FIFO Partitioner Megafunction

User Guide

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

Document Version: 1.2
Document Date: August 2005

Contents

About this User Guide

This user guide provides comprehensive information about the Altera® FIFO Partitoner.

Table 1 shows the user guide revision history.

Table 1. User Guide Revision History

Date	Description
August 2005	Updated with missing images, minor edits.
December 2004	Updated the *How to Contact Altera* section- version 1.1
January 2003	Initial release - version 1.0

How to Find Information

- When using this document in PDF format, the Adobe Acrobat Find feature allows you to search the contents of the document. Click the binoculars toolbar icon in the Adobe Acrobat software to open the Find dialog box.
- Bookmarks serve as an additional table of contents.
- Thumbnail icons, which provide miniature previews of each page, provide a link to the pages.
- Numerous links, shown in green text, allow you to jump to related information.

How to Contact Altera

For the most up-to-date information about Altera products, go to the Altera world-wide web site at www.altera.com. For technical support on this product, go to www.altera.com/mysupport. For additional information about Altera products, consult the sources shown below.

Information Type	USA & Canada	All Other Locations
Technical support	www.altera.com/mysupport/	www.altera.com/mysupport/
	(800) 800-EPLD (3753) (7:00 a.m. to 5:00 p.m. Pacific Time)	+1 408-544-8767 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
Product literature	www.altera.com	www.altera.com
Altera literature services	literature@altera.com	literature@altera.com
Non-technical customer service	(800) 767-3753	+ 1 408-544-7000 7:00 a.m. to 5:00 p.m. (GMT -8:00) Pacific Time
FTP site	ftp.altera.com	ftp.altera.com

Typographic Conventions

The *FIFO Partitioner Megafunction User Guide* uses the typographic conventions shown in Table 2.

Table 2. Conventions

Visual Cue	Meaning
Bold Type with Initial Capital Letters	Command names, dialog box titles, checkbox options, and dialog box options are shown in bold, initial capital letters. Example: **Save As** dialog box.
bold type	External timing parameters, directory names, project names, disk drive names, filenames, filename extensions, and software utility names are shown in bold type. Examples: $\mathbf{f_{MAX}}$, **\QuartusII** directory, **d:** drive, **chiptrip.gdf** file.
Bold italic type	Book titles are shown in bold italic type with initial capital letters. Example: ***1999 Device Data Book***.
Italic Type with Initial Capital Letters	Document titles are shown in italic type with initial capital letters. Example: *AN 75 (High-Speed Board Design).*
Italic type	Internal timing parameters and variables are shown in italic type. Examples: t_{PIA}, $n + 1$. Variable names are enclosed in angle brackets (< >) and shown in italic type. Example: <*file name*>, <*project name*>**.pof** file.
Initial Capital Letters	Keyboard keys and menu names are shown with initial capital letters. Examples: Delete key, the Options menu.
"Subheading Title"	References to sections within a document and titles of Quartus II Help topics are shown in quotation marks. Example: "Configuring a FLEX 10K or FLEX 8000 Device with the BitBlaster™ Download Cable."

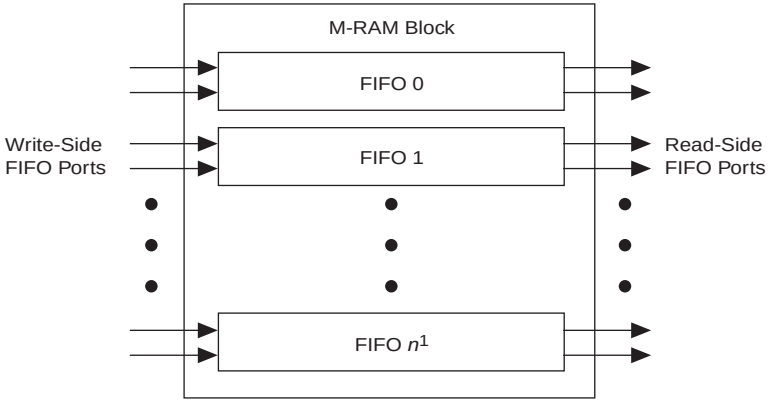
Table 2. Conventions

Visual Cue	Meaning
`Courier type`	Signal and port names are shown in lowercase Courier type. Examples: `data1`, `tdi`, `input.` Active-low signals are denoted by suffix n, e.g., `resetn`. Anything that must be typed exactly as it appears is shown in Courier type. For example: `c:\quartusII\qdesigns\tutorial\chiptrip.gdf`. Also, sections of an actual file, such as a Report File, references to parts of files (e.g., the AHDL keyword `SUBDESIGN`), as well as logic function names (e.g., `TRI`) are shown in Courier.
1., 2., 3., and a., b., c.,...	Numbered steps are used in a list of items when the sequence of the items is important, such as the steps listed in a procedure.
■	Bullets are used in a list of items when the sequence of the items is not important.
✓	The checkmark indicates a procedure that consists of one step only.
☞	The hand points to information that requires special attention.
↵	The angled arrow indicates you should press the Enter key.
👣	The feet direct you to more information on a particular topic.

