



Single- and Dual-Clock FIFO Megafunction

User Guide



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

Quartus II Version:	7.1
Document Version:	4.0
Document Date:	May 2007

Copyright © 2007 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.



I.S. EN ISO 9001

UG-MFNALT_FIFO-4.0

About this User Guide

Revision History	v
Referenced Documents	v
How to Contact Altera	vi
Typographic Conventions	vi

Chapter 1. About These Megafunctions

Device Family Support	1-1
Introduction	1-1
Features	1-2
General Description	1-3
The scfifo Megafunction	1-3
The dcfifo Megafunction	1-4
Flag Latency in the dcfifo Megafunction	1-7
Resource Utilization and Performance	1-8

Chapter 2. Getting Started

Software and System Requirements	2-1
Mega Wizard Plug-In Manager Customization	2-1
FIFO MegaWizard Page Descriptions	2-2
Inferring Megafunctions from HDL Code	2-11
Instantiating Megafunctions in HDL Code or Schematic Designs	2-11
Generating a Netlist for EDA Tool Use	2-12
Using the Port and Parameter Definitions	2-12
Identifying a Megafunction after Compilation	2-13
Simulation	2-13
Quartus II Simulation	2-13
EDA Simulation	2-14
SignalTap II Embedded Logic Analyzer	2-14
Design Examples for the FIFO Megafunctions	2-14
Design Files	2-14
Run Functional Simulation	2-15
Example 1: scfifo Megafunction Behavior in Legacy Mode	2-16
Initial Write Operation to an Empty Single-Clock FIFO (Legacy Mode):	
Simulation Results (0 ns to 90 ns)	2-17
Asynchronous and Synchronous Clear Operations in a Single-Clock FIFO (Legacy mode):	
Simulation Results (90 ns to 190 ns)	2-17
Read and Write Operations to an Almost Full Single-Clock FIFO (Legacy Mode):	
Simulation Results (290 ns to 410 ns)	2-18
Read Operation to an Almost Empty Single-Clock FIFO (Legacy Mode):	
Simulation Results (440 ns to 560 ns)	2-19

Example 2: scfifo Megafunction Behavior in Show-Ahead Mode	2–20
Initial Write Operation to an Empty Single-Clock FIFO (Show-Ahead Mode):	
Simulation Results (0 ns to 90 ns)	2–21
Asynchronous and Synchronous Clear Operations in Single-Clock FIFO (Show-Ahead Mode): Simulation Results (90 ns to 190 ns)	2–22
Read and Write Operations to an Almost Full Single-Clock FIFO (Show-Ahead Mode):	
Simulation Results (290 ns to 410 ns)	2–23
Read Operation to an Almost Empty Single-Clock FIFO (Show-Ahead Mode):	
Simulation Results (440 ns to 560 ns)	2–23
Example 3: dcfifo Megafunction Behavior in Legacy Mode	2–24
Initial Write Operation to an Empty Dual-Clock FIFO (Legacy Mode):	
Simulation Results (0 ns to 150 ns)	2–24
Read and Write Operations to an Almost Full Dual-Clock FIFO (Legacy Mode):	
Simulation Results (280 ns to 400 ns)	2–25
Read Operation to an Almost Empty Dual-Clock FIFO (Legacy Mode):	
Simulation Results (470 ns to 570 ns)	2–26
wrempty, rdempty, rdusedw[] and wrusedw[] Behaviors in a Dual-Clock FIFO (Legacy Mode) When rdclk is Faster Than wrclk: Simulation Results (580 ns to 680 ns)	2–27
Example 4: dcfifo Megafunction Behavior in Show-Ahead Mode	2–29
Initial Write Operation to an Empty Dual-Clock FIFO (Show-Ahead Mode):	
Simulation Results (0 ns to 150 ns)	2–29
Read and Write Operations to an Almost Full Dual-Clock FIFO (Show-Ahead Mode):	
Simulation Results (280 ns to 400 ns)	2–30
Read Operation to an Almost Empty Dual-Clock FIFO (Show-Ahead Mode):	
Simulation Results (470 ns to 570 ns)	2–31
Example 5: dcfifo Megafunction Behavior with Wide Write Data and Narrow Read Output	2–32
Example 6: dcfifo Megafunction Behavior with Narrow Write Data and Wide Read Output	2–34

Chapter 3. Specifications

Specifications for the scfifo Megafunction	3–1
Design Requirements for the scfifo Megafunction	3–1
SCFIFO Ports and Parameters	3–2
Flag Latency in the scfifo Megafunction	3–6
Specifications for the dcfifo Megafunction	3–7
Design Requirements for the dcfifo Megafunction	3–7
DCFIFO Ports and Parameters	3–9
Flag Latency in the dcfifo Megafunction	3–15



About this User Guide

Revision History The following table shows the revision history for this user guide.

Date and Document Version	Changes Made	Summary of Changes
May 2007 v4.0	Updates for Quartus II v7.1: <ul style="list-style-type: none">• Added support for Arria GX devices.• Updated for new GUI.• Added six design examples in place of functional description.• Reorganized and updated Chapter 3, Specifications to have separate tables for the scfifo and dcfifo megafunctions.• Added Referenced Documents section.	Major updates for Quartus II v7.1, including addition of six new design examples in place of the previous functional description, and a major reorganization of Chapter 3 to present specifications for the scfifo and the dcfifo megafunctions separately.
March 2007 v3.3	Updates for Quartus II v7.0: <ul style="list-style-type: none">• Minor content changes, including adding Stratix® III and Cyclone® III information• Re-took screenshots for software version 7.0	Minor updates for Quartus II v7.0, including addition of Stratix III and Cyclone III
September 2005 v3.2	Minor content changes.	

Referenced Documents

This user guide references the following documents:

- [AN 42: Metastability in Altera Devices](#)
- [Recommended HDL Coding Styles](#) chapter in volume 1 of the *Quartus II Handbook*
- [Synthesis](#) section in volume 1 of the *Quartus II Handbook*
- [Quartus II Integrated Synthesis](#) chapter in volume 1 of the *Quartus II Handbook*
- [Design Debugging Using the SignalTap II Embedded Logic Analyzer](#) chapter in volume 3 of the *Quartus II Handbook*.

How to Contact Altera

For the most up-to-date information about Altera® products, refer to the following table.

Information Type	Contact (1)
Technical support	www.altera.com/mysupport/
Technical training	www.altera.com/training/ custrain@altera.com
Product literature	www.altera.com/literature/
Altera literature services	literature@altera.com
FTP site	ftp.altera.com

Note to table:

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

Typographic Conventions

This document uses the typographic conventions shown in the following table.

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} , lqdesigns 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: <i><file name></i> , <i><project name>.pdf</i> 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: <code>data1</code> , <code>tdi</code> , <code>input</code> . Active-low signals are denoted by suffix <code>n</code> , e.g., <code>resetn</code> . Anything that must be typed exactly as it appears is shown in Courier type. For example: <code>c:\qdesigns\tutorial\chiptrip.gdf</code> . Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword <code>SUBDESIGN</code>), as well as logic function names (e.g., <code>TRI</code>) are shown in Courier.

About this User Guide

Visual Cue	Meaning
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.
	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.

Device Family Support

The Single-Clock FIFO (scfifo) and Dual-Clock FIFO (dcfifo) megafunctions support the following target Altera® device families:

- Arria™ GX
- Stratix® III
- Stratix II
- Stratix II GX
- Stratix
- Stratix GX
- Cyclone® III
- Cyclone II
- Cyclone
- HardCopy® II
- HardCopy Stratix
- MAX® II
- MAX 7000AE
- MAX 7000B
- MAX 7000S
- MAX 3000A
- ACEX 1K®
- APEX™ II
- APEX 20KC
- APEX 20KE
- FLEX 10K®
- FLEX® 10KA
- FLEX 10KE
- FLEX 6000

Introduction

As design complexities increase, use of vendor-specific IP blocks has become a common design methodology. Altera provides parameterizable megafunctions that are optimized for Altera device architectures. Using megafunctions instead of coding your own logic saves valuable design time. Additionally, the Altera-provided functions may offer more efficient logic synthesis and device implementation. You can scale a megafunction's implementation size by simply setting parameters.

To help you efficiently implement FIFOs in your design, the Quartus II software provides a FIFO MegaWizard® Plug-In Manager that supports both the scfifo and the dcfifo megafunctions. The FIFO MegaWizard Plug-In Manager is a user-view wizard that selects and customizes the megafunction based on the properties you select.

If the FIFO's read and write operations use the same clock, the Plug-In Manager customizes the scfifo megafunction. If the FIFO's read and write operations use different clocks, the Plug-In Manager customizes the dcfifo megafunction.

Features

Table 1–1 shows the features of the scfifo and dcfifo megafunctions.

Table 1–1. scfifo and dcfifo Megafunction Features (Part 1 of 2) (1)	
Features	Description
Support most of the common FIFO status flags	The following status flags are supported: <ul style="list-style-type: none"> ● full ● empty ● almost_empty (scfifo only) ● almost_full (scfifo only) ● usedw[] (scfifo only) ● wrusedw[] (dcfifo only) ● rdusedw[] (dcfifo only)
Support different write input data width and read output data width	Only applicable to dcfifo
Support legacy mode	Legacy synchronous FIFO mode treats the signal as a read-request signal: the FIFO outputs the data upon assertion of <code>rdreq</code> .
Support show-ahead mode	Show-ahead synchronous FIFO mode treats the <code>rdreq</code> signal as a read-acknowledge signal to minimize the FIFO's read latency. The FIFO outputs data when it is available; assertion of <code>rdreq</code> acknowledges the data and causes the FIFO to output the next data word if available).
Support optimization for speed or area	Optimizing for speed produces a faster FIFO. Optimizing for area results in fewer resources being used.
Support circuitry protection for FIFO overflow and underflow	Overflow checking disables write access while the FIFO is full. Underflow checking disables read access while the FIFO is empty. Turning on both these features prevents FIFO data corruption.

Table 1–1. scfifo and dcfifo Megafunction Features (Part 2 of 2) (1)

Support simultaneous read and write	Read and write operations can occur simultaneously.
Support additional bit as MSB extension of usedw[] ports	An additional bit becomes the MSB for the usedw [] port, allowing it to hold the correct count when the FIFO is full instead of the value 0. This feature is available for the dcfifo megafunction in the Arria GX, Stratix III, Stratix II, Stratix II GX, Cyclone III, and Cyclone II device families.

Note to Table 1–1:

- (1) Refer to the “SCFIFO Ports and Parameters” and “DCFIFO Ports and Parameters” sections in [Chapter 3, Specifications](#) for more detailed descriptions of these features.

General Description

The Altera parameterizable scfifo and dcfifo megafunctions can be customized to meet a wide range of data-buffering requirements. For example, the scfifo megafunction is a simple function that can simultaneously read and write in the same clock cycle. The dcfifo megafunction provides data buffering for both synchronous clock-domain and asynchronous clock-domain applications (high-speed data buffering). The dcfifo megafunction also supports simultaneous read and write operations.

You can parameterize the scfifo or dcfifo megafunctions to implement almost any combination of width and depth. The FIFO depth must be a power of two, but its only other limitation is the available memory space in the device itself. If the desired combination is not available on the Plug-In Manager menu, simply enter the width and depth required for your design. You can also configure the scfifo and dcfifo megafunctions to output several optional FIFO flag signals, including empty and full flags and the number of words read or written.

For a more detailed specification and requirements description for the scfifo and dcfifo megafunctions, refer to the “[Design Requirements for the scfifo Megafunction](#)” and “[Design Requirements for the dcfifo Megafunction](#)” sections in [Chapter 3, Specifications](#).

The scfifo Megafunction

The scfifo megafunction uses a single clock signal for FIFO read and write operations, and allows simultaneous read and write transactions.

The scfifo megafunction provides two operational modes:

- Legacy synchronous FIFO mode (Legacy mode)
- Show-ahead synchronous FIFO mode (Show-ahead mode)

When the scfifo megafunction is configured in Legacy mode, read data is available on the first clock cycle after `rdreq` is asserted (that is, the `rdreq` signal operates as a normal read request). However, when the scfifo megafunction is configured in Show-ahead mode, the `rdreq` signal operates as a read-acknowledge signal. Thus, the first word of valid data in the FIFO appears on its outputs without a read operation occurring.

The `almost_full` and `almost_empty` output ports are optional and are only available in scfifo. These flags provide an early indication that the FIFO is almost full or almost empty.

The value on `usedw []` increments for every write operation and decrements for every read operation. The `full` signal is asserted on the rising clock edge at which `usedw []` reaches the maximum number of words the FIFO can store, while the `empty` signal is asserted on the rising clock edge at which `usedw []` reaches its minimum value of zero.

For a more detailed description of these ports, refer to the “[SCFIFO Ports and Parameters](#)” section in [Chapter 3, Specifications](#).

The dcfifo Megafunction

The dcfifo megafunction is well-suited for the complex data processing requirements of high-data-rate, asynchronous-clock-domain applications, in which data is transferred or buffered between asynchronous clock domains. This megafunction can also perform simultaneous read and write transactions. The dcfifo megafunction is a dual-pipelined function that stores data in the dual-port memory structures available in Altera devices.

Like the scfifo megafunction, the dcfifo megafunction provides two operational modes:

- Legacy synchronous FIFO mode (Legacy mode)
- Show-ahead synchronous FIFO mode (Show-ahead mode)

When the dcfifo megafunction is configured in Legacy mode, read data is available on the first clock cycle after `rdreq` is asserted (that is, the `rdreq` signal operates as a normal read request). However, when the dcfifo megafunction is configured in Show-ahead mode, the `rdreq` signal operates as a read-acknowledge signal. Thus, the first word of valid data in the FIFO appears on its outputs without a read operation occurring.

You can configure the `dcfifo` megafunction to output any of the following six FIFO status flags:

- The `wrusedw[]` and `rdusedw[]` signals, which represent the number of words stored in the FIFO. The `wrusedw[]` signal is a registered output that is synchronous with the write clock (`wrc1k`), and the `rdusedw[]` signal is a registered output that is synchronous with the read clock (`rdc1k`).
- The `rdfull` and `rdempty` signals, which indicate whether the FIFO is full or empty. These signals are registered outputs synchronous with the read clock.
- The `wrfull` and `wrempty` signals, which indicate whether the FIFO is full or empty. These signals are registered outputs synchronous with the write clock.

Figure 1-1 is a block diagram that shows how the `dcfifo` megafunction is implemented with an Altera Memory Block. All control logic is implemented in logic elements (LEs).

