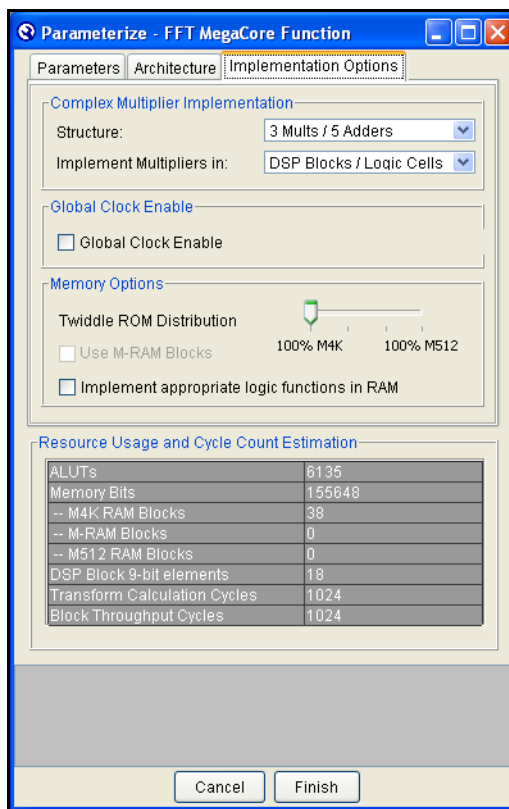
If you select the **Streaming** I/O data flow, the FFT Megacore function automatically generate a design with a **Quad Output** FFT engine architecture and the minimum number parallel FFT engines for the required throughput.



A single FFT engine architecture provides enough performance for up to a 1,024-point streaming I/O data flow FFT.

- Click the **Implementation Options** tab (see [Figure 2–7](#)).

Figure 2-7. Implementation Options Tab



7. Choose the complex multiplier structure.
8. Choose how you want to implement the multipliers.
9. Turn on **Global Clock Enable**, if you want to add a global clock enable to your design.
10. Specify the memory options.
11. Click **Finish** when the implementation options are set.



The implementation options are not available for the variable streaming architecture.

Step 2: Set Up Simulation

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

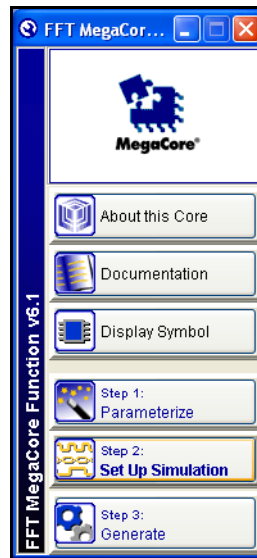


You may only use these simulation model output files for simulation purposes and expressly not for synthesis or any other purposes. Using these models for synthesis will create a nonfunctional design.

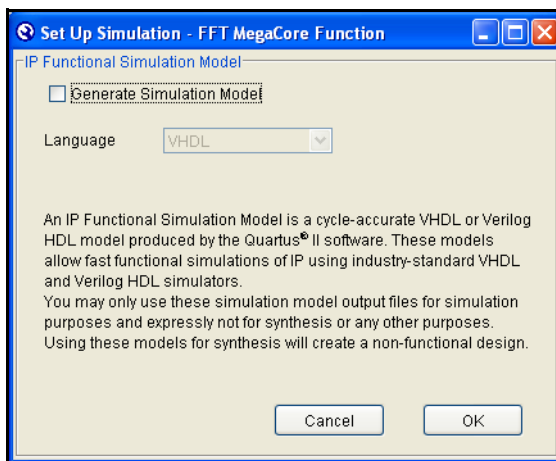
To generate an IP functional simulation model for your MegaCore function, follow these steps:

1. Click the **Step 2: Set Up Simulation** button in IP Toolbench (see [Figure 2-8](#)).

Figure 2-8. IP Toolbench—Set Up Simulation



2. Turn on **Generate Simulation Model** (see [Figure 2-9](#)).

Figure 2–9. Generate Simulation Model

3. Choose the language in the **Language** list.
4. Click **OK**.

Step 3: Generate

To generate your MegaCore function, follow these steps:

1. Click the **Step 3: Generate** button in IP Toolbench (see [Figure 2–10](#)).

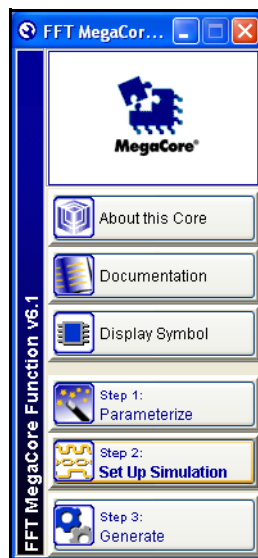
Figure 2–10. IP Toolbench—Generate

Figure 2–11 shows the generation report.

Figure 2–11. Generation Report

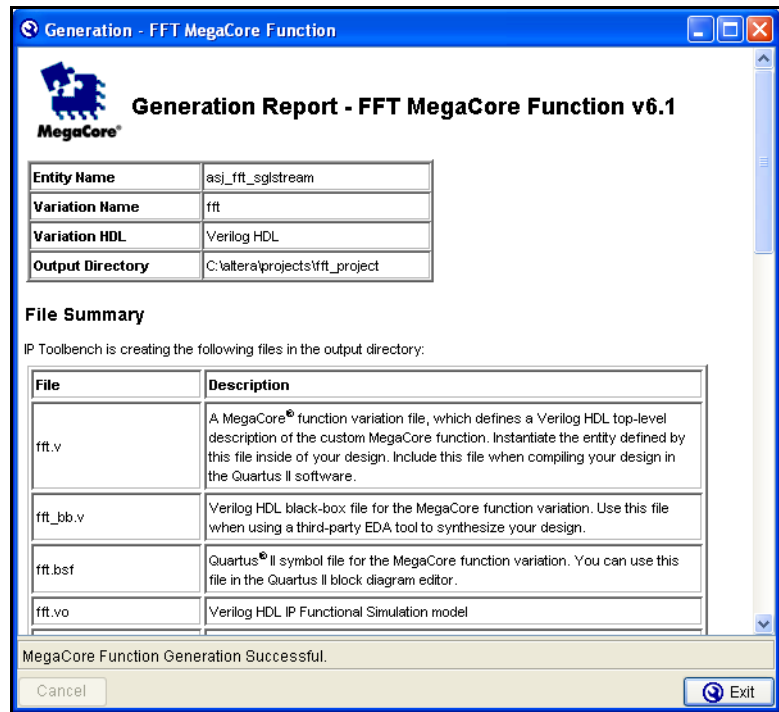


Table 2–1 describes the generated files and other files that may be in your project directory. The names and types of files specified in the IP Toolbench report vary based on whether you created your design with VHDL or Verilog HDL

Table 2–1. Generated Files (Part 1 of 2) Notes (1) & (2)	
Filename	Description
imag_input.txt	The text file contains input imaginary component random data. This file is read by the generated VHDL or Verilog HDL MATLAB testbenches.
real_input.txt	Test file containing real component random data. This file is read by the generated VHDL or Verilog HDL and MATLAB testbenches.
<variation name>.bsf	Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.

