LogiCORE IP FIFO Generator v8.1

User Guide

UG175 March 1, 2011

Xilinx is providing this product documentation, hereinafter “Information,” to you “AS IS” with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice.

XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Except as stated herein, none of the Information 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.

© 2005-2011 Xilinx, Inc. XILINX, the Xilinx logo, Virtex, Spartan, ISE, and other designated brands included herein are trademarks of Xilinx in the United States and other countries. 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 ISE v7.1i, 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.
7/13/06	4.0	Added Virtex-5 support, reorganized Chapter 5, added ISE v8.2i, version to 3.1
9/21/06	5.0	Core version updated to v3.2; support added for Spartan-3A.
2/15/07	6.0	Core version updated to 3.3; Xilinx tools updated to 9.1i.
4/02/07	6.5	Added support for Spartan-3A DSP devices.
8/8/07	6.6	Updated core to v4.1, ISE tools 9.2i, Cadence IUS v5.8.
10/10/07	7.0	Updated core to v4.2, IUS to v6.1, Xilinx trademark references.
3/24/08	8.0	Updated core to v4.3, ISE tools 10.1, Mentor Graphics® ModelSim® v6.3c.
9/19/08	9.0	Updated core to v4.4, ISE tools 10.1, SP3.
12/17/08	9.0.1	Early access documentation.
4/24/09	10.0	Updated core to v5.1, and ISE tools to v11.1.
6/24/09	10.5	Updated core to v5.2 and ISE tools to v11.2.
6/24/09	10.6	Updated Appendix A, “Performance Information.”
9/16/09	11.0	Updated core to v5.3 and ISE tools to v11.3.
4/19/10	12.0	Updated core to v6.1 and ISE tools to v12.1.
7/23/10	13.0	Updated core to v6.2 and ISE tools to v12.2.
9/21/10	14.0	Updated core to v7.2 and ISE tools to v12.3. Added AXI4 Interface support.
3/1/11	15.0	Updated core to v8.1 and ISE tools to v13.1.

Table of Contents

Chapter 3: Generating the Native FIFO Core

Schedule of Figures

Schedule of Tables

Chapter 5: Designing with the Core

Appendix C: DOUT Reset Value Timing

About This Guide

The *LogicCORE™ IP FIFO Generator User Guide* describes the function and operation of the FIFO Generator, as well as 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 and the conventions used in this guide.
- Chapter 1, "Introduction," describes the core and related information, including recommended design experience, additional resources, technical support, and submitting feedback to Xilinx.
- Chapter 2, "Core Overview," describes the core configuration options and their interfaces.
- Chapter 3, "Generating the Native FIFO Core," describes how to generate the Native interface core using the Xilinx CORE Generator™ Graphical User Interface (GUI).
- Chapter 4, "Generating the AXI4 FIFO Core," describes how to generate the AXI4 interface FIFO core using the Xilinx CORE 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 designing 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 the CORE Generator GUI for the FIFO Generator.
- Appendix C, "DOUT Reset Value Timing," provides the timing diagram for DOUT reset value for various FIFO configurations.

Additional Resources

To find additional documentation, see the Xilinx website at:

http://www.xilinx.com/support/documentation/index.htm.

To search the Answer Database of silicon, software, and IP questions and answers, or to create a technical support WebCase, see the Xilinx website at:

http://www.xilinx.com/support/mysupport.htm.

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 that you enter in a syntactical statement	**`ngdbuild`** *design_name*
Helvetica bold	Commands that you select from a menu	**File** → **Open**
	Keyboard shortcuts	**Ctrl+C**
Italic font	Variables in a syntax statement for which you must supply values	**`ngdbuild`** *design_name*
	References to other manuals	See the *User Guide* for more information.
	Emphasis in text	If a wire is drawn so that it overlaps the pin of a symbol, the two nets are *not* connected.
Dark Shading	Items that are not supported or reserved	This feature is not supported
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`**`}`
Angle brackets < >	User-defined variable or in code samples	<directory name>

Vertical ellipsis
.
.
.
	Repetitive material that has been omitted	`IOB #1: Name = QOUT'`
		`IOB #2: Name = CLKIN'`
		.
		.
		.

Convention	Meaning or Use	Example
Horizontal ellipsis . . .	Repetitive material that has been omitted	**`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 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.

Chapter 1

Introduction

The FIFO Generator core is a fully verified first-in first-out memory queue for use in any application requiring in-order storage and retrieval, enabling high-performance and area-optimized designs. The core provides an optimized solution for all FIFO configurations and delivers maximum performance (up to 500 MHz) while utilizing minimum resources.

The Xilinx FIFO Generator core supports Native interface FIFOs and AXI4 Interface FIFOs. Native interface FIFO Generators (FIFOs) are the original standard FIFO functions delivered by the previous versions of the FIFO Generator (up to v6.2). AXI4 Interface FIFOs are derived from the Native interface FIFO. Three AXI4 interface styles are available: AXI4-Stream, AXI4 and AXI4-Lite.

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 CORE Generator IP core, included with the latest ISE® Design Suite release in the Xilinx Download Center. The core licensed under the terms of the Xilinx End User License and no FLEX license key is required. For detailed information about the core, see the FIFO Generator product page.

Windows

- Windows XP® Professional 32-bit/64-bit
- Windows Vista® Business 32-bit/64-bit

Linux

- Red Hat® Enterprise WS 4.0 32-bit/64-bit
- Red Hat Enterprise Desktop 5.0 32-bit/64-bit (with Workstation option)
- SUSE Linux Enterprise (SLE) v10.1 32-bit/64-bit

Software

- ISE v13.1

Recommended Design Experience

The FIFO Generator is a fully verified solution, and can be used by all levels of design engineers.

Important: When implementing a FIFO with independent write and read clocks, special care must be taken to ensure the FIFO Generator is correctly used. Synchronization Considerations, page 97 provides important information to help ensure correct design configuration.

Similarly, asynchronous designs should also be aware that the behavioral models do not model synchronization delays. See Chapter 7, Simulating Your Design for details.

Technical Support

For technical support, visit www.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 www.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 www.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

Chapter 2

Core Overview

The FIFO Generator core supports the generation of Native interface FIFOs and AXI4 Interface FIFOs. Native interface FIFO Generators ("FIFOs") are the original standard FIFO functions delivered by the previous versions of the LogiCORE FIFO Generator (up to Version 6.2). Native interface FIFO Generators are optimized for buffering, data-width conversion and clock domain de-coupling applications, providing in-order storage and retrieval.

AXI4 Interface FIFOs are derived from the Native interface FIFOs. Three AXI4 interface styles are available: AXI4-Stream, AXI4 and AXI4-Lite.

Native Interface FIFOs

The Native interface FIFO can be customized to utilize block RAM, distributed RAM or built-in FIFO FPGA resources available in some FPGA families to create high-performance, area-optimized FPGA designs.

Standard mode and First Word Fall Through are the two operating modes available for Native interface FIFOs.

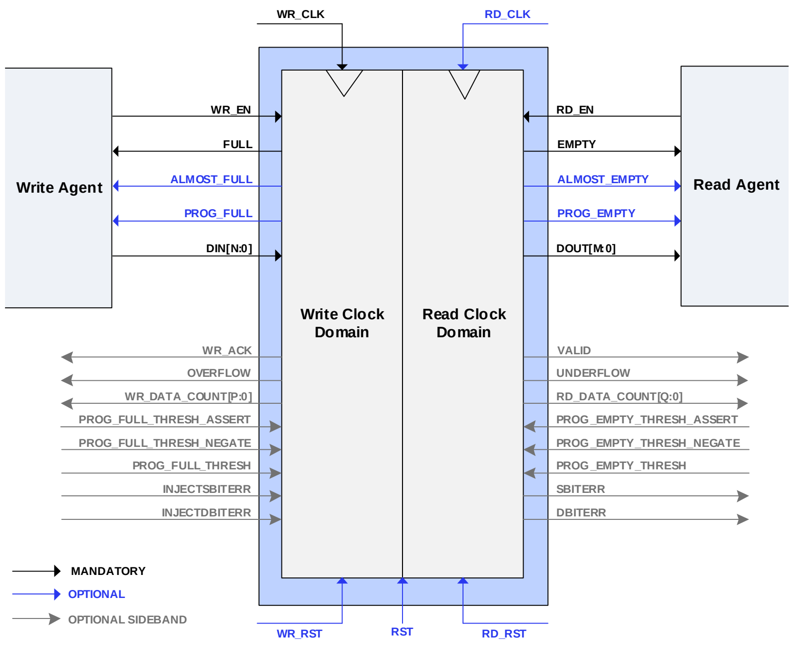


Figure 2-1: **Native Interface FIFOs Signal Diagram**

Native FIFO Feature Overview

Clock Implementation and Operation

The FIFO Generator can configure FIFOs 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 relationships between clocks. A common clock domain implementation optimizes the core for data buffering within a single clock domain.

Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA Built-in FIFO Support

The FIFO Generator supports the Virtex®-6 and Virtex-5 FPGA built-in FIFO modules, enabling the creation of large FIFOs 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. The built-in Error Injection and Correction Checking (ECC) feature in the built-in FIFO macro is also available.

Virtex-4 FPGA Built-in FIFO Support

The FIFO Generator supports a single instantiation of the Virtex-4 FPGA built-in FIFO module. The core also implements a FIFO flag patch ("Solution 1: Synchronous/Asynchronous Clock Work-Arounds," defined in the *Virtex-4 FPGA User Guide*), based on estimated clock frequencies. This patch is implemented in fabric. See Appendix A, Performance Information for resource utilization estimates.

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 data out 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.

See Table 2-2 for FWFT availability. 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 107)
- Programmable empty (see Non-symmetric Aspect Ratio and First-Word Fall-Through, page 123)
- Data counts (see First-Word Fall-Through Data Count, page 118 and Non-symmetric Aspect Ratio and First-Word Fall-Through, page 123)

Supported Memory Types

The FIFO Generator implements FIFOs built from block RAM, distributed RAM, shift registers, or the built-in FIFOs for Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs. The core combines memory primitives in an optimal configuration based on the selected width and depth of the FIFO. Table 2-1 provides best-use recommendations for specific design requirements.

Table 2-1: **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 Support

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.

Embedded Registers in Block RAM and FIFO Macros

In Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA block RAM and FIFO macros and Virtex-4 FPGA block RAM macros, embedded output registers are available to increase performance and add a pipeline register to the macros. This feature can be leveraged to add one additional latency to the FIFO core (DOUT bus and VALID outputs) or implement the output registers for FWFT FIFOs. The embedded registers available in Kintex-7, Virtex-7, and Virtex-6 FPGAs can be reset (DOUT) to a default or user programmed value for common clock built-in FIFOs. See Embedded Registers in Block RAM and FIFO Macros (Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs), page 124 for more information.

Error Injection and Correction (ECC) Support

The block RAM and FIFO macros are equipped with built-in Error Correction Checking (ECC) in the Virtex-5 FPGA architecture and built-in Error Injection and Correction Checking in the Kintex-7, Virtex-7, and Virtex-6 FPGA architectures. Error Injection and Correction are available for both the common and independent clock block RAM or built-in based FIFOs.

Native FIFO Core Configuration and Implementation

Table 2-2 provides a summary of the supported memory and clock configurations.

Table 2-2: **FIFO Configurations**

Clock Domain	Memory Type	Non-symmetric Aspect Ratios	First-Word Fall-Through	ECC Support	Embedded Register Support	Error Injection	Reset Option for Embedded Register (with/without DOUT Reset Value)[a]
Common	Block RAM		✓	✓	✓[b]	✓	✓
Common	Distributed RAM		✓				
Common	Shift Register						
Common	Built-in FIFO[c]		✓[d]	✓	✓[e]	✓	✓
Independent	Block RAM	✓	✓	✓	✓[b]	✓	
Independent	Distributed RAM		✓				
Independent	Built-in FIFO[c]		✓[d]	✓		✓[f]	

a. Available only if Embedded register option is selected.
b. Embedded register support is only available for Kintex-7, Virtex-7, Virtex-6, Virtex-4, and Virtex-5 FPGA block RAM-based FIFOs.
c. The built-in FIFO primitive is only available in the Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 architectures.
d. FWFT is only supported for built-in FIFOs in Kintex-7, Virtex-7, Virtex-6 and Virtex-5 devices.
e. Only available for Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA common clock built-in FIFOs.
f. Available only if ECC option is selected.

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. In addition, for the block RAM and distributed RAM implementations, the user has the option to select a synchronous or asynchronous reset for the core. For Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA designs, the block RAM FIFO configuration also supports ECC.

Common Clock: Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA Built-in FIFO

This implementation category allows the user to select the built-in FIFO available in the Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA architectures 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). The Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA built-in FIFO configurations also support the built-in ECC feature.

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). For Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA designs, the block RAM FIFO configuration also supports ECC.

Independent Clocks: Built-in FIFO for Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGAs

This implementation category allows the user to select the built-in FIFO available in the Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA architectures. 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). The Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA built-in FIFO configurations also support the built-in ECC feature.

Native FIFO Generator Feature Summary

Table 2-3 summarizes the FIFO Generator features supported for each clock configuration and memory type.

Table 2-3: **FIFO Configurations Summary**

FIFO Feature	Independent Clocks			Common Clock		
	Block RAM	Distributed RAM	Built-in FIFO[a]	Block RAM	Distributed RAM, Shift Register	Built-in FIFO[a]
Non-symmetric Aspect Ratios[b]	✓					
Symmetric Aspect Ratios	✓	✓	✓	✓	✓	✓
Almost Full	✓	✓		✓	✓	
Almost Empty	✓	✓		✓	✓	
Handshaking	✓	✓	✓	✓	✓	✓
Data Count	✓	✓		✓	✓	

Table 2-3: **FIFO Configurations Summary** ***(Cont'd)***

FIFO Feature	Independent Clocks			Common Clock		
	Block RAM	Distributed RAM	Built-in FIFO[a]	Block RAM	Distributed RAM, Shift Register	Built-in FIFO[a]
Programmable Empty/Full Thresholds	✓	✓	✓[c]	✓	✓	✓[c]
First-Word Fall-Through	✓	✓	✓[d]	✓	✓[e]	✓[d]
Synchronous Reset				✓	✓	
Asynchronous Reset	✓[f]	✓[f]	✓	✓[f]	✓[f]	✓
DOUT Reset Value	✓	✓		✓	✓	✓[g]
ECC	✓[i]		✓[h]	✓[i]		✓[i]
Embedded Register	✓[i]			✓[j]		✓[j]

a. For Virtex-4 FPGA Built-in FIFO macro, the valid width range is 4, 9, 18 and 36 and the valid depth range automatically varies based on write width selection. For Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA Built-in FIFO macros, the valid width range is 1 to 1024 and the valid depth range is 512 to 4194304. Only depths with powers of 2 are allowed.

b. 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.

c. For built-in FIFOs, the range of Programmable Empty/Full threshold is limited to take advantage of the logic internal to the macro.

d. First-Word-Fall-Through is only supported for the Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA built-in FIFOs.

e. First-Word-Fall-Through is supported for distributed RAM FIFO only.

f. Asynchronous reset is optional for all FIFOs built using distributed and block RAM.

g. DOUT reset value is supported only in Kintex-7, Virtex-7, and Virtex-6 FPGA common clock built-in FIFOs with embedded register option selected.

h. ECC is only supported for the Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA block RAM and built-in FIFOs.

i. Embedded register option is only supported in Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA block RAM FIFOs.

j. Embedded register option is supported only in Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA common clock built-in FIFOs. See Embedded Registers in Block RAM and FIFO Macros, page 25.

Using Block RAM FIFOs Versus Built-in FIFOs

The Built-In FIFO solutions were implemented to take advantage of logic internal to the Built-in FIFO macro. Several features, for example, non-symmetric aspect ratios, almost full, almost empty, and so forth were not implemented because they are not native to the macro and require additional logic in the fabric to implement.

Benchmarking suggests that the advantages the Built-In FIFO implementations have over the block RAM FIFOs (for example, logic resources) diminish as external logic is added to implement features not native to the macro. This is especially true as the depth of the implemented FIFO increases. It is strongly recommended that users requiring features not available in the Built-In FIFOs implement their design using block RAM FIFOs.

Native FIFO Interface Signals

The following sections define the FIFO interface signals. Figure 2-2 illustrates these signals (both standard and optional ports) for a FIFO core that supports independent write and read clocks.

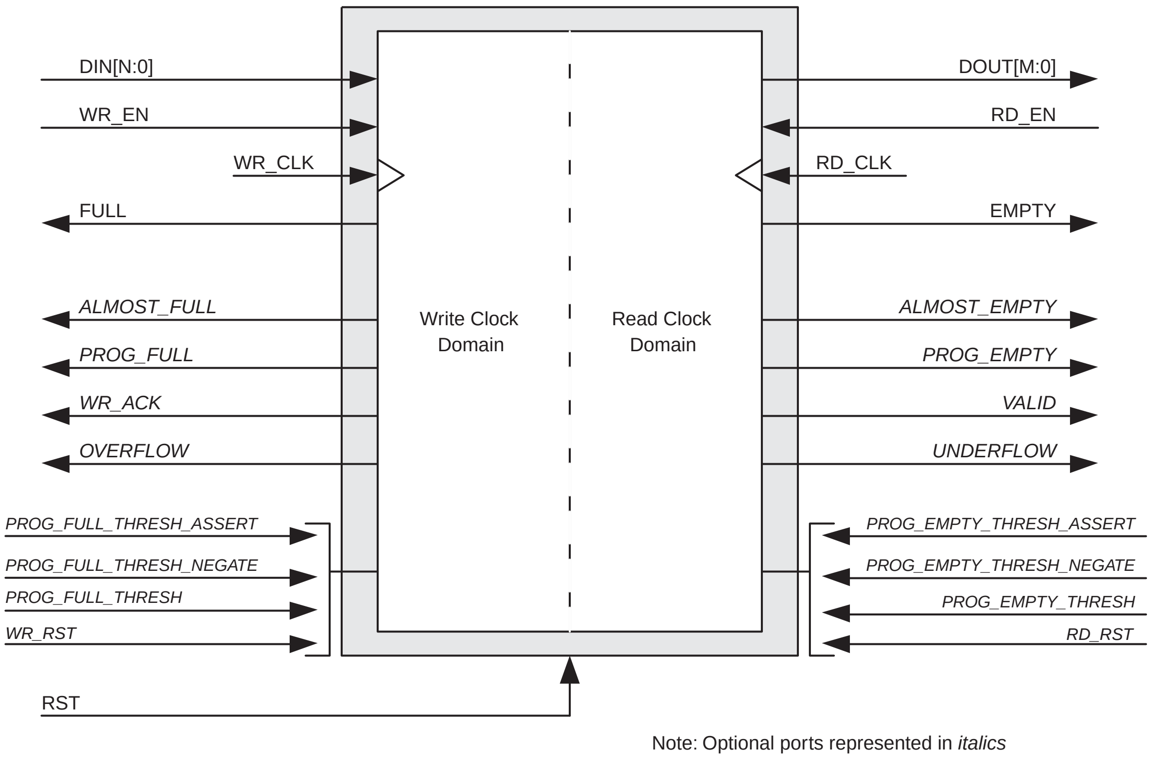


Figure 2-2: **FIFO with Independent Clocks: Interface Signals**

Interface Signals: FIFOs With Independent Clocks

The RST signal, as defined Table 2-4, causes a reset of the entire core logic (both write and read clock domains. It is an asynchronous input synchronized internally in the core before use. The initial hardware reset should be generated by the user.

Table 2-4: **Reset Signal for FIFOs with Independent Clocks**

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

Table 2-5 defines the write interface signals 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 2-5: **Write Interface Signals for FIFOs with Independent Clocks**

Name	Direction	Description
		Required
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.
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 not destructive to the contents of the FIFO.
		Optional
WR_RST	Input	Write Reset: Synchronous to write clock. When asserted, initializes all internal pointers and flags of write clock domain.
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 written into 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. The exception to this behavior is when a write operation occurs at the rising edge of WR_CLK, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. If D is less than log2(FIFO depth)-1, the bus is truncated by removing the least-significant bits.
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 not 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 de-assertion 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).

Table 2-5: **Write Interface Signals for FIFOs with Independent Clocks *(Cont'd)***

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 de-asserted. The threshold can be dynamically set in-circuit during reset.
INJECTSBITERR	Input	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, and Virtex-6 FPGA block RAMs or built-in FIFO macros.
INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, and Virtex-6 FPGA block RAMs or built-in FIFO macros.

Table 2-6 defines the read interface signals of a FIFO with independent clocks. Read interface signals are divided into required signals and optional signals, and all signals are synchronous to the read clock (`RD_CLK`).

Table 2-6: **Read Interface Signals for FIFOs with Independent Clocks**

Name	Direction	Description
Required		
RD_RST	Input	Read Reset: Synchronous to read clock. When asserted, initializes all internal pointers, flags and output registers of read clock domain.
RD_CLK	Input	Read Clock: All signals on the read domain are synchronous to this clock.
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 not destructive to the FIFO.
Optional		
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 de-asserted when the number of words in the FIFO exceeds the programmable threshold.

Table 2-6: **Read Interface Signals for FIFOs with Independent Clocks *(Cont'd)***

Name	Direction	Description
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. The exception to this behavior is when the read operation occurs at the rising edge of RD_CLK, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. If C is less than log2(FIFO depth)-1, the bus is truncated by removing the least-significant bits.
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	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) 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_EMPTY_THRESH), or the user can control these values independently (using PROG_EMPTY_THRESH_ASSERT and PROG_EMPTY_THRESH_NEGATE).
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 de-asserted. The threshold can be dynamically set in-circuit during reset.
SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, Virtex-6 or Virtex-5 FPGA block RAM or built-in FIFO macro.
DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, Virtex-6 or Virtex-5 FPGA block RAM or built-in FIFO macro and data in the FIFO core is corrupted.

Interface Signals: FIFOs with Common Clock

Table 2-7 defines the interface signals of a FIFO with a common write and read clock and is divided into standard and optional interface signals. All signals (except asynchronous reset) are synchronous to the common clock (CLK). Users have the option to select synchronous or asynchronous reset for the distributed or block RAM FIFO implementation. See the *FIFO Generator User Guide* for specific information on reset requirements and behavior.

Table 2-7: **Interface Signals for FIFOs with a Common Clock**

Name	Direction	Description
		Required
RST	Input	Reset: An asynchronous reset that initializes all internal pointers and output registers.
SRST	Input	Synchronous Reset: A synchronous 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 not 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 not destructive to the FIFO.
		Optional
DATA_COUNT [C:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. If C is less than log2(FIFO depth)-1, the bus is truncated by removing the least-significant bits.
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 not destructive to the FIFO.

Table 2-7: **Interface Signals for FIFOs with a Common Clock *(Cont'd)***

Name	Direction	Description
PROG_FULL_THRESH	Input	Programmable Full Threshold: This signal is used to set the threshold value for the assertion and de-assertion 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 de-asserted. 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 de-asserted 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	Programmable Empty Threshold: This signal is used to set the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) 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_EMPTY_THRESH), or the user can control these values independently (using PROG_EMPTY_THRESH_ASSERT and PROG_EMPTY_THRESH_NEGATE).
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 de-asserted. The threshold can be dynamically set in-circuit during reset.

Table 2-7: **Interface Signals for FIFOs with a Common Clock *(Cont'd)***

Name	Direction	Description
SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, Virtex-6 or Virtex-5 FPGA block RAM or built-in FIFO macro.
DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, Virtex-6 or Virtex-5 FPGA block RAM or built-in FIFO macro and data in the FIFO core is corrupted.
INJECTSBITERR	Input	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM or built-in FIFO macro. For detailed information, see Chapter 5, "Designing with the Core."
INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM or built-in FIFO macro. For detailed information, see Chapter 5, "Designing with the Core."

AXI4 Interface FIFOs

AXI4 interface FIFOs are derived from the Native interface FIFO, as shown in Figure 2-3. Three AXI4 interface styles are available: AXI4-Stream, AXI4 and AXI4-Lite. In addition to applications supported by the Native interface FIFO, AXI4 FIFOs can also be used in AXI4 System Bus and Point-to-Point high speed applications.

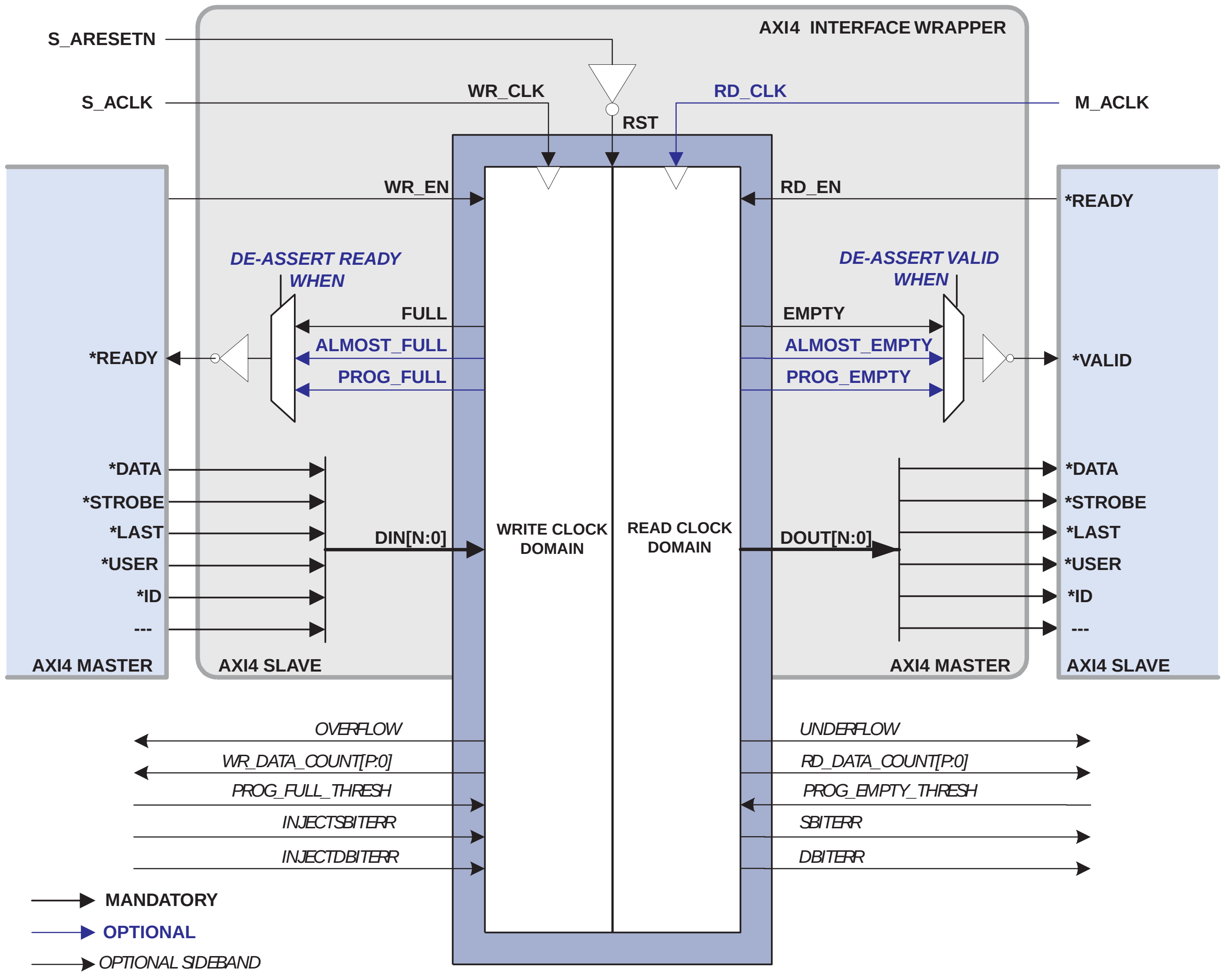


Figure 2-3: **AXI4 Interface FIFOs SIgnal Diagram**

AXI4 Interface FIFOs do not support built-in FIFO and Shift Register FIFO configurations.

The AXI4 interface protocol uses a two-way VALID and READY handshake mechanism. The information source uses the VALID signal to show when valid data or control information is available on the channel. The information destination uses the READY signal to show when it can accept the data. Figure 2-4 shows an example timing diagram for Write and Read operations to the AXI4 FIFO.

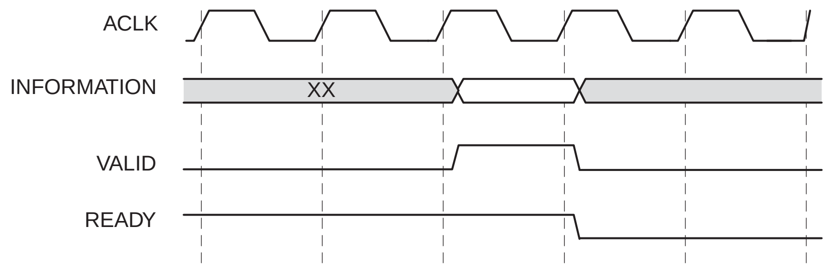


Figure 2-4: **AXI4 FIFO Timing Diagram**

In this example timing diagram, the information source generates the VALID signal to indicate when the data is available. The destination generates the READY signal to indicate that it can accept the data, and transfer occurs only when both the VALID and READY signals are high.

Because AXI4 FIFOs are derived from Native Interface FIFOs, much of the behavior is common between them. The READY signal is generated based on availability of space in the FIFO and is held high to allow writes to the FIFO. The READY signal is pulled low only when there is no space in the FIFO left to perform additional writes. The VALID signal is generated based on availability of data in the FIFO and is held high to allow reads to be performed from the FIFO. The VALID signal is pulled low only when there is no data available to be read from the FIFO. The INFORMATION signals are mapped to the DIN and DOUT bus of Native Interface FIFOs. The width of the AXI4 FIFO is determined by concatenating all of the INFORMATION signals of the AXI4 Interface. The INFORMATION signals include all AXI4 signals except for VALID and READY handshake signals.

AXI4 FIFOs operate in First-Word Fall-Through mode only. 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.

AXI4 FIFOs Feature Overview

AXI4 support is available for Kintex-7, Virtex-7, Virtex-6, and Spartan-6 FPGA devices only.

Easy Integration of Independent FIFOs for Read and Write Channels

For AXI4 and AXI4-Lite interfaces, AXI4 specifies Write Channels and Read Channels. Write Channels include a Write Address Channel, Write Data Channel and Write Response Channel. Read Channels include a Read Address Channel and Read Data Channel. The FIFO Generator provides the ability to generate either Write Channels or Read Channels for AXI4, or both Write Channels and Read Channels. Three FIFOs are integrated for Write Channels and two FIFOs are integrated for Read Channels. When both Write and Read Channels are selected, the FIFO Generator integrates five independent FIFOs.

For AXI4 and AXI4-Lite interfaces, the FIFO Generator provides the ability to implement independent FIFOs for each channel, as shown in Figure 2-5. For each channel, the core can be independently configured to generate a block RAM or distributed memory-based FIFO. The depth of each FIFO can also be independently configured.

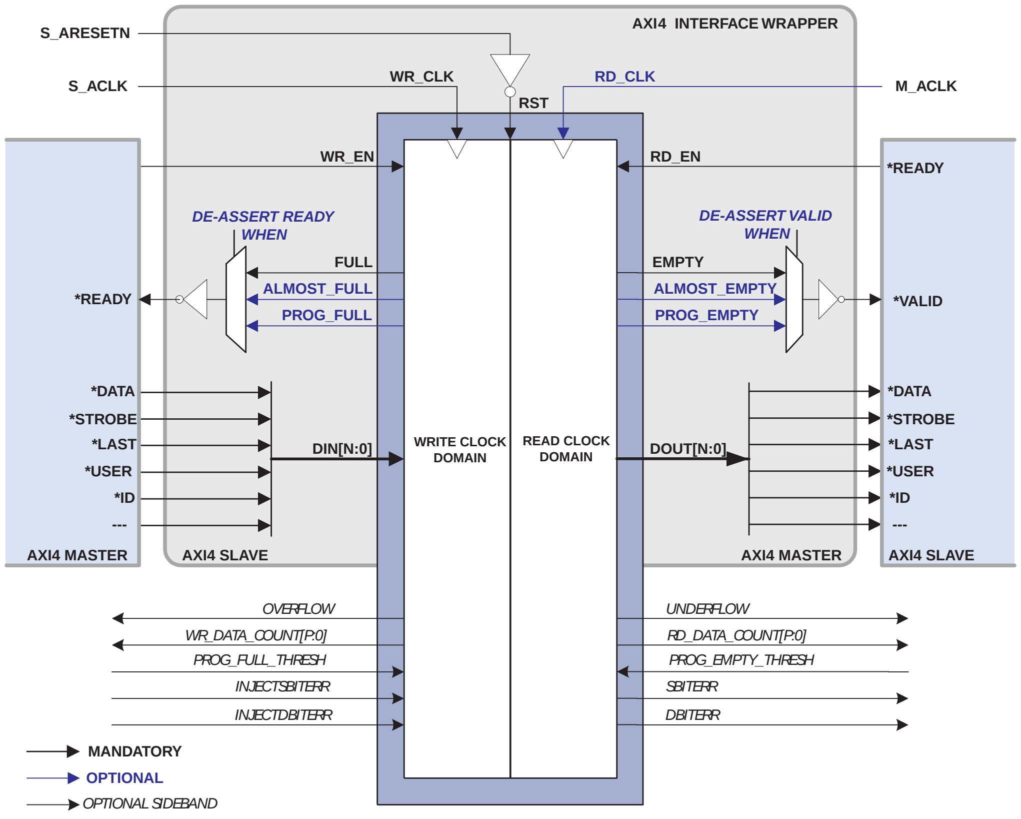


Figure 2-5: **AXI4 Block Diagram**

Clock and Reset Implementation and Operation

For the AXI4-Stream, AXI4 and AXI4-Lite interfaces, all instantiated FIFOs share clock and asynchronous active low reset signals (as shown Figure 2-5). In addition, all instantiated FIFOs can support either independent clock or common clock operation.

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 a single clock by selecting the common clock option.

Automatic FIFO Width Calculation

AXI4 FIFOs support symmetric widths for the FIFO Read and Write ports. The FIFO width for the AXI4 FIFO is determined by the selected interface type (AXI4-Stream, AXI4 or AXI4-Lite) and user-selected signals and signal widths within the given interface. The AXI4 FIFO width is then calculated automatically by the aggregation of all signal widths in a respective channel.

Supported Memory Types

The FIFO Generator implements FIFOs built from block RAM or distributed RAM. The core combines memory primitives in an optimal configuration based on the calculated width and selected depth of the FIFO.

Error Injection and Correction (ECC) Support

The block RAM macros are equipped with built-in Error Injection and Correction Checking in the Kintex-7, Virtex-7, and Virtex-6 FPGA architectures. This feature is available for both the common and independent clock block RAM FIFOs.

For more details on Error Injection and Correction, see Built-in Error Correction Checking in Chapter 5.

AXI4 Slave Interface for Performing Writes

The AXI4 FIFO provides an AXI4 Slave interface for performing Writes. In the timing diagram shown in Figure 2-4, the AXI4 Master provides INFORMATION and VALID signals; the AXI4 Slave interface indicates it is ready to accept the INFORMATION by asserting the READY signal. The READY signal will be de-asserted only when the FIFO is either Full, Almost Full, or when the Programmable Full threshold is reached. The de-assertion of READY can be controlled by setting "De-assert READY When" option.

AXI4 Master Interface for Performing Reads

For performing reads, the AXI4 FIFO provides an AXI4 Master interface. In Figure 2-4, the AXI4 FIFO provides INFORMATION and VALID signals; on detecting READY signal asserted from AXI4 Slave interface, the AXI4 FIFO will place the next INFORMATION on the bus. The VALID signal will be de-asserted only when the FIFO is either Empty or Almost Empty or when the FIFO occupancy is less than the Programmable Empty threshold. The de-assertion of VALID can be controlled by setting "De-assert VALID When" option.

AXI4 FIFOs Feature Summary

Table 2-8 summarizes the supported FIFO Generator features for each clock configuration and memory type.

Table 2-8: **AXI4 FIFOs Configuration Summary**

FIFO Options	Common Clock		Independent Clock	
	Block RAM	Distributed Memory	Block RAM	Distributed Memory
Full	✓	✓	✓	✓
Almost Full	✓	✓	✓	✓
Programmable Full	✓	✓	✓	✓
Empty	✓	✓	✓	✓
Almost Empty	✓	✓	✓	✓
Programmable Empty	✓	✓	✓	✓
Data Counts	✓	✓	✓	✓

Table 2-8: **AXI4 FIFOs Configuration Summary**

FIFO Options	Common Clock		Independent Clock	
	Block RAM	Distributed Memory	Block RAM	Distributed Memory
ECC	✓		✓	
Interrupt Flags	✓	✓	✓	✓

AXI4 FIFOs Interface Signals

The following sections define the AXI4 FIFO interface signals.

The value of `S_AXIS_TREADY`, `S_AXI_AWREADY`, `S_AXI_WREADY`, `M_AXI_BREADY`, `S_AXI_ARREADY` and `M_AXI_RREADY` is 1 when `S_ARESETN` is 0. To avoid unexpected behavior, do not perform any transactions while `S_ARESETN` is 0.

Global Signals

Table 2-9 defines the global interface signals for AXI4 FIFO.

The `S_ARESETN` signal causes a reset of the entire core logic. It is an active low, asynchronous input synchronized internally in the core before use. The initial hardware reset should be generated by the user.

Table 2-9: **AXI4 FIFO - Global Interface Signals**

Name	Direction	Description
Global Clock and Reset Signals Mapped to FIFO Clock and Reset Inputs		
M_ACLK	Input	Global Master Interface Clock: All signals on Master Interface of AXI4 FIFO are synchronous to M_ACLK
S_ACLK	Input	Global Slave Interface Clock: All signals are sampled on the rising edge of this clock.
S_ARESETN	Input	Global reset: This signal is active low.

AXI4-Stream FIFO Interface Signals

Table 2-10 defines the AXI4-Stream FIFO interface signals.

Table 2-10: **AXI4-Stream FIFO Interface Signals**

Name	Direction	Description
AXI4-Stream Interface: Handshake Signals for FIFO Write Interface		
S_AXIS_TVALID	Input	TVALID: Indicates that the master is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
S_AXIS_TREADY	Output	TREADY: Indicates that the slave can accept a transfer in the current cycle.
AXI4-Stream Interface: Information Signals Mapped to FIFO Data Input (DIN) Bus		
S_AXIS_TDATA[m-1:0]	Input	TDATA: The primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.

Table 2-10: **AXI4-Stream FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
S_AXIS_TSTRB[m/8-1:0]	Input	TSTRB: The byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid
S_AXIS_TKEEP[m/8-1:0]	Input	TKEEP: The byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream. Associated bytes that have the TKEEP byte qualifier deasserted are null bytes and can be removed from the data stream. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • KEEP[0] = 1b, DATA[7:0] is a NULL byte • KEEP [7] = 0b, DATA[63:56] is not a NULL byte
S_AXIS_TLAST	Input	TLAST: Indicates the boundary of a packet.
S_AXIS_TID[m:0]	Input	TID: The data stream identifier that indicates different streams of data.
S_AXIS_TDEST[m:0]	Input	TDEST: Provides routing information for the data stream.
S_AXIS_TUSER[m:0]	Input	TUSER: The user-defined sideband information that can be transmitted alongside the data stream.
AXI4-Stream Interface: Handshake Signals for FIFO Read Interface		
M_AXIS_TVALID	Output	TVALID: Indicates that the master is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
M_AXIS_TREADY	Input	TREADY: Indicates that the slave can accept a transfer in the current cycle.
AXI4-Stream Interface: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXIS_TDATA[m-1:0]	Output	TDATA: The primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.
M_AXIS_TSTRB[m/8-1:0]	Output	TSTRB: The byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid

Table 2-10: **AXI4-Stream FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
M_AXIS_TKEEP[m/8-1:0]	Output	TKEEP: The byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream. Associated bytes that have the TKEEP byte qualifier deasserted are null bytes and can be removed from the data stream. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • KEEP[0] = 1b, DATA[7:0] is a NULL byte • KEEP [7] = 0b, DATA[63:56] is not a NULL byte
M_AXIS_TLAST	Output	TLAST: Indicates the boundary of a packet.
M_AXIS_TID[m:0]	Output	TID: The data stream identifier that indicates different streams of data.
M_AXIS_TDEST[m:0]	Output	TDEST. Provides routing information for the data stream.
M_AXIS_TUSER[m:0]	Output	TUSER: The user-defined sideband information that can be transmitted alongside the data stream.
AXI4-Stream FIFO: Optional Sideband Signals		
AXIS_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. *D* = log2(FIFO depth)-1
AXIS_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. *D* = log2(FIFO depth)-1
AXIS_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single-bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO. For detailed information, see UG175, *FIFO Generator User Guide*.
AXIS_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double-bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO. For detailed information, see UG175, *FIFO Generator User Guide*.
AXIS_SBITERR	Output	Single-Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO. For detailed information, see UG175, *FIFO Generator User Guide*.
AXIS_DBITERR	Output	Double-Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted. For detailed information, see UG175, *FIFO Generator User Guide*.
AXIS_OVERFLOW	Output	Overflow: Indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO.

