Single- and Dual-Clock FIFO Megafunction User Guide

© January 2009 UG-MFNALT_FIFO-5.0

Introduction

The Quartus® II software provides the FIFO MegaWizard plug-in through the MegaWizard™ Plug-In Manager. The MegaWizard Plug-In Manager helps you to efficiently configure and build FIFO megafunctions. You can find the FIFO MegaWizard plug-in under the **Memory Compiler** category.

For future references about the MegaWizard Plug-In Manager, refer to the *Megafunction Overview User Guide*. This user guide is only available after the Quartus II 9.0 software launch.

This user guide describes the different features of the FIFO megafunctions that you can customize through the FIFO MegaWizard plug-in such as FIFO modes, FIFO status flags, and metastability protection. This user guide also describes the timing constraints settings for Dual-Clock FIFO (DCFIFO) when you use the TimeQuest timing analyzer tool.

This user guide does not include the design example for FIFO megafunction. For an example of how to make use of the DCFIFO megafunction, refer to the *AN473: Using DCFIFO for Data Transfer between Asynchronous Clock Domains*.

FIFO Megafunction Features

The FIFO megafunction supports the following features:

- "FIFO Modes" on page 2
- "FIFO Status Flags" on page 6
- "Different Input and Output Width" on page 9
- "Latency and Related Options" on page 11
- "Additional Bits as MSB Extension of the usedw[] Ports" on page 12
- "Synchronous Clear and Asynchronous Clear" on page 13
- "Other Supported Features" on page 15. Other supported features include output modes, optimization options, circuitry protection for FIFO overflow and underflow, and simultaneous read and write operation

The following sections describe each supported features in the FIFO megafunction. Understanding the features helps you to build the FIFO with the functions you desire.

FIFO Modes

The FIFO megafunction supports two types of FIFO modes: single-clock FIFO (SCFIFO) and dual-clock FIFO (DCFIFO).

Figure 1 shows the input and output ports of the SCFIFO block.

Figure 1. SCFIFO Input and Output Ports

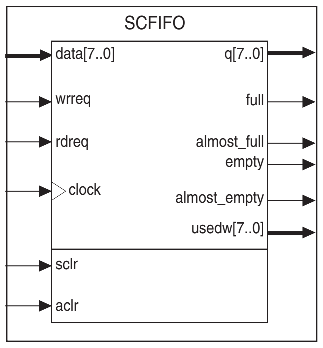
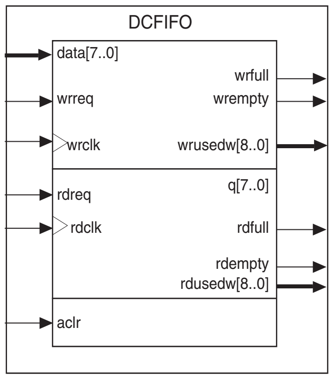


Figure 2 shows the input and output ports of the DCFIFO block.

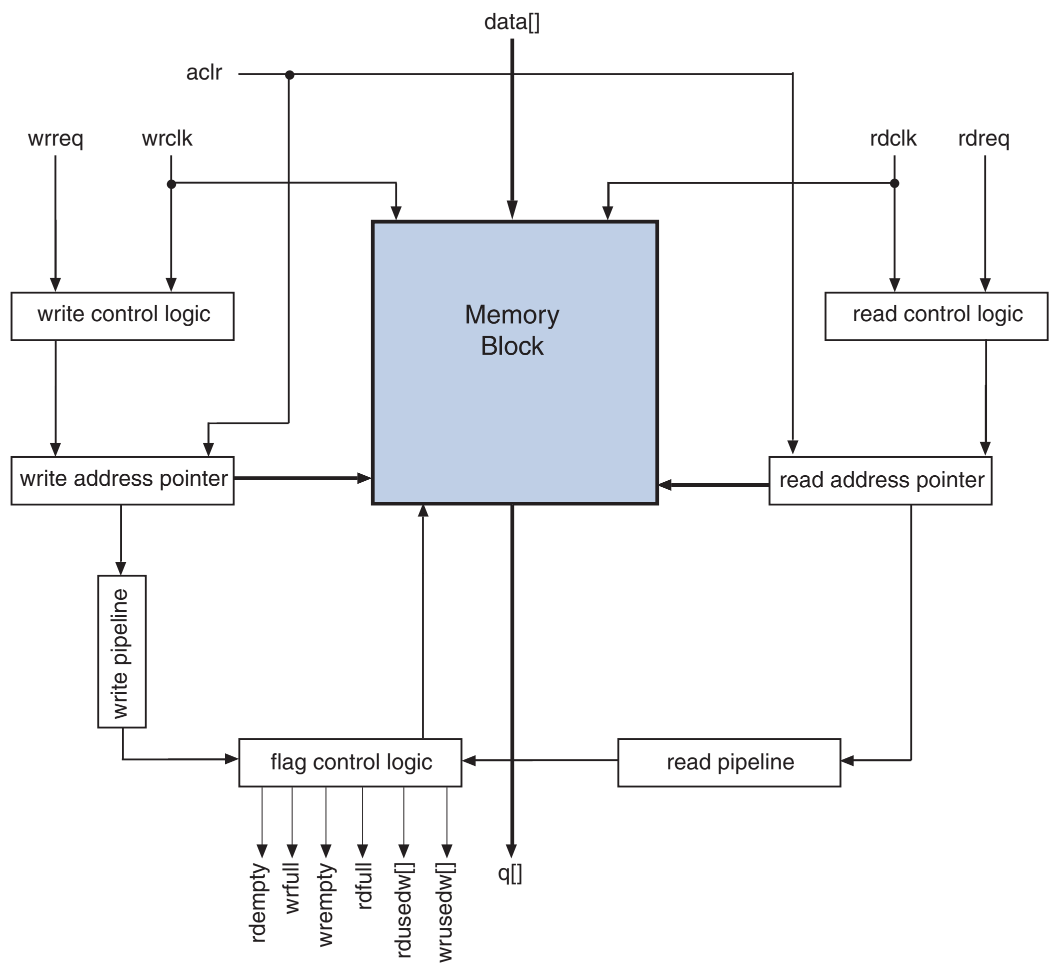
Figure 2. DCFIFO Input and Output Ports



In SCFIFO mode, the read and write signals are synchronized to the same clock. While in DCFIFO mode, the read and write signals are synchronized to `rdclk` and `wrclk`, respectively. For DCFIFO, the input signals are involved in clock domain crossing; therefore, flag generation is more complex in DCFIFO mode than in SCFIFO mode.