Table 2–1. Generated Files (Part 2 of 2) <i>Notes (1) & (2)</i>	
Filename	Description
<variation name>.cmp	A VHDL component declaration file for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.
<variation name>.html	MegaCore function report file.
<variation name>.vo or .vho	VHDL or Verilog HDL IP functional simulation model.
<variation name>.vhd, or .v	A MegaCore function variation file, which defines a VHDL or Verilog HDL top-level description of the custom MegaCore function. Instantiate the entity defined by this file inside of your design. Include this file when compiling your design in the Quartus II software.
<variation name>_bb.v	Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
<variation name>_bit_reverse_top.vhd	Example top-level VHDL design with bit-reversal module.
<variation name>_1n1024cos.hex, <variation name>_2n1024cos.hex, <variation name>_3n1024cos.hex	Intel hex-format ROM initialization files (not generated for variable streaming FFT).
<variation name>_1n1024sin.hex, <variation name>_2n1024sin.hex, <variation name>_3n1024sin.hex	Intel hex-format ROM initialization files (not generated for variable streaming FFT).
<variation name>_model.m	MATLAB m-file describing a MATLAB bit-accurate model.
<variation name>_tb.m	MATLAB testbench (not generated for variable streaming FFTs).
<variation name>_tb.v or <variation name>_tb.vhd	Verilog HDL or VHDL testbench file.
<variation name>_nativelink.tcl	Tcl Script that sets up NativeLink in the Quartus II software to natively simulate the design using selected EDA tools see “Simulating in Third-Party Simulation Tools Using NativeLink” on page 2–16 .
twr1.hex, twi1.hex, twr2.hex, twi2.hex, twr3.hex, twi3.hex, twr4.hex, twi4.hex,	Intel hex-format ROM initialization files (variable streaming FFT only).
Notes to Table 2–1: (1) These files are variation dependent, some may be absent or their names may change. (2) <variation name> is a prefix variation name supplied automatically by IP Toolbench.	

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

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

Simulate the Design

This section describes the following simulation techniques:

- [Simulate in the MATLAB Software \(Fixed Transform Size Architectures Only\)](#)
- [Simulate with IP Functional Simulation Models](#)
- [Simulating in Third-Party Simulation Tools Using NativeLink](#)

Simulate in the MATLAB Software (Fixed Transform Size Architectures Only)

The FFT MegaCore function outputs a bit-accurate MATLAB model `<variation name>_model.m`, which you can use to model the behavior of your custom FFT variation in the MATLAB software. The model takes a complex vector as input and it outputs the transform-domain complex vector and corresponding block exponent values. The length and direction of the transform (FFT/IFFT) is also passed as an input to the model.



This MATLAB model is not generated for variable streaming FFTs.

If the input vector length is an integral multiple of N , the transform length, the length of the output vector(s) is equal to the length of the input vector. However, if the input vector is not an integral multiple of N , it is zero-padded to extend the length to be so.



For additional information on exponent values, refer to the application note, *AN 404: FFT/IFFT Block Floating Point Scaling*, available on the Altera web site at www.altera.com/literature/an/an404.pdf.

The wizard also creates the MATLAB testbench file `<variation name>_tb.m`. This file creates the stimuli for the MATLAB model by reading the input complex random data from files generated by IP Toolbench. To model your FFT MegaCore function variation in the MATLAB software, follow these steps:

1. Run the MATLAB software.
2. In the MATLAB command window, change to the working directory for your project.

3. Perform the simulation:

- a. Type `help <variation name>_model` at the command prompt to view the input and output vectors that are required to run the MATLAB model as a standalone M-function. Create your input vector and make a function call to `<variation name>_model`. For example:

```
N=2048;

INVERSE = 0; % 0 => FFT 1=> IFFT

x = (2^12)*rand(1,N) + j*(2^12)*rand(1,N);

[y,e] = <variation name>_model(x,N,INVERSE);
```

or

- b. Run the provided testbench by typing the name of the testbench, `<variation name>_tb` at the command prompt.



For more information on MATLAB and Simulink, refer to the MathWorks web site at www.mathworks.com.

Simulate with IP Functional Simulation Models

To simulate your design, use the IP functional simulation models generated by IP Toolbench. The IP functional simulation model is the `.vo` or `.vho` file generated as specified in “[Step 2: Set Up Simulation](#)” on [page 2–10](#). Compile the `.vo` or `.vho` file in your simulation environment to perform functional simulation of your custom variation of the MegaCore function.



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

Simulating in Third-Party Simulation Tools Using NativeLink

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



For more information on NativeLink, refer to the *Simulating Altera IP Using NativeLink* chapter in volume 3 of the *Quartus II Handbook*.

You can use the Tcl script file `<variation name>_nativelink.tcl` to assign default NativeLink testbench settings to the Quartus II project.

To set up simulation in the Quartus II software using NativeLink, follow these steps:

1. Create a custom variation but ensure you specify your variation name to match the Quartus II project name.
2. Check that the absolute path to your third-party simulator executable is set. On the Tools menu click **Options** and select **EDA Tools Options**.
3. On the Processing menu, point to **Start** and click **Start Analysis & Elaboration**.
4. On the Tools menu click **Tcl scripts**. Select the the *<variation name>_nativelink.tcl* Tcl script and click **Run**. Check for a message confirming that the Tcl script was successfully loaded.
5. On the Assignments menu click **Settings**, expand **EDA Tool Settings** and select **Simulation**. Select a simulator under **Tool Name** and in **NativeLink Settings**, select **Test Benches**.
6. On the Tools menu point to **EDA Simulation Tool** and click **Run EDA RTL Simulation**.

Compile the Design

Use the Quartus II software to synthesize and place and route your design. Refer to Quartus II Help for instructions on performing compilation.

Fixed Transform Architectures

To compile your fixed-transform architecture design, perform the following steps:

1. If you are using the Quartus II software to synthesize your design, skip to step 2. If you are using a third-party synthesis tool to synthesize your design, follow these steps:
 - a. Set a black box attribute for your FFT MegaCore function custom variation before you synthesize the design. Refer to Quartus II Help for instructions on setting black-box attributes per synthesis tool.
 - b. Run the synthesis tool to produce an EDIF Netlist File (.edf) or Verilog Quartus Mapping (VQM) file (.vqm) for input to the Quartus II software.

- c. Add the EDIF or VQM file to your Quartus II project.
2. Choose **Add/Remove Files in Project** (Project menu).
3. If the **fft_pack.vhd** file is not listed, browse to the `<path>\fft-v2.2.1\lib` directory and choose the VHDL package, **fft_pack.vhd**.
4. Click **Open** to add the **fft_pack.vhd** file to your Quartus II project.



Ensure the **fft_pack.vhd** file is at the top of the list in the **File Name** list window. If the **fft_pack.vhd** file is not at the top of the **File Name** list, choose **fft_pack.vhd** and click **Up**.

- -
 -
 -
 5. Choose **Start Compilation** (Processing menu).

Variable Streaming Architecture

To compile your variable streaming architecture design, follow these steps:

1. If you are using the Quartus II software to synthesize your design, skip to step 2. If you are using a third-party synthesis tool to synthesize your design, follow these steps:
 - a. Set a black box attribute for your FFT MegaCore function custom variation before you synthesize the design. Refer to Quartus II Help for instructions on setting black-box attributes per synthesis tool.
 - b. Run the synthesis tool to produce an EDIF Netlist File (**.edf**) or Verilog Quartus Mapping (VQM) file (**.vqm**) for input to the Quartus II software.
 - c. Add the EDIF or VQM file to your Quartus II project.
2. Choose **Add/Remove Files in Project** (Project menu).
3. You should see a list of files in the project. If there are no files listed browse to the `<path>\fft-v6.1\lib`, then select and add all files with the prefix **auk_dspip_r22sdf** and **auk_dspip_bit_reverse**. Browse to the `<project>` directory and select all files with prefix **auk_dspip**.
4. Chose **Start Compilation** (Processing menu).