Table 2-10: **AXI4-Stream FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXIS_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock; that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXIS_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.
AXIS_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock; that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXIS_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$

AXI4 FIFO Interface Signals

Write Channels

Table 2-11 defines the AXI4 FIFO interface signals for Write Address Channel.

Table 2-11: **AXI4 Write Address Channel FIFO Interface Signals**

Name	Direction	Description
AXI4 Interface Write Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus		
S_AXI_AWID[m:0]	Input	Write Address ID: Identification tag for the write address group of signals.
S_AXI_AWADDR[m:0]	Input	Write Address: The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
S_AXI_AWLEN[7:0]	Input	Burst Length: The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
S_AXI_AWSIZE[2:0]	Input	Burst Size: Indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.
S_AXI_AWBURST[1:0]	Input	Burst Type: The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.

Table 2-11: **AXI4 Write Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
S_AXI_AWLOCK[2:0]	Input	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.
S_AXI_AWCACHE[4:0]	Input	Cache Type: Indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction.
S_AXI_AWPROT[3:0]	Input	Protection Type: Indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
S_AXI_AWQOS[3:0]	Input	Quality of Service (QoS): Sent on the write address channel for each write transaction.
S_AXI_AWREGION[3:0]	Input	Region Identifier: Sent on the write address channel for each write transaction.
S_AXI_AWUSER[m:0]	Input	Write Address Channel User
AXI4 Interface Write Address Channel: Handshake Signals for FIFO Write Interface		
S_AXI_AWVALID	Input	Write Address Valid: Indicates that valid write address and control information are available: • 1 = Address and control information available. • 0 = Address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes high.
S_AXI_AWREADY	Output	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4 Interface Write Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_AWID[m:0]	Output	Write Address ID: This signal is the identification tag for the write address group of signals.
M_AXI_AWADDR[m:0]	Output	Write Address: The write address bus gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
M_AXI_AWLEN[7:0]	Output	Burst Length: The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
M_AXI_AWSIZE[2:0]	Output	Burst Size: This signal indicates the size of each transfer in the burst. Byte lane strobes indicate exactly which byte lanes to update.
M_AXI_AWBURST[1:0]	Output	Burst Type: The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
M_AXI_AWLOCK[2:0]	Output	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.

Table 2-11: **AXI4 Write Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
M_AXI_AWCACHE[4:0]	Output	Cache Type: This signal indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction.
M_AXI_AWPROT[3:0]	Output	Protection Type: This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
M_AXI_AWQOS[3:0]	Output	Quality of Service (QoS): Sent on the write address channel for each write transaction.
M_AXI_AWREGION[3:0]	Output	Region Identifier: Sent on the write address channel for each write transaction.
M_AXI_AWUSER[m:0]	Output	Write Address Channel User
AXI4 Interface Write Address Channel: Handshake Signals for FIFO Read Interface		
M_AXI_AWVALID	Output	Write Address Valid: Indicates that valid write address and control information are available: • 1 = address and control information available • 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes high.
M_AXI_AWREADY	Input	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4 Write Address Channel FIFO: Optional Sideband Signals		
AXI_AW_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_AW_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_AW_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_AW_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AW_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.

Table 2-11: **AXI4 Write Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_AW_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_AW_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO .
AXI_AW_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2$(FIFO depth)+1
AXI_AW_UNDERFLOW	Output	Underflow: Indicates that the read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO .
AXI_AW_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2$(FIFO depth)+1
AXI_AW_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2$(FIFO depth)+1

Table 2-12 defines the AXI4 FIFO interface signals for Write Data Channel.

Table 2-12: **AXI4 Write Data Channel FIFO Interface Signals**

Name	Direction	Description
AXI4 Interface Write Data Channel: Information Signals mapped to FIFO Data Input (DIN) Bus		
S_AXI_WID[m:0]	Input	Write ID Tag: This signal is the ID tag of the write data transfer. The WID value must match the AWID value of the write transaction.
S_AXI_WDATA[m-1:0]	Input	Write Data: The write data bus can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
S_AXI_WSTRB[m/8-1:0]	Input	Write Strobes: Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)]. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid

Table 2-12: **AXI4 Write Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
S_AXI_WLAST	Input	Write Last: Indicates the last transfer in a write burst.
S_AXI_WUSER[m:0]	Input	Write Data Channel User
AXI4 Interface Write Data Channel: Handshake Signals for FIFO Write Interface		
S_AXI_WVALID	Input	Write Valid: Indicates that valid write data and strobes are available: • 1 = Write data and strobes available. • 0 = Write data and strobes not available.
S_AXI_WREADY	Output	Write Ready: Indicates that the slave can accept the write data: • 1 = Slave ready. • 0 = Slave not ready.
AXI4 Interface Write Data Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_WID[m:0]	Output	Write ID Tag: This signal is the ID tag of the write data transfer. The WID value must match the AWID value of the write transaction.
M_AXI_WDATA[m-1:0]	Output	Write Data: The write data bus can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
M_AXI_WSTRB[m/8-1:0]	Output	Write Strobes: Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)]. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid
M_AXI_WLAST	Output	Write Last: Indicates the last transfer in a write burst.
M_AXI_WUSER[m:0]	Output	Write Data Channel User
AXI4 Interface Write Data Channel: Handshake Signals for FIFO Read Interface		
M_AXI_WVALID	Output	Write valid: Indicates that valid write data and strobes are available: • 1 = Write data and strobes available . • 0 = Write data and strobes not available.
M_AXI_WREADY	Input	Write ready: Indicates that the slave can accept the write data: • 1 = Slave ready. • 0 = Slave not ready.
AXI4 Write Data Channel FIFO: Optional Sideband Signals		
AXI_W_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$

Table 2-12: **AXI4 Write Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_W_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_W_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_W_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_W_SBITERR	Output	Single-Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_W_DBITERR	Output	Double-Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_W_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO
AXI_W_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_W_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO
AXI_W_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_W_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$

Table 2-13 defines the AXI4 FIFO interface signals for Write Response Channel.

Table 2-13: **AXI4 Write Response Channel FIFO Interface Signals**

Name	Direction	Description
AXI4 Interface Write Response Channel: Information Signals Mapped to FIFO Data Output (DOUT) Bus		
S_AXI_BID[m:0]	Output	Response ID: The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.
S_AXI_BRESP[1:0]	Output	Write Response: Indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
S_AXI_BUSER[m:0]	Output	Write Response Channel User
AXI4 Interface Write Response Channel: Handshake Signals for FIFO Read Interface		
S_AXI_BVALID	Output	Write Response Valid: Indicates that a valid write response is available: • 1 = Write response available. • 0 = Write response not available.
S_AXI_BREADY	Input	Response Ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.
AXI4 Interface Write Response Channel: Information Signals Derived from FIFO Data Input (DIN) Bus		
M_AXI_BID[m:0]	Input	Response ID: The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.
M_AXI_BRESP[1:0]	Input	Write Response: Indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
M_AXI_BUSER[m:0]	Input	Write Response Channel User
AXI4 Interface Write Response Channel: Handshake Signals for FIFO Write Interface		
M_AXI_BVALID	Input	Write Response Valid: Indicates that a valid write response is available: • 1 = Write response available. • 0 = Write response not available.
M_AXI_BREADY	Output	Response Ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.
AXI4 Write Response Channel FIFO: Optional Sideband Signals		
AXI_B_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$

Table 2-13: **AXI4 Write Response Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_B_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_B_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_B_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_B_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_B_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_B_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO.
AXI_B_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_B_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.
AXI_B_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_B_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$

Read Channels

Table 2-14 defines the AXI4 FIFO interface signals for Read Address Channel.

Table 2-14: **AXI4 Read Address Channel FIFO Interface Signals**

Name	Direction	Description
AXI4 Interface Read Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus		
S_AXI_ARID[m:0]	Input	Read Address ID: This signal is the identification tag for the read address group of signals.
S_AXI_ARADDR[m:0]	Input	Read Address: The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
S_AXI_ARLEN[7:0]	Input	Burst Length: The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
S_AXI_ARSIZE[2:0]	Input	Burst Size: This signal indicates the size of each transfer in the burst.
S_AXI_ARBURST[1:0]	Input	Burst Type: The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
S_AXI_ARLOCK[2:0]	Input	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.
S_AXI_ARCACHE[4:0]	Input	Cache Type: This signal provides additional information about the cacheable characteristics of the transfer.
S_AXI_ARPROT[3:0]	Input	Protection Type: This signal provides protection unit information for the transaction.
S_AXI_ARQOS[3:0]	Input	Quality of Service (QoS): Sent on the read address channel for each read transaction.
S_AXI_ARREGION[3:0]	Input	Region Identifier: Sent on the read address channel for each read transaction.
S_AXI_ARUSER[m:0]	Input	Read Address Channel User
AXI4 Interface Read Address Channel: Handshake Signals for FIFO Write Interface		
S_AXI_ARVALID	Input	Read Address Valid: When high, indicates that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
S_AXI_ARREADY	Output	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4 Interface Read Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_ARID[m:0]	Output	Read Address ID. This signal is the identification tag for the read address group of signals.

Table 2-14: **AXI4 Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
M_AXI_ARADDR[m:0]	Output	Read Address: The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
M_AXI_ARLEN[7:0]	Output	Burst Length: The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
M_AXI_ARSIZE[2:0]	Output	Burst Size: This signal indicates the size of each transfer in the burst.
M_AXI_ARBURST[1:0]	Output	Burst Type: The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
M_AXI_ARLOCK[2:0]	Output	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.
M_AXI_ARCACHE[4:0]	Output	Cache Type: This signal provides additional information about the cacheable characteristics of the transfer.
M_AXI_ARPROT[3:0]	Output	Protection Type: This signal provides protection unit information for the transaction.
M_AXI_ARQOS[3:0]	Output	Quality of Service (QoS) signaling, sent on the read address channel for each read transaction.
M_AXI_ARREGION[3:0]	Output	Region Identifier: Sent on the read address channel for each read transaction.
M_AXI_ARUSER[m:0]	Output	Read Address Channel User
AXI4 Interface Read Address Channel: Handshake Signals for FIFO Read Interface		
M_AXI_ARVALID	Output	Read Address Valid: Indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal, **ARREADY**, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
M_AXI_ARREADY	Input	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4 Read Address Channel FIFO: Optional Sideband Signals		
AXI_AR_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$

Table 2-14: **AXI4 Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_AR_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_AR_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_AR_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AR_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AR_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_AR_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO
AXI_AR_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_AR_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.
AXI_AR_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_AR_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$

Table 2-15 defines the AXI4 FIFO interface signals for Read Data Channel.

Table 2-15: **AXI4 Read Data Channel FIFO Interface Signals**

Name	Direction	Description
AXI4 Interface Read Data Channel: Information Signals Mapped to FIFO Data Output (DOUT) Bus		
S_AXI_RID[m:0]	Output	Read ID Tag: ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.
S_AXI_RDATA[m-1:0]	Output	Read Data: Can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
S_AXI_RRESP[1:0]	Output	Read Response: Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
S_AXI_RLAST	Output	Read Last: Indicates the last transfer in a read burst.
S_AXI_RUSER[m:0]	Output	Read Data Channel User
AXI4 Interface Read Data Channel: Handshake Signals for FIFO Read Interface		
S_AXI_RVALID	Output	Read Valid: Indicates that the required read data is available and the read transfer can complete: • 1 = Read data available. • 0 = Read data not available.
S_AXI_RREADY	Input	Read Ready: Indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.
AXI4 Interface Read Data Channel: Information Signals Derived from FIFO Data Input (DIN) Bus		
M_AXI_RID[m:0]	Input	Read ID Tag: ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.
M_AXI_RDATA[m-1:0]	Input	Read Data: Can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
M_AXI_ RRESP[1:0]	Input	Read Response: Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
M_AXI_RLAST	Input	Read Last: Indicates the last transfer in a read burst.
M_AXI_RUSER[m:0]	Input	Read Data Channel User
AXI4 Interface Read Data Channel: Handshake Signals for FIFO Write Interface		
M_AXI_RVALID	Input	Read Valid: Indicates that the required read data is available and the read transfer can complete: • 1 = Read data available. • 0 = Read data not available.
M_AXI_RREADY	Output	Read Ready: Indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.

Table 2-15: **AXI4 Read Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI4 Read Data Channel FIFO: Optional Sideband Signals		
AXI_R_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_R_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_R_INJECTSBITERR	Input	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_R_INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_R_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_R_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_R_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO
AXI_R_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_R_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO
AXI_R_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_R_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$

AXI4-Lite FIFO Interface Signals

Write Channels

Table 2-16 defines the AXI4-Lite FIFO interface signals for Write Address Channel.

Table 2-16: ***AXI4-Lite Write Address Channel FIFO Interface Signals***

Name	Direction	Description
AXI4-Lite Interface Write Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus		
S_AXI_AWADDR[m:0]	Input	Write Address: Gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
S_AXI_AWPROT[3:0]	Input	Protection Type: Indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
AXI4-Lite Interface Write Address Channel: Handshake Signals for FIFO Write Interface		
S_AXI_AWVALID	Input	Write Address Valid: Indicates that valid write address and control information are available: • 1 = Address and control information available. • 0 = Address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes high.
S_AXI_AWREADY	Output	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Interface Write Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_AWADDR[m:0]	Output	Write Address: Gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
M_AXI_AWPROT[3:0]	Output	Protection Type: This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
AXI4-Lite Interface Write Address Channel: Handshake Signals for FIFO Read Interface		
M_AXI_AWVALID	Output	Write Address Valid: Indicates that valid write address and control information are available: • 1 = Address and control information available. • 0 = Address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes high.
M_AXI_AWREADY	Input	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Write Address Channel FIFO: Optional Sideband Signals		

Table 2-16: ***AXI4-Lite Write Address Channel FIFO Interface Signals (Cont'd)***

Name	Direction	Description
AXI_AW_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_AW_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_AW_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_AW_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AW_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AW_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_AW_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO
AXI_AW_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_AW_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.

Table 2-16: ***AXI4-Lite Write Address Channel FIFO Interface Signals (Cont'd)***

Name	Direction	Description
AXI_AW_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_AW_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$

Table 2-17 defines the AXI4-Lite FIFO interface signals for Write Data Channel.

Table 2-17: **AXI4-Lite Write Data Channel FIFO Interface Signals**

Name	Direction	Description
AXI4-Lite Interface Write Data Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus		
S_AXI_WDATA[m-1:0]	Input	Write Data: Can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
S_AXI_WSTRB[m/8-1:0]	Input	Write Strobes: Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)]. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid
AXI4-Lite Interface Write Data Channel: Handshake Signals for FIFO Write Interface		
S_AXI_WVALID	Input	Write Valid: Indicates that valid write data and strobes are available: • 1 = Write data and strobes available. • 0 = Write data and strobes not available.
S_AXI_WREADY	Output	Write Ready: Indicates that the slave can accept the write data: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Interface Write Data Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_WDATA[m-1:0]	Output	Write Data: Can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
M_AXI_WSTRB[m/8-1:0]	Output	Write Strobes: Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)]. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid

Table 2-17: **AXI4-Lite Write Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI4-Lite Interface Write Data Channel: Handshake Signals for FIFO Read Interface		
M_AXI_WVALID	Output	Write Valid: Indicates that valid write data and strobes are available: • 1 = Write data and strobes available. • 0 = Write data and strobes not available.
M_AXI_WREADY	Input	Write Ready: Indicates that the slave can accept the write data: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Write Data Channel FIFO: Optional Sideband Signals		
AXI_W_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2$(FIFO depth)-1
AXI_W_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2$(FIFO depth)-1
AXI_W_INJECTSBITERR	Input	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_W_INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_W_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_W_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_W_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO.
AXI_W_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2$(FIFO depth)+1
AXI_W_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.

Table 2-17: **AXI4-Lite Write Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_W_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_W_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$

Table 2-18 defines the AXI4-Lite FIFO interface signals for Write Response Channel.

Table 2-18: **AXI4-Lite Write Response Channel FIFO Interface Signals**

Name	Direction	Description
AXI4-Lite Interface Write Response Channel: Information Signals Mapped to FIFO Data Output (DOUT) Bus		
S_AXI_BRESP[1:0]	Output	Write Response: Indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
AXI4-Lite Interface Write Response Channel: Handshake Signals for FIFO Read Interface		
S_AXI_BVALID	Output	Write Response Valid: Indicates that a valid write response is available: • 1 = Write response available. • 0 = Write response not available.
S_AXI_BREADY	Input	Response Ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.
AXI4-Lite Interface Write Response Channel: Information Signals Derived from FIFO Data Input (DIN) Bus		
M_AXI_BRESP[1:0]	Input	Write response: Indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
AXI4-Lite Interface Write Response Channel: Handshake Signals for FIFO Write Interface		
M_AXI_BVALID	Input	Write response valid: Indicates that a valid write response is available: • 1 = Write response available. • 0 = Write response not available.
M_AXI_BREADY	Output	Response ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.
AXI4-Lite Write Response Channel FIFO: Optional Sideband Signals		

Table 2-18: **AXI4-Lite Write Response Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_B_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2$(FIFO depth)-1
AXI_B_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. D is than log2(FIFO depth)-1
AXI_B_INJECTSBITERR	Input	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_B_INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_B_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_B_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_B_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO.
AXI_B_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2$(FIFO depth)+1
AXI_B_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.
AXI_B_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2$(FIFO depth)+1
AXI_B_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2$(FIFO depth)+1

Read Channels

Table 2-19 defines the AXI4-Lite FIFO interface signals for Read Address Channel.

Table 2-19: **AXI4-Lite Read Address Channel FIFO Interface Signals**

Name	Direction	Description
AXI4-Lite Interface Read Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus		
S_AXI_ARADDR[m:0]	Input	Read Address: The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
S_AXI_ARPROT[3:0]	Input	Protection Type: This signal provides protection unit information for the transaction.
AXI4-Lite Interface Read Address Channel: Handshake Signals for FIFO Write Interface		
S_AXI_ARVALID	Input	Read Address Valid: When high, indicates that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
S_AXI_ARREADY	Output	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Interface Read Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_ARADDR[m:0]	Output	Read Address: The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
M_AXI_ARPROT[3:0]	Output	Protection Type: This signal provides protection unit information for the transaction.
AXI4-Lite Interface Read Address Channel: Handshake Signals for FIFO Read Interface		
M_AXI_ARVALID	Output	Read Address Valid: WHen high, indicates that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
M_AXI_ARREADY	Input	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Read Address Channel FIFO: Optional Sideband Signals		

Table 2-19: **AXI4-Lite Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_AR_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2$(FIFO depth)-1
AXI_AR_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2$(FIFO depth)-1
AXI_AR_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single-bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_AR_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double-bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AR_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AR_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_AR_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO.
AXI_AR_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2$(FIFO depth)+1
AXI_AR_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.

Table 2-19: **AXI4-Lite Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_AR_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_AR_DATA_COUNT[D:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$

Table 2-20 defines the AXI4-Lite FIFO interface signals for Write Data Channel.

Table 2-20: **AXI4-Lite Read Data Channel FIFO Interface Signals**

Name	Direction	Description
AXI4-Lite Interface Read Data Channel: Information Signals Mapped to FIFO Data Output (DOUT) Bus		
S_AXI_RDATA[m-1:0]	Output	Read Data: The read data bus can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
S_AXI_RRESP[1:0]	Output	Read Response: Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
AXI4-Lite Interface Read Data Channel: Handshake Signals for FIFO Read Interface		
S_AXI_RVALID	Output	Read Valid: Indicates that the required read data is available and the read transfer can complete: • 1 = Read data available. • 0 = Read data not available.
S_AXI_RREADY	Input	Read Ready: indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.
AXI4-Lite Interface Read Data Channel: Information Signals Derived from FIFO Data Input (DIN) Bus		
M_AXI_RDATA[m-1:0]	Input	Read Data: The read data bus can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
M_AXI_ RRESP[1:0]	Input	Read Response: Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
AXI4-Lite Interface Read Data Channel: Handshake Signals for FIFO Write Interface		
M_AXI_RVALID	Input	Read Valid: Indicates that the required read data is available and the read transfer can complete: • 1 = Read data available. • 0 = Read data not available.

Table 2-20: **AXI4-Lite Read Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
M_AXI_RREADY	Output	Read ready: Indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.
AXI4-Lite Read Data Channel FIFO: Optional Sideband Signals		
AXI_R_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_R_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_R_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_R_INJECTDBITERR	Input	Inject DOuble-Bit Error. Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_R_SBITERR	Output	Single-Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_R_DBITERR	Output	Double-Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_R_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO.
AXI_R_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure the user never overflows the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_R_UNDERFLOW	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.

Table 2-20: **AXI4-Lite Read Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_R_RD_DATA_COUNT[*D*: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. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
AXI_R_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$

Chapter 3

Generating the Native FIFO Core

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

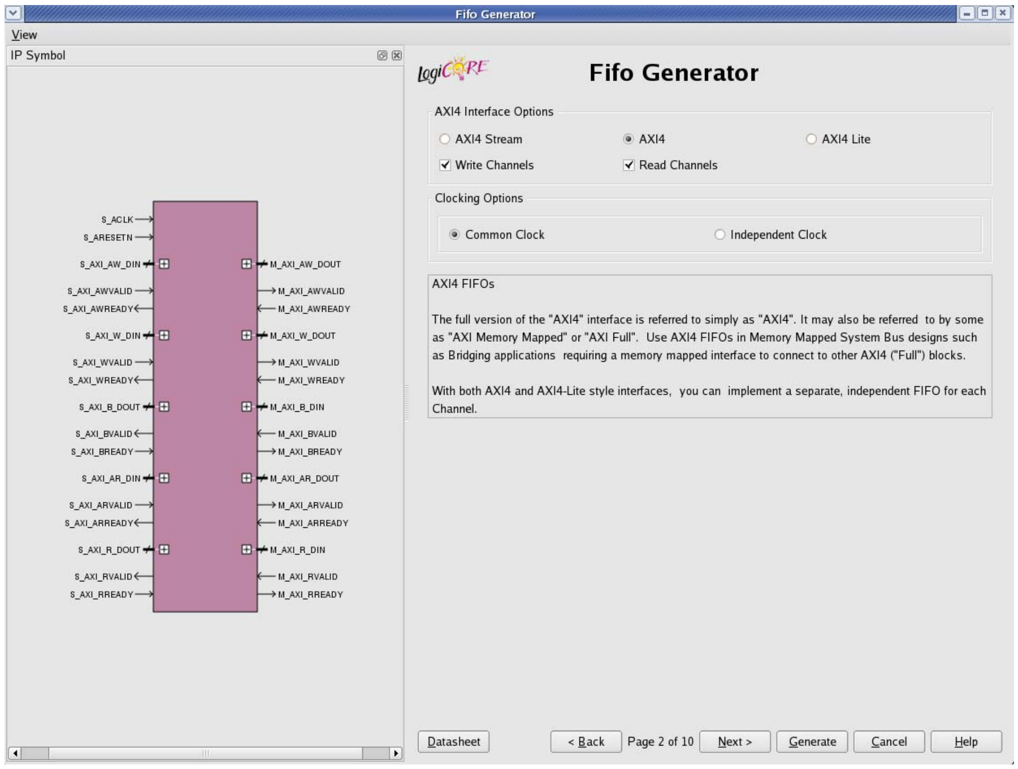
CORE Generator Graphical User Interface

The Native FIFO Interface GUI includes seven configuration screens.

- Interface Type
- FIFO Implementation
- Performance Options and Data Port Parameters
- Optional Flags, Handshaking, and Initialization
- Initialization and Programmable Flags
- Data Count
- Summary

Interface Type

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



Fifo Generator

AXI4 Interface Options

AXI4 Stream [✓ AXI4] AXI4 Lite

[✓ Write Channels] [✓ Read Channels]

Clocking Options

[✓ Common Clock] Independent Clock

AXI4 FIFOs

The full version of the "AXI4" interface is referred to simply as "AXI4". It may also be referred to by some as "AXI Memory Mapped" or "AXI Full". Use AXI4 FIFOs in Memory Mapped System Bus designs such as Bridging applications requiring a memory mapped interface to connect to other AXI4 ("Full") blocks.

With both AXI4 and AXI4-Lite style interfaces, you can implement a separate, independent FIFO for each Channel.

Figure 3-1: **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 "_".

Interface Type

- Native

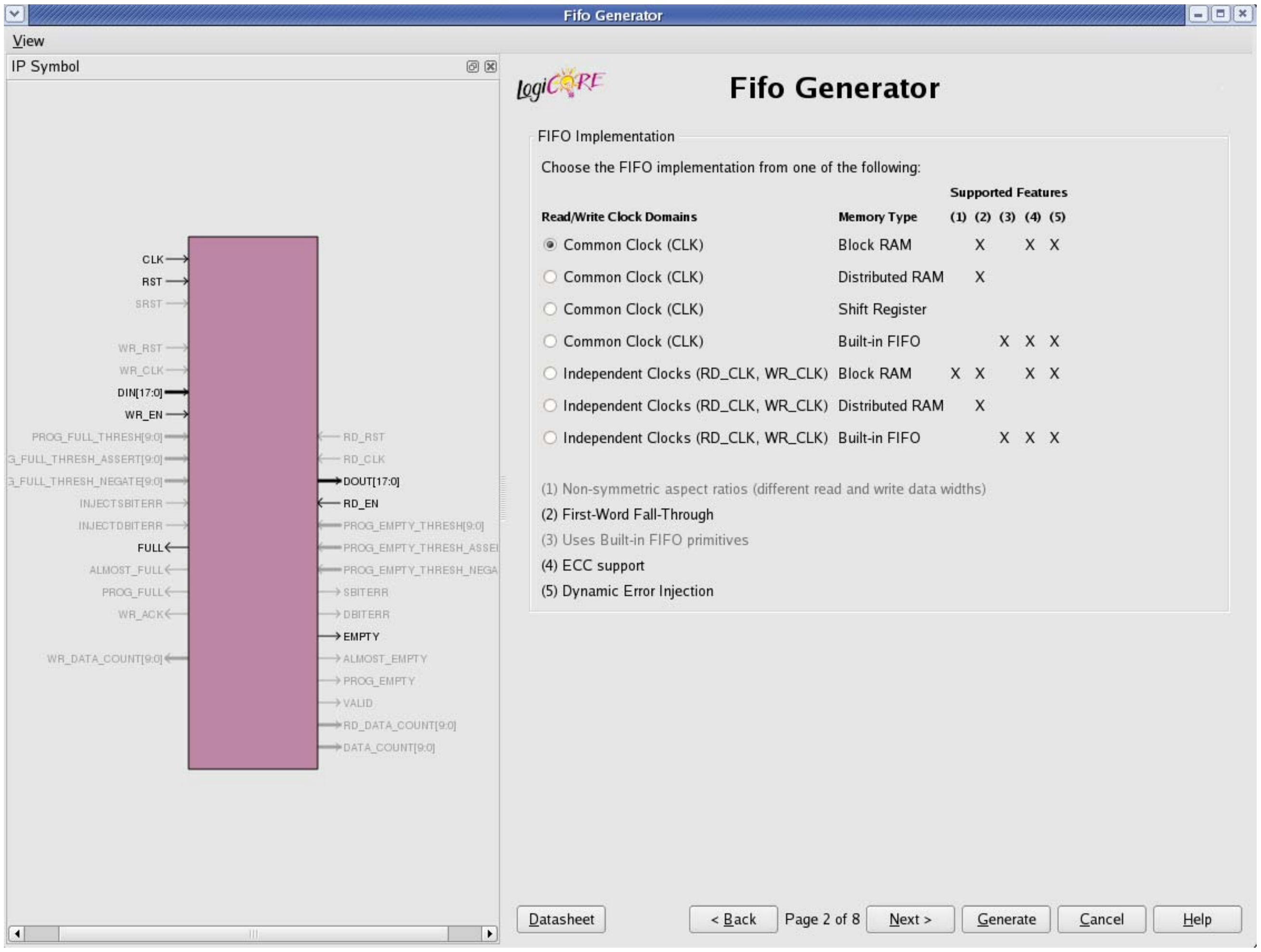
  Implements a Native FIFO.

- AXI4

  Implements an AXI4 FIFO in First-Word-Fall-Through mode.

FIFO Implementation

The FIFO Implementation screen is used to define the configuration options for the core.



FIFO Implementation

Choose the FIFO implementation from one of the following:

Read/Write Clock Domains	Memory Type	Supported Features (1)	(2)	(3)	(4)	(5)
● Common Clock (CLK)	Block RAM		X		X	X
○ Common Clock (CLK)	Distributed RAM		X			
○ Common Clock (CLK)	Shift Register					
○ Common Clock (CLK)	Built-in FIFO			X	X	X
○ Independent Clocks (RD_CLK, WR_CLK)	Block RAM	X	X		X	X
○ Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		X			
○ Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO			X	X	X

(1) Non-symmetric aspect ratios (different read and write data widths)
(2) First-Word Fall-Through
(3) Uses Built-in FIFO primitives
(4) ECC support
(5) Dynamic Error Injection

Figure 3-2: **FIFO Implementation Screen**

This screen 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 details, see Common Clock FIFO: Block RAM and Distributed RAM, page 103. This implementation optionally supports first-word-fall-through (selectable in the second GUI screen, shown in Figure 3-3).

Common Clock (CLK), Distributed RAM

For details, see Common Clock FIFO: Block RAM and Distributed RAM, page 103. This implementation optionally supports first-word-fall-through (selectable in the second GUI screen, shown in Figure 3-3).

Common Clock (CLK), Shift Register

For details, see Common Clock FIFO: Shift Registers, page 104. This implementation is only available in Virtex-4 FPGA and newer architectures.

Common Clock (CLK), Built-in FIFO

For details, see Common Clock: Built-in FIFO, page 103. This implementation is only available when using the Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA architectures. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, shown in Figure 3-3).

Independent Clocks (RD_CLK, WR_CLK), Block RAM

For details, see Independent Clocks: Block RAM and Distributed RAM, page 99. This implementation optionally supports asymmetric read/write ports and first-word fall-through (selectable in the second GUI screen, shown in Figure 3-3).

Independent Clocks (RD_CLK, WR_CLK), Distributed RAM

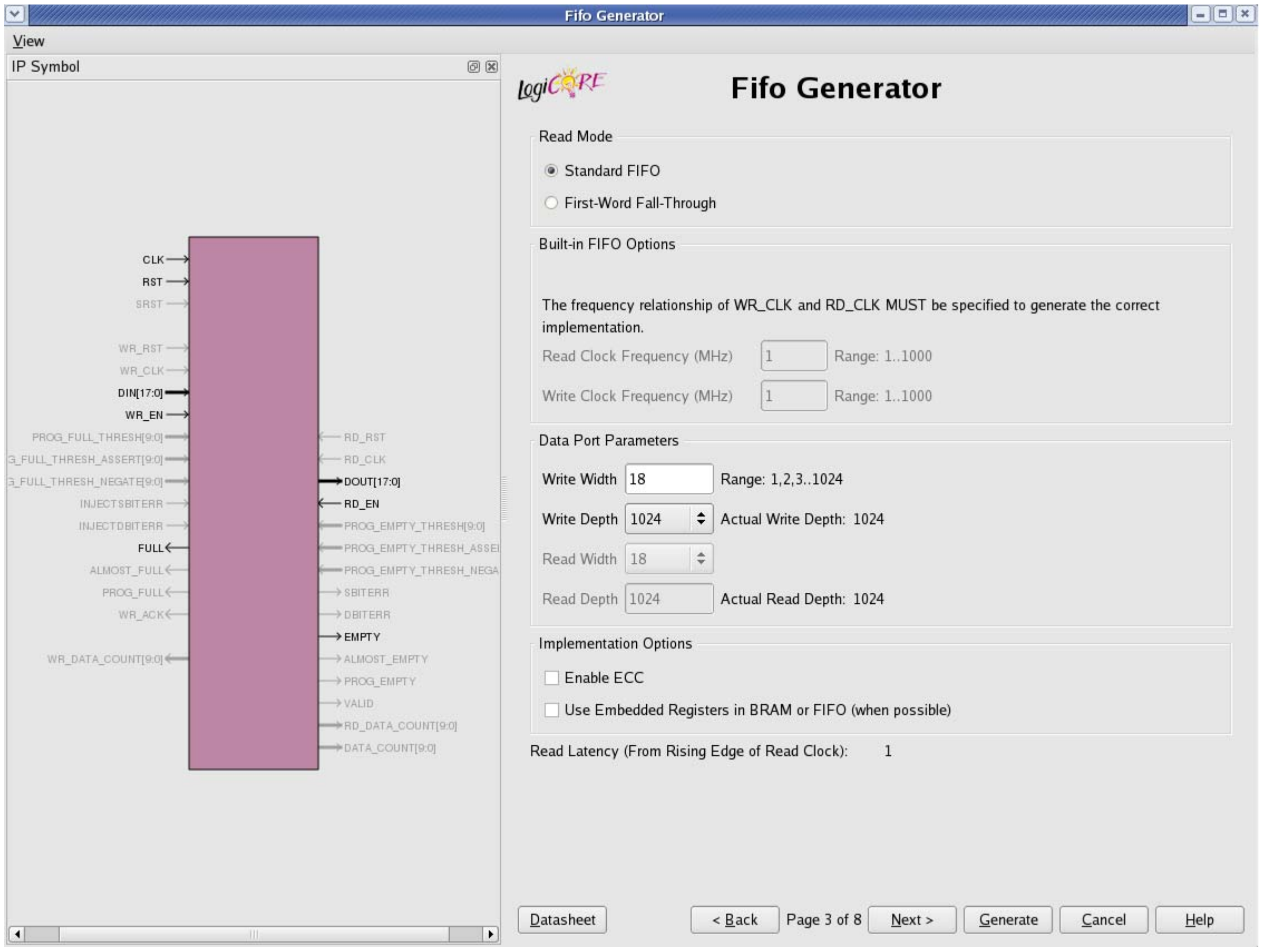
For more information, see Independent Clocks: Block RAM and Distributed RAM, page 99. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, shown in Figure 3-3).

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

For more information, see Independent Clocks: Built-in FIFO, page 101. This implementation is only available when using Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA architectures. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, shown in Figure 3-3).

Performance Options and Data Port Parameters

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



Fifo Generator

Read Mode

[✓ Standard FIFO] First-Word Fall-Through

Built-in FIFO Options

The frequency relationship of WR_CLK and RD_CLK MUST be specified to generate the correct implementation.

Read Clock Frequency (MHz) 1 Range: 1..1000

Write Clock Frequency (MHz) 1 Range: 1..1000

Data Port Parameters

Write Width 18 Range: 1,2,3..1024

Write Depth 1024 Actual Write Depth: 1024

Read Width 18

Read Depth 1024 Actual Read Depth: 1024

Implementation Options

☐ Enable ECC

☐ Use Embedded Registers in BRAM or FIFO (when possible)

Read Latency (From Rising Edge of Read Clock): 1

Figure 3-3: **Performance Options and Data Port Parameters Screen**

Read Mode

Available only when block RAM or distributed RAM FIFOs are selected. Support for built-in FIFOs is only available for Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA implementations.

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 FIFO Read Operation, page 107.

Built-in FIFO Options

Read/Write Clock Frequencies

The Read Clock Frequency and Write Clock Frequency fields can be any integer from 1 to 1000. They are used to determine the optimal implementation of the domain-crossing logic in the core. This option is only available for built-in FIFOs with independent clocks. If the desired frequency is not within the allowable range, scale the read and write clock frequencies so that they fit within the valid range, while maintaining their ratio relationship.

Important: It is critical that this information is entered and accurate. If this information is not provided, it can result in a sub-optimal solution with incorrect core behavior.

Data Port Parameters

Write Width

For Virtex-4 FPGA Built-in FIFO macro, the valid range is 4, 9, 18 and 36. For other memory type configurations, the valid range is 1 to 1024.

Write Depth

For Virtex-4 FPGA Built-in FIFO macro, the valid range automatically varies based on write width selection. For Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA Built-in FIFO macro, the valid range is 512 to 4194304. Only depths with powers of 2 are allowed.

For non Built-in FIFO, the valid range is 1 to 4194304. Only depths with powers of 2 are allowed.

Read Width

Available only if independent clocks configuration with block RAM is selected. Valid range must comply with asymmetric port rules. See Non-symmetric Aspect Ratios, page 121.

Read Depth

Automatically calculated based on Write Width, Write Depth, and Read Width.

Implementation Options

Error Correction Checking in Block RAM or Built-in FIFO

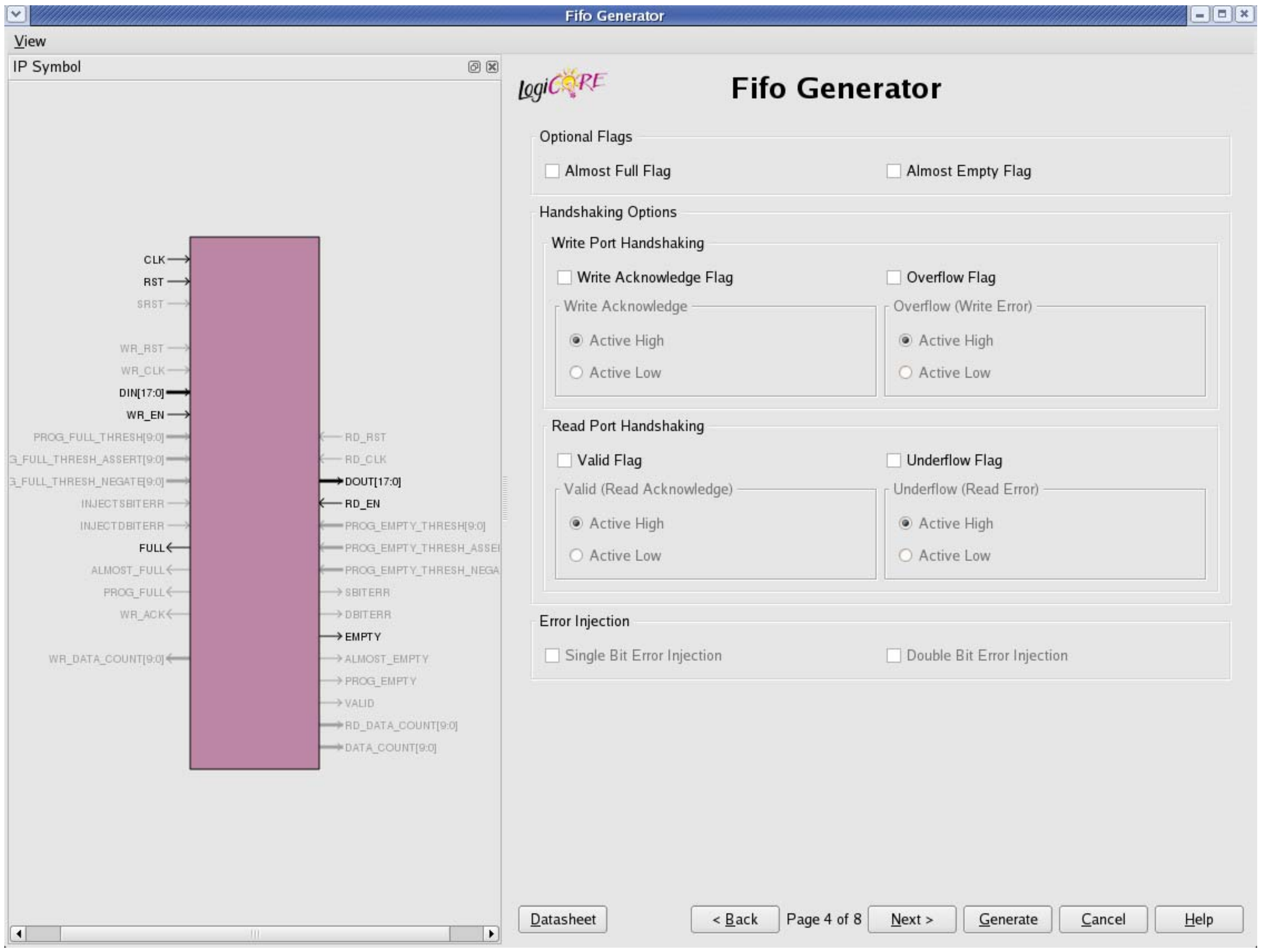
The Error Correction Checking (ECC) feature enables built-in error correction in the Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA block RAM and built-in FIFO macros. When this feature is enabled, the block RAM or built-in FIFO is set to the full ECC mode, where both the encoder and decoder are enabled.

Use Embedded Registers in Block RAM or FIFO

The block RAM macros available in Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA, as well as built-in FIFO macros available in Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA, have built-in embedded registers that can be used to pipeline data and improve macro timing. This option enables users to add one pipeline stage to the output of the FIFO and take advantage of the available embedded registers; however, the ability to reset the data output of the Virtex-5 FPGA built-in FIFO is disabled when this feature is used. For built-in FIFOs, this feature is only supported for synchronous FIFO configurations that have only 1 FIFO macro in depth. See Embedded Registers in Block RAM and FIFO Macros (Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs), page 124.