Figure 1–1. DCFIFO Megafunction Block Diagram

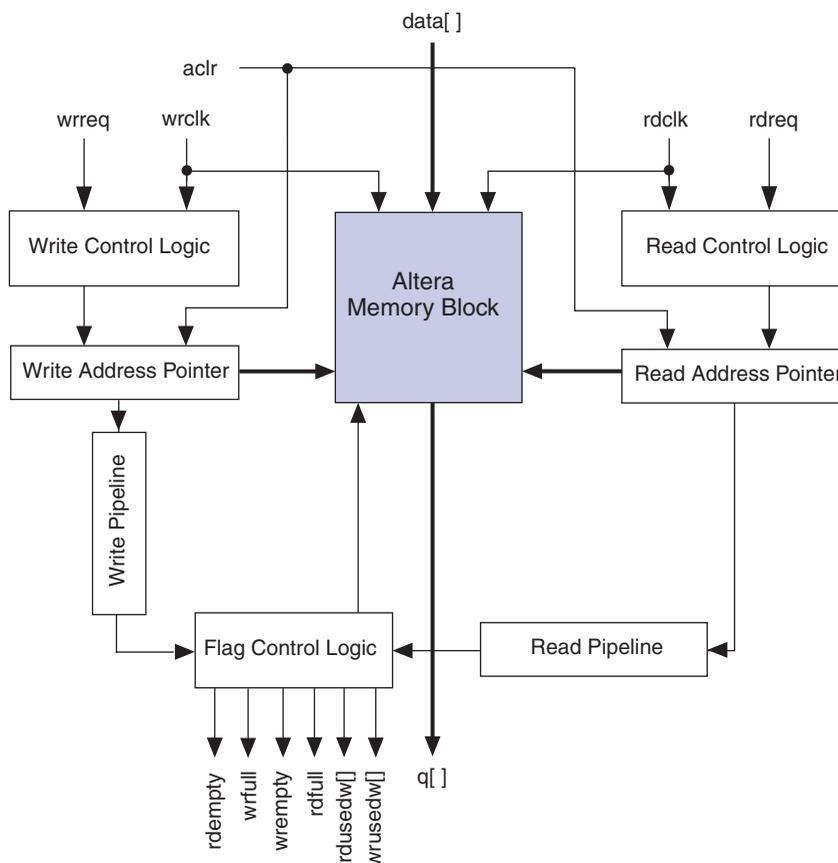


Figure 1–1 shows the `dcfifo` megafunction uses two independent clocks, `wrclk` and `rdclk`. The `wrclk` synchronizes data write transactions to the Altera Memory Block at the memory location indicated by the `Write Address Pointer`. The `rdclk` synchronizes data read transactions from the Altera Memory Block from the memory location indicated by the `Read Address Pointer`. The `Flag Control Logic` generates the six FIFO status flags by subtracting the value of the `Read Address Pointer` from the value of the `Write Address Pointer`.

Because the `Read Address Pointer` and the `Write Address Pointer` are in different clock domains, unsynchronized values output on the FIFO status flag signals can be incorrect as a result of metastability, especially when the read and write clock domains are not related. Synchronization in the `Read Pipeline` eliminates the effects of this

metastability in the paths from the `Read Address Pointer` to the Flag Control Logic. Similarly, synchronization in the Write Pipeline eliminates the effects of this metastability in the paths from the `Write Address Pointer` to the Flag Control Logic. The register delays in the pipelines delay the effects of updates to the two pointers on the FIFO status flags. Therefore, these synchronization pipelines cause the FIFO status signals `rdempty`, `rdfull`, `rdusedw[]`, `wrempty`, `wrfull`, and `wrusedw[]` to be delayed by multiple clock cycles.



For more information about synchronization pipelines, refer to *AN 42: Metastability in Altera Devices*.

Flag Latency in the `dcfifo` Megafunction

The Read Pipeline and the Write Pipeline shown in [Figure 1-1](#) cause the FIFO status flag output signals to be delayed by multiple clock cycles. The `rdfull`, `rdempty`, `wrfull`, and `wrempty` flags are generated from direct internal counter comparisons, and have lower latencies than the `rdusedw[]` and `wrusedw[]` signals. This section explains how the `rdusedw[]` and `wrusedw[]` signals are generated and how they contribute to the pipeline latency.

The value on the `wrusedw[]` signal is calculated in the `wrcclk` clock domain. The `Read Address Pointer` value is passed from the `rdclk` clock domain into the Flag Control Logic and is compared with the `Write Address Pointer` value in the `wrcclk` clock domain. The Flag Control Logic calculates the value on the `wrusedw[]` signal by subtracting the `Read Address Pointer` value from the current `Write Address Pointer` value. However, the `Read Address Pointer` cannot be passed to the Flag Status Logic directly because the `Read Address Pointer` value may be changing during the calculation.

Instead, the `Read Address Pointer` value is Gray-code encoded prior to the calculation. This encoding ensures only one bit changes at once. If the Flag Status Logic uses the encoded `Read Address Pointer` value while it is changing, the value used is either the old address or the new address, and not an intermediate, irrelevant value. However, two issues remain:

- If multiple registers use the changing bit as input, some registers may capture a zero and others a one, leading to an inconsistency in the logic calculations.
- If the setup or hold time of the changing bit is violated, a register may be in a metastable state, and may require time to recover.

To solve these problems, the encoded `Read Address Pointer` value is synchronized two or more times with the `wrcclk` clock. This synchronization occurs in the Read Pipeline. The Flag Control Logic then

converts the synchronized, encoded Read Address Pointer back into standard binary format, to enable subtraction, and generates the `wrusedw []` flag output signal.

The `rdusedw []` flag is generated similarly in the `rdclk` clock domain, requiring synchronization of the `Write Address Pointer` value in the Write Pipeline. The encoding and synchronization lead to a delay of several clock cycles to update the `rdusedw []` and `wrusedw []` flags. Because of this latency, the `rdusedw []` and `wrusedw []` output signals are unlikely to have the same value until several clock cycles pass with no read or write transactions.

The flag latencies may vary for different configuration settings and in difference device families.

For more detailed information about the latency of the `dcfifo` megafunction FIFO status flag, refer to the “[Flag Latency in the dcfifo Megafunction](#)” section in [Chapter 3, Specifications](#).

Resource Utilization and Performance

Altera recommends that you implement FIFO functions in the memory structures of Altera devices. If all the memory resources are used for other memory functions, the Quartus II software can implement the FIFOs in LEs as well. You may see a significant decrease in performance when these functions are implemented in LEs. Altera recommends you select the **Auto** option and let the Quartus II program decide on the memory implementation.

The MegaWizard Plug-In Manager displays the estimated resource usage. Different megafunction configurations produce different resource usage counts.



For the `scfifo` and `dcfifo` megafunctions, resource usage is reduced when overflow and underflow checking are disabled.



For the `dcfifo` megafunction, resource usage is further reduced when the optional output signals `rdusedw []` and `wrusedw []` are not selected.

Software and System Requirements

The instructions in this section require the following software:

- For operating system support information, refer to:
www.altera.com/support/software/os_support/oss-index.html
- Quartus® II software version 7.1 or higher

Mega Wizard Plug-In Manager Customization

The MegaWizard® Plug-In Manager creates or modifies design files that contain custom megafunction variations which can then be instantiated in a design file. The MegaWizard Plug-In Manager provides a wizard that allows you to specify options for the FIFO megafunctions. You can use the FIFO MegaWizard Plug-In Manager to set the `scfifo` or `dcfifo` megafunction features for each FIFO in the design.

Start the MegaWizard Plug-In Manager in one of the following ways:

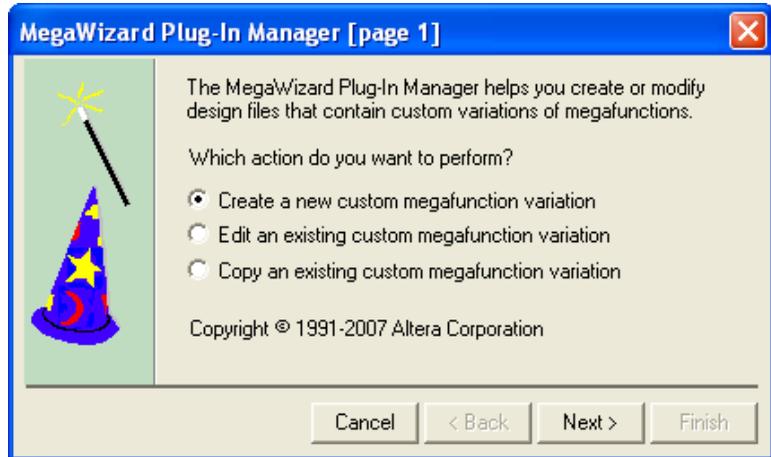
- On the Tools menu, click **MegaWizard Plug-In Manager**.
- When working in the Block Editor, from the Edit menu, click **Insert Symbol as Block**, or right-click in the Block Editor, point to **Insert**, and click **Symbol as Block**. In the **Symbol** dialog box, click **MegaWizard Plug-In Manager**.
- Start the stand-alone version of the MegaWizard Plug-In Manager by typing the following command at the command prompt:
`qmegawiz` ←

FIFO MegaWizard Page Descriptions

The FIFO MegaWizard Plug-In Manager instantiates the single- and dual-clock FIFO megafunctions (`scfifo` and `dcfifo`). This section describes the options available in the individual pages of the FIFO MegaWizard Plug-In Manager.

Page 1 of the MegaWizard Plug-In Manager is shown in [Figure 2-1](#).

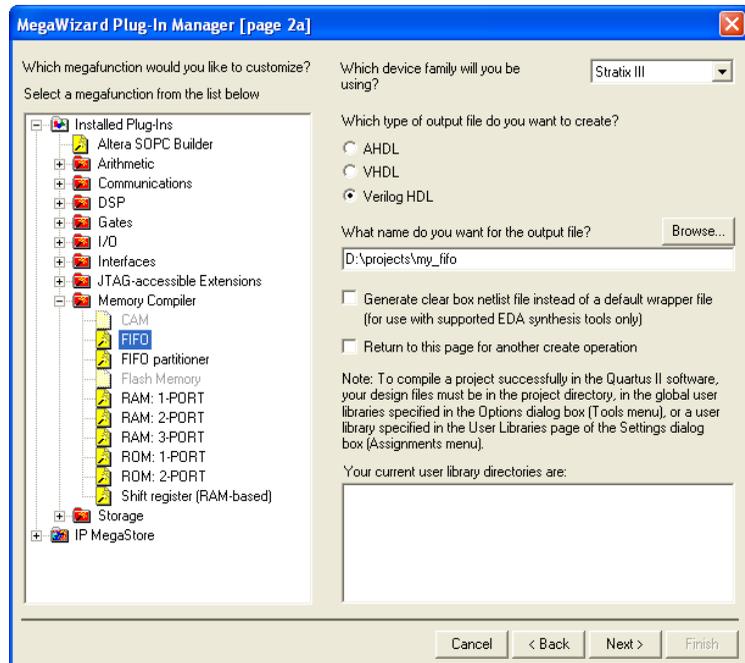
Figure 2-1. MegaWizard Plug-In Manager [page 1]



You can choose to create, edit, or copy a custom megafunction variation.

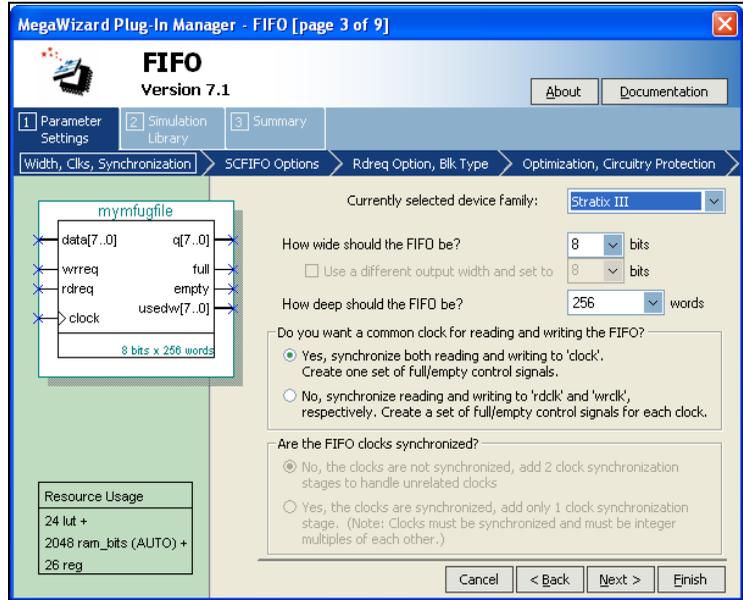
On page 2a of the MegaWizard Plug-In Manager, specify the plug-in, the device family you want to use, the type of output file to create, and the name of the output file (Figure 2-2). Choose AHDL (.tdf), VHDL (.vhd), or Verilog HDL (.v) as the output file type. You can also create a clear box instantiation for third-party EDA tools.

Figure 2-2. MegaWizard Plug-In Manager [page 2a]



On page 3 of the FIFO MegaWizard Plug-In Manager, you modify or retain the device family you previously specified, and you specify the FIFO width and depth. Figure 2-3 shows page 3 of the FIFO megafunction wizard.

Figure 2-3. MegaWizard Plug-In Manager—FIFO [page 3 of 9]



Starting on Page 3 of the FIFO megafunction wizard, you can generate a sample simulation waveform or launch the Quartus II software Help by selecting the **Generate Sample Waveforms** or **Quartus II Megafunction Reference** options from the **Documentation** button.

Table 2-1 shows the features and settings on page 3 of the FIFO megafunction wizard.

Function	Description	Available in scfifo	Available in dcfifo
Currently selected device family	Select which Altera device family you are using.	Yes	Yes
How wide should the FIFO be?	Specify the FIFO data width. If the width is not listed, type in the number.	Yes	Yes

Table 2–1. FIFO MegaWizard Plug-In Manager Page 3 Options (Part 2 of 2)

Function	Description	Available in scfifo	Available in dcfifo
Use a different output width and set to	Specify the different FIFO output data width. It defaults to FIFO data width. If the width is not listed, type in the number.	No	Yes
How deep should the FIFO be?	Specify the FIFO depth. If the depth is not listed, type in the number. The depth must be a power of two.	Yes	Yes
Do you want a common clock for reading and writing the FIFO?	Select Yes if you want a single-clock FIFO; No if you want a dual-clock FIFO.	Yes	Yes
Are the FIFO clocks synchronized?	Select Yes if your clocks are synchronized; No if your clocks are not synchronized (that is, are unrelated clocks).	No	Yes

On pages 4 and 5 of the FIFO MegaWizard Plug-In Manager, you select the FIFO control signals and flags. Page 4 is displayed when you choose a single-clock FIFO (Figure 2-4), and page 5 is displayed when you choose a dual-clock FIFO (Figure 2-5).

