SCFIFO and DCFIFO Megafunctions

UG-MFNALT_FIFO-7.0 User Guide

Altera provides FIFO functions through the parameterizable single-clock FIFO (SCFIFO) and dual-clock FIFO (DCFIFO) megafunctions. The FIFO functions are mostly applied in data buffering applications that comply with the first-in-first-out data flow in synchronous or asynchronous clock domains. The specific names of the megafunctions are as follows:

- SCFIFO: single-clock FIFO
- DCFIFO: dual-clock FIFO (supports same port widths for input and output data)
- DCFIFO_MIXED_WIDTHS: dual-clock FIFO (supports different port widths for input and output data)

In this user guide, the term "DCFIFO" refers to both the DCFIFO and DCFIFO_MIXED_WIDTHS megafunctions, unless specified.

This user guide contains the following sections:

- "Configuration Methods" on page 2
- "Specifications" on page 2
- "Parameter Specifications" on page 10
- "Functional Timing Requirements" on page 13
- "Output Status Flags and Latency" on page 14
- "Metastability Protection and Related Options" on page 16
- "Synchronous Clear and Asynchronous Clear Effect" on page 18
- "Different Input and Output Width" on page 19
- "Constraint Settings" on page 21
- "Coding Example for Manual Instantiation" on page 22
- "Design Example" on page 23

Before you configure and build the FIFO megafunction, refer to "Specifications" on page 2 and "Parameter Specifications" on page 10. The description about input ports, output ports, and parameters is important especially if you decide to manually instantiate the megafunctions.

101 Innovation Drive
San Jose, CA 95134
www.altera.com

© 2012 Altera Corporation. All rights reserved. ALTERA, ARRIA, CYCLONE, HARDCOPY, MAX, MEGACORE, NIOS, QUARTUS and STRATIX words and logos are trademarks of Altera Corporation and registered in the U.S. Patent and Trademark Office and in other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders as described at www.altera.com/common/legal.html. 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. 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.

ISO 9001:2008 Registered

Configuration Methods

There are two methods to configure and build the FIFO megafunctions:

- Using the FIFO parameter editor launched from the MegaWizard™ Plug-In Manager in the Quartus® II software.

  Altera recommends using this method to build your FIFO megafunctions. It is an efficient way to configure and build the FIFO megafunctions. The FIFO parameter editor provides options that you can easily use to configure the FIFO megafunctions.

  For general information about the Quartus II MegaWizard Plug-In Manager, refer to the *Megafunction Overview User Guide*.

- Manually instantiating the FIFO megafunctions.

  Use this method only if you are an expert user. This method requires that you know the detailed specifications of the megafunctions. You must ensure that the input and output ports used, and the parameter values assigned are valid for the FIFO megafunction you instantiate for your target device.

  For coding examples about how to manually instantiate the FIFO megafunctions, you can refer to "Coding Example for Manual Instantiation" on page 22.

Specifications

This section describes the prototypes, component declarations, ports, and parameters of the SCFIFO and DCFIFO megafinctions. These ports and parameters are available to customize the SCFIFO and DCFIFO megafunctions according to your application.

Verilog HDL Prototype

You can locate the following Verilog HDL prototypes in the Verilog Design File (**.v**) **altera_mf.v** in the <*Quartus II installation directory*>**\eda\synthesis** directory.

SCFIFO

```
module  scfifo
#(  parameter   add_ram_output_register = "OFF",
    parameter   allow_rwcycle_when_full = "OFF",
    parameter   almost_empty_value = 0,
    parameter   almost_full_value = 0,
    parameter   intended_device_family = "unused",
    parameter   lpm_numwords = 1,
    parameter   lpm_showahead = "OFF",
    parameter   lpm_width = 1,
    parameter   lpm_widthu = 1,
    parameter   overflow_checking = "ON",
    parameter   underflow_checking = "ON",
    parameter   use_eab = "ON",
    parameter   lpm_type = "scfifo",
    parameter   lpm_hint = "unused")
(   input   wire    aclr,
    output  wire    almost_empty,
    output  wire    almost_full,
```

```
    input   wire    clock,
    input   wire    [lpm_width-1:0] data,
    output  wire    empty,
    output  wire    full,
    output  wire    [lpm_width-1:0] q,
    input   wire    rdreq,
    input   wire    sclr,
    output  wire    [lpm_widthu-1:0]    usedw,
    input   wire    wrreq)/* synthesis syn_black_box=1 */;
endmodule
```

DCFIFO

```
module  dcfifo_mixed_widths
#(  parameter   add_ram_output_register = "OFF",
    parameter   add_usedw_msb_bit = "OFF",
    parameter   clocks_are_synchronized = "FALSE",
    parameter   delay_rdusedw = 1,
    parameter   delay_wrusedw = 1,
    parameter   intended_device_family = "unused",
    parameter   lpm_numwords = 1,
    parameter   lpm_showahead = "OFF",
    parameter   lpm_width = 1,
    parameter   lpm_width_r = 0,
    parameter   lpm_widthu = 1,
    parameter   lpm_widthu_r = 1,
    parameter   overflow_checking = "ON",
    parameter   rdsync_delaypipe = 0,
    parameter   underflow_checking = "ON",
    parameter   use_eab = "ON",
    parameter   write_aclr_synch = "OFF",
    parameter   wrsync_delaypipe = 0,
    parameter   lpm_type = "dcfifo_mixed_widths",
    parameter   lpm_hint = "unused")
(   input   wire    aclr,
    input   wire    [lpm_width-1:0] data,
    output  wire    [lpm_width_r-1:0]   q,
    input   wire    rdclk,
    output  wire    rdempty,
    output  wire    rdfull,
    input   wire    rdreq,
    output  wire    [lpm_widthu_r-1:0]  rdusedw,
    input   wire    wrclk,
    output  wire    wrempty,
    output  wire    wrfull,
    input   wire    wrreq,
    output  wire    [lpm_widthu-1:0]    wrusedw)/* synthesis syn_black_box=1 */;

endmodule //dcfifo_mixed_widths
```

VHDL Component Declaration

You can locate the following VHDL component declarations in the VHDL Design File (**.vhd**) **altera_mf.vhd** in the <*Quartus II installation directory*>**\libraries\vhdl\altera_mf** directory.

SCFIFO

```
component scfifo
    generic (
        add_ram_output_register:   string := "OFF";
        allow_rwcycle_when_full:   string := "OFF";
        almost_empty_value:    natural := 0;
        almost_full_value:     natural := 0;
        intended_device_family: string := "unused";
        lpm_numwords:   natural;
        lpm_showahead:  string := "OFF";
        lpm_width:      natural;
        lpm_widthu:     natural := 1;
        overflow_checking:  string := "ON";
        underflow_checking: string := "ON";
        use_eab:    string := "ON";
        lpm_hint:   string := "UNUSED";
        lpm_type:   string := "scfifo"
    );
    port(
        aclr:   in std_logic := '0';
        almost_empty:   out std_logic;
        almost_full:    out std_logic;
        clock:  in std_logic;
        data:   in std_logic_vector(lpm_width-1 downto 0);
        empty:  out std_logic;
        full:   out std_logic;
        q   :   out std_logic_vector(lpm_width-1 downto 0);
        rdreq:  in std_logic;
        sclr:   in std_logic := '0';
        usedw:  out std_logic_vector(lpm_widthu-1 downto 0);
        wrreq:  in std_logic
    );
end component;
```

