FFT

MegaCore Function User Guide

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

MegaCore Function Version:	2.1.0
Document Version:	2.1.0 rev.1
Document Date:	June 2004

I.S. EN ISO 9001

Part Number: UG-FFT-1.5

Contents

Chapter 3. Specifications

About this User Guide

Revision History

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

Chapter	Date	Version	Changes Made
1	June 2004	2.1.0	● Updated release information and device family support tables ● Updated the features ● Added OpenCore® Plus description ● Added DSP Builder support information ● Updated the performance information ● Enhancements include support for Cyclone™ II devices; bit-accurate MATLAB models; DSP Builder ready
2	June 2004	2.1.0	● 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	● 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	**lit_req@altera.com**	**lit_req@altera.com**
Non-technical customer service	(800) 767-3753 (7:00 a.m. to 5:00 p.m. Pacific Time)	Contact your local Altera sales office or sales representative.
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.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75: High-Speed Board Design.*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: *<file name>*, *<project name>***.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
“Subheading Title”	References to sections within a document and titles of 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 (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., 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.
⚠ CAUTION	The caution indicates required information that needs special consideration and understanding and should be read prior to starting or continuing with the procedure or process.
⚠	The warning indicates information that should be read prior to starting or continuing the procedure or processes
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

Chapter 1. About this MegaCore Function

Release Information

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

Table 1–1. Product Name Release Information

Item	Description
Version	2.1.0
Release Date	June 2004
Ordering Code	IP-FFT
Product ID(s)	0034
Vendor ID(s)	6AF7

Device Family Support

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

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

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

Table 1–2. Device Family Support

Device Family	Support
Stratix® II	Full
Stratix GX	Full
Stratix	Full
Cyclone™ II	Full
Cyclone	Full
HardCopy®	Full
Other device families	No support

Introduction

The FFT MegaCore function, version 2.1.0, is a high performance, highly-parameterizable Fast Fourier transform (FFT) processor, optimized for the Altera Stratix II, Stratix GX, Stratix, Cyclone II, and Cyclone device families. The FFT MegaCore function implements a complex FFT or inverse FFT (IFFT) for high-performance applications.

New in Version 2.1.0

- Support for Cyclone II devices
- Bit-accurate MATLAB models
- DSP Builder ready

Features

- Radix-4 and mixed Radix-4/2 implementations
- Block floating-point architecture—maintain the maximum dynamic range of data during processing
- Uses embedded memory
- Maximum system clock frequency >300 MHz
- Optimized to use Stratix II, Stratix GX, and Stratix DSP blocks and TriMatrix™ memory architecture
- Support for Windows, Solaris, and Linux operating systems
- 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
- Atlantic™ 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, version 2.1.0, is a high performance, highly-parameterizable FFT processor, optimized for the Altera Stratix II, Stratix GX, Stratix, Cyclone II, and Cyclone device families. The FFT function implements a Radix-2/4 decimation-in-frequency (DIF) FFT algorithm for transform lengths of 2^m where $6 \le m \le 14$. Internally, the FFT uses block-floating point representations to achieve the best trade-off between maximum SNR and minimum size requirements.

The FFT MegaCore function 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.

OpenCore Plus Evaluation

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

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

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

For more information on OpenCore Plus hardware evaluation using the FFT Compiler, see "OpenCore Plus Time-Out Behavior" on page 3–15 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 v2.2.0 or higher and the Quartus II sofware version 4.1 or higher.

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

Performance

Because performance varies depending on the FFT engine architecture and I/O data flow, all performance results cannot be represented here. Thus, most of the performance and device utilization results are listed in "Performance & Resource Utilization" on page 3–10. Tables 1–3 and 1–4 only list the streaming data flow performance for Stratix II and Stratix devices.

Table 1–3. Stratix II Device Performance Using the Streaming Data Flow Engine Architecture

Device	Points	Width *(1)*	LEs *(2)*	18*18 Mults	M4K *(3)*	MRAM	f_{MAX} (MHz)	Clock Cycle Count	Transform Time (us)
EP2S15F484C3	256	16	4360	9	19	0	335.23	256	0.76
EP2S15F484C3	512	16	4826	9	19	0	319.19	512	1.6
EP2S15F484C3	1024	16	5137	9	38	0	325.73	1,024	3.14
EP2S15F484C3	2048	16	6946	18	75/44	0/2	309.89	2,048	6.61
EP2S30F484C3	4096	16	7429	18	144/88	0/2	301.84	4,096	13.57
EP2S60F484C3	8192	16	7351	18	304/176	0/2	293.06	8,192	27.95

Notes to Table 1–3:
(1) Represents data and twiddle factor precision.
(2) The Quartus® II software reports the number of adaptive look-up tables (ALUTs) that the design uses in Stratix II devices. The logic element (LE) count is based on this number of ALUTs.
(3) For cases in which M-RAM utilization is permitted, the first number indicates the number of M4K RAM blocks when M-RAM usage is disabled from within the MegaWizard® Plug-In.

Table 1–4 lists Stratix device performance.

Table 1–4. Stratix Device Performance Using the Streaming Data Flow Engine Architecture

Device	Points	Width (1)	LEs	18*18 Mults	M4K (2)	MRAM	f_{MAX} (MHz)	Clock Cycle Count	Transform Time (us)
1S20F780C5	256	16	4,309	9	19	0	241.11	256	1.06
1S20F780C5	512	16	4964	9	19	0	255.61	512	2.0
1S20F780C5	1024	16	5,068	9	38	0	254.25	1,024	4.03
1S20F780C5	2048	16	6,961	18	75/44	0/2	238.51	2,048	8.58
1S40F780C5	4096	16	7,369	18	144/88	0/2	242.07	4,096	16.92
1S40F780C5	8192	16	7,159	18	304/176	0/2	226.04	8,192	36.24

Notes to Table 1–4:
(1) Represents data and twiddle factor precision.
(2) For cases in which M-RAM utilization is permitted, the first number indicates the number of M4K RAM blocks when M-RAM usage is disabled from within the MegaWizard® Plug-In.

Chapter 2. Getting Started

System Requirements

The instructions in this section require the following hardware and software:

- A PC running the Windows NT/2000/XP, Red Hat Linux 7.3 or 8.0, or Red Hat Enterprise Linux 3.0 operating system; or a Sun workstation running the Solaris 7 or 8 operating system
- The Quartus® II software version 4.1
- An Altera®-supported VHDL or Verilog HDL simulator (optional).

Design Flow

To evaluate the FFT MegaCore® function using the OpenCore® Plus feature, the design flow involves the following steps:

1. Obtain and install the FFT MegaCore function.

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

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

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 white paper entitled *Using IP Functional Simulation Models to Verify Your Design*.

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

   You can 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 Chapter 3, OpenCore Plus Time-Out Behavior, and *AN 320: OpenCore Plus Evaluation of Megafunctions*.

6. Purchase a license for the FFT MegaCore function.

Once you have purchased a license for the FFT, the design flow involves the following additional steps:

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

Obtain & Install the FFT

To begin using the FFT MegaCore function, you must obtain the FFT and install it on your computer. Altera MegaCore functions can be installed from the MegaCore IP Library CD-ROM either during or after Quartus II installation, or downloaded individually from the Altera web site and installed separately.

The following instructions describe the process of downloading and installing the FFT. If you have already installed the FFT from the MegaCore IP Library CD-ROM, skip to "Directory Structure" on page 2–4.

Download the FFT MegaCore Function

If you have Internet access, you can download the FFT MegaCore function from Altera's web site at **www.altera.com**. Follow the instructions below to obtain the FFT via the Internet. If you do not have Internet access, contact your local Altera representative to obtain the MegaCore IP Library CD-ROM.

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

1. Point your web browser to **www.altera.com/ipmegastore**.
2. Type `FFT` in the **IP MegaSearch** box.
3. Click **Go**.