Figure 2-4. MegaWizard Plug-In Manager—FIFO [page 4 of 9]

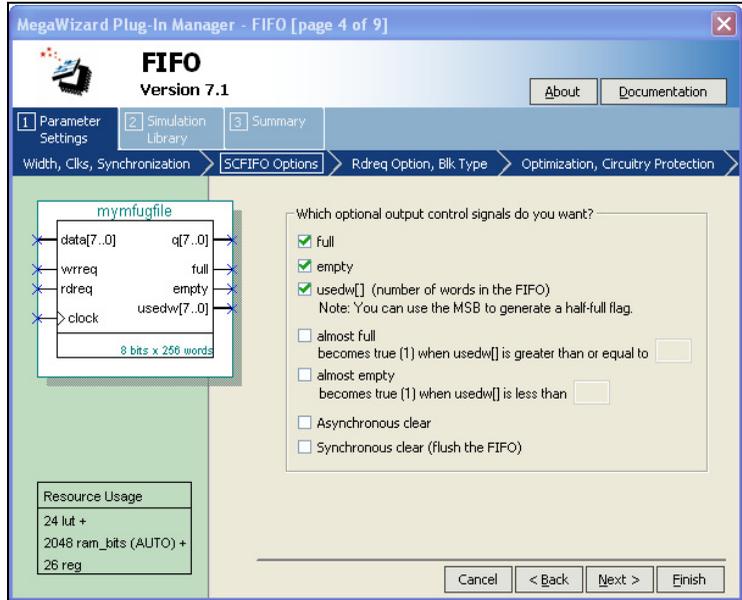


Figure 2-5. MegaWizard Plug-In Manager—FIFO [page 5 of 9]

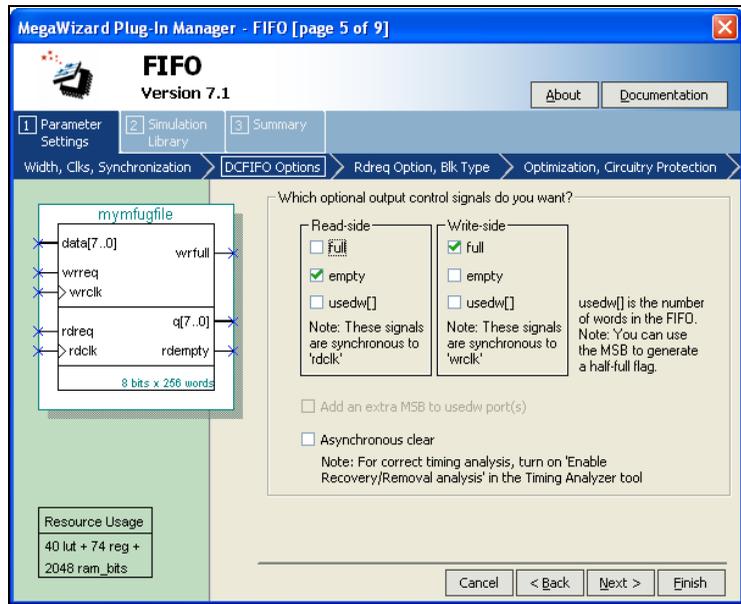


Table 2–2 shows the options and settings on pages 4 and 5 of the FIFO megafunction wizard.

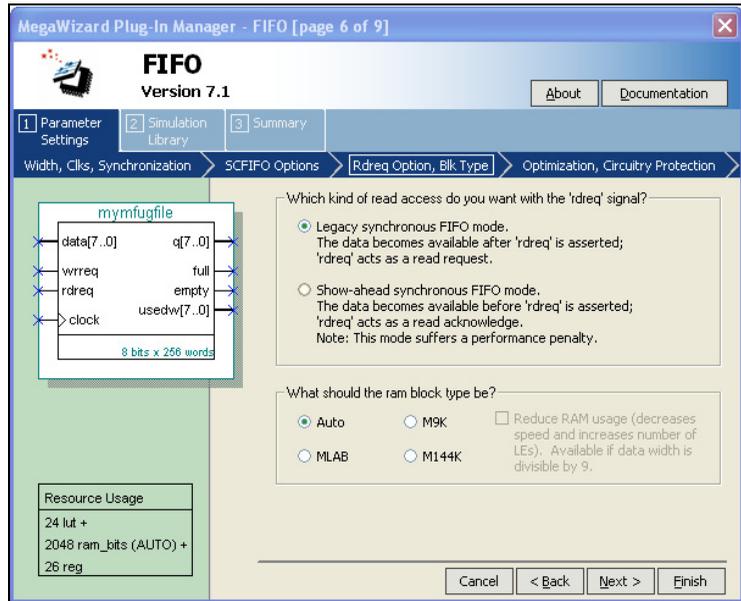
Table 2–2. FIFO MegaWizard Plug-In Manager Pages 4 and 5 Options	
Function	Description
Which optional output control signals do you want?	Specify the output control signals to use. For more information about the output control signals, see Tables 3–2 and 3–6 .
Asynchronous clear	Turn on this feature to implement an asynchronous clear input to empty the FIFO. For more information, see Tables 3–1 and 3–5 . For correct timing analysis, Altera recommends enabling the Removal and Recovery Analysis option in the timing analyzer for all device families when the <code>aclr</code> signal is used in a single-clock FIFO. If a dual-clock FIFO is instantiated in a Stratix II or Cyclone II device, enabling the Removal and Recovery Analysis option in the timing analyzer is required whether or not the <code>aclr</code> signal is used. If a dual-clock FIFO is instantiated in a device not in the Stratix II or Cyclone II device families, Altera recommends enabling the Removal and Recovery Analysis option in the timing analyzer when the <code>aclr</code> signal is used. See the “ Design Requirements for the scfifo Megafunction ” and “ Design Requirements for the dcfifo Megafunction ” sections in Chapter 3, Specifications . (1)
Synchronous clear	Turn on this feature to implement a synchronous clear input to empty the FIFO. This port is only available for the <code>scfifo</code> megafunction. For more information, see Table 3–1 .
Add an extra MSB to usedw port(s)	Turn on this feature to add one extra bit as the MSB for each <code>usedw []</code> port, allowing it to hold the correct count when the FIFO is full instead of holding the value 0. This feature is available only for the <code>dcfifo</code> megafunction, in Arria GX, Stratix III, Stratix II, Stratix II GX, Cyclone III, and Cyclone II devices. For more information, see Table 3–5 .

Note to Table 2–2:

- (1) For more information about removal and recovery, refer to the *Quartus II TimeQuest Timing Analyzer* chapter or the *Quartus II Classic Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

On page 6 of the FIFO megafunction wizard, you specify Legacy or Show-ahead mode and the RAM block options (Figure 2–6).

Figure 2–6. MegaWizard Plug-In Manager—FIFO [page 6 of 9]



Both the `scfifo` and `dcfifo` megafunctions can be configured in Legacy or Show-ahead mode for all supported devices.

Table 2-3 shows the features and settings on page 6 of the FIFO megafunction wizard.

Function	Description
What kind of read access do you want with the 'rdreq' signal?	Specify Legacy or Show-ahead mode. In Legacy mode, the rdreq signal acts as a read request. In Show-ahead mode, the rdreq signal acts as a read acknowledge. Show-ahead mode minimizes the FIFO read latency. For more information about the FIFO megafunction modes, see the section “General Description” on page 1-3.
What should the RAM block type be?	Specify the RAM block type. Altera recommends you select the Auto option.

On page 7 of the FIFO megafunction wizard, you specify the optimization, overflow and underflow checking, and placement setting options (Figure 2-7).

Figure 2-7. MegaWizard Plug-In Manager—FIFO [page 7 of 9]

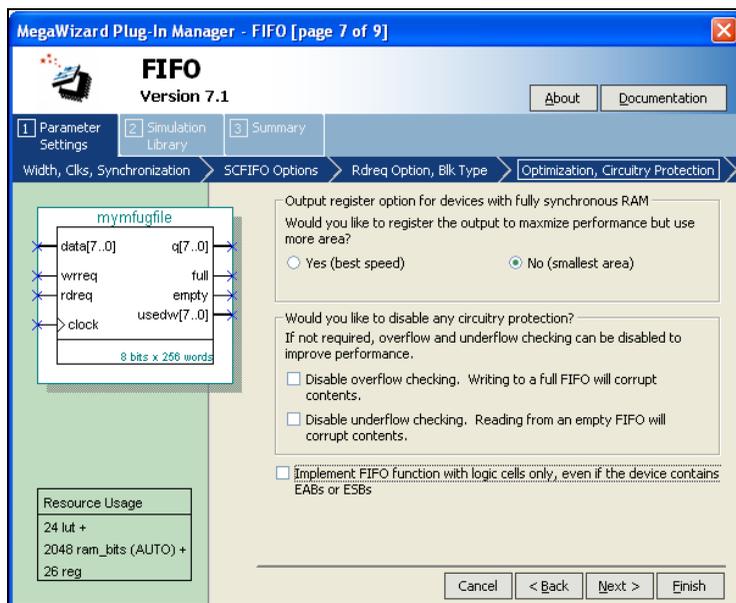


Table 2–4 shows the features and settings on page 7 of the FIFO megafunction wizard.

Function	Description
Optimization	Specify whether you want to optimize for speed or area.
Disable circuitry protection?	Specify what kind of circuitry protection you want to disable. Disabling overflow or underflow protection will improve FIFO performance and reduce the use of device resources, but may lead to data corruption in the FIFO.
Implement the FIFO function with logic cells.	Turn on this feature to implement the FIFO with logic cells even if the device contains EABs or ESBs.

Inferring Megafunctions from HDL Code

Synthesis tools, including the Quartus II integrated synthesis, recognize certain types of HDL code and automatically infer the appropriate megafunction when a megafunction will provide optimal results. That is, the Quartus II software uses the Altera megafunction code when compiling your design, even though you did not specifically instantiate the megafunction. The Quartus II software infers megafunctions because they are optimized for Altera devices, so their area or performance, or both, may be better than that of generic HDL code. Additionally, you must use megafunctions to access certain Altera architecture-specific features—such as memory, DSP blocks, and shift registers—that generally provide improved performance compared with basic logic elements.



Refer to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook* for specific information about your particular megafunction.

Instantiating Megafunctions in HDL Code or Schematic Designs

When you use the MegaWizard Plug-In Manager to customize and parameterize a megafunction, it creates a set of output files that allow you to instantiate the customized function in your design. Depending on the language you choose in the MegaWizard Plug-In Manager, the MegaWizard instantiates the megafunction with the correct parameter values and generates a megafunction variation file (wrapper file) in Verilog HDL (.v), VHDL (.vhd), or AHDL (.tdf), along with other supporting files.

The MegaWizard Plug-In Manager provides options to create the following files:

- A sample instantiation template for the language of the variation file (`_inst.v`, `_inst.vhd`, or `_inst.tdf`)
- Component Declaration File (`.cmp`) that can be used in VHDL Design Files
- ADHL Include File (`.inc`) that can be used in Text Design Files (`.tdf`)
- Quartus II Block Symbol File (`.bsf`) that can be used in schematic designs
- Verilog HDL module declaration file that can be used when instantiating the megafunction as a black box in a third-party synthesis tool (`_bb.v`)



For more information about the MegaWizard-generated files, refer to Quartus II Help or to the *Recommended HDL Coding Styles* chapter in volume 1 of the *Quartus II Handbook*.

Generating a Netlist for EDA Tool Use

If you use a third-party EDA synthesis tool, you can instantiate the megafunction variation file as a black box for synthesis. Use the VHDL component declaration or Verilog module declaration black-box file to define the function in your synthesis tool, and then include the megafunction variation file in your Quartus II project.

If you enable the option to generate a synthesis area and timing estimation netlist in the MegaWizard Plug-In Manager, the MegaWizard generates an additional netlist file (`_syn.v`). The netlist file is a representation of the customized logic used in the Quartus II software. The file provides the connectivity of the architectural elements in the megafunction but may not represent true functionality. This information enables certain third-party synthesis tools to better report area and timing estimates. In addition, synthesis tools can use the timing information to focus timing-driven optimizations and improve the quality of results.



For more information about using megafunctions in your third-party synthesis tool, refer to the appropriate chapter in the *Synthesis* section in volume 1 of the *Quartus II Handbook*.

Using the Port and Parameter Definitions

Instead of the MegaWizard Plug-In Manager, you can instantiate the megafunction directly in your Verilog HDL, VHDL, or AHDL code by calling the megafunction and settings its parameters as you would any other module, component, or subdesign.



Altera strongly recommends that you use the MegaWizard Plug-In Manager for complex megafunctions. The MegaWizard Plug-In Manager ensures that you set all megafunction parameters properly.

Refer to [Chapter 3, Specifications](#) for a list of the megafunction ports and parameters.

Identifying a Megafunction after Compilation

During compilation with the Quartus II software, analysis and elaboration is performed to build the structure of your design. To locate your megafunction in the Project Navigator window, expand the compilation hierarchy and find the megafunction by its name.

Similarly, to search for node names within the megafunction (using the Node Finder), click **Browse (...)** in the **Look in** box and select the megafunction in the **Hierarchy** box.

Simulation

The Quartus II Simulation tool provides an easy-to-use, integrated solution for performing simulations. The following sections describe the simulation options.

Quartus II Simulation

With the Quartus II Simulator, you can perform two types of simulations: functional and timing. A functional simulation in the Quartus II program enables you to verify the logical operation of your design without taking into consideration the timing delays in the FPGA. This simulation is performed using only your RTL code. When performing a functional simulation, you add only signals that exist before synthesis. You can find these signals with the **Registers: pre-synthesis**, **Design Entry**, or **Pin** filters in the Node Finder. The top-level ports of megafunctions are found using these three filters.

In contrast, timing simulation in the Quartus II software verifies the operation of your design with annotated timing information. This simulation is performed using the post place-and-route netlist. When performing a timing simulation, you add only signals that exist after place-and-route. These signals are found with the **Post-Compilation** filter of the Node Finder. During synthesis and place-and-route, the names of your RTL signals will change. Therefore, it might be difficult to find signals from your megafunction instantiation in the **Post-Compilation** filter. However, if you want to preserve the names of your signals during the synthesis and place and route stages, you must use the synthesis attributes `keep` or `preserve`. These are Verilog and VHDL synthesis

attributes that direct analysis and synthesis to keep a particular wire, register, or node intact. You can use these synthesis attributes to keep a combinational logic node so you can observe the node during simulation.



More information about these attributes is available in the *Quartus II Integrated Synthesis* chapter in volume 1 of the *Quartus II Handbook*.

EDA Simulation

Depending on the third-party simulation tool you are using, refer to the appropriate chapter in the Simulation section in volume 3 of the *Quartus II Handbook*. The *Quartus II Handbook* chapters show you how to perform functional and gate-level timing simulations that include the megafunctions, with details on the files that are needed and the directories where those files are located.

SignalTap II Embedded Logic Analyzer

The SignalTap® II embedded logic analyzer provides you with a non-intrusive method of debugging all the Altera megafunctions in your design. With the SignalTap II embedded logic analyzer, you can capture and analyze data samples for the top-level ports of the Altera megafunctions in your design while your system is running at full speed.

To monitor signals from your Altera megafunctions, you must first configure the SignalTap II embedded logic analyzer in the Quartus II software, and then include the analyzer as part of your Quartus II project. The Quartus II software will then seamlessly embed the analyzer along with your design in the selected device.



For more information about using the SignalTap II embedded logic analyzer, refer to the *Design Debugging Using the SignalTap II Embedded Logic Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Design Examples for the FIFO Megafunctions

This section presents six design examples of single-clock and dual-clock FIFOs configured by the FIFO MegaWizard Plug-In Manager. For each example, waveforms illustrate the functional behavior that results from the different configuration settings. For demonstration purposes, the Stratix III family of devices is selected in these examples.

Design Files

The example design files are available with this user guide in the Quartus II Project section and in the User Guides section of the Altera website (www.altera.com).

Six design examples are available for the FIFO megafunctions. The design examples are explained in this section. Each section contains a list of `scfifo` or `dcfifo` megafunction configuration settings used to generate the design example described. You can proceed directly to the section for the specific design example you wish to download.



Latency cycles shown in the waveforms for each design example are the result of functional simulation of the particular example only. Latency cycles may differ for different device families and different configuration settings. Refer to [Tables 3-4](#) and [3-8](#) for the latency specifications for all device families with different configuration settings.