DCFIFO

```
component dcfifo_mixed_widths
    generic (
        add_ram_output_register:   string := "OFF";
        add_usedw_msb_bit:  string := "OFF";
        clocks_are_synchronized:   string := "FALSE";
        delay_rdusedw:  natural := 1;
        delay_wrusedw:  natural := 1;
        intended_device_family: string := "unused";
        lpm_numwords:   natural;
        lpm_showahead:  string := "OFF";
        lpm_width:      natural;
        lpm_width_r:    natural := 0;
        lpm_widthu:     natural := 1;
        lpm_widthu_r:   natural := 1;
```

```
        overflow_checking:  string := "ON";
        rdsync_delaypipe:   natural := 0;
        underflow_checking:string := "ON";
        use_eab:    string := "ON";
        write_aclr_synch:   string := "OFF";
        wrsync_delaypipe:   natural := 0;
        lpm_hint:   string := "UNUSED";
        lpm_type:   string := "dcfifo_mixed_widths"
    );
    port(
        aclr:   in std_logic := '0';
        data:   in std_logic_vector(lpm_width-1 downto 0);
        q   :   out std_logic_vector(lpm_width_r-1 downto 0);
        rdclk:  in std_logic;
        rdempty:    out std_logic;
        rdfull:     out std_logic;
        rdreq:      in std_logic;
        rdusedw:    out std_logic_vector(lpm_widthu_r-1 downto 0);
        wrclk:      in std_logic;
        wrempty:    out std_logic;
        wrfull:     out std_logic;
        wrreq:      in std_logic;
        wrusedw:    out std_logic_vector(lpm_widthu-1 downto 0)
    );
end component;
```

VHDL LIBRARY-USE Declaration

The VHDL LIBRARY-USE declaration is not required if you use the VHDL component declaration.

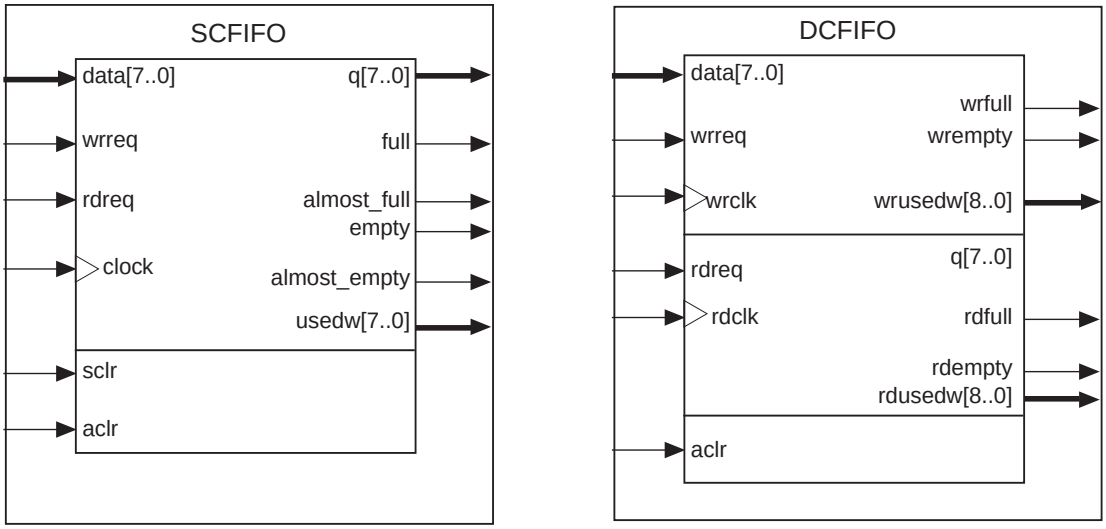
```
LIBRARY altera_mf;
USE altera_mf.altera_mf_components.all;
```

Ports Specifications

This section provides diagrams of the SCFIFO and DCFIFO blocks to help in visualizing their input and output ports. This section also describes each port in detail to help in understanding their usages, functionality, or any restrictions. For better illustrations, some descriptions might refer you to a specific section in this user guide.

Figure 1 shows the input and output ports of the SCFIFO and DCFIFO megafunctions.

Figure 1. Input and Output Ports



For the SCFIFO block, the read and write signals are synchronized to the same clock; for the DCFIFO block, the read and write signals are synchronized to the `rdclk` and `wrclk` clocks respectively. The prefixes `wr` and `rd` represent the signals that are synchronized by the `wrclk` and `rdclk` clocks respectively.

Table 1 describes the ports of the megafunctions.

The term "series" refers to all the device families of a particular device. For example, "Stratix series" refers to the Stratix®, Stratix GX, Stratix II, Stratix II GX, Stratix III, and new devices, unless specified otherwise.

Table 1. Input and Output Ports Description (Part 1 of 3)

Port	Type	Required	Description
`clock` *(1)*	Input	Yes	Positive-edge-triggered clock.
`wrclk` *(2)*	Input	Yes	Positive-edge-triggered clock. Use to synchronize the following ports: ■ `data` ■ `wrreq` ■ `wrfull` ■ `wrempty` ■ `wrusedw`
`rdclk` *(2)*	Input	Yes	Positive-edge-triggered clock. Use to synchronize the following ports: ■ `q` ■ `rdreq` ■ `rdfull` ■ `rdempty` ■ `rdusedw`
`data` *(3)*	Input	Yes	Holds the data to be written in the FIFO megafunction when the `wrreq` signal is asserted. If you manually instantiate the FIFO megafunction, ensure the port width is equal to the `lpm_width` parameter.

Table 1. Input and Output Ports Description (Part 2 of 3)

Port	Type	Required	Description
wrreq (3)	Input	Yes	Assert this signal to request for a write operation. Ensure that the following conditions are met: ■ Do not assert the wrreq signal when the full (for SCFIFO) or wrfull (for DCFIFO) port is high. Enable the overflow protection circuitry or set the overflow_checking parameter to **ON** so that the FIFO megafunction can automatically disable the wrreq signal when it is full. ■ The wrreq signal must meet the functional timing requirement based on the full or wrfull signal. Refer to "Functional Timing Requirements" on page 13. ■ Do not assert the wrreq signal during the deassertion of the aclr signal. Violating this requirement creates a race condition between the falling edge of the aclr signal and the rising edge of the write clock if the wrreq port is set to high. For both the DCFIFO megafunctions that target Stratix and Cyclone series (except Stratix, Stratix GX, and Cyclone devices), you have the option to automatically add a circuit to synchronize the aclr signal with the wrclk clock, or set the write_aclr_synch parameter to **ON**. Use this option to ensure that the restriction is obeyed.
rdreq (3)	Input	Yes	Assert this signal to request for a read operation. The rdreq signal acts differently in normal mode and show-ahead mode. For more information about the two different FIFO modes, refer to the description of the lpm_showahead parameter in Table 2 on page 10. Ensure that the following conditions are met: ■ Do not assert the rdreq signal when the empty (for SCFIFO) or rdempty (for DCFIFO) port is high. Enable the underflow protection circuitry or set the underflow_checking parameter to **ON** so that the FIFO megafunction can automatically disable the rdreq signal when it is empty. ■ The rdreq signal must meet the functional timing requirement based on the empty or rdempty signal. Refer to "Functional Timing Requirements" on page 13.
sclr (1) aclr (3)	Input	No	Assert this signal to clear all the output status ports, but the effect on the q output may vary for different FIFO configurations. For more information about the effects on asserting the reset signals for the SCFIFO and DCFIFO, refer to Table 7 on page 18 or Table 8 on page 19 respectively.
q (3)	Output	Yes	Shows the data read from the read request operation. For the SCFIFO megafunction and DCFIFO megafunction, the width of the q port must be equal to the width of the data port. If you manually instantiate the megafunctions, ensure that the port width is equal to the lpm_width parameter. For the DCFIFO_MIXED_WIDTHS megafunction, the width of the q port can be different from the width of the data port. If you manually instantiate the megafunction, ensure that the width of the q port is equal to the lpm_width_r parameter. The megafunction supports a wide write port with a narrow read port, and vice versa. However, the width ratio is restricted by the type of RAM block used, and in general, are in the power of 2. Refer to "Different Input and Output Width" on page 19.