4. Choose **FFT** from the search results page. The product description web page displays.

5. Click **Download Free Evaluation**.

6. Complete the registration form and click **Submit Request**.

7. Read the Altera MegaCore license agreement, turn on the **I have read the license agreement** check box, and click **Proceed to Download Page**.

8. Follow the instructions on the FFT download and installation page to download the MegaCore function.

Install the FFT MegaCore Function Files

The following instructions describe how you install the FFT on computers running the Windows, Solaris, or Linux operating systems.

Windows

To install the FFT on a PC running the Windows operating system, follow these steps:

1. Choose **Run** (Start menu).

2. Type <*path*>\fft-v2.1.0`.exe` ↵

   where <*path*> is the location of the downloaded MegaCore function, and <*version*> is the version of the MegaCore function.

3. Click **OK**. The **FFT Installation** dialog box appears. Follow the on-screen instructions to finish installation.

Solaris

To install the FFT on a computer running the Solaris operating system, follow these steps:

1. Decompress the package by typing the following command:

```
gunzip fft-v2.1.0_solaris.tar.gz ↵
```

2. Extract the package contents by typing the following command:

```
tar vxf fft-v2.1.0_solaris.tar ↵
```

Linux

To install the FFT on a computer running the Linux operating system, follow these steps:

1. Decompress the package by typing the following command:

```
gunzip fft-v2.1.0_linux.tar.gz ↵
```

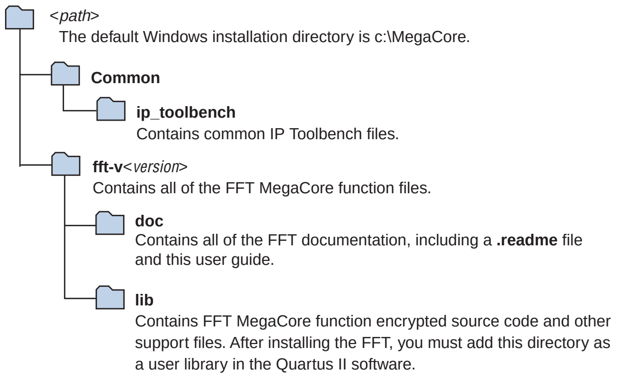
2. Extract the package contents by typing the following command:

```
tar vxf fft-v2.1.0_linux.tar ↵
```

Directory Structure

Figure 2–1 shows the directory structure for the FFT MegaCore function where <*path*> is the installation directory.

Figure 2–1. FFT MegaCore Function Directory Structure



<*path*>
The default Windows installation directory is c:\MegaCore.
Common
ip_toolbench
Contains common IP Toolbench files.
fft-v<*version*>
Contains all of the FFT MegaCore function files.
doc
Contains all of the FFT documentation, including a **.readme** file and this user guide.
lib
Contains FFT MegaCore function encrypted source code and other support files. After installing the FFT, you must add this directory as a user library in the Quartus II software.

FFT Tutorial

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

☞ IP Toolbench only allows you to select legal combinations of parameters, and warns you of any invalid configurations.

This tutorial consists of the following steps:

- "Create a New Quartus II Project" on page 2–5
- "Launch IP Toolbench" on page 2–6
- "Step 1: Parameterize" on page 2–8
- "Step 2: Set Up Simulation" on page 2–12
- "Step 3: Generate" on page 2–14

Create a New Quartus II Project

Before you begin, you must create a new Quartus II project. With the **New Project** wizard, you specify the working directory for the project, assign the project name, and designate the name of the top-level design entity. You must also specify the FFT MegaCore function user library. To create a new project, follow these steps:

1. Choose **Programs >Altera > Quartus II** <*version*> (Windows Start menu) to run the Quartus II software.

2. Choose **New Project Wizard** (File menu).

3. Click **Next** in the introduction (the introduction will not display if you turned it off previously).

4. Specify the working directory for your project. This tutorial uses the directory **c:\altera\qdesigns41\fft**.

5. Specify the name of the project. This tutorial uses **fft_example**.

6. Click **Next**.

7. Click **User Library Pathnames**.

8. Type <*path*>`\fft-v`<*version*>`\lib\` into the **Library Name** box, where <*path*> is the directory in which you installed the FFT MegaCore. The default Windows installation directory is **c:\MegaCore**.

9. Click **Add**.

10. Click **OK**.

11. Click **Next**.

12. Click **Next**.

13. In the **Family** list, select **Stratix® II**. Under **Do you want to select a specific device?** select **No, I want to allow the compiler to choose a device**.

14. Click **Finish**.

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 is displayed.

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

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

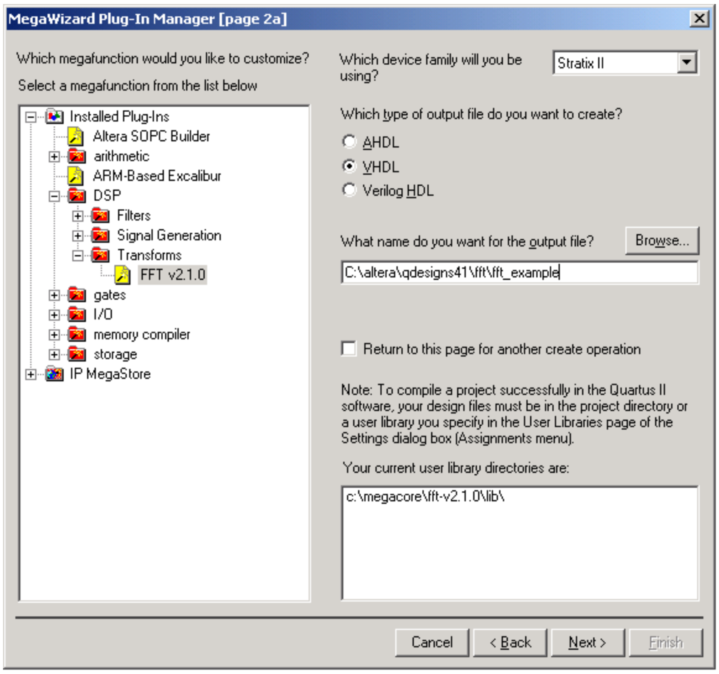
3. Expand the **DSP** > **Transforms** directory under **Installed Plug-Ins** by clicking the **+** icon next to the name.

4. Select **FFT v2.1.0**.

5. Choose the output file type for your design; the wizard supports AHDL, VHDL, and Verilog HDL. This tutorial uses VHDL.

Figure 2–2 shows the wizard after you have made these settings.

Figure 2–2. The MegaWizard Plug-In Manager



Which megafunction would you like to customize?

Select a megafunction from the list below

Which device family will you be using? Stratix II

Which type of output file do you want to create?

( ) AHDL
(●) VHDL
( ) Verilog HDL

What name do you want for the output file? C:\altera\qdesigns41\fft\fft_example

[ ] Return to this page for another create operation

Note: To compile a project successfully in the Quartus II software, your design files must be in the project directory or a user library you specify in the User Libraries page of the Settings dialog box (Assignments menu).

Your current user library directories are:

c:\megacore\fft-v2.1.0\lib\

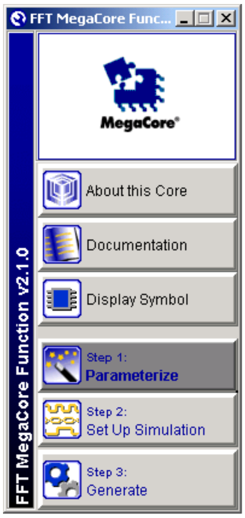
6. Click **Next** to launch IP Toolbench for the FFT MegaCore function.