The FFT variable streaming architecture produces a top-level design file that instantiates the FFT variable streaming engine and the bit reversal module. The model allows you to produce natural-order outputs from your design. To compile this top-level design file, follow these steps:

1. If you are using the Quartus II software to synthesize your design, skip to step 2. If you are using a third-party synthesis tool to synthesize your design, follow these steps:
 - a. Set a black box attribute for your FFT MegaCore function custom variation before you synthesize the design. Refer to Quartus II Help for instructions on setting black-box attributes per synthesis tool.
 - b. Run the synthesis tool to produce an EDIF Netlist File (.edf) or Verilog Quartus Mapping (VQM) file (.vqm) for input to the Quartus II software.
 - c. Add the EDIF or VQM file to your Quartus II project.
2. Choose **Add/Remove Files in Project** (Project menu).
3. You should see a list of files in the project. If there are no files listed browse to the `<path>\fft-v6.1\lib`, then select and add all files with the prefix **auk_dspip_r22sdf** and **auk_dspip_bit_reverse**. Browse to the `<project>` directory and select all files with prefix **auk_dspip**. Select the file `<variation name>_bit_reverse_top.vhd`. Click **Add** to add this file to the project
4. Change the top-level entity name to `<variation name>_bit_reverse_top`:
 - a. On the Assignments menu click **Settings**.
 - b. Click **General**.
 - c. Select `<variation name>_bit_reverse_top` and click **OK**.
5. Choose **Start Compilation** (Processing menu)

Program a Device

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

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



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

You can simulate the FFT in your design, and perform a time-limited evaluation of your design in hardware.



For more information on OpenCore Plus hardware evaluation using the FFT, see “[OpenCore Plus Time-Out Behavior](#)” on page 3–4 and AN 320: *OpenCore Plus Evaluation of Megafunctions*.

Set Up Licensing

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

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

Functional Description

The discrete Fourier transform (DFT), of length N , calculates the sampled Fourier transform of a discrete-time sequence at N evenly distributed points $\omega_k = 2\pi k/N$ on the unit circle.

Equation 1 shows the length- N forward DFT of a sequence $x(n)$:

$$X[k] = \sum_{n=0}^{N-1} x(n)e^{(-j2\pi nk)/N}$$

where $k = 0, 1, \dots, N-1$

Equation 2 shows the length- N inverse DFT:

$$x(n) = (1/N) \sum_{k=0}^{N-1} X[k]e^{(j2\pi nk)/N}$$

where $n = 0, 1, \dots, N-1$

The complexity of the DFT direct computation can be significantly reduced by using fast algorithms that use a nested decomposition of the summation in equations one and two—in addition to exploiting various symmetries inherent in the complex multiplications. One such algorithm is the Cooley-Tukey radix- r decimation-in-frequency (DIF) FFT, which recursively divides the input sequence into N/r sequences of length r and requires $\log_r N$ stages of computation.

Each stage of the decomposition typically shares the same hardware, with the data being read from memory, passed through the FFT processor and written back to memory. Each pass through the FFT processor is required to be performed $\log_r N$ times. Popular choices of the radix are $r = 2, 4$, and 16 . Increasing the radix of the decomposition leads to a reduction in the number of passes required through the FFT processor at the expense of device resources.

Buffered, Burst, & Streaming Architectures

A radix-4 decomposition, which divides the input sequence recursively to form four-point sequences, has the advantage that it requires only trivial multiplications in the four-point DFT and is the chosen radix in the Altera® FFT MegaCore® function. This results in the highest throughput decomposition, while requiring non-trivial complex multiplications in the post-butterfly twiddle-factor rotations only. In cases where N is an odd power of two, the FFT MegaCore automatically implements a radix-2 pass on the last pass to complete the transform.

To maintain a high signal-to-noise ratio throughout the transform computation, the FFT MegaCore function uses a block-floating-point architecture, which is a trade-off point between fixed-point and full-floating point architectures.

In a fixed-point architecture, the data precision needs to be large enough to adequately represent all intermediate values throughout the transform computation. For large FFT transform sizes, an FFT fixed-point implementation that allows for word growth can make either the data width excessive or can lead to a loss of precision.

In a floating-point architecture each number is represented as a mantissa with an individual exponent—while this leads to greatly improved precision, floating-point operations tend to demand increased device resources.

In a block-floating point architecture, all of the values have an independent mantissa but share a common exponent in each data block. Data is input to the FFT function as fixed point complex numbers (even though the exponent is effectively 0, you do not enter an exponent).

The block-floating point architecture ensures full use of the data width within the FFT function and throughout the transform. After every pass through a radix-4 FFT, the data width may grow up to $4\sqrt{2} = 5.65$ bits. The data is scaled according to a measure of the block dynamic range on the output of the previous pass. The number of shifts is accumulated and then output as an exponent for the entire block. This shifting ensures that the minimum of least significant bits (LSBs) are discarded prior to the rounding of the post-multiplication output. In effect, the block-floating point representation acts as a digital automatic gain control. To yield uniform scaling across successive output blocks, you must scale the FFT function output by the final exponent.



In comparing the block-floating point output of the Altera FFT MegaCore function to the output of a full precision FFT from a tool like MATLAB, the output should be scaled by $2^{(-\text{exponent_out})}$ to account for the discarded LSBs during the transform (see “FFT/IFFT Exponent Scaling Values” on page A-1).



For additional information on exponent values, refer to the application note, “AN 404: FFT/IFFT Block Floating Point Scaling,” available on the Altera web site at www.altera.com/literature/an/an404.pdf.

Variable Streaming Architecture:

The variable streaming architecture uses a radix 2^2 single delay feedback architecture, which is a fully pipelined architecture. For a length N transform there are $\log_4(N)$ stages concatenated together. The radix 2^2 algorithm has the same multiplicative complexity of a fully pipelined radix-4 architecture, however the butterfly unit retains a radix-2 architecture. The butterfly units use the DIF decomposition.

The variable streaming architecture uses fixed point representation and the architecture allows for natural word growth through the pipeline. The maximum growth of each stage is $\log_2(4\sqrt{2}) = 2.5$ bits, which is accommodated in the design by growing the pipeline stages by either 2 bits or 3 bits. After the complex multiplication the data is rounded down (towards 0) to the expanded data size.

Output from the variable streaming architecture is bit-reversed. IP Toolbench generates a separate top-level file `<variation name>_bit_reverse_top.vhd`, which provides a bit reversal module in addition to the FFT MegaCore function, to produce in-order outputs.

The Avalon Streaming Interface

The Avalon® Streaming (Avalon-ST) interface is an evolution of the Atlantic™ interface. The Avalon-ST interface defines a standard, flexible, and modular protocol for data transfers from a source interface to a sink interface and simplifies the process of controlling the flow of data in a datapath. The Avalon-ST interface signals can describe traditional streaming interfaces supporting a single stream of data without knowledge of channels or packet boundaries. Such interfaces typically contain data, ready, and valid signals. The Avalon-ST interface can also support more complex protocols for burst and packet transfers with packets interleaved across multiple channels. The Avalon-ST interface inherently synchronizes multi-channel designs, which allows you to achieve efficient, time-multiplexed implementations without having to implement complex control logic.