Table 1. Input and Output Ports Description (Part 3 of 3)

Port	Type	Required	Description
full (1) wrfull (2) rdfull (2)	Output	No	When asserted, the FIFO megafunction is considered full. Do not perform write request operation when the FIFO megafunction is full. In general, the rdfull signal is a delayed version of the wrfull signal. However, for Stratix III devices and later, the rdfull signal function as a combinational output instead of a derived version of the wrfull signal. Therefore, you must always refer to the wrfull port to ensure whether or not a valid write request operation can be performed, regardless of the target device.
empty (1) wrempty (2) rdrempty (2)	Output	No	When asserted, the FIFO megafunction is considered empty. Do not perform read request operation when the FIFO megafunction is empty. In general, the wrempty signal is a delayed version of the rdempty signal. However, for Stratix III devices and later, the wrempty signal function as a combinational output instead of a derived version of the rdempty signal. Therefore, you must always refer to the rdempty port to ensure whether or not a valid read request operation can be performed, regardless of the target device.
almost_full (1)	Output	No	Asserted when the usedw signal is greater than or equal to the almost_full_value parameter. It is used as an early indication of the full signal.
almost_empty (1)	Output	No	Asserted when the usedw signal is less than the almost_empty_value parameter. It is used as an early indication of the empty signal.
usedw (1) wrusedw (2) rdusedw (2)	Output	No	Show the number of words stored in the FIFO. Ensure that the port width is equal to the lpm_widthu parameter if you manually instantiate the SCFIFO megafunction or the DCFIFO megafunction. For the DCFIFO_MIXED_WIDTH megafunction, the width of the wrusedw and rdusedw ports must be equal to the LPM_WIDTHU and lpm_widthu_r parameters respectively. For Stratix, Stratix GX, and Cyclone devices, the FIFO megafunction shows full even before the number of words stored reaches its maximum value. Therefore, you must always refer to the full or wrfull port for valid write request operation, and the empty or rdempty port for valid read request operation regardless of the target device.

Notes to Table 1:

(1) Only applicable for the SCFIFO megafunction.

(2) Applicable for both of the DCFIFO megafunctions.

(3) Applicable for the SCFIFO, DCFIFO, and DCFIFO_MIXED_WIDTH megafunctions.

The output latency information of the FIFO megafunctions is important, especially for the q output port, because there is no output flag to indicate when the output is valid to be sampled. For more information about the output latency (including other status flags), refer to Table 3 on page 15 or Table 4 on page 16.

Parameter Specifications

This section describes the parameters that you can use to configure the megafunctions.

Table 2. Parameter Specifications (Part 1 of 4)

Parameter	Type	Required	Description
lpm_width	Integer	Yes	Specifies the width of the data and q ports for the SCFIFO megafunction and DCFIFO megafunction. For the DCFIFO_MIXED_WIDTHS megafunction, this parameter specifies only the width of the data port.
lpm_width_r *(1)*	Integer	Yes	Specifies the width of the q port for the DCFIFO_MIXED_WIDTHS megafunction.
lpm_widthu	Integer	Yes	Specifies the width of the usedw port for the SCFIFO megafunction, or the width of the rdusedw and wrusedw ports for the DCFIFO megafunction. For the DCFIFO_MIXED_WIDTHS megafunction, it only represents the width of the wrusedw port.
lpm_widthu_r *(1)*	Integer	Yes	Specifies the width of the rdusedw port for the DCFIFO_MIXED_WIDTHS megafunction.
lpm_numwords	Integer	Yes	Specifies the depths of the FIFO you require. The value must be at least **4**. The value assigned must comply with this equation, $2^{LPM_WIDTHU-1} < LPM_NUMWORDS \leq 2^{LPM_WIDTHU}$. For example, if the lpm_widthu parameter is **3**, the valid value for the lpm_numwords parameter is **5**, **6**, **7**, or **8**.
lpm_showahead	String	Yes	Specifies whether the FIFO is in normal mode (OFF) or show-ahead mode (ON). For normal mode, the FIFO megafunction treats the rdreq port as a normal read request that only performs read operation when the port is asserted. For show-ahead mode, the FIFO megafunction treats the rdreq port as a read-acknowledge that automatically outputs the first word of valid data in the FIFO megafunction (when the empty or rdempty port is low) without asserting the rdreq signal. Asserting the rdreq signal causes the FIFO megafunction to output the next data word, if available. If you set the parameter to **ON**, you may reduce performance.
lpm_type	String	No	Identifies the library of parameterized modules (LPM) entity name. The values are **SCFIFO** and **DCFIFO**.
maximize_speed *(2)*	Integer	No	Specifies whether or not to optimize for area or speed. The values are **0** through **10**. The values **0**, **1**, **2**, **3**, **4**, and **5** result in area optimization, while the values **6**, **7**, **8**, **9**, and 10 result in speed optimization. This parameter is applicable for Cyclone II and Stratix II devices only.

Table 2. Parameter Specifications (Part 2 of 4)

Parameter	Type	Required	Description
`overflow_checking`	String	No	Specifies whether or not to enable the protection circuitry for overflow checking that disables the `wrreq` port when the FIFO megafunction is full. The values are **ON** or **OFF**. If omitted, the default is **ON**.
`underflow_checking`	String	No	Specifies whether or not to enable the protection circuitry for underflow checking that disables the `rdreq` port when the FIFO megafunction is empty. The values are **ON** or **OFF**. If omitted, the default is **ON**. Note that reading from an empty SCFIFO gives unpredictable results.
`delay_rdusedw` *(2)* `delay_wrusedw` *(2)*	String	No	Specify the number of register stages that you want to internally add to the `rdusedw` or `wrusedw` port using the respective parameter. The default value of `1` adds a single register stage to the output to improve its performance. Increasing the value of the parameter does not increase the maximum system speed. It only adds additional latency to the respective output port.
`add_usedw_msb_bit` *(2)*	String	No	Increases the width of the `rdusedw` and `wrusedw` ports by one bit. By increasing the width, it prevents the FIFO megafunction from rolling over to zero when it is full. The values are **ON** or **OFF**. If omitted, the default value is **OFF**. This parameter is only applicable for Stratix and Cyclone series (except for Stratix, Stratix GX, and Cyclone devices).
`rdsync_delaypipe` *(2)* `wrsync_delaypipe` *(2)*	Integer	No	Specify the number of synchronization stages in the cross clock domain. The value of the `rdsync_delaypipe` parameter relates the synchronization stages from the write control logic to the read control logic; the `wrsync_delaypipe` parameter relates the synchronization stages from the read control logic to the write control logic. Use these parameters to set the number of synchronization stages if the clocks are not synchronized, and set the `clocks_are_synchronized` parameter to **FALSE**. The actual synchronization stage implemented relates variously to the parameter value assigned, depends on the target device. For Cyclone II and Stratix II devices and later, the values of these parameters are internally reduced by two. Thus, the default value of **3** for these parameters corresponds to a single synchronization stage; a value of **4** results in two synchronization stages, and so on. For these devices, choose at least **4** (two synchronization stages) for metastability protection. Refer to "Metastability Protection and Related Options" on page 16.

