

LogiCORE™ FIFO Generator v2.3

User Guide

UG175 January 11, 2006





Xilinx is disclosing this Document and Intellectual Property (hereinafter “the Design”) to you for use in the development of designs to operate on, or interface with Xilinx FPGAs. Except as stated herein, none of the Design may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx. Any unauthorized use of the Design may violate copyright laws, trademark laws, the laws of privacy and publicity, and communications regulations and statutes.

Xilinx does not assume any liability arising out of the application or use of the Design; nor does Xilinx convey any license under its patents, copyrights, or any rights of others. You are responsible for obtaining any rights you may require for your use or implementation of the Design. Xilinx reserves the right to make changes, at any time, to the Design as deemed desirable in the sole discretion of Xilinx. Xilinx assumes no obligation to correct any errors contained herein or to advise you of any correction if such be made. Xilinx will not assume any liability for the accuracy or correctness of any engineering or technical support or assistance provided to you in connection with the Design.

THE DESIGN IS PROVIDED “AS IS” WITH ALL FAULTS, AND THE ENTIRE RISK AS TO ITS FUNCTION AND IMPLEMENTATION IS WITH YOU. YOU ACKNOWLEDGE AND AGREE THAT YOU HAVE NOT RELIED ON ANY ORAL OR WRITTEN INFORMATION OR ADVICE, WHETHER GIVEN BY XILINX, OR ITS AGENTS OR EMPLOYEES. XILINX MAKES NO OTHER WARRANTIES, WHETHER EXPRESS, IMPLIED, OR STATUTORY, REGARDING THE DESIGN, INCLUDING ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE, AND NONINFRINGEMENT OF THIRD-PARTY RIGHTS.

IN NO EVENT WILL XILINX BE LIABLE FOR ANY CONSEQUENTIAL, INDIRECT, EXEMPLARY, SPECIAL, OR INCIDENTAL DAMAGES, INCLUDING ANY LOST DATA AND LOST PROFITS, ARISING FROM OR RELATING TO YOUR USE OF THE DESIGN, EVEN IF YOU HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. THE TOTAL CUMULATIVE LIABILITY OF XILINX IN CONNECTION WITH YOUR USE OF THE DESIGN, WHETHER IN CONTRACT OR TORT OR OTHERWISE, WILL IN NO EVENT EXCEED THE AMOUNT OF FEES PAID BY YOU TO XILINX HEREUNDER FOR USE OF THE DESIGN. YOU ACKNOWLEDGE THAT THE FEES, IF ANY, REFLECT THE ALLOCATION OF RISK SET FORTH IN THIS AGREEMENT AND THAT XILINX WOULD NOT MAKE AVAILABLE THE DESIGN TO YOU WITHOUT THESE LIMITATIONS OF LIABILITY.

The Design is not designed or intended for use in the development of on-line control equipment in hazardous environments requiring fail-safe controls, such as in the operation of nuclear facilities, aircraft navigation or communications systems, air traffic control, life support, or weapons systems (“High-Risk Applications”). Xilinx specifically disclaims any express or implied warranties of fitness for such High-Risk Applications. You represent that use of the Design in such High-Risk Applications is fully at your risk.

© 2006 Xilinx, Inc. All rights reserved. XILINX, the Xilinx logo, and other designated brands included herein are trademarks of Xilinx, Inc. All other trademarks are the property of their respective owners.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
04/28/05	1.1	Initial Xilinx release.
8/31/05	2.0	Updated guide for release v2.2, added SP3 to ISEv7.1i, incorporated edits from engineering specific for this release, including FWFT, and Built-in FIFO flags, etc.
1/11/06	3.0	Updated for v2.3 release, ISE v8.1i.

Table of Contents

Preface: About This Guide

Guide Contents	11
Additional Resources	12
Conventions	12
Typographical	12
Online Document	13

Chapter 1: Introduction

About the Core	15
Recommended Design Experience	15
Technical Support	15
Feedback	16
FIFO Generator	16
Document	16

Chapter 2: Installing the Core

System Requirements	17
Before You Begin	17
Installing the Core	17
CORE Generator IP Updates Installer	18
Manual Installation	18
Verifying the Installation	18

Chapter 3: Core Overview

System Overview	21
Clock Implementation Operation	21
Built-in FIFO Support in Virtex-4	21
First-Word Fall-Through	21
Memory Types	22
Non-Symmetric Aspect Ratio	22
Core Configuration and Implementation	22
Independent Clocks: Block RAM and Distributed RAM	23
Independent Clocks: Virtex-4 Built-in FIFO	23
Common Clock: Block RAM, Distributed RAM, Shift Register	23
Common Clock: Virtex-4 Built-in FIFO	23
FIFO Generator Features	24
Interface Signals: FIFOs With Independent Clocks	25
Interface Signals: FIFOs with Common Clock	28

Chapter 4: Generating the Core

CORE Generator Graphical User Interface	31
---	----

FIFO Implementation	32
Component Name	32
FIFO Implementation	32
Common Clock (CLK), Block RAM	32
Common Clock (CLK), Distributed RAM	33
Common Clock (CLK), Shift Register	33
Common Clock (CLK), Built-in FIFO	33
Independent Clocks (RD_CLK, WR_CLK), Block RAM	33
Independent Clocks (RD_CLK, WR_CLK), Distributed RAM	33
Independent Clocks (RD_CLK, WR_CLK), Built-in FIFO	33
Performance Options and Data Port Parameters	34
Performance Options	34
Standard FIFO	34
First-word Fall-through FIFO	34
Data Port Parameters	34
Input Data Width	34
Input Depth	35
Output Data Width	35
Output Depth	35
Primitive Depth	35
Optional Flags, Handshaking, and Initialization	35
Optional Flags	36
Almost Full Flag	36
Almost Empty Flag	36
Write Port Handshaking	36
Write Acknowledge	36
Overflow (Write Error)	36
Read Port Handshaking	36
Valid (Read Acknowledge)	36
Underflow (Read Error)	36
Initialization	36
Reset Pin	36
Programmable Flags	37
Programmable Flags	37
Programmable Full Type	37
Programmable Empty Type	38
Data Count and Reset	38
Data Count and Reset Options	39
Data Count	39
Resets	39
Summary	39

Chapter 5: Designing with the Core

General Design Guidelines	41
Know the Degree of Difficulty	41
Understand Signal Pipelining	41
Initializing the FIFO Generator	41
Independent Clocks: Block RAM and Distributed RAM	42
Write Operation	43
ALMOST_FULL and FULL Flags	43
Example Operation	43

Read Operation	44
ALMOST_EMPTY and EMPTY Flags	44
Modes of Read Operation	44
Handshaking Flags	46
Valid	46
Underflow	46
Write Acknowledge	46
Overflow	46
Example Operation	46
Programmable Flags	47
Programmable Full	47
Programmable Empty	49
Data Counts	51
Read Data Count	51
Write Data Count	51
First-Word Fall-Through Data Count	52
Example Operation	52
Non-symmetric Aspect Ratios	53
Reset Behavior	55
Independent Clocks: Built-in FIFO	56
Write Operation	57
FULL Flag	58
Examples of Operation	58
Read Operation	59
EMPTY Flag	59
Example Operation	59
Handshaking Flags	60
Valid	60
Underflow	60
Write Acknowledge	60
Overflow	60
Example Operation	60
Programmable Flags	61
Programmable Full	61
Programmable Empty	63
Reset Behavior	65
Synchronization Considerations	66
Common Clock FIFO: Block RAM, Distributed RAM, Shift Register	67
Common Clock FIFO: Block RAM and Distributed RAM	67
Common Clock FIFO: Shift Registers	68
Write and Read Operation	69
Write Operation	69
Read Operation	69
ALMOST_FULL and FULL Flags	69
ALMOST_EMPTY and EMPTY Flags	69
Example Operation	69
Handshaking Flags	70
Valid	70
Underflow	70
Write Acknowledge	70
Overflow	70
Example Operation	70
Programmable Flags	71

Programmable Full	71
Programmable Empty	73
Data Count	75
Reset Behavior	76
Common Clock: Built-in FIFO	77

Chapter 6: Special Design Considerations

Resetting the FIFO	79
Continuous Clocks	79
Pessimistic Full and Empty	79
Programmable Full and Empty	80
Write Data Count and Read Data Count	80
Setup and Hold Time Violations	80

Chapter 7: Simulating Your Design

Simulation Models	81
-------------------------	----

Appendix A: Performance Information

Resource Utilization and Performance	83
--	----

Appendix B: Core Parameters

FIFO Parameters	87
-----------------------	----

Schedule of Figures

Schedule of Figures	7
----------------------------------	---

Preface: About This Guide

Chapter 1: Introduction

Chapter 2: Installing the Core

<i>Figure 2-1: CORE Generator Window</i>	19
--	----

Chapter 3: Core Overview

<i>Figure 3-1: FIFO with Independent Clocks: Interface Signals</i>	25
--	----

Chapter 4: Generating the Core

<i>Figure 4-1: Main FIFO Generator Screen</i>	32
<i>Figure 4-2: Performance Options and Data Port Parameters Screen</i>	34
<i>Figure 4-3: Optional Flags, Handshaking and Initialization Options Screen</i>	35
<i>Figure 4-4: Programmable Flags Screen</i>	37
<i>Figure 4-5: Data Count and Reset Screen</i>	38
<i>Figure 4-6: Summary Screen</i>	40

Chapter 5: Designing with the Core

<i>Figure 5-1: Functional Implementation of a FIFO with Independent Clock Domains</i> .	42
<i>Figure 5-2: Write Operation for a FIFO with Independent Clocks</i>	44
<i>Figure 5-3: Standard Read Operation for a FIFO with Independent Clocks</i>	45
<i>Figure 5-4: FWFT Read Operation for a FIFO with Independent Clocks</i>	45
<i>Figure 5-5: Handshaking Signals for a FIFO with Independent Clocks</i>	47
<i>Figure 5-6: Programmable Full Single Threshold: Threshold Set to 7</i>	48
<i>Figure 5-7: Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7</i>	49
<i>Figure 5-8: Programmable Empty with Single Threshold: Threshold Set to 4</i>	50
<i>Figure 5-9: Programmable Empty with Assert and Negate Thresholds: Assert Set to 7 and Negate Set to 10</i>	51
<i>Figure 5-10: Write and Read Data Counts for FIFO with Independent Clocks</i>	53
<i>Figure 5-11: 1:4 Aspect Ratio: Data Ordering</i>	54
<i>Figure 5-12: 1:4 Aspect Ratio: Status Flag Behavior</i>	54
<i>Figure 5-13: 4:1 Aspect Ratio: Data Ordering</i>	55
<i>Figure 5-14: 4:1 Aspect Ratio: Status Flag Behavior</i>	55
<i>Figure 5-15: Reset Behavior for FIFO with Independent Clocks</i>	56

<i>Figure 5-16: Functional Implementation of Built-in FIFO</i>	57
<i>Figure 5-17: Write Operation for a FIFO with Independent Clocks</i>	58
<i>Figure 5-18: Write Operation for a FIFO Using Built-in Flags</i>	59
<i>Figure 5-19: Read Operation for a FIFO with Independent Clocks</i>	60
<i>Figure 5-20: Handshaking Signals for a FIFO with Independent Clocks</i>	61
<i>Figure 5-21: Programmable Full Single Threshold: Threshold Set to 7</i>	62
<i>Figure 5-22: Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7</i>	63
<i>Figure 5-23: Programmable Empty with Single Threshold: Threshold Set to 4</i>	64
<i>Figure 5-24: Programmable Empty with Assert and Negate Thresholds: Assert Set to 4 and Negate Set to 6</i>	65
<i>Figure 5-25: Reset Behavior for Built-In FIFO Implementations with Independent Clocks</i>	65
<i>Figure 5-26: FIFO with Independent Clocks: Write and Read Clock Domains</i>	67
<i>Figure 5-27: Functional Implementation of a Common Clock FIFO using Block RAM or Distributed RAM</i>	68
<i>Figure 5-28: Functional Implementation of a Common Clock FIFO using Shift Registers</i>	68
<i>Figure 5-29: Write and Read Operation for a FIFO with Common Clocks</i>	70
<i>Figure 5-30: Handshaking Signals for a FIFO with Common Clocks</i>	71
<i>Figure 5-31: Programmable Full Single Threshold: Threshold Set to 7</i>	72
<i>Figure 5-32: Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7</i>	73
<i>Figure 5-33: Programmable Empty with Single Threshold: Threshold Set to 4</i>	74
<i>Figure 5-34: Programmable Empty with Assert and Negate Thresholds: Assert Set to 7 and Negate Set to 10</i>	75
<i>Figure 5-35: Write and Read Data Counts for FIFO with Common Clocks</i>	76
<i>Figure 5-36: FIFO Reset Behavior for FIFO with Common Clocks</i>	76

Chapter 6: Special Design Considerations

Chapter 7: Simulating Your Design

Appendix A: Performance Information

Appendix B: Core Parameters

Schedule of Tables

Schedule of Tables	9
---------------------------------	---

Preface: About This Guide

Chapter 1: Introduction

Chapter 2: Installing the Core

Chapter 3: Core Overview

<i>Table 3-1: Memory Configuration Benefits</i>	22
<i>Table 3-2: FIFO Configurations</i>	22
<i>Table 3-3: FIFO Configurations Summary</i>	24
<i>Table 3-4: Reset Signal for FIFOs with Independent Clocks</i>	25
<i>Table 3-5: Write Interface Signals for FIFOs with Independent Clocks</i>	25
<i>Table 3-6: Read Interface Signals for FIFOs with Independent Clocks</i>	26
<i>Table 3-7: Interface Signals for FIFOs with a Common Clock</i>	28

Chapter 4: Generating the Core

Chapter 5: Designing with the Core

<i>Table 5-1: Interface Signals and Corresponding Clock Domains</i>	43
<i>Table 5-2: Independent Clock FIFO Reset Values</i>	56
<i>Table 5-3: Interface Signals and Corresponding Clock Domains</i>	57
<i>Table 5-4: Built-In FIFO Reset Values</i>	66
<i>Table 5-5: Common Clock FIFO Reset Values</i>	77

Chapter 6: Special Design Considerations

Chapter 7: Simulating Your Design

Appendix A: Performance Information

<i>Table A-1: Benchmarks: FIFO Configured without Optional Features</i>	83
<i>Table A-2: FIFO Benchmarks: Configured with Multiple Programmable Thresholds</i> .	84
<i>Table A-3: Benchmarks: FIFO Configured with Virtex-4 Built-in FIFO Resources</i>	85

Appendix B: Core Parameters

<i>Table B-1: FIFO Parameter Table</i>	87
--	----

About This Guide

The *LogicCORE™ FIFO Generator User Guide v2.3* describes the function and operation of the FIFO Generator, and contains information about designing, customizing, and implementing the core.

Guide Contents

The following chapters are included:

- [“Preface, About this Guide”](#) describes how the user guide is organized, the conventions used in the guide, and provides information about additional resources.
- [Chapter 1, “Introduction,”](#) describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- [Chapter 2, “Installing the Core,”](#) provides information about installing the core.
- [Chapter 3, “Core Overview,”](#) describes the core configuration options and their interfaces.
- [Chapter 4, “Generating the Core,”](#) describes how to generate the core using the Xilinx CORE Generator GUI.
- [Chapter 5, “Designing with the Core,”](#) discusses how to use the core in a user application.
- [Chapter 6, “Special Design Considerations,”](#) discusses specific design features that must be considered when using with the core.
- [Chapter 7, “Simulating Your Design,”](#) provides instructions for simulating the design with either behavioral or structural simulation models.
- [Appendix A, “Performance Information,”](#) provides a summary of the core’s performance data.
- [Appendix B, “Core Parameters,”](#) provides a comprehensive list of the parameters set by in the CORE Generator GUI for the FIFO Generator.

Additional Resources

For additional information, go to <http://www.xilinx.com/support>. The following table lists some of the resources you can access from this website or by using the provided URLs.

Resource	Description/URL
Tutorials	Tutorials covering Xilinx design flows, from design entry to verification and debugging http://www.xilinx.com/support/techsup/tutorials/index.htm
Answer Browser	Database of Xilinx solution records http://www.xilinx.com/xlnx/xil_ans_browser.jsp
Data Sheets	Device-specific information on Xilinx device characteristics, including readback, boundary scan, configuration, length count, and debugging http://www.xilinx.com/xlnx/xweb/xil_publications_index.jsp
Problem Solvers	Interactive tools that allow you to troubleshoot your design issues http://www.xilinx.com/support/troubleshoot/psolvers.htm
Tech Tips	Latest news, design tips, and patch information for the Xilinx design environment http://www.xilinx.com/xlnx/xil_tt_home.jsp

Conventions

This document uses the following conventions. An example illustrates each convention.

Typographical

The following typographical conventions are used in this document:

Convention	Meaning or Use	Example
Courier font	Messages, prompts, and program files that the system displays	speed grade: - 100
Courier bold	Literal commands you enter in a syntactical statement	ngdbuild design_name
<i>Italic font</i>	Variables in a syntax statement for which you must supply values	See the <i>Development System Reference Guide</i> for more information.
	References to other manuals	See the <i>User Guide</i> for details.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are <i>not</i> connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported

Convention	Meaning or Use	Example
Square brackets []	An optional entry or parameter. However, in bus specifications, such as bus[7:0] , they are required.	ngdbuild [option_name] design_name
Braces { }	A list of items from which you must choose one or more	lowpwr = {on off}
Vertical bar	Separates items in a list of choices	lowpwr = {on off}
Vertical ellipsis . . .	Repetitive material that has been omitted	IOB #1: Name = QOUT' IOB #2: Name = CLKIN' . . .
Horizontal ellipsis ...	Omitted repetitive material	allow block block_name loc1 loc2 ... locn;
Notations	The prefix '0x' or the suffix 'h' indicate hexadecimal notation	A read of address 0x00112975 returned 45524943h.
	An '_n' means the signal is active low	usr_teof_n is active low.

Online Document

The following linking conventions are used in this document:

Convention	Meaning or Use	Example
Blue text	Cross-reference link to a location in the current document	See the section " Additional Resources " for details. Refer to " Title Formats " in Chapter 1 for details.
Blue, underlined text	Hyperlink to a website (URL)	Go to http://www.xilinx.com for the latest speed files.

Introduction

The Xilinx LogiCORE FIFO Generator, a fully verified first-in first-out memory queue for use in any application requiring in-order storage and retrieval, enables high-performance and area-optimized designs. This core can be customized using the Xilinx CORE Generator™ system as a complete solution with control logic already implemented, including management of the read and write pointers and the generation of status flags.

This chapter introduces the FIFO Generator and provides related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.

About the Core

The FIFO Generator is a Xilinx CORE Generator™ IP core, included in the latest IP Update on the Xilinx IP Center. For detailed information about the core, see http://www.xilinx.com/xlnx/xebiz/designResources/ip_product_details.jsp?key=FIFO_Generator.

For information about system requirements and installation, see [Chapter 2, “Installing the Core.”](#)

Recommended Design Experience

The FIFO Generator is a fully verified solution, and can be used by all levels of design engineers. When implementing a FIFO with independent write and read clocks, special care must be taken to ensure the FIFO Generator is correctly used. For details, see [“Synchronization Considerations,”](#) page 66.

Technical Support

For technical support, visit <http://support.xilinx.com/>. Questions are routed to a team of engineers with FIFO Generator expertise.

Xilinx will provide technical support for use of this product as described in the *LogiCORE FIFO Generator User Guide*. Xilinx cannot guarantee timing, functionality, or support of this product for designs that do not follow these guidelines.

Feedback

Xilinx welcomes comments and suggestions about the FIFO Generator and the documentation supplied with the core.

FIFO Generator

For comments or suggestions about the FIFO Generator, please submit a WebCase from <http://support.xilinx.com/>. Be sure to include the following information:

- Product name
- Core version number
- Explanation of your comments

Document

For comments or suggestions about this document, please submit a WebCase from <http://support.xilinx.com/>. Be sure to include the following information:

- Document title
- Document number
- Page number(s) to which your comments refer
- Explanation of your comments

Installing the Core

This chapter provides instructions for installing the FIFO Generator. The FIFO Generator is provided under the terms of the [Xilinx LogiCORE Site License Agreement](#), which conforms to the terms of the [SignOnce](#) IP License standard defined by the Common License Consortium.

System Requirements

Windows

Windows

- Windows® 2000 Professional with Service Pack 2-4
- Windows XP Professional with Service Pack 1

Solaris/Linux

- Sun Solaris® 8/9
- Red Hat® Enterprise Linux 3.0 (32-bit and 64-bit)

Software

- Xilinx ISE™ 8.1i

Before You Begin

Before installing the core, you must have a Xilinx.com account and the ISE 8.1i software installed on your system. If you have already completed these steps, go to [“Installing the Core.”](#)

1. Click Login at the top of the [Xilinx home page](#); then follow the onscreen instructions to create a support account.
2. Install the ISE 8.1i software and the applicable service pack software. ISE service packs can be downloaded from www.xilinx.com/support/download.htm.

Installing the Core

You can install the core in two ways—using the CORE Generator IP Updates Installer, which lets you select from a list of updates, or by performing a manual installation after downloading the core from the web.

CORE Generator IP Updates Installer

Note: To use this installation method behind a firewall, you must know your proxy settings. Contact your administrator to determine the proxy host address and port number before you begin if necessary.

1. Start the CORE Generator; then open an existing project or create a new one.
2. From the main CORE Generator window, choose Tools > Updates Installer to start the Updates Installer. If you are behind a firewall, you will be prompted to enter your proxy host and port settings.
3. If prompted, enter your proxy settings; then click Set. The IP Updates installer appears.
4. Click the checkbox next to 8.1i_IP_Update1 to select it; then click Install Selected. Informational messages may appear indicating that additional installations are required.
5. Click OK to accept any messages and continue. The User Login dialog box appears.
6. Enter your login name and password; then click OK. The Updates Installer Generator downloads and installs the selected products, and then exits.
7. To confirm the installation, check the following file:
C:\Xilinx\coregen\install\install_history.

Note that this step assumes your Xilinx software is installed in C:\Xilinx.

Manual Installation

1. Close the CORE Generator application if it is running.
2. Download the IP Update ZIP file from the following location and save it to a temporary directory:
http://www.xilinx.com/xlnx/xil_sw_updates_home.jsp?update=ip&software=8.1i
3. Unpack the ZIP files using either WinZip (Windows) or Unzip (UNIX).
4. Extract the ZIP file (ise_81i_ip_update1.zip) archive to the root directory of your Xilinx software installation. (Allow the extractor utility you use to overwrite all existing files and maintain the directory structure defined in the archive.)
5. If you do not have a zip utility, do one of the following:
 - **Windows.** From a command window, type the following:
%XILINX%\bin\nt\unzip -d %XILINX% ise_81i_ip_update1.zip
 - **Linux.** From a UNIX shell, type the following:
\$XILINX/bin/lin/unzip -d \$XILINX ise_81i_ip_update1.zip
 - **Solaris.** From a UNIX shell, type the following:
\$XILINX/bin/sol/unzip -d \$XILINX ise_81i_ip_update1.zip
6. To verify the root directory of your Xilinx installation, do one of the following:
 - **Windows.** Type echo %XILINX% from a DOS prompt.
 - **UNIX.** If you have already installed the Xilinx ISE software, the Xilinx variable defined by your set-up script identifies the location of the Xilinx installation directory. After sourcing the Xilinx set-up script, type echo \$XILINX to determine the location of the Xilinx installation.

Verifying the Installation

1. Start the CORE Generator.

- After creating a new project or opening an existing one, the IP core functional categories appear at the left side of the window.