The Avalon-ST interface supports backpressure, which is a flow control mechanism, where a sink can signal to a source to stop sending data. The sink typically uses backpressure to stop the flow of data when its FIFO buffers are full or when there is congestion on its output. When designing a datapath, which includes the FFT MegaCore function, you may not need backpressure if you know the downstream components can always receive data. You may achieve a higher clock rate by driving the source ready signal `source_ready` of the FFT high, and not connecting the sink ready signal `sink_ready`.



For more information on the Avalon-ST interface, refer to the *Avalon Streaming Interface Specification*.

OpenCore Plus Time-Out Behavior

OpenCore Plus hardware evaluation can support the following two modes of operation:

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

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



For MegaCore functions, the untethered time-out is one hour; the tethered time-out value is indefinite.

The signals `source_real`, `source_imag`, and `source_exp` are forced low when the evaluation time expires.



For more information on OpenCore Plus hardware evaluation, see [“OpenCore Plus Evaluation” on page 1–4](#) and AN 320: *OpenCore Plus Evaluation of Megafunctions*.

FFT Processor Engine Architectures

The FFT MegaCore function can be parameterized to use either quad-output or single-output engine architecture. To increase the overall throughput of the FFT MegaCore function, you may also use multiple parallel engines of a variation. This section discusses the following topics:

- Radix 2^2 single-delay feedback architecture
- Quad-output FFT engine architecture
- Single-output FFT engine architecture

Radix-2² Single Delay Feedback Architecture

Radix-2² single delay feedback architecture is a fully pipelined architecture for calculating the FFT of incoming data. It is similar to radix-2 single delay feedback architectures. However, the twiddle factors are rearranged such that the multiplicative complexity is equivalent to a radix-4 single delay feedback architecture.

There are $\log_2(N)$ stages with each stage containing a single butterfly unit and a feedback delay unit that delays the incoming data by a specified number of cycles, halved at every stage. These delays effectively align the correct samples at the input of the butterfly unit for the butterfly calculations. Every second stage contains a modified radix-2 butterfly whereby a trivial multiplication by $-j$ is performed before the radix-2 butterfly operations. The output of the pipeline is in bit reversed order.

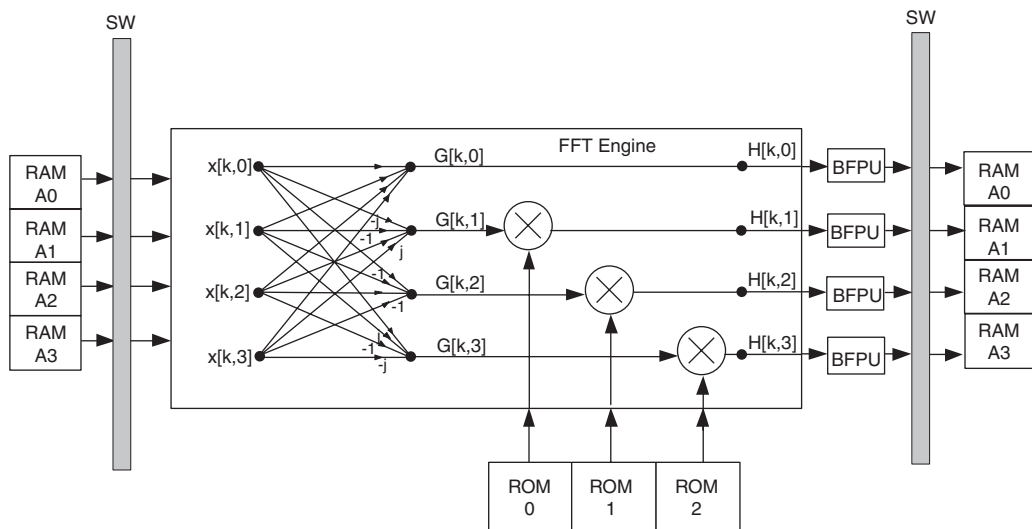
The following scheduled operations in the pipeline for an FFT of length $N = 16$ occur.

1. For the first 8 clock cycles, the samples are fed unmodified through the butterfly unit to the delay feedback unit.
2. The next 8 clock cycles perform the butterfly calculation using the data from the delay feedback unit and the incoming data. The higher order calculations are sent through to the delay feedback unit while the lower order calculations are sent to the next stage.
3. The next 8 clock cycles feeds the higher order calculations stored in the delay feedback unit unmodified through the butterfly unit to the next stage.

Subsequent data stages use the same principles. However, the delays in the feedback path are adjusted accordingly.

Quad-Output FFT Engine Architecture

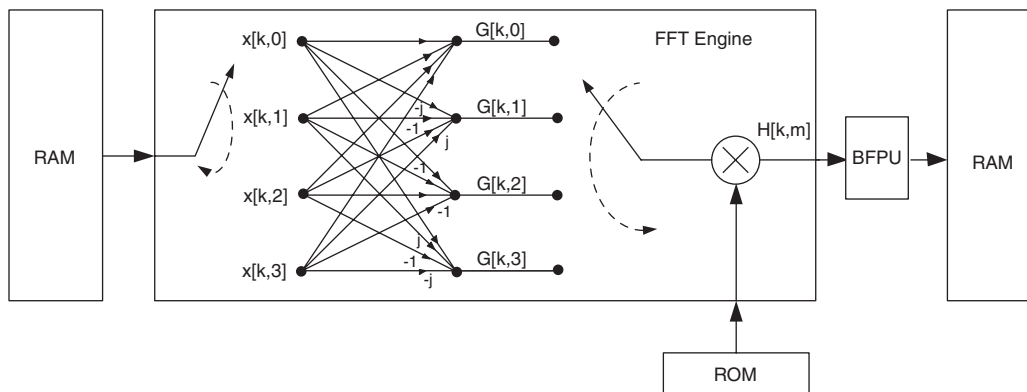
For applications where transform time is to be minimized, a quad-output FFT engine architecture is optimal. The term quad-output refers to the throughput of the internal FFT butterfly processor. The engine implementation computes all four radix-4 butterfly complex outputs in a single clock cycle. [Figure 3-1](#) shows a diagram of the quad-output FFT engine.

Figure 3–1. Quad-Output FFT Engine

Complex data samples $x[k,m]$ are read from internal memory in parallel and re-ordered by switch (SW). Next, the ordered samples are processed by the radix-4 butterfly processor to form the complex outputs $G[k,m]$. Because of the inherent mathematics of the radix-4 DIF decomposition, only three complex multipliers are required to perform the three non-trivial twiddle-factor multiplications on the outputs of the butterfly processor. To discern the maximum dynamic range of the samples, the four outputs are evaluated in parallel by the block-floating point units (BFPUs). The appropriate LSBs are discarded and the complex values are rounded and re-ordered before being written back to internal memory.

Single-Output FFT Engine Architecture

For applications where the minimum-size FFT function is desired, a single-output engine is most suitable. The term single-output again refers to the throughput of the internal FFT butterfly processor. In the engine architecture, a single butterfly output is computed per clock cycle, requiring a single complex multiplier (see [Figure 3–2](#)).

Figure 3–2. Single-Output FFT Engine Architecture

I/O Data Flow Architectures

This section describes and illustrates the following I/O data flow architectural options supported by the FFT MegaCore function:

- Streaming
- Variable Streaming
- Buffered Burst
- Burst



For information on setting the architectural parameters in IP Toolbench, refer to “[Step 1: Parameterize](#)” on page 2–6.