Optional Flags, Handshaking, and Initialization

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



Fifo Generator

Optional Flags

☐ Almost Full Flag ☐ Almost Empty Flag

Handshaking Options

Write Port Handshaking

☐ Write Acknowledge Flag ☐ Overflow Flag

Write Acknowledge: [✓ Active High] Active Low

Overflow (Write Error): [✓ Active High] Active Low

Read Port Handshaking

☐ Valid Flag ☐ Underflow Flag

Valid (Read Acknowledge): [✓ Active High] Active Low

Underflow (Read Error): [✓ Active High] Active Low

Error Injection

☐ Single Bit Error Injection ☐ Double Bit Error Injection

Figure 3-4: **Optional Flags, Handshaking, and Error Injection Options Screen**

Optional Flags

Almost Full Flag

Available in all FIFO implementations except those using Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA 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 Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA built-in FIFOs. Generates an output port that indicates the FIFO is almost empty (only one more word can be read).

Handshaking Options

Write Port Handshaking

Write Acknowledge

Generates write acknowledge flag which reports the success 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).

Error Injection

Single Bit Error Injection

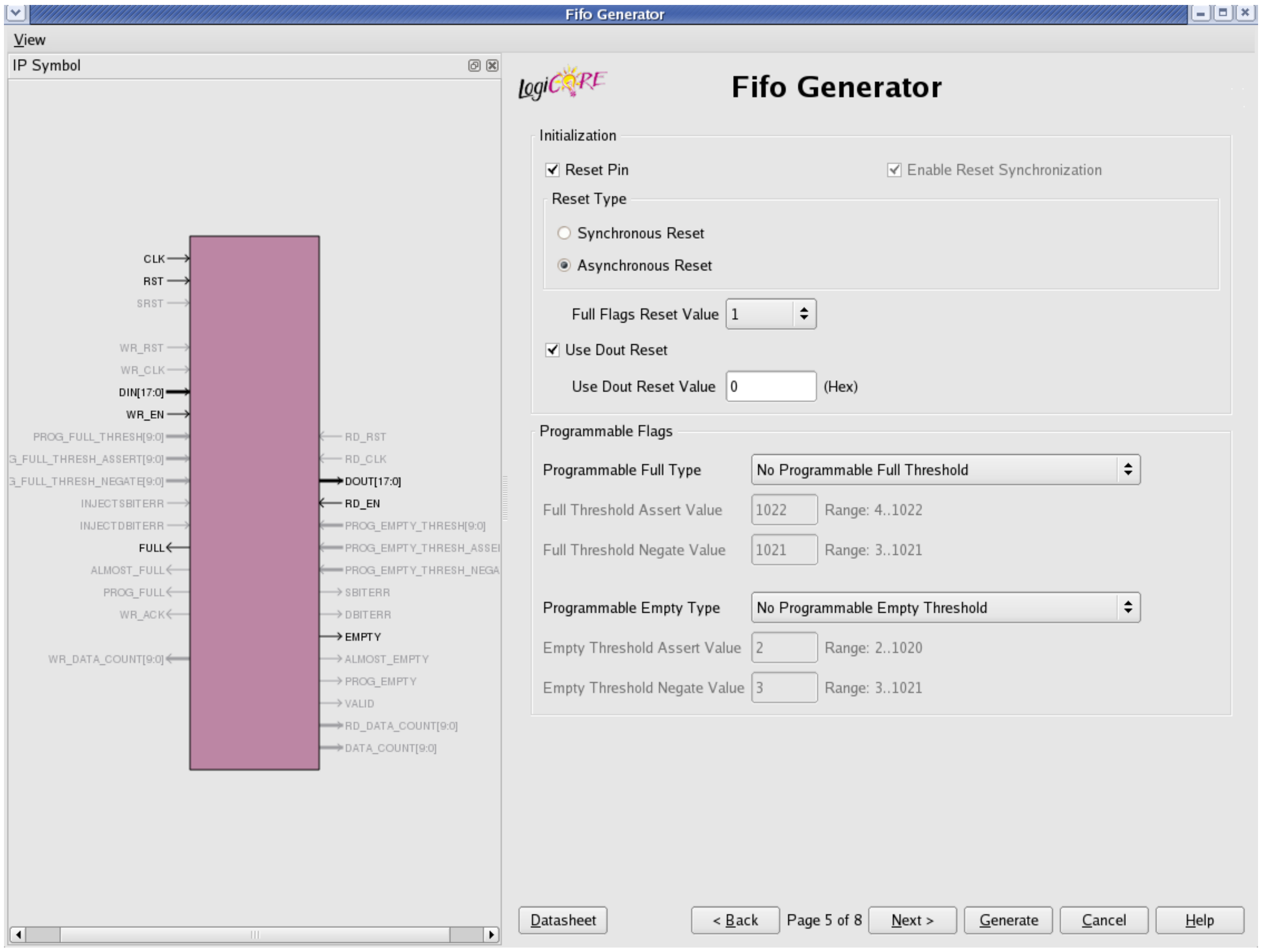
Available only in Virtex-6 FPGAs for both the common and independent clock block RAM or built-in FIFOs, with ECC option enabled. Generates an input port to inject a single bit error on write and an output port that indicates a single bit error occurred.

Double Bit Error Injection

Available only in Virtex-6 FPGAs for both the common and independent clock block RAM or built-in FIFOs, with ECC option enabled. Generates an input port to inject a double bit error on write and an output port that indicates a double bit error occurred.

Initialization and Programmable Flags

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



Initialization

✓ Reset Pin ✓ Enable Reset Synchronization

Reset Type

Synchronous Reset [✓ Asynchronous Reset]

Full Flags Reset Value: 1

✓ Use Dout Reset

Use Dout Reset Value: 0 (Hex)

Programmable Flags

Programmable Full Type	No Programmable Full Threshold	
Full Threshold Assert Value	1022	Range: 4..1022
Full Threshold Negate Value	1021	Range: 3..1021
Programmable Empty Type	No Programmable Empty Threshold	
Empty Threshold Assert Value	2	Range: 2..1020
Empty Threshold Negate Value	3	Range: 3..1021

Figure 3-5: **Programmable Flags and Reset Screen**

Initialization

Reset Pin

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

- Enable Reset Synchronization. Optional selection only available for independent clock block RAM or distributed RAM FIFOs. When unchecked, WR_RST/RD_RST is available. See Reset Behavior in Chapter 5 for details.
- Asynchronous Reset. Optional selection for a common-clock FIFO implemented using distributed or block RAM.
- Synchronous Reset. Optional selection for a a common-clock FIFO implemented using distributed or block RAM.

Full Flags Reset Value

For block RAM, distributed RAM, and shift register configurations, the user can choose the reset value of the full flags (`PROG_FULL`, `ALMOST_FULL`, and `FULL`) during reset.

Use Dout Reset

Available in Virtex-4 FPGA or newer architectures for all implementations using block RAM, distributed RAM, shift register or Virtex-6 common clock built-in with embedded register option. Only available if a reset pin option is selected. If selected, the DOUT output of the FIFO will reset to the defined DOUT Reset Value (below) when the reset is asserted. If not selected, the DOUT output of the FIFO will not be effected by the assertion of reset, and DOUT will hold its previous value.

Disabling this feature for Spartan®-3 devices may improve timing for the distributed RAM and shift register FIFO.

Use Dout Reset Value

Available only when Use Dout Reset is selected, this field indicates the hexidecimal value asserted on the output of the FIFO when RST (SRST) is asserted. See Appendix C, DOUT Reset Value Timing for the timing diagrams for different configurations.

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 varies depending on the 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. 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. 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

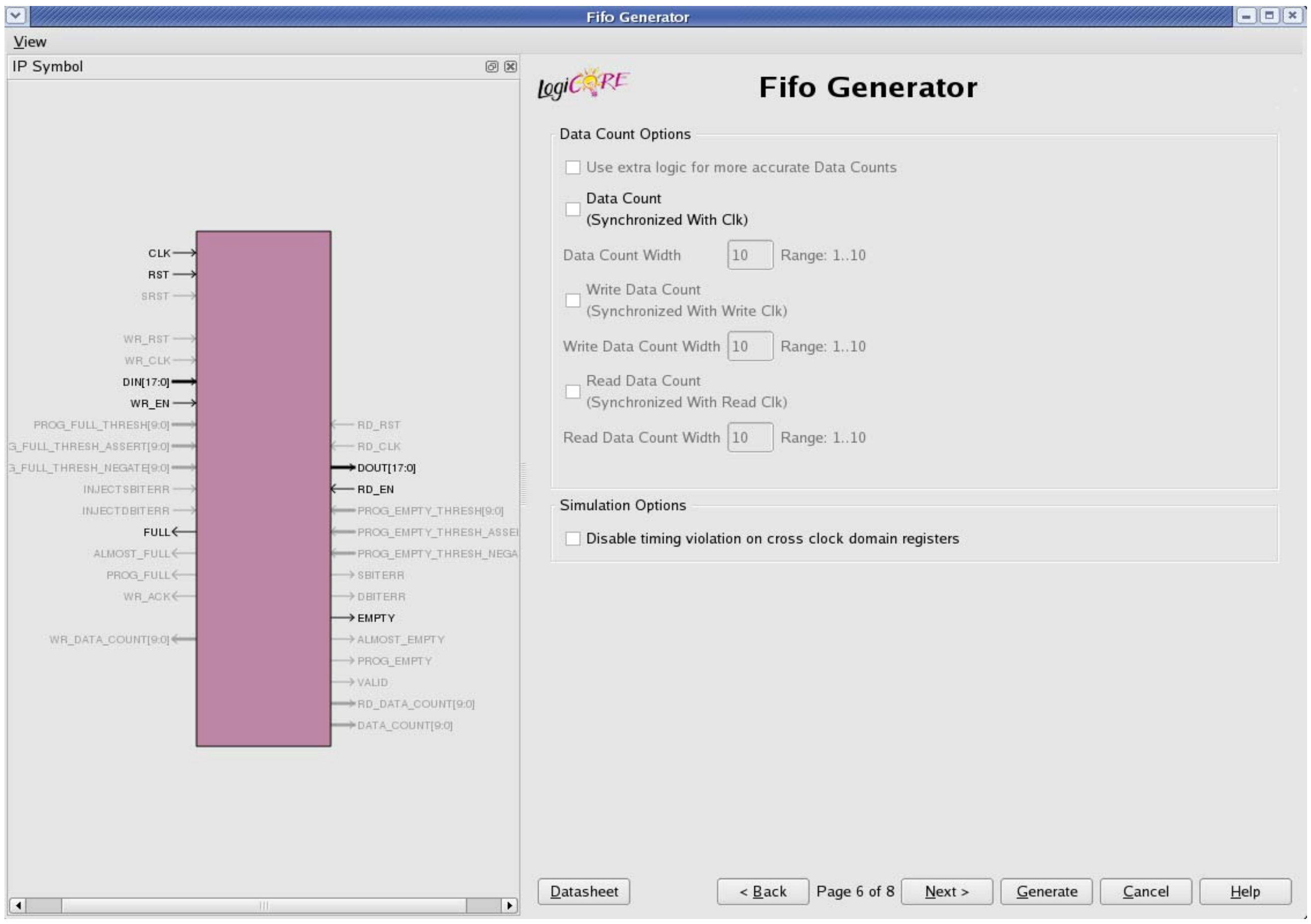
Available when Programmable Empty with Single or Multiple Threshold Constants is selected. Enter a user-defined value. The valid range for this threshold is provided in the GUI. When using a single threshold constant, only the assert value is used.

Empty Threshold Negate Value

Available when Programmable Empty with Multiple Threshold Constants is selected. Enter a user-defined value. The valid range for this threshold is provided in the GUI.

Data Count

Use this screen to set data count options.



Fifo Generator

Data Count Options

- [ ] Use extra logic for more accurate Data Counts
- [ ] Data Count (Synchronized With Clk)

Data Count Width 10 Range: 1..10

- [ ] Write Data Count (Synchronized With Write Clk)

Write Data Count Width 10 Range: 1..10

- [ ] Read Data Count (Synchronized With Read Clk)

Read Data Count Width 10 Range: 1..10

Simulation Options

- [ ] Disable timing violation on cross clock domain registers

Figure 3-6: **Data Count Screen**

Data Count Options

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. This feature is always enabled for common clock FIFOs with block RAM or distributed RAM and when using first-word-fall-through. See First-Word Fall-Through Data Count, page 118 for details.

Data Count (Synchronized With Clk)

Available when a common clock FIFO with block RAM, distributed RAM, or shift registers is selected.

Data Count Width

Available when Data Count is selected. Valid range is from 1 to $\log_2$ (input depth).

Write Data Count (Synchronized with Write Clk)

Available when an independent clocks FIFO with block RAM or distributed RAM is selected.

Write Data Count Width

Available when Write Data Count is selected. Valid range is from 1 to $\log_2$ (input depth).

Read Data Count (Synchronized with Read Clk)

Available when an independent clocks FIFO with block RAM or distributed RAM is selected.

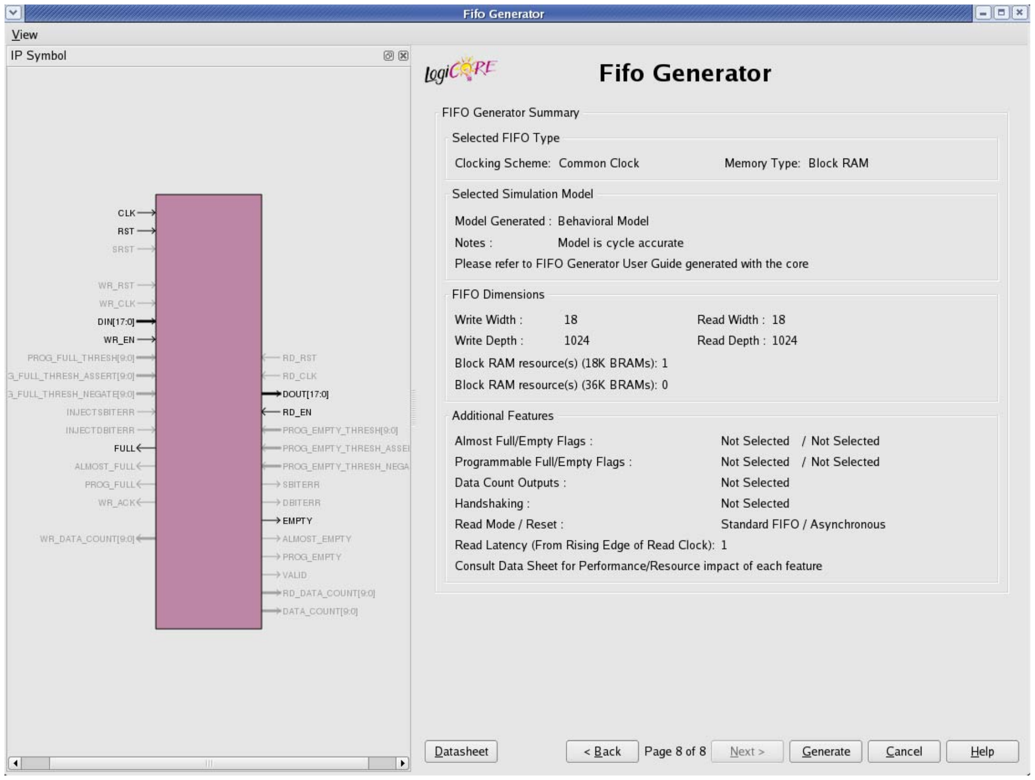
Read Data Count Width

Available when Read Data Count is selected. Valid range is from 1 to $\log_2$ (output depth).

Summary

This screen displays a summary of the selected FIFO options, including the FIFO type, FIFO dimensions, and the status of any additional features selected. In the Additional Features section, most features display either *Not Selected* (if unused), or *Selected* (if 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 screen three of the FIFO GUI.



FIFO Generator Summary

Selected FIFO Type

Clocking Scheme: Common Clock   Memory Type: Block RAM

Selected Simulation Model

Model Generated : Behavioral Model

Notes : Model is cycle accurate

Please refer to FIFO Generator User Guide generated with the core

FIFO Dimensions

Write Width :	18	Read Width :	18
Write Depth :	1024	Read Depth :	1024

Block RAM resource(s) (18K BRAMs): 1

Block RAM resource(s) (36K BRAMs): 0

Additional Features

Almost Full/Empty Flags :	Not Selected / Not Selected
Programmable Full/Empty Flags :	Not Selected / Not Selected
Data Count Outputs :	Not Selected
Handshaking :	Not Selected
Read Mode / Reset :	Standard FIFO / Asynchronous
Read Latency (From Rising Edge of Read Clock):	1

Consult Data Sheet for Performance/Resource impact of each feature

Figure 3-7: **Summary Screen**

Chapter 4

Generating the AXI4 FIFO Core

This chapter contains information and instructions for using the Xilinx CORE Generator system to customize the AXI4 FIFO Generator.

CORE Generator Graphical User Interface

For AXI4, the FIFO Generator GUI includes five configuration GUI pages:

- Interface Selection
- Width Calculation
- FIFO Configuration
- Common Page for FIFO Configuration

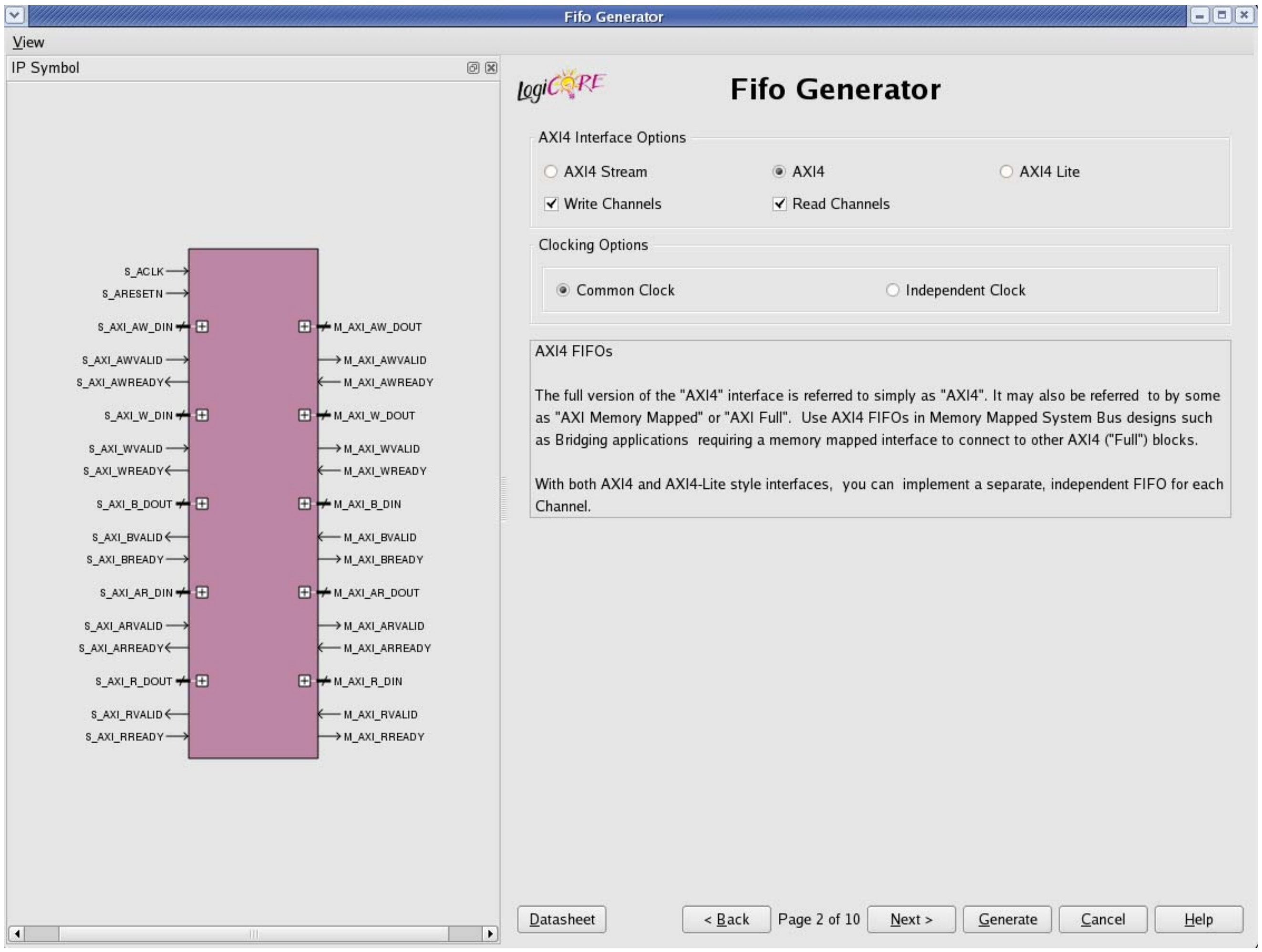
  For AXI4 and AXI4-Lite interfaces, FIFO Generator provides a separate page to configure each FIFO channel. For more details, see Easy Integration of Independent FIFOs for Read and Write Channels in Chapter 2.
- Summary

  The configuration settings specified on the Page 2 of the GUI is applied to all selected Channels of the AXI4 or AXI4-Lite interfaces

More details on these customization GUI pages are provided in the following sections.

AXI4 Interface Selection

Figure 4-1 shows the AXI4 interface selection screen.



AXI4 Interface Options

AXI4 Stream [✓ AXI4] AXI4 Lite

[✓ Write Channels] [✓ Read Channels]

Clocking Options

[✓ Common Clock] Independent Clock

AXI4 FIFOs

The full version of the "AXI4" interface is referred to simply as "AXI4". It may also be referred to by some as "AXI Memory Mapped" or "AXI Full". Use AXI4 FIFOs in Memory Mapped System Bus designs such as Bridging applications requiring a memory mapped interface to connect to other AXI4 ("Full") blocks.

With both AXI4 and AXI4-Lite style interfaces, you can implement a separate, independent FIFO for each Channel.

Figure 4-1: **AXI4 Interface Selection Screen**

AXI4 Interface Options

Three AXI4 interface styles are available: AXI4-Stream, AXI4 and AXI4-Lite.

Clocking Options

FIFOs may be configured with either independent or common clock domains for Write and Read operations.

The Independent Clock configuration 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 a single clock by selecting the Common Clocks option.

For more details on Common Clock FIFO, see Common Clock FIFO: Block RAM and Distributed RAM in Chapter 5.

For more details on Independent Clock FIFO, see Independent Clocks: Block RAM and Distributed RAM in Chapter 5.

Performing Writes with Slave Clock Enable

The Slave Interface Clock Enable allows the AXI4 Master to operate at fractional rates of AXI4 Slave Interface (or Write side) of FIFO. The above timing diagram shows the AXI4 Master operating at half the frequency of the FIFO AXI4 Slave interface. The Clock Enable in this case is Single Clock Wide, Synchronous and occurs once in every two clock cycles of the AXI4 Slave clock.

Performing Reads with Master Clock Enable

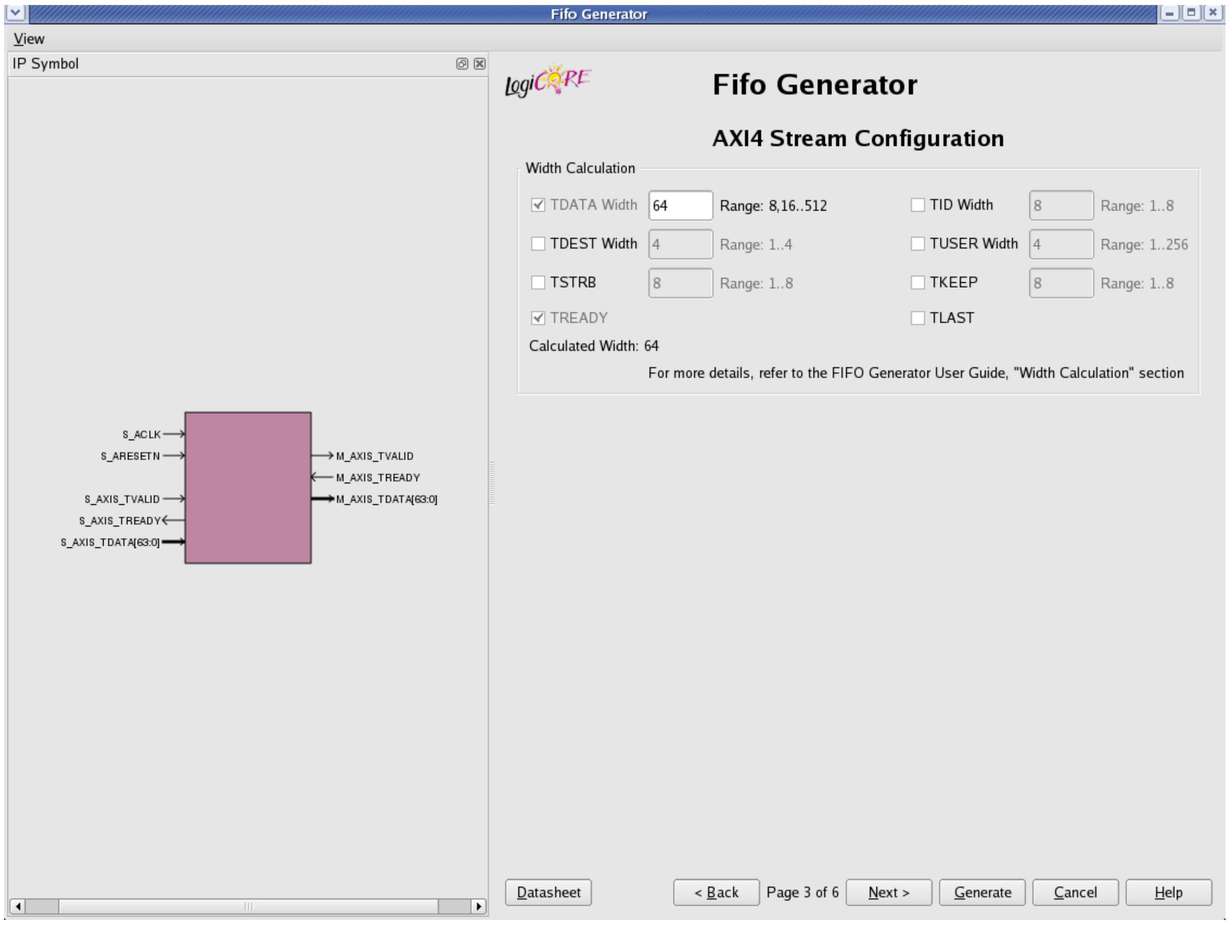
The Master Interface Clock Enable allows AXI4 Slave to operate at fractional rates of AXI4 Master Interface (or Read side) of the FIFO. The above timing diagram shows the AXI4 Slave operating at half the frequency of the FIFO AXI4 Master Interface. The Clock Enable in this case is Single Clock Wide, Synchronous and occurs once in every two clock cycles of the FIFO AXI4 Slave clock. the FIFO.

Width Calculation

The AXI4 FIFO Width is determined by aggregating all of the channel information signals in a channel. The channel information signals for AXI4-Stream, AXI4 and AXI4-Lite interfaces are listed in Table 4-1 and Table 4-2. GUI screens are available for configuring:

- AXI4-Stream Width Calculation
- AXI4 Width Calculation
- AXI4-Lite Width Calculation

AXI4-Stream Width Calculation



Fifo Generator

AXI4 Stream Configuration

Width Calculation

[✓] TDATA Width	64	Range: 8,16..512	[ ] TID Width	8	Range: 1..8
[ ] TDEST Width	4	Range: 1..4	[ ] TUSER Width	4	Range: 1..256
[ ] TSTRB	8	Range: 1..8	[ ] TKEEP	8	Range: 1..8
[✓] TREADY			[ ] TLAST		

Calculated Width: 64

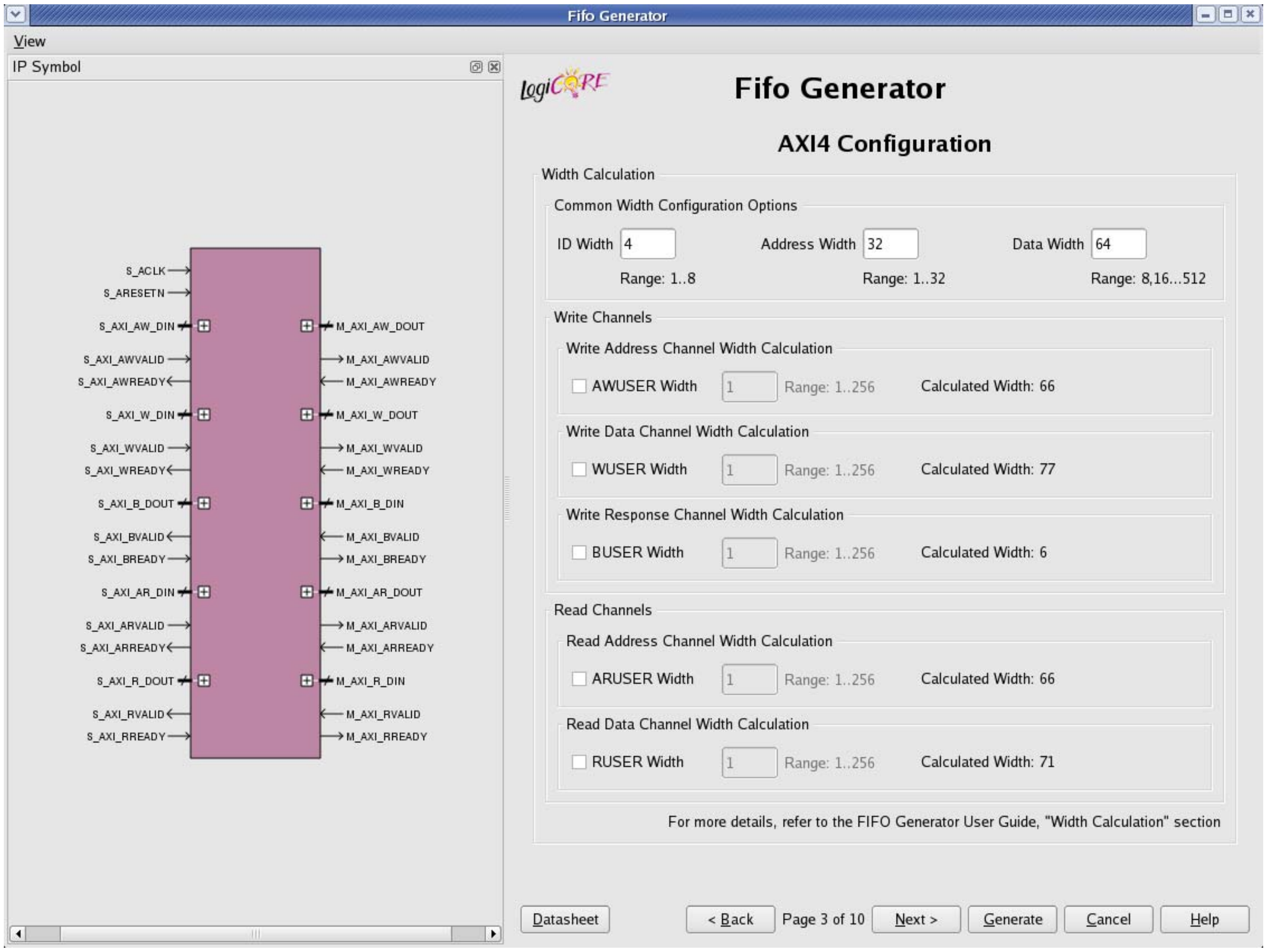
For more details, refer to the FIFO Generator User Guide, "Width Calculation" section

Figure 4-2: **AXI4-Stream Width Calculation Screen**

CORE Generator AXI4-Stream FIFO allows user to configure widths for TDATA, TUSER, TID and TDEST signals. For TKEEP and TSTRB signals the width is determined by the configured TDATA width and is internally calculated by using the equation (TDATA Width)/8.

For all the selected signals, the AXI4-Stream FIFO width is determined by summing up the widths of all the selected signals.

AXI4 Width Calculation



Fifo Generator

AXI4 Configuration

Width Calculation

Common Width Configuration Options

ID Width 4 Range: 1..8
Address Width 32 Range: 1..32
Data Width 64 Range: 8,16...512

Write Channels

Write Address Channel Width Calculation
AWUSER Width 1 Range: 1..256 Calculated Width: 66

Write Data Channel Width Calculation
WUSER Width 1 Range: 1..256 Calculated Width: 77

Write Response Channel Width Calculation
BUSER Width 1 Range: 1..256 Calculated Width: 6

Read Channels

Read Address Channel Width Calculation
ARUSER Width 1 Range: 1..256 Calculated Width: 66

Read Data Channel Width Calculation
RUSER Width 1 Range: 1..256 Calculated Width: 71

For more details, refer to the FIFO Generator User Guide, "Width Calculation" section

Figure 4-3: **AXI4 Width Calculation Screen**

The AXI4 FIFO widths can be configured for ID, ADDR, DATA and USER signals. ID Width is applied to all channels in the AXI4 interface. When both write and read channels are selected, the same ADDR and DATA widths are applied to both the write channels and read channels. The user signal is the only optional signal for the AXI4 FIFO and can be independently configured for each channel.

For all the selected signals, the AXI4 FIFO width for the respective channel is determined by summing up the widths of signals in the particular channel, as shown in Table 4-1.

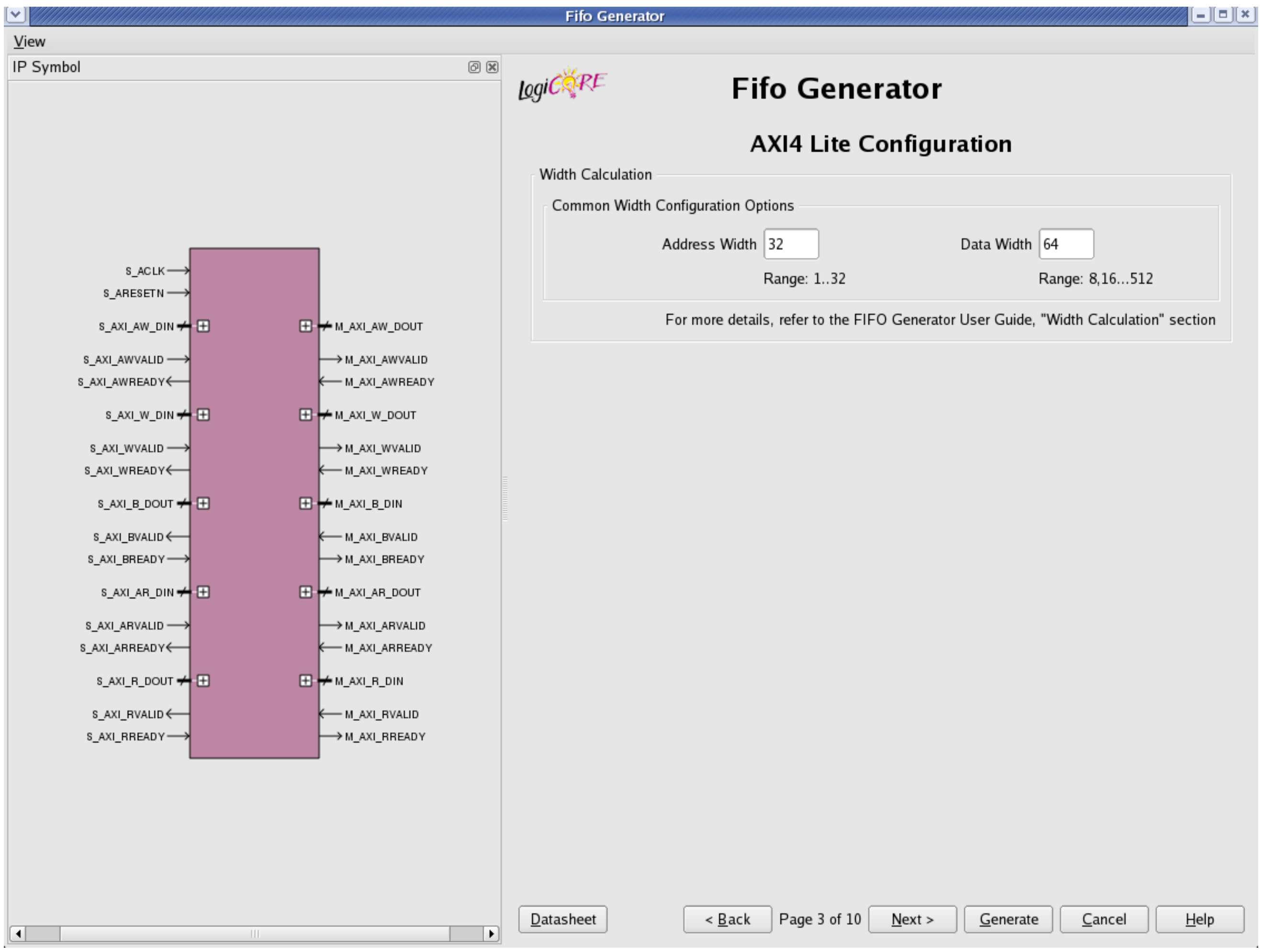
Table 4-1: **AXI4 Signals used in AXI FIFO Width Calculation**

Write Address Channel	Read Address Channel	Write Data Channel	Read Data Channel	Write Resp Channel
AWID[m:0]	ARID[m:0]	WID[m:0]	RID[m:0]	BID[m:0]
AWADDR[m:0]	ARADDR[m:0]	WDATA[m-1:0]	RDATA[m-1:0]	BRESP[1:0]
AWLEN[7:0]	ARLEN[7:0]	WLAST	RLAST	BUSER[m:0]
AWSIZE[2:0]	ARSIZE[2:0]	WSTRB[m/8-1:0]	RRESP[1:0]	

Table 4-1: **AXI4 Signals used in AXI FIFO Width Calculation *(Cont'd)***

Write Address Channel	Read Address Channel	Write Data Channel	Read Data Channel	Write Resp Channel
AWBURST[1:0]	ARBURST[1:0]	WUSER[m:0]	RUSER[m:0]	
AWLOCK[2:0]	ARLOCK[2:0]			
AWCACHE[4:0]	ARCACHE[4:0]			
AWPROT[3:0]	ARPROT[3:0]			
AWREGION[3:0]	ARREGION[3:0]			
AWQOS[3:0]	ARQOS[3:0]			
AWUSER[m:0]	ARUSER[m:0]			

AXI4-Lite Width Calculation



Fifo Generator

AXI4 Lite Configuration

Width Calculation

Common Width Configuration Options

Address Width 32 Range: 1..32

Data Width 64 Range: 8,16...512

For more details, refer to the FIFO Generator User Guide, "Width Calculation" section

Figure 4-4: **AXI4-Lite Width Calculation Screen**

The AXI4-Lite FIFO allows users to configure the widths for ADDR and DATA signals. When both write and read channels are selected, the same ADDR and DATA widths are applied to both the write channels and read channels.

AXI4-Lite FIFO width for the respective channel is determined by summing up the widths of all the signals in the particular channel, as shown in Table 4-2.