Figure 3 shows how the DCFIFO megafunction is implemented with an Altera® memory block. All control logic is implemented in logic elements (LEs).

Figure 3. DCFIFO Megafunction Block



`wrclk` synchronizes data write transactions to the memory block at the memory location indicated by the `Write Address Pointer`. `rdclk` synchronizes data read transactions from the memory block at the memory location indicated by the `Read Address Pointer`. The flag control logic generates 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 prevents the effects of metastability in the paths from the `Read Address Pointer` to the flag control logic. Similarly, synchronization in the write pipeline prevents the effects of this metastability in the paths from the `Write Address Pointer` to the flag control logic. The registers in the synchronization pipelines delay the effects of updates to the two pointers on the FIFO status flags; therefore, these synchronization pipelines cause the FIFO status signals to be delayed by multiple clock cycles.

Flag Latency in the DCFIFO Megafunction

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 pipeline latency.

The value on the `wrusedw[]` signal is calculated in the `wrclk` clock domain. The `Read Address Pointer` value passes from the `rdclk` clock domain into the flag control logic and is compared with `the Write Address Pointer` value in the `wrclk` 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 calculation. This encoding ensures only one bit changes from one code word to the next in sequence. 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.

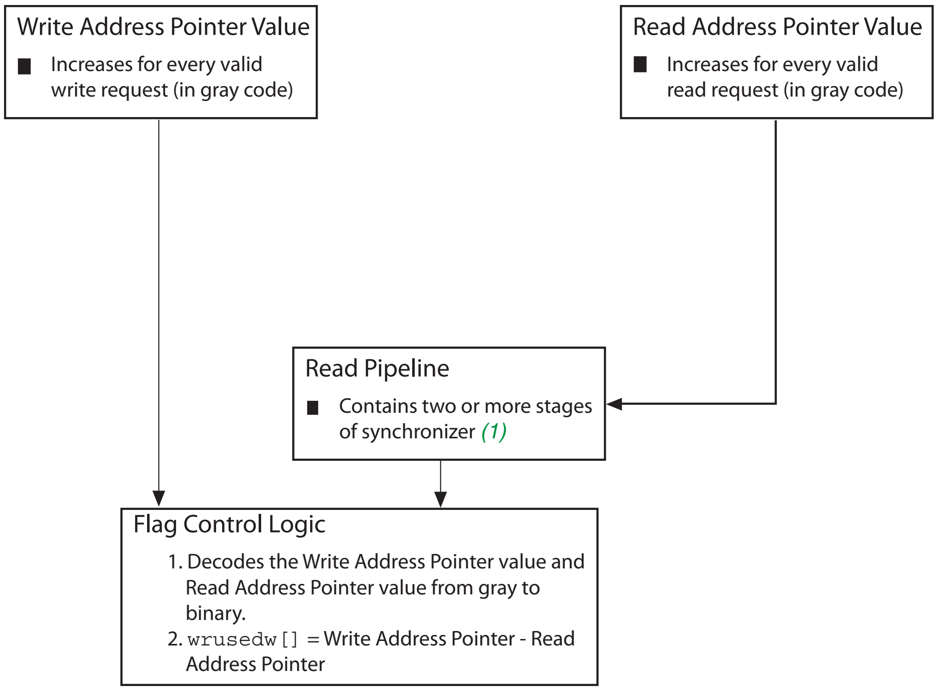
However, two issues remain:

- If multiple registers use the changing bit as input, some registers may capture a zero and others a one, leading to an inconsistency in 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 resolve these issues, the encoded `Read Address Pointer` value is synchronized two times or more with the `wrclk` clock. This synchronization occurs in the read pipeline. The flag control logic then convert 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.

Figure 4 shows the process of how `wrusedw[]` flag is calculated in `wrclk` clock domain.

Figure 4. Calculation for wrusedw[] Flag


Write Address Pointer Value
■ Increases for every valid write request (in gray code)
Read Address Pointer Value
■ Increases for every valid read request (in gray code)
Read Pipeline
■ Contains two or more stages of synchronizer (1)
Flag Control Logic
1. Decodes the Write Address Pointer value and Read Address Pointer value from gray to binary.
2. wrusedw[] = Write Address Pointer - Read Address Pointer

Note to Figure 4:

(1) You can set the number of synchronization stage from the FIFO MegaWizard plug-in. For more information about setting the number of synchronization stage, refer to the "Latency and Related Options" on page 11.

The encoding and synchronization lead to a delay of several clock cycles to update the `rdusedw[]` flags and `wrusedw[]` flags. Because of this latency, the `rdusedw[]` and `wrusedwp[]` 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 FIFO configuration settings. For more information about the flags delays, refer to the "FIFO Status Flags" on page 6.

FIFO Status Flags

The FIFO megafunction provides a list of flag signals to indicate its status of full, empty, and the number of words stored. The SCFIFO and DCFIFO megafunctions have different sets of status flags. SCFIFO supports additional flags to indicate when the SCFIFO is almost full or almost empty. For DCFIFO, both the read and write ports have their own set of flag signals synchronous under the read clock and write clock, respectively.

Table 1 shows the status flags for SCFIFO and DCFIFO megafunctions.

Table 1. SCFIFO and DCFIFO Megafunctions Status Flags

SCFIFO	**DCFIFO** *(1)*
`full`	`wrfull` *(5)*
`empty`	`wrempty`
`usedw[]` *(2)*	`wrusedw[]`
`almost_full` *(3)*	`rdfull`
`almost_empty` *(4)*	`rdempty`
	`rdusedw[]`

Notes to Table 1:

(1) The prefix `wr` and `rd` represent the signals in the write clock domain and read clock domain, respectively.

(2) The `usedw[]` represents the number of words in the FIFO. The value increases for every valid write operation and decreases for every valid read operation. You can use the MSB of the `usedw[]` to generate a half-full flag.

(3) The `almost_full` signal is optional and asserted when `usedw[]` is greater than or equal to the threshold value you set for the signal.

(4) The `almost_empty` signal is optional and asserted when `usedw[]` is less than the threshold value you set for the signal.