Run Functional Simulation

After you have downloaded a design example, you can run Quartus II functional simulation to display its functional behavior waveform. The examples are already configured and compiled.

Set up and run the Quartus II Simulator by performing the following steps:

1. On the Processing menu, click **Generate Functional Simulation Netlist**.
2. When the **Functional Simulation Netlist Generation was successful** message box appears, click **OK**.
3. On the Processing menu, click **Start Simulation**, or, on the toolbar, click the **Start Simulation** button.
4. When the **Simulator was successful** message box appears, click **OK**.
5. In the Simulation Report window, view the simulation output waveforms to verify the results.

Example 1: scfifo Megafunction Behavior in Legacy Mode

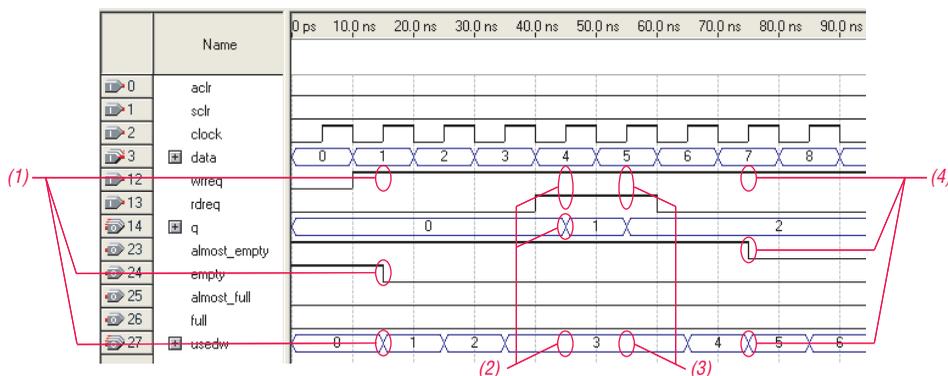
Table 2–5 shows the MegaWizard Plug-In Manager configuration settings that were used to create this design example, a single-clock FIFO in Legacy mode.

<i>Table 2–5. Configuration Settings for a scfifo in Legacy Mode (Optimized for Area)</i>	
MegaWizard Configuration Setting	Value
Family	Stratix III
FIFO data width	8 bits
FIFO output width	8 bits
FIFO data depth	16 words
Common clock for reading and writing	Yes
Output control signals	Enable all output control signals (e.g., full, empty, usedw[], almost_full, almost_empty, aclr and sclr)
Almost full	12
Almost empty	5
FIFO Mode	Legacy
RAM block type	Auto
Optimization	Area
Disable overflow & underflow checkings	No
Implement FIFO in logic cells	No

Initial Write Operation to an Empty Single-Clock FIFO (Legacy Mode): Simulation Results (0 ns to 90 ns)

Figure 2–8 shows the functional simulation results for the initial write operation to an empty FIFO from 0 ns to 90 ns. It shows how the `empty` and `almost_empty` output signals behave.

Figure 2–8. Initial Write Operation to an Empty Single-Clock FIFO (Legacy Mode)

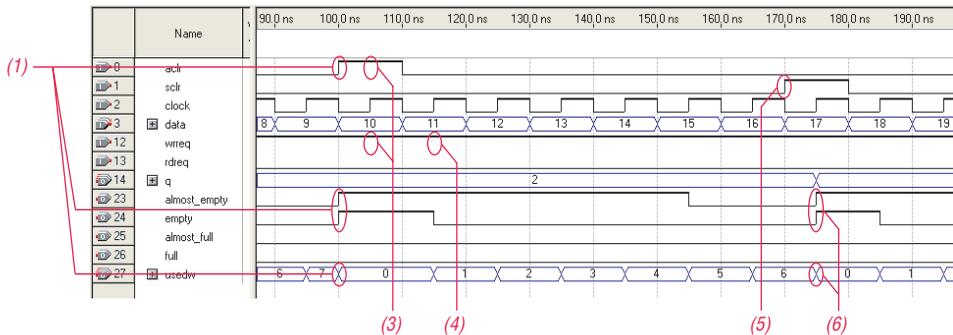


Notes to Figure 2–8:

- (1) At 15 ns, the data value 1 is written to the FIFO, the signal `empty` is deasserted, and the value of signal `usedw []` increments from 0 to 1. The value changes on `empty` and `usedw []` occur on the same rising edge of the clock as the data write, 0 clock cycles after the `wrreq` signal is asserted at 10 ns. Latencies are: `wrreq` to `empty`: 0 clock cycles, and `wrreq` to `usedw []`: 0 clock cycles.
- (2) At 45 ns, read and write operations occur simultaneously. The data value 1 is read from the FIFO on data output `q []`. However, the value of `usedw []` does not change, because the data value 4 is simultaneously written to the FIFO.
- (3) At 55 ns, simultaneous read and write operations again leave the value of the signal `usedw []` unchanged.
- (4) At 75 ns, the `almost_empty` signal is deasserted when the value of `usedw []` reaches 5, the configured threshold value.

Asynchronous and Synchronous Clear Operations in a Single-Clock FIFO (Legacy mode): Simulation Results (90 ns to 190 ns)

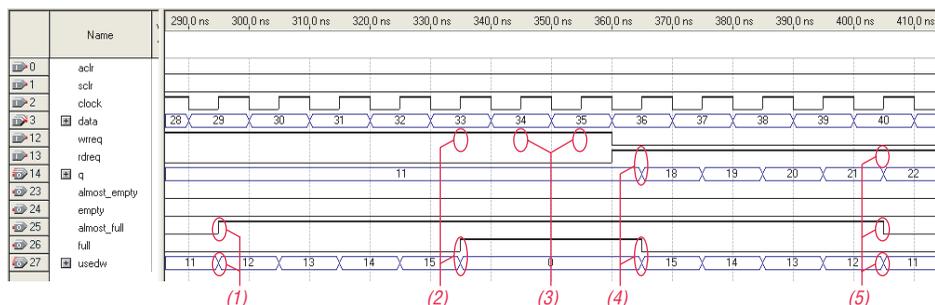
Figure 2–9 shows the functional simulation results for asserting the `aclr` and `sclr` signals from 90 ns to 190 ns. It shows the detailed behavior triggered by asserting the `aclr` and `sclr` signals.

Figure 2–9. Asynchronous and Synchronous Clear Operations in a Single-Clock FIFO (Legacy Mode)**Notes to Figure 2–9:**

- (1) At 100 ns, the `aclr` signal is asserted. Immediately, the `almost_empty` and the `empty` signals are asserted, and the `usedw []` signal resets to 0. The output data port `q []` remains at its previous value, data value 2.
- (2) In different device families, the behavior of `q []` following the assertion of `aclr` may vary. See the “Design Requirements for the `scfifo` Megafunction” section in Chapter 3, Specifications.
- (3) At 105 ns, the `aclr` signal remains asserted, and `wrreq` also remains asserted. No write transaction occurs, because the `wrreq` and `rdreq` signals are disabled until `aclr` is deasserted.
- (4) At 115 ns, on the rising edge of `clock` after `aclr` is deasserted, data begins writing to the FIFO.
- (5) At 170 ns, the `sclr` signal is asserted.
- (6) At 175 ns, the rising edge of `clock` after `sclr` is asserted, the `empty` and `almost_empty` flags are asserted, the `usedw []` signal resets to 0, and the first data word in the FIFO (data value 11 which was written to the FIFO at 115 ns) is flushed on the `q []` output.

Read and Write Operations to an Almost Full Single-Clock FIFO (Legacy Mode): Simulation Results (290 ns to 410 ns)

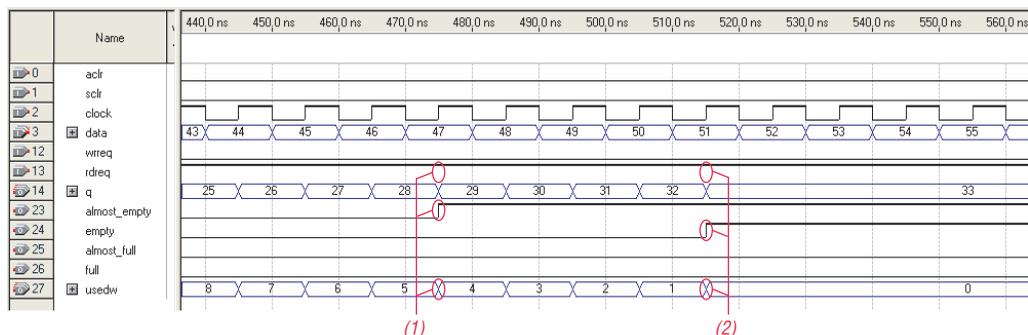
Figure 2–10 shows the functional simulation results for read and write operations into an almost full FIFO from 290 ns to 410 ns. It shows the assertion of the `almost_full` and `full` signals during a write operation and the deassertion of the `almost_full` and `full` signals during a read operation.

Figure 2–10. Read and Write Operations to an Almost Full Single-Clock FIFO (Legacy Mode)**Notes to Figure 2–10:**

- (1) At 295 ns, the `usedw []` signal value reaches 12, the configured threshold value for `almost_full`. Immediately, the `almost_full` signal is asserted.
- (2) At 335 ns, on the rising edge of `clock`, the FIFO becomes full in response to a `wrreq` signal at the previous falling edge of `clock` (at 330 ns). Immediately, the `full` signal is asserted (latency `wrreq` to `full`: 0 `clock` cycle) and the `usedw []` signal increments (modulo 16) from 15 to 0. Latencies are `rdreq` to `full`: 0 `clock` cycle, and `rdreq` to `usedw []`: 0 `clock` cycle.
- (3) At 345 ns and 355 ns, write operations should occur, because `wrreq` remains asserted. However, the `full` signal is still asserted at these times, and so the overflow circuitry checking, which is enabled, prevents the write operations.
- (4) At 365 ns, on the rising edge of `clock`, a read operation occurs. Immediately, the data port `q []` outputs the first data word in the FIFO (the data value 18, written to the FIFO at 185 ns), the `full` signal is deasserted, and the `usedw []` signal decrements (modulo 16) from 0 to 15.
- (5) At 405 ns, the `almost_full` signal is deasserted when the `usedw []` signal value is less than 12, the configured threshold value.

*Read Operation to an Almost Empty Single-Clock FIFO (Legacy Mode):
Simulation Results (440 ns to 560 ns)*

Figure 2–11 shows the functional simulation results for a read operation to an almost empty FIFO from 440 ns to 560 ns. It shows the behavior of the `empty` and `almost_empty` signals during this read operation to an almost empty FIFO.

Figure 2–11. Read Operation to an Almost Empty Single-Clock FIFO (Legacy Mode)

Notes to Figure 2–11:

- (1) At 475 ns, the `almost_empty` signal is asserted when the value on the `usedw []` signal drops below the configured threshold value 5.
- (2) At 515 ns, on the rising edge of clock, a read operation occurs and causes the final FIFO data value to be read out on the `q []` output port and the FIFO to become empty. Immediately, the `empty` signal is asserted (latency is `rdreq` to empty: 0 clock cycles) and the `usedw []` signal decrements from 1 to 0.

Example 2: `scfifo` Megafunction Behavior in Show-Ahead Mode

Table 2–6 shows the MegaWizard Plug-In Manager configuration settings that were used to create this design example, a single-clock FIFO in Show-ahead mode.

Table 2–6. Configuration Settings for a `scfifo` in Show-Ahead Mode (Optimized for Area)

MegaWizard Configuration Setting	Value
Family	Stratix III
FIFO data width	8 bits
FIFO output width	8 bits
FIFO data depth	16 words
Common clock for reading and writing	Yes
Output control signals	Enable all output control signals (e.g., full, empty, usedw[], almost_full, almost_empty, aclr and sclr)
Almost full	12
Almost empty	5
FIFO Mode	Show-ahead
RAM block type	Auto
Optimization	Area

Table 2–6. Configuration Settings for a scfifo in Show-Ahead Mode (Optimized for Area)

Disable overflow & underflow checking	No
Implement FIFO in logic cells	No

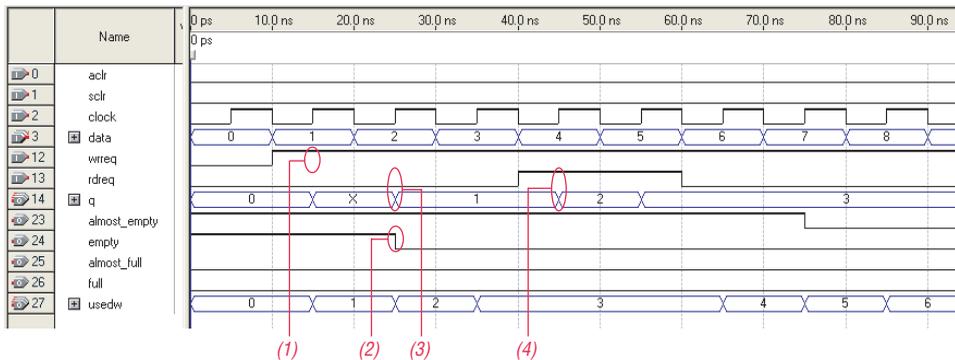
As described in the section “[General Description](#)” on page 1–3, the behavior of the `scfifo` megafunction in Legacy mode and in Show-ahead mode is identical, except that in Show-ahead mode the `rdreq` signal operates as a read-acknowledge signal. Therefore, the `q []` bus outputs the first FIFO data word regardless of whether a read operation occurs. The `q []` bus outputs different values for a FIFO in Legacy mode and a FIFO in Show-ahead mode.

The FIFO flag signals may also have different latency in the two different FIFO modes. For complete latency information with different families and different configuration settings, refer to [Table 3–4](#) on page 3–6.

*Initial Write Operation to an Empty Single-Clock FIFO
(Show-Ahead Mode): Simulation Results (0 ns to 90 ns)*

[Figure 2–12](#) shows the functional simulation results for the initial write operation to an empty FIFO from 0 ns to 90 ns. It shows how the `empty` and `almost_empty` output signals behave.

The behavior of the initial write operation to an empty single-clock FIFO in Show-ahead mode or in Legacy mode is identical except for the behavior of the `empty` and `q []` signals. [Figure 2–12](#) illustrates the differences between the behaviors of a single-clock FIFO in Show-ahead mode and in Legacy mode.