Table 4-2: **AXI4-Lite Width Calculation**

Write Address Channel	Read Address Channel	Write Data Channel	Read Data Channel	Write Resp Channel
AWADDR[m:0]	ARADDR[m:0]	WDATA[m-1:0]	RDATA[m:0]	BRESP[1:0]
AWPROT[3:0]	ARPROT[3:0]	WSTRB[m/8-1:0]	RRESP[1:0]	

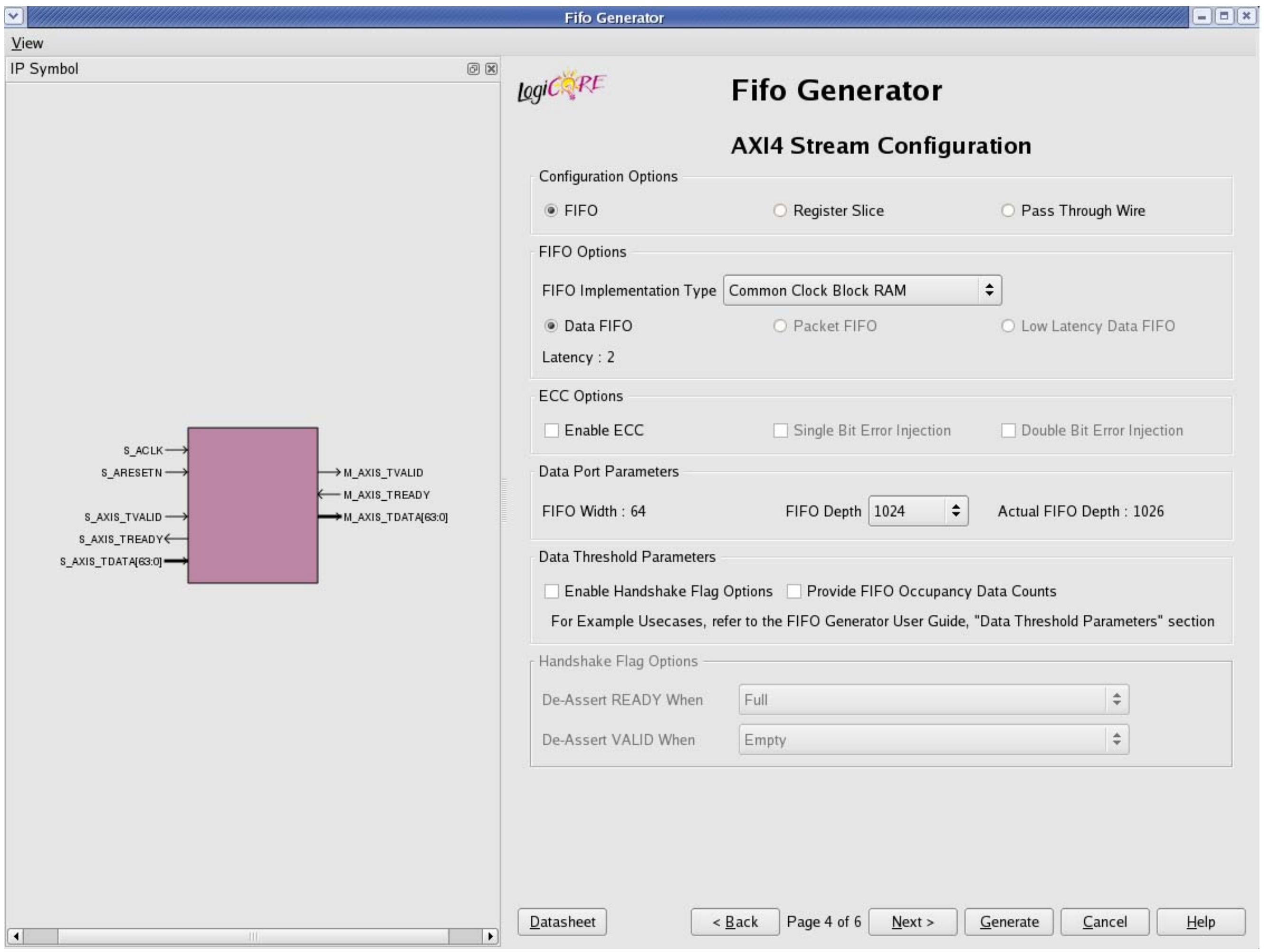
Default Settings

Table 4-3 shows the default settings for each AXI4 interface type.

Table 4-3: **AXI4 FIFO Default Settings**

Interface Type	Channels	Memory Type	FIFO Depth
AXI4 Stream	NA	Block Memory	1024
AXI4	Write Address, Read Address, Write Response	Distributed Memory	16
AXI4	Write Data, Read Data	Block Memory	1024
AXI4-Lite	Write Address, Read Address, Write Response	Distributed Memory	16
AXI4-Lite	Write Data, Read Data	Distributed Memory	16

FIFO Configurations



Fifo Generator

AXI4 Stream Configuration

Configuration Options

[✓ FIFO] Register Slice Pass Through Wire

FIFO Options

FIFO Implementation Type: Common Clock Block RAM

[✓ Data FIFO] Packet FIFO Low Latency Data FIFO

Latency : 2

ECC Options

Enable ECC Single Bit Error Injection Double Bit Error Injection

Data Port Parameters

FIFO Width : 64 FIFO Depth 1024 Actual FIFO Depth : 1026

Data Threshold Parameters

Enable Handshake Flag Options Provide FIFO Occupancy Data Counts

For Example Usecases, refer to the FIFO Generator User Guide, "Data Threshold Parameters" section

Handshake Flag Options

De-Assert READY When: Full

De-Assert VALID When: Empty

Page 4 of 6

Figure 4-5: **AXI4 FIFO Configurations Screen**

The functionality of AXI4 FIFO is the same as the Native FIFO functionality in the first-word fall-through mode. The feature set supported includes ECC (block RAM), Programable Ready Generation (full, almost full, programmable full), and Programmable Valid Generation (empty, almost empty, programmable empty). The data count option tells you the number of words in the FIFO, and there is also are optional Interrupt flags (Overflow and Underflow) for the block RAM and distributed RAM implementations.

For more details on first-word fall-through mode, see First-Word Fall-Through FIFO Read Operation in Chapter 5.

Memory Types

The FIFO Generator implements FIFOs built from block RAM or distributed RAM. The core combines memory primitives in an optimal configuration based on the calculated width and selected depth of the FIFO.

Error Injection and Correction (ECC)

The block RAM and FIFO macros are equipped with built-in Error Injection and Correction Checking in the Virtex-6 FPGA architecture. This feature is available for both common and independent clock block RAM FIFOs.

For more details on Error Injection and Correction, see Built-in Error Correction Checking in Chapter 5.

FIFO Width

AXI4 FIFOs support symmetric Write and Read widths. The width of the AXI4 FIFO is determined based on the selected Interface Type (AXI4-Stream, AXI4 or AXI4-Lite), and the selected signals and configured signal widths within the given interface. The calculation of the FIFO Write Width is defined in Width Calculation, page 83.

FIFO Depth

AXI4 FIFOs allow ranging from 16 to 4194304. Only depths with powers of 2 are allowed.

Handshake Flags

User can control Valid and Ready generation by enabling "Enable Handshake Flag Options." Table 4-4 summarizes the behavior of the Ready/Valid Signal based on the user's Programmable Flag settings.

Table 4-4: **Programmable Ready Generation**

De-Assert Ready On	Description
Full	Ready will be de-asserted when FIFO is Full.
Almost Full	Ready will be de-asserted when FIFO is Almost Full. This indicates that only one more write can be performed before the FIFO is full.
Single Programmable Full Threshold Constant	Ready will be de-asserted when the number of words in the FIFO is greater than or equal to the threshold value programmed via GUI.
Single Programmable Full Threshold Input Port	Ready will be de-asserted when the number of words in the FIFO is greater than or equal to the value set through the input pins: `[AXIS|AXI_AW|AXI_W|AXI_B|AXI_AR|AXI_R]_PROG_FULL_THRESH`

Refer to the CORE Generator GUI for a valid range of threshold values. To avoid unexpected behavior, do not give out of range threshold values.

For more details on Programmable Full Thresholds, see Programmable Full: Single Threshold in Chapter 5.

Table 4-5: **Programmable Valid Generation**

De-Assert Valid When	Description
Empty	Valid will be de-asserted when FIFO is Empty.
Almost Empty	Valid will be de-asserted when FIFO is Almost Empty. This indicates that the FIFO is almost empty and one word remains in the FIFO.

Table 4-5: **Programmable Valid Generation *(Cont'd)***

De-Assert Valid When	Description
Single Programmable Empty Threshold Constant	Valid will be de-asserted when FIFO has reached a user-defined empty threshold. The empty threshold should be programmed via GUI .
Single Programmable Empty Threshold Input Port	Valid will be de-asserted when FIFO has reached a user-defined empty threshold. The empty threshold can be set through input pins.

Refer to the CORE Generator GUI for a valid range of threshold values. To avoid unexpected behavior, do not give out of range threshold values.

Occupancy Data Counts

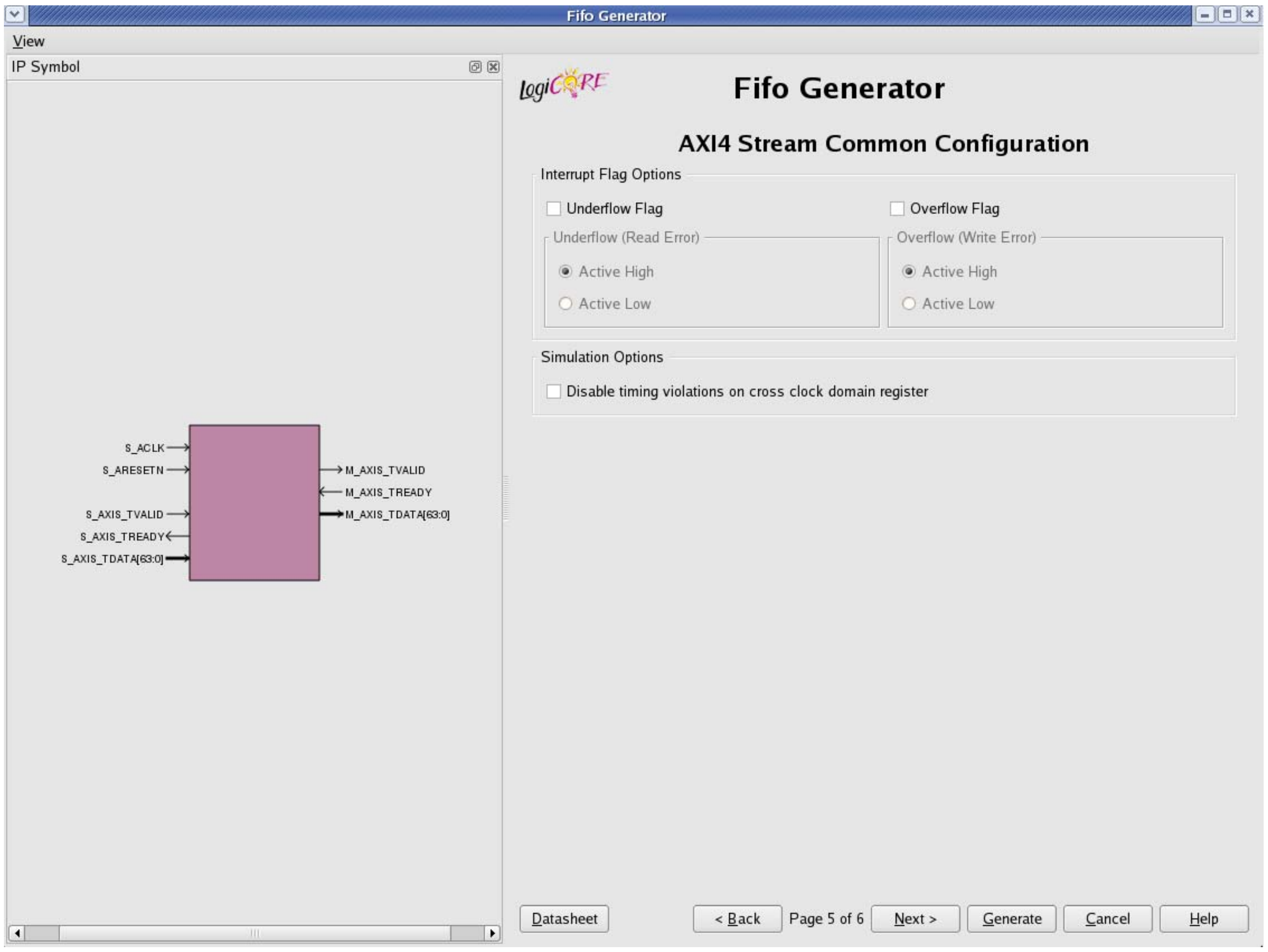
DATA_COUNT tracks the number of words in the FIFO. The width of the data count bus will be always be set to $\log_2$(FIFO depth)+1. In common clock mode, the AXI4 FIFO provides a single "Data Count" output. In independent clock mode, it provides Read Data Count and Write Data Count outputs.

For more details on Occupancy Data Counts, see First-Word Fall-Through Data Count in Chapter 5 and More Accurate Data Count (Use Extra Logic) in Chapter 5.

Examples for Data Threshold Parameters

- Programmable Full Threshold can be used to restrict FIFO Occupancy to less than 16
- Programmable Empty Threshold can be used to drain a Partial AXI4 transfer based on empty threshold
- Data Counts can be used to determine number of Transactions in the FIFO

Common Configurations



AXI4 Stream Common Configuration

Interrupt Flag Options

[ ] Underflow Flag

Underflow (Read Error): [✓ Active High] Active Low

[ ] Overflow Flag

Overflow (Write Error): [✓ Active High] Active Low

Simulation Options

[ ] Disable timing violations on cross clock domain register

Figure 4-6: **AXI4 FIFO Common Configurations Screen**

Interrupt Flags

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).

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).

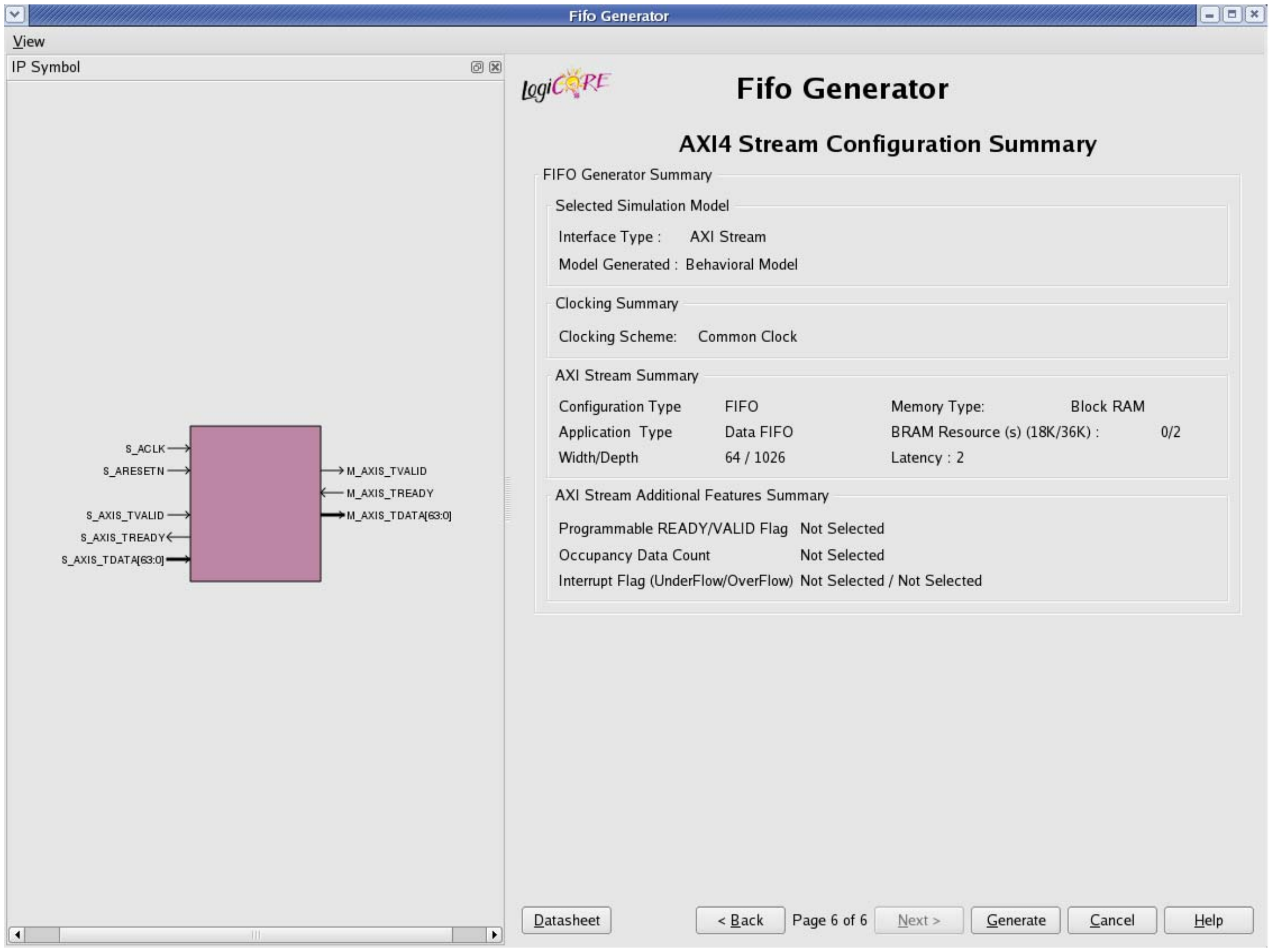
For more details on Overflow and Underflow Flags, see Underflow in Chapter 5 and Overflow in Chapter 5.

Summary

The summary screen displays a summary of the AXI4 FIFO options that have been selected by the user, including the Interface Type, FIFO type, FIFO dimensions, and the selection status of any additional features selected. In the Additional Features section, most features display either Not Selected (if unused), or Selected (if used).

Note: FIFO depth provides the actual FIFO depths for the selected configuration. These depths may differ slightly from the depth selected on screen 4 of the AXI4 FIFO GUI.

AXI4-Stream Summary



Fifo Generator

AXI4 Stream Configuration Summary

FIFO Generator Summary

Selected Simulation Model

Interface Type : AXI Stream

Model Generated : Behavioral Model

Clocking Summary

Clocking Scheme: Common Clock

AXI Stream Summary

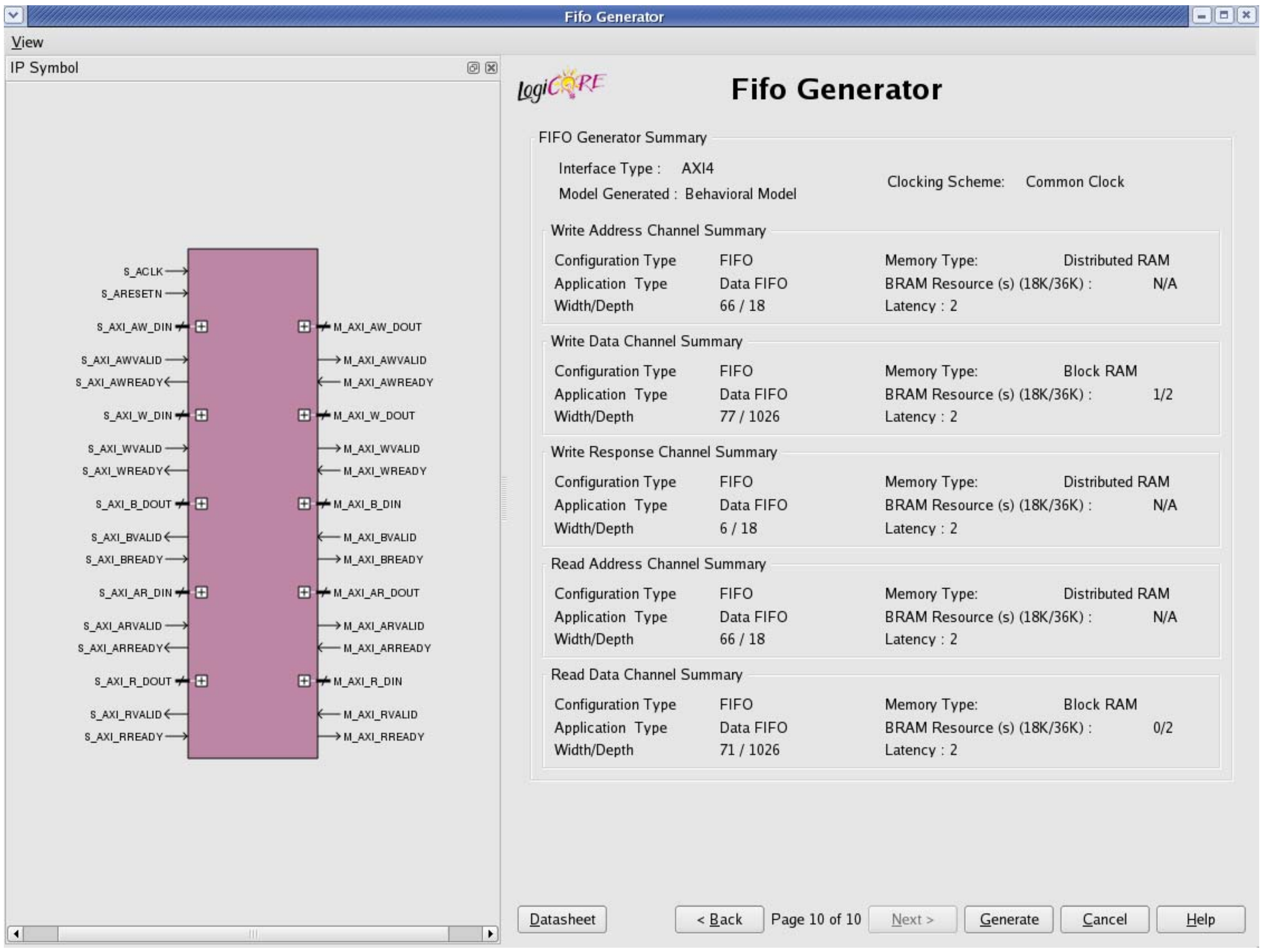
Configuration Type	FIFO	Memory Type:	Block RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	0/2
Width/Depth	64 / 1026	Latency : 2	

AXI Stream Additional Features Summary

Programmable READY/VALID Flag	Not Selected
Occupancy Data Count	Not Selected
Interrupt Flag (UnderFlow/OverFlow)	Not Selected / Not Selected

Figure 4-7: **AXI4-Stream Summary Screen**

AXI4 and AXI4-Lite Summary



Fifo Generator

FIFO Generator Summary

Interface Type : AXI4
Model Generated : Behavioral Model
Clocking Scheme: Common Clock

Write Address Channel Summary

Configuration Type	FIFO	Memory Type:	Distributed RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	N/A
Width/Depth	66 / 18	Latency : 2	

Write Data Channel Summary

Configuration Type	FIFO	Memory Type:	Block RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	1/2
Width/Depth	77 / 1026	Latency : 2	

Write Response Channel Summary

Configuration Type	FIFO	Memory Type:	Distributed RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	N/A
Width/Depth	6 / 18	Latency : 2	

Read Address Channel Summary

Configuration Type	FIFO	Memory Type:	Distributed RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	N/A
Width/Depth	66 / 18	Latency : 2	

Read Data Channel Summary

Configuration Type	FIFO	Memory Type:	Block RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	0/2
Width/Depth	71 / 1026	Latency : 2	

Figure 4-8: **AXI4 / AXI4-Lite Summary Screen**

Chapter 5

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. 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 Guideline

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 and Synchronization

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.

Synchronization Considerations

FIFOs with independent write and read clocks require that interface signals be used only in their respective clock domains. 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.

Important: FIFO Full and Empty flags must be used to guarantee proper behavior.

Figure 5-1 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.

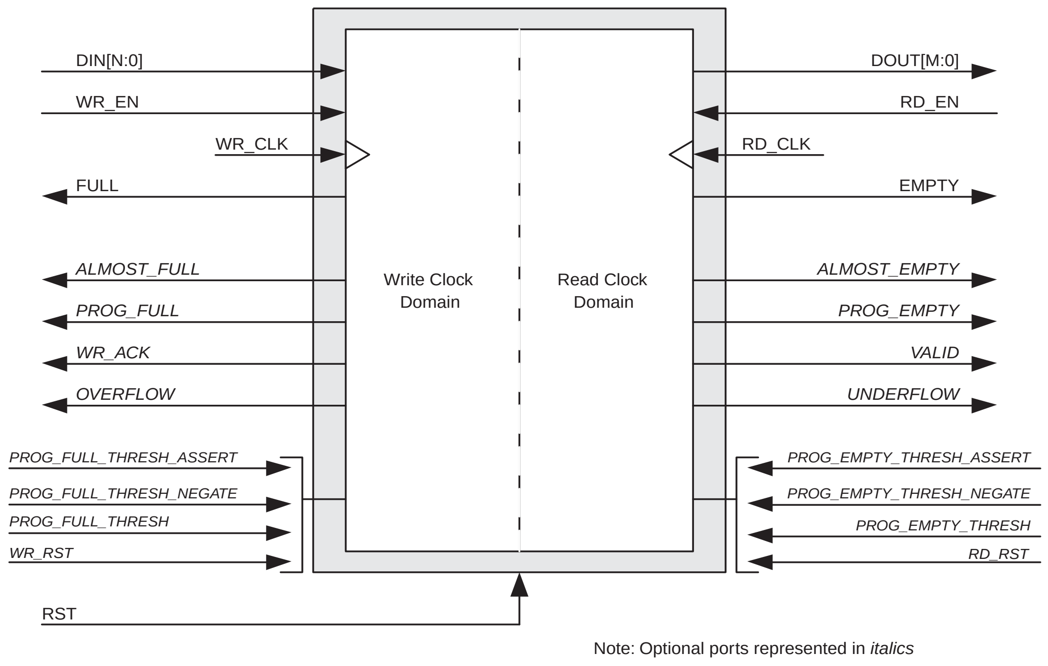


Figure 5-1: **FIFO with Independent Clocks: Write and Read Clock Domains**

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.

Initializing the FIFO Generator

When designing with the built-in FIFO or common clock shift register FIFO, the FIFO must be reset after the FPGA is configured and before operation begins. An asynchronous reset pin (RST) is provided, which 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. For common clock configurations, users have the option of asynchronous or synchronous reset. For independent clock configurations, users have the option of asynchronous reset (RST) or synchronous reset (WR_RST/RD_RST) with respect to respective clock domains.

When asynchronous reset is implemented (Enable Reset Synchronization option is selected), 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. The reset pulse and synchronization delay requirements are dependent on the FIFO implementation types.

When WR_RST/RD_RST is implemented (Enable Reset Synchronization option is not selected), the WR_RST/RD_RST is treated as a synchronous reset to the respective clock domain. The write clock domain remains in reset state as long as `WR_RST` is asserted, and the read clock domain remains in reset state as long as `RD_RST` is asserted. See Reset Behavior, page 128.

FIFO Implementations

Each FIFO configuration has a set of allowable features, as defined in Table 2-3, page 27.

Independent Clocks: Block RAM and Distributed RAM

Figure 5-2 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 synchronization across clock domains, and logic for calculating the status flags.

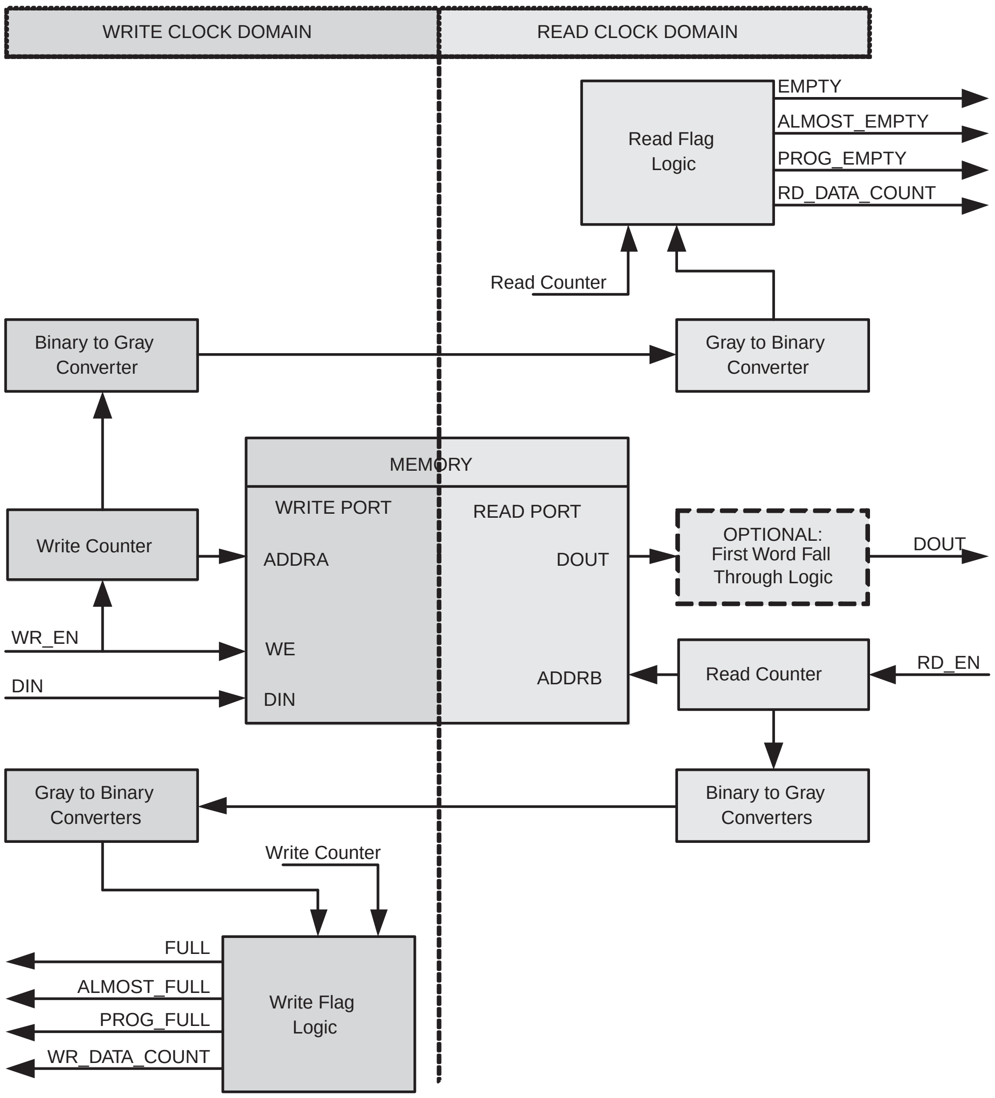


Figure 5-2: **Functional Implementation of a FIFO with Independent Clock Domains**

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. Table 5-1 summarizes the FIFO interface signals, which are only valid in their respective clock domains.

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
WR_RST	SBITERR
INJECTSBITERR	DBITERR
INJECTDBITERR	RD_RST

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 97.

Independent Clocks: Built-in FIFO

Figure 5-3 illustrates the functional implementation of FIFO configured with independent clocks using the Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA 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.

The Virtex-4 FPGA built-in FIFO implementation allows generation of a single primitive. The generated core includes a FIFO flag patch (defined in “Solution 1: Synchronous/Asynchronous Clock Work-Arounds,” in the *Virtex-4 FPGA User Guide*).

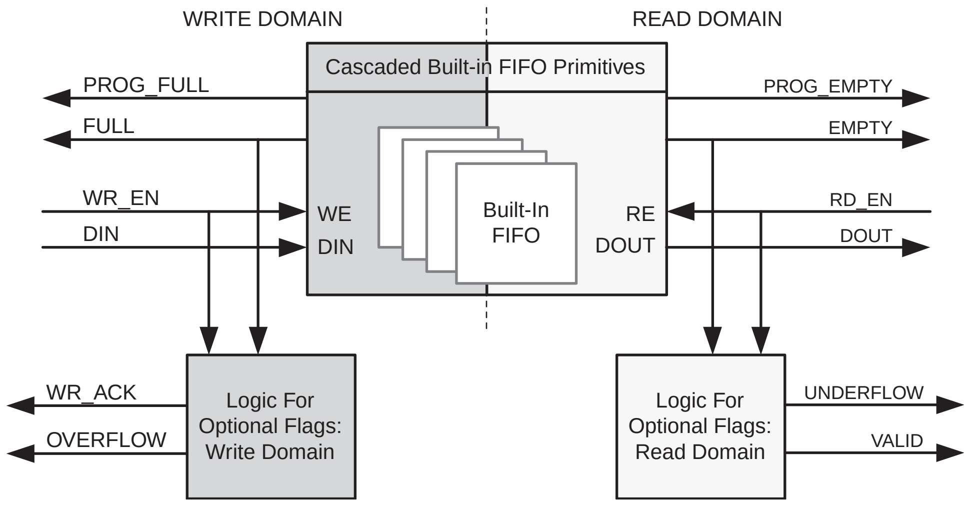


Figure 5-3: **Functional Implementation of Built-in FIFO**

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. Table 5-2 summarizes the FIFO interface signals, which are only valid in their respective clock domains.

Table 5-2: **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
INJECTSBITERR	SBITERR
INJECTDBITERR	DBITERR

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, see Synchronization Considerations, page 97.

For Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA built-in FIFO configurations, the built-in ECC feature in the FIFO macro is provided. For more information, see "Built-in Error Correction Checking," page 126.

Common Clock: Built-in FIFO

The FIFO Generator supports FIFO cores using the 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 101, for more information.

Common Clock FIFO: Block RAM and Distributed RAM

Figure 5-4 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. An optional synchronous (SRST) or asynchronous (RST) reset signal is also available.

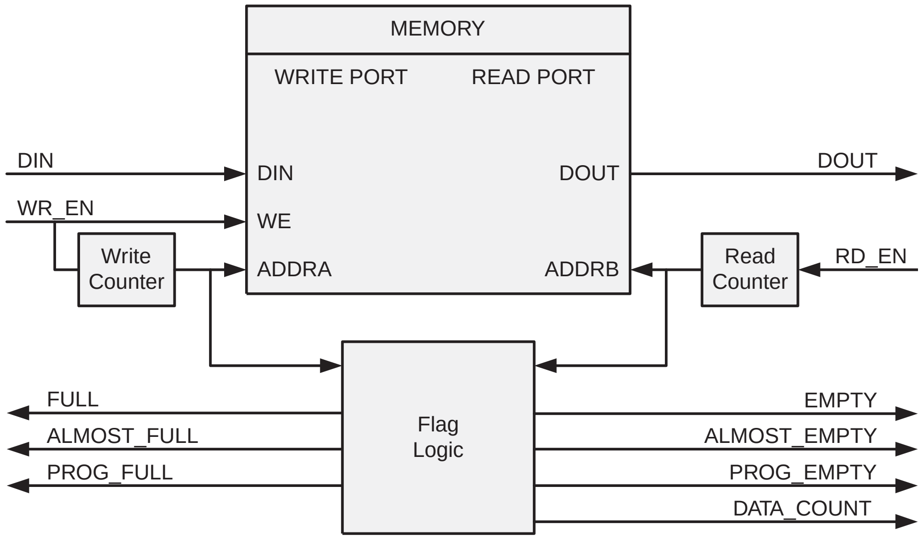


Figure 5-4: **Functional Implementation of a Common Clock FIFO using Block RAM or Distributed RAM**

Common Clock FIFO: Shift Registers

Figure 5-5 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.

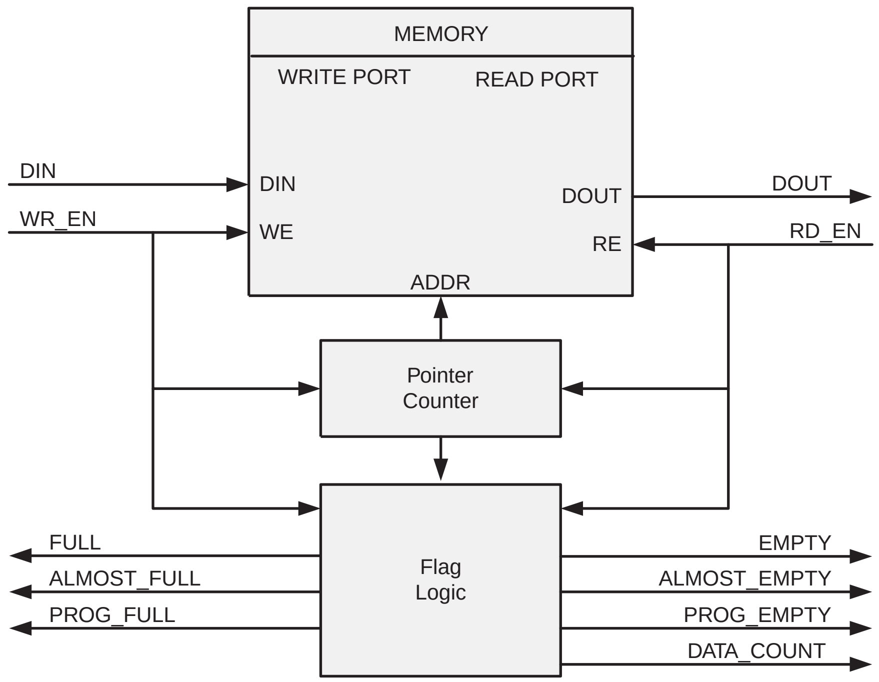


Figure 5-5: **Functional Implementation of a Common Clock FIFO using Shift Registers**

FIFO Usage and Control

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

Note: The Built-in FIFO for Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs do not support the ALMOST_FULL flag.

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.

Important: For the Virtex-4 FPGA built-in FIFO implementation, the Full signal has an extra cycle of latency. Use Write Acknowledge to verify success or Programmable Full for an earlier indication.

Example Operation

Figure 5-6 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. Because 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.

Note: The Virtex-4 FPGA built-in FIFO implementation shows an extra cycle of latency on the FULL flag.

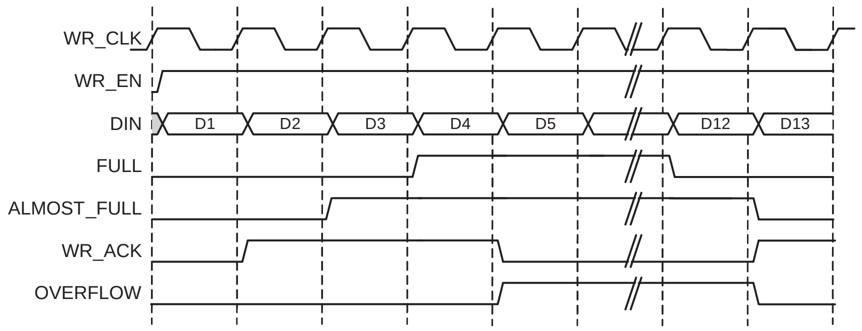


Figure 5-6: **Write Operation for a FIFO with Independent Clocks**

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

Note: The Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs built-in FIFO does not support the ALMOST_EMPTY flag.

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.

Common Clock Note

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.

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.

Table 5-3 details the supported implementations for FWFT.

Table 5-3: **Implementation-Specific Support for First-Word Fall-Through**

FIFO Implementation		FWFT Support
Independent Clocks	Block RAM	✓
	Distributed RAM	✓
	Built-in	✓(1)
Common Clock	Block RAM	✓
	Distributed RAM	✓
	Shift Register	
	Built-in	✓(1)

Notes:

1. Only supported in Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs.

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 5-7 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.

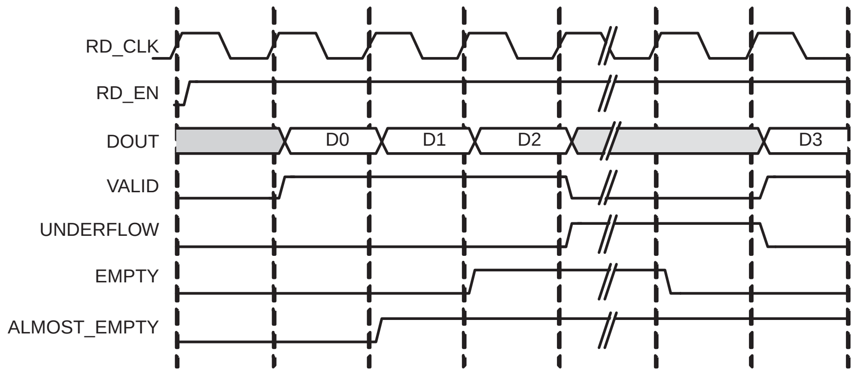


Figure 5-7: **Standard Read Operation for a FIFO with Independent Clocks**

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 5-8 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.

Unlike the standard read mode, the first-word-fall-through empty flag is asserted after the last data is read from the FIFO. When `EMPTY` is asserted, `VALID` is deasserted. In the standard read mode, when `EMPTY` is asserted, `VALID` is asserted for 1 clock cycle. The FWFT feature also increases the effective read depth of the FIFO by two read words.

The FWFT feature adds two clock cycle latency to the deassertion of empty, when the first data is written into a empty FIFO.

Note: For every write operation, an equal number of read operations is required to empty the FIFO – this is true for both the first-word-fall-through and standard FIFO.

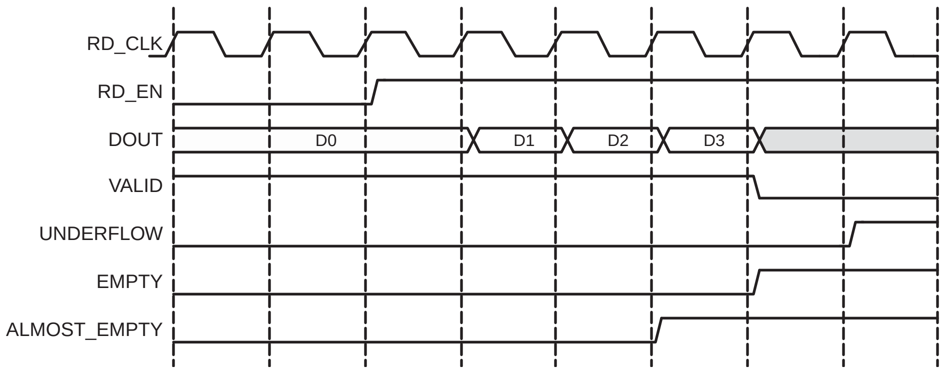


Figure 5-8: **FWFT Read Operation for a FIFO with Independent Clocks**

Common Clock FIFO, Simultaneous Read and Write Operation

Figure 5-9 shows a typical 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.