Step 1: Parameterize

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

1. Click the **Step 1: Parameterize** button in IP Toolbench (see Figure 2–3).

Figure 2–3. IP Toolbench



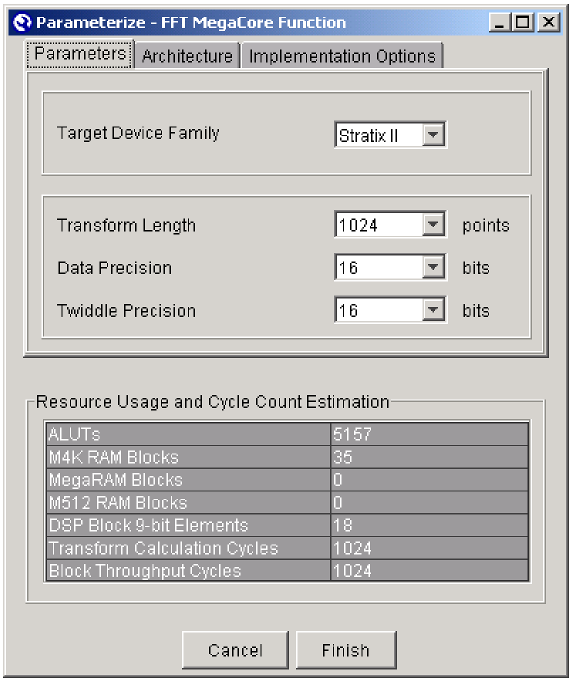
The **Parameterize - FFT MegaCore Function** window opens to the **Parameters** tab. See Figure 2–4.

2. Scroll to the target device family from the **Target Device Family** list.

3. Choose the **Transform Length**, **Data Precision**, and **Twiddle Precision** from the corresponding drop-down boxes.

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

Figure 2–4. Parameters Tab



Target Device Family: Stratix II

Transform Length	1024	points
Data Precision	16	bits
Twiddle Precision	16	bits

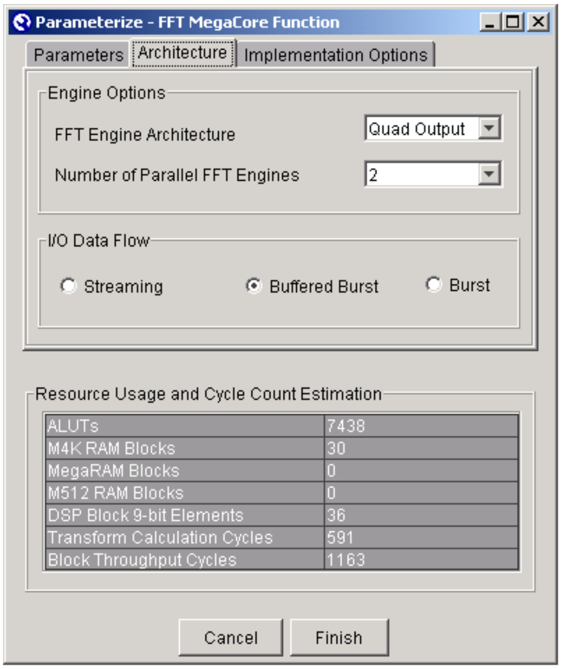
Resource Usage and Cycle Count Estimation

ALUTs	5157
M4K RAM Blocks	35
MegaRAM Blocks	0
M512 RAM Blocks	0
DSP Block 9-bit Elements	18
Transform Calculation Cycles	1024
Block Throughput Cycles	1024

4. Select the **Architecture** tab when you are finished setting the FFT MegaCore function parameters.

5. From the **Architecture** tab, select **FFT Engine Architecture**, **Number of Parallel FFT Engines**, and **I/O Data Flow**. See Figure 2–5.

Figure 2–5. Architecture Tab



Engine Options

FFT Engine Architecture: Quad Output

Number of Parallel FFT Engines: 2

I/O Data Flow

Streaming [✓ Buffered Burst] Burst

Resource Usage and Cycle Count Estimation

ALUTs	7438
M4K RAM Blocks	30
MegaRAM Blocks	0
M512 RAM Blocks	0
DSP Block 9-bit Elements	36
Transform Calculation Cycles	591
Block Throughput Cycles	1163

If you select a **Quad-Output** FFT engine architecture with a **Streaming** I/O data flow, the FFT MegaCore function generates a design with the minimum number of engines to meet the required I/O throughput.

A single FFT engine architecture provides enough performance for up to a 1024-point streaming I/O data flow FFT.

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, then 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—allowing you to select the desired area and throughput trade-off point.

For more information on device resource and transform time trade-offs, refer to “Performance & Resource Utilization” on page 3–10.

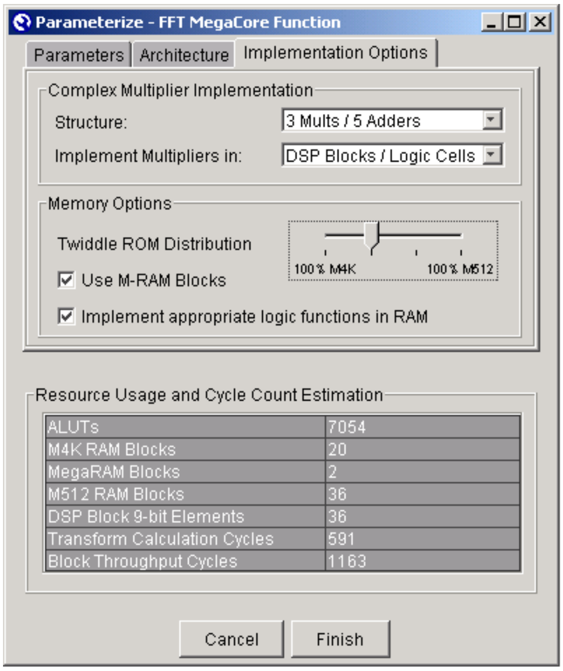
6. Select the **Implementation Options** tab when you are finished setting the FFT architecture parameters.

From the **Implementation Options** tab (see Figure 2–6):

7. Choose the desired complex multiplier structure**.**

The complex multiplier can be implemented using four real multiplications and two additions/subtractions, or using three multiplications, five additions and some additional delay elements. In Stratix II, Stratix GX and Stratix devices selecting four multipliers and two additions will maximize DSP block usage and minimize logic element (LE) usage.

Figure 2–6. Implementation Options Tab



Complex Multiplier Implementation

Structure: 3 Mults / 5 Adders

Implement Multipliers in: DSP Blocks / Logic Cells

Memory Options

Twiddle ROM Distribution (100% M4K – 100% M512)

[✓] Use M-RAM Blocks

[✓] Implement appropriate logic functions in RAM

Resource Usage and Cycle Count Estimation

ALUTs	7054
M4K RAM Blocks	20
MegaRAM Blocks	2
M512 RAM Blocks	36
DSP Block 9-bit Elements	36
Transform Calculation Cycles	591
Block Throughput Cycles	1163

8. Choose how you want to implement the multipliers.

Each real multiplication can be implemented in DSP blocks or LEs only, or using a combination of both. If you choose the option to use a combination of DSP blocks and LEs, the FFT MegaCore function will automatically extend the DSP block 18 × 18 multiplier resources with LEs as needed.

9. Specify the memory options.

High throughput FFT parameterizations can result in the requirement of multiple shallow ROM’s for twiddle factor storage. If your target device family supports M512 RAM blocks, you can choose to distribute the ROM storage requirement between M4K RAM and M512 RAM blocks as desired by setting the **Twiddle ROM Resource Distribution** slider bar. Setting the slider bar's position to the far left implements the ROM storage completely in M4K RAM blocks. Setting the slider bar to the far right results in the ROM being implemented completely in M512 RAM blocks. Positioning the slider bar at intermediate points will result in the usage of both M4K and M512 RAM blocks to implement the ROM's, with the weight of M512 usage increasing as the slider bar is moved from left to right, and vice versa.

Implementing Twiddle ROM in M512 RAM blocks can lead to a more efficient device internal memory bit usage. Alternatively, this option can be used to conserve M4K RAM blocks used for the storage of FFT data or other storage requirements in your system.