(5) If you select the **Add circuit to synchronize 'aclr' input with 'wrclk'** option from the FIFO MegaWizard plug-in, the `wrfull` signal does not de-assert during power up or when the DCFIFO is cleared. The `wrfull` signal asserts for two rising edges of the write clock after power up and for three rising edges of the write clock after the DCFIFO is cleared. Write operation is only valid at the next rising edge of the write clock after the `wrfull` signal is de-asserted.

An important concern in most FIFO design is the output latency of the status flags with respect to read and write operations. For SCFIFO, the output latency depends on the output modes and the optimization options applied.

For more information about output modes and optimization options, refer to "Other Supported Features" on page 15.

Table 2 shows the output latency with respect to the write signal (`wrreq`) and read signal (`rdreq`) for the SCFIFO megafunction for different output modes and optimization options.

Table 2. Output Latency of the Status Flags for SCFIFO

Output Mode	Optimization Option	Output Latency (in number of clock cycles) *(1)*
Normal *(2)*	Speed	`wrreq` / `rdreq` to `full`: 1
		`wrreq` to `empty`: 2
		`rdreq` to `empty`: 1
		`wrreq` / `rdreq` to `usedw[]`: 1
		`rdreq` to `q[]`: 1
	Area	`wrreq` / `rdreq` to `full`: 1
		`wrreq` / `rdreq` to `empty` : 1
		`wrreq` / `rdreq` to `usedw[]` : 1
		`rdreq` to `q[]`: 1
Show-ahead	Speed	`wrreq` / `rdreq` to `full`: 1
		`wrreq` to `empty`: 3
		`rdreq` to `empty`: 1
		`wrreq` / `rdreq` to `usedw[]`: 1
		`wrreq` to `q[]`: 3
		`rdreq` to `q[]`: 1 *(3)*
	Area	`wrreq` / `rdreq` to `full`: 1
		`wrreq` to `empty`: 2
		`rdreq` to `empty`: 1
		`wrreq` / `rdreq` to `usedw[]`: 1
		`wrreq` to `q[]`: 2
		`rdreq` to `q[]`: 1 *(3)*

Notes to Table 2:

(1) The information of the output latency is applicable for Stratix® and Cyclone® series of devices only. It may not be applicable for legacy devices such as APEX® and FLEX® series of devices.

(2) For Quartus II software before version 9.0, normal output mode is called legacy output mode.

(3) In show-ahead mode, assertion of `rdreq` is ignored before the first data is shown ahead at the output. Use the `empty` flag as an indicator of when a valid read can be performed.

Table 3 shows the output latency with respect to the write signal (`wrreq`) and read signal (`rdreq`) for DCFIFO megafunction.

Table 3. Output Latency of the Status Flag for DCFIFO *(Note 1)*

Output Latency (in number of clock cycles)
`wrreq` to `wrfull`: 1 `wrclk`
`wrreq` to `rdfull`: 2 `wrclk` cycles + following *n* `rdclk` *(2)*
`wrreq` to `wrempty`: 1 `wrclk`
`wrreq` to `rdempty`: 2 `wrclk` + following *n* `rdclk` *(2)*
`wrreq` to `wrusedw[]`: 2 `wrclk`
`wrreq` to `rdusedw[]`: 2 `wrclk` + following *n* + 1 `rdclk` *(2)*
`wrreq` to `q[]`: 1 `wrclk` + following 1 `rdclk` *(3)*
`rdreq` to `rdempty`: 1 `rdclk`
`rdreq` to `wrempty`: 1 `rdclk` + following *n* `wrclk` *(2)*
`rdreq` to `rfull`: 1 `rdclk`
`rdreq` to `wrfull`: 1 `rdclk` + following *n* `wrclk` *(2)*
`rdreq` to `rdusedw[]`: 2 `rdclk`
`rdreq` to `wrusedw[]`: 1 `rdclk` + following *n* + 1 `wrclk` *(2)*
`rdreq` to `q[]`: 1 `rdclk`

Notes to Table 3:

(1) The information of the output latency is only applicable for Arria® GX, Stratix, and Cyclone series of devices (except Stratix, Stratix GX, Hardcopy® Stratix, and Cyclone devices). It may not be applicable for legacy devices such as APEX and FLEX devices series of devices.

(2) The number of *n* cycles of `rdclk` and `wrclk` is equivalent to the number of the synchronization stages used. For more information about the setting of synchronization stage, refer to "Latency and Related Options" on page 11.

(3) This is only applied to show-ahead output modes. For more information about show-ahead output behavior, refer to "Other Supported Features" on page 15.

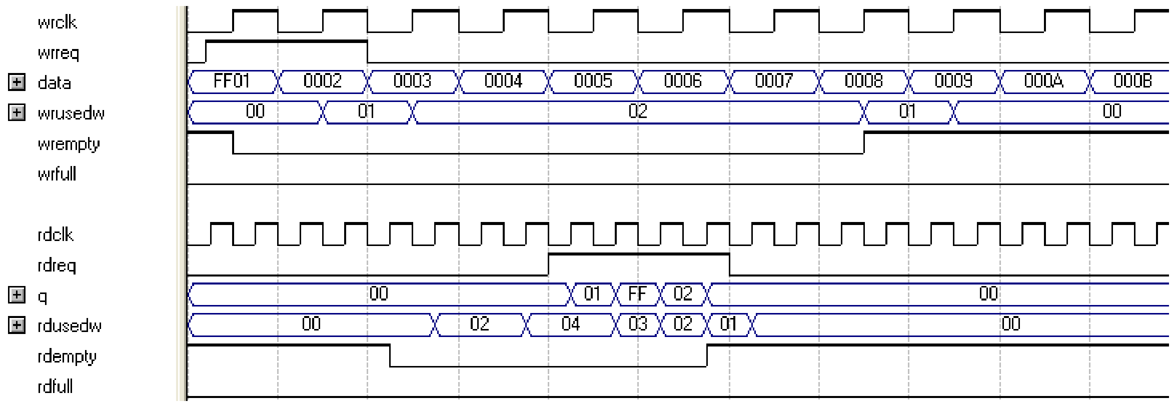
Different Input and Output Width

The DCFIFO megafunction supports different write input data width and read output data width if the width's ratio is valid. The FIFO MegaWizard plug-in prompts an error message if the combinations of the input data width and output data width produce an invalid ratio. The supported width ratio is restricted by the type of RAM block used and is generally in a power of 2.