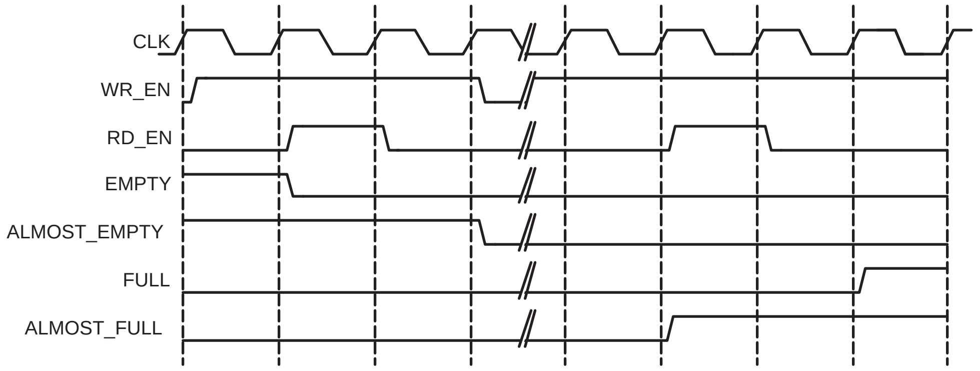


Figure 5-9: **Write and Read Operation for a FIFO with Common Clocks**

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 5-10.

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).

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.

Example Operation

Figure 5-10 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.

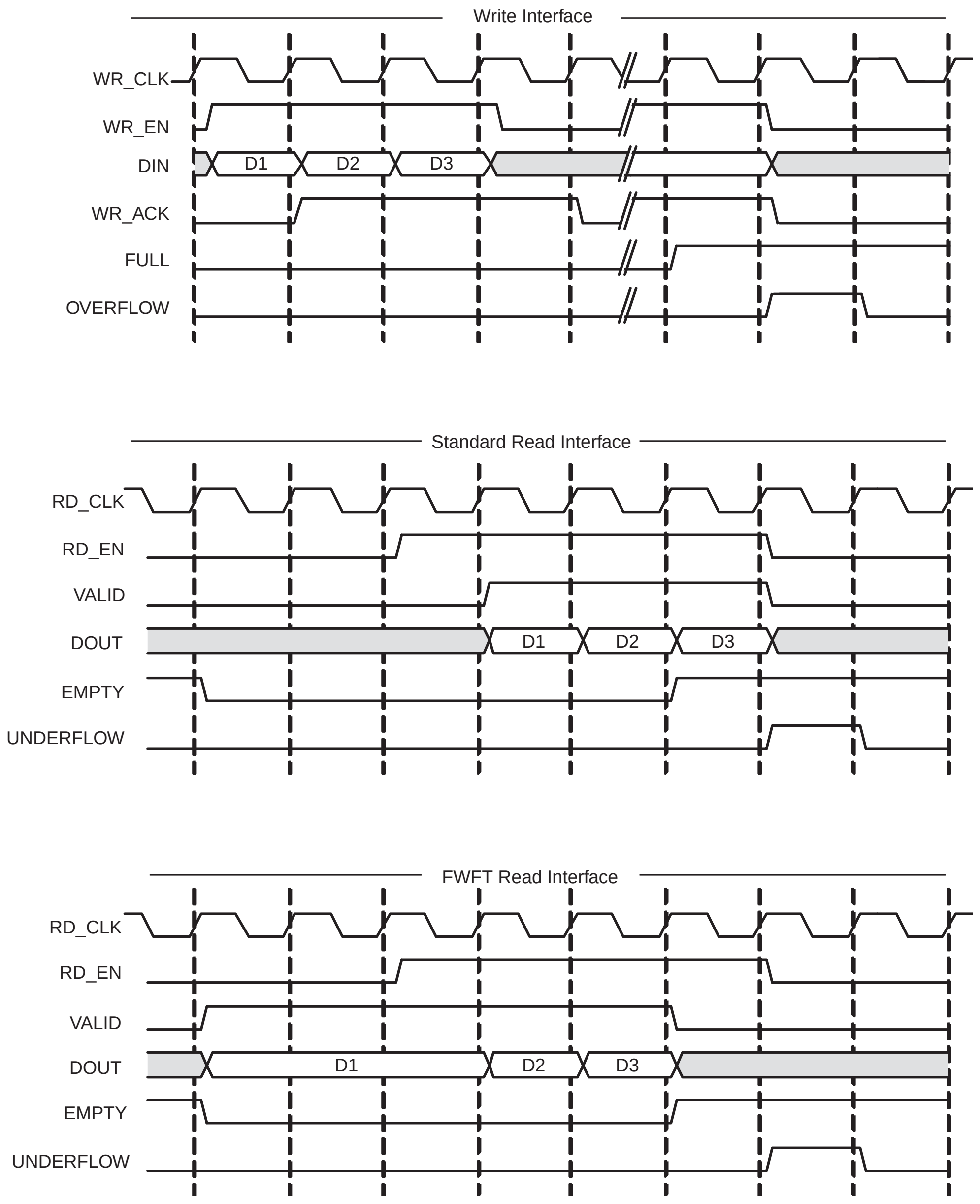


Figure 5-10: **Handshaking Signals for a FIFO with Independent Clocks**

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).

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 5-11 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.

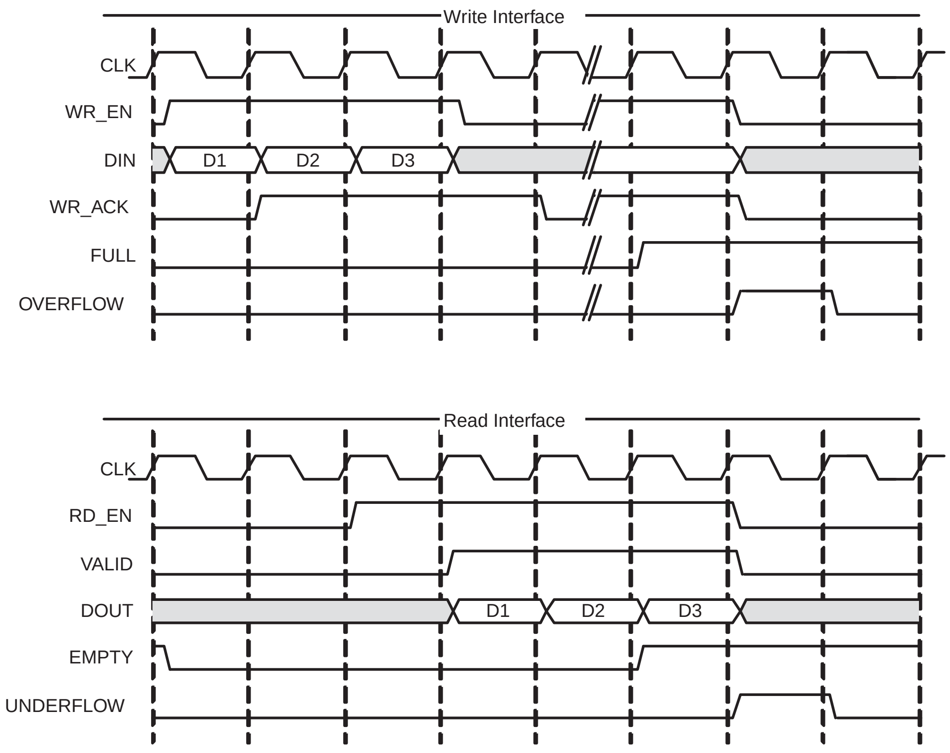


Figure 5-11: **Handshaking Signals for a FIFO with Common Clocks**

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, 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. For information about the latency behavior of the programmable flags, see "Latency," page 137.

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)

Note: The built-in FIFOs only support single-threshold constant programmable full.

These options are available in the CORE Generator GUI and accessed within the programmable flags window (Figure 3-5).

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. The deassertion behavior differs between built-in and non built-in FIFOs (block RAM, distributed RAM, and so forth).

For built-in FIFOs, the number of entries in the FIFO has to be less than the threshold value -1 before `PROG_FULL` is deasserted. For non built-in FIFOs, if the number of words in the FIFO is less than the negate threshold, the flag is deasserted.

Two options are available to implement 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** (non-built-in FIFOs only). 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: See the CORE Generator GUI screen for valid ranges for each threshold.

Figure 5-12 shows the programmable full flag with a single threshold for a non-built-in FIFO. The user writes to the FIFO until there are seven words in the FIFO. Because 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.

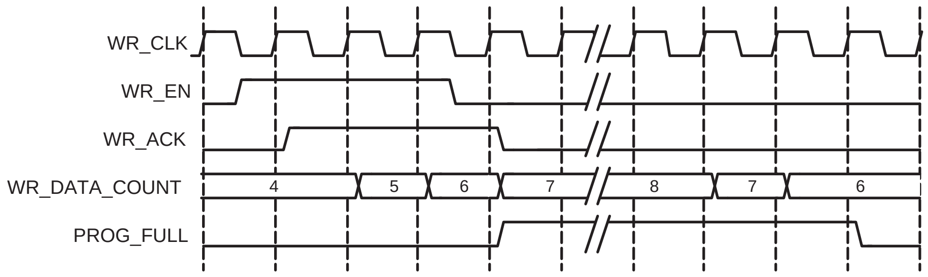


Figure 5-12: **Programmable Full Single Threshold: Threshold Set to 7**

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. Note that this feature is not available for built-in FIFOs.

Two options are available to implement 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 5-13 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. Both write data count (`WR_DATA_COUNT`) and `PROG_FULL` have one clock cycle of delay.

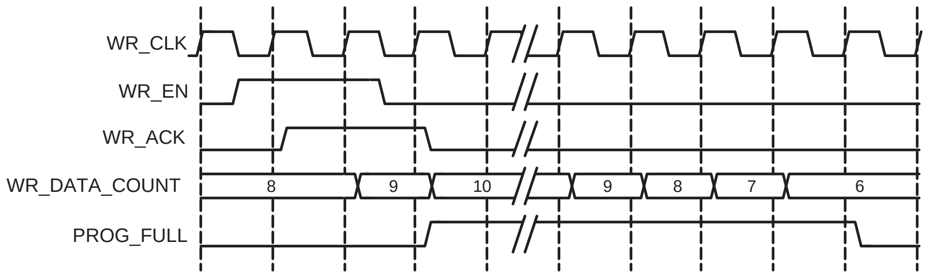


Figure 5-13: **Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7**

Programmable Full Threshold Range Restrictions

The programmable full threshold ranges depend on several features that dictate the way the FIFO is implemented, and include the following features:

- FIFO Implementation Type (Built-in FIFO or non Built-in FIFO, Common or Independent Clock FIFOs, and so forth)
- Symmetric or Non-symmetric Port Aspect Ratio
- Read Mode (Standard or First-Word-Fall-Through)
- Read and Write Clock Frequencies (Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA Built-in FIFOs only)

The FIFO Generator GUI automatically parameterizes the threshold ranges based on these features, allowing you to choose only within the valid ranges. Note that for the Common or Independent Clock Built-in FIFO implementation type, you can only choose a threshold range within 1 primitive deep of the FIFO depth, due to the core implementation. If a wider threshold range is required, use the Common or Independent Clock Block RAM implementation type.

Note: Refer to the CORE Generator GUI for valid ranges for each threshold. To avoid unexpected behavior, it is not recommended to give out-of-range threshold values.

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)

Note: The built-in FIFOs only support single-threshold constant programmable full.

These options are available in the CORE Generator GUI and accessed within the programmable flags window (Figure 3-5).

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 you 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. The deassertion behavior differs between built-in and non built-in FIFOs (block RAM, distributed RAM, and so forth).

For built-in FIFOs, the number of entries in the FIFO must be greater than the threshold value + 1 before PROG_EMPTY is deasserted. For non built-in FIFOs, if the number of entries in the FIFO is greater than threshold value, PROG_EMPTY is deasserted.

Two options are available to implement 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 flexibility to change the programmable empty threshold in-circuit without re-generating the core.

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

Figure 5-14 shows the programmable empty flag with a single threshold for a non-built-in FIFO. The user writes to the FIFO until there are five words in the FIFO. Because 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. Both read data count (RD_DATA_COUNT) and PROG_EMPTY have one clock cycle of delay.

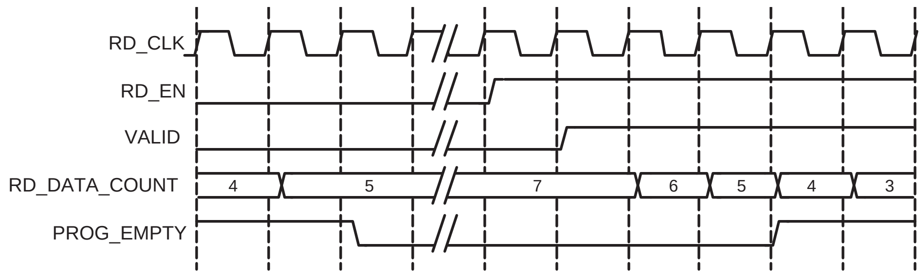


Figure 5-14: **Programmable Empty with Single Threshold: Threshold Set to 4**

Programmable Empty: Assert and Negate Thresholds

This option lets the user 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. This feature is not available for built-in FIFOs.

Two options are available to implement these 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 5-15 shows the programmable empty flag with assert and negate thresholds. The user writes to the FIFO until there are eleven words in the FIFO; because the programmable empty deassert value is set to ten, `PROG_EMPTY` is deasserted when more than ten words are in the FIFO. Once the FIFO contains less than or equal to the programmable empty negate value (set to seven), `PROG_EMPTY` is asserted. Both read data count (`RD_DATA_COUNT`) and PROG_EMPTY have one clock cycle of delay.

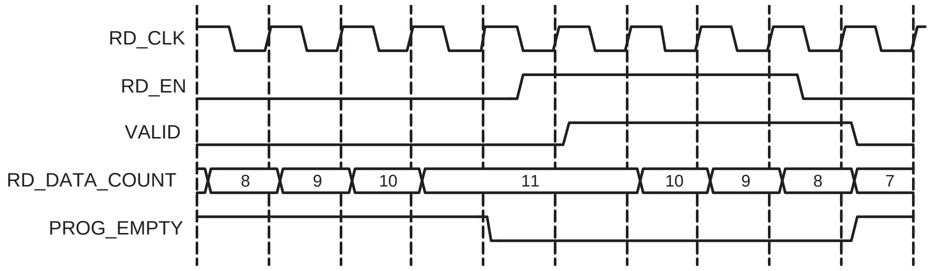


Figure 5-15: **Programmable Empty with Assert and Negate Thresholds: Assert Set to 7 and Negate Set to 10**

Programmable Empty Threshold Range Restrictions

The programmable empty threshold ranges depend on several features that dictate the way the FIFO is implemented, including the following:

- FIFO Implementation Type (Built-in FIFO or non Built-in FIFO, Common or Independent Clock FIFOs, and so forth)
- Symmetric or Non-symmetric Port Aspect Ratio
- Read Mode (Standard or First-Word-Fall-Through)
- Read and Write Clock Frequencies (Kintex-7, Virtex-7, Virtex-6, Virtex-5, and Virtex-4 FPGA Built-in FIFOs only)

The FIFO Generator GUI automatically parameterizes the threshold ranges based on these features, allowing you to choose only within the valid ranges. Note that for Common or Independent Clock Built-in FIFO implementation type, you can only choose a threshold range within 1 primitive deep of the FIFO depth due to the core implementation. If a wider threshold range is needed, use the Common or Independent Clock Block RAM implementation type.

Note: Refer to the CORE Generator GUI for valid ranges for each threshold. To avoid unexpected behavior, it is not recommended to give out-of-range threshold values.

Data Counts

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 log2 (FIFO depth). If the width specified is smaller than the maximum allowable width, the bus is truncated by removing the lower bits. These signals are optional outputs of the FIFO Generator, and are enabled through the CORE Generator GUI. Table 5-4 identifies data count support for each FIFO implementation. For information about the latency behavior of data count flags, see "Latency," page 137.

Table 5-4: **Implementation-specific Support for Data Counts**

FIFO Implementation		Data Count Support
Independent Clocks	Block RAM	✓
	Distributed RAM	✓
	Built-in	
Common Clock	Block RAM	✓
	Distributed RAM	✓
	Shift Register	✓
	Built-in	

Data Count (Common Clock FIFO Only)

Data Count output (DATA_COUNT) accurately reports the number of words available in a Common Clock FIFO. You can specify the width of the data count bus with a maximum width of log2(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 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, providing the status of the contents of the FIFO for read and write operations.

Note: If a read or write operation occurs on a rising edge of CLK, the data count port is updated at the same rising edge of CLK.

Read Data Count (Independent Clock FIFO Only)

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 design never underflows the FIFO. The user can specify the width of the read data count bus with a maximum width of log2 (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 (Independent Clock FIFO Only)

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 log2 (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, first-word fall-through (FWFT) implementations increase the depth of the FIFO by 2 read words. Using this configuration, the FIFO Generator enables the user to generate data count in two ways:

- Approximate Data Count
- More Accurate Data Count (Use Extra Logic)

Approximate Data Count

Approximate Data Count behavior is the default option in the CORE Generator GUI for independent clock block RAM and distributed RAM FIFOs. This feature is not available for common clock FIFOs. The width of the `WR_DATA_COUNT` and `RD_DATA_COUNT` is identical to the non first-word-fall-through configurations (log2 (write depth) and log2 (read depth), respectively) but the data counts reported is an approximation because the actual full depth of the FIFO is not supported.

Using this option, you can use specific bits in `WR_DATA_COUNT` and `RD_DATA_COUNT` to approximately indicate the status of the FIFO, for example, half full, quarter full, and so forth.

For example, for a FIFO with a depth of 16, symmetric read and write port widths, and the first-word-fall-through option selected, the *actual* FIFO depth increases from 15 to 17. When using approximate data count, the width of `WR_DATA_COUNT` and `RD_DATA_COUNT` is 4 bits, with a maximum of 15. For this option, you can use the assertion of the MSB bit of the data count to indicate that the FIFO is approximately half full.

More Accurate Data Count (Use Extra Logic)

This feature is enabled when Use Extra Logic for More Accurate Data Counts is selected in the CORE Generator GUI. In this configuration, the width of `WR_DATA_COUNT`, `RD_DATA_COUNT`, and `DATA_COUNT` is log2(write depth)+1, log2(read depth)+1, and log2(depth)+1, respectively to accommodate the increase in depth in the first-word-fall-through case and to ensure accurate data count is provided.

Note that when using this option, you *cannot* use any one bit of WR_DATA_COUNT, RD_DATA_COUNT, and DATA_COUNT to indicate the status of the FIFO, for example, approximately half full, quarter full, and so forth.

For example, for an independent FIFO with a depth of 16, symmetric read and write port widths, and the first-word-fall-through option selected, the *actual* FIFO depth increases from 15 to 17. When using accurate data count, the width of the WR_DATA_COUNT and RD_DATA_COUNT is 5 bits, with a maximum of 31. For this option, you must use the assertion of both the MSB and MSB-1 bit of the data count to indicate that the FIFO is at least half full.

Data Count Behavior

For FWFT implementations using More Accurate Data Counts (Use Extra Logic), DATA_COUNT is guaranteed to be accurate when words are present in the FIFO, with the exception of when its near empty or almost empty or when initial writes occur on an empty FIFO. In these scenarios, DATA_COUNT may be incorrect on up to two words.

Table 5-5 defines the value of DATA_COUNT when FIFO is empty.

From the point-of-view of the write interface, DATA_COUNT is always accurate, reporting the first word immediately once its written to the FIFO. However, from the point-of-view of the read interface, the DATA_COUNT output may over-report by up to two words until ALMOST_EMPTY and EMPTY have both deasserted. This is due to the latency of EMPTY deassertion in the first-word-fall-through FIFO (see Table 5-17). This latency allows DATA_COUNT to reflect written words which may not yet be available for reading.

From the point-of-view of the read interface, the data count starts to transition from over-reporting to accurate-reporting at the deassertion to empty. This transition completes after ALMOST_EMPTY deasserts. Before ALMOST_EMPTY deasserts, the DATA_COUNT signal may exhibit the following atypical behaviors:

- From the read-interface perspective, DATA_COUNT may over-report up to two words.

Write Data Count Behavior

Even for FWFT implementations using More Accurate Data Counts (Use Extra Logic), WR_DATA_COUNT will still pessimistically report the number of words written into the FIFO. However, the addition of this feature will cause WR_DATA_COUNT to further over-report up to two read words (and 1 to 16 write words, depending on read and write port aspect ratio) when the FIFO is at or near empty or almost empty.

Table 5-5 defines the value of WR_DATA_COUNT when the FIFO is empty.

The WR_DATA_COUNT starts to transition out of over-reporting two extra read words at the deassertion of EMPTY. This transition completes several clock cycles after ALMOST_EMPTY deasserts. Note that prior to the transition period, WR_DATA_COUNT will always over-report by at least two read words. During the transition period, the WR_DATA_COUNT signal may exhibit the following strange behaviors:

- WR_DATA_COUNT may decrement although no read operation has occurred.
- WR_DATA_COUNT may not increment as expected due to a write operation.

Note: During reset, WR_DATA_COUNT and DATA_COUNT value is set to 0.

Table 5-5: **Empty FIFO WR_DATA_COUNT/DATA_COUNT Value**

Write Depth to Read Depth Ratio	Approximate WR_DATA_COUNT	More Accurate WR_DATA_COUNT	More Accurate DATA_COUNT
1:1	0	2	2
1:2	0	1	N/A
1:4	0	0	N/A
1:8	0	0	N/A
2:1	0	4	N/A
4:1	0	8	N/A
8:1	0	16	N/A

The RD_DATA_COUNT value at empty (when no write is performed) is 0 with or without Use Extra Logic for all write depth to read depth ratios.

Example Operation

Figure 5-16 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 5-16, 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.

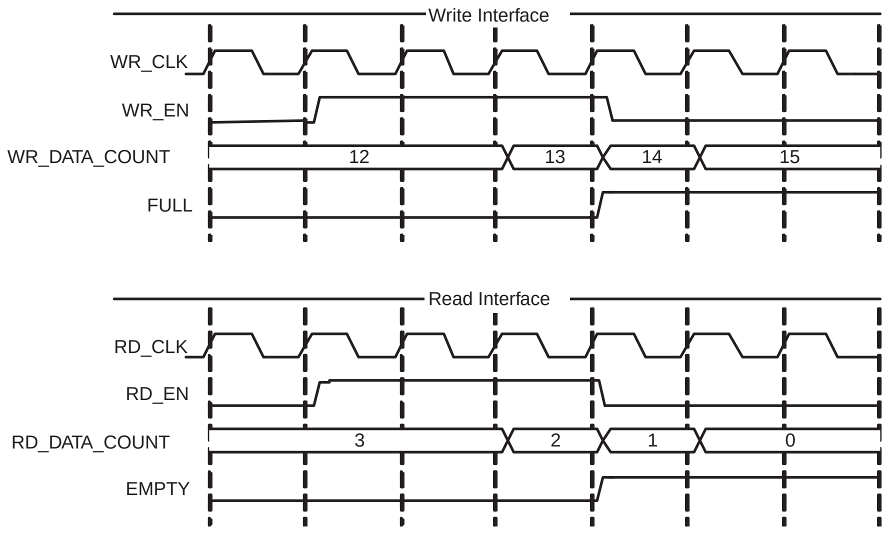


Figure 5-16: **Write and Read Data Counts for FIFO with Independent Clocks**

Non-symmetric Aspect Ratios

Table 5-6 identifies support for non-symmetric aspect ratios.

Table 5-6: **Implementation-specific Support for Non-symmetric Aspect Ratios**

FIFO Implementation		Non-symmetric Aspect Ratios Support
Independent Clocks	Block RAM	✓
	Distributed RAM	
	Built-in	
Common Clock	Block RAM	
	Distributed RAM	
	Shift Register	
	Built-in	

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 5-17 and Figure 5-18).

Figure 5-17 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 01, followed by 00, 11, and finally 10. The memory is filling up from the left to the right (MSB to LSB). When a read operation is performed, the received data is 01_00_11_10.

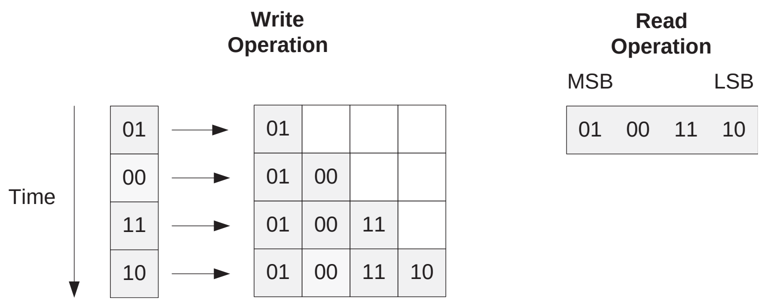


Figure 5-17: **1:4 Aspect Ratio: Data Ordering**

Figure 5-18 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.

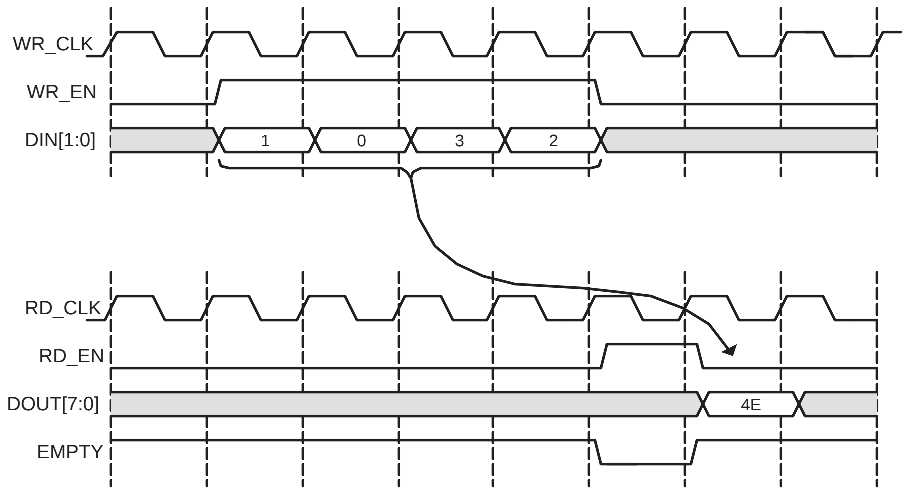


Figure 5-18: **1:4 Aspect Ratio: Status Flag Behavior**

Figure 5-19 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.

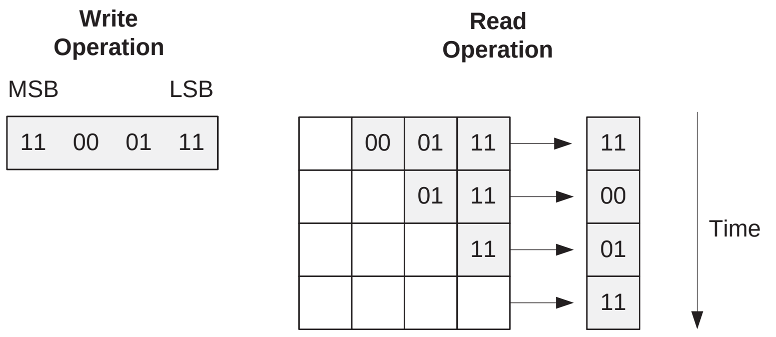


Figure 5-19: **4:1 Aspect Ratio: Data Ordering**

Figure 5-20 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`. Because no other writes occur, the FIFO reasserts empty after four reads.

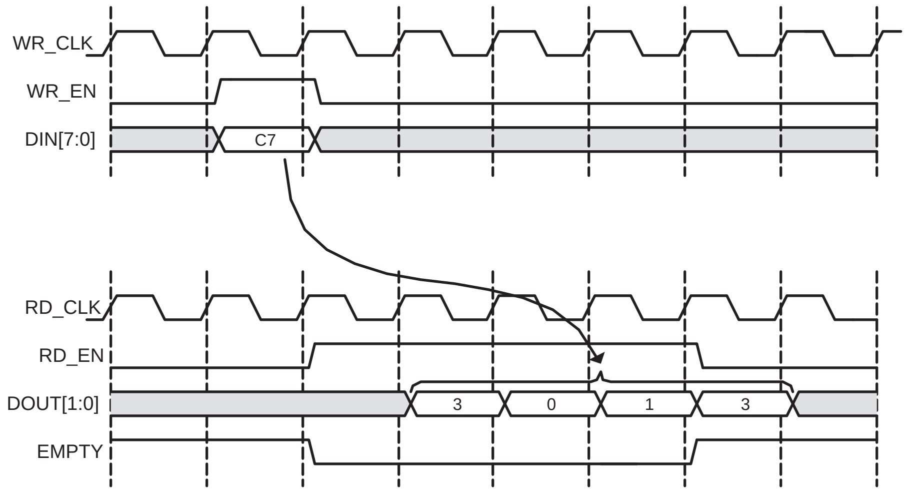


Figure 5-20: **4:1 Aspect Ratio: Status Flag Behavior**

Non-symmetric Aspect Ratio and First-Word Fall-Through

A FWFT FIFO has 2 extra read words available on the read port when compared to a standard FIFO. For write-to-read aspect ratios that are larger or equal to 1 (1:1, 2:1, 4:1, and 8:1), the FWFT implementation also increases the number of words that can be written into the FIFO by depth_ratio*2 (depth_ratio = write depth / read depth). For write-to-read aspect ratios smaller than 1 (1:2, 1:4 and 1:8), the addition of 2 extra read words only amounts to a fraction of 1 write word. The creation of these partial words causes the behavior of the `PROG_EMPTY` and `WR_DATA_COUNT` signals of the FIFO to differ in behavior than as previously described.

Programmable Empty

In general, PROG_EMPTY is guaranteed to assert when the number of readable words in the FIFO is less than or equal to the programmable empty assert threshold. However, when the write-to-read aspect ratios are smaller than 1 (depending on the read and write clock frequency) it is possible for PROG_EMPTY to violate this rule, but only while EMPTY is asserted. To avoid this condition, the user should set the programmable empty assert threshold to 3*depth_ratio*frequency_ratio (depth_ratio = write depth/read depth and frequency_ratio = write clock frequency / read clock frequency). If the programmable empty assert threshold is set lower than this value, the user should assume that PROG_EMPTY may or can be asserted when EMPTY is asserted.

Write Data Count

In general, WR_DATA_COUNT pessimistically reports the number of words written into the FIFO and is guaranteed to never under-report the number of words in the FIFO, to ensure that the user never overflows the FIFO. However, when the write-to-read aspect ratios are smaller than 1, if the read and write operations result in partial write words existing in the FIFO, it is possible to under-report the number of words in the FIFO. This behavior is most crucial when the FIFO is 1 or 2 words away from full, because in this state the WR_DATA_COUNT is under-reporting and cannot be used to gauge if the FIFO is full. In this configuration, you should use the FULL flag to gate any write operation to the FIFO.

Embedded Registers in Block RAM and FIFO Macros (Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGAs)

The block RAM macros available in Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA, as well as built-in FIFO macros available in Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA, have built-in embedded registers that can be used to pipeline data and improve macro timing. Depending on the configuration, this feature can be leveraged to add one additional latency to the FIFO core (DOUT bus and VALID outputs) or implement the output registers for FWFT FIFOs. For built-in FIFOs configuration, this feature is only available for common clock FIFOs.

Standard FIFOs

When using the embedded registers to add an output pipeline register to the standard FIFOs, only the DOUT and VALID output ports are delayed by 1 clock cycle during a read operation. These additional pipeline registers are always enabled, as illustrated in Figure 5-21.

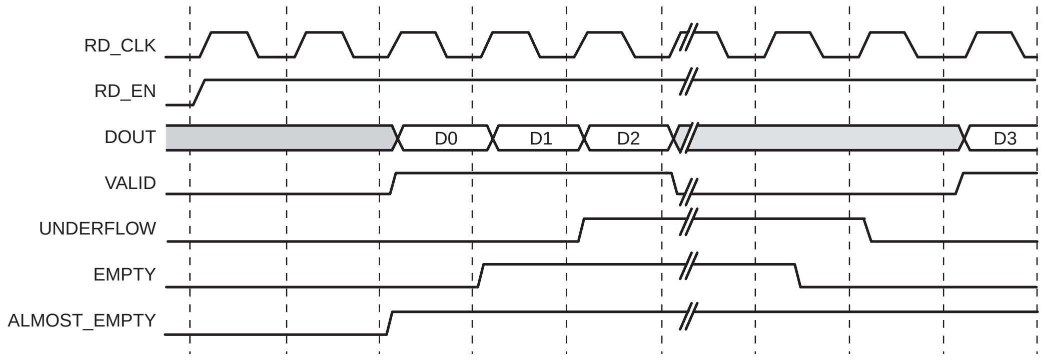


Figure 5-21: **Standard Read Operation for a Block RAM or built-in FIFO with Use Embedded Registers Enabled**

Block RAM Based FWFT FIFOs

When using the embedded output registers to implement the FWFT FIFOs, the behavior of the core is identical to the implementation without the embedded registers.

Built-in Based FWFT FIFOs (Common Clock Only)

When using the embedded output registers with a common clock built-in based FIFO with FWFT, the embedded registers add an output pipeline register to the FWFT FIFO. The DOUT and VALID output ports are delayed by 1 clock cycle during a read operation. These pipeline registers are always enabled, and the DOUT reset value feature is not supported in Virtex-4 and Virtex-5 FPGAs, as illustrated in Figure 5-22. For this configuration, the embedded output register feature is only available for FIFOs that use only 1 FIFO macro in depth.

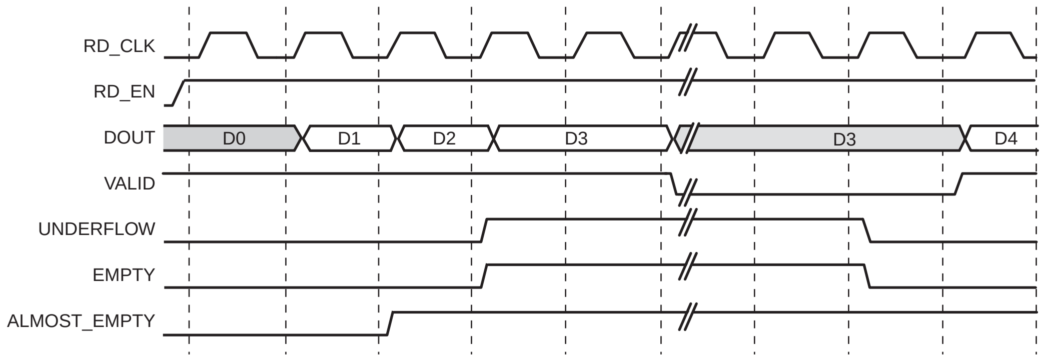


Figure 5-22: **FWFT Read Operation for a Synchronous Built-in FIFO with User Embedded Registers Enabled**

Note: Virtex-5 FPGA built-in FIFOs with independent clocks and FWFT always use the embedded output registers in the macro to implement the FWFT registers.

When using the embedded output registers with a common clock built-in FIFO in Kintex-7, Virtex-7, and Virtex-6 FPGAs, the DOUT reset value feature is supported, as illustrated in Figure 5-23.

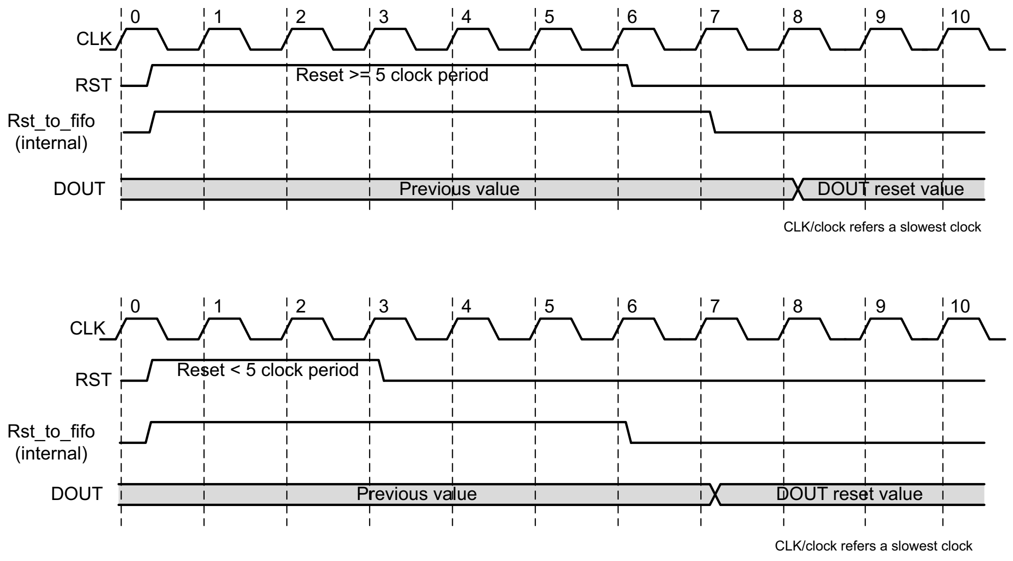


Figure 5-23: **DOUT Reset Value for Kintex-7, Virtex-7, and Virtex-6 Common Clock Built-in FIFO Embedded Register**

Built-in Error Correction Checking

Built-in ECC is supported for FIFOs configured with independent or common clock block RAM and built-in FIFOs targeting Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs. In addition, error injection is supported for FIFOs configured with independent or common clock block RAM and built-in FIFOs targeting Kintex-7, Virtex-7, and Virtex-6 FPGAs. When ECC is enabled, the block RAM and built-in FIFO primitive used to create the FIFO is configured in the full ECC mode (both encoder and decoder enabled), providing two additional outputs to the FIFO Generator core: SBITERR and DBITERR. These outputs indicate three possible read results: no error, single error corrected, and double error detected. In the full ECC mode, the read operation does not correct the single error in the memory array, it only presents corrected data on DOUT.

Figure 5-24 shows how the SBITERR and DBITERR outputs are generated in the FIFO Generator core. The output signals are created by combining all the SBITERR and DBITERR signals from the FIFO or block RAM primitives using an OR gate. Because the FIFO primitives may be cascaded in depth, when SBITERR or DBITERR is asserted, the error may have occurred in any of the built-in FIFO macros chained in depth or block RAM macros. For this reason, these flags are not correlated to the data currently being read from the FIFO Generator core or to a read operation. For this reason, when the DBITERR is flagged, the user should assume that the data in the entire FIFO has been corrupted and the user logic needs to take the appropriate action. As an example, when DBITERR is flagged, an appropriate action for the user logic is to halt all FIFO operation, reset the FIFO, and restart the data transfer.

The SBITERR and DBITERR outputs are not registered and are generated combinatorially. If the configured FIFO uses two independent read and write clocks, the SBITERR and DBITERR outputs may be generated from either the write or read clock domain. The signals generated in the write clock domain are synchronized before being combined with the SBITERR and DBITERR signals generated in the read clock domain.

Note that due to the differing read and write clock frequencies and the OR gate used to combine the signals, the number of read clock cycles that the SBITERR and DBITERR flags assert is not an accurate indicator of the number of errors found in the built-in FIFOs.

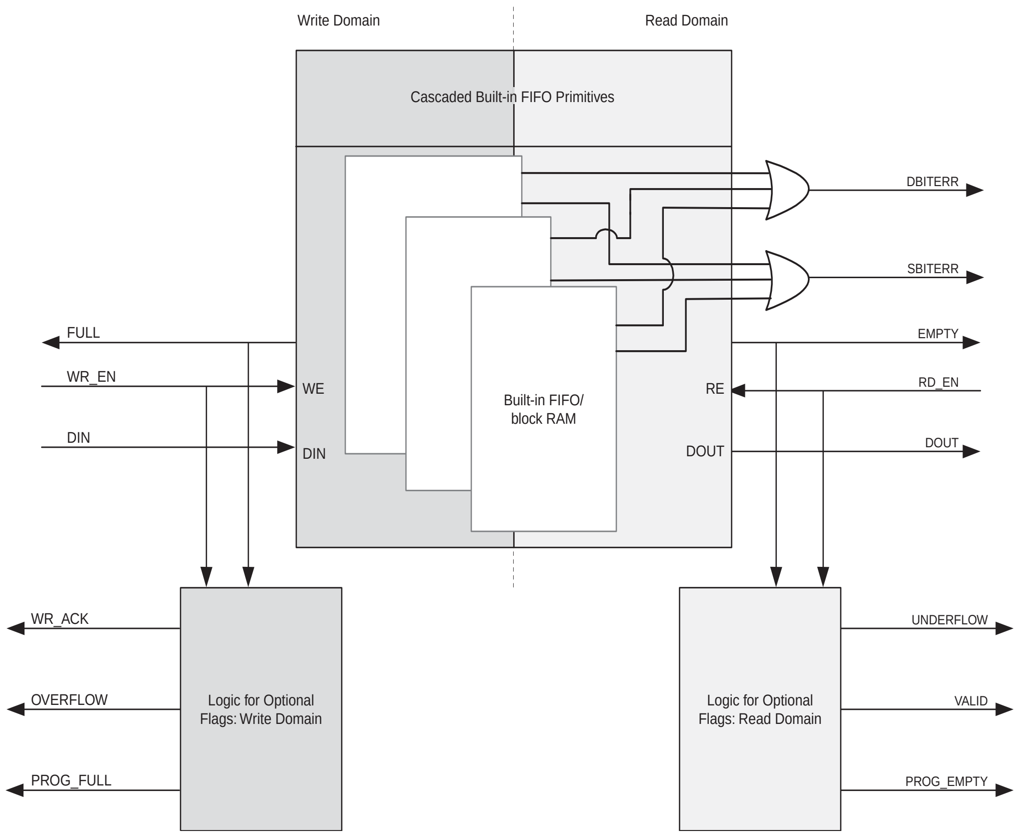


Figure 5-24: **SBITERR and DBITERR Outputs in the FIFO Generator Core**

Built-in Error Injection

Built-in Error Injection is supported for FIFOs configured with independent or common clock block RAM and built-in FIFOs in Kintex-7, Virtex-7, and Virtex-6 FPGAs. When ECC and Error Injection are enabled, the block RAM and built-in FIFO primitive used to create the FIFO is configured in the full ECC error injection mode, providing two additional inputs to the FIFO Generator core: INJECTSBITERR and INJECTDBITERR. These inputs indicate three possible results: no error injection, single bit error injection, or double bit error injection.