Introduction

The FIFO Partitioner is a tool for mapping multiple FIFOs into a single physical memory block. The ability to partition M-RAM blocks into multiple FIFOs greatly enhances the utility of the TriMatrix memory architecture. The FIFO Partitioner automatically generates the logic necessary to time-domain-multiplex Stratix M-RAM blocks between multiple user-specified FIFOs. Use the FIFO Partitioner for FIFO functions that can't be fit efficiently into available M512 or M4K blocks.

Figure 1. User's View of a FIFO Partitioner Function Note (1)



Notes:

(1) Total number of FIFOs, *n*, should not exceed 10 for most designs. Up to 256 FIFOs are possible for designs with moderate performance requirements.

Features

Table 1 summarizes key features of the FIFO Partitioner.

Table 1. FIFO Partitioner Features (1)

Feature	Description
Partition M-RAM Blocks into Multiple FIFO functions	Automatically generates all circuitry required to implement multiple, fully independent FIFOs in a single M-RAM Block. User specifies width, depth, frequency, and behavior of each FIFO individually, or specifies multiple identical FIFOs.
Dual-clock FIFO Operation	Read and Write ports of each FIFO can be fully asynchronous to each other. This makes the FIFO Partitioner ideal for buffering multiple channels of data between clock domains.
Mixed-width FIFO Operation	Possible to write data in at one width and read it out at another width. This feature is ideal for aggregating data for parallel processing tasks. *(1)*
Read Request Mode Operation	Data becomes available after `read_b_`*n* is asserted. This mode is similar to the Legacy synchronous FIFO mode for LPM_FIFO MegaFunctions.
Read Acknowledge Mode Operation	Data becomes available before `read_b_`*n* is asserted. In this mode, `read_b_`*n* acts as a read acknowledge. This mode is similar to the Show-ahead synchronous FIFO mode for LPM_FIFO MegaFunctions.
Supports parity bits and wide data configurations	8, 9, 16, 18, 32, 36, 64, 72, 132, and 144-bit wide configurations supported.*(1)*

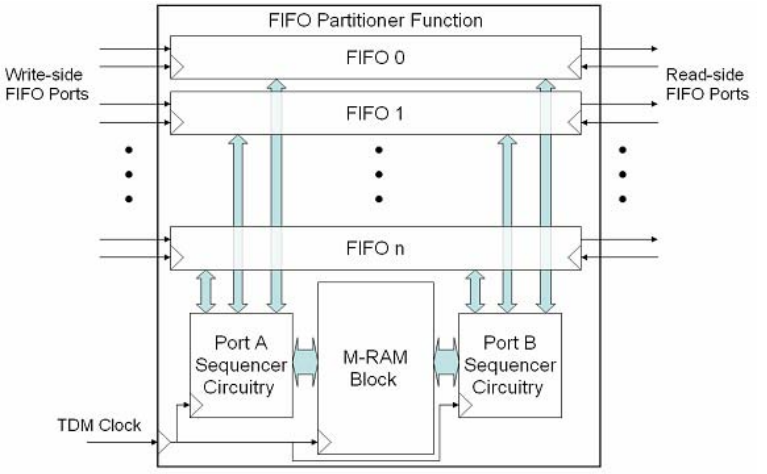
Note to Table 1:
(1) Maximum width for mixed-width configurations is 72.

Time-Domain-Multiplexing M-RAM Blocks For Multiple FIFOs

Each FIFO operates independently. Each FIFO port has unique clock, data, control input, and status flag output signals. Total memory bandwidth is shared, limiting the speed at which each FIFO can be accessed. Refer to Figure 1.

FIFO ports can be operated in completely asynchronous clock domains. You must supply frequency information for each FIFO, but you do not need to specify the phase relationship between clock domains. You must specify a clock called `TDM_clk` which is used to time-domain-multiplex access to the physical memory. A sequencer circuit transfers address, data and control information between the clock domains of each FIFO and the TDM clock domain (Figure 2).

Figure 2. Detailed view of FIFO Partitioner function



When a read or write operation to a FIFO port is initiated, the sequencer circuitry accesses the physical memory in the fast TDM clock domain. The sequencer then transfers information back to the clock domain of the FIFO port completing the memory access. Because of these clock-domain transfers, performance of the FIFOs is limited by the speed of the sequencer circuitry and the number of FIFOs in the FIFO Partitioner function.