The DCFIFO megafunction supports wide write port with narrow read port and narrow write port with wide read port.

Figure 5 shows an example of a wide write port (16-bit input) and a narrow read port (8-bit output).

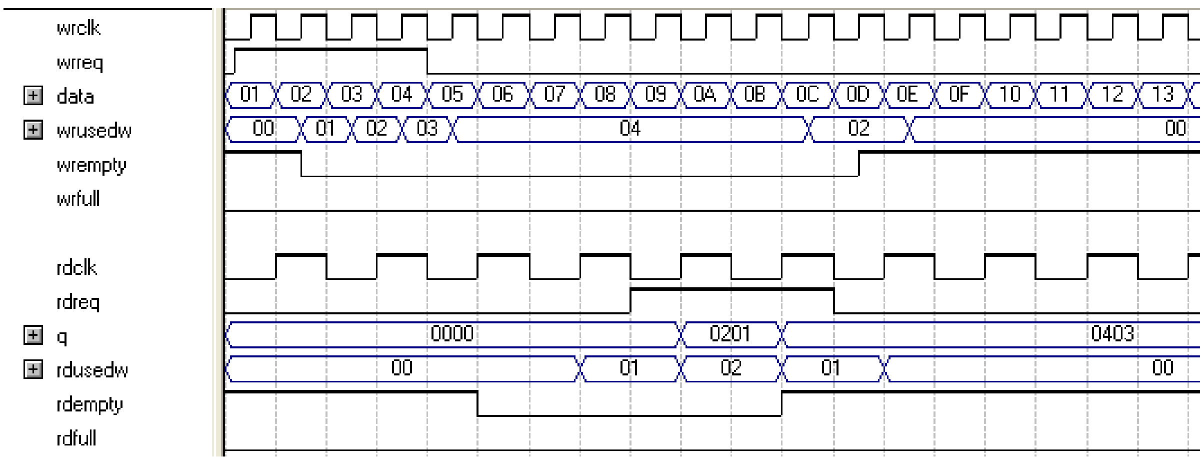
Figure 5. Writing 16-bit Words and Reading 8-bit Words



In this example, the read port is operating at twice the frequency of the write port. Two 16-bit words are written to the DCFIFO that result in the `wrusedw` flag increased to two and the `rdusedw` flag increased to four. Four 8-bit read operations empty the DCFIFO. The read begins with the least-significant 8 bits from the 16-bit word written followed by the most-significant 8 bits.

Figure 6 shows an example of a narrow write port (8-bit input) with a wide read port (16-bit output).

Figure 6. Writing 8-Bit Words and Reading 16-Bit Words



In this example, the read port is operating at half the frequency of the write port. Four 8-bit words are written to the DCFIFO that results in the `wrusedw` flag increased to four and the `rdusedw` flag increased to two. Two 16-bit read operations empty the DCFIFO. The first 8-bit and the second 8-bit written are equivalent to the LSB and the MSB of the 16-bit output words, respectively. The `rdempty` does not de-assert until enough words have been written on the narrow write port to fill an entire word on the wide read port.

Latency and Related Options

The FIFO MegaWizard plug-in provides the total latency, clock synchronization, metastability protection, area, and f_{max} options as a group setting for the DCFIFO megafunction.

Table 4 shows the available group setting.

Table 4. DCFIFO Megafunction Group Setting for Latency and Related Options *(Note 1)*

Group Setting	Comment
Lowest latency but requires synchronized clocks	This option uses one synchronization stage with no metastability protection. It utilizes the smallest size and provides good f_{max}. Select this option if the read clock and write clock are related clocks.
Minimal setting for unsynchronized clocks	This option uses two synchronization stages with good metastability protection. It utilizes the medium size and provides good f_{max}.
Best metastability protection, best f_{max} and unsynchronized clocks	This option allows you to use three or more synchronization stages with the best metastability protection. *(2)* It utilizes the largest size but gives the best f_{max}.

Note to Table 4:

(1) The latency and related options setting are only available for the DCFIFO megafunctions.

(2) You can set the number of sync stages to more than three from the FIFO MegaWizard plug-in.

Altera's TimeQuest timing analyzer includes the capability to estimate the robustness of asynchronous transfers in your design, and to generate a report that details the mean time between failures (MTBF) for all detected synchronization register chains.

For more information about enabling metastability analysis and reporting metastability in TimeQuest, refer to *Area and Timing Optimization* chapter of volume 2, and *Quartus II TimeQuest Timing Analyzer* chapter of volume 3 in the *Quartus II Handbook*

The **number of synchronization stage** set is related to the value of the WRSYNC_DELAYPIPE and RDSYNC_DELAYPIPE pipeline parameters. For some cases, these pipeline parameters are internally scaled down by two to reflect the actual synchronization stage.

Table 5 shows the cases for the relationship between the actual synchronization stage and the pipeline parameters.

Table 5. Relationship between Actual Synchronization Stage and Pipeline Parameters for Different Target Devices

Stratix II, Cyclone II, and onwards	Stratix and Cyclone Devices in Low-Latency Version *(1)*	Other Devices
Actual synchronization stage = value of pipeline parameter - 2		Actual synchronization stage = value of pipeline parameter

Note to Table 5:

(1) You can obtain the low-latency of DCFIFO when the clocks are set to not synchronized in show-ahead mode with unregistered output from the FIFO MegaWizard plug-in. The corresponding parameter settings for the low-latency version are ADD_RAM_OUTPUT_REGISTER=OFF, LPM_SHOWAHEAD=ON, and CLOCKS_ARE_SYNCHRONIZED=FALSE. These parameter settings are only applicable for Stratix and Cyclone devices.

Additional Bits as MSB Extension of the usedw[] Ports

In FIFO design, there is potential for confusion when dealing with the number of words stored in the FIFO, when they reach the maximum value but show zero. This is because the maximum value of an n-bit address is always 2^{n-1}, and Altera FIFO supports 2^{n} entries (most FIFO IP support only 2^{n-1} entries and does not need an extra MSB). For DCFIFO megafunction, you can avoid this scenario by using an additional bit as a MSB extension for each `rdusedw[]` and `wrusedw[]` ports.