You can choose to implement suitable data RAM blocks within the FFT MegaCore function in M-RAM to reduce M4K RAM block usage—in device families that support M-RAM blocks.

You can also select to use embedded RAM blocks to implement internal logic functions, e.g., tapped delay lines in the FFT MegaCore function. This option has the advantage of reducing the overall LE count.

10. Click **Finish** when the implementation options are set.

Step 2: Set Up Simulation

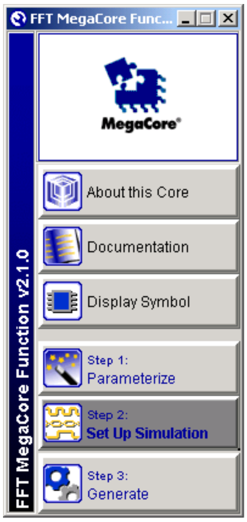
An IP functional simulation model is a cycle-accurate VHDL or Verilog HDL model file produced by the Quartus II software (version 3.0 or higher). It allows for fast functional simulation of IP using industry-standard VHDL and Verilog HDL simulators.

☞ 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 creates a non-functional design.

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

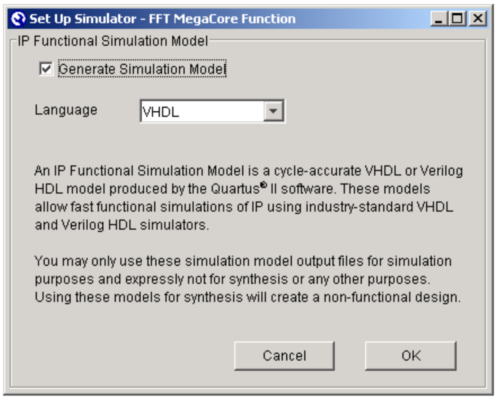
1. Click the **Step 2: Set Up Simulation** button in IP Toolbench (see Figure 2–7).

Figure 2–7. Set Up Simulation



2. Turn on **Generate Simulation Model** (see Figure 2–8).

Figure 2–8. Set Up Simulator


Set Up Simulator - FFT MegaCore Function
IP Functional Simulation Model
Cancel
OK

[✓] Generate Simulation Model

Language: VHDL

An IP Functional Simulation Model is a cycle-accurate VHDL or Verilog HDL model produced by the Quartus® II software. These models allow fast functional simulations 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 non-functional design.

3. Choose the language in the **Language** list.

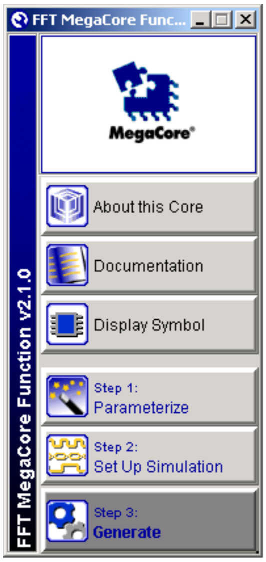
4. Click **OK**.

Step 3: Generate

To generate your MegaCore function, follow these steps:

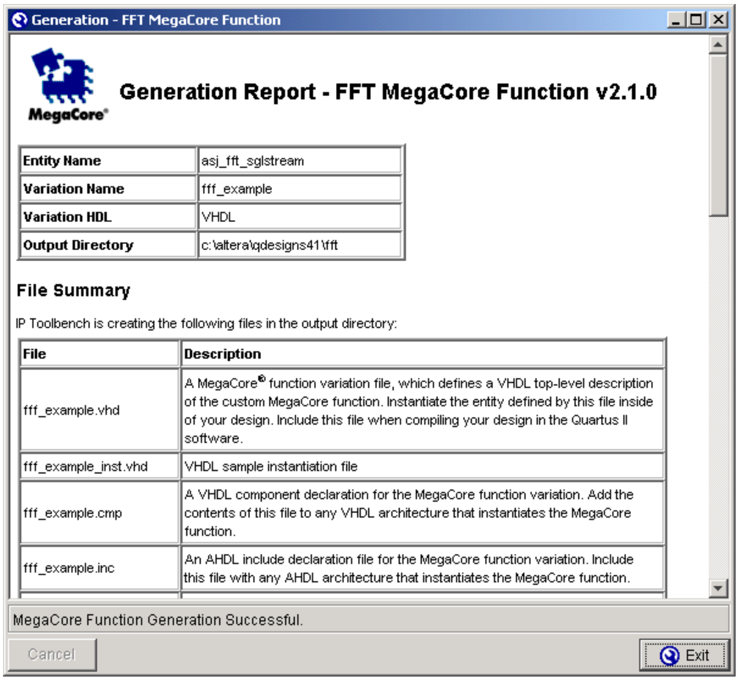
1. Click the **Step 3: Generate** button in IP Toolbench (see Figure 2–9).

Figure 2–9. IP Toolbench—Generation



2. The generation report lists the design files that IP Toolbench creates (see Figure 2–10). Click **Exit**.

Figure 2–10. Generation Report—FFT MegaCore



Generation Report - FFT MegaCore Function v2.1.0

Entity Name	asj_fft_sglstream
Variation Name	fff_example
Variation HDL	VHDL
Output Directory	c:\altera\qdesigns41\fft

File Summary

IP Toolbench is creating the following files in the output directory:

File	Description
fff_example.vhd	A MegaCore® function variation file, which defines a VHDL 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.
fff_example_inst.vhd	VHDL sample instantiation file
fff_example.cmp	A VHDL component declaration for the MegaCore function variation. Add the contents of this file to any VHDL architecture that instantiates the MegaCore function.
fff_example.inc	An AHDL include declaration file for the MegaCore function variation. Include this file with any AHDL architecture that instantiates the MegaCore function.

MegaCore Function Generation Successful.

Table 2–1 describes the standard IP Toolbench-generated files.

Table 2–1. Standard IP Toolbench-Generated Files (Part 1 of 2)

Extension	Description
.vhd, **.v**, or **.tdf**	A MegaCore function variation file that defines a VHDL, Verilog HDL, or AHDL 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.
.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.
.inc	An AHDL include declaration file for the MegaCore function variation. Include this file with any AHDL architecture that instantiates the MegaCore function.
_bb.v	A Verilog HDL black-box file for the MegaCore function variation. Use this file when using a third-party EDA tool to synthesize your design.
.bsf	A Quartus II symbol file for the MegaCore function variation. You can use this file in the Quartus II block diagram editor.

Table 2–1. Standard IP Toolbench-Generated Files (Part 2 of 2)

Extension	Description
.html	A MegaCore function report file.
.vo or **.vho**	A VHDL or Verilog HDL IP functional simulation model.
_inst.vhd or **_inst.v**	A VHDL or Verilog HDL sample instantiation file.

Table 2–2 lists the FFT MegaCore function's output files.

Table 2–2. FFT MegaCore Function's Output Files

Extension	Description
<*variation_name*>**_ tb.vhd**	Architecture-specific VHDL testbench file.
<*variation_name*>**_tb.v**	Architecture-specific Verilog HDL testbench file.
<*variation_name*>**_wave.do**	ModelSim® waveform file to view simulation results verification of the FFT operation in the ModelSim software.
<*variation_name*>**_vho_msim.tcl**	ModelSim TCL script to compile, load, and run VHDL IP functional simulation models.
<*variation_name*>**_vo_msim.tcl**	ModelSim TCL script to compile, load, and run Verilog HDL IP functional simulation models.
<*variation_name*>**_model.m**	MATLAB bit-accurate model.
<variation_name>**_tb.m**	MATLAB test bench to source and sink data to/from text files.

3. After you have reviewed the generation report, click **Exit** to close IP Toolbench and return to the Quartus II software.