Figure 2–12. Initial Write Operation to an Empty Single-Clock FIFO (Show-Ahead Mode)**Notes to Figure 2–12:**

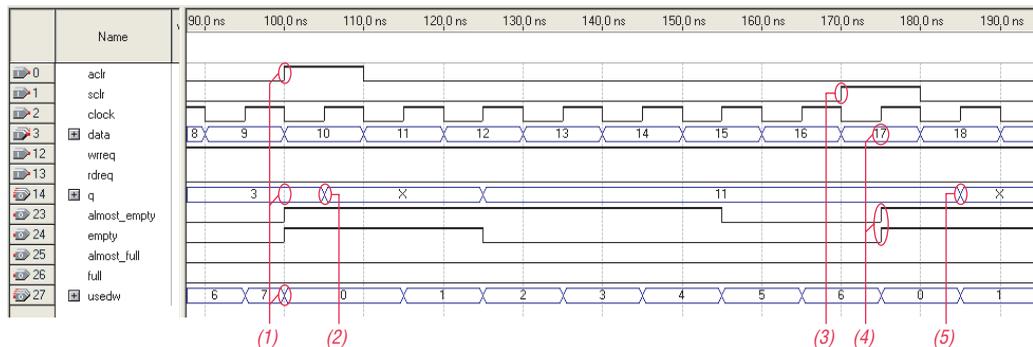
- (1) At 15 ns, on the rising edge of `clock`, the first FIFO data word is written.
- (2) At 25 ns, on the next rising edge of `clock`, the `empty` signal is deasserted. In Legacy mode, the `empty` signal would be asserted immediately at the first write, at 15 ns.
- (3) Also at 25 ns, the `q []` data port outputs the first word of valid FIFO data, even though `rdreq` is not asserted. In Legacy mode, this data would not appear on `q []` until the first rising `clock` edge after `rdreq` is asserted (at 45 ns).
- (4) At 45 ns, a read operation occurs (`rdreq` is high on the rising `clock` edge). Immediately, the `q []` data port outputs the second word of valid FIFO data, the data value 2, which was written to the FIFO at 25 ns. In Legacy mode, the first word of valid FIFO data, the data value 1, which was written to the FIFO at 15 ns, would appear at 45 ns, and the second word would not appear until 55 ns.

Asynchronous and Synchronous Clear Operations in Single-Clock FIFO (Show-Ahead Mode): Simulation Results (90 ns to 190 ns)

Figure 2–13 shows the functional simulation results for asserting the `aclr` and `sclr` signals from 90 ns to 190 ns. It shows the detailed behavior triggered by asserting the `aclr` and `sclr` signals.

The behavior of an empty single-clock FIFO after the `aclr` or `sclr` signal is asserted is identical whether the single-clock FIFO is in Show-ahead mode or in Legacy mode, except for the behavior of the `q []` output signal.

Figure 2–13 illustrates the differences between the behaviors of a single-clock FIFO in Show-ahead mode and in Legacy mode. Note that the `q []` port may behave differently in different device families.

Figure 2–13. Asserting the `aclr` and `sclr` Signals in a Single-Clock FIFO (Show-Ahead Mode)**Notes to Figure 2–13:**

- (1) At 100 ns, the `aclr` signal is asserted. Immediately, the `empty` and `almost_empty` signals are asserted. However, the output port `q []` maintains its previous value until the next rising `clock` edge. In Legacy mode, the `q []` port would maintain its previous value until a read operation occurs.
- (2) At 105 ns, on the next rising `clock` edge, the output port `q []` reverts to an undefined value.
- (3) At 170 ns, the `sclr` signal is asserted.
- (4) At 175 ns, on the next rising `clock` edge, the `empty` and `almost_empty` ports are asserted. The `q []` port maintains its previous value (data value 11) until the following rising `clock` edge. In Legacy mode, the `q []` port would maintain its previous value until a read operation occurs.
- (5) At 185 ns, on the following rising `clock` edge, the output port `q []` reverts to an undefined value.

Read and Write Operations to an Almost Full Single-Clock FIFO (Show-Ahead Mode): Simulation Results (290 ns to 410 ns)

The behavior of read and write operations to an almost-full single-clock FIFO in Show-ahead mode or in Legacy mode is the same. See [Figure 2–10](#) for the waveform descriptions.

Read Operation to an Almost Empty Single-Clock FIFO (Show-Ahead Mode): Simulation Results (440 ns to 560 ns)

The behavior of a read operation to an almost-empty FIFO in Show-ahead mode or in Legacy mode is the same. See [Figure 2–11](#) for the waveform descriptions.

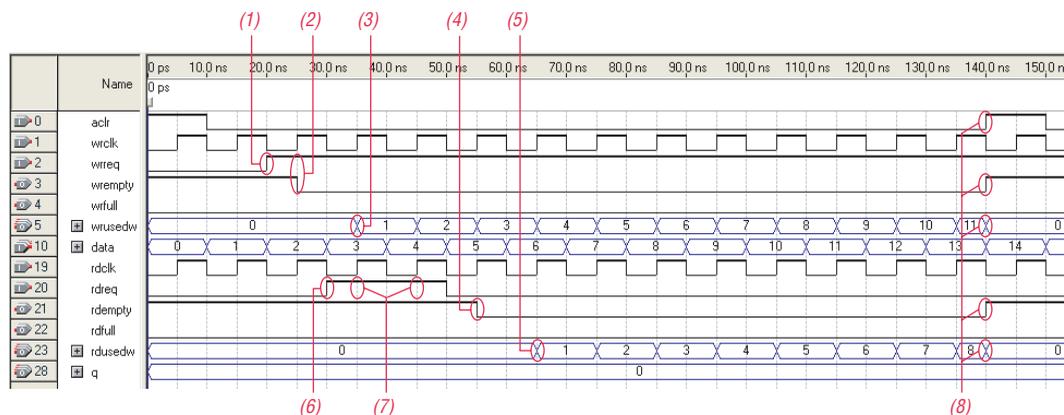
Example 3: dcfifo Megafunction Behavior in Legacy Mode

Table 2–7 shows the MegaWizard Plug-In Manager configuration settings that were used to create this design example, a dual-clock FIFO in Legacy mode.

<i>Table 2–7. Configuration Settings for a dcfifo in Legacy Mode (Optimized for Area)</i>	
MegaWizard Configuration Setting	Value
Family	Stratix III
FIFO data width	8 bits
FIFO output width	8 bits
FIFO data depth	16 words
Common clock for reading and writing	No
FIFO Clocks are synchronized	No
Output control signals	Enable all the control signals (e.g. <code>full</code> , <code>empty</code> , <code>usedw[]</code> , <code>aclr</code>) for both the read and write clock domains.
FIFO Mode	Legacy
RAM block type	Auto
Optimization	Area
Disable overflow & underflow checkings	No
Implement FIFO in logic cells	No

Initial Write Operation to an Empty Dual-Clock FIFO (Legacy Mode): Simulation Results (0 ns to 150 ns)

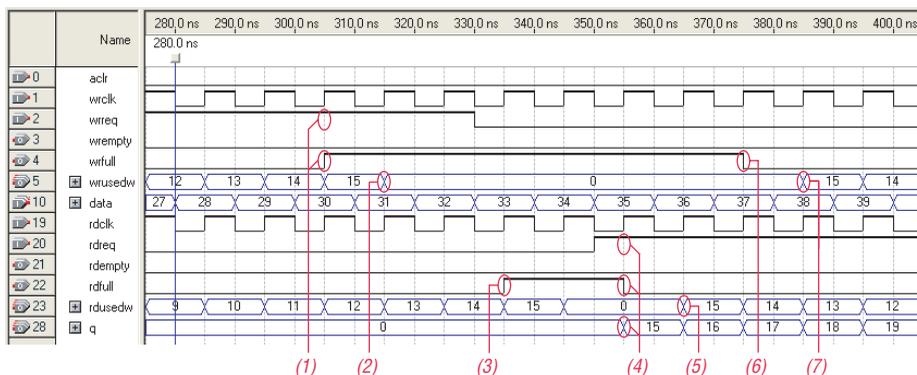
Figure 2–14 shows the functional simulation results for an initial write operation to an empty FIFO from 0 ns to 150 ns. It describes the latency behaviors for the `wrempty`, `rdempty`, `rdusedw[]`, and `wrusedw[]` signals.

Figure 2–14. Initial Write Operation to an Empty Dual-Clock FIFO (Legacy Mode)**Notes to Figure 2–14:**

- (1) At 20 ns, the `wrreq` signal is asserted.
- (2) At 25 ns, the data value 2 is written into the FIFO. The `wrempty` signal is deasserted on the same rising `wrclk` edge, 0 `wrclk` cycles after the `wrreq` signal is asserted at 20 ns (latency is `wrreq` to `wrempty`: 0 `wrclk` cycles).
- (3) At 35 ns, the `wrusedw []` signal increments from 0 to 1 (latency is `wrreq` to `wrusedw []`: 1 `wrclk` cycle).
- (4) At 55 ns, one `wrclk` cycle and then two rising `rdclk` edges after the `wrreq` signal detection at 25 ns, the `rdempty` signal is deasserted (latency is `wrreq` to `rdempty`: 1 `wrclk` cycle + next 2 rising `rdclk` edges).
- (5) At 65 ns, one `wrclk` cycle and then three rising `rdclk` edges after the `wrreq` signal detection at 25 ns, the `rdusedw []` counter increments from 0 to 1 (latency is `wrreq` to `rdusedw []`: 1 `wrclk` cycle + next 3 rising `rdclk` edges).
- (6) At 30 ns, the `rdreq` signal is asserted.
- (7) At 35 ns and 45 ns, read operations should occur, because `rdreq` remains asserted. However, the `rdempty` signal is still asserted at these times, and so the underflow circuitry checking, which is enabled, prevents the read operations.
- (8) At 140 ns, the `aclr` signal is asserted. Immediately, the `wrempty` and `rdempty` signals are asserted and the `wrusedw []` and `rdusedw []` signals reset to 0.

Read and Write Operations to an Almost Full Dual-Clock FIFO (Legacy Mode): Simulation Results (280 ns to 400 ns)

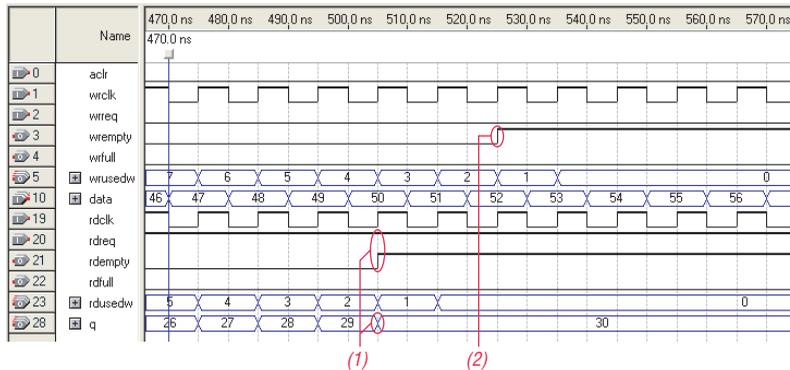
Figure 2–15 shows the functional simulation results for read and write operations into an almost full dual-clock FIFO from 280 ns to 400 ns. It shows the detailed behaviors of the `wrfull`, `rdfull`, `wrusedw []`, `rdusedw []`, `wrempty`, and `rdempty` signals during both read and write operations.

Figure 2–15. Read and Write Operations to an Almost Full Dual-Clock FIFO (Legacy Mode)**Notes to Figure 2–15:**

- (1) At 305 ns, the FIFO is full after 16 words are written to it. The **wrfull** signal is asserted, zero **wrclk** cycles after the final **wrreq** signal is detected (latency is **wrreq** to **wrfull**: 0 **wrclk** cycles).
- (2) At 315 ns, the **wrusedw** [] signal increments (modulo 16) from 15 to 0.
- (3) At 335 ns, the **rdfull** signal is finally asserted, one **wrclk** cycle and then the next two rising edges of **rdclk** after the final **wrreq** signal (latency is **wrreq** to **rdfull**: 1 **wrclk** cycle + next 2 **rdclk** rising edges).
- (4) At 355 ns, a read operation occurs and **q** [] outputs the first data word in the FIFO, which is data value 15 (as written to the FIFO at 150 ns). The **rdfull** signal is deasserted immediately (latency is **rdreq** to **rdfull**: 0 **rdclk** cycles).
- (5) At 365 ns, the **rdusedw** [] signal decrements (modulo 16) from 0 to 15, one **rdclk** cycle after the original **rdreq** assertion (at 350 ns) (latency is **rdreq** to **rdusedw** []: 1 **rdclk** cycle).
- (6) At 375 ns, the **wrfull** signal is finally deasserted, one **rdclk** cycle and then the next rising edge of **wrclk** after the original **rdreq** assertion (at 350 ns) (latency is **rdreq** to **wrfull**: 1 **rdclk** cycle + next **wrclk** rising edge).
- (7) At 385 ns, the **wrusedw** [] signal finally decrements (modulo 16) from 0 to 15, one **rdclk** cycle and then the next two rising edges of **wrclk** after the original **rdreq** assertion (at 350 ns) (latency is **rdreq** to **wrusedw** []: 1 **rdclk** cycle + next 2 rising **wrclk** edges).

**Read Operation to an Almost Empty Dual-Clock FIFO (Legacy Mode):
Simulation Results (470 ns to 570 ns)**

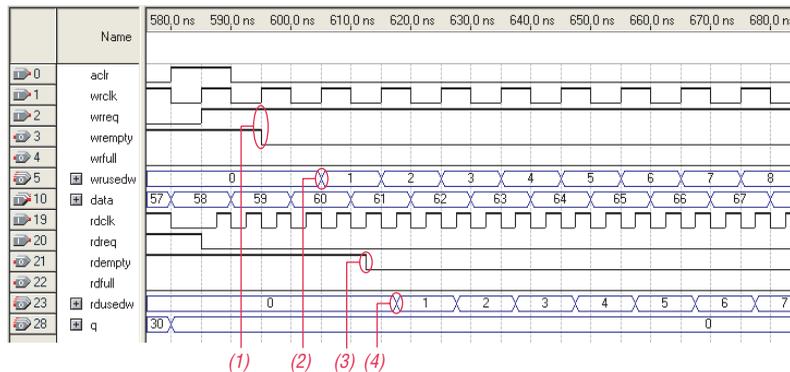
Figure 2–16 shows the functional simulation results for a read operation to an almost empty dual-clock FIFO from 470 ns to 570 ns. It shows the detailed behavior of the **wrempty** and **rdempty** signals during a sequence of read operations that continues until the FIFO is empty.

Figure 2–16. Read Operation to an Almost Empty Dual-Clock FIFO (Legacy Mode)**Notes to Figure 2–16:**

- (1) At 505 ns, the FIFO is empty and the rdempty signal is asserted. The q[] port outputs data value 30, the final data word in the FIFO (latency is rdreq to rdempty: 0 rdclk cycles).
- (2) At 525 ns, the wrempty signal is finally deasserted, one rdclk cycle and then the next rising edge of wrclk after the final rdreq signal is detected (at 500 ns) (latency is rdreq to wrempty: 1 rdclk cycle + next rising wrclk edge).

wrempty, rdempty, rdusedw[] and wrusedw[] Behaviors in a Dual-Clock FIFO (Legacy Mode) When rdclk is Faster Than wrclk: Simulation Results (580 ns to 680 ns)

Figure 2–17 shows functional simulation results when the rdclk is twice as fast as the wrclk. It illustrates the behavior of the wrempty, rdempty, rdusedw[], and wrusedw[] signals from 580 ns to 680 ns.

Figure 2–17. Status Flags in a Dual-Clock FIFO (Legacy Mode) When rdclk is Faster Than wrclk

Notes to Figure 2–17:

- (1) At 595 ns, a write operation occurs and the first data value is written to the FIFO. The `wrempty` signal is deasserted immediately (latency is `wrreq` to `wrempty`: 0 `wrclk` cycles).
- (2) At 605 ns, one `wrclk` cycle after the original `wrreq` assertion (at 585 ns), the `wrusedw` [] signal increments from 0 to 1 (latency is `wrreq` to `wrusedw` []: 1 `wrclk` cycle).
- (3) At 612.5 ns, the `rdempty` signal is finally deasserted, one `wrclk` cycle and then two rising edges of `rdclk` after the original write operation (at 595 ns) (latency is `wrreq` to `rdempty`: 1 `wrclk` cycle + next 2 rising `rdclk` edges).
- (4) At 617.5 ns, the `rdusedw` [] signal increments from 0 to 1, one `wrclk` cycle and then three rising edges of `rdclk` after the original write operation (at 595 ns) (latency is `wrreq` to `rdusedw` []: 1 `wrclk` cycle + next 3 rising `rdclk` edges).