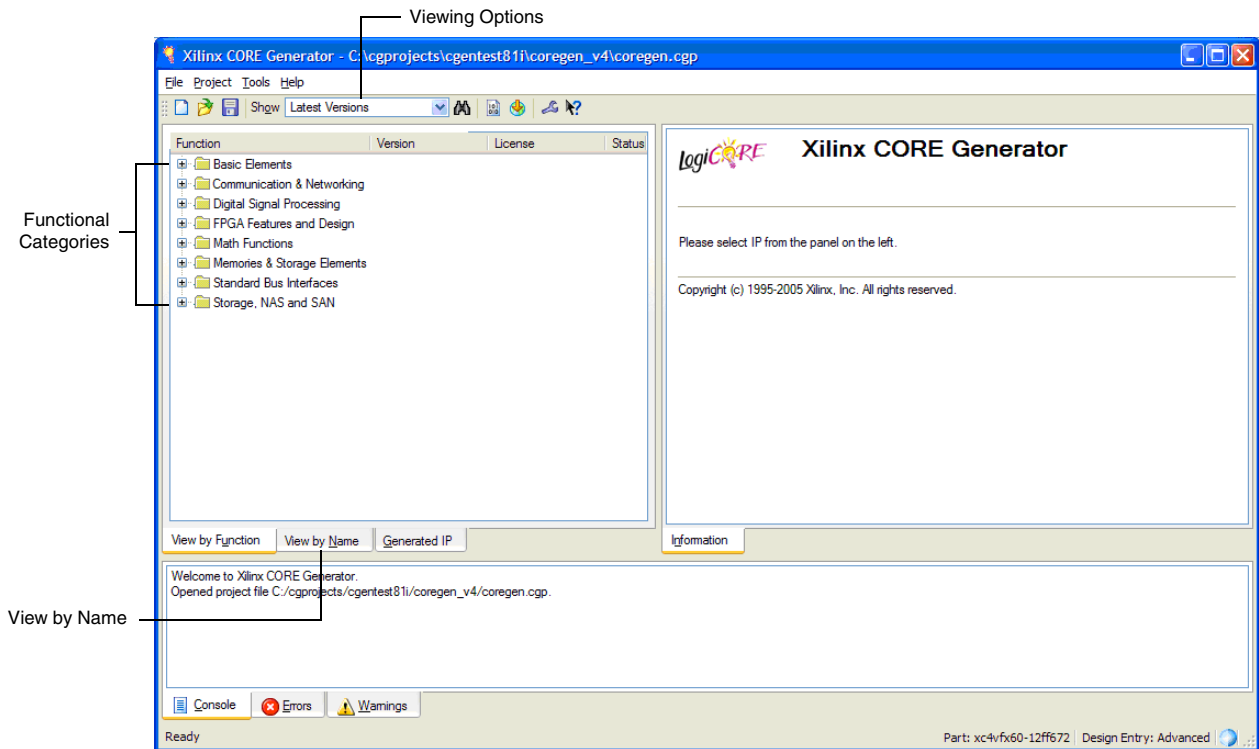


Figure 2-1: CORE Generator Window

- Click to expand or collapse the view of individual functional categories, or click the View by Name tab at the bottom of the list to see an alphabetical list of all cores in all categories.
- To view specific versions of the cores, choose an option from the Show drop-down list at the top of the window:
 - Latest Versions.** Display the latest versions of all cores.
 - All Versions.** Display all versions of cores, including new cores and new versions of cores.
 - All Versions including Obsolete.** Display all cores, including those scheduled to become obsolete.
- Determine if the installation was successful by verifying that the new core or cores appear in the CORE Generator GUI.

For additional assistance installing the IP Update, contact www.xilinx.com/support.

Core Overview

This chapter provides an overview of the FIFO Generator configuration options and their interfaces.

System Overview

Clock Implementation Operation

The FIFO Generator enables FIFOs to be configured with either independent or common clock domains for write and read operations. The independent clock configuration of the FIFO Generator enables the user to implement unique clock domains on the write and read ports. The FIFO Generator handles the synchronization between clock domains, placing no requirements on phase and frequency. When data buffering in a single clock domain is required, the FIFO Generator can be used to generate a core optimized for that clock.

Built-in FIFO Support in Virtex-4

The FIFO Generator supports the VirtexTM-4 built-in FIFO modules, enabling large FIFOs to be created by cascading the built-in FIFOs in both width and depth. The core expands the capabilities of the built-in FIFOs by utilizing the FPGA fabric to create optional status flags not implemented in the built-in FIFO macro.

First-Word Fall-Through

The first-word fall-through (FWFT) feature provides the ability to look ahead to the next word available from the FIFO without having to issue a read operation. The FIFO accomplishes this by using output registers which are automatically loaded with data, when data appears in the FIFO. This causes the first word written to the FIFO to automatically appear on the out data bus (DOUT). Subsequent user read operations cause the output data to update with the next word, as long as data is available in the FIFO. The use of registers on the FIFO DOUT bus improves clock-to-output timing, and the FWFT functionality provides low-latency access to data. This is ideal for applications that require throttling, based on the contents of the data that are read.

FWFT support is included in independent clock FIFOs created with either block RAM or distributed RAM. The use of this feature impacts the behavior of many other features, such as:

- Read operations (see [“First-Word Fall-Through FIFO Read Operation,”](#) page 45).
- Programmable empty (see [“Programmable Empty for First-Word Fall-Through,”](#) page 51).
- Data counts (see [“First-Word Fall-Through Data Count,”](#) page 52).

Memory Types

The FIFO Generator implements FIFOs built from block RAM, distributed RAM, shift registers, or the Virtex-4 built-in FIFOs. The core combines memory primitives in an optimal configuration based on the selected width and depth of the FIFO. [Table 3-1](#) provides best-use recommendations for specific design requirements.

Table 3-1: Memory Configuration Benefits

	Independent Clocks	Common Clock	Small Buffering	Medium-Large Buffering	High Performance	Minimal Resources
Built-in FIFO	✓	✓		✓	✓	✓
Block RAM	✓	✓		✓	✓	✓
Shift Register		✓	✓		✓	
Distributed RAM	✓		✓		✓	

Non-Symmetric Aspect Ratio

The core supports generating FIFOs whose write and read ports have different widths, enabling automatic width conversion of the data width. Non-symmetric aspect ratios ranging from 1:8 to 8:1 are supported for the write and read port widths. This feature is available for FIFOs implemented with block RAM that are configured to have independent write and read clocks.

Core Configuration and Implementation

[Table 3-2](#) provides a summary of the supported memory and clock configurations.

Table 3-2: FIFO Configurations

Clock Domain	Memory Type	Supported Configuration	Non-symmetric Aspect Ratios	First-Word Fall-Through
Common	Block RAM	✓		
Common	Distributed RAM	✓		
Common	Shift Register	✓		
Common	Built-in FIFO ¹	✓		
Independent	Block RAM	✓	✓	✓
Independent	Distributed RAM	✓		✓
Independent	Built-in FIFO ^{1,2,3}	✓		

1. The built-in FIFO primitive is only available in the Virtex-4 architecture.
2. For non-symmetric aspect ratios, use the block RAM implementation (feature not supported in Virtex-4 built-in FIFO primitive).
3. FWFT for the built-in FIFO will be supported by the FIFO Generator in a future release. Contact your Xilinx representative for more details.

Independent Clocks: Block RAM and Distributed RAM

This implementation category allows the user to select block RAM or distributed RAM and supports independent clock domains for write and read data accesses. Operations in the read domain are synchronous to the read clock and operations in the write domain are synchronous to the write clock.

The feature set supported for this type of FIFO includes non-symmetric aspect ratios (different write and read port widths), status flags (full, almost full, empty, and almost empty), as well as programmable full and empty flags generated with user-defined thresholds. Optional read data count and write data count indicators provide the number of words in the FIFO relative to their respective clock domains. In addition, optional handshaking and error flags are available (write acknowledge, overflow, valid, and underflow).

Independent Clocks: Virtex-4 Built-in FIFO

This implementation category allows the user to select the built-in FIFO available in the Virtex-4 architecture. Operations in the read domain are synchronous to the read clock and operations in the write domain are synchronous to the write clock.

The feature set supported for this configuration includes status flags (full and empty) and programmable full and empty flags generated with user-defined thresholds. In addition, optional handshaking and error flags are available (write acknowledge, overflow, valid, and underflow).

Common Clock: Block RAM, Distributed RAM, Shift Register

This implementation category allows the user to select block RAM, distributed RAM, or shift register and supports a common clock for write and read data accesses.

The feature set supported for this configuration includes status flags (full, almost full, empty, and almost empty) and programmable empty and full flags generated with user-defined thresholds. In addition, optional handshaking and error flags are supported (write acknowledge, overflow, valid, and underflow), and an optional data count provides the number of words in the FIFO.

Common Clock: Virtex-4 Built-in FIFO

This implementation category allows the user to select the built-in FIFO available in the Virtex-4 architecture and supports a common clock for write and read data accesses.

The feature set supported for this configuration includes status flags (full and empty) and optional programmable full and empty flags with user-defined thresholds. In addition, optional handshaking and error flags are available (write acknowledge, overflow, valid, and underflow).

FIFO Generator Features

Table 3-3 summarizes the FIFO Generator features supported for each clock configuration and memory type.

Table 3-3: FIFO Configurations Summary

FIFO Feature	Independent Clocks			Common Clock		
	Block RAM	Distributed RAM	Built-in FIFO	Block RAM	Distributed RAM, Shift Register	Built-in FIFO
Non-symmetric Aspect Ratios ¹	✓					
Symmetric Aspect Ratios	✓	✓	✓	✓	✓	✓
Almost Full	✓	✓		✓	✓	
Almost Empty	✓	✓		✓	✓	
Handshaking	✓	✓	✓	✓	✓	✓
Data Count	✓	✓		✓	✓	
Programmable Empty/Full Thresholds	✓	✓	✓	✓	✓	✓
First-Word Fall-Through	✓	✓				
DOUT Reset Value	✓ ²	✓		✓ ²	✓	

1. For applications with a single clock that require non-symmetric ports, use the independent clock configuration and connect the write and read clocks to the same source. A dedicated solution for common clocks will be available in a future release. Contact your Xilinx representative for more details.
2. All architectures except for Virtex, Virtex-E, Spartan™-II, and Spartan-II-E.

FIFO Interfaces

The following two sections provide definitions for the FIFO interface signals. Figure 3-1 illustrates these signals (both the standard and optional ports) for a FIFO core that supports independent write and read clocks.

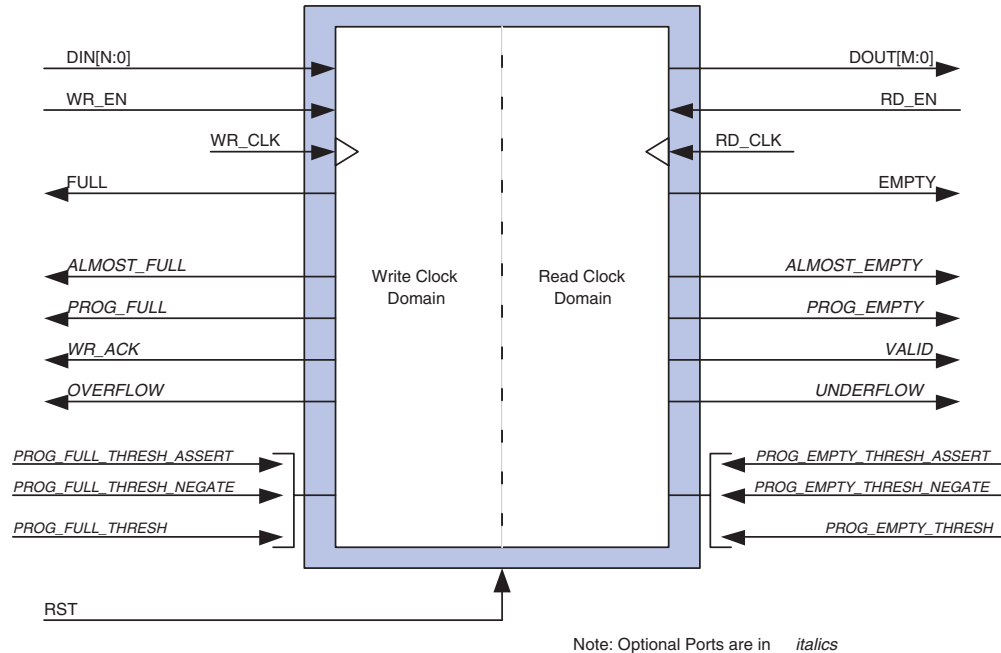


Figure 3-1: FIFO with Independent Clocks: Interface Signals

Interface Signals: FIFOs With Independent Clocks

The signal (RST) causes a reset of the entire core logic (both write and read clock domains) and is defined in Table 3-4. It is an asynchronous input which is synchronized internally in the core before being used. The initial hardware reset should be generated by the user.

Table 3-4: Reset Signal for FIFOs with Independent Clocks

Name	Direction	Description
RST	Input	Reset: This signal is an asynchronous reset that initializes all internal pointers and output registers.

Table 3-5 defines the signals for the write interface for FIFOs with independent clocks. The write interface signals are divided into required and optional signals and all signals are synchronous to the write clock (WR_CLK).

Table 3-5: Write Interface Signals for FIFOs with Independent Clocks

Name	Direction	Description
<i>Required</i>		
WR_CLK	Input	Write Clock: All signals on the write domain are synchronous to this clock.
DIN[N:0]	Input	Data Input: The input data bus used when writing the FIFO.

Table 3-5: Write Interface Signals for FIFOs with Independent Clocks (*Continued*)

Name	Direction	Description
WR_EN	Input	Write Enable: If the FIFO is not full, asserting this signal causes data (on DIN) to be written to the FIFO.
FULL	Output	Full Flag: When asserted, this signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is non-destructive to the contents of the FIFO.
<i>Optional</i>		
ALMOST_FULL	Output	Almost Full: When asserted, this signal indicates that only one more write can be performed before the FIFO is full.
PROG_FULL	Output	Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the assert threshold. It is deasserted when the number of words in the FIFO is less than the negate threshold.
WR_DATA_COUNT [D:0]	Output	Write Data Count: This bus indicates the number of words stored in the FIFO. The count is guaranteed to never under-report the number of words in the FIFO, to ensure the user never overflows the FIFO.
WR_ACK	Output	Write Acknowledge: This signal indicates that a write request (WR_EN) during the prior clock cycle succeeded.
OVERFLOW	Output	Overflow: This signal indicates that a write request (WR_EN) during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is non-destructive to the contents of the FIFO.
PROG_FULL_THRESH	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and deassertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. The user can either choose to set the assert and negate threshold to the same value (using PROG_FULL_THRESH), or the user can control these values independently (using PROG_FULL_THRESH_ASSERT and PROG_FULL_THRESH_NEGATE).
PROG_FULL_THRESH_ASSERT	Input	Programmable Full Threshold Assert: This signal is used to set the upper threshold value for the programmable full flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset.
PROG_FULL_THRESH_NEGATE	Input	Programmable Full Threshold Negate: This signal is used to set the lower threshold value for the programmable full flag, which defines when the signal is deasserted. The threshold can be dynamically set in-circuit during reset.

Table 3-6 defines the signals on the read interface of a FIFO with independent clocks. The read interface signals are divided into required signals and optional signals, and all signals are synchronous to the read clock (RD_CLK).

Table 3-6: Read Interface Signals for FIFOs with Independent Clocks

Name	Direction	Description
<i>Required</i>		
RD_CLK	Input	Read Clock: All signals on the read domain are synchronous to this clock.

Table 3-6: Read Interface Signals for FIFOs with Independent Clocks (Continued)

Name	Direction	Description
DOUT[M:0]	Output	Data Output: The output data bus is driven when reading the FIFO.
RD_EN	Input	Read Enable: If the FIFO is not empty, asserting this signal causes data to be read from the FIFO (output on DOUT).
EMPTY	Output	Empty Flag: When asserted, this signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is non-destructive to the FIFO.
<i>Optional</i>		
ALMOST_EMPTY	Output	Almost Empty Flag: When asserted, this signal indicates that the FIFO is almost empty and one word remains in the FIFO.
PROG_EMPTY	Output	Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold.
RD_DATA_COUNT [C:0]	Output	Read Data Count: This bus indicates the number of words available for reading in the FIFO. The count is guaranteed to never over-report the number of words available for reading, to ensure that the user does not underflow the FIFO.
VALID	Output	Valid: This signal indicates that valid data is available on the output bus (DOUT).
UNDERFLOW	Output	Underflow: Indicates that the read request (RD_EN) during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.
PROG_EMPTY_THRESH	Input	<p>Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and deassertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset.</p> <p>The user can either choose to set the assert and negate threshold to the same value (using PROG_EMPTY_THRESH), or the user can control these values independently (using PROG_EMPTY_THRESH_ASSERT and PROG_EMPTY_THRESH_NEGATE).</p>
PROG_EMPTY_THRESH_ASSERT	Input	Programmable Empty Threshold Assert: This signal is used to set the lower threshold value for the programmable empty flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset.
PROG_EMPTY_THRESH_NEGATE	Input	Programmable Empty Threshold Negate: This signal is used to set the upper threshold value for the programmable empty flag, which defines when the signal is deasserted. The threshold can be dynamically set in-circuit during reset.

Interface Signals: FIFOs with Common Clock

Table 3-7 defines the interface signals of a FIFO with a common write and read clock. The table is divided into standard and optional interface signals, and all signals (except reset) are synchronous to the common clock (CLK).

Table 3-7: Interface Signals for FIFOs with a Common Clock

Name	Direction	Description
<i>Required</i>		
RST	Input	Reset: This signal is an asynchronous reset that initializes all internal pointers and output registers.
CLK	Input	Clock: All signals on the write and read domains are synchronous to this clock.
DIN[N:0]	Input	Data Input: The input data bus used when writing the FIFO.
WR_EN	Input	Write Enable: If the FIFO is not full, asserting this signal causes data (on DIN) to be written to the FIFO.
FULL	Output	Full Flag: When asserted, this signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is non-destructive to the contents of the FIFO.
DOUT[M:0]	Output	Data Output: The output data bus driven when reading the FIFO.
RD_EN	Input	Read Enable: If the FIFO is not empty, asserting this signal causes data to be read from the FIFO (output on DOUT).
EMPTY	Output	Empty Flag: When asserted, this signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is non-destructive to the FIFO.
<i>Optional</i>		
DATA_COUNT [C:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO.
ALMOST_FULL	Output	Almost Full: When asserted, this signal indicates that only one more write can be performed before the FIFO is full.
PROG_FULL	Output	Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the assert threshold. It is deasserted when the number of words in the FIFO is less than the negate threshold.
WR_ACK	Output	Write Acknowledge: This signal indicates that a write request (WR_EN) during the prior clock cycle succeeded.
OVERFLOW	Output	Overflow: This signal indicates that a write request (WR_EN) during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is non-destructive to the contents of the FIFO.
PROG_FULL_THRESH	Input	Programmable Full Threshold: This signal is used to set the threshold value for the assertion and deassertion of the programmable full flag (PROG_FULL). The threshold can be dynamically set in-circuit during reset. The user can either choose to set the assert and negate threshold to the same value (using PROG_FULL_THRESH), or the user can control these values independently (using PROG_FULL_THRESH_ASSERT and PROG_FULL_THRESH_NEGATE).

Table 3-7: Interface Signals for FIFOs with a Common Clock (Continued)

Name	Direction	Description
PROG_FULL_THRESH_ASSERT	Input	Programmable Full Threshold Assert: This signal is used to set the upper threshold value for the programmable full flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset.
PROG_FULL_THRESH_NEGATE	Input	Programmable Full Threshold Negate: This signal is used to set the lower threshold value for the programmable full flag, which defines when the signal is deasserted. The threshold can be dynamically set in-circuit during reset.
ALMOST_EMPTY	Output	Almost Empty Flag: When asserted, this signal indicates that the FIFO is almost empty and one word remains in the FIFO.
PROG_EMPTY	Output	Programmable Empty: This signal is asserted after the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold.
VALID	Output	Valid: This signal indicates that valid data is available on the output bus (DOUT).
UNDERFLOW	Output	Underflow: Indicates that read request (RD_EN) during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.
PROG_EMPTY_THRESH	Input	<p>Programmable Empty Threshold: This signal is used to set the threshold value for the assertion and deassertion of the programmable empty flag (PROG_EMPTY). The threshold can be dynamically set in-circuit during reset.</p> <p>The user can either choose to set the assert and negate threshold to the same value (using PROG_EMPTY_THRESH), or the user can control these values independently (using PROG_EMPTY_THRESH_ASSERT and PROG_EMPTY_THRESH_NEGATE).</p>
PROG_EMPTY_THRESH_ASSERT	Input	Programmable Empty Threshold Assert: This signal is used to set the lower threshold value for the programmable empty flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset.
PROG_EMPTY_THRESH_NEGATE	Input	Programmable Empty Threshold Negate: This signal is used to set the upper threshold value for the programmable empty flag, which defines when the signal is deasserted. The threshold can be dynamically set in-circuit during reset.

Generating the Core

This chapter contains information and instructions for using the Xilinx CORE Generator system to customize the FIFO Generator.

CORE Generator Graphical User Interface

The FIFO Generator graphical user interface (GUI) contains the following six configuration screens.

- FIFO Implementation
- Performance Options and Data Port Parameters
- Optional Flags and Handshaking Options
- Programmable Flags
- Data Count and Reset
- Summary

FIFO Implementation

The main FIFO Generator screen is used to define the component name and provides configuration options for the core.

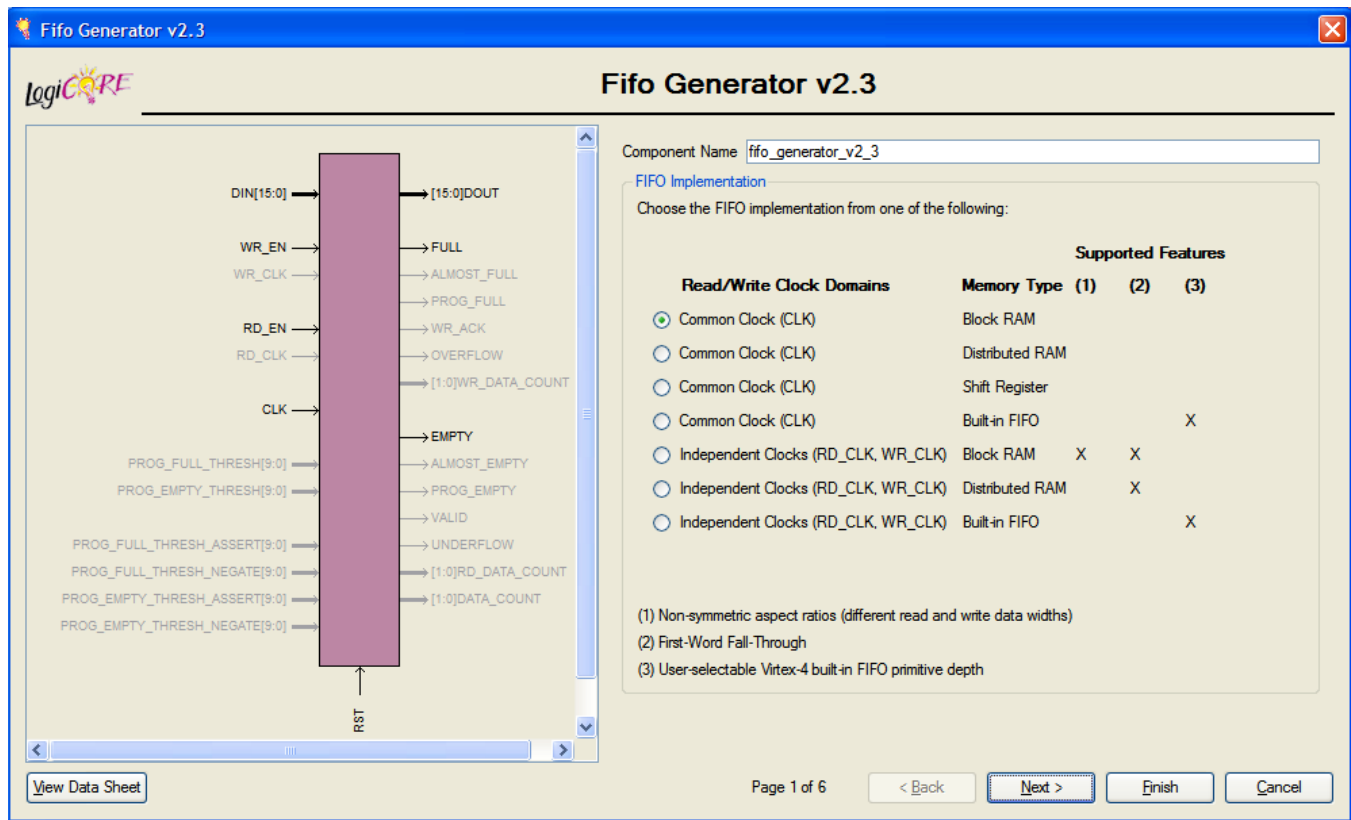


Figure 4-1: Main FIFO Generator Screen

Component Name

Base name of the output files generated for this core. The name must begin with a letter and be composed of the following characters: a to z, 0 to 9, and “_”.

