# LogiCORE IP FIFO Generator v10.0

## *Product Guide for Vivado Design Suite*

**PG057 March 20, 2013**

XILINX®

XILINX®

# Table of Contents

## Appendix D: DOUT Reset Value Timing

## Appendix E: Supplemental Information

## Appendix F: Additional Resources

# Introduction

The Xilinx LogiCORE™ IP FIFO Generator is a fully verified first-in first-out (FIFO) memory queue for applications requiring in-order storage and retrieval. The core provides an optimized solution for all FIFO configurations and delivers maximum performance (up to 500 MHz) while utilizing minimum resources. Delivered through the Vivado Design Suite, the structure can be customized by the user including the width, depth, status flags, memory type, and the write/read port aspect ratios.

The FIFO Generator core supports Native interface FIFOs and AXI4 interface FIFOs. The Native interface FIFO cores include the original standard FIFO functions delivered by the previous versions of the FIFO Generator (up to v6.2). Native interface FIFO cores are optimized for buffering, data width conversion and clock domain decoupling applications, providing in-order storage and retrieval.

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

For more details on the features of each interface, see Feature Summary in Chapter 1.

| LogiCORE IP Facts Table | |
|---|---|
| **Core Specifics** | |
| Supported Device Family(1) | Zynq™-7000, Artix-7, Virtex®-7, Kintex®-7 |
| Supported User Interfaces | Native, AXI4-Stream, AXI4, AXI4-Lite |
| Resources | See Table 2-1 through Table 2-6. |
| **Provided with Core** | |
| Design Files | Encrypted RTL |
| Example Design | VHDL |
| Test Bench | VHDL |
| Constraints File | XDC |
| Simulation Model | Verilog and VHDL Behavioral(2) |
| Supported S/W Driver | N/A |
| **Tested Design Flows(3)** | |
| Design Entry | Vivado Design Suite |
| Simulation | Mentor Graphics Questa SIM<br>Vivado XSIM |
| Synthesis | Vivado Synthesis |
| **Support** | |
| Provided by Xilinx @ www.xilinx.com/support | |

**Notes:**

1. For a complete listing of supported devices, see the Vivado IP catalog.
2. Behavioral models do not model synchronization delay. See Simulating Your Design in Appendix E for details.
3. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.

# Overview

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 Vivado IP customizers in the IP Catalog 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.

## Native Interface FIFOs

The Native interface FIFO can be customized to utilize block RAM, distributed RAM or built-in FIFO 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.

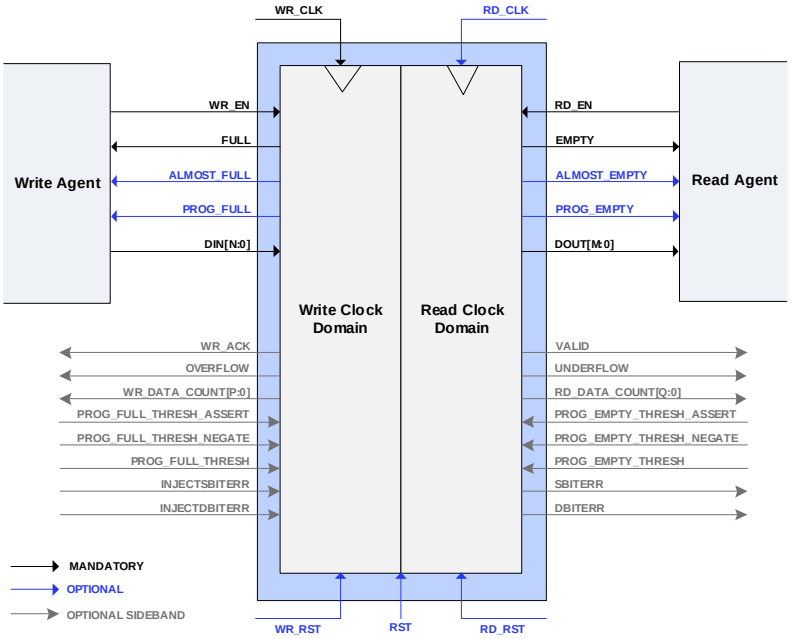


*Figure 1-1:* **Native Interface FIFOs Signal Diagram**

# AXI4 Interface FIFOs

AXI4 interface FIFOs are derived from the Native interface FIFO, as shown in Figure 1-2. 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.

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

Use the AXI4 FIFOs in the same applications supported by the Native Interface FIFO when you need to connect to other AXI functions. AXI4 FIFOs can also be integrated into an EDK embedded system IP by using the EDK Create/Import Peripheral (CIP) wizard. Refer to Chapter 7: Creating Your Own Intellectual Property of the EDK Concepts, Tools and Techniques Guide for details.

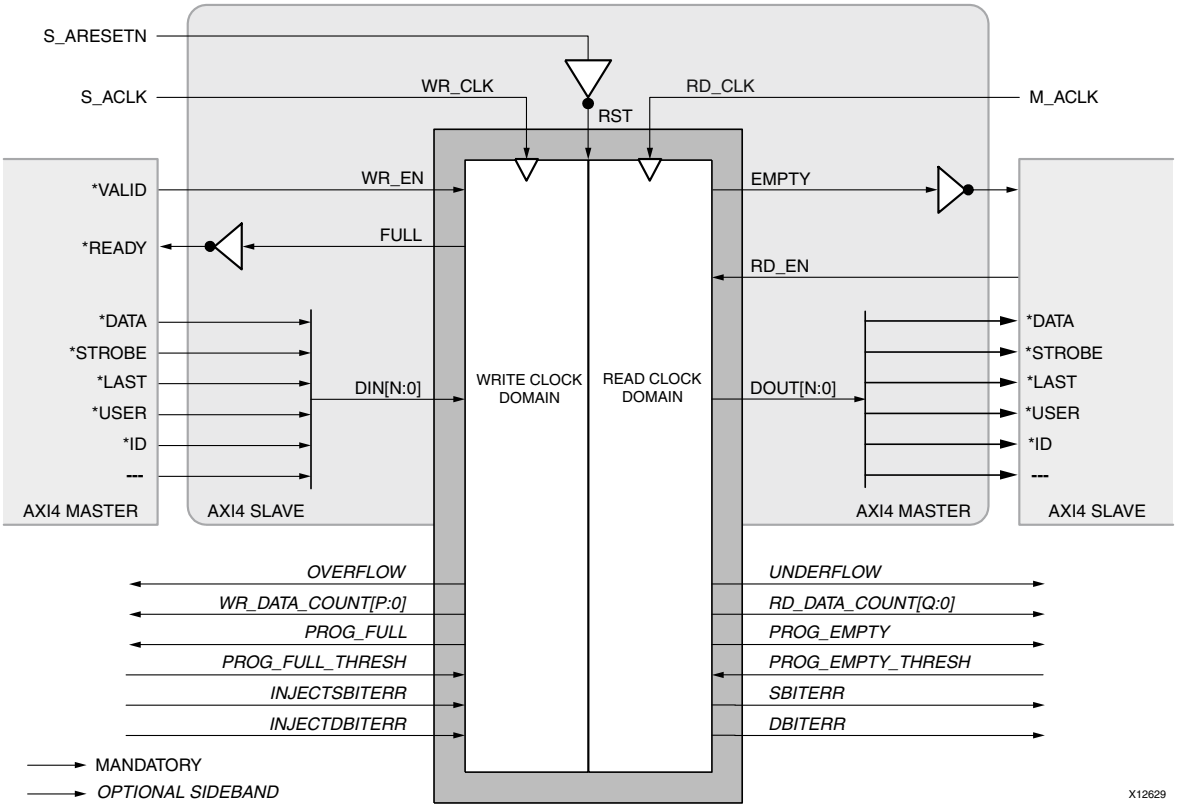


*Figure 1-2:* **AXI4 FIFO Derivation**

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 1-3 shows an example timing diagram for write and read operations to the AXI4-Stream FIFO, and Figure 1-4 shows an example timing diagram for write and read operations to the AXI4/AXI4-Lite FIFO.