The ECC is calculated on a 64-bit wide data of Kintex-7, Virtex-7, and Virtex-6 FPGA ECC primitives. If the data width chosen by the user is not an integral multiple of 64 (for example, there are spare bits in any ECC primitive), then a double bit error (DBITERR) may indicate that one or more errors have occurred in the spare bits. So, the accuracy of the DBITERR signal cannot be guaranteed in this case. For example, if the user's data width is 16, then 48 bits of the ECC primitive are left empty. If two of the spare bits are corrupted, the DBITERR signal would be asserted even though the actual user data is not corrupt.

When INJECTSBITERR is asserted on a write operation, a single bit error is injected and SBITERR is asserted upon read operation of a specific write. When INJECTDBITERR is asserted on a write operation, a double bit error is injected and DBITERR is asserted upon read operation of a specific write. When both INJECTSBITERR and INJECTDBITERR are asserted on a write operation, a double bit error is injected and DBITERR is asserted upon read operation of a specific write. Figure 5-25 shows how the SBITERR and DBITERR outputs are generated in the FIFO Generator core.

Note: Reset is not supported by the FIFO/BRAM macros when using the ECC option. Therefore, outputs of the FIFO core (DOUT, DBITERR and SBITERR) will not be affected by reset, and they hold their previous values. See Reset Behavior for more details.

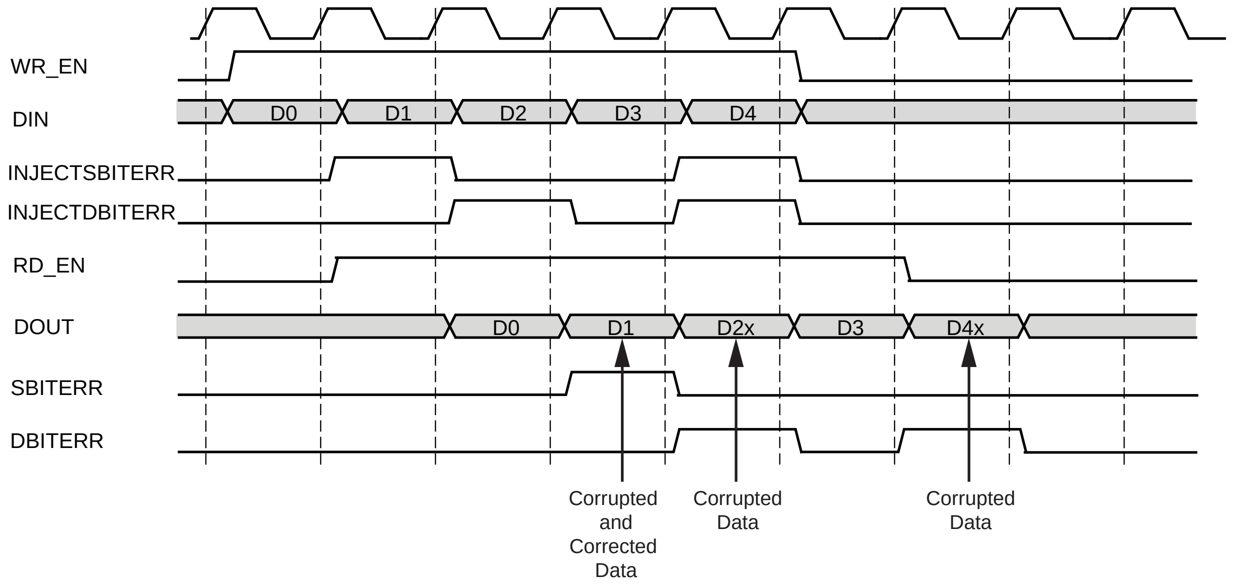


Figure 5-25: **Error Injection and Correction in Kintex-7, Virtex-7, and Virtex-6 FPGAs**

Reset Behavior

The FIFO Generator provides a reset input that resets all counters, output registers, and memories when asserted. For block RAM or distributed RAM implementations, resetting the FIFO is not required, and the reset pin can be disabled in the FIFO. There are two reset options: asynchronous and synchronous.

Asynchronous Reset (Enable Reset Synchronization Option is Selected)

The asynchronous reset (RST) input asynchronously resets all counters, output registers, and memories when asserted. 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.

Common/Independent Clock: Block RAM, Distributed RAM, and Shift RAM FIFOs

Table 5-7 defines the values of the output ports during power-up and reset state for block RAM, distributed RAM, and shift RAM FIFOs. Note that the underflow signal is dependent on RD_EN. If RD_EN is asserted and the FIFO is empty, underflow is asserted. The overflow signal is dependent on WR_EN. If WE_EN is asserted and the FIFO is full, overflow is asserted.

There are two asynchronous reset behaviors available for these FIFO configurations: Full flags reset to 1 and full flags reset to 0. The reset requirements and the behavior of the FIFO is different depending on the full flags reset value chosen.

Note: The reset is edge sensitive and not level sensitive. The synchronization logic looks for the rising edge of RST and creates an internal reset for the core. Note that the assertion of asynchronous reset immediately causes the core to go into a predetermine reset state - this is not dependent on any clock toggling. The reset synchronization logic is used to ensure that the logic in the different clock domains comes OUT of the reset mode at the same time - this is by synchronizing the deassertion of asynchronous reset to the appropriate clock domain. By doing this glitches and metastability can be avoided. This synchronization takes three clock cycles (write or read) after the asynchronous reset is detected on the rising edge read and write clock respectively. To avoid unexpected behavior, it is not recommended to drive/toggle `WR_EN/RD_EN` when `RST` or `FULL` is asserted/high.

Table 5-7: **FIFO Asynchronous Reset Values for Block RAM, Distributed RAM, and Shift RAM FIFOs**

Signal	Full Flags Reset Value of 1	Full Flags Reset Value of 0	Power-up Values
DOUT	DOUT Reset Value or 0	DOUT Reset Value or 0	Same as reset values
FULL	1[1]	0	0
ALMOST FULL	1[1]	0	0
EMPTY	1	1	1
ALMOST EMPTY	1	1	1
VALID	0 (active high) or 1 (active low)	0 (active high) or 1 (active low)	0 (active high) or 1 (active low)
WR_ACK	0 (active high) or 1 (active low)	0 (active high) or 1 (active low)	0 (active high) or 1 (active low)
PROG_FULL	1[1]	0	0
PROG_EMPTY	1	1	1
RD_DATA_COUNT	0	0	0
WR_DATA_COUNT	0	0	0

Notes:

1. When reset is asserted, the FULL flags are asserted to prevent writes to the FIFO during reset.

Full Flags Reset Value of 1

In this configuration, the FIFO requires a minimum asynchronous reset pulse of 1 write clock period (`WR_CLK/CLK`). After reset is detected on the rising clock edge of write clock, 3 write clock periods are required to complete proper reset synchronization. During this time, the `FULL`, `ALMOST_FULL`, and `PROG_FULL` flags are asserted. After reset is deasserted, these flags deassert after 3 clock period (WR_CLK/CLK) and the FIFO is now ready for writing.

The `FULL` and `ALMOST_FULL` flags are asserted to ensure that no write operations occur when the FIFO core is in the reset state. After the FIFO exits the reset state and is ready for writing, the `FULL` and `ALMOST_FULL` flags deassert; this occurs approximately three clock cycles after the deassertion of asynchronous reset. See Figure 5-26 and Figure 5-27 for example behaviors. Note that the power-up values for this configuration are different from the reset state value.

Figure 5-26 shows an example timing diagram for when the reset pulse is one clock duration.

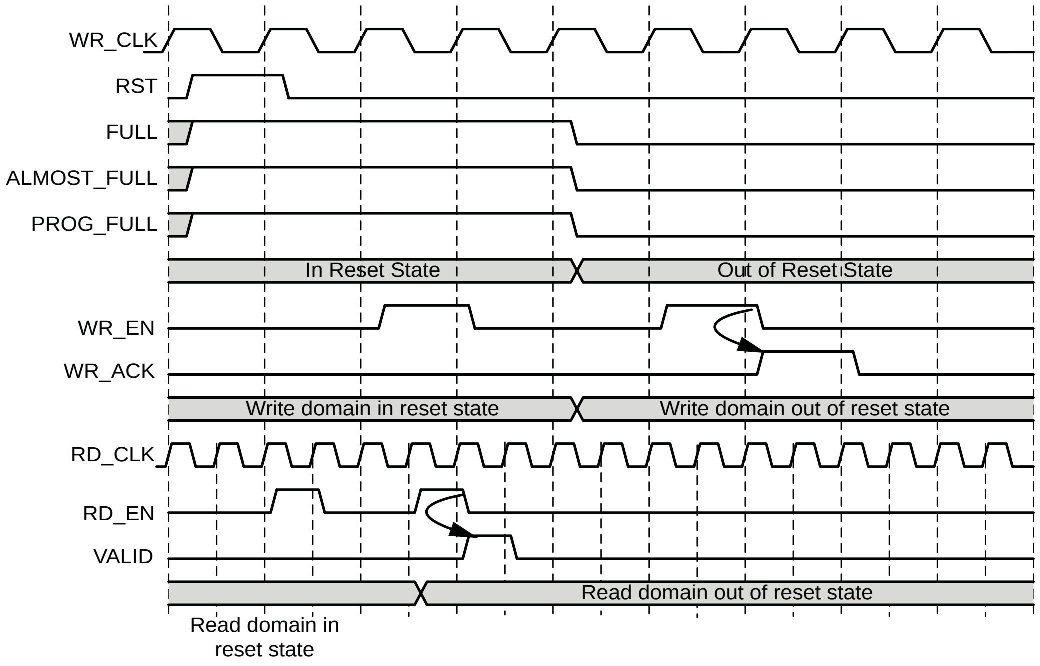

WR_CLK
RST
FULL
ALMOST_FULL
PROG_FULL
In Reset State
Out of Reset State
WR_EN
WR_ACK
Write domain in reset state
Write domain out of reset state
RD_CLK
RD_EN
VALID
Read domain out of reset state
Read domain in reset state

Figure 5-26: **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 1 for the Reset Pulse of One Clock**

Figure 5-27 shows an example timing diagram for when the reset pulse is longer than one clock duration.

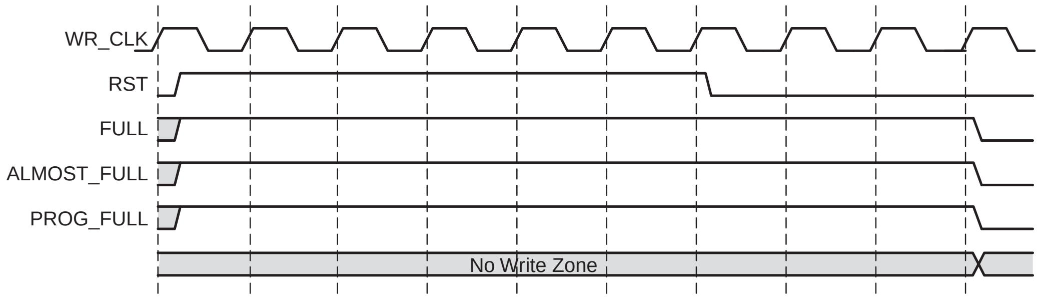


Figure 5-27: **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 1 for the Reset Pulse of More Than One Clock**

Full Flags Reset Value of 0

In this configuration, the FIFO requires a minimum asynchronous reset pulse of 1 write clock cycle to complete the proper reset synchronization. At reset, FULL, ALMOST_FULL and PROG_FULL flags are deasserted. After the FIFO exits the reset synchronization state, the FIFO is ready for writing; this occurs approximately three clock cycles after the assertion of asynchronous reset. See Figure 5-28 for example behavior.

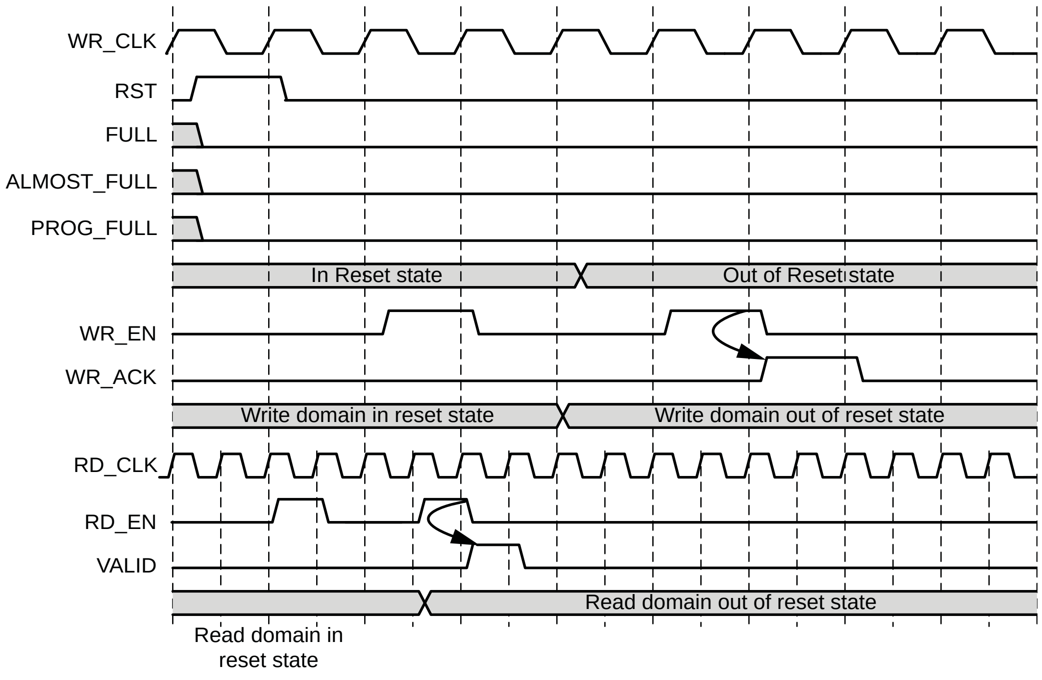

WR_CLK
RST
FULL
ALMOST_FULL
PROG_FULL
In Reset state
Out of Reset state
WR_EN
WR_ACK
Write domain in reset state
Write domain out of reset state
RD_CLK
RD_EN
VALID
Read domain out of reset state
Read domain in reset state

Figure 5-28: **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 0**

Common/Independent Clock: Built-in

Table 5-7 defines the values of the output ports during power-up and reset state for Built-in FIFOs. The DOUT reset value is supported only for Kintex-7, Virtex-7 and Virtex-6 common clock Built-In FIFOs with the embedded register option selected. The Kintex-7 and Virtex-7 FPGA Built-In FIFOs require an asynchronous reset pulse of at least 5 read and write clock cycles. To be consistent across all built-in FIFO configurations, it recommended to give an asynchronous reset pulse of at least 5 read and write clock cycles for Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA Built-in FIFOs.

During reset, the RD_EN and WR_EN ports are required to be deasserted (no read or write operation can be performed). Assertion of reset causes the FULL and PROG_FULL flags to deassert and EMPTY and PROG_EMPTY flags to assert. After asynchronous reset is released, the core exits the reset state and is ready for writing. See Figure 5-29 for example behavior.

Note that the underflow signal is dependent on RD_EN. If RD_EN is asserted and the FIFO is empty, underflow is asserted. The overflow signal is dependent on WR_EN. If WE_EN is asserted and the FIFO is full, overflow is asserted.

Table 5-8: **Asynchronous FIFO Reset Values for Built-in FIFO**

Signal	Built-in FIFO Reset Values	Power-up Values
DOUT	Last read value	Content of memory at location 0
FULL	0	0
EMPTY	1	1
VALID	0 (active high) or 1 (active low)	0 (active high) or 1 (active low)

Table 5-8: **Asynchronous FIFO Reset Values for Built-in FIFO**

PROG_FULL	0	0
PROG_EMPTY	1	1

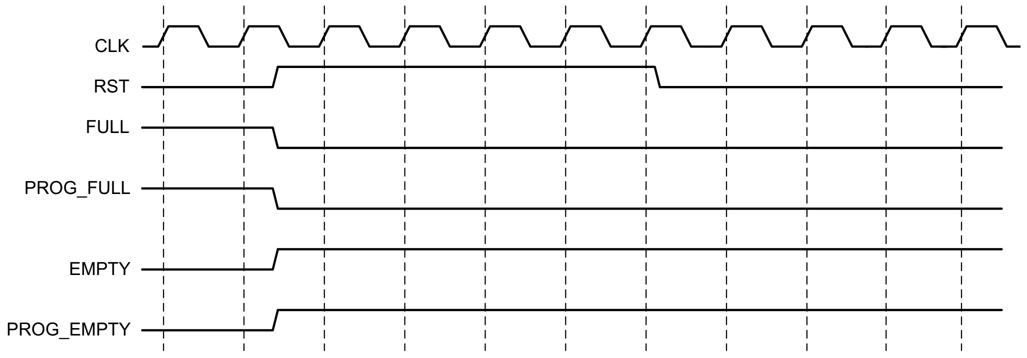


Figure 5-29: **Built-in FIFO, Asynchronous Reset Behavior**

Synchronous Reset

The synchronous reset input (SRST or WR_RST/RD_RST synchronous to WR_CLK/RD_CLK domain) is only available for the block RAM, distributed RAM, or shift RAM implementation of the common/independent clock FIFOs.

Common Clock Block, Distributed, or Shift RAM FIFOs

The synchronous reset (SRST) synchronously resets all counters, output registers and memories when asserted. Because the reset pin is synchronous to the input clock and there is only one clock domain in the FIFO, no additional synchronization logic is necessary.

Figure 5-32 illustrates the flags following the release of SRST.

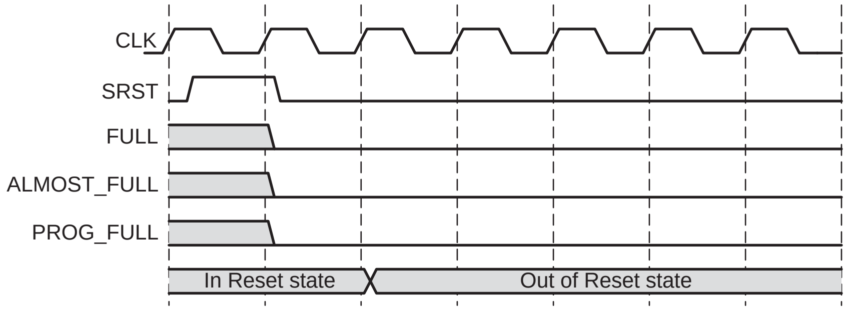


Figure 5-32: **Synchronous Reset: FIFO with a Common Clock**

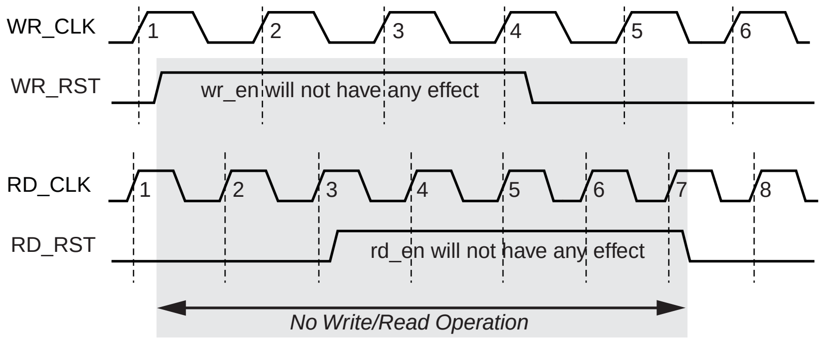
Independent Clock Block and Distributed RAM FIFOs (Enable Reset Synchronization Option not Selected)

The synchronous reset (WR_RST/RD_RST) synchronously resets all counters, output registers of respective clock domain when asserted. Because the reset pin is synchronous to the respective clock domain, no additional synchronization logic is necessary.

If one reset (WR_RST/RD_RST) is asserted, the other reset must also be applied. The time at which the resets are asserted/de-asserted may differ, and during this period the FIFO outputs become invalid. To avoid unexpected behavior, it is not recommended to perform write or read operations from the assertion of the first reset to the de-assertion of the last reset.

Note: For FIFOs built with First-Word-Fall-Through and ECC configurations, the SBITERR and DBITERR may be high until a valid read is performed after the de-assertion of both `WR_RST` and `RD_RST`.

Figure 5-33 and Figure 5-34 illustrate the rules to be considered.



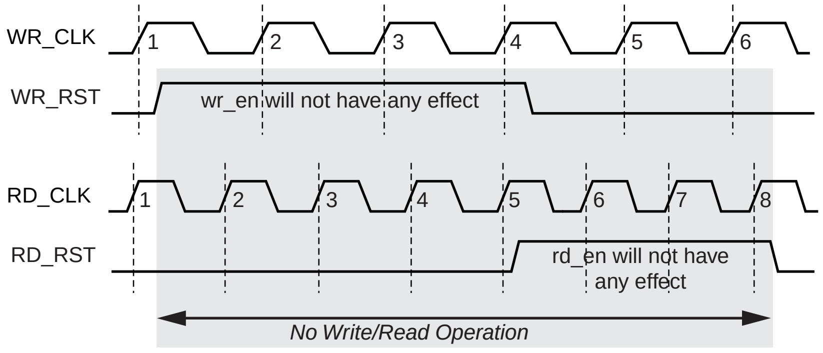
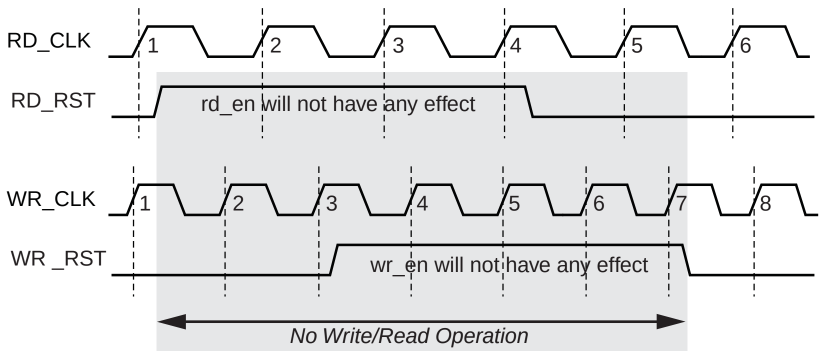


Figure 5-33: **Synchronous Reset: FIFO with Independent Clock - WR_RST then RD_RST**


RD_CLK
RD_RST
rd_en will not have any effect
WR_CLK
WR _RST
wr_en will not have any effect
No Write/Read Operation

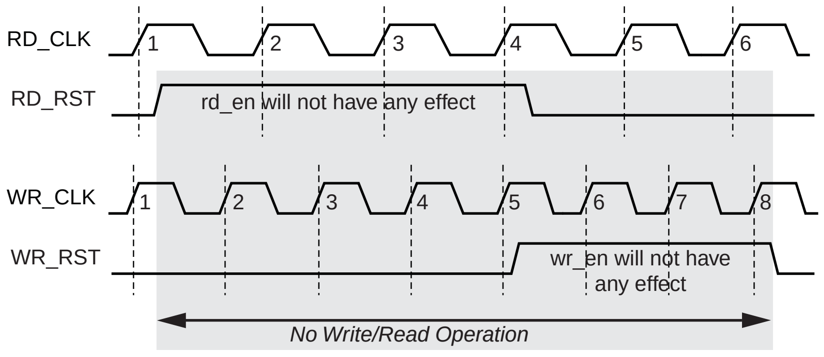

RD_CLK
RD_RST
rd_en will not have any effect
WR_CLK
WR_RST
wr_en will not have any effect
No Write/Read Operation

Figure 5-34: **Synchronous Reset: FIFO with Independent Clock - RD_RST then WR_RST**

Table 5-9 defines the values of the output ports during power-up and the reset state. If the user does not specify a DOUT reset value, it defaults to 0. The FIFO requires a reset pulse of only 1 clock cycle. The FIFOs are available for transaction on the clock cycle after the reset is released. The power-up values for the synchronous reset are the same as the reset state.

Note that the underflow signal is dependent on `RD_EN`. If `RD_EN` is asserted and the FIFO is empty, underflow is asserted. The overflow signal is dependent on `WR_EN`. If `WE_EN` is asserted and the FIFO is full, overflow is asserted.

Table 5-9: **Synchronous FIFO Reset and Power-up Values**

Signal	Block Memory and Distributed Memory Values of Output Ports During Reset and Power-up
DOUT	DOUT Reset Value or 0
FULL	0
ALMOST FULL	0
EMPTY	1
ALMOST EMPTY	1
VALID	0 (active high) or 1 (active low)

Table 5-9: **Synchronous FIFO Reset and Power-up Values *(Cont'd)***

WR_ACK	0 (active high) or 1 (active low)
PROG_FULL	0
PROG_EMPTY	0
RD_DATA_COUNT	0
WR_DATA_COUNT	0

Actual FIFO Depth

Of critical importance is the understanding that the *effective* or *actual* depth of a FIFO is *not necessarily* consistent with the *depth* selected in the GUI, because the actual depth of the FIFO depends on its implementation and the features that influence its implementation. In the FIFO Generator GUI, the actual depth of the FIFO is reported: the following section provides formulas or calculations used to report this information.

Block RAM, Distributed RAM and Shift RAM FIFOs

The actual FIFO depths for the block RAM, distributed RAM, and shift RAM FIFOs are influenced by the following features that change its implementation:

- Common or Independent Clock
- Standard or FWFT Read Mode
- Symmetric or Non-symmetric Port Aspect Ratio

Depending on how a FIFO is configured, the calculation for the actual FIFO depth varies.

- Common Clock FIFO in Standard Read Mode

  actual_write_depth = gui_write_depth

actual_read_depth = gui_read_depth

- Common Clock FIFO in FWFT Read Mode

  actual_write_depth = gui_write_depth +2

  actual_read_depth = gui_read_depth +2

- Independent Clock FIFO in Standard Read Mode

  actual_write_depth = gui_write_depth - 1

  actual_read_depth = gui_read_depth - 1

- Independent Clock FIFO in FWFT Read Mode

  actual_write_depth = (gui_write_depth - 1) + (2*round_down(gui_write_depth/gui_read_depth))

  actual_read_depth = gui_read_depth + 1

Notes

1. Gui_write_depth = actual write (input) depth selected in the GUI
2. Gui_read_depth = actual read (output) depth selected in the GUI
3. Non-symmetric port aspect ratio feature (gui_write_depth not equal to gui_read_depth) is only supported in block RAM based FIFOs.

Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA Built-In FIFOs

The actual FIFO depths for the Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA built-in FIFOs are influenced by the following features, which change its implementation:

- Common or Independent Clock
- Standard or FWFT Read Mode
- Built-In FIFO primitive used in implementation (minimum depth is 512)

Depending on how a FIFO is configured, the calculation for the actual FIFO depth varies.

- Independent Clock FIFO in Standard Read Mode

actual_write_depth = (primitive_depth+2)*(N-1) + (primitive_depth+1)

- Independent Clock FIFO in FWFT Read Mode

actual_write_depth = (primitive_depth+2)*N

- Common Clock FIFO in Standard Read Mode

actual_write_depth = (primitive_depth+1)*(N-1) + primitive_depth

- Common Clock FIFO in FWFT Read Mode

actual_write_depth = (primitive_depth+1)*N

Notes

1. primitive_depth = depth of the primitive used to implement the FIFO; this information is reported in the GUI
2. N = number of primitive cascaded in depth or roundup (gui_write_depth/primitive_depth)

Virtex-4 FPGA Built-In FIFOs

The actual FIFO depths for the Virtex-4 FPGA Built-in FIFOs are influenced by the following features, which change its implementation:

- Read and Write Clock Frequencies
- Built-In FIFO primitive used in implementation (minimum depth is 512)

Depending on how a FIFO is configured, the calculation for the actual FIFO depth varies.

- Common/Independent Clock FIFO in Standard Read Mode and RD_CLK frequency > WR_CLK frequency

actual_write_depth = primitive_depth+17

- Common/Independent Clock FIFO in Standard Read Mode and RD_CLK frequency <= WR_CLK frequency

actual_write_depth = primitive_depth+17

Note: primitive_depth = depth of the primitive used to implement the FIFO. For more details, see UG070, *Virtex-4 FPGA User Guide*.

Latency

This section defines the latency in which different output signals of the FIFO are updated in response to read or write operations.

Note: Latency is defined as the number of clock edges after a read or write operation occur before the signal is updated. Example: if latency is 0, that means that the signal is updated at the clock edge in which the operation occurred, as shown in Figure 5-35 in which WR_ACK is getting updated in which WR_EN is high.

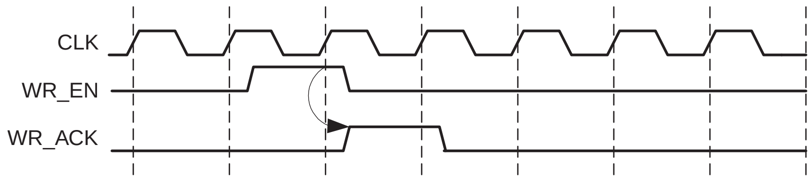


Figure 5-35: **Latency 0 Timing**

Non-Built-in FIFOs: Common Clock and Standard Read Mode Implementations

Table 5-10 defines the write port flags update latency due to a write operation for non-Built-in FIFOs such as block RAM, distributed RAM, and shift RAM FIFOs.

Table 5-10: **Non-Built-in FIFOs, Common Clock and Standard Read Mode Implementations: Write Port Flags Update Latency Due to Write Operation**

Signals	Latency (CLK)
FULL	0
ALMOST_FULL	0
PROG_FULL	1
WR_ACK	0
OVERFLOW	0

Table 5-11 defines the read port flags update latency due to a read operation.

Table 5-11: **Non-Built-in FIFOs, Common Clock and Standard Read Mode Implementations: Read Port Flags Update Latency Due to Read Operation**

Signals	Latency (CLK)
EMPTY	0
ALMOST_EMPTY	0
PROG_EMPTY	1
VALID	0
UNDERFLOW	0
DATA_COUNT	0

Table 5-12 defines the write port flags update latency due to a read operation.

Table 5-12: **Non-Built-in FIFOs, Common Clock and Standard Read Mode Implementations: Write Port Flags Update Latency Due to Read Operation**

Signals	Latency (CLK)
FULL	0
ALMOST_FULL	0
PROG_FULL	1
WR_ACK[a]	N/A
OVERFLOW[a]	N/A

a. Write handshaking signals are only impacted by a write operation.

Table 5-13 defines the read port flags update latency due to a write operation.

Table 5-13: **Non-Built-in FIFOs, Common Clock and Standard Read Mode Implementations: Read Port Flags Update Latency Due to Write Operation**

Signals	Latency (CLK)
EMPTY	0
ALMOST_EMPTY	0
PROG_EMPTY	1
VALID[a]	N/A
UNDERFLOW[a]	N/A
DATA_COUNT	0

a. Read handshaking signals are only impacted by a read operation.

Non-Built-in FIFOs: Common Clock and FWFT Read Mode Implementations

Table 5-14 defines the write port flags update latency due to a write operation for non-Built-in FIFOs such as block RAM, distributed RAM, and shift RAM FIFOs.

Table 5-14: **Non-Built-in FIFOs, Common Clock and FWFT Read Mode Implementations: Write Port Flags Update Latency due to Write Operation**

Signals	Latency (CLK)
FULL	0
ALMOST_FULL	0
PROG_FULL	1
WR_ACK	0
OVERFLOW	0

Table 5-15 defines the read port flags update latency due to a read operation.

Table 5-15: **Non-Built-in FIFOs, Common Clock and FWFT Read Mode Implementations: Read Port Flags Update Latency due to Read Operation**

Signals	Latency (CLK)
EMPTY	0
ALMOST_EMPTY	0
PROG_EMPTY	1
VALID	0
UNDERFLOW	0
DATA_COUNT	0

Table 5-16 defines the write port flags update latency due to a read operation.

Table 5-16: **Non-Built-in FIFOs, Common Clock and FWFT Read Mode Implementations: Write Port Flags Update Latency Due to Read Operation**

Signals	Latency (CLK)
FULL	0
ALMOST_FULL	0
PROG_FULL	1
WR_ACK[a]	N/A
OVERFLOW[a]	N/A

a. Write handshaking signals are only impacted by a write operation.

Table 5-17 defines the read port flags update latency due to a write operation.

Table 5-17: **Non-Built-in FIFOs, Common Clock and FWFT Read Mode Implementations: Read Port Flags Update Latency Due to Write Operation**

Signals	Latency (CLK)
EMPTY	2
ALMOST_EMPTY	1
PROG_EMPTY	1
VALID[a]	N/A
UNDERFLOW[a]	N/A
DATA_COUNT	0

a. Read handshaking signals are only impacted by a read operation.

Non-Built-in FIFOs: Independent Clock and Standard Read Mode Implementations

Table 5-18 defines the write port flags update latency due to a write operation.

Table 5-18: **Non-Built-in FIFOs, Independent Clock and Standard Read Mode Implementations: Write Port Flags Update Latency Due to a Write Operation**

Signals	Latency (WR_CLK)
FULL	0
ALMOST_FULL	0
PROG_FULL	1
WR_ACK	0
OVERFLOW	0
WR_DATA_COUNT	1

Table 5-19 defines the read port flags update latency due to a read operation.

Table 5-19: **Non-Built-in FIFOs, Independent Clock and Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

Signals	Latency (RD_CLK)
EMPTY	0
ALMOST_EMPTY	0
PROG_EMPTY	1
VALID	0
UNDERFLOW	0
RD_DATA_COUNT	1

Table 5-20 defines the write port flags update latency due to a read operation.

Table 5-20: **Non-Built-in FIFOs, Independent Clock and Standard Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

Signals	Latency
FULL	1 RD_CLK + 4 WR_CLK (+1 WR_CLK)[a]
ALMOST_FULL	1 RD_CLK + 4 WR_CLK (+1 WR_CLK)[a]
PROG_FULL	1 RD_CLK + 5 WR_CLK (+1 WR_CLK)[a]
WR_ACK[b]	N/A
OVERFLOW[b]	N/A
WR_DATA_COUNT	1 RD_CLK + 4 WR_CLK (+1 WR_CLK)[a]

a. The crossing clock domain logic in independent clock FIFOs introduces a 1 WR_CLK uncertainty to the latency calculation.

b. Write handshaking signals are only impacted by a write operation.

Table 5-21 defines the read port flags update latency due to a write operation.

Table 5-21: **Non-Built-in FIFOs, Independent Clock and Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

Signals	Latency
EMPTY	1 WR_CLK + 4 RD_CLK (+1 RD_CLK)[a]
ALMOST_EMPTY	1 WR_CLK + 4 RD_CLK (+1 RD_CLK)[a]
PROG_EMPTY	1 WR_CLK + 5 RD_CLK (+1 RD_CLK)[a]
VALID[b]	N/A
UNDERFLOW[b]	N/A
RD_DATA_COUNT	1 WR_CLK + 4 RD_CLK (+1 RD_CLK)[a]

Note: Read handshaking signals only impacted by read operation.

a. The crossing clock domain logic in independent clock FIFOs introduces a 1 RD_CLK uncertainty to the latency calculation.

b. Read handshaking signals are only impacted by a read operation.

Non-Built-in FIFOs: Independent Clock and FWFT Read Mode Implementations

Table 5-22 defines the write port flags update latency due to a write operation.

Table 5-22: **Non-Built-in FIFOs, Independent Clock and FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Write Operation**

Signals	Latency (WR_CLK)
FULL	0
ALMOST_FULL	0
PROG_FULL	1
WR_ACK	0
OVERFLOW	0
WR_DATA_COUNT	1

Table 5-23 defines the read port flags update latency due to a read operation.

Table 5-23: **Non-Built-in FIFOs, Independent Clock and FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

Signals	Latency (RD_CLK)
EMPTY	0
ALMOST_EMPTY	0
PROG_EMPTY	1
VALID	0
UNDERFLOW	0
RD_DATA_COUNT	1

Table 5-24 defines the write port flags update latency due to a read operation.

Table 5-24: **Non-Built-in FIFOs, Independent Clock and FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

Signals	Latency
FULL	1 RD_CLK + 4 WR_CLK (+1 WR_CLK)[a]
ALMOST_FULL	1 RD_CLK + 4 WR_CLK (+1 WR_CLK)[a]
PROG_FULL	1 RD_CLK + 5 WR_CLK (+1 WR_CLK)[a]
WR_ACK[b]	N/A
OVERFLOW[b]	N/A
WR_DATA_COUNT	1 RD_CLK + 4 WR_CLK (+1 WR_CLK)[a]

a. The crossing clock domain logic in independent clock FIFOs introduces a 1 WR_CLK uncertainty to the latency calculation.

b. Write handshaking signals are only impacted by a write operation.

Table 5-25 defines the read port flags update latency due to a write operation.

Table 5-25: **Non-Built-in FIFOs, Independent Clock and FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

Signals	Latency
EMPTY	1 WR_CLK + 6 RD_CLK (+1 RD_CLK)[a]
ALMOST_EMPTY	1 WR_CLK + 6 RD_CLK (+1 RD_CLK)[a]
PROG_EMPTY	1 WR_CLK + 5 RD_CLK (+1 RD_CLK)[a]
VALID[b]	N/A
UNDERFLOW[b]	N/A
RD_DATA_COUNT	1 WR_CLK + 4 RD_CLK (+1 RD_CLK)[a] + [2 RD_CLK (+1 RD_CLK)][c]

Note: Read handshaking signals only impacted by read operation.

a. The crossing clock domain logic in independent clock FIFOs introduces a 1 RD_CLK uncertainty to the latency calculation.

b. Read handshaking signals are only impacted by a read operation.

c. This latency is the worst-case latency. The addition of the [2 RD_CLK (+1 RD_CLK)] latency depends on the status of the EMPTY and ALMOST_EMPTY flags.

Built-in FIFOs: Common Clock and Standard Read Mode Implementations

Note: N is the number of primitives cascaded in depth; this can be calculated by dividing the GUI depth by the primitive depth. The term "Built-in FIFOs" refers to the hard FIFO macros of Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs.

Table 5-26 defines the write port flags update latency due to a write operation.

Table 5-26: **Common Clock Built-in FIFOs with Standard Read Mode Implementations: Write Port Flags Update Latency Due to Write Operation**

Signals	Latency (CLK)
FULL	0
PROG_FULL	1
WR_ACK	0
OVERFLOW	0

Table 5-27 defines the read port flags update latency due to a read operation.

Table 5-27: **Common Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to Read Operation**

Signals	Latency (CLK)
EMPTY	0
PROG_EMPTY	1
VALID	0
UNDERFLOW	0

Table 5-28 defines the write port flags update latency due to a read operation.

Table 5-28: **Common Clock Built-in FIFOs with Standard Read Mode Implementations: Write Port Flags Update Latency Due to Read Operation**

Signals	Latency (CLK)
FULL	(N-1)
PROG_FULL	N
WR_ACK[a]	N/A
OVERFLOW[a]	N/A

a. Write handshaking signals are only impacted by a write operation.

Table 5-29 defines the read port flags update latency due to a write operation.

Table 5-29: **Common Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to Write Operation**

Signals	Latency (CLK)
EMPTY	(N-1)*2
PROG_EMPTY	(N-1)*2+1
VALID[a]	N/A
UNDERFLOW[a]	N/A

Note: Read handshaking signals only impacted by read operation.

a. Read handshaking signals are only impacted by a read operation.

Built-in FIFOs: Common Clock and FWFT Read Mode Implementations

Note: N is the number of primitives cascaded in depth; this can be calculated by dividing the GUI depth by the primitive depth. The term “Built-in FIFOs” refers to the hard FIFO macros of Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs.

Table 5-30 defines the write port flags update latency due to a write operation.

Table 5-30: **Common Clock Built-in FIFOs with FWFT Read Mode Implementations: Write Port Flags Update Latency Due to Write Operation**

Signals	Latency (CLK)
FULL	0
PROG_FULL	1
WR_ACK	0
OVERFLOW	0

Table 5-31 defines the read port flags update latency due to a read operation.

Table 5-31: **Common Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

Signals	Latency (CLK)
EMPTY	0
PROG_EMPTY	1
VALID	0
UNDERFLOW	0

Table 5-32 defines the write port flags update latency due to a read operation.