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

Simulate the Design

This section provides steps for simulating the design:

- In the MATLAB software
- Using IP functional simulation models
- In ModelSIM

Simulating the Design in the MATLAB Software

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 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.

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.

The wizard also creates the MATLAB test bench 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 the FFT MegaWizard. To model your FFT MegaCore Function variation in the MATLAB software, perform the following steps:

1. Run the MATLAB software.

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

3. Perform the simulation by either:

   a. Typing `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. Running the provided test bench by typing the name of the test bench, *<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 the Design Using 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 2–12. 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 white paper entitled, *Using IP Functional Simulation Models To Verify Your System Design*.

Simulating Your design in the Modelsim Software

The following instructions are ModelSim-specific; however, you can simulate the FFT MegaCore function in any Altera-supported VHDL or Verilog HDL simulator.

To simulate your design with the MegaWizard Plug-In-generated ModelSim TCL script, perform the following steps:

- Change your ModelSim working directory to your set project directory and run the MegaWizard-generated TCL script. See Table 2–2.
  - If you have selected VHDL as your functional simulation language, run the TCL script <*variation_name*>**_vho_msim.tcl**.
  - If you selected Verilog HDL as your functional simulation language, run the TCL script <*variation_name*>**_vo_msim.tcl**.

  The TCL script creates a ModelSim project, maps the libraries, compiles the top-level design and associated testbench, and then outputs the simulation results to the waveform viewer.

Compile the Design

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

To compile your custom FFT MegaCore function variation you must have Quartus II software, version 4.1, installed on your computer.

The Quartus II user libraries for your project must include <*path*>**\fft-v**<*version*>**\lib**, where <*path*> is the directory in which you installed the FFT MegaCore function.

To compile your 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, perform the following 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. Select **Settings** (Assignments menu). The **Settings** dialog box appears.

3. Select **Add/Remove** from the **Files & Directories** box.

4. Browse to the <*path*>**\fft-v2.1.0\lib** directory and choose the VHDL package, **fft_pack.vhd**.

5. Click **Open**.

6. The **fft_pack.vhd** file is now added to your Quartus II project.

☞ Be sure that 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, select **fft_pack.vhd** and click the **Up** button.

7. Choose **Start Compilation** (Processing menu) in Quartus II software.

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 white paper entitled *Using IP Functional Simulation Models to Verify Your System Design*.

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 Chapter 3, OpenCore Plus Time-Out Behavior 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 web site at **www.altera.com/licensing** and install it on your computer. When you request a license file, Altera e-mails you a **license.dat** file. If you do not have Internet access, contact your local Altera representative.

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

Before you set up licensing for the FFT, you must already have the Quartus II software installed and licensed on your computer.

Append the License to Your license.dat File

To append the license, follow these steps:

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

   - Quartus II software
   - MAX+PLUS® II software
   - LeonardoSpectrum synthesis tool
   - Synplify software
   - ModelSim simulator

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

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

4. Copy the `FEATURE` line from the FFT license file and paste it into the Quartus II license file.

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

5. Save the Quartus II license file.

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

Specify the License File in the Quartus II Software

To specify the MegaCore function's license file, follow these steps:

1. Start the Quartus II software.

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

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

4. Type the path and filename of the MegaCore function license file after the semicolon.

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

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

Chapter 3. Specifications

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.

The length-*N* forward DFT of a sequence x(n) is given by Equation 1:

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

The length-*N* inverse DFT is given by Equation 2:

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

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 decompostion 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.

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. An FFT fixed-point implementation that allows for word growth makes either the data width excessive or leads 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 (i.e. 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 $(^{-exponent\_out})$ to account for the discarded LSB's during the transform.

FFT Processor Engine Architectures

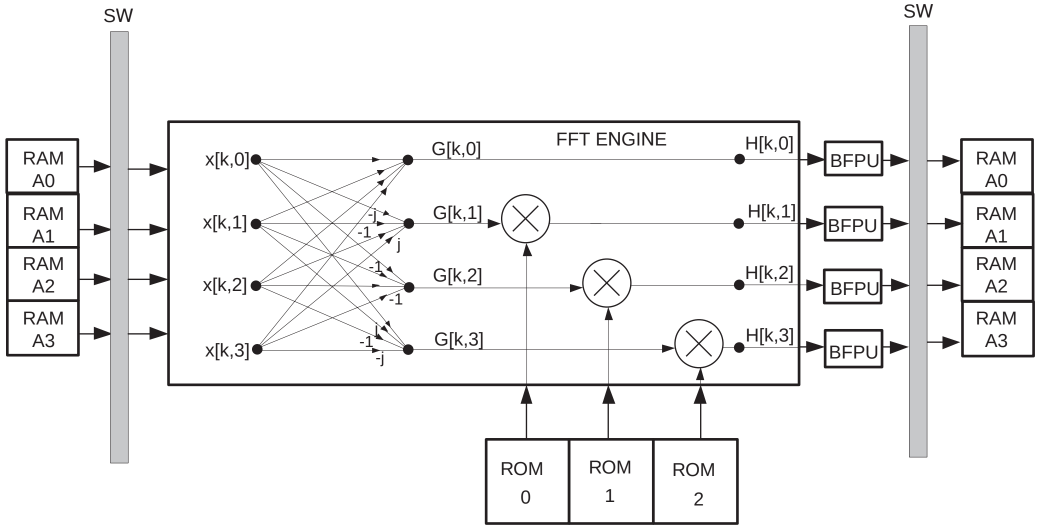
The FFT MegaCore function can be parameterized to use one of two different engine architectures: quad-output or single-output engine architecture. To increase the overall throughput of the FFT MegaCore function, you may also use multiple parallel engines of a variation. This section discusses:

- Quad-output FFT engine architecture
- Single-output FFT engine architecture

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

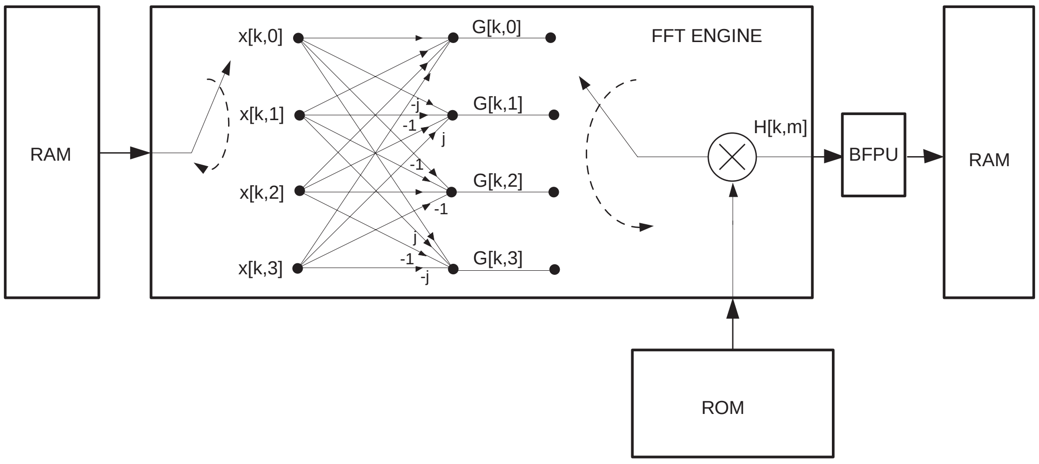


Complex data samples x[k,m] are read from internal memory in parallel and re-ordered by switch (SW). Next, the ordered samples are processed by the Radix-4 butterfly processor to form the complex outputs G[k,m]. Due to the inherent mathematics of the Radix-4 DIF decomposition, only three complex multipliers 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 (BFPU). The appropriate LSB's 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:

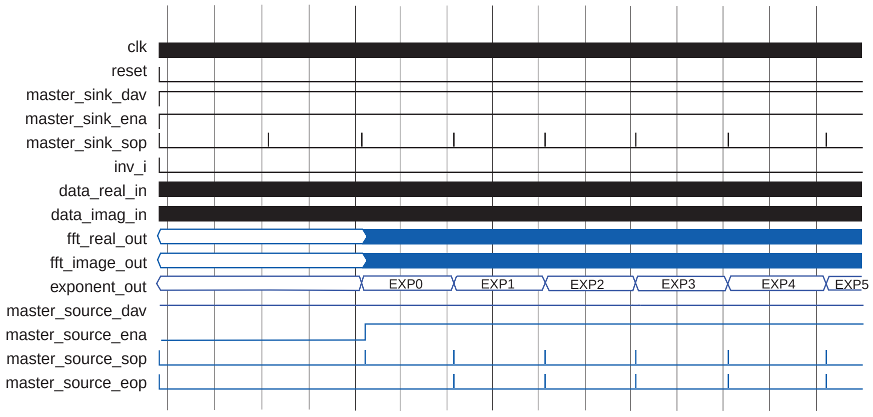
- Streaming
- Buffered Burst
- Burst

For information on setting the architectural parameters in IP Toolbench, refer to Chapter 2, Step 1: Parameterize.