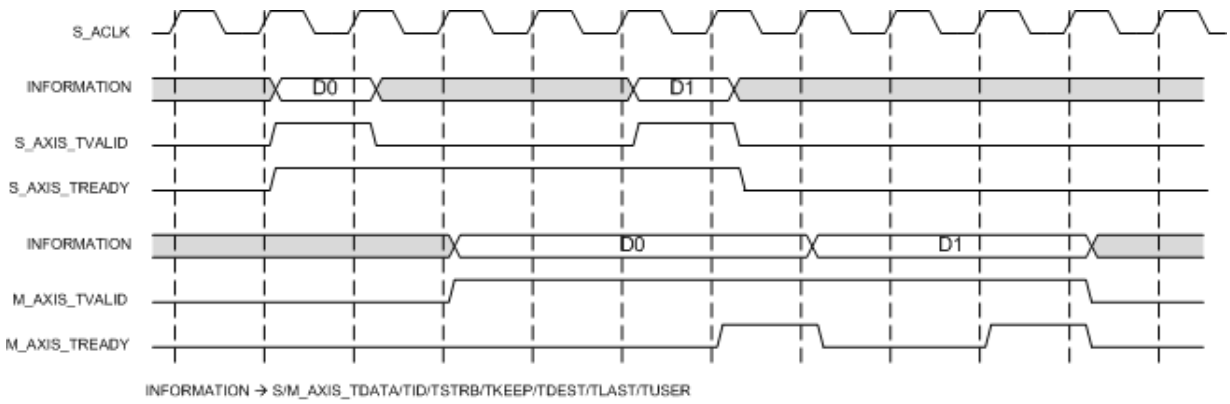


*Figure 1-3:* **AXI4-Stream FIFO Timing Diagram**

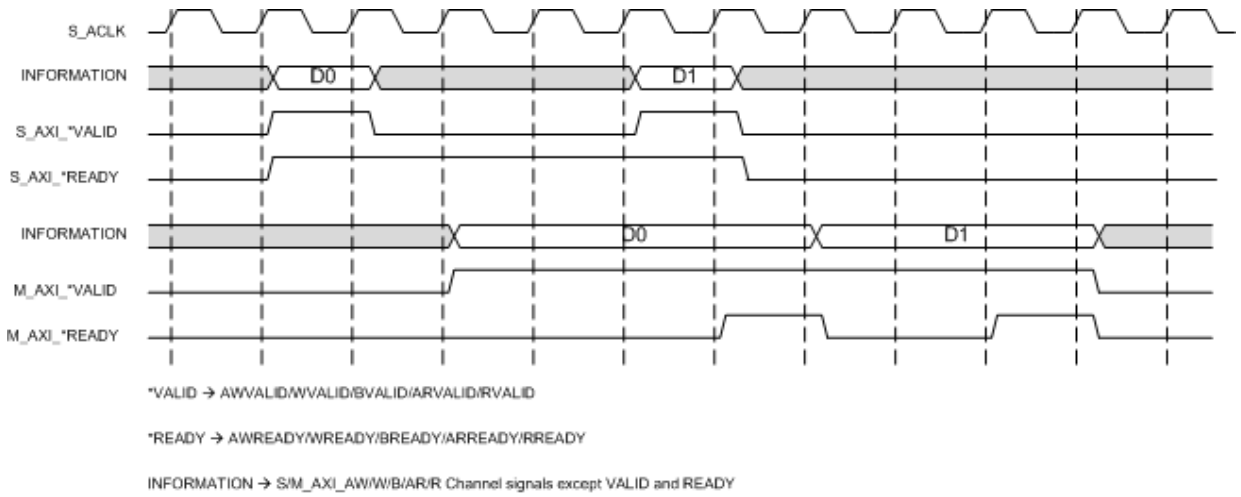


*Figure 1-4:* **AXI4/AXI4-Lite FIFO Timing Diagram**

In Figure 1-4 and Figure 1-3, 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 the VALID and READY handshake signals.

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

# Feature Summary

## Common Features

- Supports Native, AXI4-Stream, AXI4 and AXI4-Lite interfaces
- FIFO depths up to 4,194,304 words

- Independent or common clock domains
- VHDL example design and demonstration test bench demonstrating the IP core design flow, including how to instantiate and simulate it
- Fully configurable using the Xilinx Vivado IP Catalog customizer

## Native FIFO Specific Features

- FIFO data widths from 1 to 1024 bits
- Symmetric or Non-symmetric aspect ratios (read-to-write port ratios ranging from 1:8 to 8:1)
- Synchronous or asynchronous reset option
- Selectable memory type (block RAM, distributed RAM, shift register, or built-in FIFO)
- Option to operate in Standard or First-Word Fall-Through modes (FWFT)
- Full and Empty status flags, and Almost Full and Almost Empty flags for indicating one-word-left
- Programmable Full and Empty status flags, set by user-defined constant(s) or dedicated input port(s)
- Configurable handshake signals
- Hamming Error Injection and Correction Checking (ECC) support for block RAM and Built-in FIFO configurations
- Embedded register option for block RAM and built-in FIFO configurations

## AXI4 FIFO Features

- FIFO data widths from 1 to 4096 bits
- Supports all three AXI4 interface protocols - AXI4, AXI4-Stream, and AXI4-Lite
- Symmetric aspect ratios
- Asynchronous active low reset
- Selectable configuration type (FIFO, Register Slice, or Pass Through Wire)
- Selectable memory type (block RAM, or distributed RAM)
- Selectable application type (Data FIFO, Packet FIFO, or low latency FIFO)
    - Packet FIFO feature is available only for common clock AXI4-Stream and AXI4 FIFOs
- Operates in First-Word Fall-Through mode (FWFT)
- Configurable Interrupt signals

- Auto-calculation of FIFO width based on AXI signal selections and data and address widths
- Hamming Error Injection and Correction Checking (ECC) support for block RAM FIFO configurations
- Configurable programmable Full/Empty flags as sideband signals

# Native FIFO Feature Overview

## Clock Implementation and Operation

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

## Built-in FIFO Support

The FIFO Generator supports the Zynq™-7000 and 7 series (Artix™-7, Virtex-7, and Kintex™-7) FPGA built-in FIFO modules, enabling large FIFOs to be created by cascading the built-in FIFOs in both width and depth. The core expands the capabilities of the built-in FIFOs by utilizing the FPGA fabric to create optional status flags not implemented in the built-in FIFO macro. The built-in Error Correction Checking (ECC) feature in the built-in FIFO macro is also available to the user.

See the appropriate FPGA user guide for frequency requirements.

## First-Word Fall-Through (FWFT)

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). FWFT is useful in applications that require low-latency access to data and to applications that require throttling based on the contents of the data that are read. FWFT support is included in FIFOs created with block RAM, distributed RAM, or built-in FIFOs.

See Table 1-2 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 88).
- Programmable empty (see Non-symmetric Aspect Ratio and First-Word Fall-Through, page 105).

- Data counts (see First-Word Fall-Through Data Count, page 100 and Non-symmetric Aspect Ratio and First-Word Fall-Through, page 105).

### Supported Memory Types

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

*Table 1-1:* **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 with write and read ports of 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 FPGA block RAM and FIFO 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 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, page 106 for more information.

### Error Injection and Correction (ECC) Support

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

## Native FIFO Configuration and Implementation

Table 1-2 defines the supported memory and clock configurations.

*Table 1-2:* **FIFO Configurations**

| Clock Domain | Memory Type | Non-symmetric Aspect Ratios | First-word Fall-Through | ECC Support | Embedded Register Support |
|---|---|---|---|---|---|
| Common | Block RAM | | ✓ | ✓ | ✓[a] |
| Common | Distributed RAM | | ✓ | | |
| Common | Shift Register | | | | |
| Common | Built-in FIFO | | ✓[b] | ✓ | ✓(a) |
| Independent | Block RAM | ✓ | ✓ | ✓ | ✓(a) |
| Independent | Distributed RAM | | ✓ | | |
| Independent | Built-in FIFO[c] | | ✓(b) | ✓ | |

a. Embedded register support is only available for block RAM-based FIFOs and common clock built-in FIFOs.
b. FWFT is supported for Built-in FIFOs only.
c. For non-symmetric aspect ratios, use the block RAM implementation (feature not supported in built-in FIFO primitive).

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

This implementation category allows you 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, you have the option to select a synchronous or asynchronous reset for the core. The block RAM FIFO configuration also supports ECC.

## Common Clock: Built-in FIFO

This implementation category allows you to select the built-in FIFO 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 built-in FIFO configuration also supports the built-in ECC feature.

## Independent Clocks: Block RAM and Distributed RAM

This implementation category allows you 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). The block RAM FIFO configuration also supports ECC.

### Independent Clocks: Built-in FIFO

This implementation category allows you to select the built-in FIFO. 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 built-in FIFO configuration also supports the built-in ECC feature.

## Native FIFO Generator Feature Summary

Table 1-3 summarizes the supported FIFO Generator features for each clock configuration and memory type.