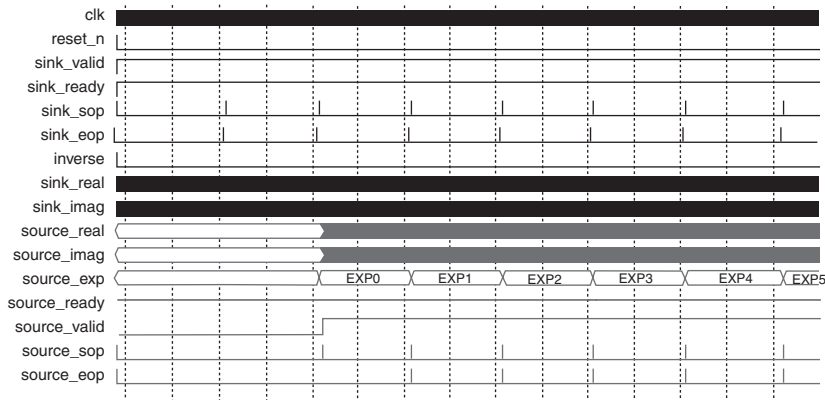
Streaming

The streaming I/O data flow FFT architecture allows continuous processing of input data, and outputs a continuous complex data stream without the requirement to halt the data flow in or out of the FFT function. [Figure 3–3](#) shows an example simulation waveform.



For more information on the signals, see [Table 3–3](#) on page 3–18.

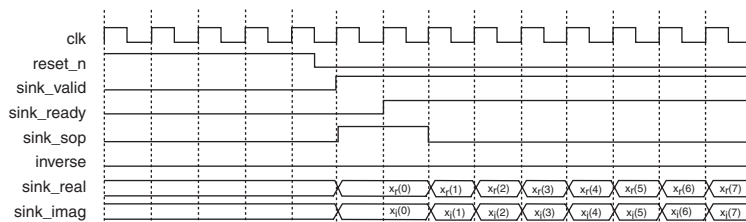
Figure 3–3. FFT Streaming Data Flow Architecture Simulation Waveform



Following the de-assertion of the system reset, the data source asserts `sink_valid` to indicate to the FFT function that it has data samples available for input. In response, the FFT function asserts `sink_ready` to indicate to the FFT function that it has valid data available.. The data source loads the first complex data sample into the FFT function by simultaneously asserting `sink_sop` and `sink_valid` to indicate the start of the input block.

When the data transfer is complete (`sink_eop` is asserted), `sink_sop` is de-asserted and the data samples are loaded in natural order. [Figure 3–4](#) shows the input flow control. When the samples are loaded the `sink_eop` must be asserted together with `valid` for the last data transfer.

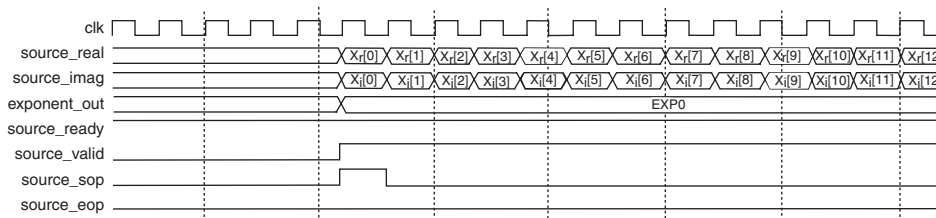
Figure 3–4. FFT Streaming Data Flow Architecture Input Flow Control



To change direction on a block-by-block basis, assert/de-assert `inverse` (appropriately) simultaneously with the application of the `sink_sop` pulse (concurrent with the first input data sample of the block).

When the FFT has completed the transform of the input block, it asserts `source_valid` and outputs the complex transform domain data block in natural order. The FFT function asserts the pulse signal `source_sop` to indicate the first output sample. [Figure 3–5](#) shows the output flow control.

Figure 3–5. FFT Streaming Data Flow Architecture Output Flow Control



After N data transfers, `source_eop` is asserted to indicate the end of the output data block (see [Figure 3–3](#)).



The `sink_valid` signal must be asserted for `source_valid` to be asserted (and a valid data output). You must therefore leave `sink_valid` signal asserted at the end of data transfers to extract the final frames of data from the FFT.

Variable Streaming

The variable streaming architecture allows continuous streaming of input data and produces a continuous stream of output data similar to the streaming architecture.

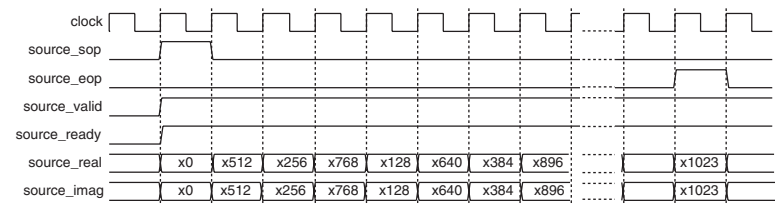
You change the size of the FFT on a block-by-block basis by changing the value of the `fftpoints` simultaneously with the application of the `sink_sop` pulse (concurrent with the first input data sample of the block). `fftpoints` uses a binary representation of the size of the transform, therefore for a block with maximum transfer size of 1,024. [Table 3–1](#) shows the value of the `fftpoints` signal and the equivalent transform size.

Table 3–1. <i>fftpoints & Transform Size</i>	
fftpoints	Transform Size
1000000000	1,024
0100000000	512
0010000000	256
0001000000	128
0000100000	64

To change direction on a block-by-block basis, assert or de-assert `inverse` (appropriately) simultaneously with the application of the `sink_sop` pulse (concurrent with the first input data sample of the block).

When the FFT has completed the transform of the input block, it asserts `source_valid` and outputs the complex transform domain data block in bit-reversed order. The FFT function asserts the pulse signal `source_sop` to indicate the first output sample. [Figure 3–6](#) shows the output flow control.

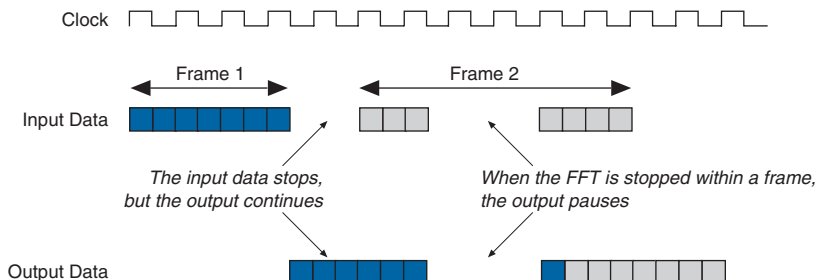
Figure 3–6. Output Flow Control—Bit Reversed Order



Enabling the FFT

The FFT processes data when there is valid data transferred to the module (`sink_valid` asserted). [Figure 3-7](#) shows the FFT behavior when `sink_valid` is de-asserted.

Figure 3-7. FFT Behavior When `sink_valid` is Deasserted



When `sink_valid` is de-asserted during a frame, the FFT stalls and no data is processed until `sink_valid` is reasserted. This implies that any previous frames that are still in the FFT also stall.