FIFO Implementation

This section of the GUI allows the user to select from a set of available FIFO implementations and supported features. The key supported features that are only available for certain implementations are highlighted by checks in the right-margin. The available options are listed below, with cross-references to additional information.

Common Clock (CLK), Block RAM

For more information, see [“Common Clock FIFO: Block RAM, Distributed RAM, Shift Register,”](#) page 67.

Common Clock (CLK), Distributed RAM

For more information, see [“Common Clock FIFO: Block RAM, Distributed RAM, Shift Register,”](#) page 67.

Common Clock (CLK), Shift Register

For more information, see [“Common Clock: Built-in FIFO,”](#) page 77. This implementation is only available in Virtex-II and newer architectures.

Common Clock (CLK), Built-in FIFO

For more information, see [“Common Clock: Built-in FIFO,”](#) page 77. This implementation is only available when using the Virtex-4 architecture.

Independent Clocks (RD_CLK, WR_CLK), Block RAM

For more information, see [“Independent Clocks: Block RAM and Distributed RAM,”](#) page 42. This implementation optionally supports asymmetric read/write ports and first-word fall-through (selectable in the second window of the GUI as shown in [Figure 4-2](#)).

Independent Clocks (RD_CLK, WR_CLK), Distributed RAM

For more information, see [“Independent Clocks: Block RAM and Distributed RAM,”](#) page 42. This implementation optionally supports first-word fall-through (selectable in the second window of the GUI as shown in [Figure 4-2](#)).

Independent Clocks (RD_CLK, WR_CLK), Built-in FIFO

For more information, see [“Independent Clocks: Built-in FIFO,”](#) page 56. This implementation is only available when using the Virtex-4 architecture.

Performance Options and Data Port Parameters

This screen provides performance options and data port parameters for the core.

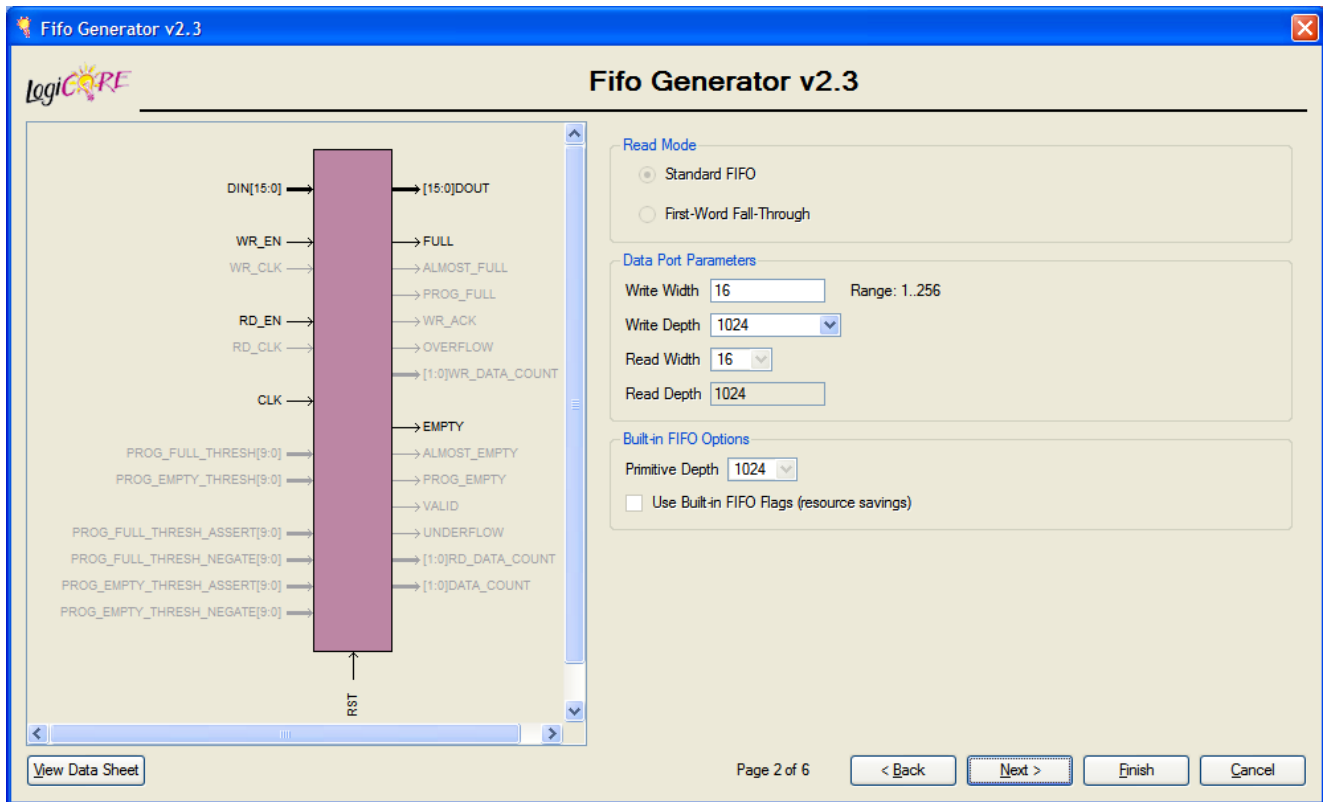


Figure 4-2: Performance Options and Data Port Parameters Screen

Performance Options

Only available when an independent clock FIFO with block RAM or distributed RAM is selected. For more information, see [“Read Operation,” page 44](#).

Standard FIFO

Implements a FIFO with standard latencies, and without using output registers.

First-word Fall-through FIFO

Implements a FIFO with registered outputs. For more information about FWFT functionality, see [“First-Word Fall-Through,” page 21](#).

Data Port Parameters

Input Data Width

Valid range is 1 to 256.

Input Depth

Valid range is 16 to 4194394. Only depths with powers of 2 are allowed.

Output Data Width

Available if independent clocks configuration with block RAM is selected. Valid range must comply with asymmetric port rules. See [“Non-symmetric Aspect Ratios,”](#) page 53.

Output Depth

Automatically calculated based on Input Data Width, Input Depth, and Output Data Width.

Primitive Depth

The primitive depth is the configuration of the built-in FIFO primitive used to generate the core. Valid options are 512, 1024, 2048, and 4096. The default primitive selection will use the minimum number of primitives required, but can be over-written by the user if required. This option is available only if independent clocks or common clock configuration with built-in FIFO is selected.

Optional Flags, Handshaking, and Initialization

This screen allows you to select the optional status flags and set the handshaking options.

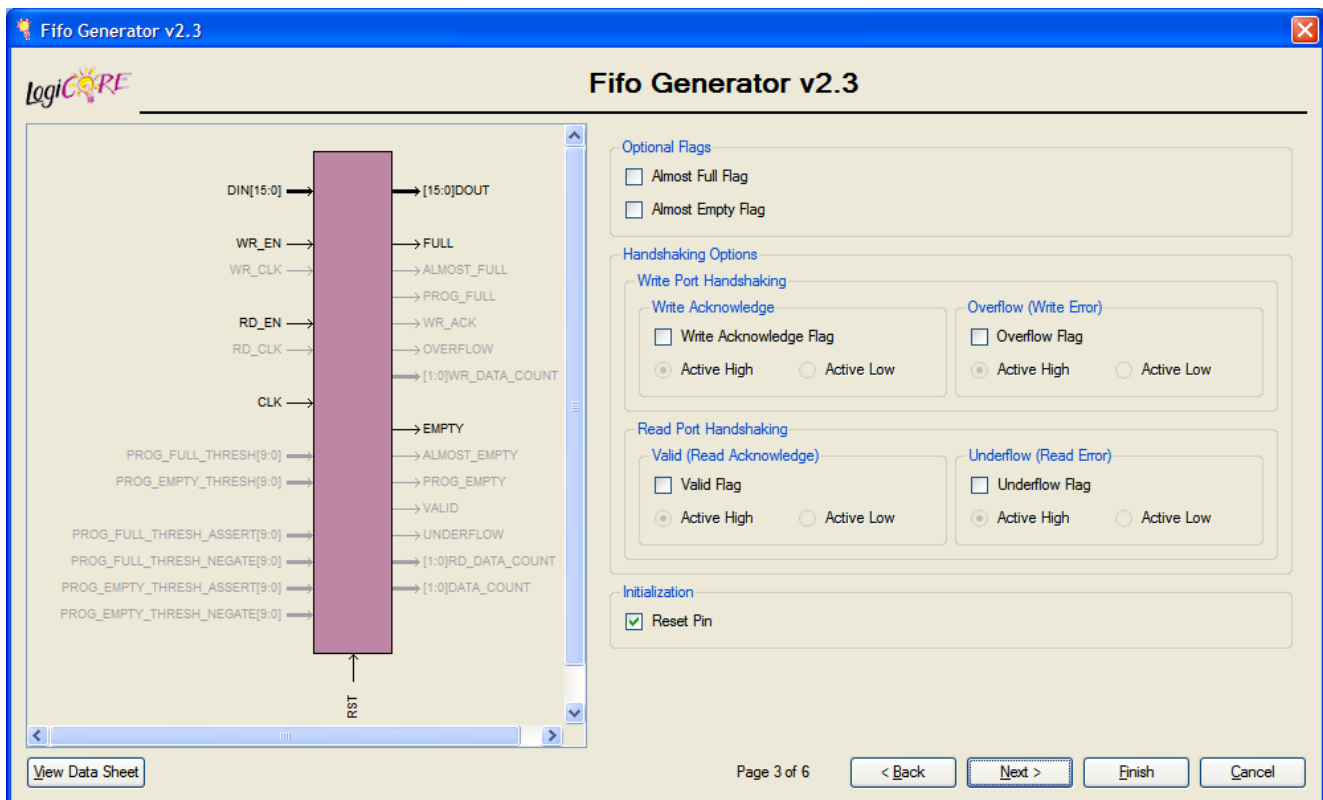


Figure 4-3: Optional Flags, Handshaking and Initialization Options Screen

Optional Flags

Almost Full Flag

Available in all FIFO implementations except those using built-in FIFOs. Generates an output port that indicates the FIFO is almost full (only one more word can be written).

Almost Empty Flag

Available in all FIFO implementations except in those using built-in FIFOs. Generates an output port that indicates the FIFO is almost empty (only one more word can be read).

Write Port Handshaking

Write Acknowledge

Generates write acknowledge flag which reports the status of a write operation. This signal can be configured to be active high or low (default active high).

Overflow (Write Error)

Generates overflow flag which indicates when the previous write operation was not successful. This signal can be configured to be active high or low (default active high).

Read Port Handshaking

Valid (Read Acknowledge)

Generates valid flag which indicates when the data on the output bus is valid. This signal can be configured to be active high or low (default active high).

Underflow (Read Error)

Generates underflow flag to indicate that the previous read request was not successful. This signal can be configured to be active high or low (default active high).

Initialization

Reset Pin

For FIFOs implemented with block RAM or distributed RAM, a reset pin is not required, and the input pin is optional.

Programmable Flags

Use this screen to select the programmable flag type when generating a specific FIFO Generator configuration.

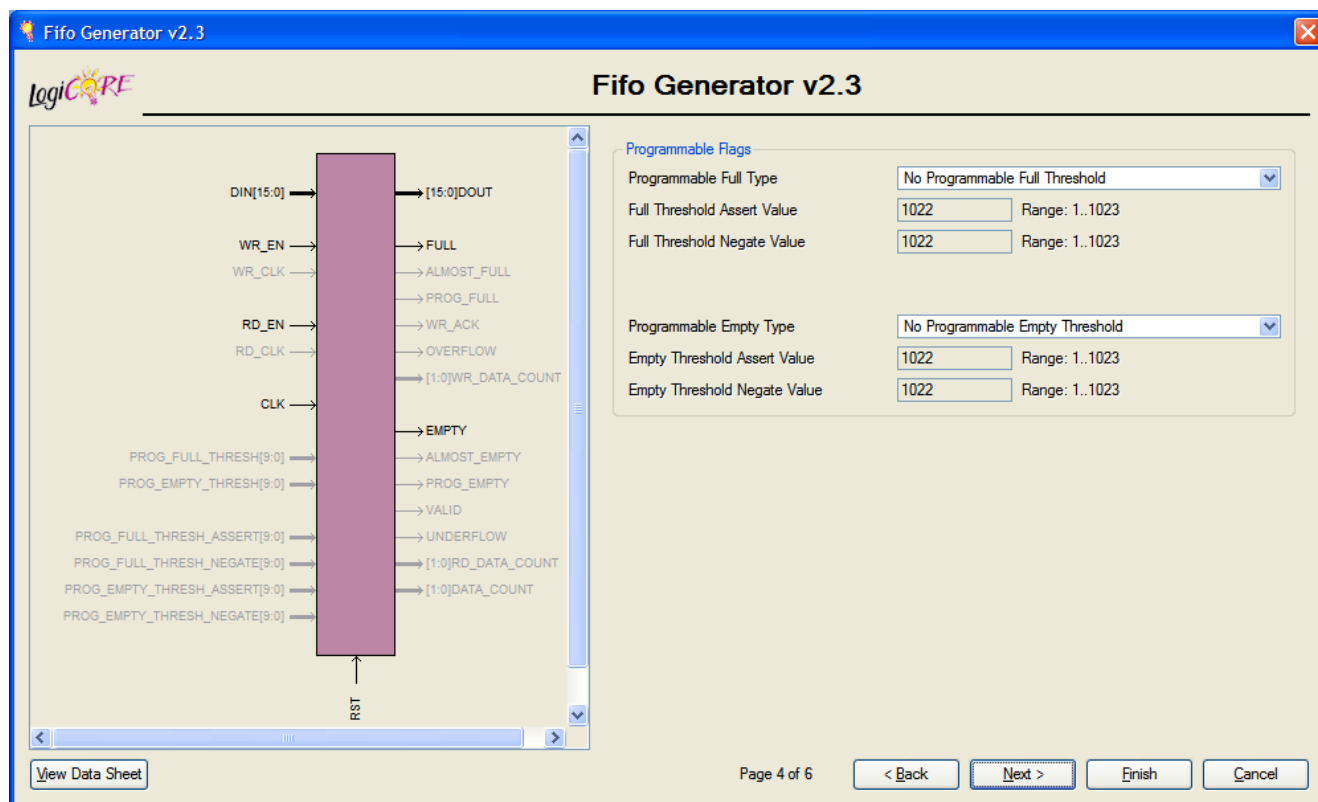


Figure 4-4: Programmable Flags Screen

Programmable Flags

Programmable Full Type

Select a programmable full threshold type from the drop-down menu. The valid range for each threshold is displayed, and will vary, depending on options selected elsewhere in the GUI.

Full Threshold Assert Value

Available when Programmable Full with Single or Multiple Threshold Constants is selected. Enter a user-defined value, or select a preset value from the drop-down menu. The valid range for this threshold is provided in the GUI. When using a single threshold constant, only the assert threshold value is used.

Full Threshold Negate Value

Available when Programmable Full with Multiple Threshold Constants is selected. Enter a user-defined value, or select a preset value from the drop-down menu. The valid range for this threshold is provided in the GUI.

Programmable Empty Type

Select a programmable empty threshold type from the drop-down menu. The valid range for each threshold is displayed, and will vary depending on options selected elsewhere in the GUI.

Empty Threshold Assert Value

Available when Programmable Empty with Single or Multiple Threshold Constants is selected. Enter a user-defined value, or select a preset value from the drop-down menu. The valid range for this threshold is provided in the GUI. When using a single threshold constant, only the assert value is used.

Empty Threshold Negate Value

Available when Programmable Empty with Multiple Threshold Constants is selected. Enter a user-defined value, or select a preset value from the drop-down menu. The valid range for this threshold is provided in the GUI.

Data Count and Reset

Use this screen to set data count and reset parameters.

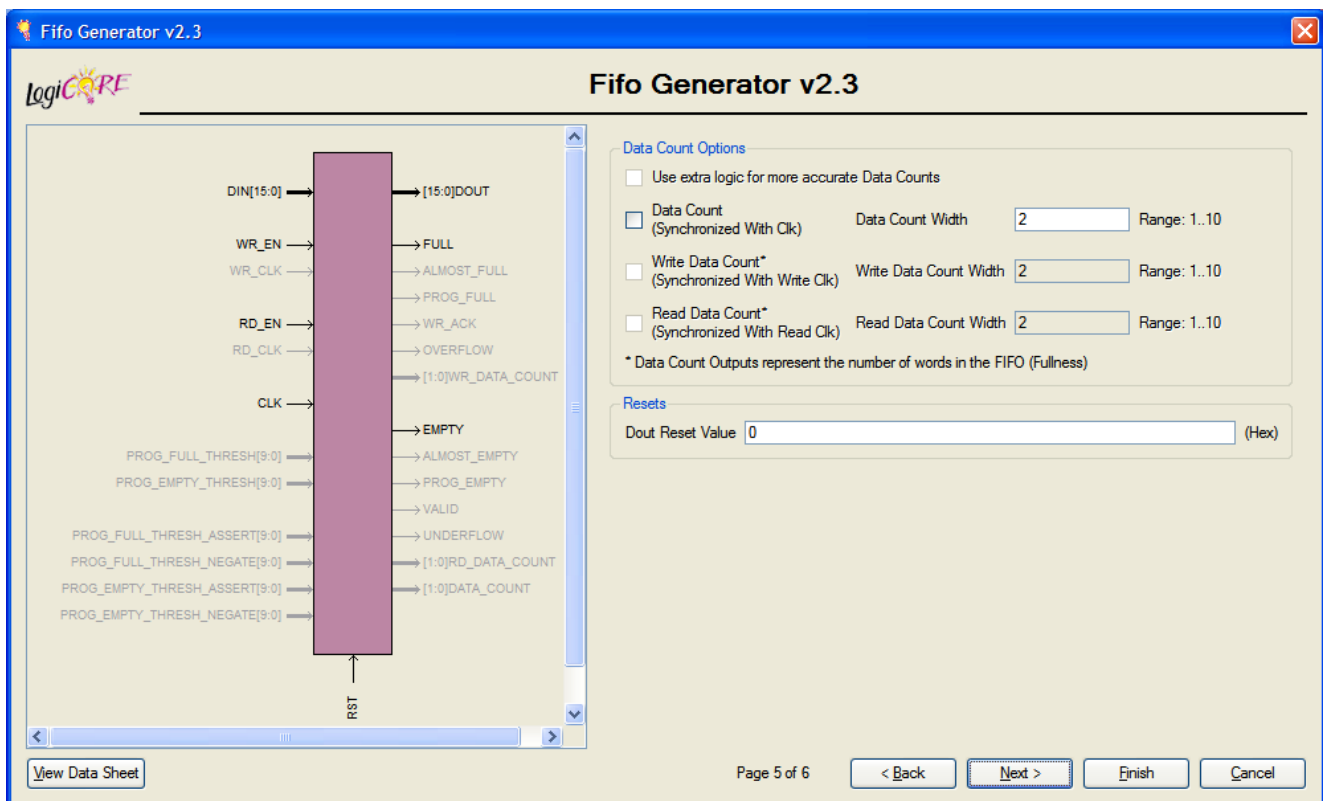


Figure 4-5: Data Count and Reset Screen

Data Count and Reset Options

Data Count

Use Extra Logic For More Accurate Data Counts

Only available for independent clocks FIFO with block RAM or distributed RAM, and when using first-word fall-through. This option uses additional external logic to generate a more accurate data count. See [“First-Word Fall-Through Data Count,” page 307](#) for details.

Write Data Count

Available when an independent clocks FIFO with block RAM or distributed RAM is selected. Valid range is from 1 to \log_2 (write depth).

Read Data Count

Available when an independent clocks FIFO with block RAM or distributed RAM is selected. Valid range is from 1 to \log_2 (read depth).

Data Count

Available when a common clock FIFO with block RAM, distributed RAM or shift registers is selected. Valid range is from 1 to \log_2 of input depth.

Resets

Dout Reset Value

Available in Virtex-II and newer architectures for all implementations using block RAM, distributed RAM, or shift register memory. This text box indicates the hexadecimal value which is asserted on the output of the FIFO when RST is asserted.

Summary

This screen summarizes the FIFO type, dimensions, and any additional features selected. In the Additional Features section, most features display either *Not Selected* (if the feature is not used), or *Selected* (if the feature is used).

Note: Write depth and read depth provide the actual FIFO depths for the selected configuration. These depths may differ slightly from the depth selected on page 2 of the FIFO GUI.

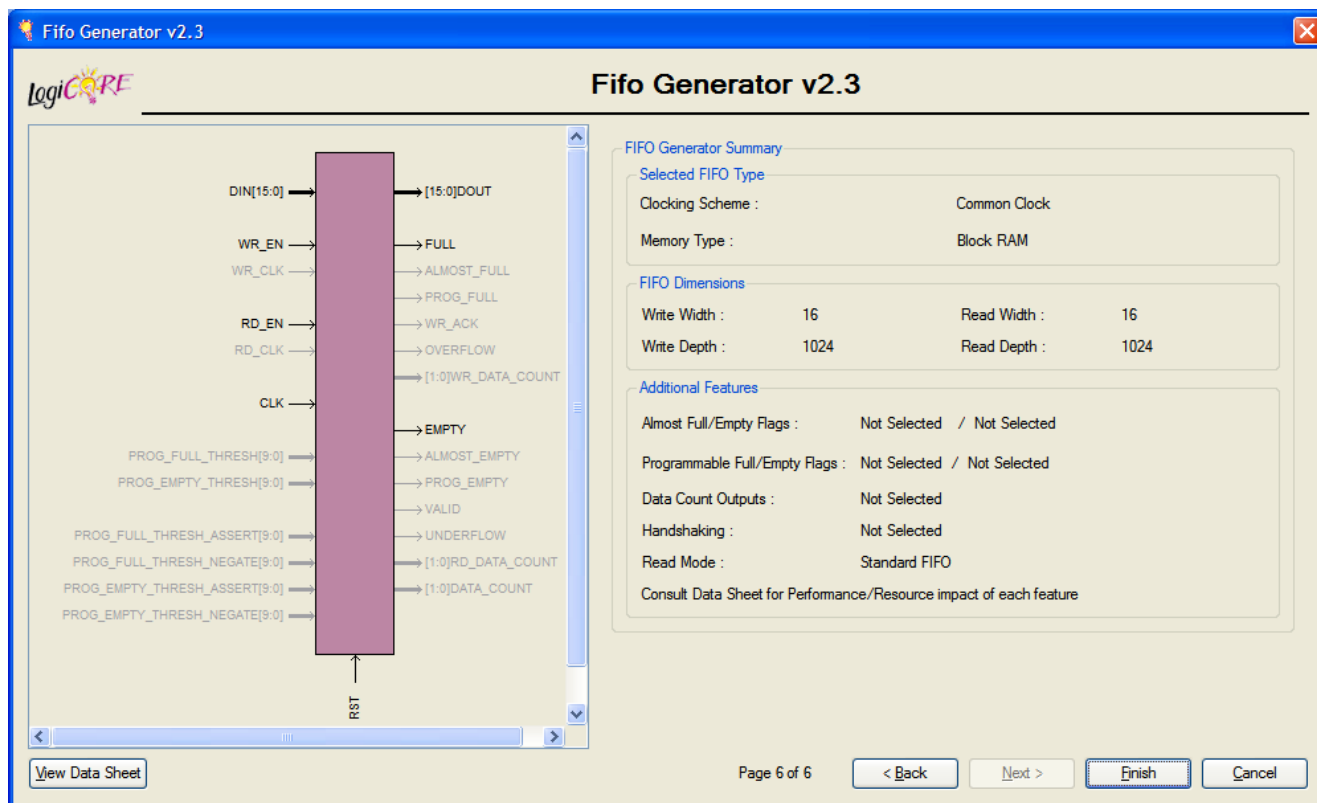


Figure 4-6: Summary Screen

Designing with the Core

This chapter describes the steps required to turn a FIFO Generator core into a fully-functioning design integrated with the user application logic. This chapter is divided into sections that describe the design steps required for each feature's implementation. It is important to note that depending on the configuration of the FIFO core, only a subset of the implementation details provided are applicable. For successful use of a FIFO core, the design guidelines discussed in this chapter must be observed.