*Table 1-3:* **FIFO Configurations Summary**

| FIFO Feature | Independent Clocks | | | Common Clock | | |
|---|---|---|---|---|---|---|
| | Block RAM | Distributed RAM | Built-in FIFO | Block RAM | Distributed RAM, Shift Register | Built-in FIFO |
| Non-symmetric Aspect Ratios[a] | ✓ | | | | | |
| Symmetric Aspect Ratios | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Almost Full | ✓ | ✓ | | ✓ | ✓ | |
| Almost Empty | ✓ | ✓ | | ✓ | ✓ | |
| Handshaking | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Data Count | ✓ | ✓ | | ✓ | ✓ | |
| Programmable Empty/Full Thresholds | ✓ | ✓ | ✓[b] | ✓ | ✓ | ✓(b) |
| First-Word Fall-Through | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |

*Table 1-3:* **FIFO Configurations Summary *(Cont'd)***

| FIFO Feature | Independent Clocks | | | Common Clock | | |
|---|---|---|---|---|---|---|
| | Block RAM | Distributed RAM | Built-in FIFO | Block RAM | Distributed RAM, Shift Register | Built-in FIFO |
| Synchronous Reset | | | | ✓ | ✓ | |
| Asynchronous Reset | ✓[c] | ✓[(c)] | ✓ | ✓[(c)] | ✓[(c)] | ✓ |
| DOUT Reset Value | ✓ | ✓ | | ✓ | ✓ | ✓[(d)] |
| ECC | ✓[(e)] | | ✓[e] | ✓[(e)] | | ✓[(e)] |
| Embedded Register | ✓[(f)] | | | ✓[(f)] | | ✓[(f)] |

a. 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.
b. For built-in FIFOs, the range of Programmable Empty/Full threshold is limited to take advantage of the logic internal to the macro.
c. Asynchronous reset is optional for all FIFOs built using distributed and block RAM.
d. DOUT Reset Value is supported only in common clock built-in FIFOs.
e. ECC is only supported in block RAM and built-in FIFOs.
f. Embedded register option is only supported in block RAM FIFOs and common clock built-in FIFOs. See Embedded Registers in Block RAM and FIFO Macros in Chapter 1.

## 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 1-5 illustrates these signals (both standard and optional ports) for a FIFO core that supports independent write and read clocks.

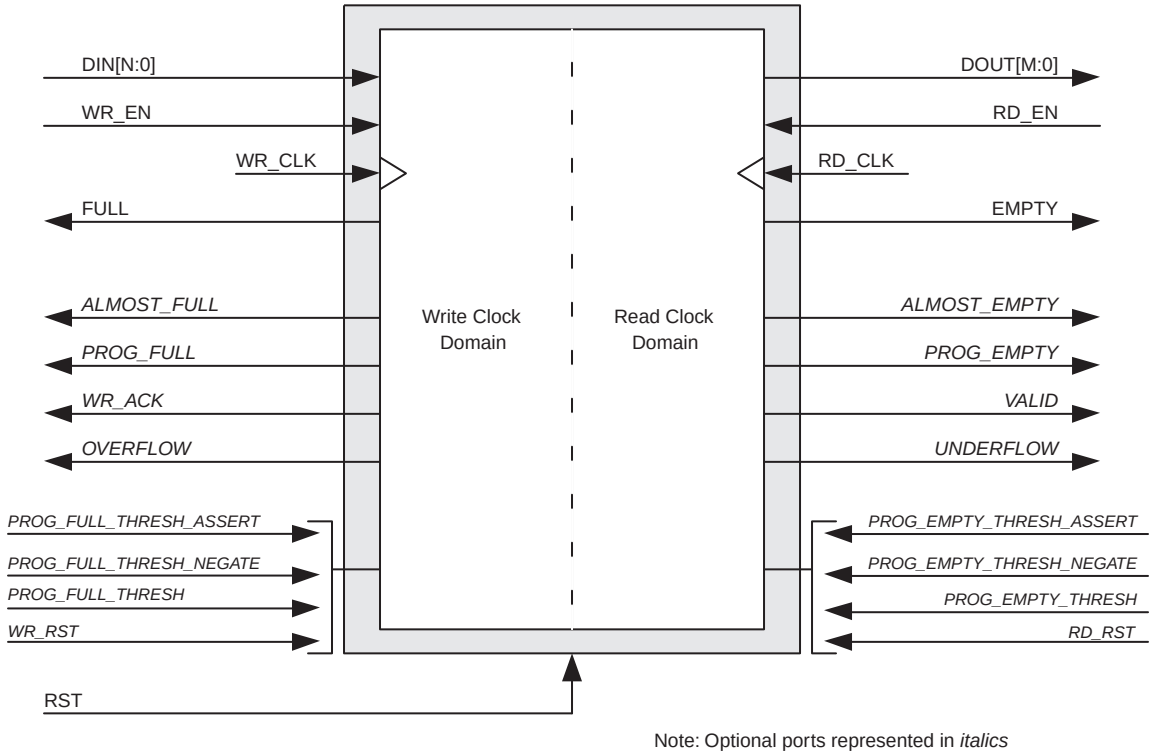


*Figure 1-5:* **FIFO with Independent Clocks: Interface Signals**

## Interface Signals: FIFOs With Independent Clocks

The RST signal, as defined Table 1-4, 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 1-4:* **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 1-5 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 1-5:* **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. |

*Table 1-5:* **Write Interface Signals for FIFOs with Independent Clocks *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| WR_EN | Input | Write Enable: If the FIFO is not full, asserting this signal causes data (on DIN) to be written to the FIFO. |
| FULL | Output | Full Flag: When asserted, this signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is 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 you never overflow 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.<br>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.<br>You can either choose to set the assert and negate threshold to the same value (using PROG_FULL_THRESH), or you 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. Refer to the FIFO Generator GUI for the valid range of values[(a)]. |

*Table 1-5:* **Write Interface Signals for FIFOs with Independent Clocks *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| 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. Refer to FIFO Generator GUI for the valid range of values[(a)]. |
| INJECTSBITERR | Input | Injects a single bit error if the ECC feature is used on block RAMs or built-in FIFO macros. |
| INJECTDBITERR | Input | Injects a double bit error if the ECC feature is used on block RAMs or built-in FIFO macros. |

a. Valid range of values shown in the GUI are the actual values even though they are grayed out for some selections.

Table 1-6 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 1-6:* **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 1-6:* **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 you do 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.<br>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.<br>You can either choose to set the assert and negate threshold to the same value (using PROG_EMPTY_THRESH), or you 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. Refer to the FIFO Generator GUI for the valid range of values[(a)]. |
| 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. Refer to the FIFO Generator GUI for the valid range of values[(a)]. |
| SBITERR | Output | Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on block RAM or built-in FIFO macro. |
| DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error on block RAM or built-in FIFO macro and data in the FIFO core is corrupted. |

a. Valid range of values shown in the GUI are the actual values even though they are grayed out for some selections.

## Interface Signals: FIFOs with Common Clock

Table 1-7 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.

*Table 1-7:* **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. |
| 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.<br>You can either choose to set the assert and negate threshold to the same value (using PROG_FULL_THRESH), or you can control these values independently (using PROG_FULL_THRESH_ASSERT and PROG_FULL_THRESH_NEGATE). |

*Table 1-7:* **Interface Signals for FIFOs with a Common Clock *(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. Refer to the FIFO Generator GUI for the valid range of values[(a)]. |
| 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. Refer to the FIFO Generator GUI for the valid range of values[(a)]. |
| 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.<br>you can either choose to set the assert and negate threshold to the same value (using PROG_EMPTY_THRESH), or you 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. |
| DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error and data in the FIFO core is corrupted. |
| INJECTSBITERR | Input | Injects a single bit error if the ECC feature is used. For detailed information, see Chapter 3, "Designing with the Core." |
| INJECTDBITERR | Input | Injects a double bit error if the ECC feature is used. For detailed information, see Chapter 3, "Designing with the Core." |

a. Valid range of values shown in the GUI are the actual values even though they are grayed out for some selections.

# AXI4 FIFO Feature Overview

## 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, or both Write Channels and Read Channels for AXI4. 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 1-6. 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.