☞ Altera strongly recommends using the MSB feature that can be set through the FIFO MegaWizard plug-in.

☞ This feature is not supported in SCFIFO mode for all devices. In DCFIFO mode, it is supported for Stratix and Cyclone series of devices (except Stratix, Stratix GX, Hardcopy Stratix, Cyclone, and legacy devices).

Synchronous Clear and Asynchronous Clear

The FIFO megafunction supports the synchronous clear (`sclr`) and asynchronous clear (`aclr`) signals, depending on the FIFO modes. The effects of these signals are varied for different FIFO configurations. SCFIFO supports both synchronous and asynchronous clear while DCFIFO supports asynchronous clear and asynchronous clear that synchronize with the write clock.

Table 6 shows the synchronous clear and asynchronous clear supported in SCFIFO.

Table 6. Synchronous Clear and Asynchronous Clear in SCFIFO

Mode	Synchronous Clear (sclr)	Asynchronous Clear (aclr)
Effects on status ports	De-asserts the `full` and `almost_full`.	
	Asserts the `empty` and `almost_empty`.	
	Resets the `usedw[]`.	
Commencement of effects upon assertion	At the rising edge of the clock.	Immediate (except for `q[]` output)
Effects on the `q[]` output for normal output modes	If the `q[]` output is not registered, the output shows the first data word of the SCFIFO; otherwise, it remains its previous value.	The `q[]` output remains at its previous value.
Effects on the `q[]` output for show-ahead output modes	If the `q[]` output is not registered, the output remains at its previous value for only one clock cycle and shows the first data word of the SCFIFO at the next rising clock edge. *(1)* Otherwise, the `q[]` output remains at its previous value.	If the `q[]` output is not registered, the output shows the first data word of the SCFIFO starting at the first rising clock edge. *(1)* Otherwise, the `q[]` output remains its previous value.

Note to Table 6:

(1) If you perform a write operation when the `empty` signal is asserted, the `q[]` output shows X (instead of showing first data word of the SCFIFO). For more information about the characteristic of show-ahead FIFO, refer to "Other Supported Features" on page 15.

Table 7 shows the asynchronous clear supported by DCFIFO.

Table 7. Asynchronous Clear in DCFIFO

Mode	Asynchronous Clear (aclr)	aclr (synchronize with write clock) *(1)*, *(2)*
Effects on status ports	De-asserts the `wrfull`.	Asserts the `wrfull` for three rising edges of write clock before de-asserting the signal.
	De-asserts the `rdfull`.	
	Asserts the `wrempty` and `rdempty`.	
	Reset the `wrusedw[]` and `rdusedw[]`.	
Commencement of effects upon assertion	Immediate.	
Effects on the `q[]` output for normal output modes	Asserting the `aclr` signal immediately clear the `q[]` output. *(3)*	
Effect on the `q[]` output for show-ahead output modes	Asserting the `aclr` signal immediately shows 'X' at the `q[]` output until the first rising edge of read clock, then it shows the first data word of DCFIFO. *(4)*	

Notes to Table 7:

(1) The `wrreq` signal must be low when the DCFIFO comes out of reset (the instant when the `aclr` signal is de-asserted) at the rising edge of the write clock to avoid a race condition between write and reset. If this condition cannot be guaranteed in your design, the `aclr` signal needs to be synchronized with the write clock. This can be done by setting the **Add circuit to synchronize 'aclr' input with 'wrclk'** option from the FIFO MegaWizard plug-in, or setting the parameter `WRITE_ACLR_SYNCH=ON`.

(2) Even though `aclr` is synchronized with the write clock, asserting the `aclr` signal still affects all the status flags asynchronously.

(3) The `q[]` output is registered in DCFIFO with normal output mode.

(4) The `q[]` output is not registered in DCFIFO with show-ahead output mode.

For correct timing analysis, Altera recommends enabling the **Removal and Recovery Analysis** option in the Classic timing analyzer tool when you use the `aclr` signal. The analysis is turned on by default in the TimeQuest timing analyzer tool.

Other Supported Features

The FIFO megafunction supports additional features such as output modes, optimization options, overflow and underflow circuitry protection, and simultaneous read and write.

Table 8 describes characteristics of these features.

Table 8. Feature Descriptions

Feature		Description
Output mode	Normal *(1)*	Treats the `rdreq` signal as a normal read-request that only performs read operations when the signal is asserted.
	Show-ahead *(2), (3)*	Treats the `rdreq` signal as a read-acknowledge that automatically outputs the first word of valid data in the FIFO without asserting the `rdreq` signal. Asserting the `rdreq` signal causes the FIFO to output the next data word, if available. This mode minimizes the read latency of the FIFO.
Optimization option *(4)*	Area	Results in fewer resources being used. The `q[]` output is not registered.
	Speed	Results in faster FIFO performance. The `q[]` output is registered.
Circuitry Protection *(5)*	Underflow	Disables read access while the FIFO is empty.
	Overflow	Disables write access while the FIFO is full.
Simultaneous read and write		Allow simultaneous read and write operation provided the FIFO is not empty or full.
Power-up behavior *(6)*		FIFO contents at power up are undefined because the memory cannot be initialized. If the `q` output is not registered, it shows 'X'; otherwise, the `q` output shows zero.
		After power up, if the first write operation is performed (when FIFO is empty), `q[]` output shows 'X' if the output is not registered in show-ahead mode.

Notes to Table 8:

(1) For DCFIFO in normal mode, the `q[]` output is registered by design.

(2) For DCFIFO in show-ahead mode, the `q[]` output is not registered by design.

(3) Asserting the `rdreq` signal before the first data show-ahead at `q[]` output port is invalid and is ignored by the FIFO megafunction in show-ahead mode. Always refer to the empty status to determine when a read operation can be performed. Assert `rdreq` only when empty is low.

(4) For Stratix and Cyclone series of devices (except Stratix, Stratix GX, Hardcopy Stratix, and Cyclone devices), this option is only available for SCFIFO. For DCFIFO, the `q[]` output is automatically registered in normal mode and unregistered in show-ahead mode.