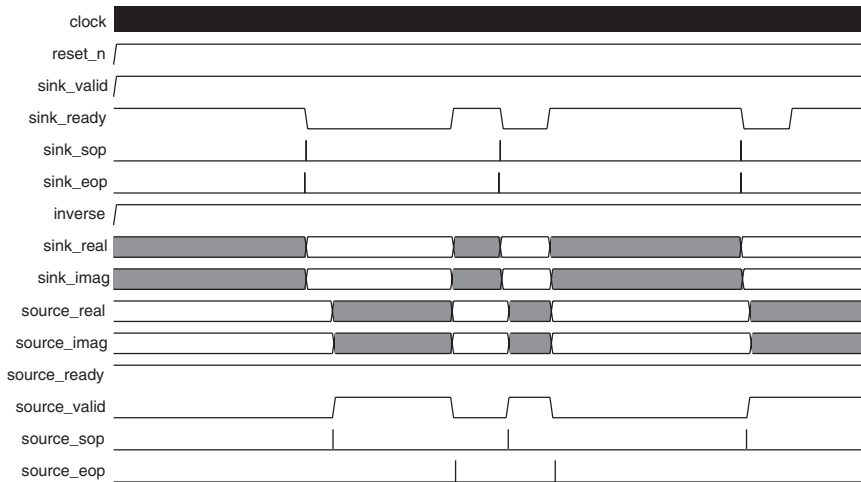
If `sink_valid` is de-asserted between frames, the data currently in the FFT continues to be processed and transferred to the output. [Figure 3-7](#) shows the FFT behavior when `sink_valid` is de-asserted between frames and within a frame.

This behavior is different to the behavior of the FFT MegaCore function v2.2.1 and earlier, where the `sink_valid` signal must be asserted for the `source_valid` to be asserted (and valid data output). You must leave the `sink_valid` signal asserted at the end of the data transfers to extract the final frames of data from the FFT.

The FFT may optionally be disabled by deasserting the `clk_en` signal.

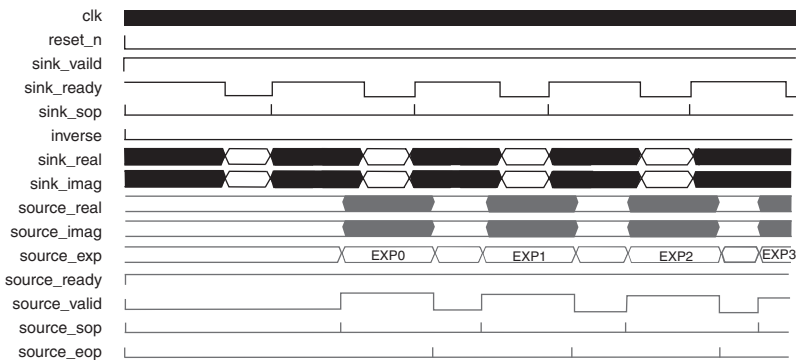
Dynamically Changing the FFT Size

When the size of the incoming FFT changes, the FFT stalls the incoming data (deasserts the `sink_ready` signal) until all of the previous FFT frames of the previous FFT size have been processed and transferred to the output. [Figure 3-8](#) shows dynamically changing the FFT size.

Figure 3–8. Dynamically Changing the FFT Size

Buffered Burst

The buffered burst I/O data flow architecture FFT requires fewer memory resources than the streaming I/O data flow architecture, but the tradeoff is an average block throughput reduction. [Figure 3–9](#) shows an example simulation waveform.

Figure 3–9. FFT Buffered Burst Data Flow Architecture Simulation Waveform

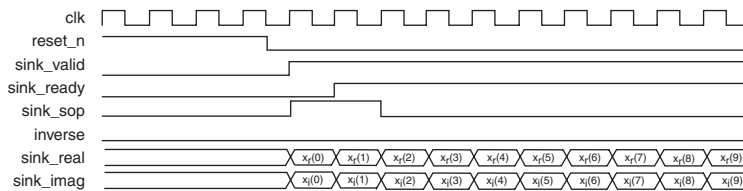
Following the de-assertion of the system reset, the data source asserts `sink_valid` to indicate to the FFT function that valid data is available for input. In response, the FFT asserts `sink_ready` to indicate that it can accept input data.

The data source loads the first complex data sample into the FFT function and simultaneously asserts `sink_sop` to indicate the start of the input block. On the next clock cycle, `sink_sop` is de-asserted and the following $N - 1$ complex input data samples should be loaded in natural order.

When the input block is loaded, the FFT function de-asserts `sink_ready`, indicating that it cannot receive any more data. At this point, the FFT function begins computing the transform on the stored input block.

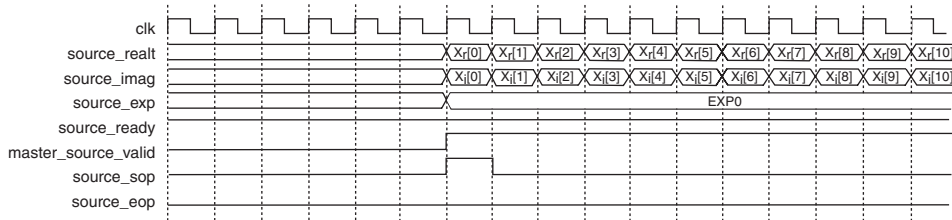
Figure 3–10 shows the input flow control.

Figure 3–10. FFT Buffered Burst Data Flow Architecture Input Flow Control



Following the interval of time where the FFT processor reads the input samples from an internal input buffer, it re-asserts `sink_ready` indicating it is ready to read in the next input block. The beginning of the subsequent input block should be demarcated by the application of a pulse on `sink_sop` aligned in time with the first input sample of the next block.

As in all data flow architectures, the logical level of `inverse` for a particular block is registered by the FFT function at the time of the assertion of the start-of-packet signal, `sink_sop`. When the FFT has completed the transform of the input block, it asserts the `source_valid` and outputs the complex transform domain data block in natural order (see Figure 3–11).

Figure 3–11. FFT Buffered Burst Data Flow Architecture Output Flow Control

Signals `source_sop` and `source_eop` indicate the start-of-packet and end-of-packet for the output block data respectively (see [Figure 3–9](#)).

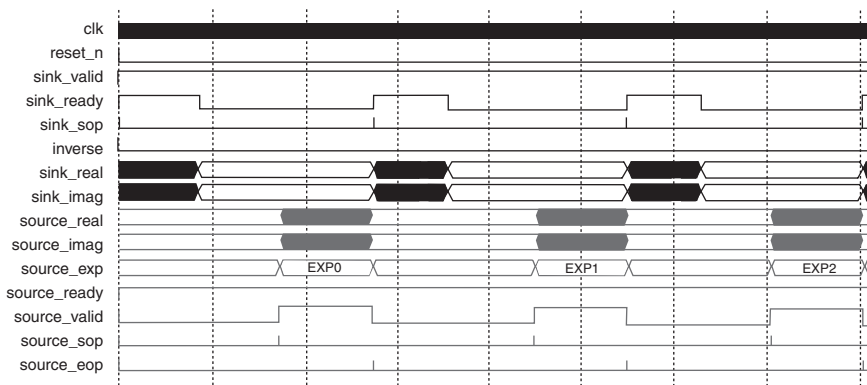


The `sink_valid` signal must be asserted for `source_valid` to be asserted (and a valid data output). You must therefore leave `sink_valid` signal asserted at the end of data transfers to extract the final frames of data from the FFT.

Burst

The burst I/O data flow architecture operates similarly to the buffered burst architecture, except that the burst architecture requires even lower memory resources for a given parameterization at the expense of reduced average throughput.

[Figure 3–12](#) shows the simulation results for the burst architecture. Again, the signals `source_valid` and `sink_ready` indicate, to the system data sources and slave sinks either side of the FFT, when the FFT can accept a new block of data and when a valid output block is available on the FFT output.