Streaming I/O Data Flow Architecture

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.

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



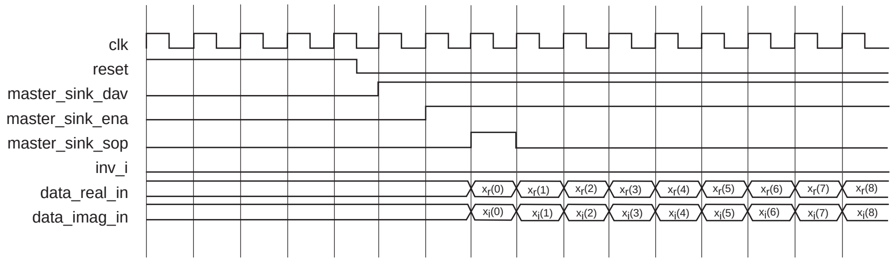
The FFT MegaCore function uses the Altera Atlantic™ interface I/O protocol. The input interface is a master sink, while the output interface is a master source.

For more information on the Atlantic interface protocol, refer to *FS 13: Atlantic Interface.*

Following the de-assertion of the system reset, the data source asserts `master_sink_dav` to indicate to the FFT function that it has at least *N* complex data samples available for input. In response, the FFT function asserts `master_sink_ena` to indicate that it is capable of accepting input data. The data source loads the first complex data sample into the FFT function and simultaneously asserts `master_sink_sop` to indicate the start of the input block.

On the next clock cycle, `master_sink_sop` is de-asserted and the data samples are loaded in natural order. Figure 3–4 illustrates the input flow control in detail.

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



In the streaming data flow architecture, the FFT function expects the input data to be continuously available on the input. Therefore, `master_sink_ena` remains asserted unless the system reset is applied or the slave source indicates the lack of availability of a full block of input data by deasserting `master_sink_dav`.

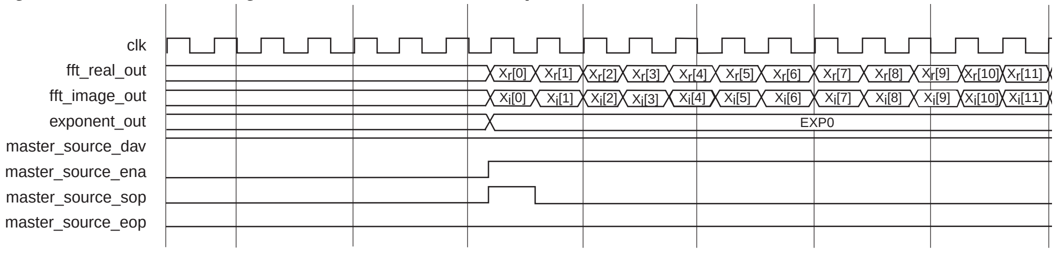
Although not shown in Figure 3–3, if you want to stop the streaming of block data at an input block boundary, hold `master_sink_sop` low following the last input sample of the previous block. The FFT Function continues to process the blocks already loaded and awaits a pulse on `master_sink_sop` before trying to read in the next block. This way, data that is continous sample-wise within a block but bursty in a block-wise sense can be handled with no overhead.

To change direction on a block-by-block basis, `inv_i` should be asserted/de-asserted (appropriately) simultaneously with the application of the `master_sink_sop` pulse (concurrent with the first input data sample of the block).

When the FFT has completed the transform of the input block—and the slave sink has asserted `master_source_dav` (indicating the data slave sink can accept the output data block)—it asserts `master_source_ena` and outputs the complex transform domain data block in natural order.

The FFT function asserts the pulse signal `master_source_sop` to indicate the first output sample, see Figure 3–5. Figure 3–5 illustrates the output flow control in detail.

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

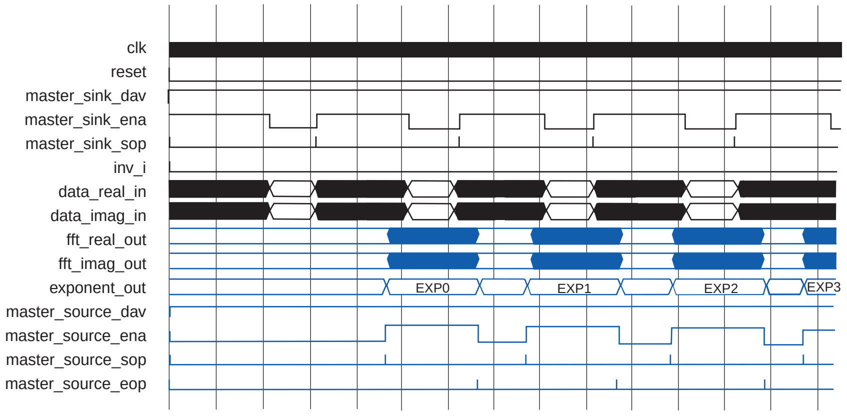

clk
fft_real_out
fft_image_out
exponent_out
master_source_dav
master_source_ena
master_source_sop
master_source_eop
Xr[0] Xr[1] Xr[2] Xr[3] Xr[4] Xr[5] Xr[6] Xr[7] Xr[8] Xr[9] Xr[10] Xr[11]
Xi[0] Xi[1] Xi[2] Xi[3] Xi[4] Xi[5] Xi[6] Xi[7] Xi[8] Xi[9] Xi[10] Xi[11]
EXP0

After *N* clock cycles, `master_source_eop` is asserted to indicate the end of the output data block, as shown in Figure 3–3.

Buffered Burst I/O Data Flow Architecture

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 3–6 shows an example simulation waveform.

Figure 3–6. FFT Buffered Burst Data Flow Architecture Simulation Waveform

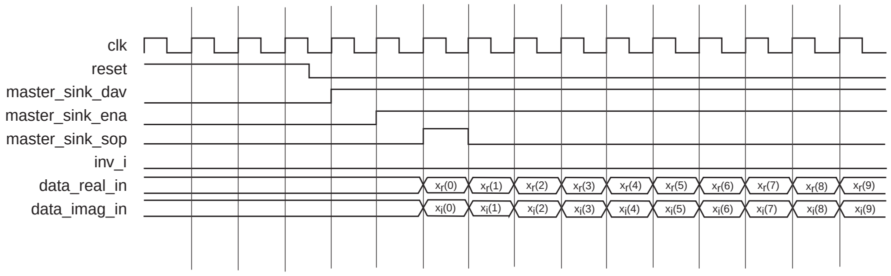


Following the de-assertion of the system reset, the data source asserts `master_sink_dav` to indicate to the FFT function that it has at least *N* complex data samples available for input. In response, the FFT asserts `master_sink_ena` to indicate that it is capable of accepting input data.