Table 2. Parameter Specifications (Part 3 of 4)

Parameter	Type	Required	Description
`use_eab`	String	No	Specifies whether or not the FIFO megafunction is constructed using the RAM blocks. The values are **ON** or **OFF**. Setting this parameter value to **OFF** yields the FIFO megafunction implemented in logic elements regardless of the type of the TriMatrix memory block type assigned to the `ram_block_type` parameter.
`write_aclr_synch` *(2)*	String	No	Specifies whether or not to add a circuit that causes the `aclr` port to be internally synchronized by the `wrclk` clock. Adding the circuit prevents the race condition between the `wrreq` and `aclr` ports that could corrupt the FIFO megafunction. The values are **ON** or **OFF**. If omitted, the default value is `OFF`. This parameter is only applicable for Stratix and Cyclone series (except for Stratix, Stratix GX, and Cyclone devices).
`clocks_are_synchronized` *(2)*	String	No	Specifies whether or not the write and read clocks are synchronized which in turn determines the number of internal synchronization stages added for stable operation of the FIFO. The values are **TRUE** and **FALSE**. If omitted, the default value is **FALSE**. You must only set the parameter to **TRUE** if the write clock and the read clock are always synchronized and they are multiples of each other. Otherwise, set this to **FALSE** to avoid metastability problems. If the clocks are not synchronized, set the parameter to **FALSE**, and use the `rdsync_delaypipe` and `wrsync_delaypipe` parameters to determine the number of synchronization stages required.
`ram_block_type`	String	No	Specifies the target device's Trimatrix Memory Block to be used. To get the proper implementation based on the RAM configuration that you set, allow the Quartus II software to automatically choose the memory type by ignoring this parameter and set the `use_eab` parameter to **ON**. This gives the compiler the flexibility to place the memory function in any available memory resource based on the FIFO depth required.
`add_ram_output_register`	String	No	Specifies whether to register the q output. The values are **ON** and **OFF**. If omitted, the default value is **OFF**. You can set the parameter to **ON** or **OFF** for the SCFIFO or the DCFIFO, that do not target Stratix II, Cyclone II, and new devices. This parameter does not apply to these devices because the q output must be registered in normal mode and unregistered in show-ahead mode for the DCFIFO.
`almost_full_value` *(3)*	Integer	No	Sets the threshold value for the `almost_full` port. When the number of words stored in the FIFO megafunction is greater than or equal to this value, the `almost_full` port is asserted.

Table 2. Parameter Specifications (Part 4 of 4)

Parameter	Type	Required	Description
almost_empty_value (3)	Integer	No	Sets the threshold value for the almost_empty port. When the number of words stored in the FIFO megafunction is less than this value, the almost_empty port is asserted.
allow_wrcycle_when_full (3)	String	No	Allows you to combine read and write cycles to an already full SCFIFO, so that it remains full. The 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**.
intended_device_family	String	No	Specifies the intended device that matches the device set in your Quartus II project. This parameter is only used for functional simulation.

Notes to Table 2:

(1) Only applicable for the DCFIFO_MIXED_WIDTHS megafunction.

(2) Only applicable for the DCFIFO.

(3) Only applicable for the SCFIFO megafunction.

Functional Timing Requirements

The wrreq signal is ignored (when FIFO is full) if you enable the overflow protection circuitry in the FIFO parameter editor, or set the OVERFLOW_CHECKING parameter to ON. The rdreq signal is ignored (when FIFO is empty) if you enable the underflow protection circuitry in the FIFO MegaWizard interface, or set the UNDERFLOW_CHECKING parameter to ON.

If the protection circuitry is not enabled, you must meet the following functional timing requirements:

- DCFIFO
  - Deassert the wrreq signal in the same clock cycle when the wrfull signal is deasserted.
  - Deassert the rdreq signal in the same clock cycle when the rdempty signal is asserted.

  ☞ You must observe these requirements regardless of expected behavior based on wrclk and rdclk frequencies.

- SCFIFO
  - Deassert the wrreq signal in the same clock cycle when the full signal is asserted.
  - Deassert the rdreq signal in the same clock cycle when the empty signal is asserted.

Figure 2 shows the behavior for the `wrreq` and the `wrfull` signals.

Figure 2. Functional Timing for the wrreq Signal and the wrfull Signal

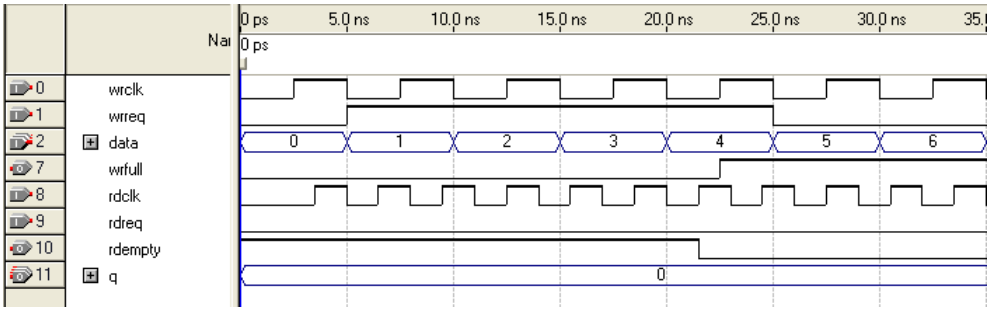
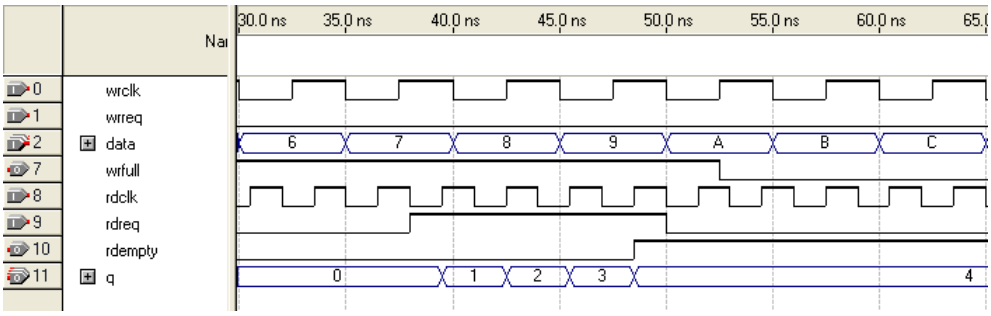


Figure 3 shows the behavior for the `rdreq` the `rdempty` signals.

Figure 3. Functional Timing for the rdreq Signal and the rdempty Signal



The required functional timing for the DCFIFO as described previously is also applied to the SCFIFO. The difference between the two modes is that for the SCFIFO, the `wrreq` signal must meet the functional timing requirement based on the `full` signal and the `rdreq` signal must meet the functional timing requirement based on the `empty` signal.

Output Status Flags and Latency

The main concern in most FIFO design is the output latency of the read and write status signals. Table 3 shows the output latency of the write signal (`wrreq`) and read signal (`rdreq`) for the SCFIFO according to the different output modes and optimization options.

Table 3. Output Latency of the Status Flags for SCFIFO

Output Mode	Optimization Option (1)	Output Latency (in number of clock cycles) (2)
Normal (3)	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 (3)	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
	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

Notes to Table 3:

(1) Speed optimization is equivalent to setting the ADD_RAM_OUTPUT_REGISTER parameter to ON. Setting the parameter to OFF is equivalent to area optimization.

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

(3) For the Quartus II software versions earlier than 9.0, the normal output mode is called legacy output mode. Normal output mode is equivalent to setting the LPM_SHOWAHEAD parameter to OFF. For Show-ahead mode, the parameter is set to ON.

Table 4 shows the output latency of the write signal (wrreq) and read signal (rdreq) for the DCFIFO.

Table 4. Output Latency of the Status Flag for the DCFIFO (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 (4) + 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 (4)
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 4:

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

(2) The number of *n* cycles for rdclk and wrclk is equivalent to the number of synchronization stages used and are related to the WRSYNC_DELAYPIPE and RDSYNC_DELAYPIPE parameters. For more information about how the actual synchronization stage (*n*) is related to the parameters set for different target device, refer to Table 6 on page 17.

(3) You must keep the wrclk running for at least one cycle after the wrreq signal goes low. Otherwise, the last word written does not appear at the output. This is because rdempty goes high when the second-to-last word is displayed at the output due to address counters not propagating from the wrclk side to the rdclk side.

(4) This is applied only to Show-ahead output modes. Show-ahead output mode is equivalent to setting the LPM_SHOWAHEAD parameter to ON.

Metastability Protection and Related Options

The FIFO parameter editor provides the total latency, clock synchronization, metastability protection, area, and f_{MAX} options as a group setting for the DCFIFO.

Table 5 shows the available group setting.

Table 5. DCFIFO Group Setting for Latency and Related Options

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

The group setting for latency and related options is available through the FIFO parameter editor. The setting mainly determines the number of synchronization stages used, depending on the group setting you select. You can also set the number of synchronization stages you desire through the `WRSYNC_DELAYPIPE` and `RDSYNC_DELAYPIPE` parameters, but you need to understand how the actual number of synchronization stages relates to the parameter values set in different target devices.