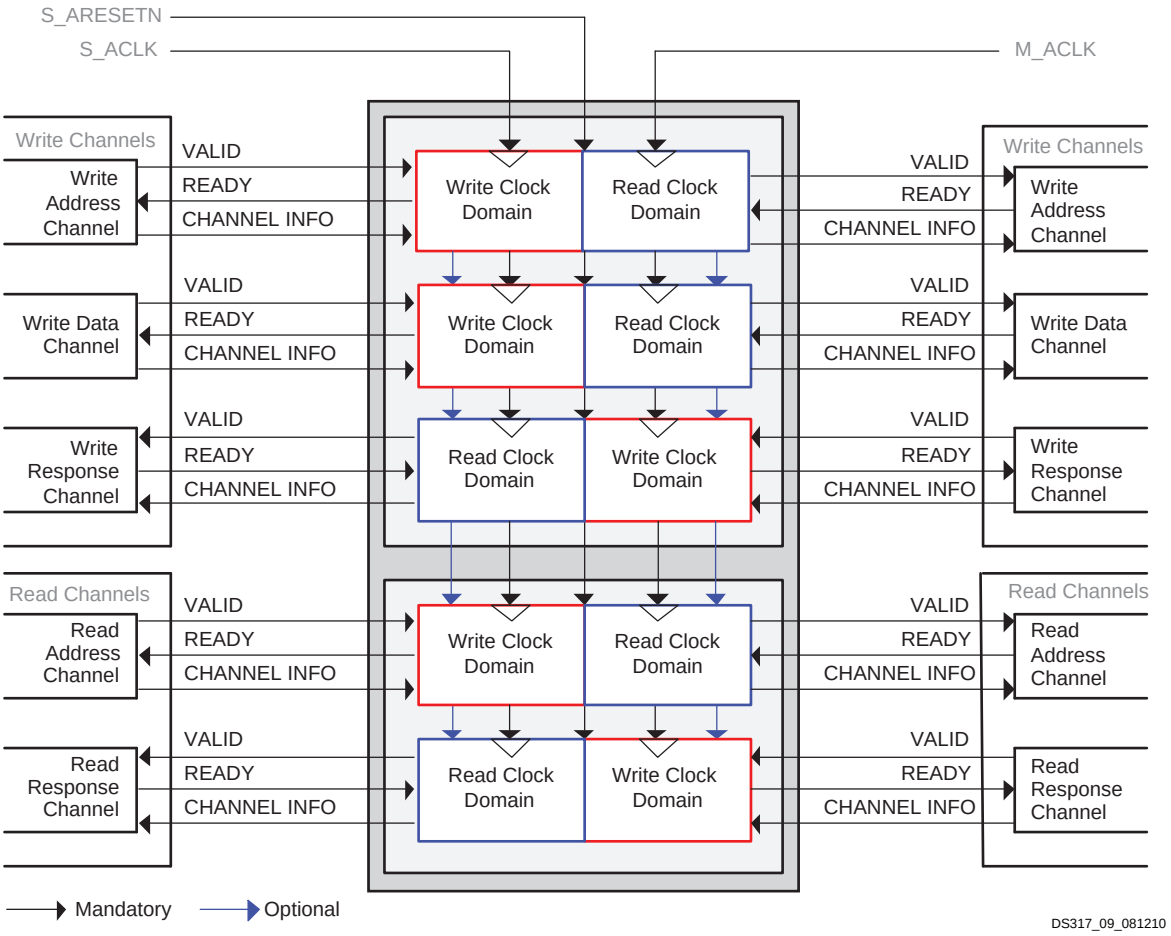


*Figure 1-6:* **AXI4 Block Diagram**

## 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 1-6). In addition, all instantiated FIFOs can support either independent clock or common clock operation.

The independent clock configuration of the FIFO Generator enables you 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 Configuration, Memory and Application Types

The FIFO Generator provides selectable configuration options: FIFO, Register Slice and Pass Through Wire. The core implements FIFOs built from block RAM or distributed RAM memory types. Depending on the application type selection (Data FIFO, Packet FIFO, or low latency FIFO), the core combines memory primitives in an optimal configuration based on the calculated width and selected depth of the FIFO.

## Packet FIFO

The Packet FIFO configuration delays the start of packet (burst) transmission until the end (LAST beat) of the packet is received. This ensures uninterrupted availability of data once master-side transfer begins, thus avoiding source-end stalling of the AXI data channel. This is valuable in applications in which data originates at a master device. Examples of this include a real-time signal channels that operate at a lower data-rate than the downstream AXI switch and/or slave destination, such as a high-bandwidth memory.

The Packet FIFO principle applies to both AXI4 memory-mapped burst transactions (both write and read) and AXI4-Stream packet transmissions. This feature is sometimes referred to as “store-and-forward”, referring to the behavior for memory-mapped writes and stream transmissions. For memory-mapped reads, transactions are delayed until there are enough vacancies in the FIFO to guarantee uninterrupted buffering of the entire read data packet, as predicted by the AR-channel transaction. Read transactions do not actually rely on the RLAST signal.

The Packet FIFO feature is supported for Common Clock AXI4 and AXI4-Stream configurations. It is not supported for AXI4-Lite configurations.

### AXI4-Stream Packet FIFO

The FIFO Generator uses AXI4-Stream Interface for the AXI4-Stream Packet FIFO feature. The FIFO Generator indicates a TVALID on the AXI4-Stream Master side when a complete packet (marked by TLAST) is received on the AXI4-Stream Slave side or when the AXI4-Stream FIFO is FULL. Indicating TVALID on the Master side due to the FIFO becoming FULL is an exceptional case, and in such case, the Packet FIFO acts as a normal FWFT FIFO forwarding the data received on the Slave side to the Master side until it receives TLAST on the Slave side.

**IMPORTANT:** *The depth of the FIFO should be set to at least twice of the maximum packet size. For example, if the maximum size of a packet is 512, then the FIFO depth should be set to 1024.*

### AXI4 Packet FIFO

The FIFO Generator uses the AXI4 Interface for the AXI4 Packet FIFO feature (for both write and read channels).

- Packet FIFO on Write Channels: The FIFO Generator indicates an AWVALID on the AXI4 AW channel Master side when a complete packet (marked by WLAST) is received on the AXI4 W channel Slave side. The Write Channel Packet FIFO is coupled to the Write Address Channel so that AW transfers are not posted to the AXI4 Write Address Channel until all of the data needed for the requested transfer is received on the AXI4 W channel Slave side. The minimum depth of the W channel is set to 512 and enables the Write Channel Packet FIFO to hold two packets of its maximum length.
- Packet FIFO on Read Channels: The FIFO Generator indicates an RVALID on the AXI4 R channel Slave side when a complete packet (marked by RLAST) is received on the AXI4 R channel Master side. The Read Channel Packet FIFO is coupled to the Read Address Channel so that AR transfers are not posted to the AXI4 Read Address Channel if there is not enough space left in the Packet FIFO for the associated data. The minimum depth of the R channel is set to 512, and enables the Read Channel Packet FIFO to hold two packets of its maximum length.

## Error Injection and Correction (ECC) Support

The block RAM macros are equipped with built-in Error Injection and Correction Checking. 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 3.

## AXI4 Slave Interface for Performing Writes

AXI4 FIFOs provide an AXI4 Slave interface for performing Writes. In Figure 1-4, the AXI4 Master provides INFORMATION and VALID signals; the AXI4 FIFO accepts the INFORMATION by asserting the READY signal. The READY signal will be de-asserted only when the FIFO is full.

## AXI4 Master Interface for Performing Reads

The AXI4 FIFO provides an AXI4 Master interface for performing Reads. In Figure 1-4, the AXI4 FIFO provides INFORMATION and VALID signals; upon detecting a READY signal asserted from the 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 empty.

# AXI4 FIFO Feature Summary

Table 1-8 summarizes the supported FIFO Generator features for each clock configuration and memory type.

*Table 1-8:* **AXI4 FIFO Configuration Summary**

| FIFO Options | Common Clock | | Independent Clock | |
|---|---|---|---|---|
| | Block RAM | Distributed Memory | Block RAM | Distributed Memory |
| Full [a] | ✓ | ✓ | ✓ | ✓ |
| Programmable Full [b] | ✓ | ✓ | ✓ | ✓ |
| Empty [c] | ✓ | ✓ | ✓ | ✓ |
| Programmable Empty [b] | ✓ | ✓ | ✓ | ✓ |
| Data Counts | ✓ | ✓ | ✓ | ✓ |
| ECC | ✓ | | ✓ | |
| Interrupt Flags | ✓ | ✓ | ✓ | ✓ |

a. Mapped to S_AXIS_TREADY/S_AXI_AWREADY/S_AXI_WREADY/M_AXI_BREADY/S_AXI_ARREADY/M_AXI_RREADY depending on the Handshake Flag Options in the GUI.

b. Provided as sideband signal depending on the GUI option.

c. Mapped to M_AXIS_TVALID/M_AXI_AWVALID/M_AXI_WVALID/S_AXI_BVALID/M_AXI_ARVALID/S_AXI_RVALID depending on the Handshake Flag Options in the GUI.