Performance of FIFO Partitioner Functions

To ensure that FIFO memory accesses can be serviced, the frequency of the TDM clock must run at or above Nx the speed of the fastest FIFO port, where $N \geq 3$. (For FIFO Partitioner functions more than 3 FIFOs, N equals the number of FIFOs.)

The upper bound of the TDM clock is the maximum speed of the memory (approximately 275 MHz for Stratix™ M-RAM blocks). Depending on how you configure the FIFO Partitioner, the sequencer circuitry may limit performance.

Considering the restriction on the TDM_clk, it is easy to estimate the performance of each FIFO based on the number of FIFOs implemented and the reported f_{MAX} of the TDM clock after compilation. Table 2 shows approximate values for performance of various configurations given best-case (275 MHz) TDM_clk performance, and a conservative estimate of achievable TDM_clk performance (200 MHz).

Table 2. Estimated Performance of FIFO Ports

Number of FIFOs *(1)*	f_{MAX} of Each FIFO Port Assuming Bandwidth is Shared Evenly (MHz)	
	TDM_clk at 275 MHz	TDM_clk at 200 MHz *(2)*
2	91.67	66.6
3	91.67	66.6
4	68.75	50.00
5	55.00	40.00
6	45.83	33.3
7	39.28	28.6
8	34.37	25.00
9	30.56	22.20
10	27.50	20.00

Notes to Table 2:

(1) Configurations with more than 10 FIFOs are not recommended for most designs due to performance limitations.

(2) TDM_clk frequencies above 200 MHz are achievable. Refer to compilation results for achievable f_{MAX} for a given configuration.

Specifying a FIFO Partitioner Function Using the FIFO Partitioner MegaWizard

You specify the desired FIFO Partitioner configuration using the MegaWizard® Plug-In Manager. The wizard prompts you for the following information:

- How many FIFOs to implement
- The individual configuration options for each FIFO

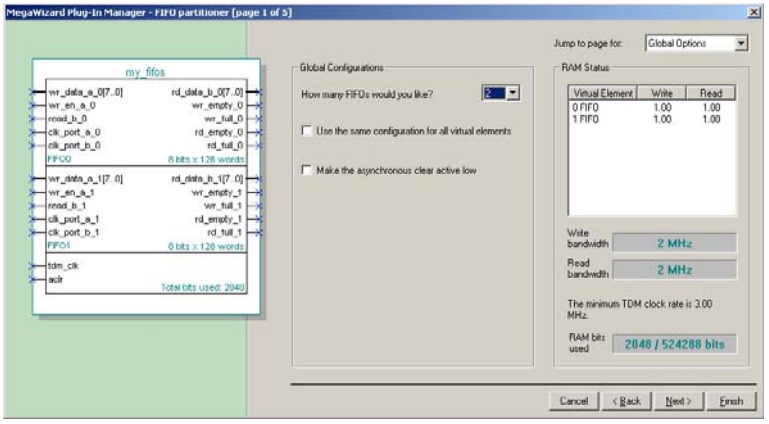
The wizard generates HDL files implementing the specified FIFO Partitioner configuration, and a information file with information about the FIFO Partitioner function. The information file contains frequency requirements for the clocks associated with the function.

Specifying Global Configuration Options

Specify the number of FIFOs you require in the MegaWizard. You can implement from 2 to 256 FIFOs using the FIFO Partitioner. Because independent FIFO ports must be time-domain-multiplexed together, the possible operating frequency of each FIFO port is inversely proportional to the number of FIFOs specified. See the restrictions on `TDM_clk` in the performance section of this document for more information.

Specify whether all FIFOs use the same configuration. If you select this option, you input width, depth, frequency, and mode information only once, applying it to all FIFOs. Specify the polarity of the ACLR input signal (Figure 3).

Figure 3. Global Configuration Options in the FIFO Partitioner MegaWizard



You must assert the asynchronous clear for 10 ns or longer before operating the FIFO Partitioner function.

Specifying FIFO Configurations

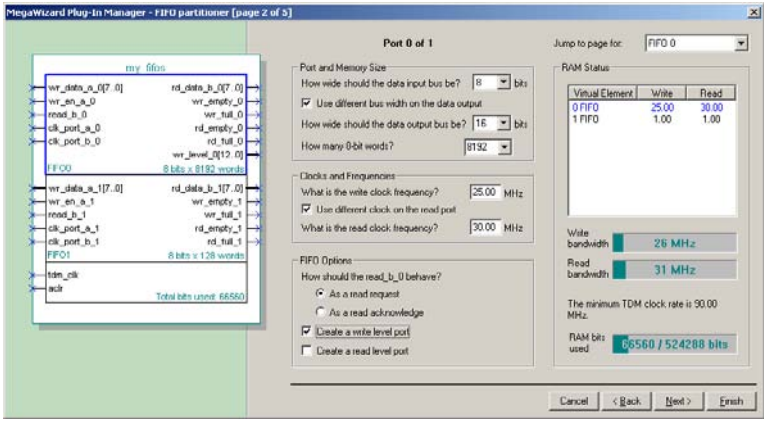
Specify the width and depth of each FIFO. Read and write ports of FIFOs can have different widths. Mixed-widths are supported for FIFOs with one port up to 72 bits wide. Specify the depth of each FIFO in terms of the number of bytes of the write port width. The total number of bits used by a FIFO Partitioned function is displayed in the MegaWizard. You should not create FIFO Partitioner functions which consume more than 1 M-RAM block.