As shown in Figure 2–17, when `rdclk` is faster than `wrclk`, the latency of `rdusedw` [] and `rdempty` with respect to `wrclk` appears faster. The `rdfull` signal is also asserted more quickly with respect to `wrclk`. However, the latency specification is valid regardless of the `rdclk` and `wrclk` frequencies.

If `rdclk` is faster than `wrclk`, the `rdusedw` [] counter holds a more accurate word count for the FIFO, because it is updated more frequently than the `wrusedw` [] signal. However, if `wrclk` is many times faster than `rdclk`, the `wrusedw` [] signal holds a more accurate word count for the FIFO than does the `rdusedw` [] signal.

Example 4: dcfifo Megafunction Behavior in Show-Ahead Mode

Table 2–8 shows the MegaWizard Plug-In Manager configuration settings that were used to create this design example, a dual-clock FIFO in Show-ahead mode.

<i>Table 2–8. Configuration Settings for a dcfifo in Show-Ahead Mode (Optimized for Area)</i>	
MegaWizard Configuration Setting	Value
Family	Stratix III
FIFO data width	8 bits
FIFO output width	8 bits
FIFO data depth	16 words
Common clock for reading and writing	No
FIFO Clocks are synchronized	No
Output control signals	Enable all the control signals (e.g. <code>full</code> , <code>empty</code> , <code>usedw[]</code> , <code>aclr</code>) for both the read and write clock domains.
FIFO Mode	Show-ahead
RAM block type	Auto
Optimization	Area
Disable overflow & underflow checking	No
Implement FIFO in logic cells	No

As described in the section “[General Description](#)” on page 1–3, the behavior of the `dcfifo` megafunction in Legacy mode and in Show-ahead mode is identical, except that in Show-ahead mode the `rdreq` signal operates as a read-acknowledge signal. Therefore, the `q[]` bus outputs the first FIFO data word regardless of whether a read operation occurs. The `q[]` bus outputs different values for a FIFO in Legacy mode and a FIFO in Show-ahead mode.

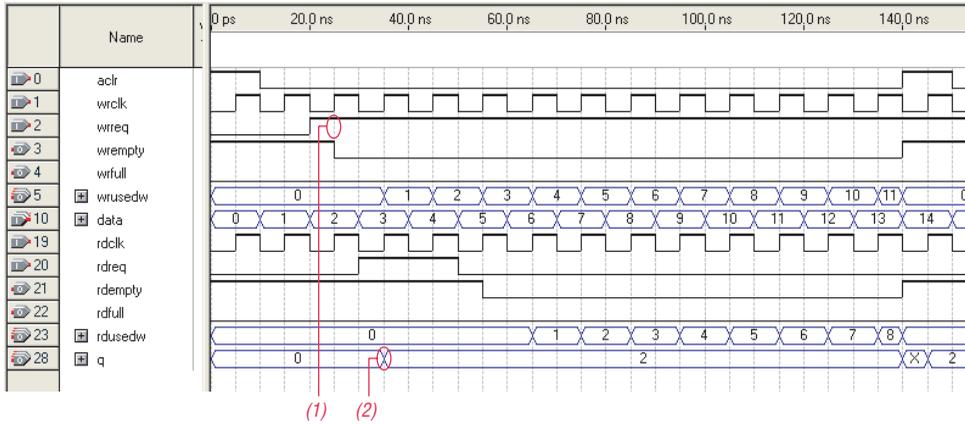
The FIFO flag signals may also have different latency in the two different FIFO modes. For complete latency information for different families and configuration settings, refer to [Table 3–8](#) on page 3–15.

Initial Write Operation to an Empty Dual-Clock FIFO (Show-Ahead Mode): Simulation Results (0 ns to 150 ns)

Figure 2–18 shows the functional simulation results for an initial write operation to an empty FIFO from 0 ns to 150 ns. It shows the latency behaviors for the `wempty`, `rdempty`, `rdusedw[]`, and `wrusdew[]` signals.

The behavior of the initial write operation to an empty dual-clock FIFO in Show-ahead mode or in Legacy mode is identical except for the behavior of the $q[]$ signal. **Figure 2–18** highlights the differences between the behaviors of a dual-clock FIFO in Show-ahead mode and in Legacy mode.

Figure 2–18. Initial Write Operation to an Empty Dual-Clock FIFO (Show-Ahead Mode)



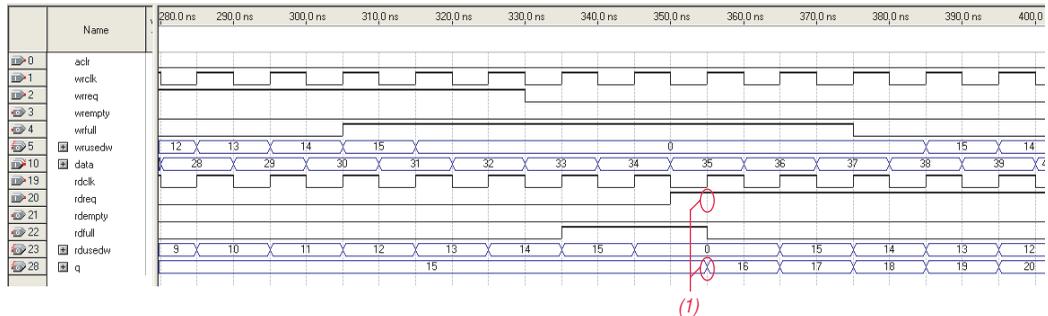
Notes to Figure 2–18:

- (1) At 25 ns, data value 2 is written into the FIFO.
- (2) At 35 ns, the $q[]$ port outputs the data value 2 after one $rdclk$ cycle. In Legacy mode, the $q[]$ port would output the data value only when a read operation occurs.

Read and Write Operations to an Almost Full Dual-Clock FIFO (Show-Ahead Mode): Simulation Results (280 ns to 400 ns)

Figure 2–19 shows the functional simulation results for read and write operations into an almost full dual-clock FIFO from 280 ns to 400 ns. It shows the detailed behaviors of the $wrfull$, $rdfull$, $wrusedw[]$, $rdusedw[]$, $wrempty$, and $rdempty$ signals during both read and write operations.

The behavior of read and write operations in an almost-full dual-clock FIFO in Show-ahead mode or in Legacy mode is identical, except for the $q[]$ bus behavior. **Figure 2–19** highlights the differences as compared to the dual-clock FIFO in legacy mode.

Figure 2–19. Read and Write Operations to an Almost Full Dual-Clock FIFO (Show-Ahead Mode)**Note to Figure 2–19:**

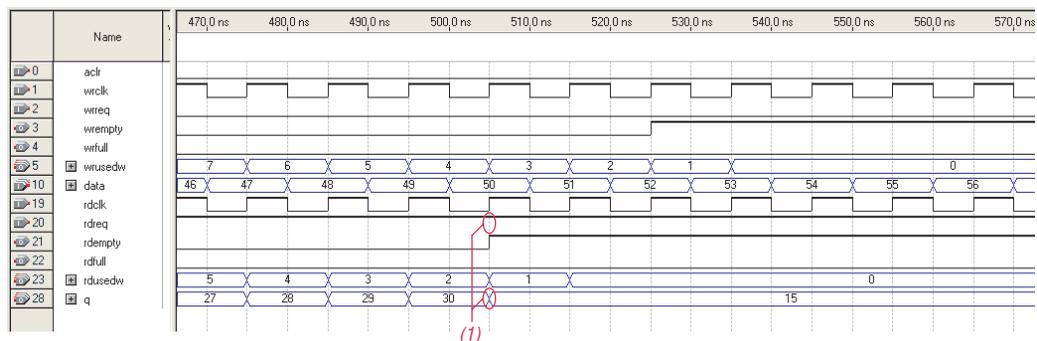
- (1) At 355 ns, the first read operation occurs. Immediately, the **q []** port outputs the second data value in the FIFO (which was written to the FIFO at 164 ns). In Legacy mode, the **q []** port would output the first data value in the FIFO.

Read Operation to an Almost Empty Dual-Clock FIFO (Show-Ahead Mode): Simulation Results (470 ns to 570 ns)

Figure 2–20 shows the functional simulation results for a read operation to an almost empty dual-clock FIFO from 470 ns to 570 ns. It shows the detailed behaviors of the **wrenpty** and **rdempty** signals during this read operation until the FIFO is empty.

The behavior of an almost-empty dual-clock FIFO in Show-ahead mode and that of an almost-empty dual-clock FIFO in Legacy mode during read and write operations is identical except for the **q []** signal behavior.

Figure 2–20 highlights the behavior of the dual-clock FIFO in Legacy mode and compares it to the behavior in Show-ahead mode.

Figure 2–20. Read Operation to an Almost Empty Dual-Clock FIFO (Show-Ahead Mode)

Notes to Figure 2–20:

- (1) At 505 ns, the FIFO is empty following a final read operation. The `q []` port outputs the results of this final read operation, the first data word in the FIFO, data value 15, and maintains this value until a change in the FIFO status.
- (2) In Legacy mode, the `q []` port would output this final remaining data value, data value 30, only after the read operation completes.

Example 5: dcfifo Megafunction Behavior with Wide Write Data and Narrow Read Output

Table 2–9 shows the MegaWizard Plug-In Manager configuration settings that were used to create this design example, a dual-clock FIFO in Legacy mode with a wide write-data port (width 16 bits) and a narrow read-output port (width 8 bits).

Table 2–9. Configuration Settings for a Dual-Clock FIFO with Wider Write Than Read (Optimized for Area) (Part 1 of 2)

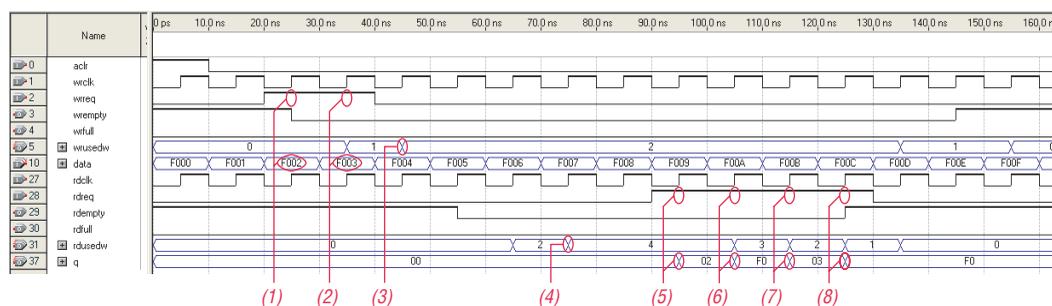
MegaWizard Configuration Setting	Value
Family	Stratix III
FIFO data width	16 bits
FIFO output width	8 bits
FIFO data depth	16 words
Common clock for reading and writing	No
FIFO clocks are synchronized	No
Output control signals	Enable all the control signals (e.g. <code>full</code> , <code>empty</code> , <code>usedw []</code> , <code>aclr</code>) for both the read and write clock domains.
FIFO Mode	Legacy
RAM block type	Auto

Table 2–9. Configuration Settings for a Dual-Clock FIFO with Wider Write Than Read (Optimized for Area) (Part 2 of 2)

Optimization	Area
Disable overflow & underflow checkings	No
Implement FIFO in logic cells	No

Figure 2–21 shows the functional simulation results for read and write operations to a dual-clock FIFO with a 16-bit wide write port and an 8-bit wide read port from 0 ns to 160 ns.

Figure 2–21. Read and Write Operations with Wide Write and Narrow Read



Notes to Figure 2–21:

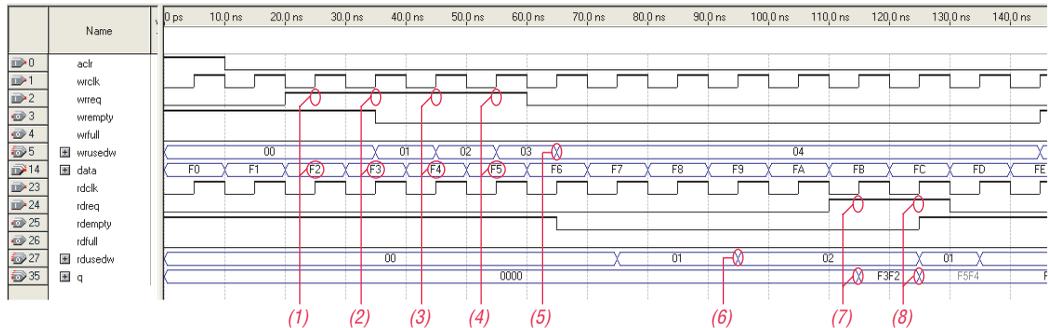
- At 25 ns, the 16-bit data value 0xF002 is written to the FIFO.
- At 35 ns, the 16-bit data value 0xF003 is written to the FIFO.
- At 45 ns, the `wrusedw` signal increments to value 2: the FIFO contains two 16-bit words.
- At 75 ns, the `rdusedw` signal increments to value 4: the FIFO contains four 8-bit words.
- At 95 ns, the `q` port outputs data value 0x02, the least significant 8 bits of the 16-bit word written at 25 ns.
- At 105 ns, the `q` port outputs data value 0xF0, the most significant 8 bits of the 16-bit word written at 25 ns.
- At 115 ns, the `q` port outputs data value 0x03, the least significant 8 bits of the 16-bit word written at 35 ns.
- At 125 ns, the `q` port outputs data value 0xF0, the most significant 8 bits of the 16-bit word written at 35 ns.

Example 6: dcfifo Megafunction Behavior with Narrow Write Data and Wide Read Output

Table 2–10 shows the MegaWizard Plug-In Manager configuration settings that were used to create this design example, a dual-clock FIFO in Legacy mode with a narrow write-data port (width 8 bits) and a wide read-output port (width 16 bits).

MegaWizard Configuration Setting	Value
Family	Stratix III
FIFO data width	8 bits
FIFO output width	16 bits
FIFO data depth	16 words
Common clock for reading and writing	No
FIFO clocks are synchronized	No
Output control signals	Enable all the control signals (e.g. <code>full</code> , <code>empty</code> , <code>usedw[]</code> , <code>aclr</code>) for both the read and write clock domains.
FIFO Mode	Legacy
RAM block type	Auto
Optimization	Area
Disable overflow & underflow checkings	No
Implement FIFO in logic cells	No

Figure 2–22 shows the functional simulation results for read and write operations to a dual-clock FIFO with an 8-bit wide write port and a 16-bit wide read port from 0 ns to 140 ns.