(5) Underflow and overflow circuitry protections are enabled by default to prevent FIFO data corruption. Turning on the option from the FIFO MegaWizard plug-in disables the protection circuitry. When the protection circuitry is disabled, writing to a full FIFO can corrupt the FIFO data and reading from an empty FIFO can return unknown data and produce unpredictable future results; however, when the protection circuitry is disabled, the FIFO megafunction uses fewer device resources.

(6) RTL simulation and gate-level simulation have the same power-up behavior.

Functional Timing and Constraints Settings

The `wrreq` signal is ignored (when FIFO is full) if you enable the overflow protection circuitry from the FIFO MegaWizard plug-in, or set the parameter `OVERFLOW_CHECKING` = `ON`. The `rdreq` signal is ignored (when FIFO is empty) if you enable the underflow protection circuitry from the FIFO MegaWizard plug-in, or set the parameter `UNDERFLOW_CHECKING` = `ON`.

For DCFIFO mode, you must ensure the following functional timing requirement if you do not enable the protection circuitry.

- The `wrreq` signal must be de-asserted at the same clock cycle when the `wrfull` signal is asserted.
- The `rdreq` signal must be de-asserted at the same clock cycle when the `rdempty` signal is asserted.

Figure 7 shows the required functional timing for the `wrreq` signal with respect to the `wrfull` signal.

Figure 7. Functional Timing for `wrreq` the Signal with the `wrfull` Signal

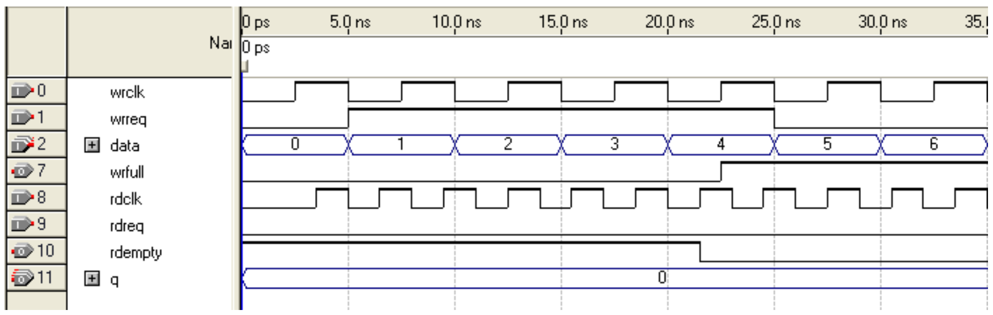
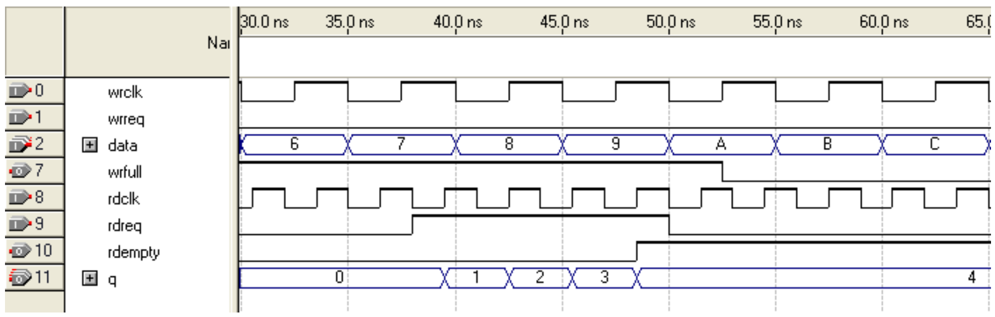


Figure 8 shows the required functional timing for the `rdreq` signal with respect to the `rdempty` signal.

Figure 8. Functional Timing for `rdreq` the Signal with `rdempty` the Signal



☞ The required functional timing for DCFIFO as described previously is also applied to SCFIFO. The difference between the two modes is that for SCFIFO, the `wrreq` signal is with respect to `full` signal and the `rdreq` signal is with respect to the `empty` signal.

When using the Quartus II TimeQuest timing analyzer and your design contains a DCFIFO megafunction, the following false paths are required to avoid timing failures in the synchronization registers:

1. For paths crossing from the write into the read domain, apply a false path assignment between the `delayed_wrptr_g` and `rs_dgwp` registers:

   ```
   set_false_path -from [get_registers
   {*dcfifo*delayed_wrptr_g[*]}] -to [get_registers
   {*dcfifo*rs_dgwp*}]
   ```

2. For paths crossing from the read into the write domain, apply a false path assignment between the `rdptr_g` and `ws_dgrp` registers:

   ```
   set_false_path -from [get_registers {*dcfifo*rdptr_g[*]}]
   -to [get_registers {*dcfifo*ws_dgrp*}]
   ```

For Quartus II software version 8.1 and onwards, you do not have to add the false path assignments because they are automatically applied through the HDL embedded SDC commands when you compile your design. Note that the constraints are internally applied but not written to the **.sdc** file.

If you use the Quartus II Classic timing analyzer, the false paths are applied automatically for the DCFIFO megafunction.

For more information about setting the timing constraint, refer to the *Quartus II TimeQuest Timing Analyzer* chapter in volume 3 of the *Quartus II Handbook*.

Ports and Parameters

If you select **FIFO** under **Memory Compiler** from the MegaWizard Plug-In Manager, it instantiates the single clock FIFO megafunction (SCFIFO) or dual-clock FIFO (DCFIFO) megafunction, depending on the FIFO mode selected. The SCFIFO megafunction is called SCFIFO. There are two types of dual-clock megafunctions, DCFIFO_MIXED_WIDTH (for mixed-width DCFIFO functions) and DCFIFO (for same-width DCFIFO functions). The details of the ports and parameters of the megafunctions are hidden from the MegaWizard Plug-In Manager interface. These information are only relevant if you bypass the MegaWizard Plug-In Manager interface and use the megafunction as a directly parameterized instantiation in your design.

☞ Altera recommends using the MegaWizard Plug-In Manager to configure and build your FIFO functions. This ensures the combination of options you set for the FIFO megafunction are valid.

☞ Dual-clock FIFO megafunctions is referring to both DCFIFO megafunction and DCFIFO_MIXED_WIDTHS megafunction, unless specified otherwise.

Table 9 shows the input and output ports for DCFIFO megafunctions.