Specify the clock frequencies of the read and write clock ports of each FIFO. For single-clock FIFOs you need only specify the frequency once, but the design still has separate clock inputs for the read and write ports. You should connect them accordingly.

Specify whether the `read_b_n` signal should behave as a read request or a read acknowledge input for each FIFO. This determines whether data is made available on the `rd_data_n` port before or after `read_b_n` is asserted. See the section of this document on FIFO functionality for details of `read_b_n` operation.

Specify whether to include read-level and write-level signals for each FIFO. These signals are optional and indicated to the user how many words in the FIFO are used. Read-level and write-level signals specify the current level in words in terms of the respective bus-width. Therefore, mixed-width FIFOs show different values on the read-level and write-level output signals. Figure 4 shows the FIFO Partitioner MegaWizard page for configuring FIFOs.

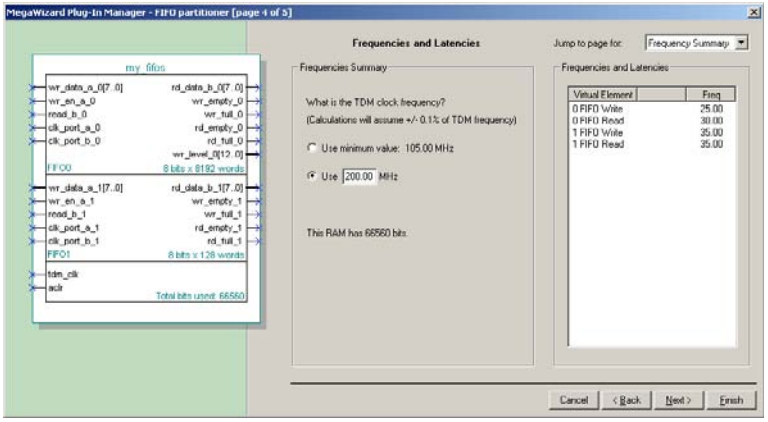
Figure 4. Configuration Options For FIFOs in the FIFO Partitioner MegaWizard



Frequency Information

After all FIFO configuration information has been specified, the FIFO Partitioner MegaWizard displays a summary of the frequency information for all FIFOs. The minimum TDM clock frequency is displayed and selected by default. You can enter a frequency higher than the minimum (Figure 5).

Figure 5. Frequency Information in the FIFO Partitioner MegaWizard



To minimize the effects of internal pipelining on the full and empty flags, use the highest achievable frequency for the TDM clock. In order to determine the highest achievable TDM clock performance, run a preliminary compilation and make sure that the reported f_{MAX} for `TDM_clk` is greater than the frequency you have chosen. It is important that you specify a value for `TDM_clk` that is accurate to within +/- 0.1% in order to ensure that the FIFO Partitioner function can operate correctly and realize the reported latency. Refer to "Latency Effects on Full and Empty Flags of FIFOs" on page 14 for a description of how internal pipelining affects the full and empty flags.

FIFO Functionality

This section describes the functionality of FIFO's. For more information, refer to Table 3, FIFO Partitioner Port listing.

Initializing the FIFO

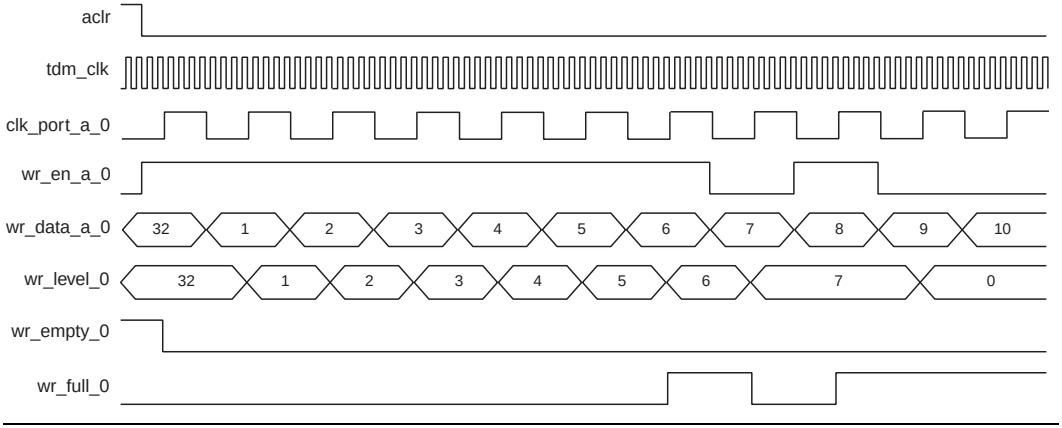
`ACLR` must be asserted for a minimum of 10 ns in order to initialize the FIFO control logic before any read or write operations may be performed. The port clocks and the `TDM_clk` must be oscillating at the specified frequencies in order for the FIFO Partitioner function to operate correctly. To check what frequency is specified, look at the <*functionname*>**_info.txt** file output by the MegaWizard Plug-In Manager. In order to modify these frequencies, you must rerun the MegaWizard Plug-In Manager.