General Design Guidelines

Know the Degree of Difficulty

A fully-compliant and feature-rich FIFO design is challenging to implement in any technology. For this reason, it is important to understand that the degree of difficulty can be significantly influenced by:

- Maximum system clock frequency.
- Targeted device architecture.
- Specific user application.

Ensure that design techniques are used to facilitate implementation, including pipelining and use of constraints (timing constraints, and placement and/or area constraints).

Understand Signal Pipelining

To understand the nature of FIFO designs, it is important to understand how pipelining is used to maximize performance and implement synchronization logic for clock-domain crossing. Data written into the write interface may take multiple clock cycles before it can be accessed on the read interface. Refer to the selected implementation below for details on the latency of a particular FIFO configuration.

Initializing the FIFO Generator

When designing with the Virtex-4 built-in FIFO or common clock shift register FIFO, the FIFO must be reset after the FPGA is configured and before operation begins. A reset pin (RST) is provided, and is an asynchronous reset that clears the internal counters and output registers. For FIFOs implemented with block RAM or distributed RAM, a reset is not required, and the input pin is optional. When a reset is implemented, it is synchronized to the clock domain in which it is used to ensure that the FIFO initializes to a known state. This synchronization logic allows for proper reset timing of the core logic, avoiding

glitches and metastable behavior. Refer to the selected implementation below for details on the reset behavior for a specific FIFO configuration.

Independent Clocks: Block RAM and Distributed RAM

Figure 5-1 illustrates the functional implementation of a FIFO configured with independent clocks. This implementation uses block RAM or distributed RAM for memory, counters for write and read pointers, conversions between binary and Gray code for memory, and logic for calculating the status flags.

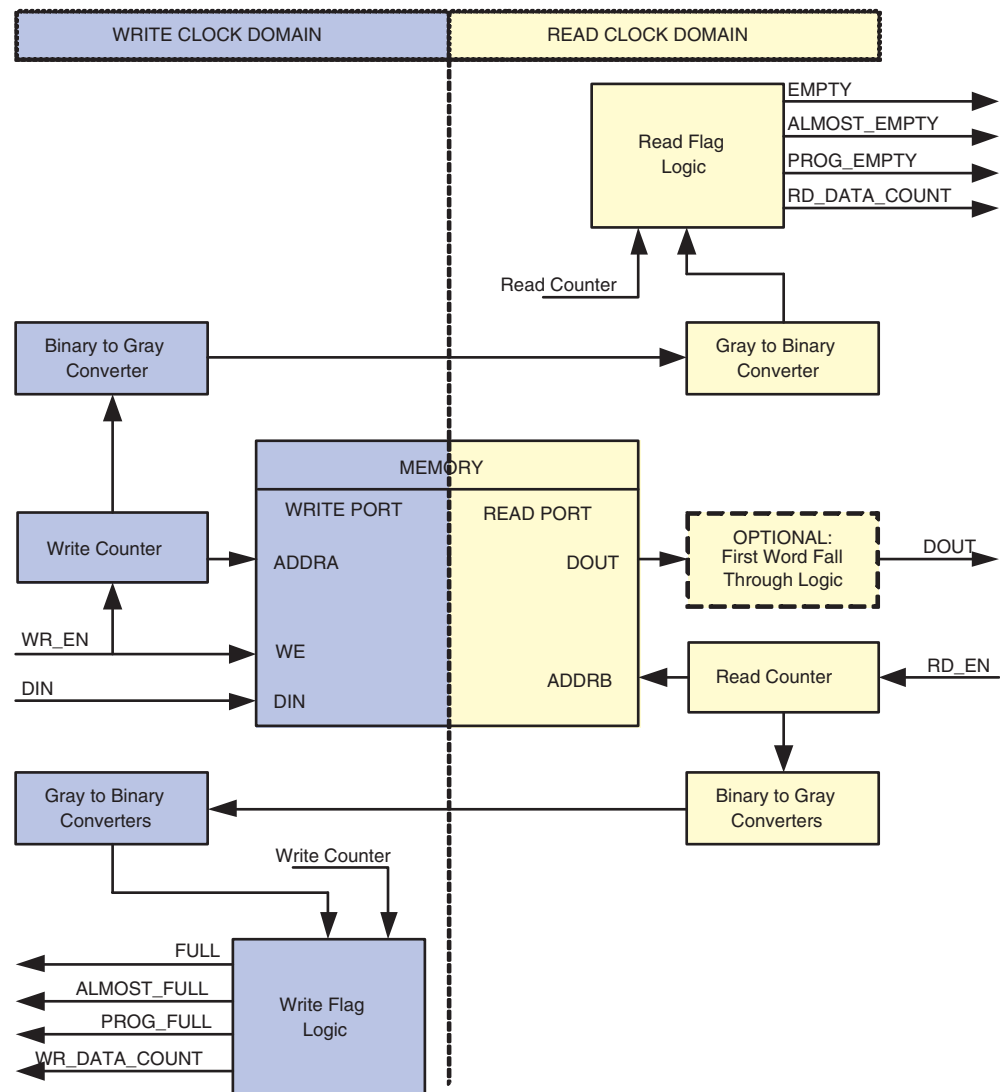


Figure 5-1: Functional Implementation of a FIFO with Independent Clock Domains

This FIFO is designed to support an independent read clock (RD_CLK) and write clock (WR_CLK); in other words, there is no required relationship between RD_CLK and

WR_CLK with regard to frequency or phase. The FIFO interface signals are only valid in their respective clock domains and are summarize in [Table 5-1](#).

Table 5-1: Interface Signals and Corresponding Clock Domains

WR_CLK	RD_CLK
DIN	DOUT
WR_EN	RD_EN
FULL	EMPTY
ALMOST_FULL	ALMOST_EMPTY
PROG_FULL	PROG_EMPTY
WR_ACK	VALID
OVERFLOW	UNDERFLOW
WR_DATA_COUNT	RD_DATA_COUNT

For FIFO cores using independent clocks, the timing relationship between the write and read operations and the status flags is affected by the relationship of the two clocks. For example, the timing between writing to an empty FIFO and the deassertion of EMPTY is determined by the phase and frequency relationship between the write and read clocks. For additional information refer to the [“Synchronization Considerations,”](#) page 66.

Write Operation

This section describes the behavior of a FIFO write operation and the associated status flags. When write enable is asserted and the FIFO is not full, data is added to the FIFO from the input bus (DIN) and write acknowledge (WR_ACK) is asserted. If the FIFO is continuously written to without being read, it fills with data. Write operations are only successful when the FIFO is not full. When the FIFO is full and a write is initiated, the request is ignored, the overflow flag is asserted and there is no change in the state of the FIFO (overflowing the FIFO is non-destructive).

ALMOST_FULL and FULL Flags

The almost full flag (ALMOST_FULL) indicates that only one more write can be performed before FULL is asserted. This flag is active high and synchronous to the write clock (WR_CLK).

The full flag (FULL) indicates that the FIFO is full and no more writes can be performed until data is read out. This flag is active high and synchronous to the write clock (WR_CLK). If a write is initiated when FULL is asserted, the write request is ignored and OVERFLOW is asserted.

Example Operation

[Figure 5-2](#) shows a typical write operation. The user asserts WR_EN, causing a write operation to occur on the next rising edge of the WR_CLK. Since the FIFO is not full, WR_ACK is asserted, acknowledging a successful write operation. When only one additional word can be written into the FIFO, the FIFO asserts the ALMOST_FULL flag. When ALMOST_FULL is asserted, one additional write causes the FIFO to assert FULL. When a write occurs after FULL is asserted, WR_ACK is deasserted and OVERFLOW is

asserted, indicating an overflow condition. Once the user performs one or more read operations, the FIFO deasserts `FULL`, and data can successfully be written to the FIFO, as is indicated by the assertion of `WR_ACK` and deassertion of `OVERFLOW`.

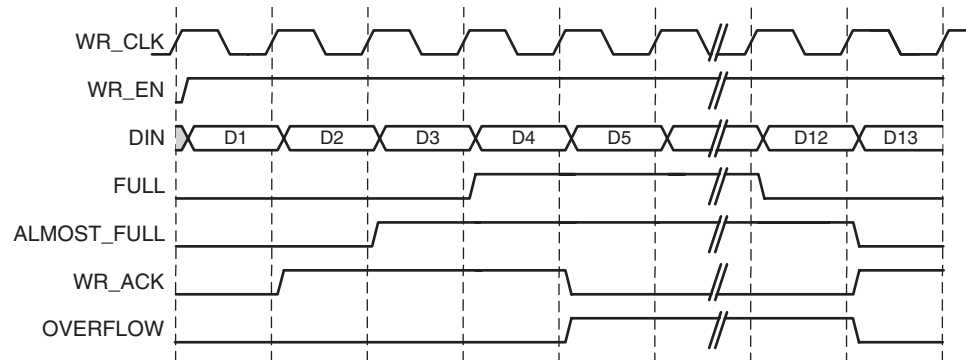


Figure 5-2: Write Operation for a FIFO with Independent Clocks

Read Operation

This section describes the behavior of a FIFO read operation and the associated status flags. When read enable is asserted and the FIFO is not empty, data is read from the FIFO on the output bus (`DOUT`), and the valid flag (`VALID`) is asserted. If the FIFO is continuously read without being written, the FIFO empties. Read operations are successful when the FIFO is not empty. When the FIFO is empty and a read is requested, the read operation is ignored, the underflow flag is asserted and there is no change in the state of the FIFO (underflowing the FIFO is non-destructive).

ALMOST_EMPTY and EMPTY Flags

The almost empty flag (`ALMOST_EMPTY`) indicates that the FIFO will be empty after one more read operation. This flag is active high and synchronous to `RD_CLK`. This flag is asserted when the FIFO has one remaining word that can be read.

The empty flag (`EMPTY`) indicates that the FIFO is empty and no more reads can be performed until data is written into the FIFO. This flag is active high and synchronous to the read clock (`RD_CLK`). If a read is initiated when `EMPTY` is asserted, the request is ignored and `UNDERFLOW` is asserted.

Modes of Read Operation

The FIFO Generator supports two modes of read options, standard read operation and first-word fall-through (FWFT) read operation. The standard read operation provides the user data on the cycle after it was requested. The FWFT read operation provides the user data on the same cycle in which it is requested.

Standard FIFO Read Operation

For a standard FIFO read operation, after read enable is asserted and if the FIFO is not empty, the next data stored in the FIFO is driven on the output bus (`DOUT`) and the valid flag (`VALID`) is asserted.

Figure 5-3 shows a standard read access. Once the user writes at least one word into the FIFO, `EMPTY` is deasserted—indicating data is available to be read. The user asserts `RD_EN`, causing a read operation to occur on the next rising edge of `RD_CLK`. The FIFO outputs the next available word on `DOUT` and asserts `VALID`, indicating a successful read

operation. When the last data word is read from the FIFO, the FIFO asserts `EMPTY`. If the user continues to assert `RD_EN` while `EMPTY` is asserted, the read request is ignored, `VALID` is deasserted, and `UNDERFLOW` is asserted. Once the user performs a write operation, the FIFO deasserts `EMPTY`, allowing the user to resume valid read operations, as indicated by the assertion of `VALID` and deassertion of `UNDERFLOW`.

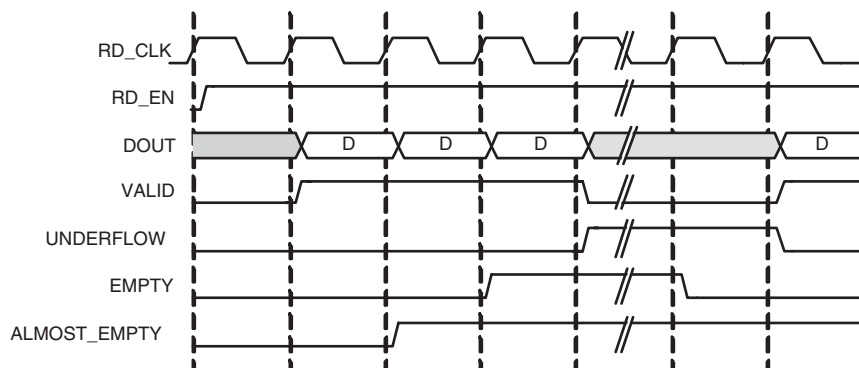


Figure 5-3: Standard Read Operation for a FIFO with Independent Clocks

First-Word Fall-Through FIFO Read Operation

The first-word fall-through (FWFT) feature provides the ability to look-ahead to the next word available from the FIFO without issuing a read operation. When data is available in the FIFO, the first word falls through the FIFO and appears automatically on the output bus (DOUT). Once the first word appears on DOUT, `EMPTY` is deasserted indicating one or more readable words in the FIFO, and `VALID` is asserted, indicating a valid word is present on DOUT.

[Figure 5-4](#) shows a FWFT read access. Initially, the FIFO is not empty, the next available data word is placed on the output bus (DOUT), and `VALID` is asserted. When the user asserts `RD_EN`, the next rising clock edge of `RD_CLK` places the next data word onto DOUT. After the last data word has been placed on DOUT, an additional read request by the user causes the data on DOUT to become invalid, as indicated by the deassertion of `VALID` and the assertion of `EMPTY`. Any further attempts to read from the FIFO results in an underflow condition.

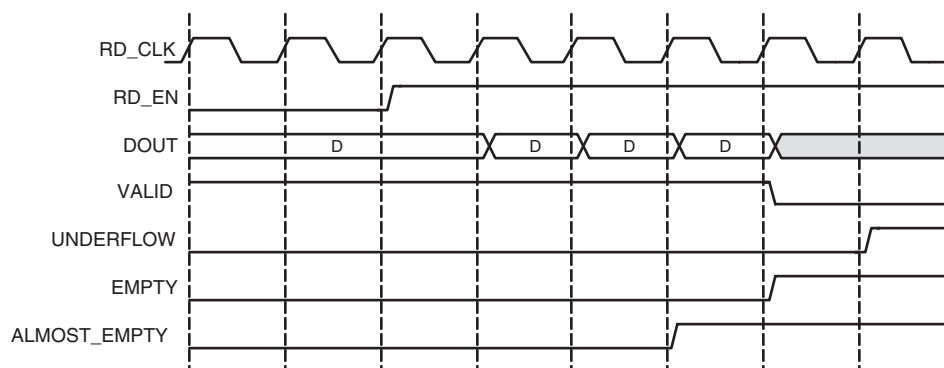


Figure 5-4: FWFT Read Operation for a FIFO with Independent Clocks

Handshaking Flags

Handshaking flags (valid, underflow, write acknowledge and overflow) are supported to provide additional information regarding the status of the write and read operations. The handshaking flags are optional, and can be configured as active high or active low through the CORE Generator GUI (see Handshaking Options in Chapter 4 for more information). These flags (configured as active high) are illustrated in [Figure 5-5](#).

Valid

The operation of the valid flag (`VALID`) is dependent on the read mode of the FIFO. This flag is synchronous to the read clock (`RD_CLK`).

Standard FIFO Read Operation

For standard read operation, the `VALID` flag is asserted at the rising edge of `RD_CLK` for each successful read operation, and indicates that the data on the `DOUT` bus is valid. When a read request is unsuccessful (when the FIFO is empty), `VALID` is not asserted.

FWFT FIFO Read Operation

For FWFT read operation, the `VALID` flag indicates the data on the output bus (`DOUT`) is valid for the current cycle. A read request does not have to happen for data to be present and valid, as the first-word fall-through logic automatically places the next data to be read on the `DOUT` bus. `VALID` is asserted if there is one or more words in the FIFO. `VALID` is deasserted when there are no more words in the FIFO.

Underflow

The underflow flag (`UNDERFLOW`) is used to indicate that a read operation is unsuccessful. This occurs when a read is initiated and the FIFO is empty. This flag is synchronous with the read clock (`RD_CLK`). Underflowing the FIFO does not change the state of the FIFO (it is non-destructive).

Write Acknowledge

The write acknowledge flag (`WR_ACK`) is asserted at the completion of each successful write operation and indicates that the data on the `DIN` port has been stored in the FIFO. This flag is synchronous to the write clock (`WR_CLK`).

Overflow

The overflow flag (`OVERFLOW`) is used to indicate that a write operation is unsuccessful. This flag is asserted when a write is initiated to the FIFO while `FULL` is asserted. The overflow flag is synchronous to the write clock (`WR_CLK`). Overflowing the FIFO does not change the state of the FIFO (it is non-destructive).

Example Operation

[Figure 5-5](#) illustrates the behavior of the FIFO flags. On the write interface, `FULL` is not asserted and writes to the FIFO are successful (as indicated by the assertion of `WR_ACK`). When a write occurs after `FULL` is asserted, `WR_ACK` is deasserted and `OVERFLOW` is asserted, indicating an overflow condition. On the read interface, once the FIFO is not `EMPTY`, the FIFO accepts read requests. In standard FIFO operation, `VALID` is asserted and `DOUT` is updated on the clock cycle following the read request. In FWFT operation, `VALID` is asserted and `DOUT` is updated prior to a read request being issued. When a read request

is issued while `EMPTY` is asserted, `VALID` is deasserted and `UNDERFLOW` is asserted, indicating an underflow condition.

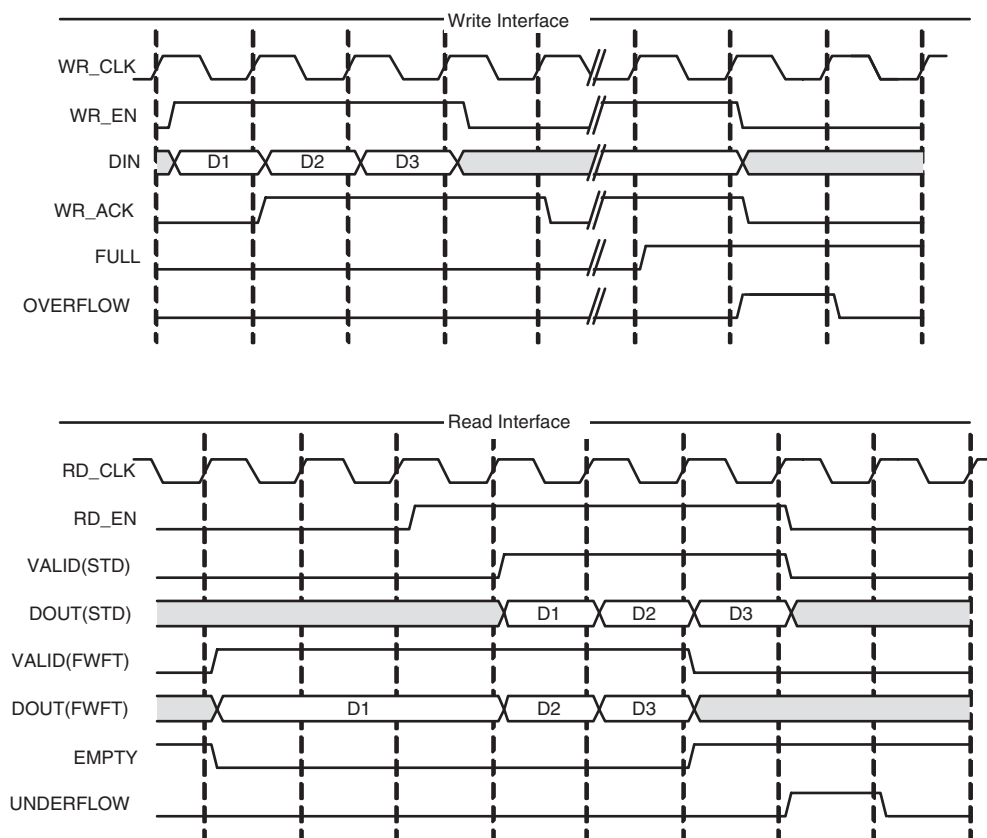


Figure 5-5: Handshaking Signals for a FIFO with Independent Clocks

Programmable Flags

The FIFO supports programmable flags to indicate that the FIFO has reached a user-defined fill level.

- Programmable full (`PROG_FULL`) indicates that the FIFO has reached a user-defined full threshold.
- Programmable empty (`PROG_EMPTY`) indicates that the FIFO has reached a user-defined empty threshold.

For these thresholds, the user can set a constant value or choose to have dedicated input ports, enabling the thresholds to change dynamically in circuit. Hysteresis is also optionally supported, by providing unique assert and negate values for each flag. Detailed information about these options are provided below.

Programmable Full

The FIFO Generator supports four ways to define the programmable full threshold:

- Single threshold constant
- Single threshold with dedicated input port
- Assert and negate threshold constants (provides hysteresis)

- Assert and negate thresholds with dedicated input ports (provides hysteresis)

These options are available in the CORE Generator GUI and accessed within the programmable flags window (Figure 4-4).

The programmable full flag (PROG_FULL) is asserted when the number of entries in the FIFO is greater than or equal to the user-defined assert threshold. When the programmable full flag is asserted, the FIFO can continue to be written to until the full flag (FULL) is asserted. If the number of words in the FIFO is less than the negate threshold, the flag is deasserted.

Note: If a write operation occurs on a rising clock edge that causes the number of words to meet or exceed the programmable full threshold, then the programmable full flag will assert on the next rising clock edge. The deassertion of the programmable full flag has a longer delay, and depends on the relationship between the write and read clocks.

Programmable Full: Single Threshold

This option enables the user to set a single threshold value for the assertion and deassertion of PROG_FULL. When the number of entries in the FIFO is greater than or equal to the threshold value, PROG_FULL is asserted. When the number of entries in the FIFO is less than the threshold value, PROG_FULL is deasserted.

There are two options for implementing this threshold:

- **Single threshold constant.** User specifies the threshold value through the CORE Generator GUI. Once the core is generated, this value can only be changed by re-generating the core. This option consumes fewer resources than the single threshold with dedicated input port.
- **Single threshold with dedicated input port.** User specifies the threshold value through an input port (PROG_FULL_THRESH) on the core. This input can be changed while the FIFO is in reset, providing the user the flexibility to change the programmable full threshold in-circuit without re-generating the core.

Note: Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-6 shows the programmable full flag with a single threshold. The user writes to the FIFO until there are seven words in the FIFO. Since the programmable full threshold is set to seven, the FIFO asserts PROG_FULL once seven words are written into the FIFO. Note that both write data count (WR_DATA_COUNT) and PROG_FULL have one clock cycle of delay. Once the FIFO has six or fewer words in the FIFO, PROG_FULL is deasserted.

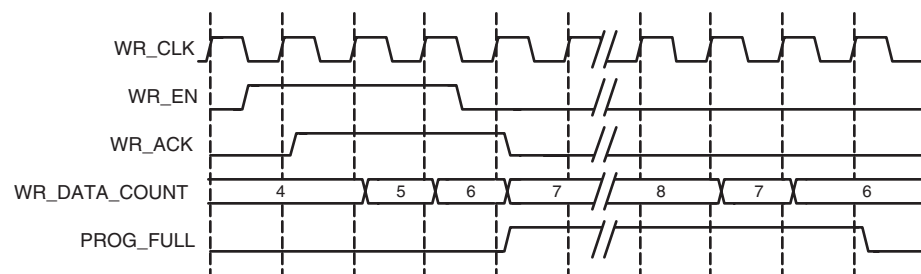


Figure 5-6: Programmable Full Single Threshold: Threshold Set to 7

Programmable Full: Assert and Negate Thresholds

This option enables the user to set separate values for the assertion and deassertion of PROG_FULL. When the number of entries in the FIFO is greater than or equal to the assert value, PROG_FULL is asserted. When the number of entries in the FIFO is less than the negate value, PROG_FULL is deasserted.