The data source loads the first complex data sample into the FFT function and simultaneously asserts `master_sink_sop` to indicate the start of the input block. On the next clock cycle, `master_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 `master_sink_ena`, indicating that it cannot receive any more data. At this point, the FFT function begins computing the transform on the stored input block. Figure 3–7 shows the input flow control in detail.

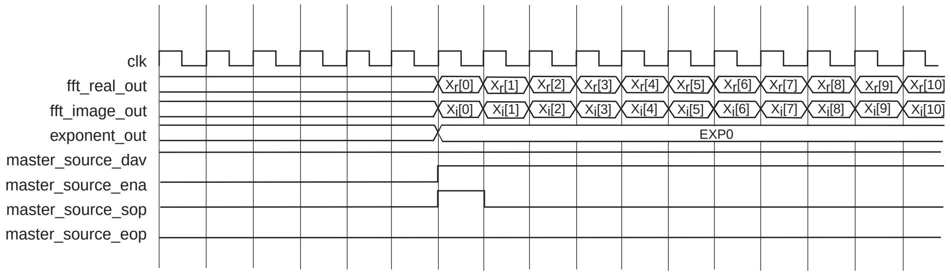
Figure 3–7. FFT Buffered Burst Data Flow Architecture Input Flow Control


clk
reset
master_sink_dav
master_sink_ena
master_sink_sop
inv_i
data_real_in
data_imag_in
$x_r(0)$ $x_r(1)$ $x_r(2)$ $x_r(3)$ $x_r(4)$ $x_r(5)$ $x_r(6)$ $x_r(7)$ $x_r(8)$ $x_r(9)$
$x_i(0)$ $x_i(1)$ $x_i(2)$ $x_i(3)$ $x_i(4)$ $x_i(5)$ $x_i(6)$ $x_i(7)$ $x_i(8)$ $x_i(9)$

Following the interval of time where the FFT processor reads the input samples from an internal input buffer, it re-asserts `master_sink_ena` 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 `master_sink_sop` aligned in time with the first input sample of the next block.

As in all data flow architectures, the logical level of `inv_i` for a particular block is registered by the FFT function at the time of the assertion of the start-of-packet signal, `master_sink_sop`. When the FFT has completed the transform of the input block,—and the slave sink has asserted `master_source_dav` (i.e., indicating the data slave sink can accept the output data block)—it asserts the `master_source_ena` and outputs the complex transform domain data block in natural order. See Figure 3–8.

Figure 3–8. FFT Buffered Burst Data Flow Architecture Output Flow Control



Signals master_source_sop and master_source_eop indicate the start-of-packet and end-of-packet for the output block data respectively, as shown in Figure 3–6.

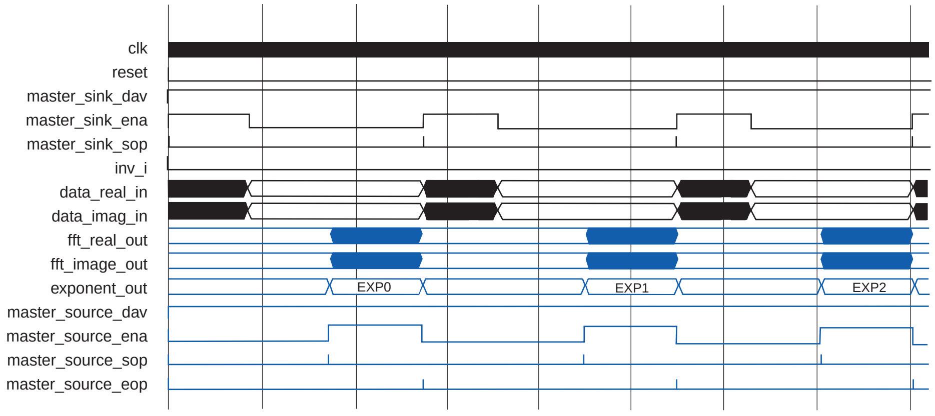
Burst I/O Data Flow Architecture

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 3–9 shows the simulation results for the burst architecture. Again, the signals master_source_ena and master_sink_ena indicate to the system data sources and slave sinks either side of the FFT, i.e., when the FFT can accept a new block of data and when a valid output block is available on the FFT output.

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

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



Performance & Resource Utilization

This section lists the FFT MegaCore function's device performance and resource utilization. Because performance varies depending on the FFT engine architecture and I/O data flow, the results are grouped accordingly.

Table 3–1 list Stratix® II device performance and resource utilization using the streaming data flow architecture.

Table 3–1. Stratix II Device Performance using the Streaming Data Flow Engine Architecture

Device	Points	Width *(1)*	LEs *(2)*	18*18 Mults	M4K *(3)*	MRAM	f_{MAX} (MHz)	Clock Cycle Count	Transform Time (us)
EP2S15F484C3	256	16	4360	9	19	0	335.23	256	0.76
EP2S15F484C3	512	16	4826	9	19	0	319.19	512	1.6
EP2S15F484C3	1024	16	5137	9	38	0	325.73	1,024	3.14
EP2S15F484C3	2048	16	6946	18	75/44	0/2	309.89	2,048	6.61
EP2S30F484C3	4096	16	7429	18	144/88	0/2	301.84	4,096	13.57
EP2S60F484C3	8192	16	7351	18	304/176	0/2	293.06	8,192	27.95

Notes to Table 3–1:
(1) Represents data and twiddle factor precision.
(2) The Quartus® II software reports the number of adaptive look-up tables (ALUTs) that the design uses in Stratix II devices. The logic element (LE) count is based on this number of ALUTs.
(3) For cases in which M-RAM utilization is permitted, the first number indicates the number of M4K RAM blocks when M-RAM usage is disabled from within the FFT wizard.

Table 3–2 lists Stratix II device resource utilization using buffered burst data flow architecture.

Table 3–2. Stratix II Device Resource Utilization Using Buffered Burst Data Flow Architecture

Device	Points	Width *(1)*	Number of Engines *(2)*	LEs *(3)*	18*18 Mults	M4K *(4)*	MRAM
EP2S15F484C3	256	16	1	4,458	9	15	0
EP2S15F484C3	256	16	2	7,203	18	30	0
EP2S30F484C3	256	16	4	13,101	36	60	0
EP2S15F484C3	1024	16	1	4,512	9	30	0
EP2S15F484C3	1024	16	2	7,467	18	30	0
EP2S30F484C3	1024	16	4	13,439	36	60	0
EP2S30F484C3	4096	16	1	4,777	9	112/80	0/1
EP2S30F484C3	4096	16	2	7,666	18	112/80	0/2
EP2S90F484C3	4096	16	4	13,693	36	110/88	0/4

Notes to Table 3–2:
(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.
(3) The Quartus II software reports the number of adaptive look-up tables (ALUTs) that the design uses in Stratix II devices. The LE count is based on this number of ALUTs.
(4) For cases in which M-RAM utilization is permitted, the first number indicates the number of M4K RAM blocks when M-RAM usage is disabled from within the MegaWizard Plug-In.

Table 3–3 lists Stratix II device performance using buffered burst data flow architecture.