Table 5-32: **Common Clock Built-in FIFOs with FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

Signals	Latency (CLK)
FULL	(N-1)
PROG_FULL[a]	N

Table 5-32: **Common Clock Built-in FIFOs with FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation *(Cont'd)***

WR_ACK[a]	N/A
OVERFLOW	N/A

a. Write handshaking signals are only impacted by a write operation.

Table 5-33 defines the read port flags update latency due to a write operation

Table 5-33: **Common Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

Signals	Latency (CLK)
EMPTY	((N-1)*2+1)
PROG_EMPTY	((N-1)*2+1)
VALID[a]	N/A
UNDERFLOW[a]	N/A

a. Read handshaking signals are only impacted by a read operation.

Built-in FIFOs: Independent Clocks and Standard Read Mode Implementations

Note: N is the number of primitives cascaded in depth; this can be calculated by dividing the GUI depth by the primitive depth. `Faster_Clk` is the clock domain, either `RD_CLK` or `WR_CLK`, that has a larger frequency. The term "Built-in FIFOs" refers to the hard FIFO macros of Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs.

Table 5-34 defines the write port flags update latency due to a write operation.

Table 5-34: **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Write Port Flags Update Latency Due to a Write Operation**

Signals	Latency (WR_CLK)
FULL	0
PROG_FULL	1
WR_ACK	0
OVERFLOW	0

Table 5-35 defines the read port flags update latency due to a read operation.

Table 5-35: **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

Signals	Latency (RD_CLK)
EMPTY	0
PROG_EMPTY	1
VALID	0
UNDERFLOW	0

Table 5-36 defines the write port flags update latency due to a read operation.

Table 5-36: **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

Signals	Latency
FULL	(N-1)*5 faster_clk + 4 WR_CLK
PROG_FULL	(N-1)*4 faster_clk + 3 WR_CLK
WR_ACK[a]	N/A
OVERFLOW[a]	N/A

a. Write handshaking signals are only impacted by a write operation.

Table 5-37 defines the read port flags update latency due to a write operation.

Table 5-37: **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

Signals	Latency
EMPTY	(N-1)*5 faster_clk + 3 RD_CLK
PROG_EMPTY	(N-1)*4 faster_clk + 3 RD_CLK
VALID[a]	N/A
UNDERFLOW[a]	N/A

a. Read handshaking signals are only impacted by a read operation.

Built-in FIFOs: Independent Clocks and FWFT Read Mode Implementations

Note: N is the number of primitives cascaded in depth, which can be calculated by dividing the GUI depth by the primitive depth. `Faster_Clk` is the clock domain, either `RD_CLK` or `WR_CLK`, that has a larger frequency. The term "Built-in FIFOs" refers to the hard FIFO macros of Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs.

Table 5-38 defines the write port flags update latency due to a write operation.

Table 5-38: **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Write Operations**

Signals	Latency (WR_CLK)
FULL	0
PROG_FULL	1
WR_ACK	0
OVERFLOW	0

Table 5-39 defines the read port flags update latency due to a read operation.

Table 5-39: **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

Signals	Latency (RD_CLK)
EMPTY	0
PROG_EMPTY	1
VALID	0
UNDERFLOW	0

Table 5-40 defines the write port flags update latency due to a read operation.

Table 5-40: **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

Signals	Latency
FULL	(N-1)*5 faster_clk + 4 WR_CLK
PROG_FULL	(N-1)*4 faster_clk + 3 WR_CLK
WR_ACK[a]	N/A
OVERFLOW[a]	N/A

a. Write handshaking signals are only impacted by a write operation.

Table 5-41 defines the read port flags update latency due to a write operation.

Table 5-41: **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

Signals	Latency
EMPTY	(N-1)*5 faster_clk + 4 RD_CLK

Table 5-41: **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation *(Cont'd)***

PROG_EMPTY	(N-1)*4 faster_clk + 3 RD_CLK
VALID[a]	N/A
UNDERFLOW[a]	N/A

a. Read handshaking signals are only impacted by a read operation.

Virtex-4 FPGA Built-in FIFO

The Virtex-4 FPGA supports only one Built-in FIFO with a data width of 4, 9, 18 or 36. For more details for the write and read port flags update latency, see UG070, *Virtex-4 FPGA User Guide*.

Chapter 6

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. Two reset pins are available, asynchronous (`RST`) and synchronous (`SRST`), and both clear the internal counters and output registers.

- For asynchronous reset, 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. To avoid unexpected behavior, it is not recommended to drive/toggle `WR_EN`/`RD_EN` when `RST` is asserted/high .
- For common clock block and distributed RAM synchronous reset, because the reset pin is synchronous to the input clock and there is only one clock domain in the FIFO, no additional synchronization logic is needed.
- For independent clock block and distributed RAM synchronous reset, because the reset pin (`WR_RST`/`RD_RST`) is synchronous to the respective clock domain, no additional synchronization logic is needed. However, it is recommended to follow these rules to avoid unexpected behavior:
  - If `WR_RST` is applied, then `RD_RST` must also be applied and vice versa.
  - No write or read operations should be performed until both clock domains are reset.

The generated FIFO core will be initialized after reset to a known state. For details about reset values and behavior, see Reset Behavior in Chapter 5 of this guide.

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. For more information see Simultaneous Assertion of Full and Empty Flag

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. For more information see Simultaneous Assertion of Full and Empty Flag

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 Programmable Flags in Chapter 5 of this guide for more information about the latency and behavior of the programmable flags.

Simultaneous Assertion of Full and Empty Flag

For independent clock FIFO, there are delays in the assertion/deassertion of the full and empty flags due to cross clock domain logic. These delays may cause unexpected FIFO behavior like full and empty asserting at the same time. To avoid this, the following A and B equations must be true.

A) Time it takes to update full flag due to read operation < time it takes to empty a full FIFO

B) Time it takes to update empty flag due to write operation < time it takes to fill an empty FIFO

For example, assume the following configurations:

Independent clock (non built-in), standard FIFO

write clock frequency = 3MHz, wr_clk_period = 333 ns

read clock frequency = 148 MHz, rd_clk_period = 6.75 ns

write depth = read depth = 20

actual_wr_depth = actual_rd_depth = 19 (as mentioned in Actual FIFO Depth in Chapter 5)

Apply equation A:

Time it takes to update full flag due to read operation < time it takes to empty a full FIFO = 1*rd_clk_period + 5*wr_clk_period < actual_rd_depth*rd_clk_period

1*6.75 + 5*333 < 19*6.75

1671.75 ns < 128.5 ns --> Equation VIOLATED!

Note: Left side equation is the latency of full flag updating due to read operation as mentioned in Table 5-20.

Conclusion: Violation of this equation proves that for this design, when a FULL FIFO is read from continuously, the empty flag asserts before the full flag deasserts due to the read operations that occurred.

Apply Equation B:

Time it takes to update empty flag due to write operation < time it takes to fill an empty FIFO

1*wr_clk_period + 5*rd_clk_period < actual_wr_depth*wr_clk_period

1*333 + 5*6.75 < 19*333

366.75 ns < 6327 ns --> Equation MET!

Note: Left side equation is the latency of empty flag updating due to write operation as mentioned in Table 5-21.

Conclusion: Because this equation is met for this design, an EMPTY FIFO that is written into continuously has its empty flag deassert before the full flag is asserted.

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).

See Data Counts in Chapter 5 of this guide for details about the latency and behavior of the data count flags.

Setup and Hold Time Violations

When generating a FIFO with independent clock domains (whether a DCM is used to derive the write/read clocks or not), the core internally synchronizes the write and read clock domains. For this reason, setup and hold time violations are 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.

Alternatively, there are two ways to disable these expected setup and hold time violations due to data synchronization between clock domains:

- Add the following constraint to your design – this constraint sets a timing constraint to the synchronization logic by requiring a maximum set of delays. The maximum delays used is defined by 2x of the slower clock period.

```
NET <fifo_instance>/grf.rf/gcx.clkx/wr_pntr_gc<0> MAXDELAY = 12 ns;
NET <fifo_instance>/grf.rf/gcx.clkx/wr_pntr_gc<1> MAXDELAY = 12 ns;
  ...
NET <fifo_instance>/grf.rf/gcx.clkx/wr_pntr_gc<9> MAXDELAY = 12 ns;

NET <fifo_instance>/grf.rf/gcx.clkx/rd_pntr_gc<0> MAXDELAY = 12 ns;
NET <fifo_instance>/grf.rf/gcx.clkx/rd_pntr_gc<1> MAXDELAY = 12 ns;
  ...
NET <fifo_instance>/grf.rf/gcx.clkx/rd_pntr_gc<9> MAXDELAY = 12 ns;
```

- Add the following constraint to your design – this constraint directs the tool to ignore the appropriate paths that are part of the synchronization logic:

```
NET <fifo_instance>/grf.rf/gcx.clkx/wr_pntr_gc<0> TIG;
NET <fifo_instance>/grf.rf/gcx.clkx/wr_pntr_gc<1> TIG;
  ...
NET <fifo_instance>/grf.rf/gcx.clkx/wr_pntr_gc<9> TIG;

NET <fifo_instance>/grf.rf/gcx.clkx/rd_pntr_gc<0> TIG;
NET <fifo_instance>/grf.rf/gcx.clkx/rd_pntr_gc<1> TIG;
  ...
NET <fifo_instance>/grf.rf/gcx.clkx/rd_pntr_gc<9> TIG;
```

- If distributed RAM FIFO is used, the following constraints may also be required to improve the timing.

```
INST “<fifo_instance>/grf.rf/mem/gdm.dm/Mram*" TNM= RAMSOURCE;
INST “<fifo_instance>/grf.rf/mem/gdm.dm/dout*” TNM= FFDEST;
TIMESPEC TS_RAM_FF= FROM “RAMSOURCE” TO “FFDEST” <<one read clock
period>> DATAPATHONLY;
```

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. The models are available in both VHDL and Verilog®. Both types of models are described in detail in this chapter.

To choose a model:

1. Open the CORE Generator.
2. Select Options from the Project drop-down list.
3. Click the Generation tab.
4. Choose to generate a behavioral model or a structural model.

Behavioral Models

Important! The behavioral models provided do not model synchronization delay, and are designed to reproduce the behavior and functionality of the FIFO Generator. The models maintain the assertion/deassertion of the output signals to match the FIFO Generator.

The behavioral models are functionally correct, and will represent the behavior of the configured FIFO. The write-to-read latency and the behavior of the status flags will accurately match the actual implementation of the FIFO design.

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

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 match the VHDL model, Verilog model, and core.
- The deassertion of the status flags (full, almost full, programmable full, empty, almost empty, programmable empty) will match the VHDL model, Verilog model, and core.

Note: If independent clocks or common clocks with built-in FIFO is selected, the user must use the structural model, as the behavioral model does not support the built-in FIFO configurations.

Structural Models

The structural models are designed to provide a more accurate model of FIFO behavior at the cost of simulation time. These models will provide a closer approximation of cycle accuracy across clock domains for asynchronous FIFOs. No asynchronous FIFO model can be 100% cycle accurate as physical relationships between the clock domains, including temperature, process, and frequency relationships, affect the domain crossing indeterminately.

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

Migrating to the Latest Version

This chapter provides step-by-step instructions for migrating existing designs containing instances of older versions of FIFO Generator Cores to the latest version of the FIFO Generator.

Note: For all new designs, it is recommended to use the most recent version of the FIFO Generator core.

Migrating Older Versions to the Most Recent Version

The FIFO Generator Migration Kit uses a perl script to automate the migration process from older versions of the FIFO Generator core (from v1.0 to v7.2) to the newest version of the FIFO Generator core (v8.1).

Use the `fifo_migrate.pl` script shipped with the FIFO Migration Kit zip file (`xapp992.zip`) to convert older versions of FIFO Generator core to the latest version of the FIFO Generator core.

Differences between Cores

This section defines the feature differences between the older versions of FIFO Generator core and the latest version of FIFO Generator. Before migrating existing designs, evaluate the differences because they may affect the behavior of your current design. In some cases, designs may need to be adjusted for obsolete features.

Differences in port names and XCO parameters from previous FIFO Generator cores are defined in Convert the XCO File, page 158 and Modifying the Instantiations of the Old Core, page 160.

Migrating a Design

The Migration Kit provides a Perl script to help automate the process of converting the previous version of the FIFO Generator core to the latest FIFO Generator core version. The migration script automates all the steps, from converting the XCO file to modifying the instantiation of the old core. In addition, this script can be used to isolate and automate specific steps, making it also useful when following the Manual Migration Process, page 157.

Migration Script

Note: The ISE v13.1 software requires a CGP file when CORE Generator is run in command line mode. The migration script comes with a sample CGP file (`coregen.cgp`) which the user can modify according to their requirements. The modified CGP file should be kept in the user directory where the XCO file and instantiation files are located and must be named `coregen.cgp` to work with the migration script.

If you can provide a complete set of original design files (XCO files and instantiation template files), the migration script (`fifo_migrate.pl`) completely and seamlessly automates the migration process by executing the following steps:

1. Converts old XCO files to new format XCO files.
2. Converts instantiations of old cores to new core instantiations including changing port names.
3. Generates new netlist(s) by calling the CORE Generator software with the new XCO file.

The migration script, `fifo_migrate.pl`, can operate on various inputs and create a variety of outputs based on user-specified command line options.

When using the script as part of the standard, fully-automated flow, supply the script with either of these two file types or both:

- Old XCO core configuration files (created by the GUI when the FIFO core was generated).
- HDL source file(s) containing the core instantiations (VHDL or Verilog).

From the script options, choose one or more of the following migration steps. All selected steps are automatically performed by the script.

- Old FIFO XCO files to FIFO Generator v8.1 XCO files (use -x option).
- Generate the new netlists and convert the instantiations of the old FIFO cores in the HDL source code to latest FIFO Generator core instances by running the CORE Generator software (-x and -m options).

The script modifies and overwrites all input files so that the external project files and scripts do not need to be updated with new file names or locations. Although the script also automatically generates a backup of all files it modifies, it is strongly recommended that you create a backup of all project files before running the migration script.

Output Products

Depending on the chosen command line option, the script overwrites the input XCO files, modifies the input HDL files, and optionally generates FIFO Generator netlists (in the same location as the XCO files).

The script creates a `./fifo_migrate_bak_filename[xco|v|vhd]` backup directory in which a copy of all files modified by the migration process are placed. It also generates a restore script in this directory, `restore_files.pl`, to restore the original files if necessary.

Using the Migration Script

To start the migration script, type the commands specific to your environment at the command prompt.

Linux

```
<path to script>/fifo_migrate.pl -x -m <HDL file(s)> <xco file>
```

Windows

```
xilperl <path to script>\fifo_migrate.pl -x -m <HDL file(s)> <xco file>
```

You must use at least one of the following options in the command string:

- -x. Creates XCO output files needed by the CORE Generator software to generate the core. Requires an input of the older version XCO file.
- -m. Calls the CORE Generator software to generate FIFO Generator netlists necessary to synthesize the design. This option must be used in tandem with the -x option. It modifies the HDL source files containing the core instantiations, converting the older instantiations and adding compatibility code. Requires either XCO or HDL file or both.

While using -x along with -m, the user can input only one XCO file, but one or more HDL files(s) that correspond(s) to the input XCO file. The modification of the HDL file will be according to the input XCO file. In addition, the user has to input the respective XCO file for modification of HDL files containing legacy version of instantiation(s).

`<xco file(s)>` is a list of one or more core configuration files corresponding to old FIFO cores which are to be converted to latest FIFO Generator core. These files can be referenced from directories other than the working directory.

To reverse the changes made by the script, go to the backup directory at `./fifo_migrate_bak_filename[xco|v|vhd]` and then run perl script `restore_files.pl`.

Examples

```
fifo_migrate.pl -x -m my_design.v my_core.xco
```

1. Creates a FIFO Generator version of `my_core.xco`.
2. Modifies the instantiations of `my_core in my_design.v`.
3. Runs CORE Generator software to generate the FIFO Generator version of `my_core.ngc` netlist file.

```
fifo_migrate.pl -x my_mem_core.xco
```

1. Creates a FIFO Generator version of `my_mem_core.xco`.
2. Script prompts the user to input a valid FIFO Generator version on execution.

Manual Migration Process

This section provides the instructions for the manual migration of an existing design to a FIFO Generator core. A summary of the required steps are provided below, followed by specific step-by-step instructions.

1. Convert the XCO File.

   The XCO file is used by the CORE Generator to determine a core's configuration.

   Note: If you plan to generate a new FIFO Generator core via the CORE Generator GUI, skip this step.
2. Generate the FIFO Generator Core.
3. Parameterizing the FIFO Generator GUI.
   Modify the instantiations of the old core. As the final step in the migration, you must update all instantiations of the old cores in your HDL source code to reference the new core. This includes changing the port names, as explained in Modifying the Instantiations of the Old Core, page 160. Differences between Cores, page 155 discusses whether design modifications are needed to compensate for obsolete features.

Convert the XCO File

Table 8-1 defines the mapping between previous FIFO Generator cores and the latest FIFO Generator cores.

In addition to changes in the parameter section of the XCO file, the core name specified in the XCO must be changed. Update the core name and version from the old core to the new core.

For the previous FIFO Generator core, change the line:

```
    SELECT FIFO_Generator family Xilinx,_Inc. x.y
```

to:

```
    SELECT FIFO_Generator family Xilinx,_Inc. 8.1
```

Table 8-1: **XCO Parameter Mapping: Previous FIFO Generator Cores**

Previous FIFO Cores[a]	FIFO XCO Parameter	Description of Conversion
component_name	component_name	No change required.
memory_type	fifo_implementation	
-	reset_type	Add this parameter and set the value to synchronous_reset.
-	full_flags_reset_value	1 (for backward compatibility).
-	use_dout_reset	True (for backward compatibility).
-	use_dout_reset True (for backward compatibility)	-
-	disable_timing_violation	-
-	enable_ecc	-
-	enable_int_clk	-
-	performance_options	-
-	read_clock_requency	-
-	reset_pin	-
-	use_embedded_register	-
-	use_extra_logic	-
-	use_dout_reset	-
-	write_clock_frequency	-
-	programmable_empty_type	-
-	programmable_full_type	-
-	inject_sbit_error	-
-	inject_dbit_error	-

a. Previous FIFO cores are FIFO Generator core versions from v1.0 to v3.3.

Generate the FIFO Generator Core

Generate a new FIFO Generator netlist. Note that Xilinx ISE must be installed on your system. The newly generated netlist (NGC) file replaces the old netlist file.

There are two ways to generate a new FIFO Generator netlist: using the CORE Generator GUI, or by executing an updated XCO file.

Parameterizing the FIFO Generator GUI

With an existing project open, or after creating a new project, the FIFO Generator core is available through the CORE Generator GUI.

To open the FIFO Generator core, do the following:

1. Click **View by Function** (active by default), and then open **Memories & Storage Elements** > **FIFOs**.
2. Double-click **FIFO Generator** to display the main GUI screen

Table 8-2 compares GUI parameters between the old and new versions of the FIFO Generator core. These tables help you choose the appropriate options when creating a new core using the FIFO Generator GUI.

Table 8-2: **GUI Parameter Comparison: Previous FIFO Generator**

Previous FIFO Generator[a]	FIFO Generator	Functionality of GUI Parameter
-	Built-in FIFO Options • Read Clock Frequency • Write Clock Frequency	Set Read, Write Clock frequencies. This option is only available for built-in FIFOs.
-	Implementation Options • Enable ECC • Use Embedded Registers in BRAM or FIFO (when possible)	Enable Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA-specific features. These options are only available for block RAM and built-in based FIFOs.
-	Initialization • Reset Pin • Asynchronous Reset • Synchronous Reset • Full Flags Reset Value • Use Dout Reset • Enable Reset Synchronization	Full Flags Reset Value determines the value of full flags (FULL, ALMOST_FULL, PROG_FULL) during asynchronous reset. Set the value to '1' for backward compatibility. Use Dout Reset to determine if the DOUT output resets to a specified value when the reset signal is asserted. Set to true for backward compatibility. Enable Reset Synchronization determines the use of internal reset synchronization logic. Set to true for backward compatibility.

Table 8-2: **GUI Parameter Comparison: Previous FIFO Generator *(Cont'd)***

Previous FIFO Generator[a]	FIFO Generator	Functionality of GUI Parameter
-	Programmable Flags • Programmable Full Type • full_threshold_assert_value • full_threshold_negate_value • Programmable Empty Type • empty_threshold_assert_value • empty_threshold_negate_value	Programmable Full Type provides the following options: • 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 Full threshold assert value is used to set the upper threshold value for the programmable full flag. Full Threshold Negate is used to set the lower threshold value for the programmable full flag. Programmable Empty Type provides the following options: • No_Programmable_Empty_Threshold • Single_Programmable_Empty_Threshold_Constant • Multiple_Programmable_Empty_Threshold_Constants • Single_Programmable_Empty_Threshold_Input_Port • Multiple_Programmable_Empty_Threshold_Input_Port Empty Threshold Assert is used to set the lower threshold value for the programmable empty flag. Empty Threshold Negate is used to set the upper threshold value for the programmable empty flag.
-	Error Injection • Single Bit Error Injection • Double Bit Error Injection	Enable Kintex-7, Virtex-7, and Virtex-6 FPGA-specific features. These options are only available for block RAM and built-in FIFOs.

a. Previous FIFO Cores refer the FIFO Generator core versions from v1.0 to v3.3.

Modifying the Instantiations of the Old Core

For each FIFO Generator core instantiation, do the following:

1. Change the name of the module. (Only necessary if component name of the core has changed.)
2. Change the port names. For port name conversions, see Table 8-3.

Table 8-3: **Port Name Mapping: Previous FIFO Generator Cores**

Previous FIFO Generator Core[a]		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
DIN[N:0]	Available	DIN[N:0]	Available	Same	Available
WR_EN	Available	WR_EN	Available	Same	Available
WR_CLK	Available	WR_CLK	Available	Same	Available
RD_EN	Available	RD_EN	Available	Same	Available
RD_CLK	Available	RD_CLK	Available	Same	Available

Table 8-3: **Port Name Mapping: Previous FIFO Generator Cores *(Cont'd)***

Previous FIFO Generator Core[a]		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
FULL	Available	FULL	Available	Same	Available
ALMOST_FULL	Optional	ALMOST_FULL	Available	Same	Available
-	-	PROG_FULL	Optional	-	Optional
-	-	PROG_FULL_THRESH	Optional	-	Optional
-	-	PROG_FULL_THRESH_ASSERT	Optional	-	Optional
-	-	PROG_FULL_THRESH_NEGATE	Optional	-	Optional
WR_COUNT[W:0]	Optional	WR_DATA_COUNT[D:0]	Optional	Direct Replacement	Optional
WR_ACK	Optional	WR_ACK		Same	Optional
WR_ERR	Optional	OVERFLOW	Optional	Direct Replacement	Optional
DOUT[N:0]	Available	DOUT[M:0]	Available	Same	Optional
EMPTY	Available	EMPTY	Available	Same	Optional
ALMOST_EMPTY	Optional	ALMOST_EMPTY	Optional	Same	Optional
-	-	PROG_EMPTY	Optional	-	Optional
-	-	PROG_EMPTY_THRESH	Optional	-	Optional
-	-	PROG_EMPTY_THRESH_ASSERT	Optional	-	Optional
-	-	PROG_EMPTY_THRESH_NEGATE	Optional	-	Optional
RD_COUNT[R:0]	Optional	RD_DATA_COUNT[C:0]	Optional	Direct Replacement	Optional
VALID	Optional	VALID	Optional	Direct Replacement	Optional
Underflow	Optional	UNDERFLOW	Optional	Direct Replacement	Optional
-	-	SBITERR	Optional	-	Optional
-	-	DBITERR	Optional	-	Optional
-	-	INJECTSBITERR	Optional	-	Optional
-	-	INJECTDBITERR	Optional	-	Optional

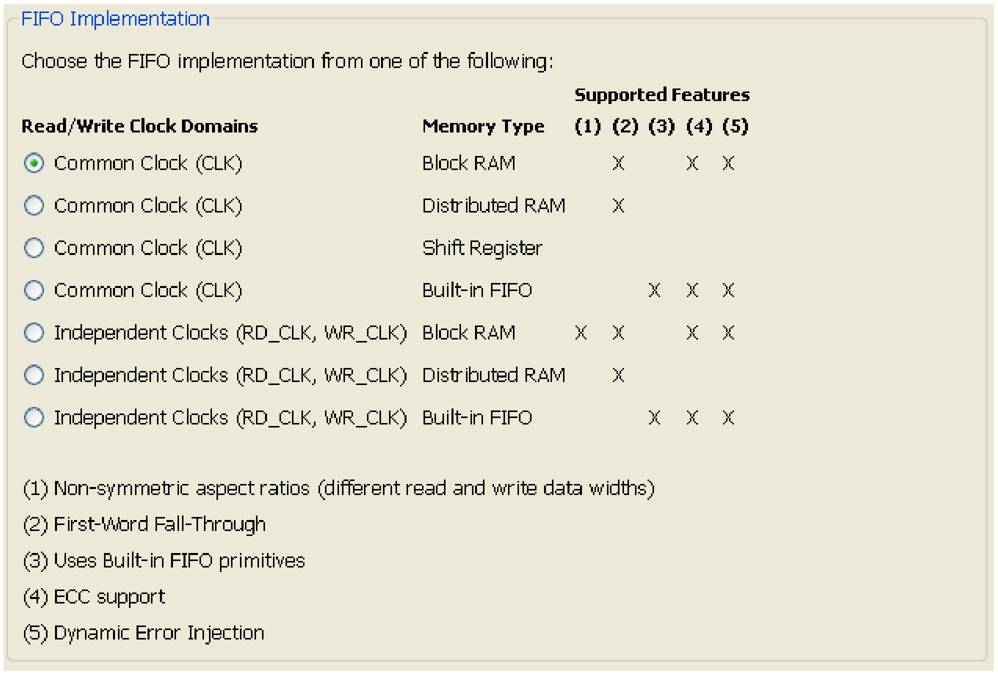
a. Previous FIFO Cores refer the FIFO Generator core versions from v1.0 to v3.3.

Converting Native Interface FIFOs to AXI4 Interface FIFOs

This section explains how an existing Native Interface FIFO configuration (prior to v7.2) can be achieved using the AXI4-Stream FIFO solution.

Component Name and FIFO Implementation Selection

Figure 8-1 shows the Native Interface FIFO Page 1 screen to set the component name and FIFO Implementation types.



FIFO Implementation

Choose the FIFO implementation from one of the following:

		Supported Features				
Read/Write Clock Domains	Memory Type	(1)	(2)	(3)	(4)	(5)
[✓ Common Clock (CLK)]	Block RAM		X		X	X
Common Clock (CLK)	Distributed RAM		X			
Common Clock (CLK)	Shift Register					
Common Clock (CLK)	Built-in FIFO			X	X	X
Independent Clocks (RD_CLK, WR_CLK)	Block RAM	X	X		X	X
Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		X			
Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO			X	X	X

(1) Non-symmetric aspect ratios (different read and write data widths)

(2) First-Word Fall-Through

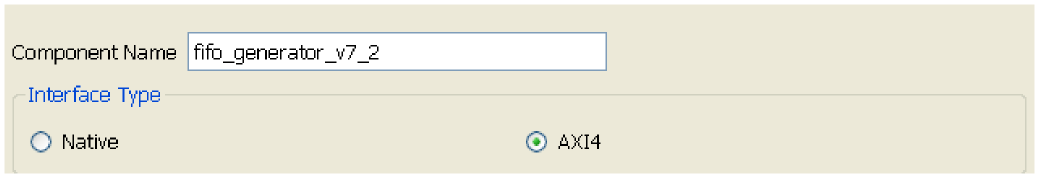
(3) Uses Built-in FIFO primitives

(4) ECC support

(5) Dynamic Error Injection

Figure 8-1: **Native Interface FIFO: Page 1 - Component Name and FIFO Implementation Selection**

Figure 8-2, Figure 8-3 and Figure 8-4 show the equivalent settings of Figure 8-1 in the AXI4-Stream GUI.

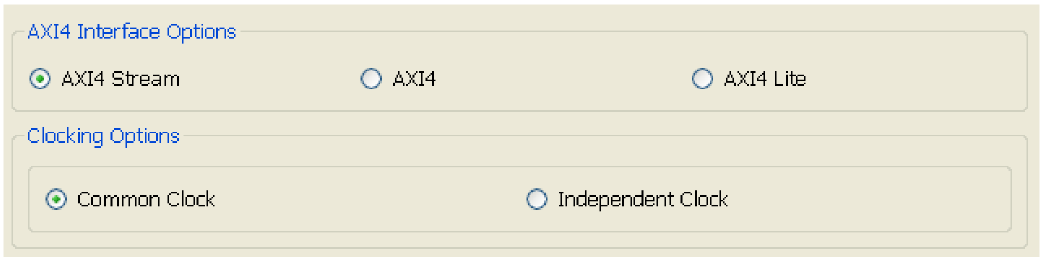


Component Name: fifo_generator_v7_2

Interface Type

Native [✓ AXI4]

Figure 8-2: **AXI4 Interface FIFO: Page 1 - Component Name and Interface Type Selection**

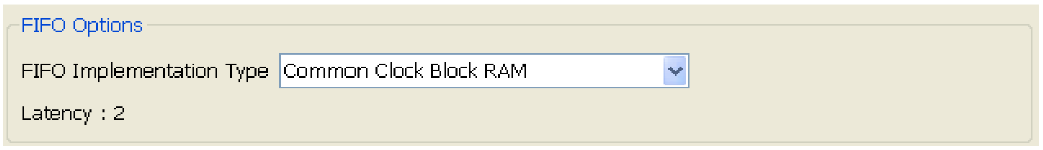

AXI4 Interface Options

[✓ AXI4 Stream] AXI4 AXI4 Lite

Clocking Options

[✓ Common Clock] Independent Clock

Figure 8-3: **AXI4 Interface FIFO: Page 2 - Interface and Clocking Options**


FIFO Options

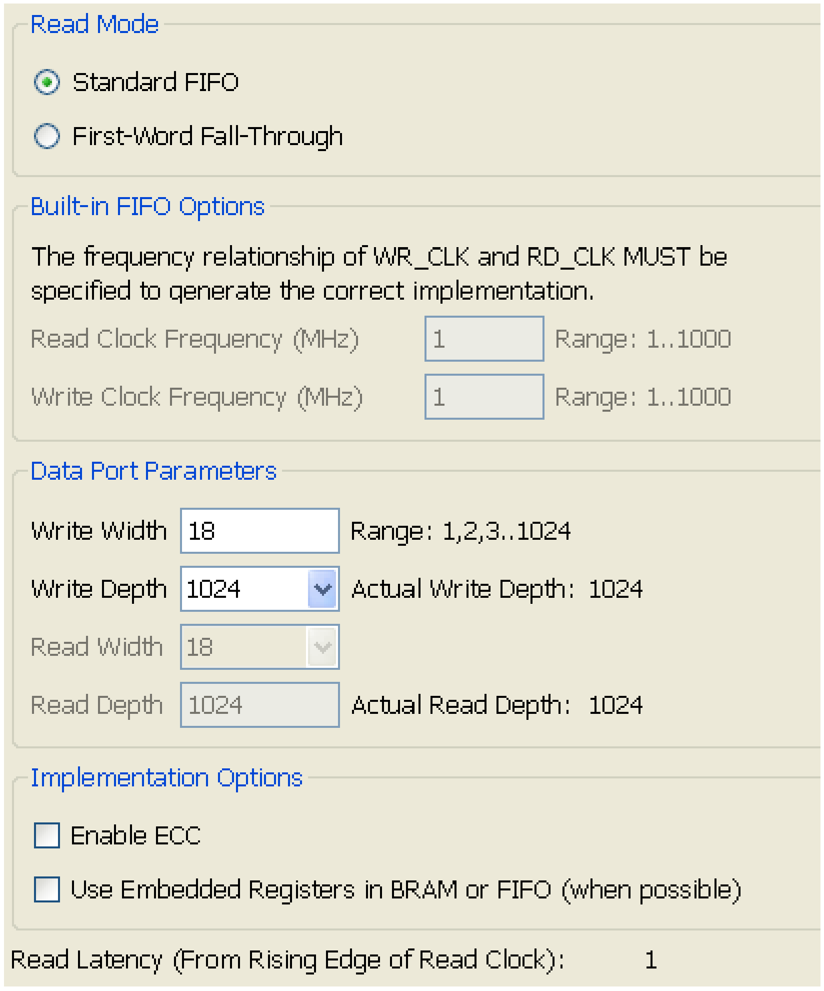
FIFO Implementation Type: Common Clock Block RAM

Latency : 2

Figure 8-4: **AXI4 Interface FIFO: Page 4 - FIFO Options**

Read Mode and Data Port Parameters Selection

Figure 8-5 shows the Native Interface FIFO Page 2 screen to set the read mode, built-in FIFO options, data port parameters, and implementation options.


Read Mode

(● Standard FIFO) ○ First-Word Fall-Through

Built-in FIFO Options

The frequency relationship of WR_CLK and RD_CLK MUST be specified to generate the correct implementation.

Read Clock Frequency (MHz)	1	Range: 1..1000
Write Clock Frequency (MHz)	1	Range: 1..1000

Data Port Parameters

Write Width	18	Range: 1,2,3..1024
Write Depth	1024	Actual Write Depth: 1024
Read Width	18	
Read Depth	1024	Actual Read Depth: 1024

Implementation Options

☐ Enable ECC

☐ Use Embedded Registers in BRAM or FIFO (when possible)

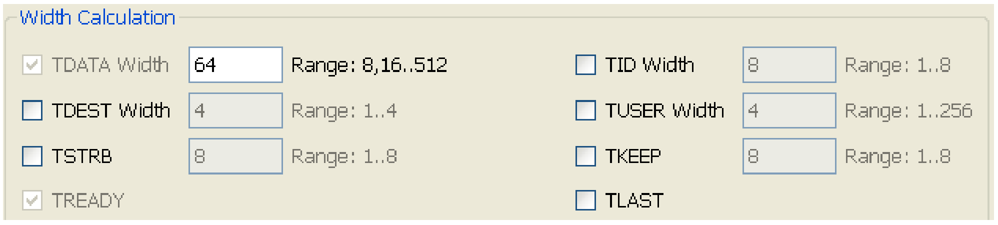
Read Latency (From Rising Edge of Read Clock): 1

Figure 8-5: **Native Interface FIFO: Page 2 - Read Mode, Built-In FIFO, Data Port Parameters and Implementation Options**

Figure 8-6, Figure 8-7 and Figure 8-8 show the equivalent settings of Figure 8-5 in the AXI4-Stream GUI.

When migrating to the AXI4 interface, note the following settings:

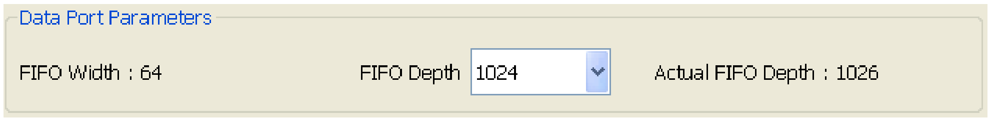
- Read Mode is always set to First-Word-Fall-Through in AXI4-Stream FIFOs.
- Built-in FIFO is not supported in AXI4-Stream FIFOs.
- Write Width of Native FIFOs can be set through TDATA Width on Page 3 of the AXI4-Stream FIFO GUI, as shown in Figure 8-6. If the Write Width is > 512, then TUSER Width can be used.


Width Calculation

✓ TDATA Width	64	Range: 8,16..512	☐ TID Width	8	Range: 1..8
☐ TDEST Width	4	Range: 1..4	☐ TUSER Width	4	Range: 1..256
☐ TSTRB	8	Range: 1..8	☐ TKEEP	8	Range: 1..8
✓ TREADY			☐ TLAST		

Figure 8-6: **AXI4 Interface FIFO: Page 3 - Write Width Calculation**

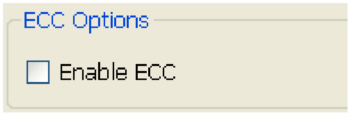
- Write Depth of Native FIFO can be set through FIFO Depth on Page 4 of the AXI4-Stream FIFO GUI, as shown in Figure 8-7.


Data Port Parameters

FIFO Width : 64　FIFO Depth 1024　Actual FIFO Depth : 1026

Figure 8-7: **AXI4 Interface FIFO: Page 4 - FIFO Depth Selection**

- The ECC can be enabled through Enable ECC in Page 4 of the AXI4-Stream FIFO GUI, as shown in Figure 8-8.


ECC Options

☐ Enable ECC

Figure 8-8: **AXI4 Interface FIFO: Page 4 - ECC Selection**

- Use Embedded Register Option is not supported in AXI4-Stream FIFOs.

Optional Flags and Error Injection Selection

Figure 8-9 shows the Native Interface FIFO Page 3 screen to set the optional flags, handshaking and error injection options.

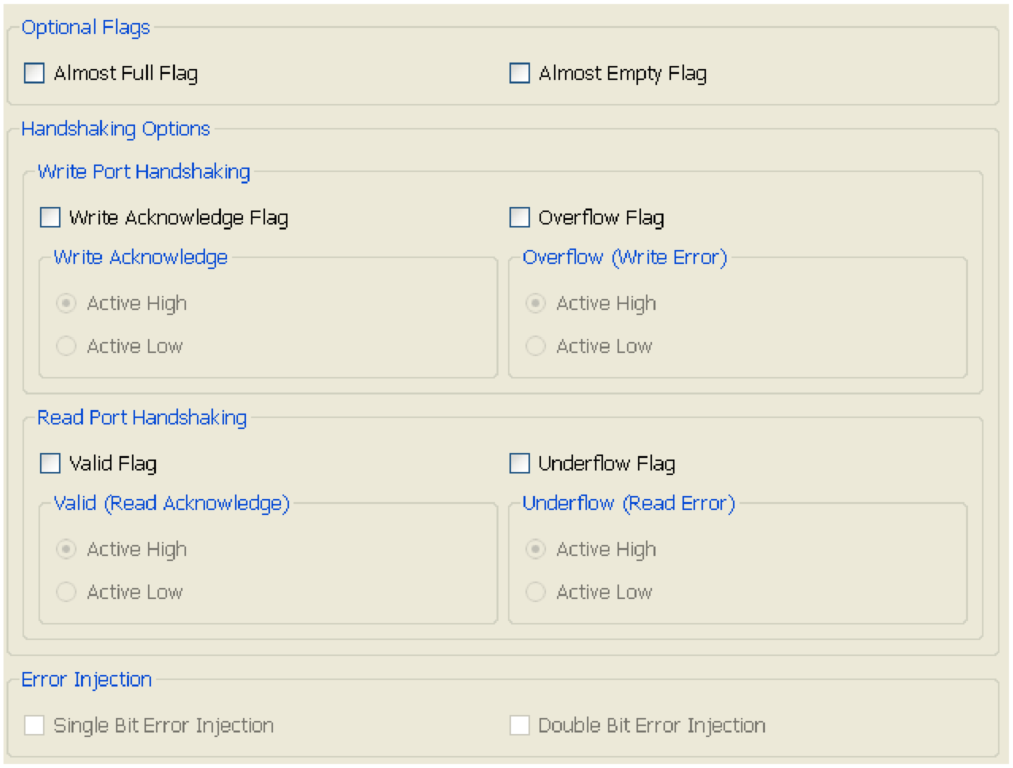

Optional Flags
☐ Almost Full Flag ☐ Almost Empty Flag
Handshaking Options
Write Port Handshaking
☐ Write Acknowledge Flag ☐ Overflow Flag
Write Acknowledge: [✓ Active High] Active Low
Overflow (Write Error): [✓ Active High] Active Low
Read Port Handshaking
☐ Valid Flag ☐ Underflow Flag
Valid (Read Acknowledge): [✓ Active High] Active Low
Underflow (Read Error): [✓ Active High] Active Low
Error Injection
☐ Single Bit Error Injection ☐ Double Bit Error Injection

Figure 8-9: **Native Interface FIFO: Page 3 - Optional Flag, Handshaking and Error Injection Options**

Equivalent selections in AXI4-Stream FIFO GUI are as follows:

- Optional Flags (Almost Full and Almost Empty) are mapped to TVALID and TREADY depending on the Enable Handshake Flag Options and Handshake Flag Options selection on Page 4 of the AXI4-Stream FIFO GUI as shown in Figure 8-10 and Figure 8-11.

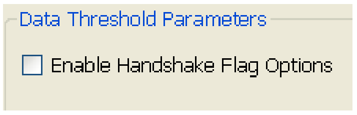

Data Threshold Parameters
☐ Enable Handshake Flag Options

Figure 8-10: **AXI4-Stream Interface FIFO: Page 4 - Data Threshold Parameters**

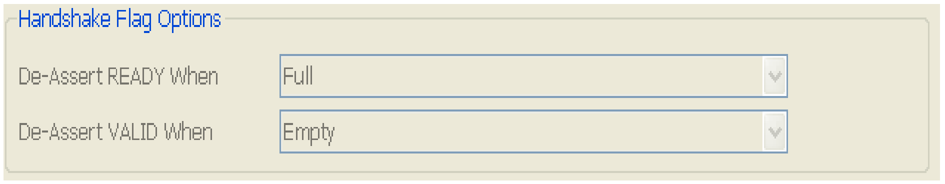

Handshake Flag Options
De-Assert READY When: Full
De-Assert VALID When: Empty

Figure 8-11: **AXI4-Stream Interface FIFO: Page 4 - Handshake Flag Options**

- The Write Acknowledge Flag and Valid Flag are not supported in AXI4-Stream FIFOs.