The **number of synchronization stages** 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 6 shows the relationship between the actual synchronization stage and the pipeline parameters.

Table 6. Relationship between the Actual Synchronization Stage and the Pipeline Parameters for Different Target Devices

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

Notes to Table 6:

(1) You can obtain the low-latency of the DCFIFO (for Stratix, Stratix GX, and Cyclone devices) when the clocks are not set to synchronized in Show-ahead mode with unregistered output in the FIFO parameter editor. 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 to Stratix, Stratix GX, and Cyclone devices.

(2) The values assigned to `WRSYNC_DELAYPIPE` and `RDSYNC_DELAYPIPE` parameters are internally reduced by 2 to represent the actual synchronization stage implemented. Thus, the default value 3 for these parameters corresponds to a single synchronization pipe stage; a value of 4 results in 2 synchronization stages, and so on. For these devices, choose 4 (2 synchronization stages) for metastability protection.

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. This report includes the MTBF analysis on the synchronization pipeline you applied between the asynchronous clock domains in your DCFIFO. You can then decide the number of synchronization stages to use in order to meet the range of the MTBF specification you require.

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

Synchronous Clear and Asynchronous Clear Effect

The FIFO megafunctions support 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. The SCFIFO supports both synchronous and asynchronous clear signals while the DCFIFO support asynchronous clear signal and asynchronous clear signal that synchronized with the write clock.

Table 7 shows the synchronous clear and asynchronous clear signals supported in the SCFIFO.

Table 7. Synchronous Clear and Asynchronous Clear in the SCFIFO

Mode	Synchronous Clear (sclr)	Asynchronous Clear (aclr)
Effects on status ports	Deasserts the `full` and `almost_full` signals.	
	Asserts the `empty` and `almost_empty` signals.	
	Resets the `usedw` flag.	
Commencement of effects upon assertion	At the rising edge of the clock.	Immediate (except for the q output)
Effects on the q output for normal output modes	The read pointer is reset and points to the first data location. If the q output is not registered, the output shows the first data word of the SCFIFO; otherwise, the q output remains at its previous value.	The q output remains at its previous value.
Effects on the q output for show-ahead output modes	The read pointer is reset and points to the first data location. 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 7:

(1) The first data word shown after the reset is not a valid Show-ahead data. It reflects the data where the read pointer is pointing to because the q output is not registered. To obtain a valid Show-ahead data, perform a valid write after the reset.

Table 8 shows the asynchronous clear supported by the DCFIFO.

Table 8. Asynchronous Clear in DCFIFO

Mode	Asynchronous Clear (aclr)	aclr (synchronize with write clock) (1), (2)
Effects on status ports	Deasserts the `wrfull` signal.	Asserts the `wrfull` signal for three rising edges of write clock before deasserting the signal.
	Deasserts the `rdfull` signal.	
	Asserts the `wrempty` and `rdempty` signals.	
	Resets the `wrusedw` and `rdusedw` flags.	
Commencement of effects upon assertion	Immediate.	
Effects on the q output for normal output modes (3)	The output remains unchanged if it is not registered. If the port is registered, it is cleared.	
Effect on the q output for show-ahead output modes (3)	The output shows 'X' if it is not registered. If the port is registered, it is cleared.	

Notes to Table 8:

(1) The `wrreq` signal must be low when the DCFIFO comes out of reset (the instant when the `aclr` signal is deasserted) 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 parameter editor, or setting the `WRITE_ACLR_SYNCH` paramter to `ON`.

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

(3) For Stratix and Cyclone series (except Stratix, Stratix GX, and Cyclone devices), the DCFIFO only supports registered q output in Normal mode, and unregistered q output in Show-ahead mode. For other devices, you have an option to register or unregister the q output (regardless of the Normal mode or Show-ahead mode used) in the FIFO parameter editor or set through the ADD_RAM_OUTPUT_REGISTER parameter.

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.

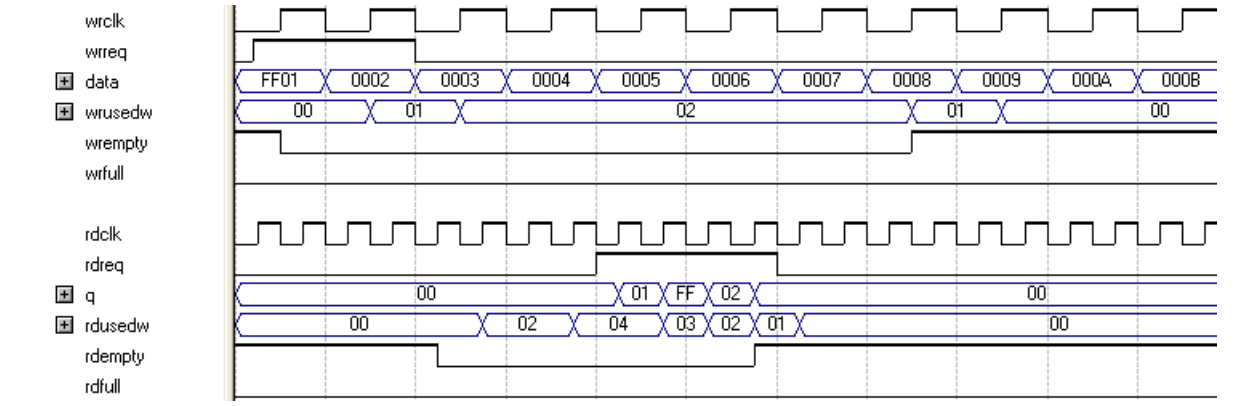
Different Input and Output Width

The DCFIFO_MIXED_WIDTHS megafunction supports different write input data and read output data widths if the width ratio is valid. The FIFO parameter editor prompts an error message if the combinations of the input and the output data widths produce an invalid ratio. The supported width ratio typically in a power of 2 and depends on the RAM used.

The megafunction supports a wide write port with a narrow read port, and vice versa.

Figure 4 shows an example of a wide write port (16-bit input) and a narrow read port (8-bit output).

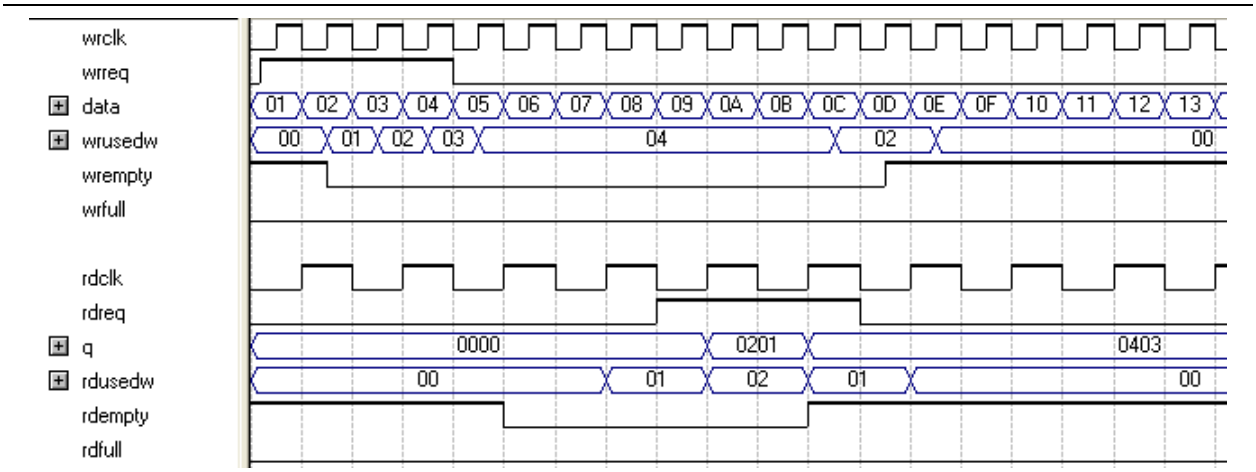
Figure 4. Writing 16-bit Words and Reading 8-bit Words



In this example, the read port is operating at twice the frequency of the write port. Writing two 16-bit words to the FIFO buffer increases the `wrusedw` flag to two and the `rusedw` flag to four. Four 8-bit read operations empty the FIFO buffer. The read begins with the least-significant 8 bits from the 16-bit word written followed by the most-significant 8 bits.