FIFO Write-port Functionality

Each FIFO write-port includes an independent set of control and data signals. The `clk_port_a_n` input is used to clock the write operations for FIFO *n*. The `wr_full_n` must not be asserted during a write operation. If `wr_full_n` is asserted, wait for it to deassert before performing a write operation. To write to the FIFO, put the data to be written on `write_data_a_n`, and assert `wr_en_a_n`, for one `clk_port_a_n` cycle. Internally, the data is written to the next location of the FIFO and a write address pointer is incremented. The `wr_full_n`, `wr_empty_n` flags are updated immediately after a write cycle. The `wr_level_n` is updated on the following clock.

Figure 6 shows the waveforms intitializing and writing to a FIFO. The FIFO in this example has a depth of eight. Therefore, after eight words have been written to the FIFO, `wr_full_n` goes high. Notice that `wr_full_n` also goes high for one clock cycle before the FIFO is full. If the `wr_full_n` flag is high while the `wr_level_n` flag does not indicate the FIFO is full, it means the FIFO is busy and cannot accept write operations. Wait for the `wr_full_n` flag to go low before attempting to write another word to the FIFO.

Figure 6. Initializing and Writing to a FIFO



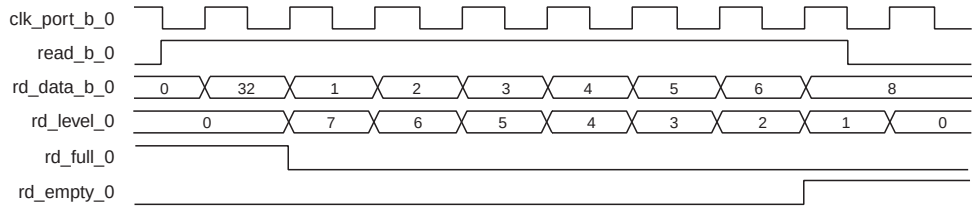
FIFO Read-port Functionality

Each FIFO read-port includes an independent set of control and data signals. The `clk_port_b_n` input is used to clock the read operations for FIFO *n*. In the MegaWizard, the user specifies whether the `read_b_n` signal should act as a read request or a read enable signal.

Read operations with `read_b_n` operating as a read request

To read from the FIFO, wait for `rd_empty` to be de-asserted indicating that data is ready to be read and assert `read_b_n`, for one `clk_port_a_n` cycle. The contents of the next FIFO location is output on the `rd_data_b_n`, and `read_empty_n` are updated during the next `clk_port_b_n` clock cycle. The `read_level_n`, and `read_full_n` flags are updated one clock cycle later (Figure 7).

Figure 7. Using read_b_n as a Read Request



In Figure 7, the contents of the FIFO written to in Figure 6 are read out. Notice that data on `rd_data_b_n` is not valid until after `read_b_n` has been asserted.

Read operations with `read_b_n` operating as a read acknowledge

When `read_b_n` operates as a read acknowledge, the contents of the next location of the FIFO as soon as it becomes available. The `rd_empty_n` flag goes is low when data is available, indicating that data on the `rd_data_b_n` bus is valid. The user asserts `read_b_n` to acknowledge that the data has been read and advance the read address pointer to the next FIFO location. After asserting `read_b_n` operating as a read acknowledge, the contents of the next FIFO location is output on the `rd_data_b_n`, and the `read_level_n`, `read_empty_n`, and `read_full_n` are updated during the next `clk_port_b_n` clock cycle (Figure 8).