Figure 2–22. Read and Write Operations with Narrow Write and Wide Read**Notes to Figure 2–22:**

- (1) At 25 ns, the 8-bit data value 0xF2 is written to the FIFO.
- (2) At 35 ns, the 8-bit data value 0xF3 is written to the FIFO.
- (3) At 45 ns, the 8-bit data value 0xF4 is written to the FIFO.
- (4) At 55 ns, the 8-bit data value 0xF5 is written to the FIFO.
- (5) At 65 ns, the `wrusedw []` signal increments to value 4: the FIFO contains four 8-bit words.
- (6) At 95 ns, the `rdusedw []` signal increments to value 2: the FIFO contains two 16-bit words.
- (7) At 115 ns, the `q []` port outputs 16-bit data value 0xF3F2. Data word 0xF2 is the 8-bit data value written to the FIFO at 25 ns; 0xF3 is the 8-bit data value written to the FIFO at 35 ns.
- (8) At 125 ns, the `q []` port outputs 16-bit data value 0xF5F4. Data word 0xF4 is the 8-bit data value written to the FIFO at 45 ns; 0xF5 is the 8-bit data value written to the FIFO at 55 ns.

Specifications for the scfifo Megafunction

This section describes the design requirements, ports and parameters, and output status flag latencies for single-clock FIFOs designed using the `scfifo` megafunction.

Design Requirements for the scfifo Megafunction

Designing a single-clock FIFO using the `scfifo` megafunction requires that you understand the megafunction's design constraints. This section summarizes the design requirements for a single-clock FIFO designed using the `scfifo` megafunction.

The following design requirements are enforced by the `scfifo` megafunction:

- The FIFO depth must be a power of two (2^N for some integer N).
- The maximum FIFO depth is limited by the available memory in the device.
- The minimum FIFO depth is 4 (N is greater than 1).
- The FIFO is full when 2^N words contain data.
- Writing to a full FIFO can corrupt the FIFO data when overflow checking is disabled.
 - When overflow checking is enabled, the write request is ignored.
 - However, when overflow checking is disabled, the FIFO uses fewer device resources.
- Reading from an empty FIFO can return unknown data and produce unpredictable future results when underflow checking is disabled.
 - When underflow checking is enabled, the read request is ignored.
 - However, when overflow checking is disabled, the FIFO uses fewer device resources.
- The `aclr` and `sclr` signals clear the FIFO flags but not the `q[]` output port for Arria™, Stratix®, and Cyclone® series of devices. For earlier series of devices, both signals reset `q[]` output port to 0.
- When the `aclr` signal is used, Altera recommends enabling the **Removal and Recovery Analysis** option in the timing analyzer for all device families.



For more information about removal and recovery, refer to the *Quartus II TimeQuest Timing Analyzer* chapter or the *Quartus II Classic Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

- When `full` is asserted, the `wrreq` is expected to be deasserted in the same clock cycle to prevent writing invalid data
 - ✓ External logic should not latch the `full` signal before it deasserts the `wrreq` signal.
- When `empty` is asserted, the `rdreq` signal is expected to be deasserted in the same clock cycle to prevent reading invalid data
 - ✓ External logic should not latch the `empty` signal before it deasserts the `rdreq` signal.
- When the FIFO is full, the `usedw []` signal increments to 0 instead of showing the true FIFO depth, 2^N . The value of `usedw []` is represented in N bits, restricting it to a maximum value of $2^N - 1$.

SCFIFO Ports and Parameters

Table 3–1 describes the input ports of the `scfifo` megafunction.

Table 3–2 describes the output ports of the `scfifo` megafunction.

Table 3–3 describes the parameters for the `scfifo` megafunction.



Refer to the latest version of the Quartus® II Help for the most current information on the ports and parameters for the `scfifo` megafunction.

The parameter details are only relevant for users who bypass the MegaWizard® Plug-In Manager interface and use the megafunction as a directly parameterized instantiation in their design. The details of these parameters are hidden from MegaWizard Plug-In Manager interface users.

Table 3–1. Single-Clock FIFO Megafunction Input Ports (Part 1 of 2)

Port Name	Required	Description	Comments
<code>data []</code>	Yes	Data input to the <code>scfifo</code>	The <code>data []</code> port is <code>LPM_WIDTH</code> bits wide.
<code>clock</code>	Yes	Positive-edge-triggered clock	—
<code>wrreq</code>	Yes	Write request. The <code>data []</code> port is written to the FIFO.	When <code>wrreq = 1</code> , the FIFO stores the data from the <code>data []</code> port at the next rising edge of the clock. Unless the overflow checking circuitry is disabled, writing is disabled if <code>wrfull = 1</code> .

Port Name	Required	Description	Comments
<code>rdreq</code>	Yes	Read request. The oldest data in the FIFO goes to the <code>q[]</code> port.	For FIFOs operating in Legacy mode, when <code>rdreq = 1</code> , the first data in the FIFO is available at the <code>q[]</code> output port on the rising clock edge. For megafunctions operating in Show-ahead mode, when <code>rdreq</code> is asserted a read is acknowledged and the <code>q[]</code> output port is updated with the next data in the FIFO. Reading is disabled if <code>empty = 1</code> , unless the underflow checking circuitry is disabled.
<code>aclr</code>	No	Asynchronous clear input to empty the FIFO	The <code>aclr</code> signal clears the FIFO flags but does not clear the <code>q[]</code> output port, in the Arria, Stratix, and Cyclone series of devices. For other, earlier device families, both the <code>aclr</code> and the <code>sclr</code> signals reset the <code>q[]</code> output port to 0.
<code>sclr</code>	No	Synchronous clear input to empty the FIFO	The <code>sclr</code> signal clears the FIFO flags but does not clear the <code>q[]</code> output port, in the Arria, Stratix, and Cyclone series of devices. For other, earlier device families, both the <code>aclr</code> and the <code>sclr</code> signals reset the <code>q[]</code> output port to 0.

Table 3–2 describes the output ports of the `scfifo` megafunction.

Port Name	Required	Description	Comments
<code>q[]</code>	Yes	Data output from the <code>scfifo</code>	The output port is <code>LPM_WIDTH</code> bits wide.
<code>full</code>	No	Indicates that the FIFO is full	Asserted when <code>usedw[] = LPM_NUMWORDS</code> . When asserted, the FIFO is considered full, and the <code>wreq</code> port is disabled unless the overflow checking circuitry is disabled.
<code>empty</code>	No	Indicates that the FIFO is empty or is not ready for a read request	Asserted when the FIFO is empty, or when the FIFO is not ready for a read request. When asserted, the FIFO is considered empty, and the <code>rdreq</code> port is disabled unless the underflow checking circuitry is disabled.

Port Name	Required	Description	Comments
almost_full	No	Indicates that the FIFO is almost full	Asserted when the value of <code>usedw[]</code> is greater than or equal to <code>ALMOST_FULL_VALUE</code> . When compared to the <code>empty</code> and <code>full</code> output ports, the <code>usedw[]</code> port has an additional latency of 1 or 2 clock cycles. Thus, the <code>usedw[]</code> port shows the state of the FIFO slower than do the <code>empty</code> and <code>full</code> ports. When using the <code>almost_full</code> port, specify the <code>ALMOST_FULL_VALUE</code> value.
almost_empty	No	Indicates that the FIFO is almost empty	Asserted when the value of <code>usedw[]</code> is less than <code>ALMOST_EMPTY_VALUE</code> . When compared to the <code>empty</code> and <code>full</code> output ports, the <code>usedw[]</code> port has an additional latency of 1 or 2 clock cycles. Thus, the <code>usedw[]</code> port shows the state of the FIFO slower than do the <code>empty</code> and <code>full</code> ports. When using the <code>almost_empty</code> port, specify the <code>ALMOST_EMPTY_VALUE</code> value.
usedw[]	No	The number of words that are currently in the FIFO	The width (in bits) of this output port is the ceiling of $\log_2(\text{LPM_NUMWORDS})$. If the value of <code>LPM_NUMWORDS</code> is a power of 2, you can use the <code>full</code> output as an extra bit and append it to <code>usedw[]</code> .

Table 3–3 describes the parameters for the `scfifo` megafunction.

Parameter	Required	Type	Description
<code>LPM_WIDTH</code>	Yes	Integer	Specifies the width of the <code>data[]</code> and <code>q[]</code> ports.
<code>LPM_NUMWORDS</code>	Yes	Integer	Specifies the depth of the FIFO, which is always a power of 2.
<code>LPM_WIDTHHU</code>	Yes	Integer	(VHDL only) Specifies the width of the <code>usedw[]</code> port. The recommended value for this port is the ceiling of $\log_2(\text{LPM_NUMWORDS})$.
<code>ALMOST_FULL_VALUE</code>	No	Integer	The threshold value for the <code>almost_full</code> port
<code>ALMOST_EMPTY_VALUE</code>	No	Integer	The threshold value for the <code>almost_empty</code> port
<code>LPM_SHOWAHEAD</code>	No	String	Specifies whether the FIFO is in Legacy mode or in Show-ahead mode. In Show-ahead mode, data appears on <code>q[]</code> before <code>rdreq</code> is asserted. Values are "ON" (in Show-ahead mode) or "OFF" (in Legacy mode). If omitted, the default is "OFF". Specifying "ON" may reduce performance.

Table 3–3. Single-Clock FIFO Megafunction Parameters (Part 2 of 2)

Parameter	Required	Type	Description
ALLOW_RWCYCLE_WHEN_FULL	No	String	Allows combined read/write cycles to an already full FIFO, so that it remains full. Values are "ON" or "OFF". If omitted, the default is "OFF". This parameter is used only when the OVERFLOW_CHECKING parameter is set to "ON".
OVERFLOW_CHECKING	No	String	Altera-specific parameter. Disables the overflow-checking logic when set to "OFF", so that full is no longer checked before a write transaction. Values are "ON" or "OFF". If omitted, the default is "ON". Note: Writing to a full FIFO produces unpredictable results.
UNDERFLOW_CHECKING	No	String	Altera-specific parameter. Disables the underflow-checking logic when set to "OFF", so that empty is no longer checked before a read transaction. Values are "ON" or "OFF". If omitted, the default is "ON". Note: Reading from an empty FIFO produces unpredictable results.
ADD_RAM_OUTPUT_REGISTER	No	String	Specifies whether to register the RAM output. Values are "ON" and "OFF". If omitted, the default is "OFF".
LPM_HINT	No	String	Allows you to assign Altera-specific parameters in VHDL Design Files (.vhd). If omitted, the default is "UNUSED".
LPM_TYPE	No	String	Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files.
USE_EAB	No	String	Altera-specific parameter. Values are "ON", "OFF", and "UNUSED". You must use the LPM_HINT parameter to specify the USE_EAB parameter in VHDL design files. Setting the USE_EAB parameter to OFF prevents the Quartus II software from using Embedded System Blocks (ESBs) to implement the logic in APEX™ II, APEX 20K, and Mercury™ devices, or Embedded Array Blocks (EABs) in ACEX 1K® and FLEX 10K® devices; the software can only use registers or latches. (The "ON" setting is not useful in memory functions: the Quartus II software automatically implements memory functions in ESBs or EABs by default.) This parameter is not available for simulation with other EDA simulators and for Stratix, Stratix GX, HardCopy® Stratix, MAX® 7000 series, MAX A3000, and FLEX® 6000 devices.
INTENDED_DEVICE_FAMILY	No	String	This parameter is used for modeling and behavioral simulation purposes. Configure the scfifo megafunction with the MegaWizard Plug-In Manager to calculate the value for this parameter.

Flag Latency in the scfifo Megafunction

Table 3–4 shows the flag latencies in single-clock FIFOs created using the `scfifo` megafunction with different mode and optimization settings and in different device families.

<i>Table 3–4. Flag Latencies in the scfifo Megafunction</i>			
Mode	Optimized for	Latencies in APEX and FLEX Devices	Latencies in Stratix III, Stratix II, Stratix II GX, Stratix, Stratix GX, Cyclone III, Cyclone II, and Cyclone Devices
Legacy mode	Speed	wrreq to empty: 0 clock cycles rdreq to empty: 0 clock cycles wrreq/rdreq to usedw []: 0 clock cycles wrreq/rdreq to full: 0 clock cycles	wrreq to empty: 1 clock cycle rdreq to empty: 0 clock cycles wrreq/rdreq to usedw []: 0 clock cycles wrreq/rdreq to full: 0 clock cycles
Legacy mode	Area	Same as with speed optimization	wrreq to empty: 0 clock cycles rdreq to empty: 0 clock cycles wrreq/rdreq to usedw []: 0 clock cycles wrreq/rdreq to full: 0 clock cycles
Show-ahead mode	Speed	Same as in Legacy mode	wrreq to empty: 2 clock cycles rdreq to empty: 0 clock cycles wrreq/rdreq to usedw []: 0 clock cycles wrreq/rdreq to full: 0 clock cycles

Table 3–4. Flag Latencies in the scfifo Megafunction

Show-ahead mode	Area	Same as in Legacy mode	wrreq to empty: 1 clock cycle rdreq to empty: 0 clock cycles wrreq/rdreq to usedw[]: 0 clock cycles wrreq/rdreq to full: 0 clock cycles
-----------------	------	------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------

Specifications for the dcfifo Megafunction

This section describes the design requirements, ports and parameters, and output status flag latencies for dual-clock FIFOs designed using the `dcfifo` megafunction.

Design Requirements for the dcfifo Megafunction

Designing a dual-clock FIFO using the `dcfifo` megafunction requires that you understand the megafunction's design constraints. This section summarizes the design requirements for a single-clock FIFO designed using the `dcfifo` megafunction.

The following design requirements are enforced by the `dcfifo` megafunction:

- The FIFO depth must be a power of two (2^N for some integer N).
- The maximum FIFO depth is limited by the available memory in the device.
- The minimum FIFO depth is 4 (N is greater than 1).
- The FIFO is full when 2^N words contain data in Stratix II devices and in device families released after the Stratix II family of devices.
- The FIFO is full when between 2^N-3 and 2^N words contain data in device families released prior to the Stratix II device family (Stratix, APEX, and FLEX family devices).
 - In these families, the `wrfull` signal is estimated conservatively before the FIFO is completely full. As many as three words of FIFO memory are not available for write transactions.
 - Due to this conservative estimation, you should configure your FIFO to be at least three words larger than the FIFO size you need.
- Writing to a full FIFO can corrupt the FIFO data when overflow checking is disabled.
 - When overflow checking is enabled, the write request is ignored.

- However, when overflow checking is disabled, the FIFO uses fewer device resources.
- Reading from an empty FIFO can return unknown data and produce unpredictable future results when underflow checking is disabled.
 - When underflow checking is enabled, the read request is ignored.
 - However, when overflow checking is disabled, the FIFO uses fewer device resources.
- The `wrreq` signal must be low when the `aclr` signal is deasserted at the rising `wrclk` edge.
 - ✓ This condition may occur when the device enters user mode or during an asynchronous clear cycle when the `wrreq` signal is high. Without the additional constraint on the `wrreq` signal, a race condition may occur, corrupting the FIFO data.
- In Stratix II and Cyclone II devices, you must enable the **Removal and Recovery Analysis** option in the timing analyzer for correct timing analysis if a `dcfifo` megafunction exists in the design, whether or not it uses the `aclr` signal. In addition, if a `dcfifo` megafunction exists in a design in any other device, Altera recommends enabling the **Removal and Recovery Analysis** option in the timing analyzer, if the `aclr` signal is used.