Table 9. DCFIFO Megafunctions Input and Output Ports (Part 1 of 2)

Port	Type	Required	Description
`aclr`	Input	No	Asynchronously reset the Write and Read Address Pointer, and the FIFO is considered empty. Asserting the `aclr` does not clear the content of the FIFO. For more information about the effects of the `aclr` to the status flags and `q` output in different FIFO configurations, refer to Table 6 on page 13 and Table 7 on page 14 .
`sclr`	Input	No	Synchronously reset the Write and Read Address Pointer and the FIFO is considered empty. Only supported by SCFIFO megafunction. Asserting the `sclr` does not clear the content of the SCFIFO. For more information about the effects of the `sclr` to the status flags and `q` output in different SCFIFO configurations, refer to Table 6 on page 13 .
`data`	Input	Yes	Data input to the FIFO. The size of the port is equal to the value of the `LPM_WIDTH` parameter.
`rdclk`	Input	Yes	Positive-edge-triggered clock to read data from the FIFO. Only applicable for dual-clock FIFO megafunctions.
`rdreq`	Input	Yes	Read request port. The `rdreq` must be de-asserted at the same clock cycle when `rdempty` or `empty` is asserted for dual-clock FIFO megafunctions or SCFIFO megafunction, respectively. For more information about the effect of the `rdreq` to the output in different FIFO output modes, refer to Table 8 on page 15 .
`wrclk`	Input	Yes	Positive-edge-triggered clock to write data to the FIFO. Only applicable for dual-clock FIFO megafunctions.

Table 9. DCFIFO Megafunctions Input and Output Ports (Part 2 of 2)

Port	Type	Required	Description
`wrreq`	Input	Yes	Write request port. The `wrreq` must be de-asserted at the same clock cycle when `wrfull` or `full` is asserted for dual-clock FIFO megafunctions or SCFIFO megafunction, respectively. Do not assert `wrreq` when the device enters user mode or during an asynchronous clear cycle. Specifically, `wrreq` 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 the `aclr` and the rising edge of the write clock if `wrreq` is high. If this occurs, it is possible for some of the write counter bits to transition while others do not, potentially resulting in an inconsistent write counter state. If this condition is not guaranteed in your design, the `aclr` should be synchronized with the write clock. For dual-clock FIFO megafunctions, this can be done by setting **Add circuit to synchronize 'aclr' input with 'wrclk'** from the FIFO MegaWizard Plug-In Manager, or set the parameter, `WRITE_ACLR_SYNCH=ON`
`clock`	input	Yes	Positive-edge-triggered clock. Only applicable for SCFIFO megafunction.
`q`	Output	Yes	Data output from the FIFO. In normal output mode, `q` output is valid the following clock after `rdreq` is asserted. The size of the port is equal to the value of the `LPM_WIDTH` parameter (for SCFIFO and DCFIFO megafunctions), or `LPM_WIDTH_R` parameter (for DCFIFO_MIXED_WIDTHS megafunction).
`rdempty` *(1)*	Output	No	Read empty flag. When asserted, the FIFO is considered empty, and the `rdreq` is disabled if `UNDERFLOW_CHECKING=ON`. The `rdempty` port is synchronized with `rdclk` and is more accurate than `wrempty`.
`rdfull` *(1)*	Output	No	Read full flag. Indicates that the FIFO is full when asserted. The `rdfull` port is a delayed version of `wrfull` that is synchronized with `rdclk`.
`rdusedw` *(1)*	Output	No	Read used flag. Indicates the number of words in the FIFO, and is synchronized with `rdclk`. The width of the `rdusedw` port is equal to the `LPM_WIDTHU` parameter (for DCFIFO megafunction), or `LPM_WIDTHU_R` parameter (for DCFIFO_MIXED_WIDTHS megafunction).
`wrempty` *(1)*	Output	No	Write empty flag. The `wrempty` port is a delayed version of `rdempty` that is synchronized with `wrclk`.
`wrfull` *(1)*	Output	No	Write full flag. When asserted, the FIFO is considered `full`, and the `wrreq` is disabled if `OVERFLOW_CHECKING=ON`. The `wrfull` port is synchronized with `wrclk` and is more accurate than `rdfull`.
`wrusedw` *(1)*	Output	No	Write used flag. Indicates the number of words in the FIFO, and is synchronized with `wrclk`. The width of the `wrusedw` port is equal to the `LPM_WIDTHU` parameter.
`almost_full`	Output	No	Asserted when `usedw` is greater than or equal to `ALMOST_FULL_VALUE` parameter. Only applicable for SCFIFO megafunction.
`almost_empty`	Output	No	Asserted when `usedw` is less than `ALMOST_EMPTY_VALUE` parameter. Only applicable for SCFIFO megafunction.

Note to Table 9:

(1) These ports are only available in dual-clock FIFO megafunctions. For SCFIFO megafunction, only `empty`, `full`, and `usedw` ports are available.

Table 10 shows the parameters for the FIFO megafunctions.

Table 10. FIFO Megafunctions Parameters (Part 1 of 3)

Parameter	Type	Required	Description
LPM_WIDTH	Integer	Yes	Width of the data[] and q[] ports for SCFIFO megafunction and DCFIFO megafunction. For DCFIFO_MIXED_WIDTHS megafunction, this parameter represents the width of the data[] port.
LPM_WIDTH_R	Integer	Yes	Width of the q[] port for DCFIFO_MIXED_WIDTHS megafunction.
LPM_WIDTHU	Integer	Yes	Width of the usedw[] port for SCFIFO megafunction, or width of the rdusedw[] and wrusedw[] ports for DCFIFO megafunctions. For DCFIFO_MIXED_WIDTHS megafunction, this parameter represents the width of the wrusedw[] port. The recommended value for this parameter is ceil, (LOG_2(LPM_NUMWORDS))
LPM_WIDTHU_R	Integer	Yes	Width of the rdusedw[] port for DCFIFO_MIXED_WIDTHS megafunction.
LPM_NUMWORDS	Integer	Yes	Number of words stored in FIFO, which is a power of 2 and minimum value is 4. For all the FIFO megafunctions, this parameter always corresponds to the write side of the FIFO, that is, LPM_WIDTHU= LOG_2(LPM_NUMWORDS)).
LPM_SHOWAHEAD	String	No	Specifies whether the FIFO is in normal mode (for example, when OFF) or show-ahead mode (for example, when ON). In Show-ahead mode, data immediately appears on q[] port without explicitly asserting the rdreq signal. Specifying ON for LPM_SHOWAHEAD parameter might reduce performance.
LPM_TYPE	String	No	Identifies the library of parameterized modules (LPM) entity name in VHDL Design Files (**.vhd**). Values are SCFIFO and DCFIFO.
MAXIMIZE_SPEED	Integer	—	Altera-specific parameter. Specifies whether to optimize for area or speed. Values are 0 through 10. Value 0,1,2,3,4, and 5 result in area optimization, while value 6,7,8,9, and 10 result in performance optimization. This parameter is applicable for dual-clock FIFO megafunctions and is available for Cyclone II and Stratix II devices only. For Cyclone III and Stratix III devices and onwards, this parameter is ignored because they use the optimized implementation.
OVERFLOW_CHECKING	String	No	Specifies the overflow checking to enable the protection circuitry that ignores wrreq when FIFO is full. Values are ON or OFF. If omitted, the default value is ON. Writing to a full FIFO yields unpredictable results.
UNDERFLOW_CHECKING	String	No	Specifies the underflow checking to enable the protection circuitry that ignores rdreq when FIFO is empty. Values are ON or OFF. If omitted, the default value is ON. Reading an empty FIFO yields unpredictable results.