There are two options for implementing these thresholds:

- Assert and negate threshold constants: User specifies the threshold values through the CORE Generator GUI. Once the core is generated, these values can only be changed by re-generating the core. This option consumes fewer resources than the assert and negate thresholds with dedicated input ports.
- Assert and negate thresholds with dedicated input ports: User specifies the threshold values through input ports on the core. These input ports can be changed while the FIFO is in reset, providing the user the flexibility to change the values of the programmable full assert (`PROG_FULL_THRESH_ASSERT`) and negate (`PROG_FULL_THRESH_NEGATE`) thresholds in-circuit without re-generating the core.

Note: The full assert value must be larger than the full negate value. Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-7 shows the programmable full flag with assert and negate thresholds. The user writes to the FIFO until there are 10 words in the FIFO. Because the assert threshold is set to 10, the FIFO then asserts `PROG_FULL`. The negate threshold is set to seven, and the FIFO deasserts `PROG_FULL` once six words or fewer are in the FIFO. Note that both write data count (`WR_DATA_COUNT`) and `PROG_FULL` have one clock cycle of delay.

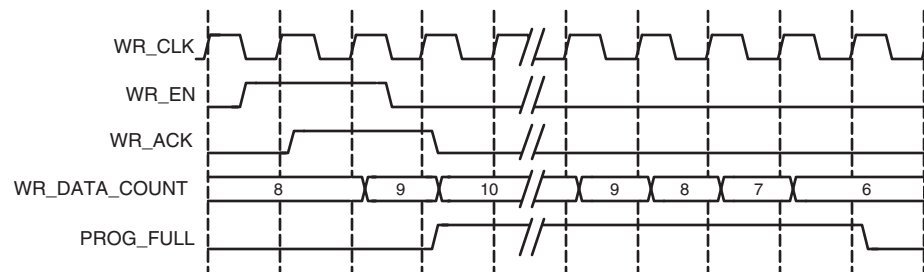


Figure 5-7: Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7

Programmable Empty

The FIFO Generator supports four ways to define the programmable empty thresholds.

- Single threshold constant
- Single threshold with dedicated input port
- Assert and negate threshold constants (provides hysteresis)
- Assert and negate thresholds with dedicated input ports (provides hysteresis)

These options are available in the CORE Generator GUI and accessed within the programmable flags window (Figure 4-4).

The programmable empty flag (`PROG_EMPTY`) is asserted when the number of entries in the FIFO is less than or equal to the user-defined assert threshold. If the number of words in the FIFO is greater than the negate threshold, the flag is deasserted.

Note: If a read operation occurs on a rising clock edge that causes the number of words in the FIFO to be equal to or less than the programmable empty threshold, then the programmable empty flag will assert on the next rising clock edge. The deassertion of the programmable empty flag has a longer delay, and depends on the read and write clocks.

Programmable Empty: Single Threshold

This option enables the user to set a single threshold value for the assertion and deassertion of `PROG_EMPTY`. When the number of entries in the FIFO is less than or equal

to the threshold value, `PROG_EMPTY` is asserted. When the number of entries in the FIFO is greater than the threshold value, `PROG_EMPTY` is deasserted.

There are two options for implementing this threshold.

- **Single threshold constant:** User specifies the threshold value through the CORE Generator GUI. Once the core is generated, this value can only be changed by re-generating the core. This option consumes fewer resources than the single threshold with dedicated input port.
- **Single threshold with dedicated input port:** User specifies the threshold value through an input port (`PROG_EMPTY_THRESH`) on the core. This input can be changed while the FIFO is in reset, providing the user the flexibility to change the programmable empty threshold in-circuit without re-generating the core.

Note: Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-8 shows the programmable empty flag with a single threshold. The user writes to the FIFO until there are five words in the FIFO. Since the programmable empty threshold is set to four, `PROG_EMPTY` is asserted until more than four words are present in the FIFO. Once five words (or more) are present in the FIFO, `PROG_EMPTY` is deasserted. Note that both read data count (`RD_DATA_COUNT`) and `PROG_EMPTY` have one clock cycle of delay.

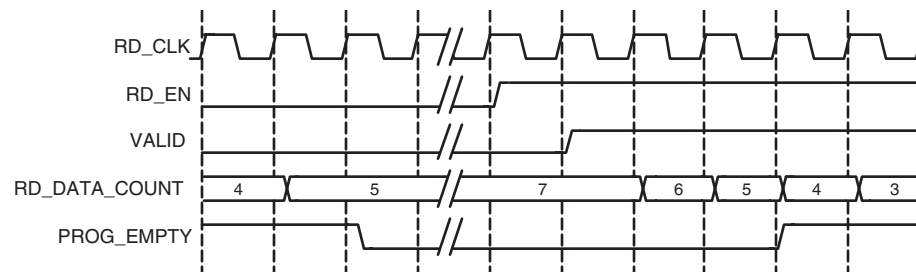


Figure 5-8: Programmable Empty with Single Threshold: Threshold Set to 4

Programmable Empty: Assert and Negate Thresholds

This option enables the user to set separate values for the assertion and deassertion of `PROG_EMPTY`. When the number of entries in the FIFO is less than or equal to the assert value, `PROG_EMPTY` is asserted. When the number of entries in the FIFO is greater than the negate value, `PROG_EMPTY` is deasserted.

There are two options for implementing the assert and negate thresholds.

- **Assert and negate threshold constants.** The threshold values are specified through the CORE Generator GUI. Once the core is generated, these values can only be changed by re-generating the core. This option consumes fewer resources than the assert and negate thresholds with dedicated input ports.
- **Assert and negate thresholds with dedicated input ports.** The threshold values are specified through input ports on the core. These input ports can be changed while the FIFO is in reset, providing the user the flexibility to change the values of the programmable empty assert (`PROG_EMPTY_THRESH_ASSERT`) and negate (`PROG_EMPTY_THRESH_NEGATE`) thresholds in-circuit without regenerating the core.

Note: The empty assert value must be less than the empty negate value. Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-9 shows the programmable empty flag with assert and negate thresholds. The user writes to the FIFO until there are eleven words in the FIFO. Since the programmable empty deassert value is set to ten, `PROG_EMPTY` is deasserted when there are more than ten words in the FIFO. Once the FIFO contains less than or equal to the programmable empty

negate value (set to seven), `PROG_EMPTY` is asserted. Note that both read data count (`RD_DATA_COUNT`) and `PROG_EMPTY` have one clock cycle of delay.

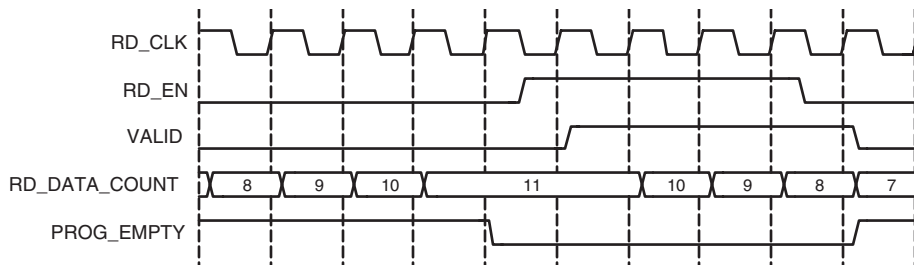


Figure 5-9: Programmable Empty with Assert and Negate Thresholds: Assert Set to 7 and Negate Set to 10

Programmable Empty for First-Word Fall-Through

For FWFT FIFOs, `PROG_EMPTY` is guaranteed to be asserted when the number of words in the FIFO is less than or equal to the programmable empty assert threshold. Under certain conditions, it is possible for `PROG_EMPTY` to violate this rule, but only while `EMPTY` is asserted. `PROG_EMPTY` should be assumed to be asserted whenever `EMPTY` is asserted.

Data Counts

The data count signals (`RD_DATA_COUNT`, `WR_DATA_COUNT`) provide an indication of the number of words in the FIFO relative to the read or write clock domain respectively. These signals are optional outputs of the FIFO Generator, and are enabled through the CORE Generator GUI.

Read Data Count

Read data count (`RD_DATA_COUNT`) pessimistically reports the number of words available for reading. The count is guaranteed to never over-report the number of words available in the FIFO (although it may temporarily under-report the number of words available) to ensure that the user never underflows the FIFO. The user can specify the width of the read data count bus with a maximum width of \log_2 (read depth). If the width specified is smaller than the maximum allowable width, the bus is truncated with the lower bits removed.

For example, the user can specify to use two bits out of a maximum allowable three bits (provided a FIFO depth of eight). These two bits indicate the number of words in the FIFO, with a quarter resolution. This provides a status of the contents of the FIFO for the read clock domain.

Note: If a read operation occurs on a rising clock edge of `RD_CLK`, that read is reflected on the `RD_DATA_COUNT` signal following the next rising clock edge. A write operation on the `WR_CLK` clock domain may take a number of clock cycles before being reflected in the `RD_DATA_COUNT`.

Write Data Count

Write data count (`WR_DATA_COUNT`) pessimistically reports the number of words written into the FIFO. The count is guaranteed to never under-report the number of words in the FIFO (although it may temporarily over-report the number of words present) to ensure that the user never overflows the FIFO. The user can specify the width of the write data count bus with a maximum width of \log_2 (write depth). If the width specified is smaller than the maximum allowable width, the bus is truncated with the lower bits removed.

For example, you can only use two bits out of a maximum allowable three bits (provided a FIFO depth of eight). These two bits indicate the number of words in the FIFO, with a quarter resolution. This provides a status of the contents of the FIFO for the write clock domain.

Note: If a write operation occurs on a rising clock edge of WR_CLK, that write will be reflected on the WR_DATA_COUNT signal following the next rising clock edge. A read operation, which occurs on the RD_CLK clock domain, may take a number of clock cycles before being reflected in the WR_DATA_COUNT.

First-Word Fall-Through Data Count

By providing the capability to read the next data word before requesting it, FWFT implementations increase the depth of the FIFO. This depth increase causes the FWFT data counts to be estimates of the number of words in the FIFO, not exact representations of the number of words in the FIFO. By selecting “Use extra logic for more accurate Data Counts” on page 4 of the CORE Generator GUI, an extra bit is added to WR_DATA_COUNT and RD_DATA_COUNT to allow them to accurately express the full depth of the FIFO. When this option is selected, RD_DATA_COUNT and WR_DATA_COUNT behave as described in the sections above. When this option is not selected, the write and read data counts are only estimates of the contents of the FIFO.

For example, an independent clocks FIFO with a user-selected input and output depth of eight, has an actual depth (as reported on the summary page of the GUI) of seven. If FWFT is selected, the additional register stages increase the FIFO’s effective depth to nine. In this case, selecting “use extra logic for more accurate data counts” will increase the data count counter sizes to display the full range of the FIFO (zero to nine words).

Note: For FWFT implementations, WR_DATA_COUNT is guaranteed to be accurate when words are present in the FIFO, but may be incorrect by 2 at or near empty.

Example Operation

Figure 5-10 shows write and read data counts. When WR_EN is asserted and FULL is deasserted, WR_DATA_COUNT increments. Similarly, when RD_EN is asserted and EMPTY is deasserted, RD_DATA_COUNT decrements.

Note: In the first part of Figure 5-10, a successful write operation occurs on the third rising clock edge, and is not reflected on WR_DATA_COUNT until the next full clock cycle is complete. Similarly, RD_DATA_COUNT transitions one full clock cycle after a successful read operation.

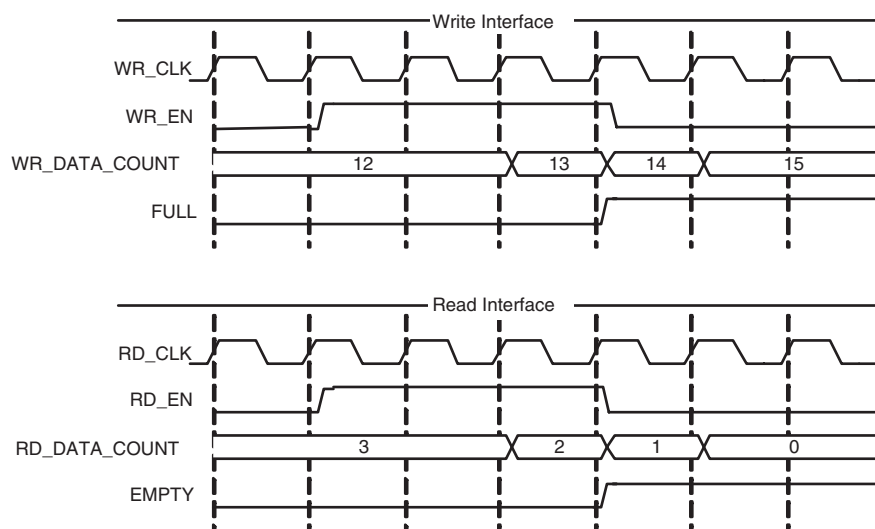


Figure 5-10: Write and Read Data Counts for FIFO with Independent Clocks

Non-symmetric Aspect Ratios

This feature is supported for FIFOs configured with independent clocks implemented with block RAM. Non-symmetric aspect ratios allow the input and output depths of the FIFO to be different. The following write-to-read aspect ratios are supported: 1:8, 1:4, 1:2, 1:1, 2:1, 4:1, 8:1. This feature is enabled by selecting unique write and read widths when customizing the FIFO using the CORE Generator. By default, the write and read widths are set to the same value (providing a 1:1 aspect ratio); but any ratio between 1:8 to 8:1 is supported, and the output depth of the FIFO is automatically calculated from the input depth and the write and read widths.

For non-symmetric aspect ratios, the full and empty flags are active only when one complete word can be written or read. The FIFO does not allow partial words to be accessed. For example, assuming a full FIFO, if the write width is 8 bits and read width is 2 bits, the user would have to complete four valid read operations before full deasserts and a write operation accepted. Write data count shows the number of FIFO words according to the write port ratio, and read data count shows the number of FIFO words according to the read port ratio.

Note: For non-symmetric aspect ratios where the write width is smaller than the read width (1:8, 1:4, 1:2), the most significant bits are read first (refer to Figure 5-11 and Figure 5-12).

Figure 5-11 is an example of a FIFO with a 1:4 aspect ratio (write width = 2, read width = 8). In this figure, four consecutive write operations are performed before a read operation can be performed. The first write operation is "10", followed by "11", "00", and finally "01."

The memory is filling up from the right to the left (LSB to MSB). When a read operation is performed, the received data is "01_00_11_10."

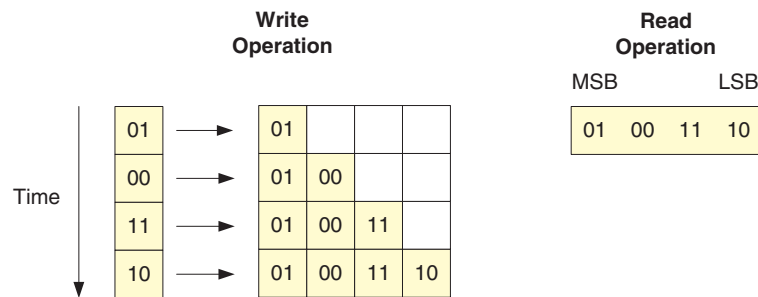


Figure 5-11: 1:4 Aspect Ratio: Data Ordering

Figure 5-12 shows DIN, DOUT and the handshaking signals for a FIFO with a 1:4 aspect ratio. After four words are written into the FIFO, EMPTY is deasserted. Then after a single read operation, EMPTY is asserted again.

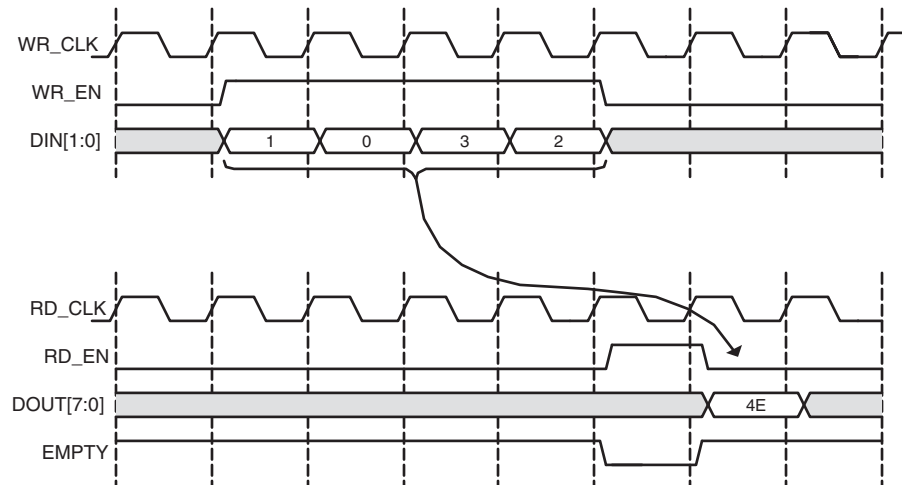


Figure 5-12: 1:4 Aspect Ratio: Status Flag Behavior

Figure 5-13 shows a FIFO with an aspect ratio of 4:1 (write width of 8, read width of 2). In this example, a single write operation is performed, after which four read operations are executed. The write operation is "11_00_01_11." When a read operation is performed, the data is received left to right (MSB to LSB). As shown, the first read results in data of "11", followed by "00", "01", and then "11."

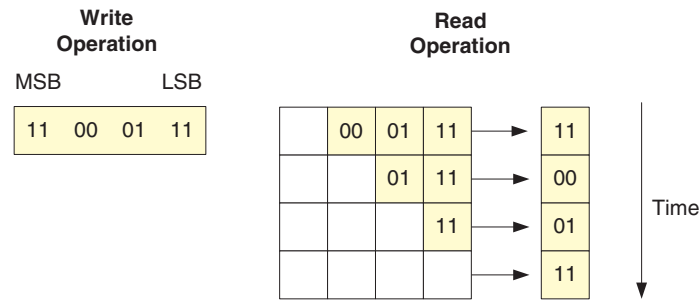


Figure 5-13: 4:1 Aspect Ratio: Data Ordering

Figure 5-14 shows DIN, DOUT, and the handshaking signals for a FIFO with an aspect ratio of 4:1. After a single write, the FIFO deasserts EMPTY. Since no other writes occur, the FIFO reasserts empty after four reads.

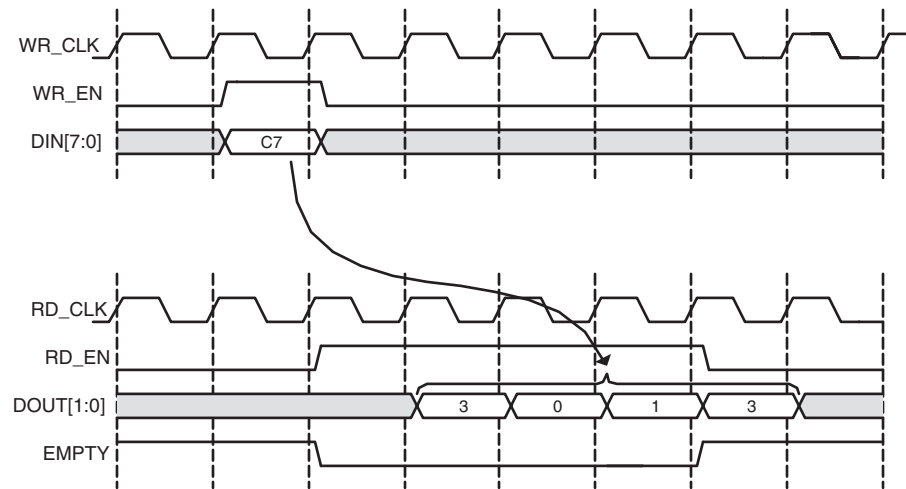


Figure 5-14: 4:1 Aspect Ratio: Status Flag Behavior

Reset Behavior

The FIFO Generator provides a single reset (RST) input that asynchronously resets all counters, output registers, and memories when asserted. For block RAM or distributed RAM implementations, resetting the FIFO is not required, and the input pin (RST) can be disabled in the FIFO. When reset is implemented, it is synchronized internally to the core with each respective clock domain for setting the internal logic of the FIFO to a known state. This synchronization logic allows for proper timing of the reset logic within the core to avoid glitches and metastable behavior. Due to the synchronization logic used, there is a

latency in the deassertion of the FULL, ALMOST_FULL, and PROG_FULL signals. Figure 5-15 shows the full flags following the release of RST.

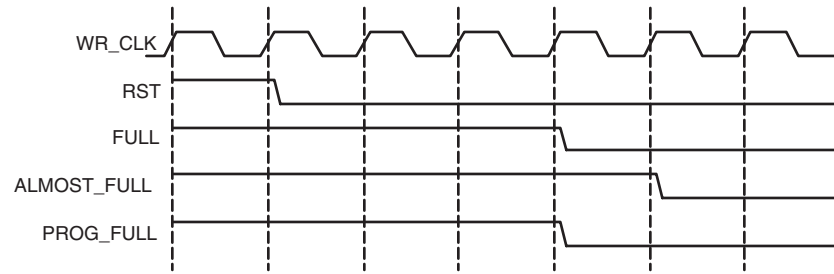


Figure 5-15: Reset Behavior for FIFO with Independent Clocks

Table 5-2 lists the values of the output ports during the reset state. DOUT reset value is supported for distributed RAM and block RAM implementations (excluding Virtex and Spartan-II). If the user does not specify a DOUT reset value, it defaults to 0. FULL and ALMOST_FULL flags are asserted during reset to ensure that no write operations occur, and handshaking signals are deasserted during reset.

The FIFO requires a reset pulse of at least 3 clock cycles in length.

Table 5-2: Independent Clock FIFO Reset Values

Signal	Value at Reset
DOUT	DOUT Reset Value or 0 ¹
FULL	1 ²
ALMOST_FULL	1 ²
EMPTY	1
VALID	0 (active high) or 1 (active low)
UNDERFLOW	0 (active high) or 1 (active low)
WR_ACK	0 (active high) or 1 (active low)
OVERFLOW	0 (active high) or 1 (active low)
PROG_FULL	1 ²
PROG_EMPTY	1
RD_DATA_COUNT	0
WR_DATA_COUNT	0

1. The ability to set DOUT to a user-defined value is not available for block RAM implementations in Virtex, Spartan-II and Spartan-IIE. DOUT resets to 0 when this feature is unavailable.
2. When reset is asserted, the FULL flags are asserted to prevent writes to the FIFO during reset.

Independent Clocks: Built-in FIFO

Figure 5-16 illustrates the functional implementation of FIFO configured with independent clocks using the Virtex-4 built-in FIFO primitive. This design implementation

consists of cascaded built-in FIFO primitives and handshaking logic. The number of built-in primitives depends on the FIFO width and depth requested.