Table 3–3. Stratix II Device Performance Using the Buffered Burst Data Flow Architecture (Part 1 of 2)

Device	Points	Width *(1)*	Number of Engines *(2)*	f_{MAX} (MHz)	Transform Calculation Time (us) *(3)*		Data Load & Transform Calculation		Block Throughput (Clock Cycles) *(4)*
					Cycles	Time (us)	Cycles	Time (us)	
EP2S15F484C3	256	16	1	324.36	235	0.73	491	1.51	331
EP2S15F484C3	256	16	2	273.07	141	0.52	397	1.45	299
EP2S30F484C3	256	16	4	263.74	118	0.45	374	1.42	283
EP2S15F484C3	1024	16	1	310.27	1,069	3.45	2,093	6.75	1,291
EP2S15F484C3	1024	16	2	289.44	593	2.04	1,617	5.58	1,163
EP2S30F484C3	1024	16	4	251.75	340	1.35	1,364	5.42	1,099
EP2S30F484C3	4096	16	1	320.10	5,167	16.14	9,263	28.94	6,157

Table 3–3. Stratix II Device Performance Using the Buffered Burst Data Flow Architecture (Part 2 of 2)

Device	Points	Width (1)	Number of Engines (2)	f_{MAX} (MHz)	Transform Calculation Time (us) (3)		Data Load & Transform Calculation		Block Throughput (Clock Cycles) (4)
					Cycles	Time (us)	Cycles	Time (us)	
EP2S30F484C3	4096	16	2	294.03	2,654	9.03	6,750	22.96	4,619
EP2S90F484C3	4096	16	4	247.93	1,378	5.56	5,474	22.08	4,363

Notes to Table 3–3:

(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. The user 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 defined as the minimum number of cycles between two successive start-of-packet (i.e., `master_sink_sop`) pulses.

Table 3–4 lists Stratix II resource utilization using burst data flow architecture.

Table 3–4. Stratix II Device Resource Utilization Using the Burst Data Flow Architecture (Part 1 of 2)

Device	Points	Width (1)	Engine Architecture	Number of Engines (2)	LEs (3)	18*18 Mults	M4K
EP2S15F484C3	256	16	Quad Output	1	4327	9	7
EP2S15F484C3	256	16	Quad Output	2	7122	18	14
EP2S30F484C3	256	16	Quad Output	4	13035	36	27
EP2S15F484C3	1024	16	Quad Output	1	4450	9	14
EP2S15F484C3	1024	16	Quad Output	2	7350	18	14
EP2S30F484C3	1024	16	Quad Output	4	13521	36	27
EP2S15F484C3	4096	16	Quad Output	1	4625	9	56
EP2S15F484C3	4096	16	Quad Output	2	7379	18	56
EP2S30F484C3	4096	16	Quad Output	4	13406	36	56
EP2S15F484C3	256	16	Single Output	1	1551	3	3
EP2S15F484C3	256	16	Single Output	2	2746	6	8
EP2S15F484C3	1024	16	Single Output	1	1608	3	9
EP2S15F484C3	1024	16	Single Output	2	2786	6	14
EP2S15F484C3	4096	16	Single Output	1	1655	3	36

Table 3–4. Stratix II Device Resource Utilization Using the Burst Data Flow Architecture (Part 2 of 2)

Device	Points	Width (1)	Engine Architecture	Number of Engines (2)	LEs (3)	18*18 Mults	M4K
EP2S15F484C3	4096	16	Single Output	2	2892	6	56

Notes to Table 3–4:
(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. The user may choose from one to two single-output engines in parallel, or from one, two, or four quad-output engines in parallel.
(3) The Quartus II software reports the number of adaptive look-up tables (ALUTs) that the design uses in Stratix II devices. The LE count is based on this number of ALUTs

Table 3–5 lists Stratix II device performance using burst data flow architecture.

Table 3–5. Stratix II Device Performance Using the Burst Data Flow Architecture (Part 1 of 2)

Device	Points	Width (1)	Engine Arch-itecture	Number of Engines (2)	f_{MAX} (MHz)	Transform Calculation Time (us) (3)		Data Load & Transform Calculation		Block Through-put (Clock Cycles) (4)
						Cycles	Time (us)	Cycles	Time (us)	
EP2S15F484C3	256	16	Quad Output	1	305.53	235	0.77	491	1.61	766
EP2S15F484C3	256	16	Quad Output	2	273.46	141	0.52	397	1.45	654
EP2S30F484C3	256	16	Quad Output	4	260.89	118	0.45	374	1.43	631
EP2S15F484C3	1024	16	Quad Output	1	315.76	1,069	3.39	2,093	6.63	3,117
EP2S15F484C3	1024	16	Quad Output	2	278.42	557	2.00	1,581	5.68	2,606
EP2S30F484C3	1024	16	Quad Output	4	249.62	340	1.36	1,364	5.46	2,389
EP2S15F484C3	4096	16	Quad Output	1	302.66	5,167	17.07	9,263	30.61	13,660
EP2S15F484C3	4096	16	Quad Output	2	278.55	2,607	9.36	6,703	24.06	10,800
EP2S15F484C3	4096	16	Quad Output	4	240.48	1,378	5.73	5,474	22.76	9,571
EP2S15F484C3	256	16	Single Output	1	318.27	1,115	3.50	1,371	4.31	1,628

Table 3–5. Stratix II Device Performance Using the Burst Data Flow Architecture (Part 2 of 2)

Device	Points	Width (1)	Engine Arch-itecture	Number of Engines (2)	f_{MAX} (MHz)	Transform Calculation Time (us) (3)		Data Load & Transform Calculation		Block Through-put (Clock Cycles) (4)
						Cycles	Time (us)	Cycles	Time (us)	
EP2S15F484C3	256	16	Single Output	2	315.56	585	1.85	841	2.67	1,098
EP2S15F484C3	1024	16	Single Output	1	303.12	5,230	17.25	6,344	20.63	7,279
EP2S15F484C3	1024	16	Single Output	2	302.76	2,652	8.76	3,676	12.14	4701
EP2S15F484C3	4096	16	Single Output	1	301.39	24,705	81.97	28,801	95.56	32,898
EP2S15F484C3	4096	16	Single Output	2	301.09	12,329	40.95	16,495	54.55	20,602

Notes to Table 3–5:
(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. The user 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 defined as the time frame when the input block is loaded until the first output sample (i.e., 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 (i.e., `master_sink_sop`) pulses.

Signals

Table 3–6 shows the FFT MegaCore function's signals.

Table 3–6. FFT MegaCore Function Signals (Part 1 of 2)

Signal	Direction	Description
`clk`	Input	FFT system clock
`reset`	Input	FFT active-high synchronous reset
`master_sink_dav`	Input	Master sink data available signal: Asserted by the FFT slave data source to indicate the availability of *N* data samples for input to the FFT function, where *N* is the transform length
`master_sink_ena`	Output	Master sink write enable signal: Asserted by the FFT function to indicate that data can be written into the function's input buffer
`master_sink_sop`	Input	Input start-of-packet: Indicates to the FFT function the start of an input data block. Should be asserted for one clock cycle synchronous with the first input data sample

Table 3–6. FFT MegaCore Function Signals (Part 2 of 2)

Signal	Direction	Description
`inv_i`	Input	Transform direction: Specifiable on a per-block basis concurrent with the assertion of master_sink_sop. `0` => FFT `1`=> IFFT
`data_real_in [M-1..0]`	Input	Input real data: Precision, *M* set in the FFT wizard.
`data_imag_in [M-1..0]`	Input	Input imaginary data: Precision, *M* set in the FFT wizard.
`fft_real_out [M-1..0]`	Output	Output real data: Precision, *M* set in the FFT wizard.
`fft_imag_out [M-1..0]`	Output	Output imaginary data: Precision, *M* set in the FFT wizard.
`exponent_out [5..0]`	Output	Signed block exponent: Accounts for scaling of internal signal values during FFT computation.
`master_source_dav`	Input	Asserted by the slave sink on the output of the FFT function to indicate that it can accept one block of *N* output samples.
`master_source_ena`	Output	Master source enable: Asserted by the FFT function when data is available to be output by the FFT.
`master_source_sop`	Output	Output start-of-packet: Asserted on first output sample of each block.
`master_source_eop`	Output	Output end-of-packet: Asserted on last output sample of each block.

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 `fft_real_out`, `fft_imag_out`, and `exponent_out` are forced low when the evaluation time expires.

For more information on OpenCore Plus hardware evaluation, see Chapter 1, OpenCore Plus Evaluation and *AN 320: OpenCore Plus Evaluation of Megafunctions*.