# AXI4 FIFO 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 1-9 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 1-9:* **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. |
| **Clock Enable Signals Gated with FIFO's WR_EN and RD_EN Inputs** | | |

*Table 1-9:* **AXI4 FIFO - Global Interface Signals**

| Name | Direction | Description |
|---|---|---|
| S_ACLK_EN | Input | Slave Clock Enable signal gated with WR_EN signal of FIFO |
| M_ACLK_EN | Input | Slave Clock Enable signal gated with RD_EN signal of FIFO |

## AXI4-Stream FIFO Interface Signals

Table 1-10 defines the AXI4-Stream FIFO interface signals.

*Table 1-10:* **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. |
| 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:<br>• STROBE[0] = 1b, DATA[7:0] is valid<br>• 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:<br>• KEEP[0] = 1b, DATA[7:0] is a NULL byte<br>• 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** | | |

*Table 1-10:* **AXI4-Stream FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| 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:<br>• STROBE[0] = 1b, DATA[7:0] is valid<br>• STROBE[7] = 0b, DATA[63:56] is not valid |
| 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:<br>• KEEP[0] = 1b, DATA[7:0] is a NULL byte<br>• 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.<br>*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.<br>*D* = log2(FIFO depth)-1 |
| AXIS_INJECTSBITERR | Input | Inject Single-Bit Error: Injects a single-bit error if the ECC feature is used. |

*Table 1-10:* **AXI4-Stream FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| AXIS_INJECTDBITERR | Input | Inject Double-Bit Error: Injects a double-bit error if the ECC feature is used. |
| AXIS_SBITERR | Output | Single-Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXIS_DBITERR | Output | Double-Bit Error: Indicates that the ECC decoder detected a double-bit error and data in the FIFO core is corrupted. |
| 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |
| 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 you do 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXIS_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXIS_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXIS_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

## AXI4 FIFO Interface Signals

Write Channels

Table 1-11 defines the AXI4 FIFO interface signals for Write Address Channel.

*Table 1-11:* **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. |
| 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:<br>• 1 = Address and control information available.<br>• 0 = Address and control information not available.<br>The address and control information remain stable until the address acknowledge signal, AWREADY, goes high. |

*Table 1-11:* **AXI4 Write Address Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| S_AXI_AWREADY | Output | Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals:<br>• 1 = Slave ready.<br>• 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. |
| 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:<br>• 1 = address and control information available<br>• 0 = address and control information not available.<br>The address and control information remain stable until the address acknowledge signal, AWREADY, goes high. |

*Table 1-11:* **AXI4 Write Address Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| M_AXI_AWREADY | Input | Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals:<br>• 1 = Slave ready.<br>• 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.<br>*D* = $\log_2$(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.<br>*D* = $\log_2$(FIFO depth)-1 |
| AXI_AW_INJECTSBITERR | Input | Inject Single-Bit Error: Injects a single bit error if the ECC feature is used. |
| AXI_AW_INJECTDBITERR | Input | Inject Double-Bit Error: Injects a double bit error if the ECC feature is used. |
| AXI_AW_SBITERR | Output | Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXI_AW_DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>*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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |

*Table 1-11:* **AXI4 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 you do 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_AW_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_AW_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_AW_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

Table 1-12 defines the AXI4 FIFO interface signals for Write Data Channel.

*Table 1-12:* **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:<br>• STROBE[0] = 1b, DATA[7:0] is valid<br>• STROBE[7] = 0b, DATA[63:56] is not valid |
| S_AXI_WLAST | Input | Write Last: Indicates the last transfer in a write burst. |
| S_AXI_WUSER[m:0] | Input | Write Data Channel User |

*Table 1-12:* **AXI4 Write Data Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| **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:<br>• 1 = Write data and strobes available.<br>• 0 = Write data and strobes not available. |
| S_AXI_WREADY | Output | Write Ready: Indicates that the slave can accept the write data:<br>• 1 = Slave ready.<br>• 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:<br>• STROBE[0] = 1b, DATA[7:0] is valid<br>• 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:<br>• 1 = Write data and strobes available .<br>• 0 = Write data and strobes not available. |
| M_AXI_WREADY | Input | Write ready: Indicates that the slave can accept the write data:<br>• 1 = Slave ready.<br>• 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.<br>$D = \log_2(\text{FIFO depth})-1$ |

*Table 1-12:* **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.<br>*D* = $\log_2$(FIFO depth)-1 |
| AXI_W_INJECTSBITERR | Input | Inject Single-Bit Error: Injects a single bit error if the ECC feature is used. |
| AXI_W_INJECTDBITERR | Input | Inject Double-Bit Error: Injects a double bit error if the ECC feature is used. |
| AXI_W_SBITERR | Output | Single-Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXI_W_DBITERR | Output | Double-Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>*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 |
| 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 you do 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.<br>*D* = $\log_2$(FIFO depth)+1 |
| AXI_W_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>*D* = $\log_2$(FIFO depth)+1 |

*Table 1-12:* **AXI4 Write Data Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| AXI_W_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_W_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

Table 1-13 defines the AXI4 FIFO interface signals for Write Response Channel.

*Table 1-13:* **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:<br>• 1 = Write response available.<br>• 0 = Write response not available. |
| S_AXI_BREADY | Input | Response Ready: Indicates that the master can accept the response information.<br>• 1 = Master ready.<br>• 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** | | |

*Table 1-13:* **AXI4 Write Response Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| M_AXI_BVALID | Input | Write Response Valid: Indicates that a valid write response is available:<br>• 1 = Write response available.<br>• 0 = Write response not available. |
| M_AXI_BREADY | Output | Response Ready: Indicates that the master can accept the response information.<br>• 1 = Master ready.<br>• 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.<br>*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.<br>*D* = $\log_2$(FIFO depth)-1 |
| AXI_B_INJECTSBITERR | Input | Inject Single-Bit Error: Injects a single bit error if the ECC feature is used. |
| AXI_B_INJECTDBITERR | Input | Inject Double-Bit Error: Injects a double bit error if the ECC feature is used. |
| AXI_B_SBITERR | Output | Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXI_B_DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>*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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |

*Table 1-13:* **AXI4 Write Response Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| 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 you do 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_B_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_B_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_B_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

Read Channels

Table 1-14 defines the AXI4 FIFO interface signals for Read Address Channel.

*Table 1-14:* **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.<br>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. |

*Table 1-14:* **AXI4 Read Address Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| 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.<br>• 1 = Address and control information valid.<br>• 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:<br>• 1 = Slave ready.<br>• 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. |
| M_AXI_ARADDR[m:0] | Output | Read Address: The read address bus gives the initial address of a read burst transaction.<br>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. |

*Table 1-14:* **AXI4 Read Address Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| 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.<br>• 1 = Address and control information valid.<br>• 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:<br>• 1 = Slave ready.<br>• 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.<br>$D = \log_2$(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.<br>$D = \log_2$(FIFO depth)-1 |
| AXI_AR_INJECTSBITERR | Input | Inject Single-Bit Error: Injects a single bit error if the ECC feature is used. |
| AXI_AR_INJECTDBITERR | Input | Inject Double-Bit Error: Injects a double bit error if the ECC feature is used. |
| AXI_AR_SBITERR | Output | Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXI_AR_DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |

*Table 1-14:* **AXI4 Read Address Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| 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 you never overflow 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |
| 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 you do 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_AR_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_AR_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_AR_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

Table 1-15 defines the AXI4 FIFO interface signals for Read Data Channel.

*Table 1-15:* **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. |

*Table 1-15:* **AXI4 Read Data Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| 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:<br>• 1 = Read data available.<br>• 0 = Read data not available. |
| S_AXI_RREADY | Input | Read Ready: Indicates that the master can accept the read data and response information:<br>• 1= Master ready.<br>• 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:<br>• 1 = Read data available.<br>• 0 = Read data not available. |
| M_AXI_RREADY | Output | Read Ready: Indicates that the master can accept the read data and response information:<br>• 1= Master ready.<br>• 0 = Master not ready. |
| **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.<br>$D = \log_2(\text{FIFO depth})-1$ |

*Table 1-15:* **AXI4 Read Data Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| 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.<br>*D* = $\log_2$(FIFO depth)-1 |
| AXI_R_INJECTSBITERR | Input | Injects a single bit error if the ECC feature is used. |
| AXI_R_INJECTDBITERR | Input | Injects a double bit error if the ECC feature is used. |
| AXI_R_SBITERR | Output | Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXI_R_DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>*D* = $\log_2$(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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |
| 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 you do 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.<br>*D* = $\log_2$(FIFO depth)+1 |
| AXI_R_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>*D* = $\log_2$(FIFO depth)+1 |

*Table 1-15:* **AXI4 Read Data Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| AXI_R_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_R_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

## AXI4-Lite FIFO Interface Signals

### Write Channels

Table 1-16 defines the AXI4-Lite FIFO interface signals for Write Address Channel.