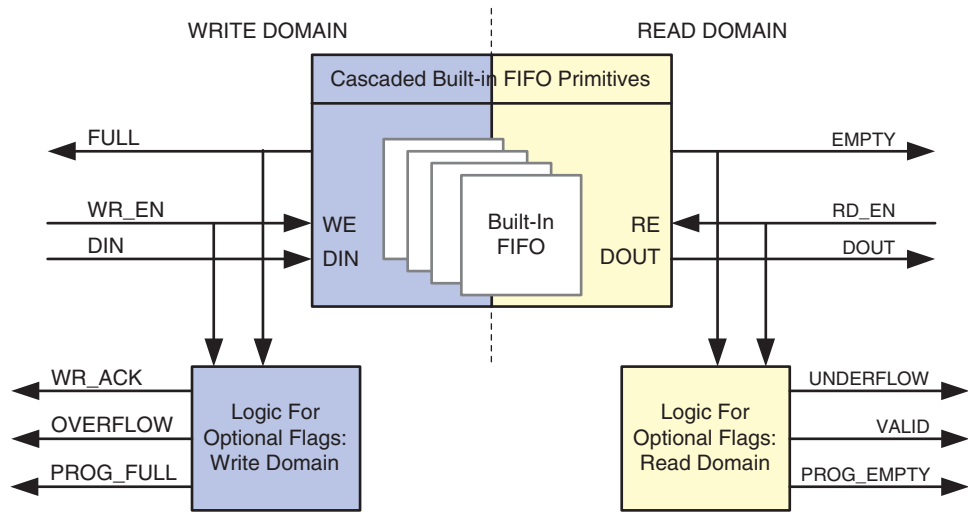


Figure 5-16: Functional Implementation of Built-in FIFO

This FIFO is designed to support an independent read clock (RD_CLK) and write clock (WR_CLK); in other words, there is no required relationship between RD_CLK and WR_CLK with regard to frequency or phase. The FIFO interface signals are only valid in their respective clock domains, and are summarized in Table 5-3.

Table 5-3: Interface Signals and Corresponding Clock Domains

WR_CLK	RD_CLK
DIN	DOUT
WR_EN	RD_EN
FULL	EMPTY
PROG_FULL	PROG_EMPTY
WR_ACK	VALID
OVERFLOW	UNDERFLOW

For FIFO cores using independent clocks, the timing relationship between the write and read operations and the status flags is affected by the relationship of the two clocks. For example, the timing between writing to an empty FIFO and the deassertion of EMPTY is determined by the phase and frequency relationship between the write and read clocks. For additional information refer to the “Synchronization Considerations,” page 66.

Write Operation

This section describes the behavior of a FIFO write operation and the associated status flags. When write enable is asserted and the FIFO is not full, data is added to the FIFO from the input bus (DIN) and write acknowledge (WR_ACK) is asserted. If the FIFO is continuously written to without being read, it fills with data. Write operations are only successful when the FIFO is not full. When the FIFO is full and a write is initiated, the

request is ignored, the overflow flag is asserted and there is no change in the state of the FIFO (overflowing the FIFO is non-destructive).

FULL Flag

FULL indicates that the FIFO is full and no more writes can be performed until data is read out. This flag is active high and synchronous to the write clock (WR_CLK).

Examples of Operation

Typical Case

Figure 5-17 shows a typical write operation. The user-asserted **WR_EN**, causes a write operation to occur on the next rising edge of the **WR_CLK**. Since the FIFO is not full, **WR_ACK** is asserted, acknowledging a successful write operation. When the FIFO is full, **FULL** is asserted and all further write requests are ignored. When a write occurs after **FULL** is asserted, **WR_ACK** is deasserted and **OVERFLOW** is asserted, indicating an overflow condition. Once the user performs one or more read operations, the FIFO deasserts **FULL**, and data can successfully be written into the FIFO, as is indicated by the assertion of **WR_ACK** and deassertion of **OVERFLOW**.

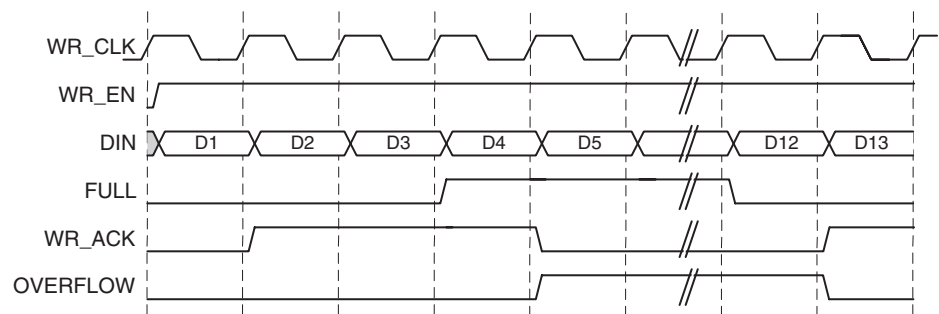


Figure 5-17: Write Operation for a FIFO with Independent Clocks

Using the Built-in FIFO Flags

Figure 5-18 shows a Use Built-in FIFO Flags operation. The FIFO Generator core uses the **FULL** and **OVERFLOW** flags directly from the built-in FIFO primitives. The behavior of the **FULL** flag on the primitives differs from the typical **FULL** behavior of the FIFO generated. As illustrated, data word **D2** is the last data word successfully written into the FIFO. Immediately following the rising edge to which the last write occurred, **WR_ACK** is asserted, and **OVERFLOW** is deasserted. While technically full, **FULL** is not yet asserted, due to the delay on that signal. On the next rising clock edge, **FULL** is asserted as a result of the

final write during the previous clock cycle. In addition, the attempted write of data word D3 will fail, resulting in the deassertion of `WR_ACK` and the assertion of `OVERFLOW`.

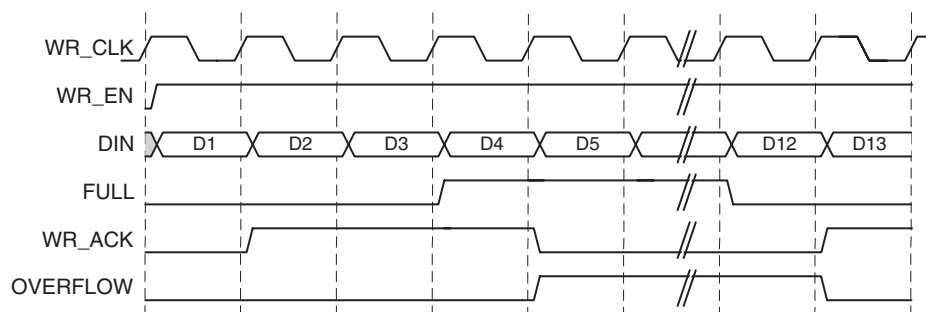


Figure 5-18: Write Operation for a FIFO Using Built-in Flags

Read Operation

This section describes the behavior of a FIFO read operation and the associated status flags. When read enable is asserted and the FIFO is not empty, data is read from the FIFO on the output bus (DOUT), and `VALID` is asserted. If the FIFO is continuously read without being written, the FIFO empties. Read operations are successful when the FIFO is not empty. When the FIFO is empty and a read is requested, the read operation is ignored, the underflow flag is asserted and there is no change in the state of the FIFO (underflowing the FIFO is non-destructive).

EMPTY Flag

`EMPTY` indicates that the FIFO is empty and no more reads can be performed until data is written into the FIFO. This flag is active high and synchronous to the read clock (`RD_CLK`). If a read is initiated when `EMPTY` is asserted, the request is ignored and `UNDERFLOW` is asserted.

Example Operation

Figure 5-19 shows a standard read access. Whenever at least one word is written into the FIFO, `EMPTY` is deasserted, indicating data is available to be read. The user asserts `RD_EN`, causing a read operation to occur on the next rising edge of `RD_CLK`. The FIFO outputs the next available word on DOUT and asserts `VALID`, indicating a successful read operation. When the last data word is read from the FIFO, the FIFO asserts `EMPTY`. If the user continues to assert `RD_EN` while `EMPTY` is asserted, the read request is ignored, `VALID` is deasserted, and `UNDERFLOW` is asserted. Once the user performs a write operation, the

FIFO deasserts `EMPTY`, allowing the user to resume valid read operations, as indicated by the assertion of `VALID` and deassertion of `UNDERFLOW`.

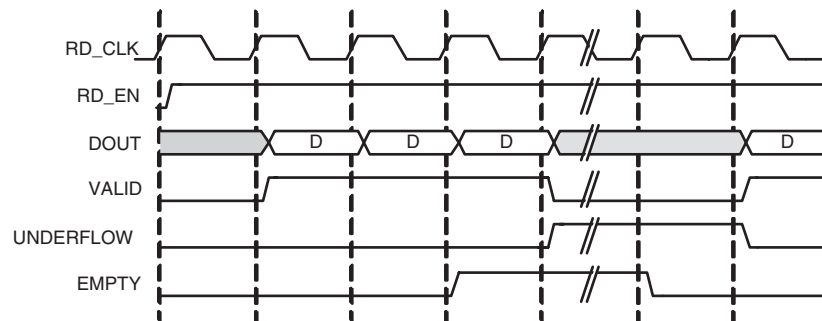


Figure 5-19: Read Operation for a FIFO with Independent Clocks

Handshaking Flags

Handshaking flags (valid, underflow, write acknowledge and overflow) are supported to provide additional information regarding the status of the write and read operations. The handshaking flags are optional, and can be configured as active high or active low through the CORE Generator GUI (see [“Optional Flags, Handshaking, and Initialization,”](#) page 35 for more information).

Valid

`VALID` is asserted at the rising edge of `RD_CLK` for each successful read operation, and indicates that the data on the `DOUT` bus is valid. When a read request is unsuccessful (when the FIFO is empty), `VALID` is not asserted. This flag is synchronous to the read clock (`RD_CLK`).

Underflow

`UNDERFLOW` indicates that a read operation is unsuccessful. This occurs when a read is initiated and the FIFO is empty. This flag is synchronous with the read clock (`RD_CLK`). Underflowing the FIFO does not change the state of the FIFO (it is non-destructive).

Write Acknowledge

`WR_ACK` is asserted at the completion of each successful write operation and indicates that the data on the `DIN` port has been stored in the FIFO. This flag is synchronous with the write clock (`WR_CLK`).

Overflow

`OVERFLOW` is used to indicate that a write operation is unsuccessful. This flag is asserted when a write is initiated to the FIFO while `FULL` is asserted. The overflow flag is synchronous to the write clock (`WR_CLK`). Overflowing the FIFO does not change the state of the FIFO (it is non-destructive).

Example Operation

Figure 5-20 shows the handshaking flags. On the write interface, `FULL` is not asserted and writes to the FIFO are successful (as indicated by the assertion of `WR_ACK`). When a write occurs after `FULL` is asserted, `WR_ACK` is deasserted and `OVERFLOW` is asserted, indicating

an overflow condition. On the read interface, once the FIFO is not `EMPTY`, the FIFO accepts read requests. Following a read request, `VALID` is asserted and `DOUT` is updated on the next clock cycle. When a read request is issued while `EMPTY` is asserted, `VALID` is deasserted and `UNDERFLOW` is asserted, indicating an underflow condition.

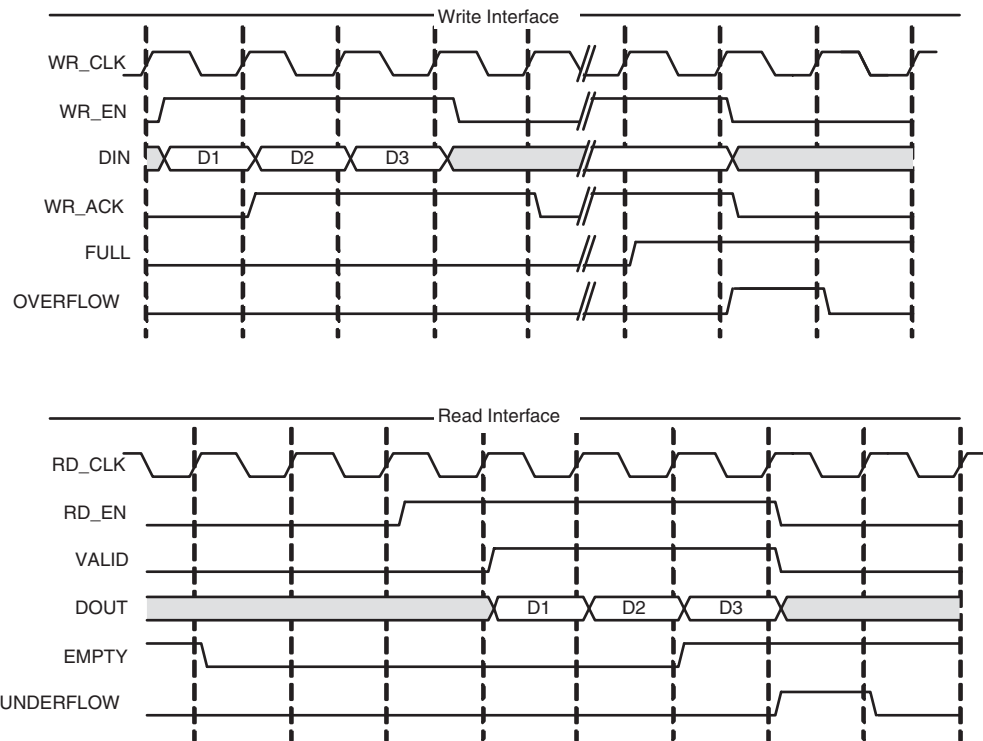


Figure 5-20: Handshaking Signals for a FIFO with Independent Clocks

Programmable Flags

The FIFO supports status flags to indicate that the FIFO has reached a user-defined fill level.

- Programmable full (`PROG_FULL`) indicates that the FIFO has reached a user-defined full threshold.
- Programmable empty (`PROG_EMPTY`) indicates that the FIFO has reached a user-defined empty threshold.

For these thresholds, you can set a constant value or choose to have dedicated input ports, enabling the thresholds to change dynamically in circuit. Hysteresis is also optionally supported, by providing unique assert and negate values for each flag. Detailed information about these options are provided in the following sections.

Programmable Full

The FIFO Generator supports four ways to define the programmable full threshold.

- Single threshold constant
- Single threshold with dedicated input port
- Assert and negate threshold constants (provides hysteresis)
- Assert and negate thresholds with dedicated input ports (provides hysteresis)

These options are available in the CORE Generator GUI and accessed within the programmable flags window (Figure 4-4, page 37).

The programmable full flag (PROG_FULL) is asserted when the number of entries in the FIFO is greater than or equal to the user-defined assert threshold. When the programmable full flag is asserted, the FIFO can continue to be written to until the full flag (FULL) is asserted. If the number of words in the FIFO is less than the negate threshold, the flag is deasserted.

Note: If a write operation occurs on a rising clock edge that causes the number of words to meet or exceed the programmable full threshold, then the programmable full flag will assert on the next rising clock edge. The deassertion of the programmable full flag has a longer delay, and depends on the relationship between the write and read clocks.

Programmable Full: Single Threshold

This option enables the user to set a single threshold value for the assertion and deassertion of PROG_FULL. When the number of entries in the FIFO is greater than or equal to the threshold value, PROG_FULL is asserted. When the number of entries in the FIFO is less than the threshold value, PROG_FULL is deasserted.

There are two options for implementing this threshold.

- **Single threshold constant:** The threshold value is specified through the CORE Generator GUI. Once the core is generated, this value can only be changed by re-generating the core. This option consumes fewer resources than the single threshold with dedicated input port.
- **Single threshold with dedicated input port:** The threshold value is specified through an input port (PROG_FULL_THRESH) on the core. This input can be changed while the FIFO is in reset, providing the user the flexibility to change the programmable full threshold in-circuit without re-generating the core.

Note: Refer to the CORE Generator GUI for valid ranges for each threshold.

The behavior of the programmable full flag with a single threshold is illustrated in Figure 5-21. The programmable full threshold is set to seven, five words are present in the FIFO, and PROG_FULL is not asserted. The user initiates two write operations, resulting in seven words present in the FIFO. Since the programmable full threshold is set to seven, PROG_FULL is asserted. Note that PROG_FULL has two clock cycles of delay. PROG_FULL will remain asserted until six words (or fewer) are present in the FIFO.

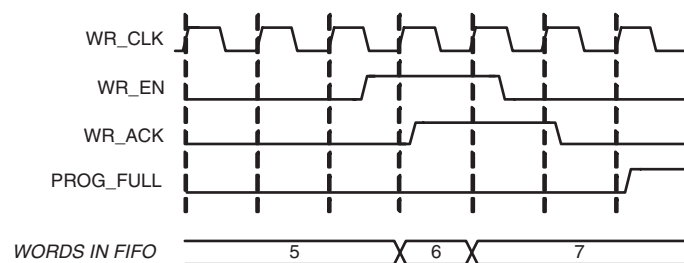


Figure 5-21: Programmable Full Single Threshold: Threshold Set to 7

If “Use Built-in FIFO Flags” is used with a single constant programmable-full threshold, the FIFO Generator core uses the PROG_FULL flag directly from the built-in FIFO primitive, producing no additional logic for implementing this flag.

Programmable Full: Assert and Negate Thresholds

This option enables the user to set separate values for the assertion and deassertion of `PROG_FULL`. When the number of entries in the FIFO is greater than or equal to the assert value, `PROG_FULL` is asserted. When the number of entries in the FIFO is less than the negate value, `PROG_FULL` is deasserted.

There are two options for implementing these thresholds:

- **Assert and negate threshold constants.** Specify the threshold values through the CORE Generator GUI. Once the core is generated, these values can only be changed by re-generating the core. This option consumes fewer resources than the assert and negate thresholds with dedicated input ports.
- **Assert and negate thresholds with dedicated input ports.** Specify the threshold values through input ports on the core. These input ports can be changed while the FIFO is in reset, providing you with the flexibility to change the values of the programmable full assert (`PROG_FULL_THRESH_ASSERT`) and negate (`PROG_FULL_THRESH_NEGATE`) thresholds in-circuit without re-generating the core.

Note: The full assert value must be larger than the full negate value. Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-22 shows the programmable full flag with assert and negate thresholds. The programmable full assert threshold is set to seven, and the negate threshold is set to six. Initially five words are present in the FIFO, and `PROG_FULL` is deasserted. The user initiates two write operations, resulting in seven words present in the FIFO. Since the programmable full assert threshold is set to seven, `PROG_FULL` is asserted. The `PROG_FULL` has two clock cycles of delay. `PROG_FULL` remains asserted until five words (or fewer) are present in the FIFO.

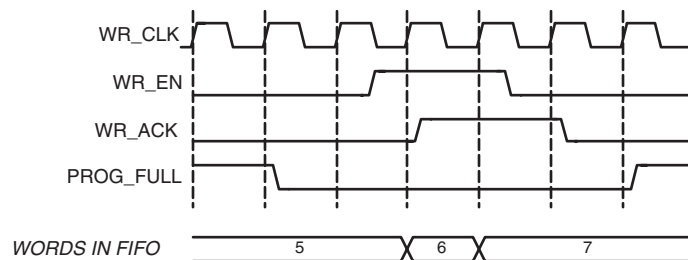


Figure 5-22: Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7

Programmable Empty

The FIFO Generator supports four ways to define the programmable empty thresholds.

- Single threshold constant
- Single threshold with dedicated input port
- Assert and negate threshold constants (provides hysteresis)
- Assert and negate thresholds with dedicated input ports (provides hysteresis)

These options are available in the CORE Generator GUI and accessed within the programmable flags window (see Figure 4-2, page 34).

The programmable empty flag (PROG_EMPTY) is asserted when the number of entries in the FIFO is less than or equal to the user-defined assert threshold. If the number of words in the FIFO is greater than the negate threshold, the flag is deasserted.

Note: If a read operation occurs on a rising clock edge that causes the number of words in the FIFO to be equal to or less than the programmable empty threshold, then the programmable empty flag will assert on the next rising clock edge. The deassertion of the programmable empty flag has a longer delay, and depends on the read and write clocks.

Programmable Empty: Single Threshold

Use this option to set a single threshold value for the assertion and deassertion of PROG_EMPTY. When the number of entries in the FIFO is less than or equal to the threshold value, PROG_EMPTY is asserted. When the number of entries in the FIFO is greater than the threshold value, PROG_EMPTY is deasserted.

There are two options for implementing this threshold:

- **Single threshold constant.** Specify the threshold value through the CORE Generator GUI. Once the core is generated, this value can only be changed by re-generating the core. This option consumes fewer resources than the single threshold with dedicated input port.
- **Single threshold with dedicated input port.** Specify the threshold value through an input port (PROG_EMPTY_THRESH) on the core. This input can be changed while the FIFO is in reset, providing the user the flexibility to change the programmable empty threshold in-circuit without re-generating the core.

Note: Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-23 shows the programmable empty flag with a single threshold. The programmable empty threshold is set to four, six words are present in the FIFO, and PROG_EMPTY is not asserted. The user initiates two read operations, resulting in four words remaining in the FIFO. Since the programmable empty threshold is set to four, PROG_EMPTY is asserted. The PROG_EMPTY has two clock cycles of delay. PROG_EMPTY will remain asserted until five words (or more) are present in the FIFO.

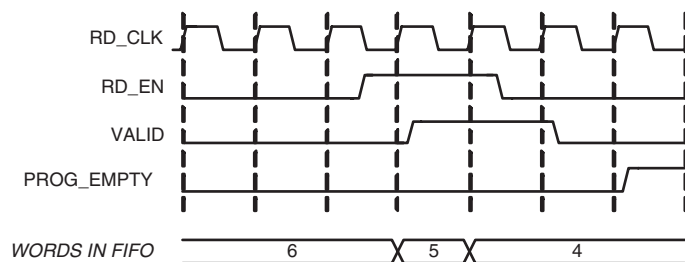


Figure 5-23: Programmable Empty with Single Threshold: Threshold Set to 4

Programmable Empty: Assert and Negate Thresholds

Use this option to set separate values for the assertion and deassertion of PROG_EMPTY. When the number of entries in the FIFO is less than or equal to the assert value, PROG_EMPTY is asserted. When the number of entries in the FIFO is greater than the negate value, PROG_EMPTY is deasserted.

There are two options for implementing these thresholds:

- **Assert and negate threshold constants.** Specify the threshold values through the CORE Generator GUI. Once the core is generated, these values can only be changed

by re-generating the core. This option consumes fewer resources than the assert and negate thresholds with dedicated input ports.

- **Assert and negate thresholds with dedicated input ports.** Specify the threshold values through input ports on the core. These input ports can be changed while the FIFO is in reset, providing the user the flexibility to change the values of the programmable empty assert (`PROG_EMPTY_THRESH_ASSERT`) and negate (`PROG_EMPTY_THRESH_NEGATE`) thresholds in-circuit without regenerating the core.

Note: The empty assert value must be less than the empty negate value. Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-24 illustrates the behavior of the programmable empty flag with assert and negate thresholds. The programmable empty assert threshold is set to four, and the negate threshold is set to six. Initially six words are present in the FIFO, and `PROG_EMPTY` is not asserted. The user initiates two read operations, resulting in four words remaining in the FIFO. Since the programmable empty assert threshold is set to four, `PROG_EMPTY` is asserted. Note that `PROG_EMPTY` has two clock cycles of delay. `PROG_EMPTY` will remain asserted until seven words (or more) are present in the FIFO.

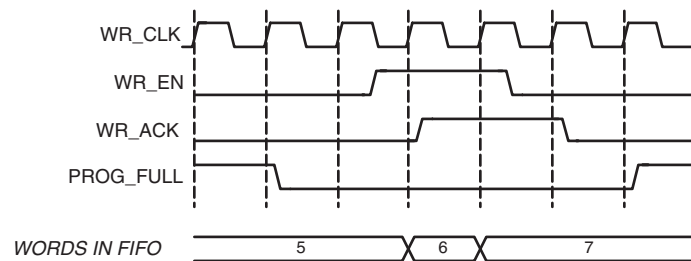


Figure 5-24: Programmable Empty with Assert and Negate Thresholds: Assert Set to 4 and Negate Set to 6

Reset Behavior

The FIFO Generator provides a single reset (`RST`) input that asynchronously resets the Virtex-4 built-in FIFO and control logic when asserted. Internal to the core, `RST` is synchronized with each respective clock domain for setting the internal logic of the FIFO to a known state. This synchronization logic allows for proper timing of the reset logic within the core to avoid glitches and metastable behavior. Due to the synchronization logic used, there is a latency in the deassertion of the `PROG_FULL` signal. Figure 5-25 illustrates the behavior of the programmable full flag following the release of `RST`.