For more information about removal and recovery, refer to the *Quartus II TimeQuest Timing Analyzer* chapter or the *Quartus II Classic Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

- When `wrfull` is asserted, the `wrreq` signal is expected to be deasserted in the same `wrclk` cycle to prevent writing invalid data
 - ✓ External logic should not latch the `wrfull` signal before it deasserts the `wrreq` signal.
- When `rdempty` is asserted, the `rdreq` signal is expected to be deasserted in the same `rdclk` cycle to prevent reading invalid data
 - ✓ External logic should not latch the `rdempty` signal before it deasserts the `rdreq` signal.
- When the FIFO is full, the `usedw []` signals increment to 0 instead of showing the true FIFO depth, 2^N . The value of `usedw []` is represented in N bits, restricting it to a maximum value of 2^N-1 . If the **Add extra MSB to usedw[] port** configuration setting is turned on, the `usedw []` port shows 2^N instead of 0 when the FIFO is full.
- When the `wrusedw []` and `rdusedw []` signals are not used, the FIFO uses fewer device resources.

- The FIFO flags may have latency due to the pipeline synchronization between the read address pointer and the write address pointer. The latency may vary in different families and for different megafunction configuration settings. [Table 3–8](#) shows the latency information for all the FIFO flags.
- To reduce the FIFO flag latencies, you can decrease the values of the `RDSYNC_DELAYPIPE` and `WRSYNC_DELAYPIPE` parameters. However, Altera recommends you use the default values to avoid metastability.
- Setup and hold-time violations that occur on the synchronization pipeline registers (`*dcfifo*dffpipe*`) can be safely ignored.

In the Classic Timing Analyzer, these paths are cut by default and therefore no timing violations are reported.

However, in the TimeQuest Timing Analyzer, these paths are not cut automatically. To remove the resulting setup and hold-time violations, you must apply the **Set False Path** timing constraint to explicitly cut these timing paths.

DCFIFO Ports and Parameters

[Table 3–5](#) describes the input ports of the `dcfifo` megafunction. [Table 3–6](#) describes the output ports of the `dcfifo` megafunction. [Table 3–7](#) describes the parameters for the `scfifo` megafunction.



Refer to the latest version of the Quartus® II Help for the most current information on the ports and parameters for the `dcfifo` megafunction.

The parameter details are only relevant for users who bypass the MegaWizard Plug-In Manager interface and use the megafunction as a directly parameterized instantiation in their design. The details of these parameters are hidden from MegaWizard Plug-In Manager interface users.

<i>Table 3–5. Dual-Clock FIFO Megafunction Input Ports (Part 1 of 2)</i>			
Port Name	Required	Description	Comments
<code>data []</code>	Yes	Data input to the <code>dcfifo</code>	The <code>data []</code> port is <code>LPM_WIDTH</code> bits wide.
<code>rdclk</code>	Yes	Positive-edge-triggered clock	Read transactions to the FIFO are synchronous with this clock.
<code>wrclk</code>	Yes	Positive-edge-triggered clock.	Write transactions to the FIFO are synchronous with this clock.

Table 3–5. Dual-Clock FIFO Megafunction Input Ports (Part 2 of 2)

Port Name	Required	Description	Comments
wrreq	Yes	Write request. The data [] port is written to the FIFO.	When wrreq = 1, the FIFO stores the data from the data [] port at the next rising edge of the clock. Unless the overflow checking circuitry is disabled, writing is disabled if wrfull = 1. Note: Do not assert the wrreq port on when the device enters user mode or during an asynchronous clear cycle. Specifically, the wrreq port must be low when aclr transitions from high to low, or when the device comes out of power-on-reset. Violating this requirement creates a race condition between the falling edge of aclr and the rising edge of the write clock if wrreq is high. If this race condition occurs, some of the write counter bits may transition while others do not, potentially resulting in an inconsistent write counter state.
rdreq	Yes	Read request. The oldest data in the FIFO goes to the q [] port.	For FIFOs operating in Legacy mode, when rdreq = 1, the first data in the FIFO is available at the q [] output port on the rising clock edge. For megafunctions operating in Show-ahead mode, when rdreq is asserted a read is acknowledged and the q [] output port is updated with the next data in the FIFO. Reading is disabled if empty = 1, unless the underflow checking circuitry is disabled.
aclr	No	Asynchronous clear input to empty the FIFO	The aclr signal clears the FIFO flags but does not clear the q [] output port for Arria GX, Stratix III, Stratix II, Stratix II GX, Stratix, Stratix GX, Cyclone III, Cyclone II, and Cyclone devices. For other, earlier device families, the aclr signal resets the q [] output port to 0.

Table 3–6 describes the output ports of the `dcfifo` megafunction.

Table 3–6. Dual-Clock FIFO Megafunction Output Ports			
Port Name	Required	Description	Comments
<code>q[]</code>	Yes	Data output from the <code>dcfifo</code>	The <code>q[]</code> port is <code>LPM_WIDTH</code> bits wide.
<code>rdempty</code>	No	Indicates that the FIFO is empty or is not ready for a read request	Asserted when <code>rdusedw[] = 0</code> , or when the FIFO is not ready for a read request. When asserted, the FIFO is considered empty, and the <code>rdreq</code> port is disabled unless the underflow checking circuitry is disabled. The <code>rdempty</code> port is synchronized with <code>rdclk</code> , and is deasserted some number of cycles after the first <code>wrreq</code> (see Table 3–8). The <code>rdempty</code> port is usually more accurate than the <code>wrempty</code> port.
<code>wrfull</code>	No	Indicates that the FIFO is full	Asserted when the FIFO is full. The value of <code>wrusedw[]</code> can be between <code>LPM_NUMWORDS-3</code> and <code>LPM_NUMWORDS</code> . When asserted, the FIFO is considered full, and the <code>wrreq</code> port is disabled unless the overflow checking circuitry is disabled. The <code>wrfull</code> port is synchronized with <code>wrcclk</code> . The <code>wrfull</code> port is usually more accurate than the <code>rdfull</code> port.
<code>wrempty</code>	No	Indicates that the FIFO is empty	The <code>wrempty</code> port is a delayed version of <code>rdempty</code> that is synchronized with <code>wrcclk</code> . Due to its longer latency, this signal is normally not used.
<code>rdfull</code>	No	Indicates that the FIFO is full	The <code>rdfull</code> port is a delayed version of <code>wrfull</code> that is synchronized with <code>rdrcclk</code> . Due to its longer latency, this signal is normally not used.
<code>rdusedw[]</code>	No	The number of words that are currently in the FIFO	This output port has width <code>LPM_WIDTHHU</code> bits and is synchronized with <code>rdclk</code> . If the Add extra MSB to usedw[] port configuration setting is turned on, the output port has width <code>LPM_WIDTHHU+1</code> . This port is required if <code>wrusedw[]</code> is used.
<code>wrusedw[]</code>	No	The number of words that are currently in the FIFO	This output port has width <code>LPM_WIDTHHU</code> bits and is synchronized with <code>wrcclk</code> . If the Add extra MSB to usedw[] port configuration setting is turned on, the output port has width <code>LPM_WIDTHHU+1</code> . This port is required if <code>rdusedw[]</code> is used.

Table 3-7 describes the parameters for the dcfifo megafunction.

Parameter	Required	Type	Description
LPM_WIDTH	Yes	Integer	Specifies the width of the <code>data []</code> and <code>q []</code> ports.
LPM_NUMWORDS	Yes	Integer	Specifies the depth of the FIFO, which is always a power of 2. The last three words of the FIFO may not be available for writing because of the synchronization pipelines between the two clock domains. These pipelines prevent internal metastability. However, they also cause the information in one clock domain about whether a read or write occurred to be temporarily unavailable in the other clock domain. The <code>rdfull</code> and <code>wrfull</code> ports of a FIFO must be asserted before the FIFO is completely full, to ensure data is not written into a full FIFO. This process may cause several words at the end of the FIFO to become unavailable. Depending on the rate of write transactions, the <code>rdfull</code> and <code>wrfull</code> ports may go high with three, two, or one word remaining unwritten in the FIFO. To ensure your FIFO can hold a specific number of words, specify a value for <code>LPM_NUMWORDS</code> that is three words greater than the number of FIFO entries you require.
LPM_WIDTHU	Yes	Integer	(VHDL only) Specifies the width of the <code>rdusedw []</code> and <code>wrusedw []</code> ports. The recommended value for this port is the ceiling of $\log_2(\text{LPM_NUMWORDS})$.
LPM_SHOWAHEAD	No	String	Specifies whether the FIFO is in Legacy mode or in Show-ahead mode. In Show-ahead mode, data appears on <code>q []</code> before <code>rdreq</code> is asserted. Values are "ON" (in Show-ahead mode) or "OFF" (in Legacy mode). If omitted, the default is "OFF". Specifying "ON" may reduce performance.
LPM_HINT	No	String	Allows you to assign Altera-specific parameters in VHDL Design Files (<code>.vhd</code>). If omitted, the default is "UNUSED".
LPM_TYPE	No	String	Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files.
OVERFLOW_CHECKING	No	String	Altera-specific parameter. You must use the <code>LPM_HINT</code> parameter to specify the <code>OVERFLOW_CHECKING</code> parameter in VHDL Design Files. Disables the overflow-checking logic when set to "OFF", so that <code>full</code> is no longer checked before a write transaction. Values are "ON" or "OFF". If omitted, the default is "ON". Note: Writing to a full FIFO produces unpredictable results.

Table 3–7. Dual-Clock Megafunction Parameters (Part 2 of 3)

Parameter	Required	Type	Description
UNDERFLOW_CHECKING	No	String	Altera-specific parameter. You must use the LPM_HINT parameter to specify the UNDERFLOW_CHECKING parameter in VHDL Design Files. Disables the underflow-checking logic when set to "OFF", so that empty is no longer checked before a read transaction. Values are "ON" or "OFF". If omitted, the default is "ON". Note: Reading from an empty FIFO produces unpredictable results.
DELAY_RDUSEDW	No	Integer	Altera-specific parameter. You must use the LPM_HINT parameter to specify the DELAY_RDUSEDW parameter in VHDL Design Files. Value is any integer greater than or equal to 0. Specifies the number of register stages that are added internally to the rdusedw [] port. The default value of 1 adds a single register stage to the output to improve its performance. This parameter is not available for simulation with other EDA simulators.
DELAY_WRUSEDW	No	Integer	Altera-specific parameter. You must use the LPM_HINT parameter to specify the DELAY_WRUSEDW parameter in VHDL Design Files. Value is any integer greater than or equal to 0. Specifies the number of register stages that are added internally to the wrusedw [] port. The default value of 1 adds a single register stage to the output to improve its performance. This parameter is not available for simulation with other EDA simulators.
RDSYNC_DELAYPIPE	No	Integer	Altera-specific parameter. You must use the LPM_HINT parameter to specify the RDSYNC_DELAYPIPE parameter in VHDL Design Files. Specifies the number of register stages that are added internally for synchronization from the write control logic to the read control logic. The default value of three is a good safeguard value that protects against the possibility of internal metastability when rdclk and wrclk are unrelated. This parameter is not available for simulation with other EDA simulators.

Table 3–7. Dual-Clock Megafunction Parameters (Part 3 of 3)

Parameter	Required	Type	Description
WRSYNC_DELAYPIPE	No	Integer	Altera-specific parameter. You must use the LPM_HINT parameter to specify the WRSYNC_DELAYPIPE parameter in VHDL Design Files. Specifies the number of register stages that are added internally for synchronization from the read control logic to the write control logic. The default value of 3 is a good safeguard value that protects against the possibility of internal metastability when rdclk and wrclk are unrelated. This parameter is not available for simulation with other EDA simulators.
USE_EAB	No	String	Altera-specific parameter. Values are "ON", "OFF", and "UNUSED". You must use the LPM_HINT parameter to specify the USE_EAB parameter in VHDL Design Files. Setting the USE_EAB parameter to OFF prevents the Quartus II software from using Embedded System Blocks (ESBs) to implement the logic in APEX II, APEX 20K, and Mercury devices, or Embedded Array Blocks (EABs) in ACEX 1K and FLEX 10K devices; the software can only use registers or latches. (The "ON" setting is not useful in memory functions: the Quartus II software automatically implements memory functions in ESBs or EABs by default.) This parameter is not available for simulation with other EDA simulators and for Stratix, Stratix GX, HardCopy Stratix, MAX 7000 series, MAX A3000, and FLEX 6000 devices.

Flag Latency in the dcfifo Megafunction

Table 3–8 shows the flag latencies in dual-clock FIFOs created using the dcfifo megafunction with different optimization settings and in different device families. The flag latencies for dual-clock FIFOs in Legacy mode and in Show-ahead mode are identical.

Table 3–8. Flag Latencies in the dcfifo Megafunction (Part 1 of 2)

Optimized for	Latency in Stratix, Stratix GX, Cyclone, APEX, and FLEX Devices	Latency in Stratix III, Stratix II, Stratix II GX, Cyclone III, and Cyclone II Devices
Speed	wrreq to wrempty: 0 wrclk cycles wrreq to rdempty: 1 wrclk cycle+next 6 rising rdclk edges wrreq to wrusedw []: 1 wrclk cycle wrreq to rdusedw []: 1 wrclk cycle + next 5 rising rdclk edges wrreq to wrfull: 0 wrclk cycles (FIFO is full between 2^N and 2^N-1 entries) wrreq to rdfull: varying latency due to conservative estimation of full FIFO rdreq to rdempty: 0 rdclk cycles rdreq to wrempty: next 6 rising wrclk edges rdreq to rdusedw []: 1 rdclk cycle rdreq to wrusedw []: next 5 rising wrclk edges rdreq to rdfull: 4 rdclk cycles rdreq to wrfull: varying latency due to conservative estimation of full FIFO	wrreq to wrempty: 0 wrclk cycles wrreq to rdempty: 1 wrclk cycle+next 3 rising rdclk edges wrreq to wrusedw []: 1 wrclk cycle wrreq to rdusedw []: 1 wrclk cycle + next 4 rising rdclk edges wrreq to wrfull: 0 wrclk cycles wrreq to rdfull: 1 wrclk cycle + next 3 rising rdclk edges rdreq to rdempty: 0 rdclk cycles rdreq to wrempty: next 3 rising wrclk edges rdreq to rdusedw []: 1 rdclk cycle rdreq to wrusedw []: next 4 rising wrclk edges rdreq to rdfull: 0 rdclk cycles rdreq to wrfull: next 3 rising wrclk edges

Table 3–8. Flag Latencies in the dcfifo Megafunction (Part 2 of 2)

Optimized for	Latency in Stratix, Stratix GX, Cyclone, APEX, and FLEX Devices	Latency in Stratix III, Stratix II, Stratix II GX, Cyclone III, and Cyclone II Devices
Area	Same as when optimized for speed	wrreq to wrempty: 0 wrclk cycles wrreq to rdempty: 1 wrclk cycle+next 2 rising rdclk edges wrreq to wrusedw []: 1 wrclk cycle wrreq to rdusedw []: 1 wrclk cycle + next 3 rising rdclk edges wrreq to wrfull: 0 wrclk cycles wrreq to rdfull: 1 wrclk cycle + next 2 rising rdclk edges rdreq to rdempty: 0 rdclk cycles rdreq to wrempty: next 2 rising wrclk edges rdreq to rdusedw []: 1 rdclk cycle rdreq to wrusedw []: next 3 rising wrclk edges rdreq to rdfull: 0 rdclk cycles rdreq to wrfull: next 2 rising wrclk edges