*Table 1-16:* **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:<br>• 1 = Address and control information available.<br>• 0 = Address and control information not available.<br>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:<br>• 1 = Slave ready.<br>• 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. |

*Table 1-16:* **AXI4-Lite Write Address Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| **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:<br>• 1 = Address and control information available.<br>• 0 = Address and control information not available.<br>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:<br>• 1 = Slave ready.<br>• 0 = Slave not ready. |
| **AXI4-Lite 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.<br>$D = \log_2(\text{FIFO depth})-1$ |
| 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.<br>$D = \log_2(\text{FIFO depth})-1$ |
| AXI_AW_INJECTSBITERR | Input | Inject Single-Bit Error: Injects a single bit error if the ECC feature is used. |
| AXI_AW_INJECTDBITERR | Input | Inject Double-Bit Error: Injects a double bit error if the ECC feature is used. |
| AXI_AW_SBITERR | Output | Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXI_AW_DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>$D = \log_2(\text{FIFO depth})+1$ |

*Table 1-16:* **AXI4-Lite Write Address Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |
| 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 you do 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_AW_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_AW_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_AW_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

Table 1-17 defines the AXI4-Lite FIFO interface signals for Write Data Channel.

*Table 1-17:* **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:<br>• STROBE[0] = 1b, DATA[7:0] is valid<br>• STROBE[7] = 0b, DATA[63:56] is not valid |
| **AXI4-Lite Interface Write Data Channel: Handshake Signals for FIFO Write Interface** | | |

*Table 1-17:* **AXI4-Lite Write Data Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| S_AXI_WVALID | Input | Write Valid: Indicates that valid write data and strobes are available:<br>• 1 = Write data and strobes available.<br>• 0 = Write data and strobes not available. |
| S_AXI_WREADY | Output | Write Ready: Indicates that the slave can accept the write data:<br>• 1 = Slave ready.<br>• 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:<br>• STROBE[0] = 1b, DATA[7:0] is valid<br>• STROBE[7] = 0b, DATA[63:56] is not valid |
| **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:<br>• 1 = Write data and strobes available.<br>• 0 = Write data and strobes not available. |
| M_AXI_WREADY | Input | Write Ready: Indicates that the slave can accept the write data:<br>• 1 = Slave ready.<br>• 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.<br>$D = \log_2(\text{FIFO depth})-1$ |
| 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.<br>$D = \log_2(\text{FIFO depth})-1$ |
| AXI_W_INJECTSBITERR | Input | Injects a single bit error if the ECC feature is used. |
| AXI_W_INJECTDBITERR | Input | Injects a double bit error if the ECC feature is used. |
| AXI_W_SBITERR | Output | Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |

*Table 1-17:* **AXI4-Lite Write Data Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| AXI_W_DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |
| 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 you do 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_W_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_W_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_W_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

Table 1-18 defines the AXI4-Lite FIFO interface signals for Write Response Channel.

*Table 1-18:* **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:<br>• 1 = Write response available.<br>• 0 = Write response not available. |
| S_AXI_BREADY | Input | Response Ready: Indicates that the master can accept the response information.<br>• 1 = Master ready.<br>• 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:<br>• 1 = Write response available.<br>• 0 = Write response not available. |
| M_AXI_BREADY | Output | Response ready: Indicates that the master can accept the response information.<br>• 1 = Master ready.<br>• 0 = Master not ready. |
| **AXI4-Lite 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.<br>*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.<br>D is than log2(FIFO depth)-1 |
| AXI_B_INJECTSBITERR | Input | Injects a single bit error if the ECC feature is used. |
| AXI_B_INJECTDBITERR | Input | Injects a double bit error if the ECC feature is used. |

*Table 1-18:* **AXI4-Lite Write Response Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| AXI_B_SBITERR | Output | Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXI_B_DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |
| 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 you do 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_B_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_B_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_B_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

Read Channels

Table 1-19 defines the AXI4-Lite FIFO interface signals for Read Address Channel.

*Table 1-19:* **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.<br>• 1 = Address and control information valid.<br>• 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:<br>• 1 = Slave ready.<br>• 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.<br>• 1 = Address and control information valid.<br>• 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:<br>• 1 = Slave ready.<br>• 0 = Slave not ready. |

*Table 1-19:* **AXI4-Lite Read Address Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| **AXI4-Lite 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.<br>*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.<br>*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. |
| AXI_AR_INJECTDBITERR | Input | Inject Double-Bit Error: Injects a double-bit error if the ECC feature is used. |
| AXI_AR_SBITERR | Output | Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXI_AR_DBITERR | Output | Double Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>*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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |
| 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 you do 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.<br>*D* = $\log_2$(FIFO depth)+1 |

*Table 1-19:* **AXI4-Lite Read Address Channel FIFO Interface Signals *(Cont'd)***

| Name | Direction | Description |
|---|---|---|
| AXI_AR_DATA_COUNT[D:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_AR_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_AR_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

Table 1-20 defines the AXI4-Lite FIFO interface signals for Write Data Channel.

*Table 1-20:* **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:<br>• 1 = Read data available.<br>• 0 = Read data not available. |
| S_AXI_RREADY | Input | Read Ready: indicates that the master can accept the read data and response information:<br>• 1= Master ready.<br>• 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:<br>• 1 = Read data available.<br>• 0 = Read data not available. |

*Table 1-20:* **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:<br>• 1= Master ready.<br>• 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.<br>*D* = $\log_2$(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.<br>*D* = $\log_2$(FIFO depth)-1 |
| AXI_R_INJECTSBITERR | Input | Inject Single-Bit Error: Injects a single bit error if the ECC feature is used. |
| AXI_R_INJECTDBITERR | Input | Inject DOuble-Bit Error. Injects a double bit error if the ECC feature is used. |
| AXI_R_SBITERR | Output | Single-Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error. |
| AXI_R_DBITERR | Output | Double-Bit Error: Indicates that the ECC decoder detected a double-bit error 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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full. |
| 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 you never overflow 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.<br>*D* = $\log_2$(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.<br>**Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty. |

*Table 1-20:* **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 you do 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.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_R_DATA_COUNT[*D*:0] | Output | Data Count: This bus indicates the number of words stored in the FIFO.<br>$D = \log_2(\text{FIFO depth})+1$ |
| AXI_R_PROG_FULL | Output | Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO is less than the programmable threshold. |
| AXI_R_PROG_EMPTY | Output | Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is deasserted when the number of words in the FIFO exceeds the programmable threshold. |

# Applications

## Native FIFO Applications

In digital designs, FIFOs are ubiquitous constructs required for data manipulation tasks such as clock domain crossing, low-latency memory buffering, and bus width conversion. Figure 1-7 highlights just one of many configurations that the FIFO Generator supports. In this example, the design has two independent clock domains and the width of the write data bus is four times wider than the read data bus. Using the FIFO Generator, you are able

to rapidly generate solutions such as this one, that is customized for their specific requirements and provides a solution fully optimized for Xilinx FPGAs.

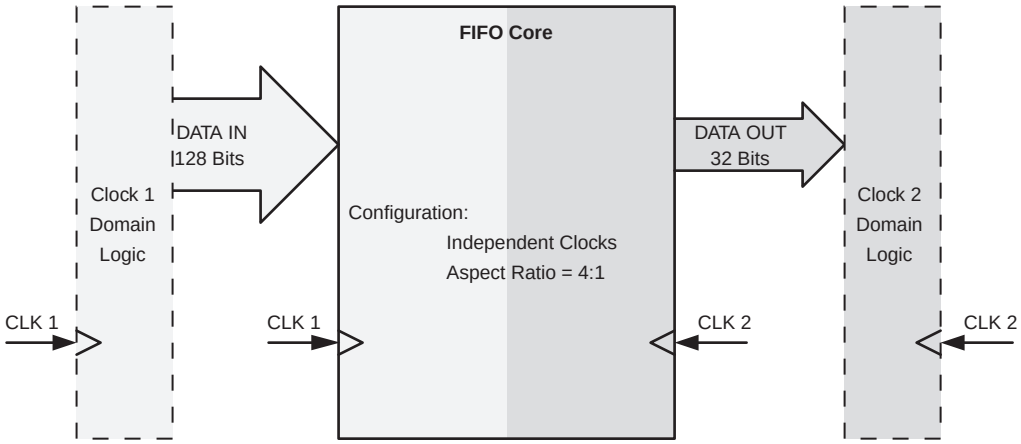


*Figure 1-7:* **FIFO Generator Application Example**

# AXI4 FIFO Applications

## AXI4-Stream FIFOs

AXI4-Stream FIFOs are best for non-address-based, point-to-point applications. Use them to interface to other IP cores using this interface (for example, AXI4 versions of DSP functions such as FFT, DDS, and FIR Compiler).