- Overflow Flag and Underflow Flag options of Native Interface FIFOs can be set through similar settings on Page 5 of the AXI4-Stream FIFO GUI, shown in Figure 8-12.

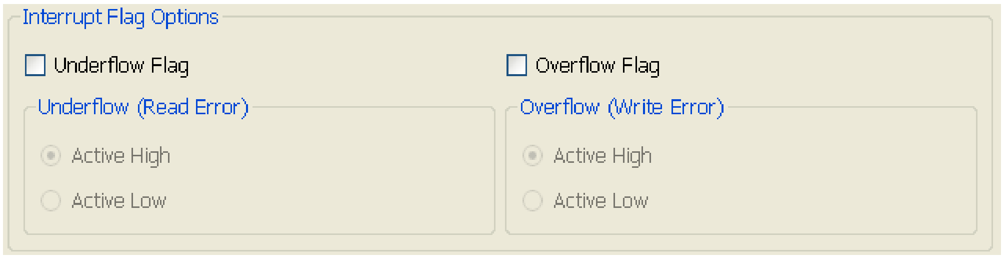

Interrupt Flag Options
☐ Underflow Flag
Underflow (Read Error)
[✓ Active High] Active Low
☐ Overflow Flag
Overflow (Write Error)
[✓ Active High] Active Low

Figure 8-12: **AXI4-Stream Interface FIFO: Page 5 - Interrupt Flag Options**

- Single Bit Error Injection and Double Bit Error Injection options of Native Interface FIFOs can be set using similar settings on Page 4 of the AXI4-Stream FIFO GUI, as shown in Figure 8-13.

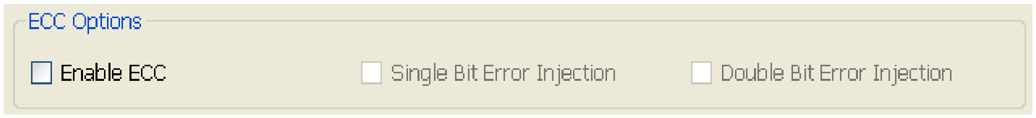
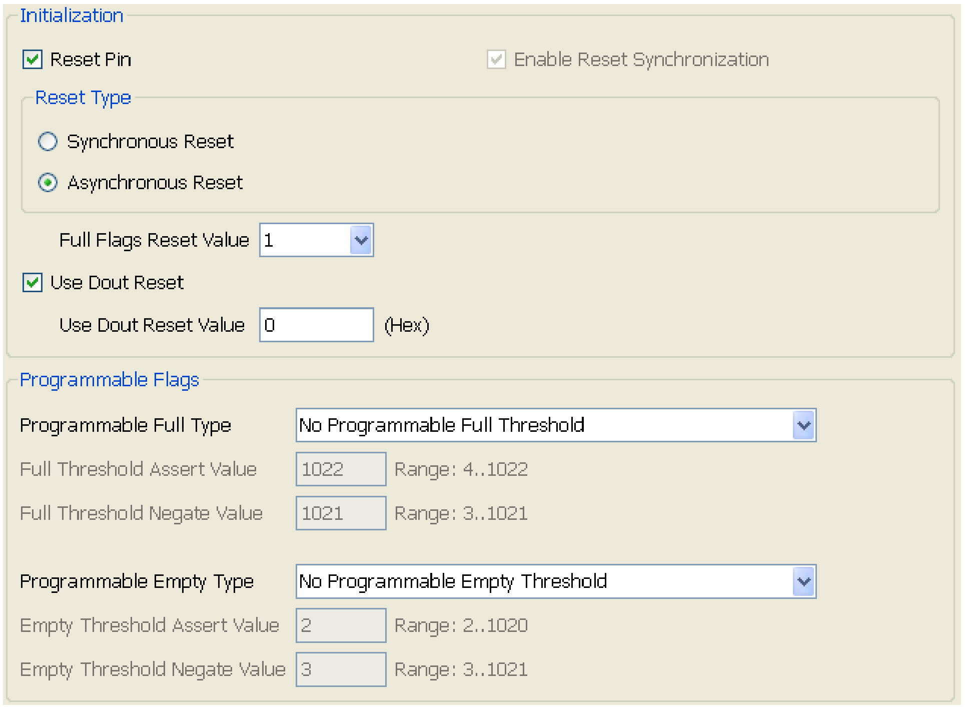

ECC Options
☐ Enable ECC ☐ Single Bit Error Injection ☐ Double Bit Error Injection

Figure 8-13: **AXI4-Stream Interface FIFO: Page 4 - Error Injection Options**

Initialization and Programmable Options Selection

Figure 8-14 shows the Native Interface FIFO Page 4 screen to set the initialization and programmable flags.

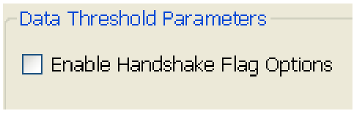

Initialization
✓ Reset Pin ✓ Enable Reset Synchronization
Reset Type
Synchronous Reset [✓ Asynchronous Reset]
Full Flags Reset Value: 1
✓ Use Dout Reset
Use Dout Reset Value: 0 (Hex)

Programmable Flags
Programmable Full Type: No Programmable Full Threshold
Full Threshold Assert Value: 1022 Range: 4..1022
Full Threshold Negate Value: 1021 Range: 3..1021
Programmable Empty Type: No Programmable Empty Threshold
Empty Threshold Assert Value: 2 Range: 2..1020
Empty Threshold Negate Value: 3 Range: 3..1021

Figure 8-14: **Native Interface FIFO: Page 4 - Initialization and Programmable Options**

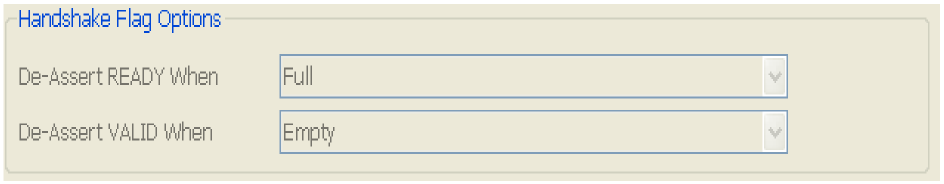
Equivalent selection in the AXI4-Stream FIFO GUI are as follows:

- All the options in the Initialization group box of Native Interface FIFO GUI are not available in AXI4-Stream FIFOs. However, Asynchronous Reset is by default enabled in AXI4-Stream FIFOs.
- The programmable flags (Programmable Full and Programmable Empty) are mapped to TVALID and TREADY depending on the Enable Handshake Flag Options and Handshake Flag Options selection on Page 4 of the AXI4-Stream FIFO GUI, as shown in Figure 8-15 and Figure 8-16.


Data Threshold Parameters

☐ Enable Handshake Flag Options

Figure 8-15: **AXI4-Stream Interface FIFO: Page 4 - Data Threshold Parameters**

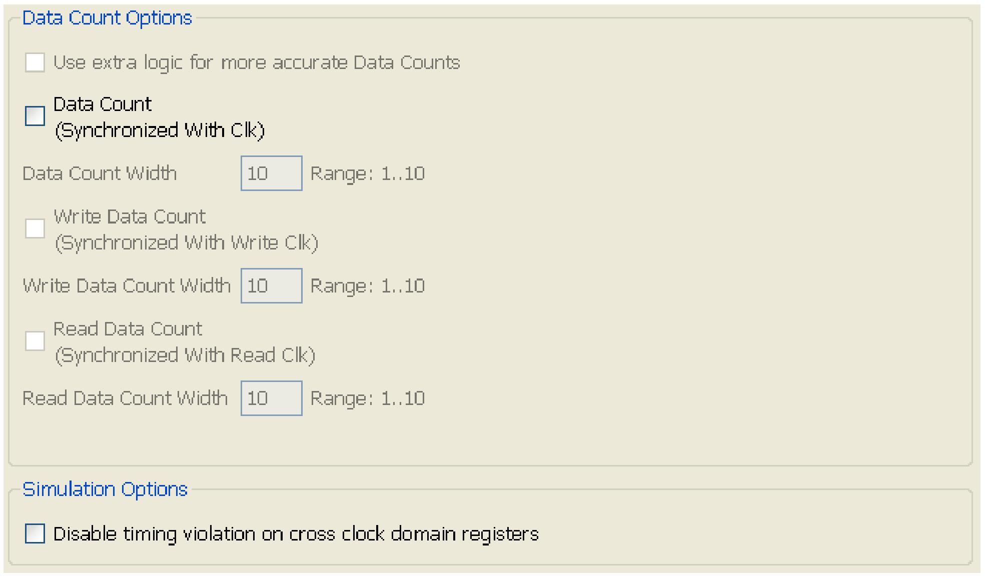

Handshake Flag Options

De-Assert READY When: Full

De-Assert VALID When: Empty

Figure 8-16: **AXI4-Stream Interface FIFO: Page 4 - Programmable Flag Options**

Figure 8-17 shows the Native Interface FIFO Page 5 screen to set the data count and simulation options.


Data Count Options

☐ Use extra logic for more accurate Data Counts

☐ Data Count
(Synchronized With Clk)

Data Count Width: 10 Range: 1..10

☐ Write Data Count
(Synchronized With Write Clk)

Write Data Count Width: 10 Range: 1..10

☐ Read Data Count
(Synchronized With Read Clk)

Read Data Count Width: 10 Range: 1..10

Simulation Options

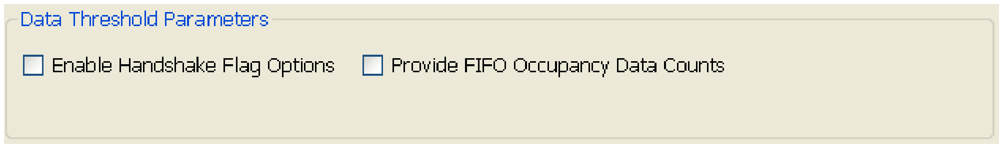
☐ Disable timing violation on cross clock domain registers

Figure 8-17: **Native Interface FIFO: Page 5 - Data Count and Simulation Options**

Equivalent selections in the AXI4-Stream FIFO GUI are as follows:

- The Use Extra Logic option is always set to true for AXI4-Stream FIFOs.
- Data Count/Write Data Count/Read Data Count options of Native Interface FIFOs can be set through the Provide FIFO Occupancy Data Counts option on Page 4 of the AXI4-Stream FIFO GUI, as shown in Figure 8-18.

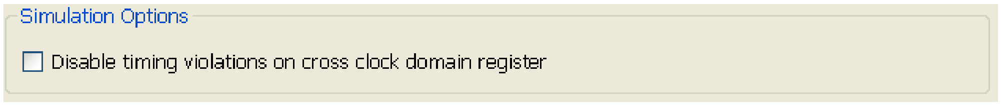
- Depending on the Clocking option set on Page 2 of the AXI4-Stream FIFO GUI, either Data Count or Write/Read Data Count will be enabled.


Data Threshold Parameters

☐ Enable Handshake Flag Options ☐ Provide FIFO Occupancy Data Counts

Figure 8-18: **AXI4-Stream Interface FIFO: Page 4 - FIFO Occupancy Option**

- Simulation Options can be set using the a similar option on Page 5 of the AXI4-Stream FIFO GUI, as shown in Figure 8-19.


Simulation Options

☐ Disable timing violations on cross clock domain register

Figure 8-19: **AXI4-Stream Interface FIFO: Page 5 - Simulation Options**

Native FIFO to AXI4-Stream FIFO XCO Parameter Map

Table 8-4 shows the correlating parameters for the Native Interface FIFOs and the AXI4-Stream FIFOs.

Table 8-4: **Native FIFO to AXI4-Stream FIFO**

	Native Interface FIFO XCO Parameter	Equivalent AXI4-Stream FIFO XCO Parameter	XCO Value
1	N/A	interface_type	AXI4
2	N/A	axi_type	AXI4_Stream
3	almost_empty_flag	programmable_empty_type_axis	Almost_Empty
4	almost_full_flag	programmable_full_type_axis	Almost_Full
5	data_count	enable_data_counts_axis	True, False
		clock_type_axi	Common_Clock
6	disable_timing_violations	disable_timing_violations_axi	True, False
7	empty_threshold_assert_value	empty_threshold_assert_ value_axis	4 - 4194302
8	enable_ecc	enable_ecc_axis	True, False
9	fifo_implementation	fifo_implementation_axis	Common_Clock_Block_RAM Common_Clock_Distributed_RAM Independent_Clocks_Block_RAM Independent_Clocks_Distributed_RAM
		fifo_application_type_axis	Data_FIFO
		axis_type	FIFO
10	full_threshold_assert_value	full_threshold_assert_value_axis	6 - 4194303
11	inject_dbit_error	inject_dbit_error_axis	True, False
12	inject_sbit_error	inject_sbit_error_axis	True, False

Table 8-4: **Native FIFO to AXI4-Stream FIFO *(Cont'd)***

	Native Interface FIFO XCO Parameter	Equivalent AXI4-Stream FIFO XCO Parameter	XCO Value
13	input_data_width	enable_tdata	True, False
		enable_tdest	True, False
		enable_tid	True, False
		enable_tkeep	True, False
		enable_tlast	True, False
		enable_tready	True, False
		enable_tstrobe	True, False
		enable_tuser	True, False
		tdata_width	2^3 - 2^9
		tdest_width	1 - 4
		tid_width	1 - 8
		tkeep_width	tdata_width/8
		tstrb_width	tdata_width/8
		tuser_width	1 - 256
14	input_depth	input_depth_axis	2^4 - 2^{16}
15	overflow_flag	overflow_flag_axi	True, False
16	overflow_sense	overflow_sense_axi	Active_High, Active_Low
17	programmable_empty_type	programmable_empty_type_axis	Empty Almost_Empty Single_Programmable_Empty_ Threshold_Constant Single_Programmable_Empty_ Threshold_Input_Port
18	programmable_full_type	programmable_full_type_axis	Full, Almost_Full, Single_Programmable_Full_ Threshold_Constant, Single_Programmable_Full_ Threshold_Input_Port
19	read_data_count	enable_data_counts_axis	True, False
		clock_type_axi	Independent_Clock
20	underflow_flag	underflow_flag_axi	True, False
21	underflow_sense	underflow_sense_axi	Active_High, Active_Low
22	write_data_count	enable_data_counts_axis	True, False
		clock_type_axi	Independent_Clock
23	data_count_width	N/A	N/A

Table 8-4: **Native FIFO to AXI4-Stream FIFO *(Cont'd)***

	Native Interface FIFO XCO Parameter	Equivalent AXI4-Stream FIFO XCO Parameter	XCO Value
24	dout_reset_value	N/A	N/A
25	empty_threshold_negate_value	N/A	N/A
26	enable_reset_synchronization	N/A	N/A
27	full_flags_reset_value	N/A	N/A
28	full_threshold_negate_value	N/A	N/A
29	output_data_width	N/A	N/A
30	output_depth	N/A	N/A
31	performance_options	N/A	N/A
32	read_clock_frequency	N/A	N/A
33	read_data_count_width	N/A	N/A
34	reset_pin	N/A	N/A
35	reset_type	N/A	N/A
36	use_dout_reset	N/A	N/A
37	use_embedded_registers	N/A	N/A
38	use_extra_logic	N/A	N/A
39	valid_flag	N/A	N/A
40	valid_sense	N/A	N/A
41	write_acknowledge_flag	N/A	N/A
42	write_acknowledge_sense	N/A	N/A
43	write_clock_frequency	N/A	N/A
44	write_data_count_width	N/A	N/A

Appendix A

Performance Information

Resource Utilization and Performance

Performance and resource utilization for a FIFO varies depending on the configuration and features selected during core customization. The following tables show resource utilization data and maximum performance values for a variety of sample FIFO configurations.

See “Resource Utilization and Performance” in *FIFO Generator Data Sheet* for the performance and resource utilization numbers.

Appendix B

Core Parameters

Native Interface FIFO XCO Parameters

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

Table B-1: **Native Interface FIFO XCO Parameter Table**

Native FIFO XCO Parameter Name	XCO File Values	Default GUI Settings
interface_type	Native, AXI4	Native
almost_empty_flag	True, False	False
almost_full_flag	True, False	False
component_name	instance_name ASCII text starting with a letter and using the following character set: a-z, 0-9, and _	fifo_generator_v8_1
data_count	True, False	False
data_count_width	1 – $\log_2$(output_depth)	10
disable_timing_violations	True, False	False
dout_reset_value	Hex value in range of 0 to output data width - 1	0
empty_threshold_assert_value	For STD: 2 - 4194300 For FWFT: 4 - 4194302	2
empty_threshold_negate_value	For STD: 3 - 4194301 For FWFT: 5 - 4194303	3
enable_ecc	True, False	False
enable_reset_synchronization	True, False	True
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
full_flags_reset_value	0, 1	1

Table B-1: **Native Interface FIFO XCO Parameter Table *(Cont'd)***

Native FIFO XCO Parameter Name	XCO File Values	Default GUI Settings
full_threshold_assert_value	For STD: 4 - 4194302 For FWFT: 6 - 4194303	1022
full_threshold_negate_value	For STD: 3 - 4194301 For FWFT: 5 - 4194302	1021
inject_dbit_error	True, False	False
inject_sbit_error	True, False	False
input_data_width	1 - 1024	18
input_depth	2^4 - 2^{22}	1024
output_data_width	1 - 1024	18
output_depth	2^4 - 2^{22}	1024
overflow_flag	True, False	False
overflow_sense	Active_High, Active_Low	Active_High
performance_options	Standard_FIFO (STD), First_Word_Fall_Through (FWFT)	Standard_FIFO
programmable_empty_type	No_Programmable_Empty_ Threshold Single_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
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
read_clock_frequency	1 - 1000	1
read_data_count	True, False	False
read_data_count_width	1 – $\log_2$(output_depth)	10
reset_pin	True, False	True
reset_type	Synchronous_Reset, Asynchronous_Reset	Asynchronous_Reset

Table B-1: **Native Interface FIFO XCO Parameter Table *(Cont'd)***

Native FIFO XCO Parameter Name	XCO File Values	Default GUI Settings
underflow_flag	True, False	False
underflow_sense	Active_High, Active_Low	Active_High
use_dout_reset	True, False	False
use_embedded_registers	True, False	False
use_extra_logic	True, False	False
valid_flag	True, False	False
valid_sense	Active_High, Active_Low	Active_High
write_acknowledge_flag	True, False	False
write_acknowledge_sense	Active_High, Active_Low	Active_High
write_clock_frequency	1 – 1000	1
write_data_count	True, False	False
write_data_count_width	1 – $\log_2$(input_depth)	10

AXI4 FIFO XCO Parameters

Table B-2 describes the AXI4 FIFO core parameters, including the XCO file value and the default settings.

Table B-2: **AXI4 FIFO XCO Parameter Table**

Parameter Name	XCO File Values	Default GUI Settings
component_name	instance_name ASCII text starting with a letter and using the following character set: a-z, 0-9, and _	fifo_generator_v8_1
interface_type	Native AXI4	Native
axi_type	AXI4_Stream AXI4_Full, AXI4_Lite	AXI4_Stream
enable_write_channel	True False	True
enable_read_channel	True False	True
clock_type_axi	Common_Clock Independent_Clock	Common_Clock
use_clock_enable[a]	True False	False

Table B-2: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
clock_enable_type[a]	Slave_Interface_Clock_Enable Master_Interface_Clock_Enable	Slave_Interface_Clock_Enable
id_width	1 - 8	8
axi_address_width	1 – 32	32
axi_data_width	2^3 - 2^9	64
enable_awuser	True False	False
enable_wuser	True False	False
enable_buser	True False	False
enable_aruser	True False	False
enable_ruser	True False	False
enable_tuser	True False	False
awuser_width	1 - 256	1
wuser_width	1 - 256	1
buser_width	1 - 256	1
aruser_width	1 - 256	1
ruser_width	1 - 256	1
tuser_width	1 - 256	4
enable_tdata	True False	True
enable_tdest	True False	False
enable_tid	True False	False
enable_tkeep	True False	False
enable_tlast	True False	False
enable_tready	True False	True

Table B-2: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
enable_tstrobe	True False	False
tdata_width	2^3 - 2^9	64
tdest_width	1 - 4	4
tid_width	1 - 8	8
tkeep_width	tdata_width/8	8
tstrb_width	tdata_width/8	8
axis_type	FIFO	FIFO
wach_type	FIFO	FIFO
wdch_type	FIFO	FIFO
wrch_type	FIFO	FIFO
rach_type	FIFO	FIFO
rdch_type	FIFO	FIFO
fifo_implementation_type_axis	Common_Clock_Block_RAM Common_Clock_Distributed_RAM Independent_Clock_Block_RAM Independent_Clock_Distributed_RAM	Common_Clock_Block_RAM
fifo_implementation_type_rach		Common_Clock_Distributed_RAM
fifo_implementation_type_rdch		Common_Clock_Block_RAM
fifo_implementation_type_wach		Common_Clock_Distributed_RAM
fifo_implementation_type_wdch		Common_Clock_Block_RAM
fifo_implementation_type_wrch		Common_Clock_Distributed_RAM
fifo_application_type_axis	Data_FIFO	Data_FIFO
fifo_application_type_rach	Data_FIFO	Data_FIFO
fifo_application_type_rdch	Data_FIFO	Data_FIFO
fifo_application_type_wach	Data_FIFO	Data_FIFO
fifo_application_type_wdch	Data_FIFO	Data_FIFO
fifo_application_type_wrch	Data_FIFO	Data_FIFO
enable_ecc_axis	True False	False
enable_ecc_rach	True False	False
enable_ecc_rdch	True False	False

Table B-2: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
enable_ecc_wach	True False	False
enable_ecc_wdch	True False	False
enable_ecc_wrch	True False	False
inject_sbit_error_axis	True False	False
inject_sbit_error_rach	True False	False
inject_sbit_error_rdch	True False	False
inject_sbit_error_wach	True False	False
inject_sbit_error_wdch	True False	False
inject_sbit_error_wrch	True False	False
inject_dbit_error_axis	True False	False
inject_dbit_error_rach	True False	False
inject_dbit_error_rdch	True False	False
inject_dbit_error_wach	True False	False
inject_dbit_error_wdch	True False	False
inject_dbit_error_wrch	True False	False
input_depth_axis	2^4 - 2^{16}	1024
input_depth_rach	2^4 - 2^{16}	16
input_depth_rdch	2^4 - 2^{16}	1024
input_depth_wach	2^4 - 2^{16}	16
input_depth_wdch	2^4 - 2^{16}	1024
input_depth_wrch	2^4 - 2^{16}	16

Table B-2: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
enable_data_counts_axis	True False	False
enable_data_counts_rach	True False	False
enable_data_counts_rdch	True False	False
enable_data_counts_wach	True False	False
enable_data_counts_wdch	True False	False
enable_data_counts_wrch	True False	False
enable_handshake_flag_options_axis	True False	False
enable_handshake_flag_options_rach	True False	False
enable_handshake_flag_options_rdch	True False	False
enable_handshake_flag_options_wach	True False	False
enable_handshake_flag_options_wdch	True False	False
enable_handshake_flag_options_wrch	True False	False
programmable_full_type_axis	Full Almost_Full Single_Programmable_Full_Threshold_Constant Single_Programmable_Full_Threshold_Input_Port	Full
programmable_full_type_rach		Full
programmable_full_type_rdch		Full
programmable_full_type_wach		Full
programmable_full_type_wdch		Full
programmable_full_type_wrch		Full
full_threshold_assert_value_axis	5 - 65535	1023
full_threshold_assert_value_rach	5 - 65535	1023

Table B-2: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
full_threshold_assert_value_rdch	5 - 65535	1023
full_threshold_assert_value_wach	5 - 65535	1023
full_threshold_assert_value_wdch	5 - 65535	1023
full_threshold_assert_value_wrch	full_threshold_assert_value_wrch	1023
programmable_empty_type_axis	Empty Almost_Empty Single_Programmable_ Empty _Threshold_Constant Single_Programmable_ Empty _Threshold_Input_Port	Empty
programmable_empty_type_rach		Empty
programmable_empty_type_rdch		Empty
programmable_empty_type_wach		Empty
programmable_empty_type_wdch		Empty
programmable_empty_type_wrch		Empty
empty_threshold_assert_value_axis	4 - 65534	1022
empty_threshold_assert_value_rach	4 - 65534	1022
empty_threshold_assert_value_rdch	4 - 65534	1022
empty_threshold_assert_value_wach	4 - 65534	1022
empty_threshold_assert_value_wdch	4 - 65534	1022
empty_threshold_assert_value_wrch	4 - 65534	1022
underflow_flag_axi	True False	False
underflow_sense_axi	Active_High Active_Low	Active_High
overflow_flag_axi	True False	False
overflow_sense_axi	Active_High Active_Low	Active_High
enable_common_overflow	True False	False
enable_common_underflow[a]	True False	False

Table B-2: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
disable_timing_violations_axi	True False	False
add_ngc_constraint_axi[a]	True False	False

a. Feature presently not supported

Comparison of Native and AXI4 FIFO XCO Parameters

Table B-3 describes the comparison of Native FIFO and AXI4 FIFO XCO parameters, including the possible values.

Table B-3: **Native FIFO and AXI4 FIFO XCO Parameter Comparison**

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
1	almost_empty_flag	almost_empty_flag	True, False
2	almost_full_flag	almost_full_flag	True, False
3	data_count	data_count	True, False
4	data_count_width	data_count_width	1 – $\log_2$(output_depth)
5	disable_timing_violations	disable_timing_violations	True, False
6	dout_reset_value	dout_reset_value	Any hexadecimal value of width 1 - 1024
7	empty_threshold_assert_value	empty_threshold_assert_value	For STD: 2 - 4194300 For FWFT: 4 - 4194302
8	empty_threshold_negate_value	empty_threshold_negate_value	For STD: 3 - 4194301 For FWFT: 5 - 4194303
9	enable_ecc	enable_ecc	True, False
10	enable_reset_synchronization	enable_reset_synchronization	True, False
11	fifo_implementation	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
12	full_flags_reset_value	full_flags_reset_value	0, 1
13	full_threshold_assert_value	full_threshold_assert_value	For STD: 4 - 4194302 For FWFT: 6 - 4194303
14	full_threshold_negate_value	full_threshold_negate_value	For STD: 3 - 4194301 For FWFT: 5 - 4194302

Table B-3: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
15	inject_dbit_error	inject_dbit_error	True, False
16	inject_sbit_error	inject_sbit_error	True, False
17	input_data_width	input_data_width	1 - 1024
18	input_depth	input_depth	2^4 - 2^{22}
19	output_data_width	output_data_width	1 - 1024
20	output_depth	output_depth	2^4 - 2^{22}
21	overflow_flag	overflow_flag	True, False
22	overflow_sense	overflow_sense	Active_High, Active_Low
23	performance_options	performance_options	Standard_FIFO (STD), First_Word_Fall_Through (FWFT)
24	programmable_empty_type	programmable_empty_type	No_Programmable_Empty_Threshold Single_Programmable_Empty_Threshold_Constant Multiple_Programmable_Empty_Threshold_Constants Single_Programmable_Empty_Threshold_Input_Port Multiple_Programmable_Empty_Threshold_Input_Ports
25	programmable_full_type	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
26	read_clock_frequency	read_clock_frequency	1 - 1000
27	read_data_count	read_data_count	True, False
28	read_data_count_width	read_data_count_width	1 – $\log_2$(output_depth)
29	reset_pin	reset_pin	True, False
30	reset_type	reset_type	Synchronous_Reset, Asynchronous_Reset
31	underflow_flag	underflow_flag	True, False
32	underflow_sense	underflow_sense	Active_High, Active_Low
33	use_dout_reset	use_dout_reset	True, False
34	use_embedded_registers	use_embedded_registers	True, False

Table B-3: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
35	use_extra_logic	use_extra_logic	True, False
36	valid_flag	valid_flag	True, False
37	valid_sense	valid_sense	Active_High, Active_Low
38	write_acknowledge_flag	write_acknowledge_flag	True, False
39	write_acknowledge_sense	write_acknowledge_sense	Active_High, Active_Low
40	write_clock_frequency	write_clock_frequency	1 – 1000
41	write_data_count	write_data_count	True, False
42	write_data_count_width	write_data_count_width	1 – $\log_2$(input_depth)
43	N/A	interface_type	Native, AXI4
44	N/A	axi_type	AXI4_Stream, AXI4_Full, AXI4_Lite
45	N/A	image_type	Application_Diagram, Block_Diagram
46	N/A	enable_write_channel	True, False
47	N/A	enable_read_channel	True, False
48	N/A	clock_type_axi	Common_Clock, Independent_Clock
49	N/A	use_clock_enable	True, False
50	N/A	clock_enable_type	Slave_Interface_Clock_Enable, Master_Interface_Clock_Enable
51	N/A	id_width	1 - 8
52	N/A	axi_address_width	1 – 32
53	N/A	axi_data_width	2^3 - 2^9
54	N/A	enable_awuser	True, False
55	N/A	enable_wuser	True, False
56	N/A	enable_buser	True, False
57	N/A	enable_aruser	True, False
58	N/A	enable_ruser	True, False
59	N/A	enable_tuser	True, False
60	N/A	awuser_width	1 - 256
61	N/A	wuser_width	1 - 256
62	N/A	buser_width	1 - 256
63	N/A	aruser_width	1 - 256
64	N/A	ruser_width	1 - 256
65	N/A	tuser_width	1 - 256

Table B-3: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
66	N/A	enable_tdata	True, False
67	N/A	enable_tdest	True, False
68	N/A	enable_tid	True, False
69	N/A	enable_tkeep	True, False
70	N/A	enable_tlast	True, False
71	N/A	enable_tready	True, False
72	N/A	enable_tstrobe	True, False
73	N/A	tdata_width	2^3 - 2^9
74	N/A	tdest_width	1 - 4
75	N/A	tid_width	1 - 8
76	N/A	tkeep_width	tdata_width/8
77	N/A	tstrb_width	tdata_width/8
78	N/A	axis_type	FIFO
79	N/A	wach_type	FIFO
80	N/A	wdch_type	FIFO
81	N/A	wrch_type	FIFO
82	N/A	rach_type	FIFO
83	N/A	rdch_type	FIFO
84	N/A	fifo_implementation_type_axis	Common_Clock_Block_RAM Common_Clock_Distributed_RAM Independent_Clock_Block_RAM Independent_Clock_Distributed_RA M
85	N/A	fifo_implementation_type_rach	
86	N/A	fifo_implementation_type_rdch	
87	N/A	fifo_implementation_type_wach	
88	N/A	fifo_implementation_type_wdch	
89	N/A	fifo_implementation_type_wrch	
90	N/A	fifo_application_type_axis	Data_FIFO
91	N/A	fifo_application_type_rach	Data_FIFO
92	N/A	fifo_application_type_rdch	Data_FIFO
93	N/A	fifo_application_type_wach	Data_FIFO
94	N/A	fifo_application_type_wdch	Data_FIFO

Table B-3: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
95	N/A	fifo_application_type_wrch	Data_FIFO
96	N/A	enable_ecc_axis	True, False
97	N/A	enable_ecc_rach	True, False
98	N/A	enable_ecc_rdch	True, False
99	N/A	enable_ecc_wach	True, False
100	N/A	enable_ecc_wdch	True, False
101	N/A	enable_ecc_wrch	True, False
102	N/A	inject_sbit_error_axis	True, False
103	N/A	inject_sbit_error_rach	True, False
104	N/A	inject_sbit_error_rdch	True, False
105	N/A	inject_sbit_error_wach	True, False
106	N/A	inject_sbit_error_wdch	True, False
107	N/A	inject_sbit_error_wrch	True, False
108	N/A	inject_dbit_error_axis	True, False
109	N/A	inject_dbit_error_rach	True, False
110	N/A	inject_dbit_error_rdch	True, False
111	N/A	inject_dbit_error_wach	True, False
112	N/A	inject_dbit_error_wdch	True, False
113	N/A	inject_dbit_error_wrch	True, False
114	N/A	input_depth_axis	2^4 - 2^{16}
115	N/A	input_depth_rach	2^4 - 2^{16}
116	N/A	input_depth_rdch	2^4 - 2^{16}
117	N/A	input_depth_wach	2^4 - 2^{16}
118	N/A	input_depth_wdch	2^4 - 2^{16}
119	N/A	input_depth_wrch	2^4 - 2^{16}
120	N/A	enable_data_counts_axis	True, False
121	N/A	enable_data_counts_rach	True, False
122	N/A	enable_data_counts_rdch	True, False
123	N/A	enable_data_counts_wach	True, False
124	N/A	enable_data_counts_wdch	True, False
125	N/A	enable_data_counts_wrch	True, False
126	N/A	enable_prog_flags_axis	True, False

Table B-3: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
127	N/A	enable_prog_flags_rach	True, False
128	N/A	enable_prog_flags_rdch	True, False
129	N/A	enable_prog_flags_wach	True, False
130	N/A	enable_prog_flags_wdch	True, False
131	N/A	enable_prog_flags_wrch	True, False
132	N/A	programmable_full_type_axis	Full, Almost_Full, Single_Programmable_Full_Threshold _Constant, Single_Programmable_Full_Threshold _Input_Port
133	N/A	programmable_full_type_rach	
134	N/A	programmable_full_type_rdch	
135	N/A	programmable_full_type_wach	
136	N/A	programmable_full_type_wdch	
137	N/A	programmable_full_type_wrch	
138	N/A	full_threshold_assert_value_axis	5 - 65535
139	N/A	full_threshold_assert_value_rach	5 - 65535
140	N/A	full_threshold_assert_value_rdch	5 - 65535
141	N/A	full_threshold_assert_value_wach	5 - 65535
142	N/A	full_threshold_assert_value_wdch	5 - 65535
143	N/A	full_threshold_assert_value_wrch	5 - 65535
144	N/A	programmable_empty_type_axis	Empty, Almost_Empty, Single_Programmable_Empty _Threshold_Constant, Single_Programmable_Empty _Threshold_Input_Port
145	N/A	programmable_empty_type_rach	
146	N/A	programmable_empty_type_rdch	
147	N/A	programmable_empty_type_wach	
148	N/A	programmable_empty_type_wdch	
149	N/A	programmable_empty_type_wrch	
150	N/A	empty_threshold_assert_value_axis	4 - 65534
151	N/A	empty_threshold_assert_value_rach	4 - 65534
152	N/A	empty_threshold_assert_value_rdch	4 - 65534
153	N/A	empty_threshold_assert_value_wach	4 - 65534

Table B-3: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
154	N/A	empty_threshold_assert_value_wdch	4 - 65534
155	N/A	empty_threshold_assert_value_wrch	4 - 65534
156	N/A	underflow_flag_axi	True, False
157	N/A	underflow_sense_axi	Active_High, Active_Low
158	N/A	overflow_flag_axi	True, False
159	N/A	overflow_sense_axi	Active_High, Active_Low
160	N/A	enable_common_overflow	True, False
161	N/A	enable_common_underflow	True, False
162	N/A	disable_timing_violations_axi	True, False
163	N/A	add_ngc_constraint_axi	True, False

DOUT Reset Value Timing

Figure C-1 shows the DOUT reset value for common clock block RAM, distributed RAM and Shift Register based FIFOs for synchronous reset (SRST), and common clock block RAM FIFO for asynchronous reset (RST).

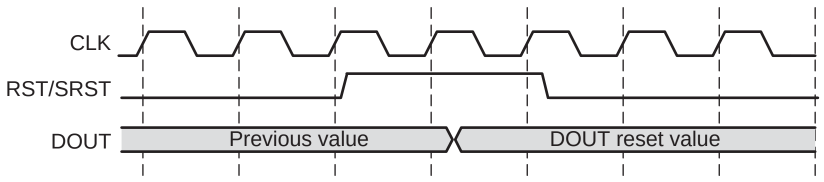


Figure C-1: **DOUT Reset Value for Synchronous Reset (SRST) and for Asynchronous Reset (RST) for Common Clock Block RAM Based FIFO**

Figure C-2 shows the DOUT reset value for common clock distributed RAM and Shift Register based FIFOs for asynchronous reset (RST).

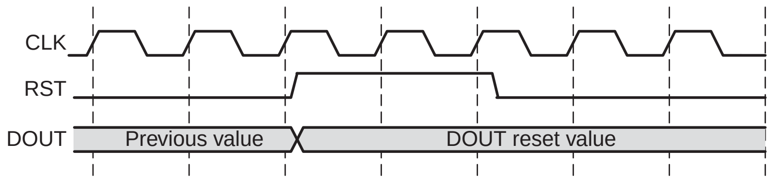


Figure C-2: **DOUT Reset Value for Asynchronous Reset (RST) for Common Clock Distributed/Shift RAM Based FIFO**

Figure C-3 shows the DOUT reset value for Kintex-7, Virtex-7, and Virtex-6 FPGA common clock Built-in FIFOs with Embedded register for asynchronous reset (RST).

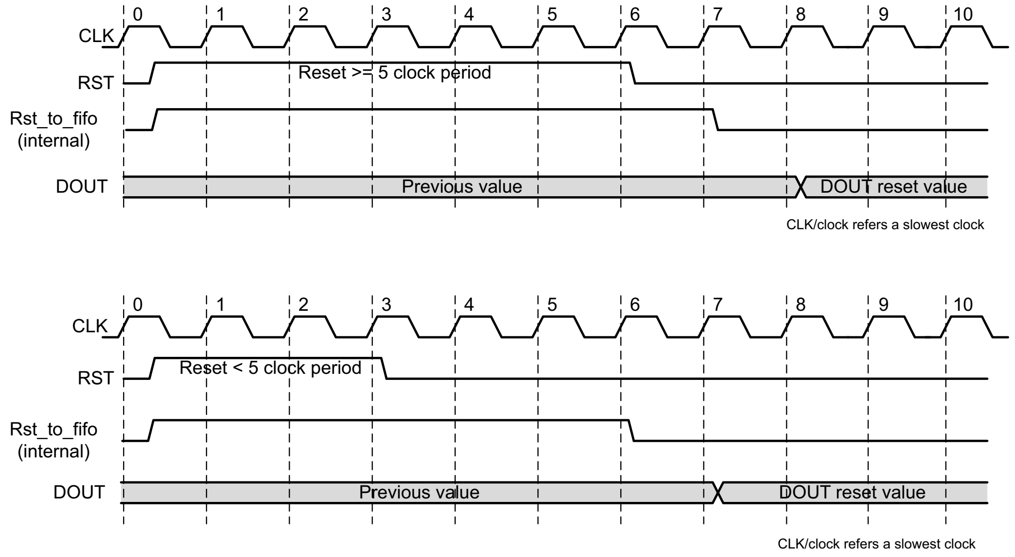


Figure C-3: **DOUT Reset Value for Common Clock Built-in FIFO**

Figure C-4 shows the DOUT reset value for independent clock block RAM based FIFOs (RD_RST).

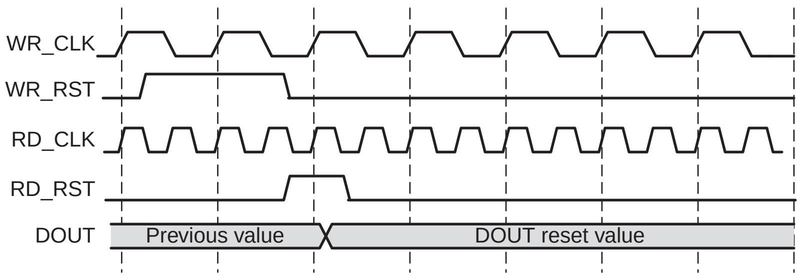


Figure C-4: **DOUT Reset Value for Independent Clock Block RAM Based FIFO**

Figure C-5 shows the DOUT reset value for independent clock distributed RAM based FIFOs (RD_RST).

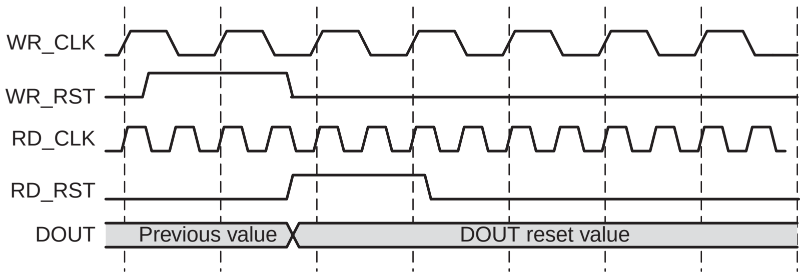


Figure C-5: **DOUT Reset Value for Independent Clock Distributed RAM Based FIFO**