Table 10. FIFO Megafunctions Parameters (Part 2 of 3)

Parameter	Type	Required	Description
ADD_USEDW_MSB_BIT	String	No	Increase the width of the rdusedw and wrusedw ports by one bit to prevent rollover to zero on a full FIFO. Values are ON and OFF. If omitted, the default value is OFF. This parameter applicable for dual-clock FIFO megafunctions only.
DELAY_RDUSEDW	Integer	No	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. Increasing the value of the DELAY_RDUSEDW parameter does not increase the maximum system speed; it only adds additional latency to rdusedw. This parameter applicable for dual-clock FIFO megafunctions only.
DELAY_WRUSEDW	Integer	No	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. Increasing the value of the DELAY_WRUSEDW parameter does not increase the maximum system speed; it only adds additional latency to wrusedw. This parameter applicable for dual-clock FIFO megafunctions only.
RDSYNC_DELAYPIPE WRSYNC_DELAYPIPE	Integer	No	Specifies the number of synchronization stages. Value of RDSYNC_DELAYPIPE represents the synchronization stages from the write control logic to the read control logic, while WRSYNC_DELAYPIPE represents the synchronization stages from the read control logic to the write control logic. The default value of 3 provides good insurance against the possibility of internal metastability when rdclock and wrclock are unrelated. For Cyclone II and Stratix II devices and onwards, the values of these parameters are internally reduced by 2. Thus, the default values of 3 for these parameters corresponds to a single synchronization pipe stage, a value of 4 results in 2 synchronization pipe stages, and so on. For more information, refer to “Latency and Related Options” on page 11.
USE_EAB	String	No	Altera-specific parameter. Specifies whether the FIFO is constructed using RAM blocks. Values are ON and OFF. Setting this parameter value to OFF yields the FIFO implemented in logic elements.
WRITE_ACLR_SYNCH	String	No	Specifies whether the dual-clock FIFO megafunctions add a circuit to synchronize the aclr with the wrclk. Values are ON and OFF. Set the value ON to avoid race condition aclr and wrreq. For more information, refer to the description of the wrreq input port, or Table 7 on page 14.

Table 10. FIFO Megafunctions Parameters (Part 3 of 3)

Parameter	Type	Required	Description
CLOCKS_ARE_SYNCHRONIZED *(1)*	String	No	Specifies whether the write and read clocks are synchronized. Values are TRUE and FALSE. If omitted, the default value is FALSE. When the ADD_RAM_OUTPUT_REGISTERED parameter is set to ON, the CLOCKS_ARE_SYNCHRONIZED parameter must be set to TRUE. This setting instruct the DCFIFO megafunction to simplify the internal logic. However, you must ensure the phase relationship of the read and write clocks guarantees that synchronization registers are not required.
RAM_BLOCK_TYPE	String	No	Specifies the target device's Trimatrix Memory Block to be used. This parameter is passed to the megafunction as LPM_HINT. For example, *<instant's name>*.**lpm_hint**= "RAM_BLOCK_TYPE=M9K"
LPM_HINT	String	No	Allows you to assign Altera-specific parameters. If omitted, the default is UNUSED. When you instantiate a LPM function in a **.vhd** file, you must use the LPM_HINT parameter to specify an Altera-specific parameter. For example: *<instant's name>*.**lpm_hint**= "MAXIMIZE_SPEED=7".
ADD_RAM_OUTPUT_REGISTER	String	No	Specifies whether to register the RAM output. Values are ON and OFF. If omitted, the default value is OFF.
ALMOST_FULL_VALUE *(2)*	Integer	No	The threshold value for the almost_full port.
ALMOST_EMPTY_VALUE *(2)*	Integer	No	The threshold value for the almost_empty port.
ALLOW_RWCYCLE_WHEN_FULL *(2)*	String	No	Allows combining read and write cycles to an already full SCFIFO, so that it remains full. Values are ON and OFF. If omitted, the default is OFF. This parameter is used only when the OVERFLOW_CHECKING parameter is set to ON.

Notes to Table 10:

(1) Only applicable for Stratix and Cyclone devices in dual-clock FIFO megafunctions. For other Stratix and Cyclone series of devices, use RDSYNC_DELAYPIPE and WRSYNC_DEPAYPIPE parameters.

(2) Only applicable for SCFIFO megafunction.

Revision History

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

Date	Document Version	Changes Made
January 2009	5.0	Complete re-write of the user guide.
May 2007	4.0	Updates for Quartus II v7.1: ■ Added support for Arria GX devices. ■ Updated for new GUI. ■ Added six design examples in place of functional ■ description. ■ Re-organized and updated Chapter 3 to have separate tables for the scfifo and dcfifo megafunctions. ■ Added Referenced Documents section.
March 2007	3.3	Updates for Quartus II v7.0: ■ Minor content changes, including adding Stratix III and Cyclone III information ■ Re-took screenshots for software version 7.0
September 2005	3.2	Minor content changes.

101 Innovation Drive
San Jose, CA 95134
www.altera.com
Technical Support
www.altera.com/support

I.S. EN ISO 9001