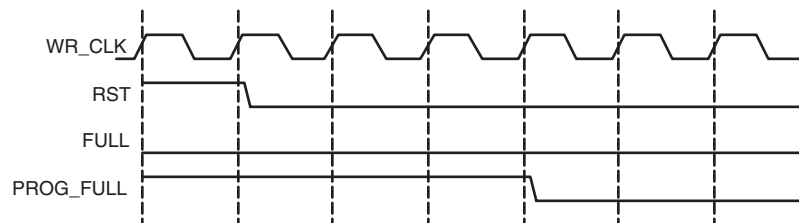


Figure 5-25: Reset Behavior for Built-In FIFO Implementations with Independent Clocks

Table 5-4 lists the values of the output ports during reset state. `DOUT` resets to the contents of memory at location 0. `FULL` and `EMPTY` flags have a value of 0 during reset, and the handshaking signals are deasserted during reset.

The FIFO requires a reset pulse of at least 3 clock cycles in length, and no read or write operations can occur until 4 clock cycles after reset is released.

Table 5-4: Built-In FIFO Reset Values

Signal	Value at Reset
DOUT	Content of memory at location 0
FULL	0
EMPTY	1
VALID	0 (active high) or 1 (active low)
UNDERFLOW	0 (active high) or 1 (active low)
WR_ACK	0 (active high) or 1 (active low)
OVERFLOW	0 (active high) or 1 (active low)
PROG_FULL	1
PROG_EMPTY	1

Synchronization Considerations

FIFOs with independent write and read clocks require that interface signals be used only in their respective clock domain. The independent clocks FIFO handles all synchronization requirements, enabling the user to cross between two clock domains that have no relationship in frequency or phase.

Figure 5-26 shows the signals with respect to their clock domains. All signals are synchronous to a specific clock, with the exception of RST, which performs an asynchronous reset of the entire FIFO.

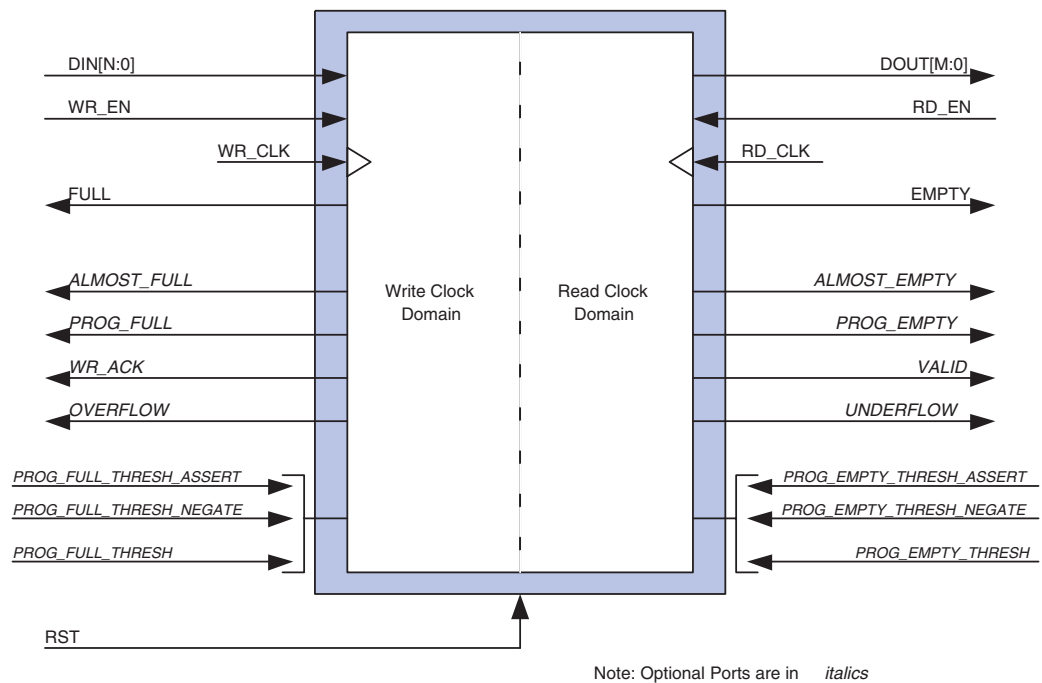


Figure 5-26: FIFO with Independent Clocks: Write and Read Clock Domains

For write operations, the write enable signal (*WR_EN*) and data input (*DIN*) are synchronous to *WR_CLK*. For read operations, the read enable (*RD_EN*) and data output (*DOUT*) are synchronous to *RD_CLK*. All status outputs are synchronous to their respective clock domains and can only be used in that clock domain. The performance of the FIFO can be measured by independently constraining the clock period for the *WR_CLK* and *RD_CLK* input signals.

The interface signals are evaluated on their rising clock edge (*WR_CLK* and *RD_CLK*). They can be made falling-edge active (relative to the clock source) by inserting an inverter between the clock source and the FIFO clock inputs. This inverter is absorbed into the internal FIFO control logic and does not cause a decrease in performance or increase in logic utilization.

Common Clock FIFO: Block RAM, Distributed RAM, Shift Register

The following sections provide an overview of the common clock FIFO.

Common Clock FIFO: Block RAM and Distributed RAM

[Figure 5-27](#) illustrates the functional implementation of a FIFO configured with a common clock using block RAM or distributed RAM for memory. All signals are synchronous to a single clock input (*CLK*). This design implements counters for write and read pointers and logic for calculating the status flags.

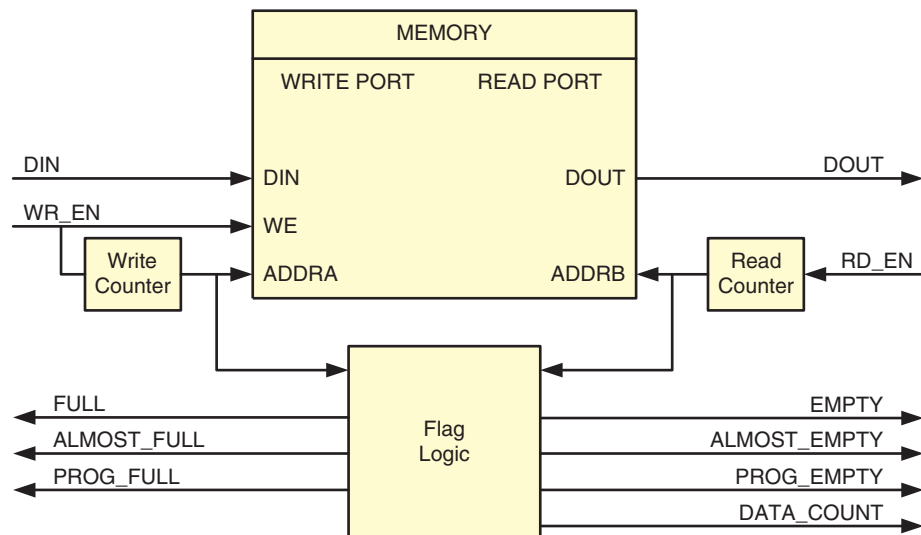


Figure 5-27: Functional Implementation of a Common Clock FIFO using Block RAM or Distributed RAM

Common Clock FIFO: Shift Registers

Figure 5-28 illustrates the functional implementation of a FIFO configured with a common clock using shift registers for memory. All operations are synchronous to the same clock input (CLK). This design implements a single up/down counter for both the write and read pointers and logic for calculating the status flags.

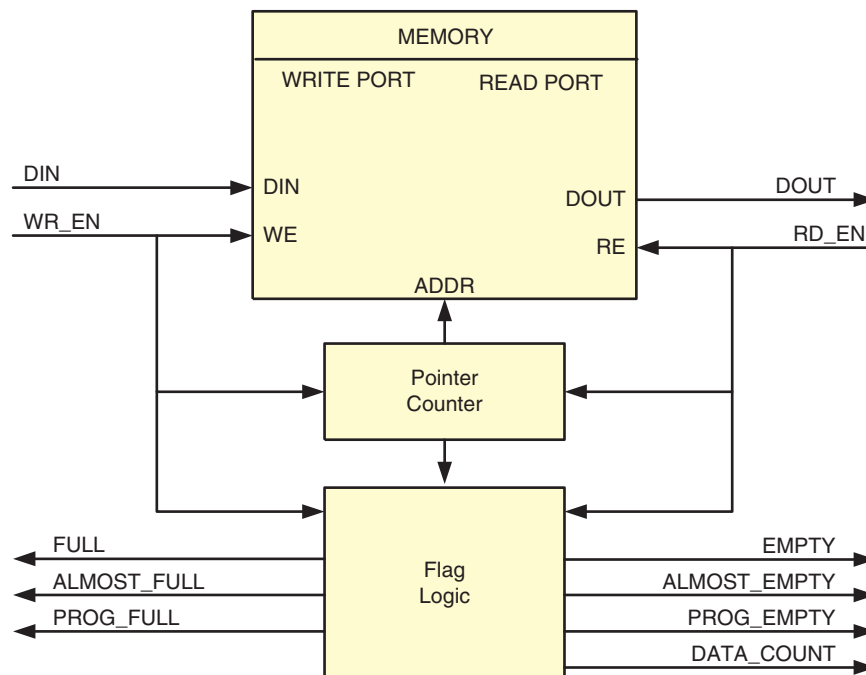


Figure 5-28: Functional Implementation of a Common Clock FIFO using Shift Registers

Write and Read Operation

This section describes the write and read behavior of a FIFO and the associated status flags.

Write Operation

When write enable is asserted and the FIFO is not full, data is added to the FIFO from the input bus (DIN) and write acknowledge (WR_ACK) is asserted. If the FIFO is continuously written without being read, it fills with data. Write operations are only successful when the FIFO is not full. When the FIFO is full and a write is requested, the write operation is ignored, the overflow flag is asserted and there is no change in the state of the FIFO (overflowing the FIFO is non-destructive).

Read Operation

When read enable is asserted and the FIFO is not empty, data is read from the FIFO on the output bus (DOUT), and the valid flag (VALID) is asserted. If the FIFO is continuously read without being written, the FIFO empties. Read operations are successful when the FIFO is not empty. When the FIFO is empty and a read is requested, the read operation is ignored, the underflow flag is asserted and there is no change in the state of the FIFO (underflowing the FIFO is non-destructive).

ALMOST_FULL and FULL Flags

The almost full flag (ALMOST_FULL) is active high and indicates that only one more write (without a simultaneous read) can be performed before FULL is asserted.

The full flag (FULL) is active high, and indicates that the FIFO is full and no more writes can be performed until data is read out. If a write is initiated when FULL is asserted, the write request is ignored and OVERFLOW is asserted.

ALMOST_EMPTY and EMPTY Flags

The almost empty flag (ALMOST_EMPTY) is active high, and indicates that the FIFO will be empty after one more read operation (without a simultaneous write). This flag is asserted when the FIFO has one remaining word that can be read.

EMPTY is active high, and indicates that the FIFO is empty and no more reads can be performed until data is written into the FIFO. If a read is initiated when EMPTY is asserted, the request is ignored and UNDERFLOW is asserted.

When write and read operations occur simultaneously while EMPTY is asserted, the write operation is accepted and the read operation is ignored. On the next clock cycle, EMPTY is deasserted and UNDERFLOW is asserted.

Example Operation

Figure 5-29 shows atypical write and read operation. A write is issued to the FIFO, resulting in the deassertion of the EMPTY flag. A simultaneous write and read is then issued, resulting in no change in the status flags. Once two or more words are present in the FIFO, the ALMOST_EMPTY flag is deasserted. Write requests are then issued to the FIFO, resulting in the assertion of ALMOST_FULL when the FIFO can only accept one more write (without a read). A simultaneous write and read is then issued, resulting in no change in the status flags. Finally one additional write without a read results in the FIFO asserting FULL, indicating no further data can be written until a read request is issued.

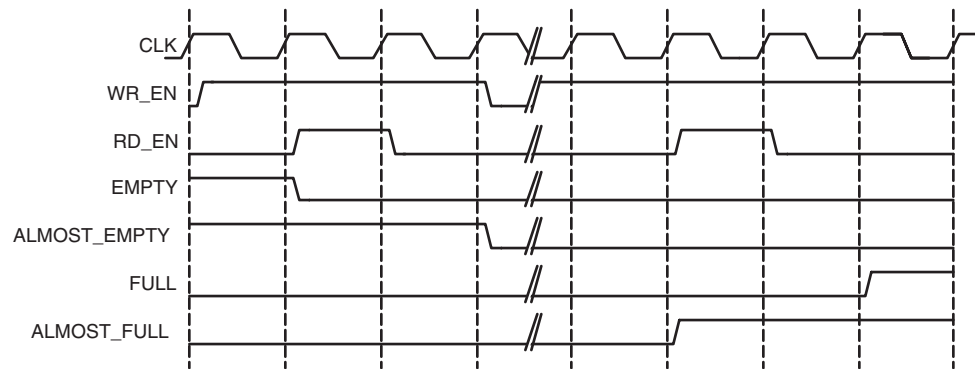


Figure 5-29: Write and Read Operation for a FIFO with Common Clocks

Handshaking Flags

Handshaking flags are supported to provide additional information regarding the status of the write and read operations. The handshaking flags (valid, underflow, write acknowledge and overflow) are optional and can be configured as active high or active low with the CORE Generator GUI (see [“Optional Flags, Handshaking, and Initialization,”](#) page 35).

Valid

VALID is asserted at the rising edge of CLK for each successful read operation, and indicates that the data on the DOUT bus is valid. When a read request is unsuccessful (when the FIFO is empty), VALID is not asserted.

Underflow

UNDERFLOW indicates an unsuccessful read operation. This occurs when a read is initiated and the FIFO is empty. Underflowing the FIFO does not change the state of the FIFO (it is non-destructive).

Write Acknowledge

The write acknowledge flag (WR_ACK) is asserted at the completion of each successful write operation and indicates that the data on the DIN port has been stored in the FIFO.

Overflow

OVERFLOW indicates that a write operation is unsuccessful. This flag is asserted when a write is initiated to the FIFO while FULL is asserted. Overflowing the FIFO does not change the state of the FIFO (it is non-destructive).

Example Operation

Figure 5-30 illustrates the Handshaking flags. On the write interface, FULL is deasserted and therefore writes to the FIFO are successful (indicated by the assertion of WR_ACK). When a write occurs after FULL is asserted, WR_ACK is deasserted and OVERFLOW is asserted, indicating an overflow condition. On the read interface, once the FIFO is not EMPTY, the FIFO accepts read requests. Following a read request, VALID is asserted and

DOUT is updated. When a read request is issued while EMPTY is asserted, VALID is deasserted and UNDERFLOW is asserted, indicating an underflow condition.

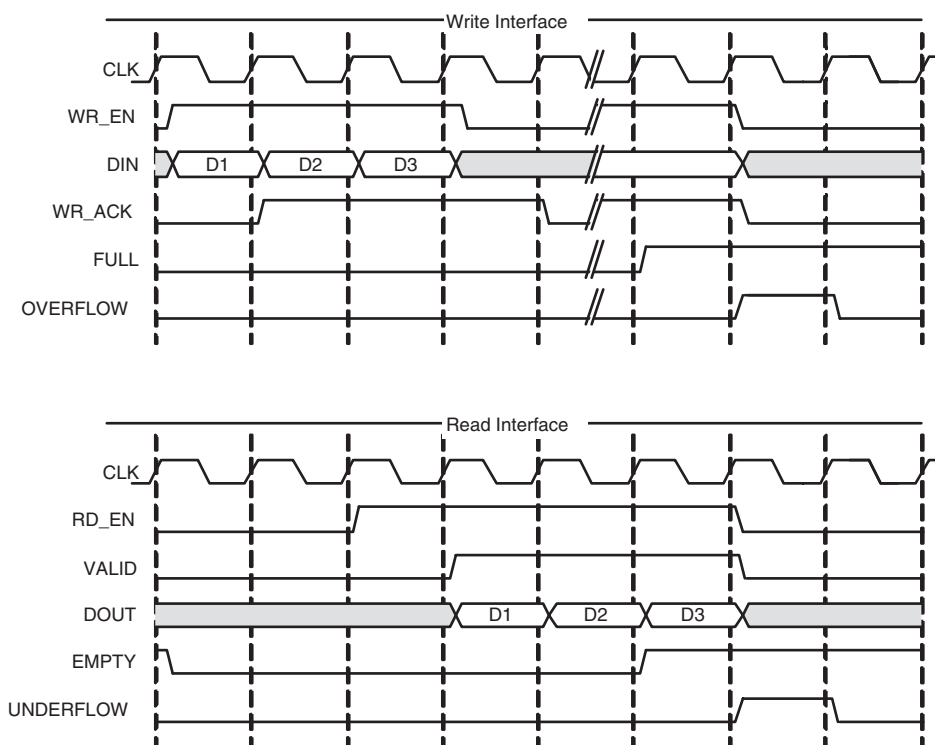


Figure 5-30: Handshaking Signals for a FIFO with Common Clocks

Programmable Flags

The FIFO supports programmable flags to indicate that the FIFO has reached a user-defined fill level.

- Programmable full (PROG_FULL) indicates that the FIFO has reached a user-defined full threshold.
- Programmable empty (PROG_EMPTY) indicates that the FIFO has reached a user-defined empty threshold.

For these thresholds, you can set a constant value or choose to have dedicated input ports, enabling the thresholds to change dynamically in circuit. Hysteresis is also optionally supported by providing unique assert and negate values for each flag. Detailed information about these options are provided in the following sections.

Programmable Full

The FIFO Generator supports four ways to define the programmable full threshold:

- Single threshold constant
- Single threshold with dedicated input port
- Assert and negate threshold constants (provides hysteresis)
- Assert and negate thresholds with dedicated input ports (provides hysteresis)

These options are available in the CORE Generator GUI and accessed within the programmable flags window (see [Figure 4-4, page 37](#)).

The programmable full flag (PROG_FULL) is asserted when the number of entries in the FIFO is greater than or equal to the user-defined assert threshold. When the programmable full flag is asserted, the FIFO can continue to be written to until the full flag (FULL) is asserted. If the number of words in the FIFO is less than the negate threshold, the flag is deasserted.

Note: If a write operation occurs on a rising clock edge that causes the number of words to meet or exceed the programmable full threshold, then the programmable full flag will assert on the next rising clock edge. The deassertion of the programmable full flag has a longer delay, and depends on the relationship between the write and read clocks.

Programmable Full: Single Threshold

Use this option to set a single threshold value for the assertion and deassertion of PROG_FULL. When the number of entries in the FIFO is greater than or equal to the threshold value, PROG_FULL is asserted. When the number of entries in the FIFO is less than the threshold value, PROG_FULL is deasserted.

There are two options for implementing this threshold.

- **Single threshold constant.** Specify the threshold value through the CORE Generator GUI. Once the core is generated, this value can only be changed by re-generating the core. This option consumes fewer resources than the single threshold with dedicated input port.
- **Single threshold with dedicated input port.** Specify the threshold value through an input port (PROG_FULL_THRESH) on the core. This input can be changed while the FIFO is in reset, enabling the user the flexibility to change the programmable full threshold in-circuit.

Note: Refer to the CORE Generator GUI for valid ranges for each threshold.

[Figure 5-31](#) depicts the programmable full flag with a single threshold. The user writes to the FIFO until there are seven words in the FIFO. Since the programmable-full threshold is set to seven, the FIFO asserts PROG_FULL once seven words are written into the FIFO. Note that PROG_FULL has one clock cycle of delay. Once the FIFO has six or fewer words in the FIFO, PROG_FULL is deasserted.

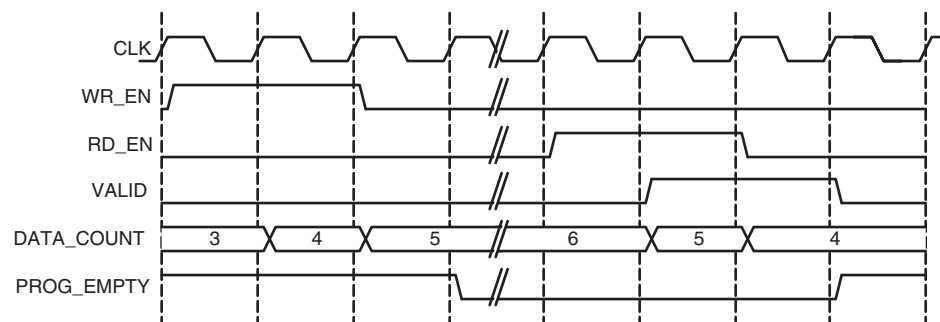


Figure 5-31: Programmable Full Single Threshold: Threshold Set to 7

Programmable Full: Assert and Negate Thresholds

Use this option to set separate values for the assertion and deassertion of PROG_FULL. When the number of entries in the FIFO is greater than or equal to the assert value, PROG_FULL is asserted. When the number of entries in the FIFO is less than the negate value, PROG_FULL is deasserted.

There are two options for implementing these thresholds.

- **Assert and negate threshold constants.** Specify the threshold values through the CORE Generator GUI. Once the core is generated, these values can only be changed by re-generating the core. This option consumes fewer resources than the assert and negate thresholds with dedicated input ports.
- **Assert and negate thresholds with dedicated input ports.** Specify the threshold values through input pins on the core. These input pins can be changed while the FIFO is in reset, enabling the user the flexibility to change the values of the programmable full assert (`PROG_FULL_THRESH_ASSERT`) and negate (`PROG_FULL_THRESH_NEGATE`) thresholds in-circuit without re-generating the core.

Note: The full assert value must be larger than the full negate value. Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-32 shows the programmable full flag with assert and negate thresholds. The user writes to the FIFO until there are 10 words in the FIFO. Since the assert threshold is set to 10, the FIFO asserts `PROG_FULL` once 10 words are in the FIFO. The negate threshold is set to seven, and the FIFO deasserts `PROG_FULL` once six words or fewer are in the FIFO. Note that `PROG_FULL` has one clock cycle of delay.

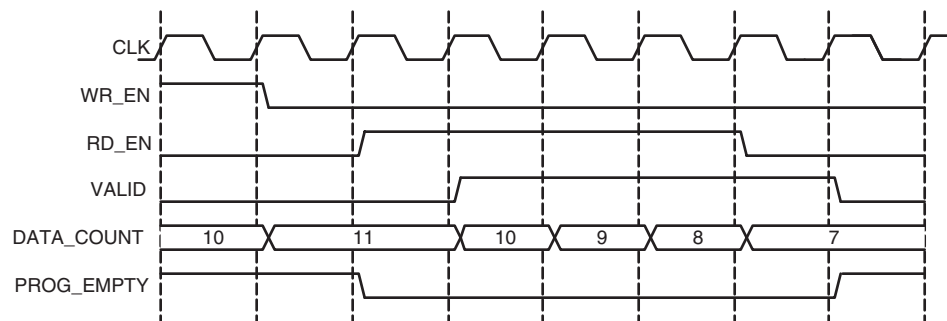


Figure 5-32: Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7

Programmable Empty

The FIFO Generator supports four ways to define the programmable empty thresholds:

- Single threshold constant
- Single threshold with dedicated input port
- Assert and negate threshold constants (provides hysteresis)
- Assert and negate thresholds with dedicated input ports (provides hysteresis)

These options are available in the CORE Generator GUI and accessed within the programmable flags window (see Figure 4-4, page 37).

The programmable empty flag (`PROG_EMPTY`) is asserted when the number of entries in the FIFO is less than or equal to the user-defined assert threshold. If the number of words in the FIFO is greater than the negate threshold, the flag is deasserted.

Note: If a read operation occurs on a rising clock edge that causes the number of words in the FIFO to be equal to or less than the programmable empty threshold, then the programmable empty flag will assert on the next rising clock edge. The deassertion of the programmable empty flag has a longer delay, and depends on the read and write clocks.

Programmable Empty: Single Threshold

Use this option to set a single threshold value for the assertion and deassertion of `PROG_EMPTY`. When the number of entries in the FIFO is less than or equal to the threshold value, `PROG_EMPTY` is asserted. When the number of entries in the FIFO is greater than the threshold value, `PROG_EMPTY` is deasserted.

There are two options for implementing this threshold.