Figure 8. Using read_b_n as a Read Acknowledge

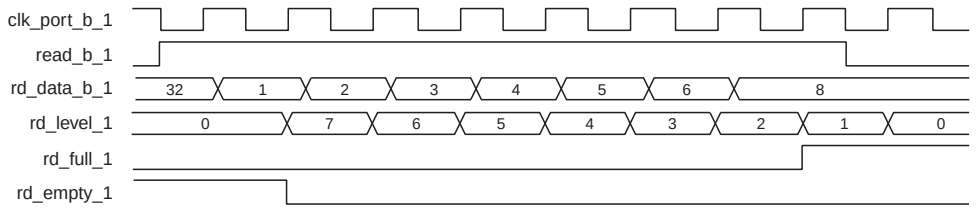


Figure 8 shows the identical situation as Figure 7, except for a FIFO with `read_b_n` operating as a read acknowledge. Notice that in this mode data is available on `rd_data_b_n` before `read_b_n` is asserted.

Latency Effects on Full and Empty Flags of FIFOs

The flags `wr_full_n` and `rd_empty_n` signal to the user when writing to and reading from the FIFO is restricted. In addition to becoming asserted when the FIFO is full or empty, these flags may become asserted for one or two clock cycles while the sequencer circuitry is busy servicing previous memory accesses. The user should never attempt to write to a FIFO whose `wr_full_n` flag is asserted, or read from a FIFO whose `rd_empty_n` flag is asserted. Doing so results in unspecified behavior.

An internal buffer is used to make FIFO data available immediately to the `rd_data_b_n`. Because this internal buffer is smaller than the FIFO, many successive read operations may cause `rd_empty_n` to be asserted for one or two port clock cycles when the FIFO is not empty. When the `rd_empty_n` flag is asserted, the user is not allowed to assert the `read_b_n` signal. To minimize the occurrences of `rd_empty_n` being asserted when the FIFO is not empty, use the maximum possible `TDM_clk` frequency.

Similarly, when performing many successive write operations to the FIFO, the `wr_full_n` flag may be asserted for one or two clock cycles before the FIFO is full. Write operations to the FIFO while `wr_full_n` is asserted are prohibited. To minimize the occurrences of `wr_full_n` being asserted when the FIFO is not empty, use the maximum possible `TDM_clk` frequency.

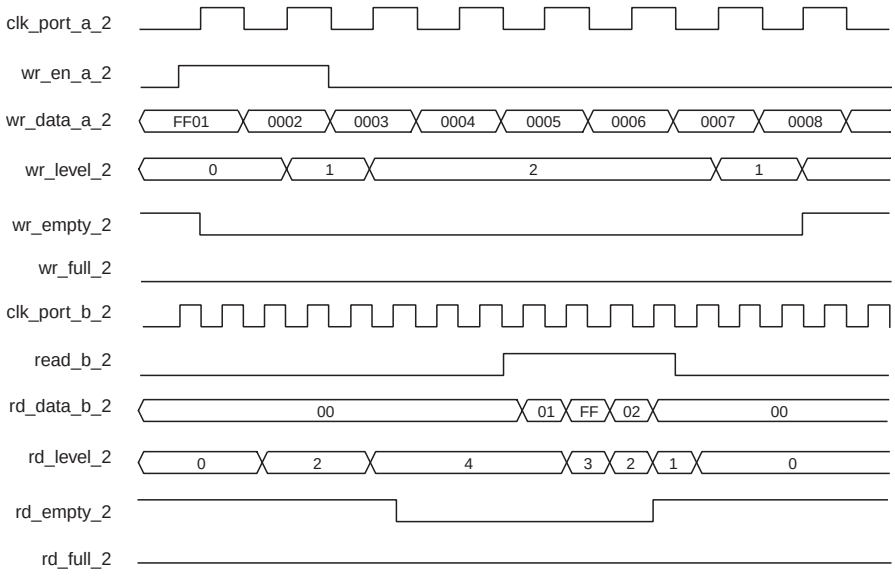
Mixed-width FIFOs

The FIFO Partitioner supports mixed-width FIFO functions. This section describes the behavior of mixed-width FIFOs. The following examples use FIFOs with `read_b_n` operating as a read request signal.

FIFOs with a wide write port and a narrow read port

Figure 9 illustrates write and read operations to a FIFO with a 16-bit-wide write-side bus, and a 8-bit-wide read-side bus. In this example the read port is operating at twice the frequency of the write port. Two 16-bit words are written to the FIFO. Notice that the `wr_level_2` flag is incremented to 2, and the `rd_level_2` flag is incremented to 4. Four 8-bit word read operations empty the FIFO. Notice that the least-significant 8 bits from the 16-bit word written are read first, followed by the most-significant 8 bits.