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

Figure 5. 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. Writing four 8-bit words to the FIFO buffer increases the `wrusedw` flag to four and the `rusedw` flag to two. Two 16-bit read operations empty the FIFO. The first and second 8-bit word written are equivalent to the LSB and MSB of the 16-bit output words, respectively. The `rdempty` signal stays asserted until enough words are written on the narrow write port to fill an entire word on the wide read port.

Constraint Settings

When using the Quartus II TimeQuest timing analyzer with a design that contains a DCFIFO block apply the following false paths to avoid timing failures in the synchronization registers:

- 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*}]
```

- 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*}]
```

In the Quartus II software version 8.1 and later, the false path assignments are automatically added through the HDL-embedded Synopsis design constraint (SDC) commands when you compile your design. The related message is shown under the TimeQuest timing analyzer report.

The constraints are internally applied but are not written to the **.sdc** file. To view the embedded-false path, type **report_sdc** in the console pane of the TimeQuest timing analyzer GUI.

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

If the DCFIFO is implemented in logic elements (LEs), you can ignore the cross-domain timing violations from the data path of the DFFE array (that makes up the memory block) to the `q` output register. To ensure the `q` output is valid, sample the output only after the `rdempty` signal is deasserted.

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*.

Coding Example for Manual Instantiation

This section provides a Verilog HDL coding example to instantiate the DCFIFO megafunction. It is not a complete coding for you to compile, but it provides a guideline and some comments for the required structure of the instantiation. You can use the same structure to instantiate other megafunctions but only with the ports and parameters that are applicable to the megafunctions you instantiated.

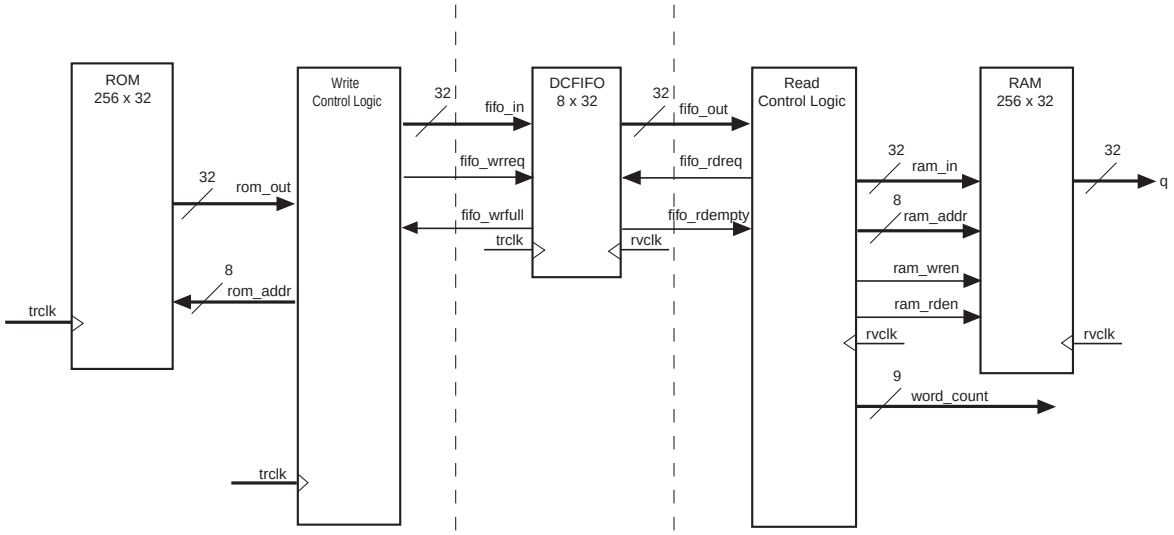
Example 1. Verilog HDL Coding Example to Instantiate the DCFIFO Megafunction

```
//module declaration
module dcfifo8x32 (aclr, data, …… ,wfull);

//Module's port declarations
input aclr;
input [31:0] data;
.
.
output wrfull;

//Module’s data type declarations and assignments
wire rdempty_w;
.
.
wire wrfull = wrfull_w;
wire [31:0] q = q_w;

/*Instantiates dcfifo megafunction. Must declare all the ports available from the
megafunction and define the connection to the module's ports.
Refer to the ports specification from the user guide for more information about the
megafunction's ports*/

//syntax: <megafunction's name> <given an instance name>
dcfifo inst1 (

//syntax: .<dcfifo's megafunction's port>(<module's port/wire>)
.wrclk (wrclk),
.rdclk (rdreq),
.
.
.wrusedw ()); //left the output open if it's not used

/*Start with the keyword “defparam”, defines the parameters and value assignments. Refer to
parameters specifications from the user guide for more information about the megafunction's
parameters*/
defparam

//syntax: <instance name>.<parameter> = <value>
inst1.intended_device_family = "Stratix III",
inst1.lpm_numwords = 8,
.
.
inst1.wrsync_delaypipe = 4;

endmodule
```

Design Example

In this design example, the data from the ROM is required to be transferred to the RAM. Assuming the ROM and RAM are driven by non-related clocks, the DCFIFO can be used to effectively transfer the data between the asynchronous clock domains. Figure 6 illustrates the component blocks and their signal interactions.

Figure 6. Component Blocks and Signal Interaction



Both the DCFIFO megafunctions are only capable of handling asynchronous data transferring issues (metastable effects). You must have a controller to govern and monitor the data buffering process between the ROM, DCFIFO, and RAM. This design example provides you the write control logic (**write_control_logic.v**), and the read control logic (**read_control_logic.v**) which are compiled with the DCFIFO specifications that control the valid write or read request to or from the DCFIFO.

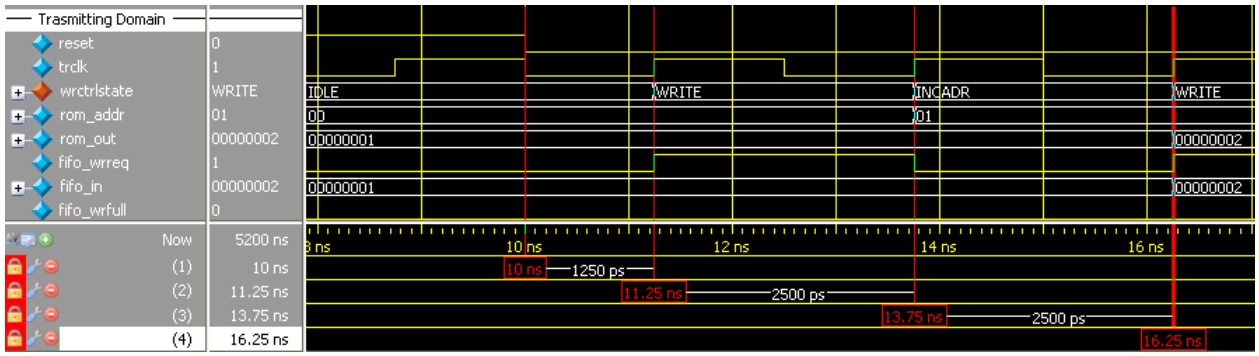
This design example is validated with its functional behavior, but without timing analysis and gate-level simulation. The design coding such as the state machine for the write and read controllers may not be optimized. The intention of this design example is to show the use the megafunction, particularly on its control signal in data buffering application, rather than the design coding and verification processes.

To obtain the DCFIFO settings used in this design example, refer to the parameter settings from the design file (**dcfifo8x32.v**). You can get all the design files including the testbench from the **dcfifo_example.zip** file from the Documentation: User Guides page on the Altera website. The zip file also includes the **.do** script (**dcfifo_de_top.do**) that automates functional simulation that you can use to run the simulation using the ModelSim®-Altera software.