Figure 3–12. FFT Burst Data Flow Architecture Simulation Waveform

In a burst I/O data flow architecture, following the loading of a valid input block, `sink_ready` is de-asserted until the FFT function has completed the transform and has unloaded the complete output data block. Only at this point, is `sink_ready` re-asserted to enable the loading of the next input block.



The `sink_valid` signal must be asserted for `source_valid` to be asserted (and a valid data output). You must therefore leave `sink_valid` signal asserted at the end of data transfers to extract the final frames of data from the FFT.

Parameters

Table 3–2 shows the FFT MegaCore function's parameters.

Table 3–2. Parameters (Part 1 of 3)		
Parameter	Value	Description
Target device family	<device family>	Choose the target device family. The device family must be the same as your Quartus II project device family.
Transform length	64, 128, 256, 512, 1,024, 2,048, 4096, 8192, 16,384	The transform length. For variable streaming, this value is the maximum FFT length.
Data precision	8, 10, 12, 14, 16, 18, 20, 24	The data precision.
Twiddle precision	8, 10, 12, 14, 16, 18, 20, 24	The twiddle precision. Twiddle factor precision must be less than or equal to data precision.

Table 3–2. Parameters (Part 2 of 3)

Parameter	Value	Description
FFT engine architecture	Quad output, single output	For both the buffered burst and burst I/O data flow architectures, you can choose between one, two, and four Quad-Output FFT engines working in parallel. Alternatively, if you have selected a single-output FFT engine architecture, you may choose to implement one or two engines in parallel. Multiple parallel engines reduce the FFT MegaCore function's transform time at the expense of device resources—which allows you to select the desired area and throughput trade-off point.
Number of parallel FFT engines	1, 2, 4	For more information on device resource and transform time trade-offs, refer to “Parameters” on page 3–15 . Not available for variable streaming.
I/O data flow	Streaming Variable streaming Buffered burst Burst	Choose the FFT architecture.
Structure	3 mults/5 adders 4 mults/2 adders	You can implement the complex multiplier structure with four real multipliers and two adders/subtractors, or three multipliers, five adders, and some additional delay elements. In Stratix® III, Stratix II, Stratix GX, and Stratix devices use four multipliers and two adders to maximize DSP block usage and minimize logic element (LE) usage. Not available for variable streaming.
Implement multipliers in	DSP block/logic cells Logic cells only DSP blocks only	Each real multiplication can be implemented in DSP blocks or LEs only, or using a combination of both. If you use a combination of DSP blocks and LEs, the FFT MegaCore function automatically extends the DSP block 18 × 18 multiplier resources with LEs as needed. Not valid for variable streaming.
Global clock enable	On or off	Turn on Global Clock Enable , if you want to add a global clock enable to your design.

Table 3–2. Parameters (Part 3 of 3)

Parameter	Value	Description
Twiddle ROM distribution	100% M4K to 100% M512 or 100% M9K to 100% MLAB	<p>High-throughput FFT parameterizations can require multiple shallow ROMs for twiddle factor storage. If your target device family supports M512 RAM blocks (or MLAB blocks in Stratix III devices), you can choose to distribute the ROM storage requirement between M4K (M9K) RAM and M512 (MLAB) RAM blocks by adjusting the Twiddle ROM Resource Distribution slider bar. Set the slider bar to the far left to implement the ROM storage completely in M4K (M9K) RAM blocks; set the slider bar to the far right to implement the ROM completely in M512 (MLAB) RAM blocks.</p> <p>Implementing twiddle ROM in M512 (MLAB) RAM blocks can lead to a more efficient device internal memory bit usage. Alternatively, this option can be used to conserve M4K (M9K) RAM blocks used for the storage of FFT data or other storage requirements in your system.</p> <p>Not available for variable streaming.</p>
Use M-RAM or M144K blocks	On or off	<p>Implements suitable data RAM blocks within the FFT MegaCore function in M-RAM (M144K in Stratix III devices) to reduce M4K (M9K) RAM block usage, in device families that support M-RAM blocks.</p> <p>Not available for variable streaming.</p>
Implement appropriate logic functions in RAM	On or off	<p>Uses embedded RAM blocks to implement internal logic functions, for example, tapped delay lines in the FFT MegaCore function. This option reduces the overall LE count.</p> <p>Not available for variable streaming.</p>

Signals

Table 3–3 shows the Avalon-ST interface signals.



For more information on the Avalon-ST interface, refer to the *Avalon Streaming Interface Specification*.

Table 3–3. Avalon-ST Signals (Part 1 of 2)

Signal Name	Direction	Avalon-ST Type	Size	Description
clk	Input	clk	1	Clock signal that clocks all internal FFT engine components.
reset_n	Input	reset_n	1	Active-low reset signal.
sink_eop	Input	endofpacket	1	Indicates the end of the incoming FFT frame.
sink_error	Input	error	2	Indicates an error has occurred in an upstream module, because of an illegal usage of the Avalon-ST protocol. The following errors are defined (see Table 3–5): <ul style="list-style-type: none"> • 00 = no error • 01 = missing SOP • 10 = missing EOP • 11 = unexpected EOP
sink_imag	Input	data	data precision width	Imaginary input data, which represents a signed number of data precision bits.
sink_ready	Output	ready	1	Asserted by the FFT engine when it is able to accept data.
sink_real	Input	data	data precision width	Real input data, which represents a signed number of data precision bits.
sink_sop	Input	startofpacket	1	Indicates the start of the incoming FFT frame.
sink_valid	Input	valid	1	Asserted when data on the data bus is valid. When <code>sink_valid</code> and <code>sink_ready</code> are asserted, a data transfer takes place. See “Enabling the FFT” on page 3–11 .
source_eop	Output	endofpacket	1	Marks the end of the outgoing FFT frame.

Table 3–3. Avalon-ST Signals (Part 2 of 2)

Signal Name	Direction	Avalon-ST Type	Size	Description
source_error	Output	error	2	Indicates an error has occurred either in an upstream module or within the FFT module (logical OR of sink_error with errors generated in the FFT. The following errors are defined (see Table 3–5): <ul style="list-style-type: none"> • 00 = no error • 01 = missing SOP • 10 = unexpected EOP • 11 = other error
source_exp	Output	data	6	Streaming burst and buffered burst architectures only. Signed block exponent: Accounts for scaling of internal signal values during FFT computation.
source_imag	Output	data	(data precision width + growth) (1)	Imaginary output data. For burst, buffered burst, and streaming FFTs, the output data width is equal to the input data width. For variable streaming FFTs, the size of the output data is dependent on the number of stages defined for the FFT and is approximately 2.5 bits per radix 2^2 stage.
source_ready	Input	ready	1	Asserted by the downstream module if it is able to accept data.
source_real	Output	data	(data precision width + growth) (1)	Real output data. For burst, buffered burst, and streaming FFTs, the output data width is equal to the input data width. For variable streaming FFTs, the size of the output data is dependent on the number of stages defined for the FFT and is approximately 2.5 bits per radix 2^2 stage.
source_sop	Output	startofpacket	1	Marks the start of the outgoing FFT frame.
source_valid	Output	valid	1	Asserted by the FFT when there is valid data to output.

Note to Table 3–3:

(1) Variable streaming FFT only. Growth is $2.5 \times (\text{number of stages}) = 2.5 \times (\log_4(\text{MAX}(\text{fftppts})))$

Table 3–4 shows the component specific signals.