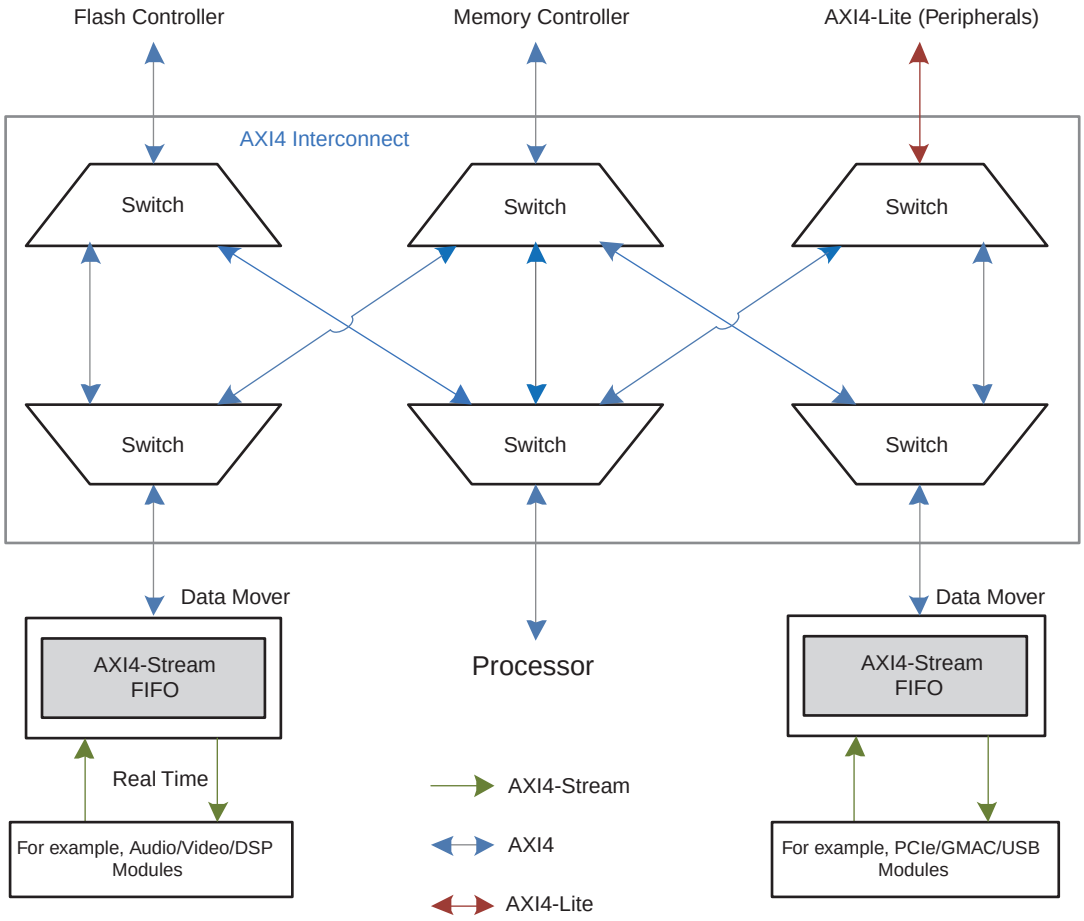


*Figure 1-8:* **AXI4-Stream Application Diagram**

Figure 1-8 illustrates the use of AXI4-Stream FIFOs to create a Data Mover block. In this application, the Data Mover is used to interface PCI Express, Ethernet MAC and USB modules which have a LocalLink to an AXI4 System Bus. The AXI4 Interconnect and Data Mover blocks shown in Figure 1-8 are Embedded IP cores which are available in the Xilinx Embedded Development Kit (EDK).

AXI4-Stream FIFOs support most of the features that the Native interface FIFOs support in first word fall through mode. Use AXI4-Stream FIFOs to replace Native interface FIFOs to make interfacing to the latest versions of other AXI4 LogiCORE IP functions easier.

## AXI4 FIFOs (Memory Mapped)

The full version of the AXI4 Interface is referred to as AXI4. It may also be referred to as AXI Memory Mapped. Use AXI4 FIFOs in memory mapped system bus designs such as bridging applications requiring a memory mapped interface to connect to other AXI4 blocks.

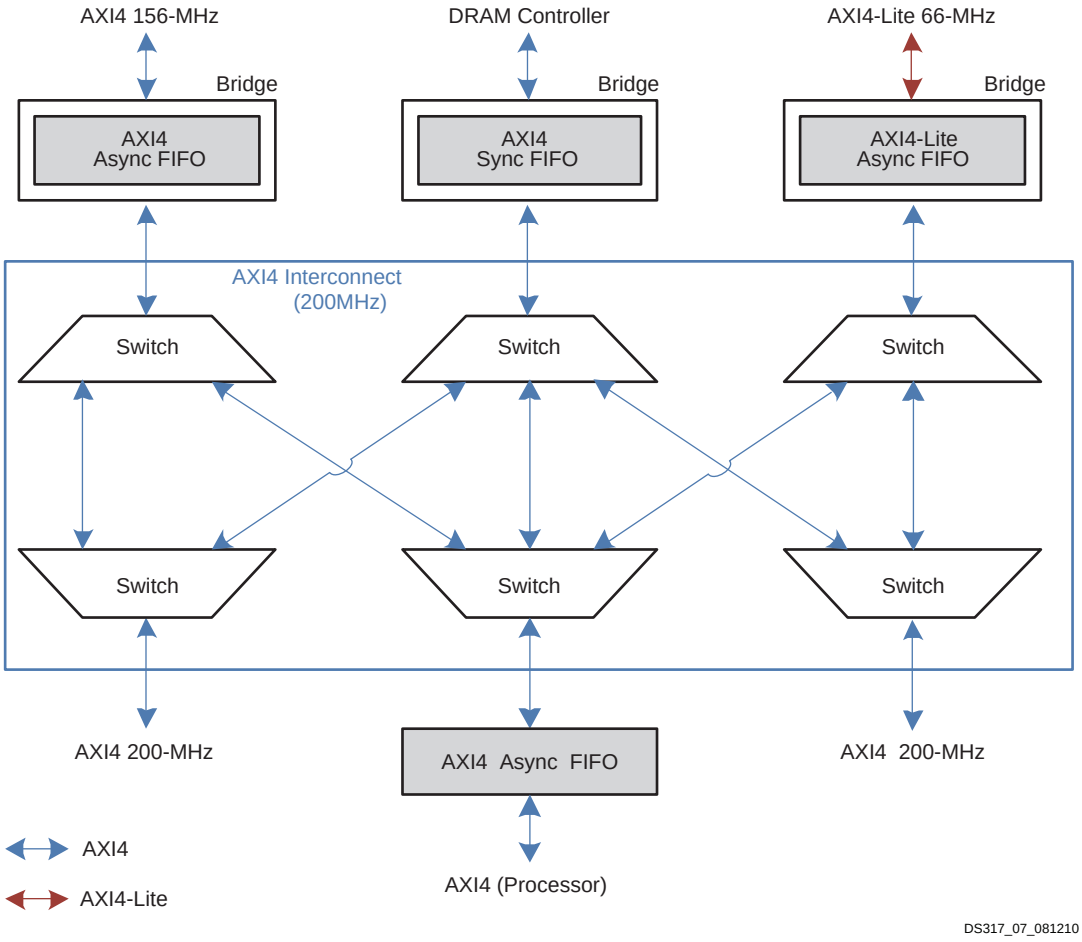


*Figure 1-9:* **AXI4 Application Diagram**

Figure 1-9 shows an example application for AXI4 FIFOs where they are used in AXI4-to-AXI4 bridging applications enabling different AXI4 clock domains running at 200, 100, 66, and 156 MHz to communicate with each other. The AXI4-to-AXI4-Lite bridging is another pertinent application for AXI4 FIFO (for example, for performing protocol conversion). The AXI4 FIFOs can also used inside an IP core to buffer data or transactions (for example, a DRAM Controller). The AXI4 Interconnect block shown in Figure 1-9 is an Embedded IP core available in the EDK.