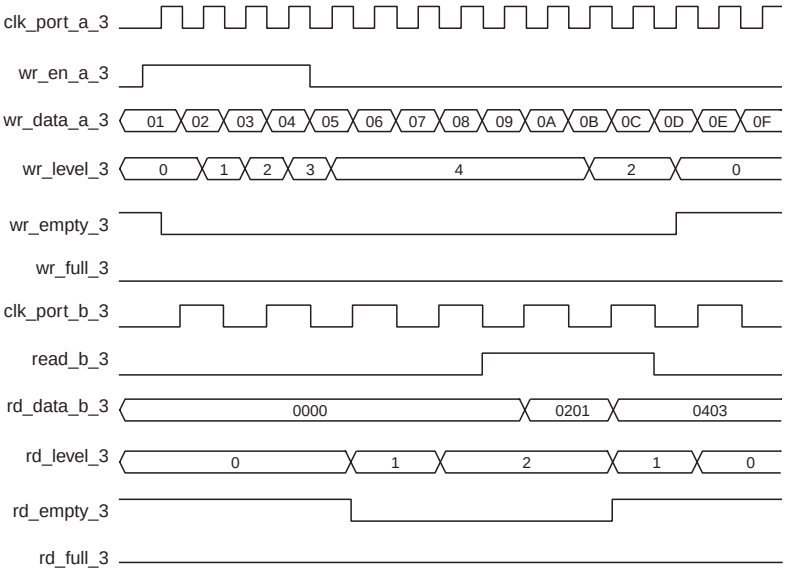
Figure 9. Writing 16-bit words and reading 8-bit words



FIFOs with a narrow write port and a wide read port

Figure 10 illustrates write and read operations to a FIFO with a 8-bit-wide write-side bus, and a 16-bit-wide read-side bus. In this example the read port is operating at half the frequency of the write port. Four 8-bit words are written to the FIFO. Notice that the `wr_level_3` flag is incremented to 4, and the `rd_level_3` flag is incremented to 2. Two 16-bit word read operations empty the FIFO. Notice that the first 8-bits written to the FIFO are the least-significant 8 bits in the 16-bit read operation. Also note that `rd_empty_`*`n`* does not deassert, indicating that data is available, until enough words have been written on the narrow write port, to fill an entire word on the wide read-side port.

Figure 10. Writing 8-bit words and reading 16-bit words



Port Listing for FIFO Partitioner Functions

Table 3 summarizes the port listing for FIFO Partitioner Functions.

Table 3. FIFO Partitioner Port Listing (Part 1 of 3)

Port Name	Required	Direction	Direction	Comments
`tdm_clk`	Yes	In	Time-domain-multiplexing clock port.	The `tdm_clk` requires a minimum frequency of 3 times the fastest FIFO port clock frequency or *n* times the fastest FIFO port clock frequency for *n* FIFOs. For optimum performance, `tdm_clk` should be run at maximum possible frequency as specified by hardware constraints and static timing analysis
`aclr`	Yes	In	Asynchronous global reset	Must be asserted for a minimum of 10 ns to initialize the FIFO Partitioner function. Polarity is user-defined in the MegaWizard.

Table 3. FIFO Partitioner Port Listing (Part 2 of 3)

Port Name	Required	Direction	Direction	Comments
clk_port_a_*n*	Yes	In	Clock for port *a* of FIFO *n*	Write side FIFO port clock. One per write port.
clk_port_b_*n*	Yes	In	Clock for port *b* of FIFO *n*	Read side FIFO port clock. One per read port.
wr_data_a_*n*[]	Yes	In	Port *a* data bus for FIFO *n*	Write side data port. One per write port.
wr_en_a_*n*	Yes	In	Port *a* write enable signal for FIFO *n*	One per FIFO. The user is restricted from asserting the wr_en_a_*n* for a FIFO whose wr_full_*n* flag asserted.
rd_data_b_*n*[]	Yes	Out	Port *b* data bus for FIFO *n*	One per FIFO. Behavior depends on mode of read_b_*n* signal.
read_b_*n*	Yes	In	Read request/ acknowledge signal for FIFO *n*	One per FIFO. The user is restricted from asserting the rd_en_b_*n* for a FIFO whose rd_empty_*n* flag asserted. When read_b_*n* port is configured as a read acknowledge signal, data is made available before read_b_*n* signal is asserted, and asserting read_b_*n* signals to the FIFO to output the contents of the next location in the FIFO. When read_b_*n* is configured as a read request, data becomes available immediately after asserting read_b_*n*.
wr_full_*n*	Yes	Out	Port *a* full flag for FIFO *n*	The write side FIFO full flag. This flag is synchronous to the write port clock. The user is restricted from asserting wr_en_a_*n* while wr_full_*n* is asserted. This flag may be asserted for one or two clock cycles when the FIFO is not full. To determine when the FIFO is truly full, the user should examine this flag in combination with the wr_level_*n* flag.
wr_empty_*n*	Yes	Out	Port *a* empty flag for FIFO *n*	The write side FIFO empty flag. This flag is synchronous to the individual write port clocks.
wr_level_*n*[]	No	Out	Port *a* level flag for FIFO *n*	The write side FIFO level bus. Indicates how many words are currently in the FIFO. These status bits are synchronous to the write port clocks. For FIFOs with a depth equal to a power of two (i.e. 8, 16, 32, …), the wr_full_*n* flag is considered the most significant bit of wr_level_*n* when wr_level_*n* = 0. This status bus has one cycle of latency.