- **Single threshold constant.** Specify the threshold value through the CORE Generator GUI. Once the core is generated, this value can only be changed by regenerating the core. This option consumes fewer resources than the single threshold with dedicated input port.
- **Single threshold with dedicated input port.** Specify the threshold value through an input port (`PROG_EMPTY_THRESH`) on the core. This input can be changed while the FIFO is in reset, enabling the user the flexibility to change the programmable empty threshold in-circuit without regenerating the core.

Note: Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-33 shows the programmable empty flag with a single threshold. The user writes to the FIFO until there are five words in the FIFO. Since the programmable empty threshold is set to four, `PROG_EMPTY` is asserted until more than four words are present in the FIFO. Once five words (or more) are present in the FIFO, `PROG_EMPTY` is deasserted. Note that `PROG_EMPTY` has one clock cycle of delay.

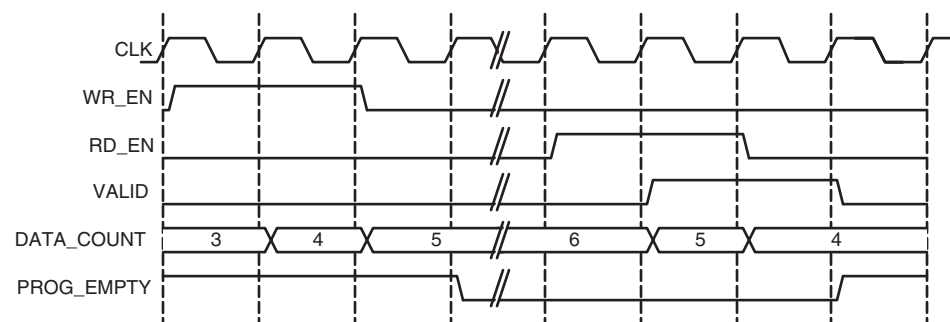


Figure 5-33: Programmable Empty with Single Threshold: Threshold Set to 4

Programmable Empty: Assert and Negate Thresholds

This option enables you to set separate values for the assertion and deassertion of `PROG_EMPTY`. When the number of entries in the FIFO is less than or equal to the assert value, `PROG_EMPTY` is asserted. When the number of entries in the FIFO is greater than the negate value, `PROG_EMPTY` is deasserted.

There are two options for implementing these thresholds.

- **Assert and negate threshold constants.** Specify the threshold values through the CORE Generator GUI. Once the core is generated, these values can only be changed by re-generating the core. This option consumes fewer resources than the assert and negate thresholds with dedicated input ports.
- **Assert and negate thresholds with dedicated input ports.** Specify the threshold values through input ports on the core. These input ports can be changed while the FIFO is in reset, enabling the user the flexibility to change the values of the programmable empty assert (`PROG_EMPTY_THRESH_ASSERT`) and negate

(`PROG_EMPTY_THRESH_NEGATE`) thresholds in-circuit without re-generating the core.

Note: The empty assert value must be less than the empty negate value. Refer to the CORE Generator GUI for valid ranges for each threshold.

Figure 5-34 illustrates the programmable empty flag with assert and negate thresholds. The user writes to the FIFO until there are 11 words in the FIFO. Since the programmable empty deassert value is set to 10, `PROG_EMPTY` is deasserted when there are more than 10 words in the FIFO. Once the FIFO contains less than or equal to the programmable empty assert value (set to seven), `PROG_EMPTY` is asserted. Note that `PROG_EMPTY` has one clock cycle of delay.

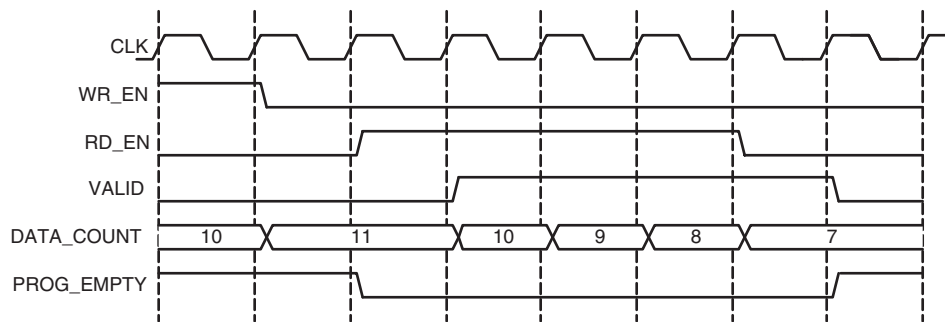


Figure 5-34: Programmable Empty with Assert and Negate Thresholds: Assert Set to 7 and Negate Set to 10

Data Count

`DATA_COUNT` tracks the number of words in the FIFO. You can specify the width of the data count bus with a maximum width of \log_2 (FIFO depth). If the width specified is smaller than the maximum allowable width, the bus is truncated by removing the lower bits.

Data count increases with every FIFO write and decreases with every FIFO read. During simultaneous write and read operations, data count also does not change values. Data count does not change when the full flag or empty flag is asserted. Figure 5-35 illustrates the behavior of data count. Note that when the FIFO is full or empty, the data count value is zero. Concatenating `FULL` with `DATA_COUNT` will create a count value that will span the entire range of the FIFO.

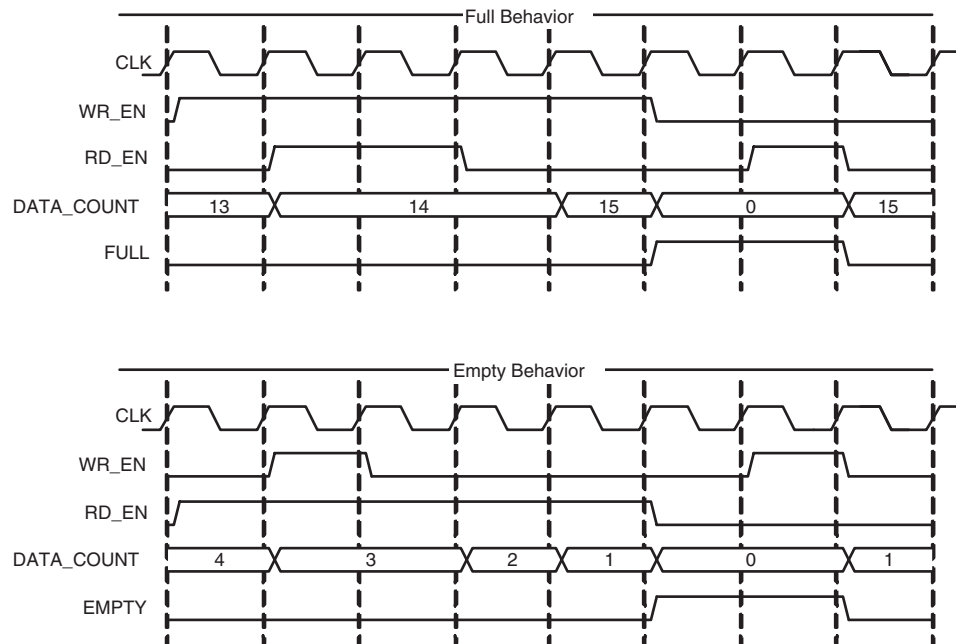


Figure 5-35: Write and Read Data Counts for FIFO with Common Clocks

Reset Behavior

The FIFO Generator has a single reset (RST) input that asynchronously resets all counters, output registers, and memories when asserted. For block RAM or distributed RAM implementations, resetting the FIFO is not required, and the input pin (RST) can be disabled in the GUI. When reset is implemented, it is synchronized internally to the core with the common clock for setting the internal logic of the FIFO to a known state. This synchronization logic allows for proper timing of the reset logic within the FIFO Generator, avoiding glitches and metastable behavior. Due to the synchronization logic used, there is a latency in the deassertion of the FULL, ALMOST_FULL, and PROG_FULL signals.

Figure 5-36 shows the full flags following the release of RST.

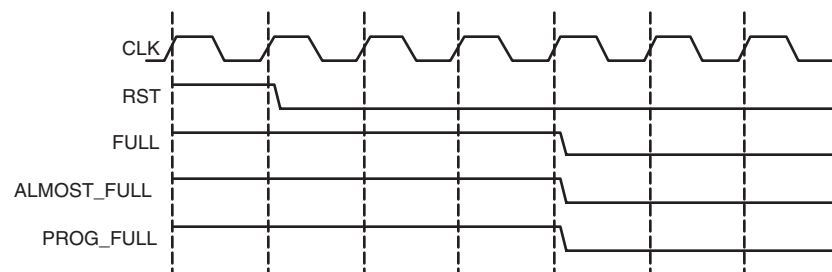


Figure 5-36: FIFO Reset Behavior for FIFO with Common Clocks

The values of the output ports during the reset state are defined in Table 5-5. A user defined value for DOUT at reset is supported for distributed RAM and block RAM implementations. If the user does not specify a DOUT reset value, it defaults to 0. FULL and ALMOST_FULL flags are asserted during reset to ensure that no write operations occur, and handshaking signals are also deasserted.

The FIFO requires a reset pulse of at least three clock cycles in length, and no read or write operations can occur until four clock cycles after reset is released.

Table 5-5: Common Clock FIFO Reset Values

Signal	Value at Reset
DOUT	DOUT Reset Value or 0 ¹
FULL	1 ²
ALMOST_FULL	1 ²
EMPTY	1
VALID	0 (active high) or 1 (active low)
UNDERFLOW	0 (active high) or 1 (active low)
WR_ACK	0 (active high) or 1 (active low)
OVERFLOW	0 (active high) or 1 (active low)
PROG_FULL	1 ²
PROG_EMPTY	1
DATA_COUNT	0

1. The ability to set DOUT to a user-defined value is not available for block RAM implementations of Virtex, Spartan-II and Spartan-IIE. DOUT resets to 0 when this feature is unavailable.
2. When reset is asserted, the FULL flags are asserted to prevent writes to the FIFO during reset.

Common Clock: Built-in FIFO

The FIFO Generator supports FIFO cores using the Virtex-4 built-in FIFO primitive with a common clock. This provides users the ability to use the built-in FIFO, while requiring only a single clock interface. The behavior of the common clock configuration with built-in FIFO is identical to the independent clock configuration with built-in FIFO, except all operations are in relation to the common clock (CLK). See [“Independent Clocks: Built-in FIFO,” page 56](#), for detailed information.

Special Design Considerations

This chapter provides additional design considerations for using the FIFO Generator core.

Resetting the FIFO

The FIFO Generator must be reset after the FPGA is configured and before operation begins. A reset pin (RST) is provided, and is an asynchronous reset that clears the internal counters and output registers. Internal to the core, RST is synchronized to the clock domain in which it is used, to ensure that the FIFO initializes to a known state. This synchronization logic allows for proper reset timing of the core logic, avoiding glitches and metastable behavior.

The generated FIFO core will initialize after reset to a known state. For details on the reset values and behavior, see [Chapter 5, “Designing with the Core.”](#)

Continuous Clocks

The FIFO Generator is designed to work only with free-running write and read clocks. Xilinx does not recommend controlling the core by manipulating RD_CLK and WR_CLK. If this functionality is required to gate FIFO operation, we recommend using the write enable (WR_EN) and read enable (RD_EN) signals.

Pessimistic Full and Empty

When independent clock domains are selected, the full flag (FULL, ALMOST_FULL) and empty flag (EMPTY, ALMOST_EMPTY) are pessimistic flags. FULL and ALMOST_FULL are synchronous to the write clock (WR_CLK) domain, while EMPTY and ALMOST_EMPTY are synchronous to the read clock (RD_CLK) domain.

The full flags are considered pessimistic flags because they assume that no read operations have taken place in the read clock domain. ALMOST_FULL is guaranteed to be asserted on the rising edge of WR_CLK when there is only one available location in the FIFO, and FULL is guaranteed to be asserted on the rising edge of WR_CLK when the FIFO is full. There may be a number of clock cycles between a read operation and the deassertion of FULL. The precise number of clock cycles for FULL to deassert is not predictable due to the crossing of clock domains and synchronization logic.

The EMPTY flags are considered pessimistic flags because they assume that no write operations have taken place in the write clock domain. ALMOST_EMPTY is guaranteed to be asserted on the rising edge of RD_CLK when there is only one more word in the FIFO, and EMPTY is guaranteed to be asserted on the rising edge of RD_CLK when the FIFO is empty. There may be a number of clock cycles between a write operation and the deassertion of

EMPTY. The precise number of clock cycles for EMPTY to deassert is not predictable due to the crossing of clock domains and synchronization logic.

See [Chapter 5, “Designing with the Core,”](#) for detailed information about the latency and behavior of the full and empty flags.

Programmable Full and Empty

The programmable full (PROG_FULL) and programmable empty (PROG_EMPTY) flags provide the user flexibility in specifying when the programmable flags assert and deassert. These flags can be set either by constant value(s) or by input port(s). These signals differ from the full and empty flags because they assert one (or more) clock cycle *after* the assert threshold has been reached. These signals are deasserted some time after the negate threshold has been passed. In this way, PROG_EMPTY and PROG_FULL are also considered pessimistic flags. See [Chapter 5, “Designing with the Core,”](#) for detailed information about the latency and behavior of the programmable flags.

Write Data Count and Read Data Count

When independent clock domains are selected, write data count (WR_DATA_COUNT) and read data count (RD_DATA_COUNT) signals are provided as an indication of the number of words in the FIFO relative to the write or read clock domains, respectively.

Consider the following when using the WR_DATA_COUNT or RD_DATA_COUNT ports.

- The WR_DATA_COUNT and RD_DATA_COUNT outputs are not an instantaneous representation of the number of words in the FIFO, but can instantaneously provide an approximation of the number of words in the FIFO.
- WR_DATA_COUNT and RD_DATA_COUNT may skip values from clock cycle to clock cycle.
- Using non-symmetric aspect ratios, or running clocks which vary dramatically in frequency, will increase the disparity between the data count outputs and the actual number of words in the FIFO.

Note: The WR_DATA_COUNT and RD_DATA_COUNT outputs will always be correct after some period of time where RD_EN=0 and WR_EN=0 (generally, just a few clock cycles after read and write activity stops).

Refer to [Chapter 5, “Designing with the Core,”](#) for details of the latency and behavior of the data count flags.

Setup and Hold Time Violations

When generating a FIFO with independent clock domains, the core internally synchronizes the write and read clock domains. Setup and hold time violations are therefore expected on certain registers within the core. In simulation, warning messages may be issued indicating these violations. If these warning messages are from the FIFO Generator core, they can be safely ignored. The core is designed to properly handle these conditions, regardless of the phase or frequency relationship between the write and read clocks.

Simulating Your Design

The FIFO Generator is provided as a Xilinx technology-specific netlist, and as a behavioral or structural simulation model. This chapter provides instructions for simulating the FIFO Generator in your design.

Simulation Models

The FIFO Generator supports two types of simulation models based on the Xilinx CORE Generator system project options. From the CORE Generator system, select Options under Project pull down menu. Under the Generation tab, the user can choose to generate either a behavioral model or a structural model. The models are available in both VHDL and Verilog.

Behavioral Models

The behavioral models provided are designed to reproduce the behavior and functionality of the FIFO Generator, but are not cycle-accurate models (except for the common clock FIFO with block RAM, distributed RAM, or shift registers). If cycle accurate models are required (and common clock FIFO with block RAM, distributed RAM or shift registers is not selected), it is recommended to use the structural model.

The behavioral models are considered to be zero-delay models, as the modeled write-to-read latency is nearly zero. The behavioral models are functionally correct, and will represent the behavior of the configured FIFO, although the write-to-read latency and the behavior of the status flags will differ from the actual implementation of the FIFO design.

To generate behavioral models, select Behavioral and VHDL or Verilog in the Xilinx CORE Generator project options. If default project options are selected, the Xilinx CORE Generator system will generate behavioral models.

The following considerations apply to the behavioral models.

- Write operations always occur relative to the write clock (WR_CLK) or common clock (CLK) domain, as do the corresponding handshaking signals.
- Read operations always occur relative to the read clock (RD_CLK) or common clock (CLK) domain, as do the corresponding handshaking signals.
- The delay through the FIFO (write-to-read latency) will differ between the VHDL model, the Verilog model, and the core.
- The deassertion of the status flags (full, almost full, programmable full, empty, almost empty, programmable empty) will vary between the VHDL model, the Verilog model, and the core.

Note: If independent clocks or common clocks with built-in FIFO is selected, it is strongly recommended to use the structural model, as the behavioral model does not correctly model the behavior of the status flags (full, programmable full, empty and programmable empty).

Structural Models

The structural models are designed to provide a cycle-accurate model of the FIFO Generator. The write-to-read latency and behavior of the status flags will match the latency and behavior seen in hardware.

To generate structural models, select Structural and VHDL or Verilog in the Xilinx CORE Generator project options.

Note: Simulation performance may be impacted when simulating the structural models compared to the behavioral models

Performance Information

Resource Utilization and Performance

Performance and resource utilization for a FIFO varies depending on the configuration and features selected when customizing the core. The tables below provide example FIFO configurations and the maximum performance and resources required.

[Table A-1](#) provides results for a FIFO configured without optional features. The benchmarks were performed using a Virtex-II 2v3000 -5 and Virtex-4 4vlx15 -11.

Table A-1: Benchmarks: FIFO Configured without Optional Features

FIFO Type	Depth x Width	Performance		Resources				
		Virtex-II	Virtex-4	LUTs	FFs	Block RAMs	Shift Registers	Distributed RAMs
Synchronous FIFO (block RAM)	512 x 16	233 MHz	275 MHz	64	43	1	0	0
Synchronous FIFO (block RAM)	4096 x 16	200 MHz	275 MHz	80	55	4	0	0
Synchronous FIFO (distributed RAM)	64 x 16	250 MHz	325 MHz	49	83	0	0	128
Synchronous FIFO (distributed RAM)	512 x 16	175 MHz	225 MHz	135	438	0	0	1024
Independent Clocks FIFO (block RAM)	512 x 16	225 MHz	275 MHz	103	105	1	0	0
Independent Clocks FIFO (block RAM)	4096 x 16	225 MHz	275 MHz	134	138	4	0	0
Independent Clocks FIFO (distributed RAM)	64 x 16	250 MHz	325 MHz	70	123	0	0	128

Table A-1: Benchmarks: FIFO Configured without Optional Features (Continued)

Independent Clocks FIFO (distributed RAM)	512 x 16	175 MHz	225 MHz	173	239	0	0	1024
Shift Register FIFO	64 x 16	225 MHz	325 MHz	68	43	0	64	0
Shift Register FIFO	512 x 16	150 MHz	200 MHz	312	82	0	512	0

Table A-2 provides results for a FIFO configured with multiple threshold inputs. The benchmarks were performed using a Virtex-II 2v3000 -5 and Virtex-4 4vlx15 -11.

Table A-2: FIFO Benchmarks: Configured with Multiple Programmable Thresholds

FIFO Type	Depth x Width	Performance		Resources				
		Virtex-II	Virtex-4	LUTs	FFs	Block RAMs	Shift Registers	Distributed RAMs
Synchronous FIFO (block RAM)	512 x 16	175 MHz	200 MHz	109	68	1	0	0
Synchronous FIFO (block RAM)	4096 x 16	175 MHz	225 MHz	135	86	4	0	0
Synchronous FIFO (distributed RAM)	64 x 16	225 MHz	250 MHz	80	109	0	0	128
Synchronous FIFO (distributed RAM)	512 x 16	150 MHz	200 MHz	182	420	0	0	1024
Independent Clocks FIFO (block RAM)	512 x 16	200 MHz	250 MHz	163	126	1	0	0
Independent Clocks FIFO (block RAM)	4096 x 16	175 MHz	250 MHz	214	165	4	0	0
Independent Clocks FIFO (distributed RAM)	64 x 16	225 MHz	325 MHz	143	146	0	0	128
Independent Clocks FIFO (distributed RAM)	512 x 16	150 MHz	225 MHz	251	277	0	0	1024
Shift Register FIFO	64 x 16	150 MHz	200 MHz	97	62	0	64	0
Shift Register FIFO	512 x 16	150 MHz	200 MHz	355	106	0	512	0

Table A-3 provides results for FIFOs configured to use the Virtex-4 built-in FIFO without optional features. The benchmarks were done against a Virtex-4 4vlx15-11 SF363 target device.

Table A-3: **Benchmarks: FIFO Configured with Virtex-4 Built-in FIFO Resources**

FIFO Type	Depth x Width	Performance	Resources		
			LUTs	FFs	Built-In FIFOs
Built-in FIFO (basic)	512 x 36	450 MHz ¹	2	2	1
Built-in FIFO (basic)	4096 x 36	300 MHz	5	2	4
Built-in FIFO (with handshaking)	512 x 36	350 MHz	4	8	1
Built-in FIFO (with handshaking)	4096 x 36	300 MHz	7	8	4

1. When Use Built-in FIFO Flags is selected.

Core Parameters

FIFO Parameters

Table B-1 describes the FIFO core parameters, including the XCO file value and the default settings.

Table B-1: FIFO Parameter Table

Parameter Name	XCO File Values	Default GUI Setting
Component_Name	instance_name ASCII text starting with a letter and using the following character set: a-z, 0-9, and _	fifo_generator_v2_1
FIFO Implementation	Common_Clock_Block_RAM Common_Clock_Distributed_RAM Common_Clock_Shift_Register Common_Clock_Builtin_FIFO Independent_Clocks_Block_RAM Independent_Clocks_Distributed_RAM Independent_Clocks_Builtin_FIFO	Common_Clock_Block_RAM
Input Data Width ¹	Integer in range 1 to 256	16
Output Data Width ¹	Integer in range 1 to 256	16
Input Depth ¹	2 ^N where N is an integer 4 to 24	1024
Output Depth ¹	2 ^M where M is an integer 4 to 24	1024
Primitive Depth	512, 1024, 2048, 4096	1024
Data Count Width	Integer in range 1 to log ₂ (Output Depth)	2
Almost Full Flag	true, false	false
Almost Empty Flag	true, false	false
Programmable Full Type	No_Programmable_Full_Threshold Single_Programmable_Full_Threshold_Constant Multiple_Programmable_Full_Threshold_Constants Single_Programmable_Full_Threshold_Input_Port Multiple_Programmable_Full_Threshold_Input_Ports	No_Programmable_Full_Threshold
Full Threshold Assert Value	See range under Programmable Flags.	768
Full Threshold Negate Value	See range under “Programmable Flags,” page 47.	768

Table B-1: FIFO Parameter Table (Continued)

Parameter Name	XCO File Values	Default GUI Setting
Programmable Empty Type	No_Programmable_Empty_Threshold Sigle_Programmable_Empty_Threshold_Constant Multiple_Programmable_Empty_Threshold_Constants Single_Programmable_Empty_Threshold_Input_Port Multiple_Programmable_Empty_Threshold_Input_Ports	No_Programmable_Empty_Threshold
Empty Threshold Assert Value	See range under “Programmable Flags,” page 47.	256
Empty Threshold Negate Value	See range under “Programmable Flags,” page 47.	256
Write Acknowledge Flag	true, false	false
Write Acknowledge Sense	Active_High, Active_Low	Active_High
Overflow Flag	true, false	false
Overflow Sense	Active_High, Active_Low	Active_High
Valid Flag	true, false	false
Valid Sense	Active_High, Active_Low	Active_High
Underflow Flag	true, false	false
Underflow Sense	Active_High, Active_Low	Active_High
Dout Reset Value	Hex value in range of 0 to output data width - 1	0
Primitive Depth	512, 1024, 2048, 4096	1024
Read Data Count	true, false	false
Read Data Count Width	Integer in range 1 to $\log^2(\text{output depth})$	2
Write Data Count	true, false	false
Write Data Count Width	Integer in range 1 to $\log^2(\text{input depth})$	2
Data Count	true, false	false
Register Outputs	true, false	false
Read Latency	integer range 0 to 1	1
Reset Pin	true, false	true

1. User customized core should not exceed the number of shift registers, built-in FIFOs, block RAM, or distributed RAM primitives available in the targeted architecture. It is the user’s responsibility to know the resource availability in the targeted device.