The following sections include separate simulation waveforms to describe how the write and read control logics generate the control signal with respect to the signal received from the DCFIFO.

For better understanding, refer to the signal names in Figure 6 on page 23 when you go through the descriptions for the simulation waveforms.

Figure 7. Initial Write Operation to the DCFIFO Megafunction



Notes to Figure 7:

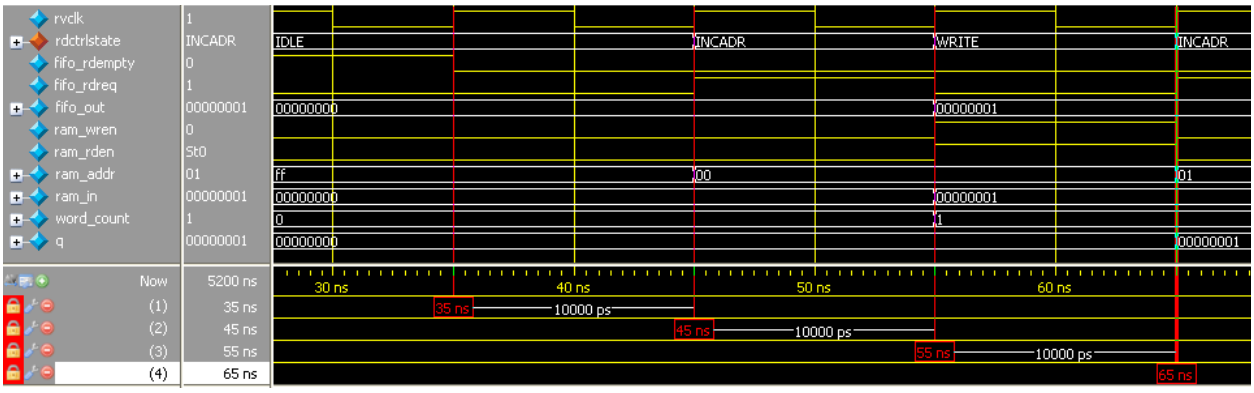
(1) Before reaching 10 ns, the `reset` signal is high and causes the write controller to be in the IDLE state. In the IDLE state, the write controller drives the `fifo_wrreq` signal to low, and requests the data to be read from `rom_addr`=00. The ROM is configured to have an unregistered output, so that the `rom_out` signal immediately shows the data from the `rom_addr` signal regardless of the reset. This shortens the latency because the `rom_out` signal is connected directly to the `fifo_in` signal, which is a registered input port in the DCFIFO. In this case, the data (00000001) is always stable and pending to be written into the DCFIFO when `fifo_wrreq` signal is high during the WRITE state.

(2) The write controller transitions from the IDLE state to the WRITE state if the `fifo_wrfull` signal is low after the `reset` signal is deasserted. In the WRITE state, the write controller drives the `fifo_wrreq` signal to high, and requests for write operation to the DCFIFO. The `rom_addr` signal is unchanged (00) so the data is stable for at least one clock cycle before the DCFIFO actually writes in the data at the next rising clock edge.

(3) The write controller transitions from the WRITE state to the INCADR state, if the `rom_addr` signal has not yet increased to ff (that is, the last data from the ROM has not been read out). In the INDADR state, the write controller drives the `fifo_wrreq` signal to low, and increases the `rom_addr` signal by 1 (00 to 01).

(4) The same state transition continues as stated in note *(2)* and note *(3)*, if the `fifo_wrfull` signal is low and the `rom_addr` signal not yet increased to ff.

Figure 8. Initial Read Operation from the DCFIFO Megafunction



Notes to Figure 8:

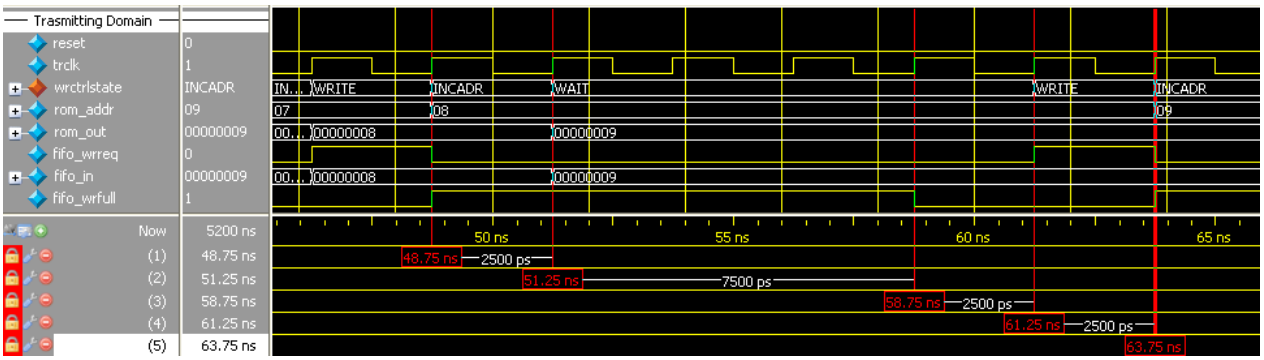
(1) Before reaching 35 ns, the read controller is in the IDLE state because the `fifo_rdempty` signal is high even when the `reset` signal is low (not shown in the figure). In the IDLE state, the `ram_addr` = ff to accommodate the increment of the RAM address in the INCADR state, so that the first data read is stored at `ram_addr` = 00 in the WRITE state.

(2) The read controller transitions from the IDLE state to the INCADR state, if the `fifo_rdempty` signal is low. In the INCADR state, the read controller drives the `fifo_rdreq` signal to high, and requests for read operation from the DCFIFO. The `ram_addr` signal is increased by one (ff to 00), so that the read data can be written into the RAM at `ram_addr` = 00.

(3) From the INCADR state, the read controller always transition to the WRITE state at the next rising clock edge. In the WRITE state, it drives the `ram_wren` signal to high, and enables the data writing into the RAM at `ram_addr` = 00. At the same time, the read controller drives the `ram_rden` signal to high so that the newly written data is output at q at the next rising clock edge. Also, it increases the `word_count` signal to 1 to indicate the number of words successfully read from the DCFIFO.

(4) The same state transition continues as stated in note *(2)* and note *(3)* if the `fifo_rdempty` signal is low.

Figure 9. Write Operation when DCFIFO is FULL



Notes to Figure 9:

(1) When the write controller is in the INCADR state, and the `fifo_wrfull` signal is asserted, the write controller transitions to the WAIT state in the next rising clock edge.

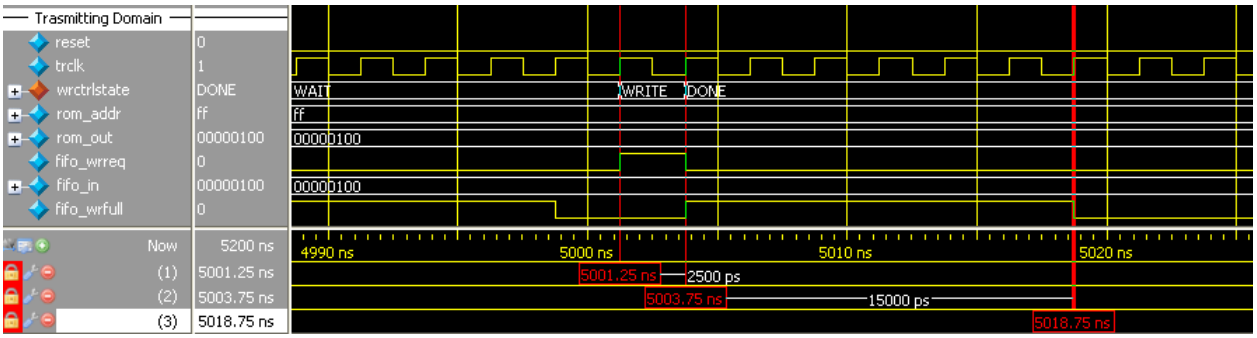
(2) In the WAIT state, the write controller holds the `rom_addr` signal (08) so that the respective data is written into the DCFIFO when the write controller transitions to the WRITE state.