Table 3–4. Component Specific Signals			
Signal Name	Direction	Size	Description
<code>fftpts</code>	Input	$\log_2(\text{maximum number of points})$	The number of points in this FFT frame. If this value is not specified, the FFT can not be a variable length. The default behavior is for the FFT to have fixed length of maximum points. Only sampled at SOP.
<code>inverse</code>	Input	1	Inverse FFT calculated if asserted. Only sampled at SOP.
<code>clk_ena</code>	Input	1	Active-high global clock enable input. When de-asserted, the FFT is disabled.

Incorrect usage of the Avalon-ST interface protocol on the sink interface results in a error on `source_error`. Table 3–5 defines the behavior of the FFT when an incorrect Avalon-ST transfer is detected. If an error occurs, the behavior of the FFT is undefined and you must reset the FFT with `reset_n`.

Table 3–5. Error Handling Behavior		
Error	source_error	Description
Missing SOP	01	At the start of an FFT frame, <code>sink_valid</code> is asserted and <code>sink_sop</code> is de-asserted.
Missing EOP	10	At the start of an FFT frame, <code>sink_valid</code> is asserted and <code>sink_eop</code> is de-asserted.
Unexpected EOP	11	An EOP is detected at an incorrect position in the frame (before the FFT has received <code>fftpts</code> data transfers)

This appendix discusses exponent scaling values for FFT/IFFT.

Calculating Possible Exponent Values

Depending upon the length of the FFT/IFFT the number of passes through the butterfly are known and therefore the range for the exponent. The possible values of the exponent are determined by the following formulas:

$$P = \text{ceil}\{\log_4(N)\} - 1 \text{ where } N \text{ is the transform length}$$

$$R = 0 \text{ if } \log_2 N \text{ is even, otherwise } R = 1.$$

$$\text{Single output range} = (-3P - 2 + R, P - 2 + R)$$

$$\text{Quad output range} = (-3P - 2 + R, P - 2 + R - 4)$$

Table A-1 shows these values for a variety of FFT lengths.

Table A-1. Exponent Scaling Values for FFT/IFF						
FFT Length	P	R	Single Output		Quad Output	
			Max	Min	Max	Min
64	2	0	-8	0	-8	-4
128	3	1	-10	2	-10	-2
256	3	0	-11	1	-11	-3
512	4	1	-13	3	-13	-1
1024	4	0	-14	2	-14	-2
2048	5	1	-16	4	-16	-0

For quad engine output FFT/IFFTs you must shift the data by $2^{-\text{exponent}}$ to fully normalize the gain relative to the input levels. For single engine output FFT/IFFTs, there is an additional factor of Q and the shift must be $2^{-\text{exponent} - Q}$, where $Q = 1$ for an even power of 2 and $Q = 2$ for an odd power of 2.

Implementing Scaling

In implementing the scaling algorithm, determine the length of the resulting full scale dynamic range storage register by adding the width of the data to the amount of maximum value in [Table A-1](#).

For example for a 16-bit data, 256-point quad output FFT/IFFT (max = -11, min = -3), the resulting full scaled data width is $16 + 11 = 27$ bits. You must then map the output data to the appropriate location within the expanded dynamic range register, based upon the exponent value. For the preceding example, the 16-bit output data [15:0] from the FFT/IFFT is mapped to [26:11] for an exponent of -11; [25:10] for an exponent of -10; [24:9] for an exponent of -9... You must also properly sign extend the data within the full scale register. [Figure A-1](#) shows a partial sample of Verilog HDL code that illustrates the scaling of output data (for exponents -11 to -9) with sign extension.

Figure A-1. Sample Verilog HDL Scaling of Output Data Code

```
case (exp)
  6'b110101 : //-11 Set date equal to MSBs
    begin
      full_range_real_out[26:0] <= {real_in[15:0],11'b0};
      full_range_imag_out[26:0] <= {imag_in[15:0],11'b0};
    end
  6'b110110 : //-10 Equals shift by 1 with sign extension
    begin
      full_range_real_out[26]   <= {real_in[15]};
      full_range_real_out[25:0] <= {real_in[15:0],10'b0};
      full_range_imag_out[26]   <= {imag_in[15]};
      full_range_imag_out[25:0] <= {imag_in[15:0],10'b0};
    end
  6'b110111 : //-9 Equals shift by 2 with sign extension
    begin
      full_range_real_out[26:25] <= {real_in[15],real_in[15]};
      full_range_real_out[24:0]  <= {real_in[15:0],9'b0};
      full_range_imag_out[26:25] <= {mag_in[15],imag_in[15]};
      full_range_imag_out[24:0]  <= {imag_in[15:0],9'b0};
    end
end
```

For this example the output provides a full scale 27-bit word. You must choose how many and which bits should be carried forward in the processing chain. The choice of bits determines the absolute gain relative to the input sample level. [Figure A-2](#) provides an illustration of scaling for all possible values for an input signal level of 5000H is assumed for the 256-point quad output FFT. The output of the FFT is 280H with exponent = -5. [Figure A-2](#) shows all cases of valid exponent values of scaling to the full scale storage register [26:0]. Because the exponent was

–5, look at the register values for that column. This data has been moved to the last two columns in the figure. The last column represents the gain compensated data after the scaling (0005000H) that agrees with input data as expected. To keep 16 bits for subsequent processing, choose the bottom 16 bits, which results in 5000H. However, if you choose a different bit range, for example, the top 16 bits, the result is 000AH. Therefore, the choice of bits influences the relative gain through the processing chain.

Figure A–2. Scaling of Input Data Sample = 50000

Bit	Input	Output Data	Exponent											Looking at Exponent = -5	
			+11	+10	+9	+8	+7	+6	+5	+4	+3			Taking All Bits	Sign Extend / Pad
	5000H	280 H													
26			0												0
25			0	0											0
24			0	0	0										0
23			0	0	0	0									0
22			0	0	0	0	0								0
21			0	0	0	0	0	0							0
20			1	0	0	0	0	0	0					0	0
19			0	1	0	0	0	0	0	0				0	0
18			1	0	1	0	0	0	0	0	0			0	0
17			0	1	0	1	0	0	0	0	0	0		0	0
16			0	0	1	0	1	0	0	0	0	0		0	0
15	0	0	0	0	0	1	0	1	0	0	0	0		0	0
14	1	0	0	0	0	0	1	0	1	0	0	0	1	1	1
13	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0
12	1	0	0	0	0	0	0	0	1	0	1	0	1	1	1
11	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
9	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Because this example has 27-bit full scale resolution and 16-bit output resolution, choose the bottom 16 bits to maintain unity gain relative to the input signal. Choosing the LSBs is not the only solution or the correct one for all cases. This solution depends upon which signal levels are important. One way to select the proper range is empirically by simulating test cases that implement expected system data. The output of the simulations should tell what range of bits to use as the output register. If the full scale data is not used (or the top MSBs), you must saturate the data to avoid wrap around problems.



Appendix B. Upgrading from v2.2.1 to v6.1

This appendix discusses the differences between FFT MegaCore® function v2.2.1 and v6.1.

Table B-1 shows the new and the old signal names.

Table B-1. New & Old Signal Names	
New Name	Old Name
inverse	inv_i
reset_n	reset
sink_eop	-
sink_error	-
sink_imag	data_imag_in
sink_ready	master_sink_ena
sink_real	data_real_in
sink_sop	master_sink_sop
sink_valid	master_sink_dav
source_eop	master_source_eop
source_error	-
source_exp	exponent_out
source_imag	fft_imag_out
source_ready	master_source_dav
source_real	fft_real_out
source_sop	master_source_sop
source_valid	master_source_ena