Table 3. FIFO Partitioner Port Listing (Part 3 of 3)

Port Name	Required	Direction	Direction	Comments
rd_full_*n*	Yes	Out	Port *b* full flag for FIFO *n*	The read side FIFO full bus. This flag is synchronous to the read port clocks. This flag has one cycle of latency.
rd_empty_*n*	Yes	Out	Port *b* empty flag for FIFO *n*	The read side FIFO empty flag. This flag is synchronous to the individual read port clocks. This port may assert for one or two read port clock cycles at times when the FIFO is not empty but the zero-latency output buffer does not have data available. The user is restricted from reading from a FIFO whenever its rd_empty_*n* flag is asserted. When a FIFO is configured with read_b_*n* as a read acknowledge, rd_empty_*n* also acts as an active-low data valid flag, that is, data on rd_data_*n* is not valid while rd_empty_*n* is asserted in read acknowledge mode.
rd_level_*n*	No	Out	Port *b* level flag for FIFO *n*	The read side FIFO level bus. Indicates how many words are currently in the FIFO. These status bits are synchronous to the individual read port clocks. This status bus has one cycle of latency.

Synthesizing FIFO Partitioner Functions

The HDL output file generated by the FIFO Partitioner MegaWizard can be synthesized using standard techniques. The HDL output file instantiates an instance of alt_csm_core. **alt_csm_core.vhd** and the other HDL files it depends on are located in the <*quartus directory*>\libraries\megafunctions directory.

Native Synthesis Using the Quartus II Software

Because the necessary files are located in the MegaFunctions directory, the files are automatically detected by the Quartus II software during synthesis. No special steps are required to synthesize the FIFO Partitioner function in the Quartus II software.

Using Third-Party EDA Tools for Synthesis

When using third-party EDA tools for synthesis, it is recommended that the FIFO Partitioner function be black-boxed. If you include only the MegaWizard-generated file along with your design, supported EDA synthesis tools automatically black-box the `alt_csm_core` instantiation the Quartus software synthesizes it during pre-place-and-route synthesis.

To synthesize the `alt_csm_core` function in a third-party EDA synthesis tool, the tool must support VHDL synthesis and your project must include:

- <*function_name*>**.vhd** or <*function_name*>**.v**
- The **ALTERA_MF_COMPONENTS.VHD** file located in the <*quartus_directory*>\libraries\vhdl\altera_mf directory
- The following library files located in the <*quartus directory*>\libraries\megafunctions directory:
  - **alt_csm_unidpram_wr.vhd**
  - **alt_csm_rom_wrapper.vhd**
  - **alt_csm_fifo.vhd**
  - **alt_csm_mux_rd.vhd**
  - **alt_csm_mux_wr.vhd**
  - **alt_csm_rom_wrapper.vhd**
  - **alt_csm_sequence.vhd**
  - **alt_csm_unidpram_com_bus_rd.vhd**
  - **alt_csm_unidpram_com_bus_wr.vhd**
  - **alt_csm_core.vhd**

The wizard-generated file must be analyzed by third-party synthesis tools before `alt_csm_core` because it defines a package which is used in `alt_csm_core`.

Static Timing Analysis for FIFO Partitioner Functions

By default, the Quartus II software automatically performs static timing analysis after placement and routing. You should specify an f_{max} timing requirement for the `TDM_clk` that is equal to or greater than the value specified in the MegaWizard output info file. After compilation, verify that the reported fmax for `TDM_clk` is greater than the intended operating frequency for `TDM_clk`. Optimize the design, or reconfigure your FIFO Partitioner function if necessary.

Logic internal to the MegaWizard-generated function must operate at the TDM clock frequency. Static timing analysis within the MegaWizard-generated function on the individual FIFO clock domains is not necessary. The FIFO Partitioner function operates correctly if the specified frequencies for all clocks are used and the TDM clock is operating within its reported fmax limit.

Simulating FIFO Partitioner Functions

Standard simulation techniques are available for FIFO Partitioner. It is necessary for third-party EDA simulation tools to synthesize the library files listed above, and altera_mf_components model located in the <*quartus_directory*>\eda\sim_lib directory, for third-party EDA synthesis in order to simulate FIFO Partitioner functions correctly. Two hex files output by the FIFO Partitioner MegaWizard, called <*function_name*>**_a.hex** and <*function_name*>**_b.hex** must also be included for successful third-party simulation.

To perform functional simulation in an third-party EDA tool without VHDL support, first synthesize the design in Quartus, and use the <*function_name*>**.vo** file output by the Quartus software in your EDA simulation tool.