(3) The write controller stays in WAIT state if the `fifo_wrfull` signal is still high. When the `fifo_wrfull` is low, the write controller always transitions from the WAIT state to the WRITE state at the next rising clock edge.

(4) In the WRITE state, then only the write controller drives the `fifo_wrreq` signal to high, and requests for write operation to write the data from the previously held address (08) into the DCFIFO. It always transitions to the INCADR state in the next rising clock edge, if the `rom_addr` signal has not yet increased to ff.

(5) The same state transition continues as stated in Notes *(1)* through *(4)* if the `fifo_wrfull` signal is high.

Figure 10. Completion of Data Transfer from ROM to DCFIFO



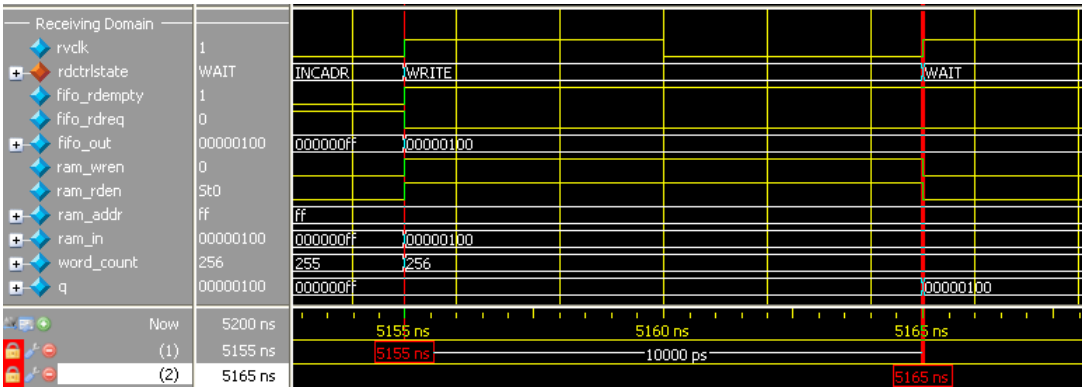
Notes to Figure 10:

(1) When the write controller is in the WRITE state, and `rom_addr` = ff, the write controller drives the `fifo_wrreq` signal to high to request for last write operation to DCFIFO. The data 100 is the last data stored in the ROM to be written into the DCFIFO. In the next rising clock edge, the write controller transitions to the DONE state.

(2) In the DONE state, the write controller drives the `fifo_wrreq` signal to low.

(3) The `fifo_wrfull` signal is deasserted because the read controller in the receiving domain continuously performs the read operation. However, the `fifo_wrfull` signal is only deasserted sometime after the read request from the receiving domain. This is due to the latency in the DCFIFO (`rdreq` signal to `wrfull` signal).

Figure 11. Completion of Data Transfer from DCFIFO to RAM



Notes to Figure 11:

(1) The `fifo_rdempty` signal is asserted to indicate that the DCFIFO is empty. The read controller drives the `fifo_rdreq` signal to low, and enables the write of the last data 100 at `ram_addr` =ff. The `word_count` signal is increased to 256 (in decimal) to indicate that all the 256 words of data from the ROM are successfully transferred to the RAM.

(2) The last data written into the RAM is shown at the q output.

To verify the results, compare the `q` outputs with the data in **rom_initdata.hex** file provided in the design example. Open the file in the Quartus II software and select the word size as 32 bit. The `q` output must display the same data as in the file.

DCFIFO Clock Domain Crossing Timing Violation

This section describes a workaround to overcome a possible timing violation that might occur when transferring data in the cross clock domain or using large clock skew.

To ensure that DCFIFO megafunction works properly when transferring data from the write domain to the read domain and vice versa, you must use the gray-code counter to ensure only 1-bit transition occurs while other bits remain stable. If the destination domain latches on the data within the metastable range (violating setup or hold time), only 1 bit is uncertain and destination domain reads the counter value as either an old counter or a new counter. In this case, the DCFIFO still works as long as the counter sequence is not corrupted (for example, skipping to the next counter when >1-bit transition occurs during timing violation). However, there's a possibility that the uncertain bit stables at a new counter. Even if the bit stables at the old counter, you still cannot be sure if it is an old counter or a new counter. This is because it cannot read a number of data that is more than what is available in the DCFIFO megafunction.

The problem with >1-bit transition occuring even if the DCFIFO implements the gray-code is due to the skew of the bus when they arrive at the cross domain bus registers.

The following section shows an example of the possible timing violation when counter width is 3-bit wide assuming that the counter bits have 0 delay relative to the destination clock (`rdclk`), excluding `bit[0]` with a skew of 1 clock period of source clock (`wrclk`).

This setting results in the following gray-code count sequence:

000,

001,

011,

010,

110....

which then transfers the data to the read domain, and on to the destination bus registers.

Because of the skew for `bit[0]`, the destination bus registers receive the following sequence:

000,

000,

011,

011,

110....

Note that because of the skew, a 2-bit transition occurs. This sequence is acceptable if the timing is met otherwise metastability happens and the 2-bit transition settles at a future or previous counter value, which corrupts the sequence.

Therefore, the skew must be within a certain skew to ensure that the sequence is not corrupted. The allowable skew is expressed by the following equation:

Maximum skew allowed = Period (source clk)

By default, the `set_max_skew` constraints already include the micro parameters—register micro-Tsu time (utsu), and register micro-Th time, (utho). However, these micro parameters are excluded from the previous equation because these parameters are automatically added when you calculate the relative delay for the latest arrival path and the earliest arrival path.

To implement the maximum skew, remove the false path and apply the following timing constraints:

```
set wrclk_per 5

set rdclk_per 10

#estimation guardband for ReqSW = utsu + utho , note utsu/utho can be
different for different registers

#set ReqSW 0.1

#set maxskew_w2r [expr $wrclk_per - $ReqSW]

#set maxskew_r2w [expr $rdclk_per - $ReqSW]


#by default, the set_max_skew alr accounted the utsu and utho,
therefore,

set maxskew_w2r $wrclk_per

set maxskew_r2w $rdclk_per


create_clock -period $wrclk_per -name wrclk [get_ports wrclk]

create_clock -period $rdclk_per -name rdclk [get_ports rdclk]

set_max_skew $maxskew_w2r -from [get_registers *delayed_wrptr_g*] -to
[get_registers *rs_dgwp*]

set_max_skew $maxskew_r2w -from [get_registers *rdptr_g*] -to
[get_registers *ws_dgrp*]
```

Document Revision History

Table 9 lists the revision history for this document.

Table 9. Document Revision History (Part 1 of 2)

Date	Version	Changes
February 2012	7.0	■ Updated the notes for Table 4 on page 16. ■ Added the "DCFIFO Clock Domain Crossing Timing Violation" section.
September 2010	6.2	Added prototype and component declarations.
January 2010	6.1	■ Updated "Functional Timing Requirements" section. ■ Minor changes to the text.

Table 9. Document Revision History (Part 2 of 2)

Date	Version	Changes
September 2009	6.0	■ Replaced “FIFO Megafunction Features” section with “Configuration Methods”. ■ Updated “Input and Output Ports”. ■ Added “Parameter Specifications”, “Output Status Flags and Latency”, “Metastability Protection and Related Options”, “Constraint Settings”, “Coding Example for Manual Instantiation”, and “Design Example”.
February 2009	5.1	Minor update in Table 8 on page 17.
January 2009	5.0	Complete re-write of the user guide.
May 2007	4.0	■ Added support for Arria GX devices. ■ Updated for new GUI. ■ Added six design examples in place of functional ■ description. ■ Reorganized and updated Chapter 3 to have separate tables for the SCFIFO and DCFIFO megafunctions. ■ Added Referenced Documents section.
March 2007	3.3	■ 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.