## AXI4-Lite FIFOs

The AXI4-Lite interface is a simpler AXI interface that supports applications that only need to perform simple Control/Status Register accesses, or peripherals access.

Figure 1-10 shows an AXI4-Lite FIFO being used in an AXI4 to AXI4-Lite bridging application to perform protocol conversion. The AXI4-Lite Interconnect in Figure 1-10 is also available as an Embedded IP core in the EDK.

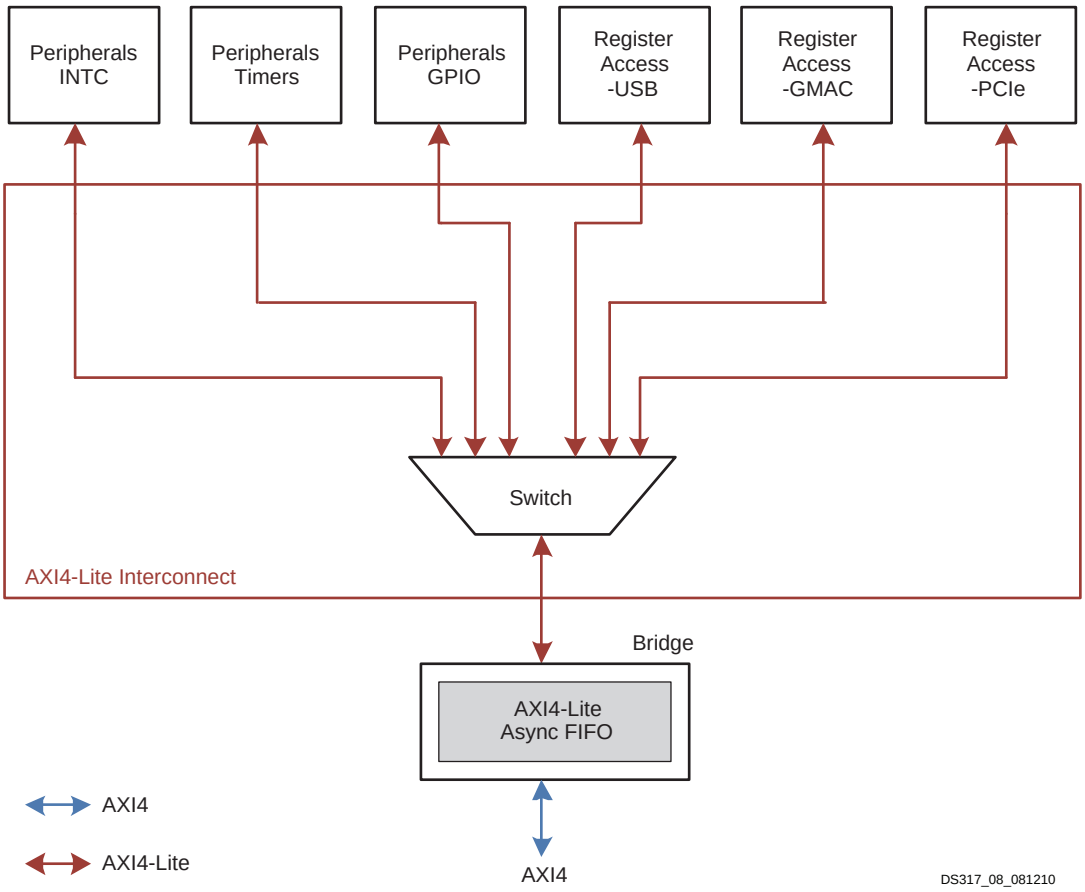


*Figure 1-10:* **AXI4-Lite Application Diagram**

# Licensing and Ordering Information

This Xilinx LogiCORE IP module is provided at no additional cost with the Xilinx Vivado Design Suite tools under the terms of the Xilinx End User License. Information about this and other Xilinx LogiCORE IP modules is available at the Xilinx Intellectual Property page. For information about pricing and availability of other Xilinx LogiCORE IP modules and tools, contact your local Xilinx sales representative.

For more information, please visit the FIFO Generator core page.

# Product Specification

This chapter includes details on performance and latency.

## 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 the Resource Utilization section for the performance and resource utilization numbers.

### Latency

The latency of output signals of FIFO varies for different configurations. See Latency in Chapter 3 for more details.

## Resource Utilization

### Native FIFO Resource Utilization and Performance

Performance and resource utilization for a Native interface FIFO varies depending on the configuration and features selected during core customization. Table 2-1 through Table 2-3 show resource utilization data and maximum performance values for a variety of sample FIFO configurations.

The benchmarks were performed while adding two levels of registers on all inputs (except clock) and outputs having only the period constraints in the XDC. To achieve the performance shown in the following tables, ensure that all inputs to the FIFO are registered and that the outputs are not passed through many logic levels.

**TIP:** *The Shift Register FIFO is more suitable in terms of resource and performance compared to the Distributed Memory FIFO, where the depth of the FIFO is around 16 or 32.*

Table 2-1 identifies the results for a FIFO configured without optional features. Benchmarks were performed using the following devices:

***Note:*** These benchmarks were obtained using ISE Design Suite. Results for use with Vivado Design Suite are expected to be similar.

- Artix-7 (XC7A350T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)

***Note:*** Zynq-7000 device benchmarks are similar to 7 series resource usage.

*Table 2-1:* **Benchmarks: FIFO Configured without Optional Features**

| FIFO Type | Depth x Width | FPGA Family | Performance (MHz) | Resources | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | LUTs | FFs | Block RAM | Shift Register | Distributed RAM |
| Common Clock FIFO (Block RAM) | 512 x 16 | Artix-7 | 270 | 47 | 48 | 1 | 0 | 0 |
| | | Kintex-7 | 325 | 114 | 48 | 1 | 0 | 0 |
| | | Virtex-7 | 325 | 112 | 48 | 1 | 0 | 0 |
| | 4096 x 16 | Artix-7 | 265 | 66 | 60 | 2 | 0 | 0 |
| | | Kintex-7 | 350 | 121 | 60 | 2 | 0 | 0 |
| | | Virtex-7 | 355 | 127 | 60 | 2 | 0 | 0 |
| Common Clock FIFO (Distributed RAM) | 512 x 16 | Artix-7 | 250 | 254 | 68 | 0 | 0 | 176 |
| | | Kintex-7 | 345 | 309 | 65 | 0 | 0 | 176 |
| | | Virtex-7 | 350 | 324 | 65 | 0 | 0 | 176 |
| | 64 x 16 | Artix-7 | 325 | 66 | 52 | 0 | 0 | 22 |
| | | Kintex-7 | 420 | 109 | 52 | 0 | 0 | 22 |
| | | Virtex-7 | 440 | 110 | 52 | 0 | 0 | 22 |
| Independent Clock FIFO (Block RAM) | 512 x 16 | Artix-7 | 265 | 82 | 132 | 1 | 0 | 0 |
| | | Kintex-7 | 335 | 141 | 132 | 1 | 0 | 0 |
| | | Virtex-7 | 335 | 150 | 132 | 1 | 0 | 0 |
| | 4096 x 16 | Artix-7 | 275 | 100 | 172 | 2 | 0 | 0 |
| | | Kintex-7 | 340 | 157 | 172 | 2 | 0 | 0 |
| | | Virtex-7 | 350 | 187 | 172 | 2 | 0 | 0 |
| Independent Clock FIFO (Distributed RAM) | 512 x 16 | Artix-7 | 275 | 279 | 148 | 0 | 0 | 176 |
| | | Kintex-7 | 355 | 338 | 148 | 0 | 0 | 176 |
| | | Virtex-7 | 370 | 354 | 148 | 0 | 0 | 176 |
| | 64 x 16 | Artix-7 | 365 | 67 | 110 | 0 | 0 | 22 |
| | | Kintex-7 | 445 | 124 | 110 | 0 | 0 | 22 |
| | | Virtex-7 | 475 | 145 | 110 | 0 | 0 | 22 |

*Table 2-1:* **Benchmarks: FIFO Configured without Optional Features *(Cont'd)***

| FIFO Type | Depth x Width | FPGA Family | Performance (MHz) | Resources | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | LUTs | FFs | Block RAM | Shift Register | Distributed RAM |
| Shifting Register FIFO | 512 x 16 | Artix-7 | 195 | 711 | 53 | 0 | 496 | 22 |
| | | Kintex-7 | 250 | 768 | 53 | 0 | 497 | 0 |
| | | Virtex-7 | 240 | 768 | 53 | 0 | 496 | 0 |
| | 64 x 16 | Artix-7 | 300 | 126 | 44 | 0 | 64 | 0 |
| | | Kintex-7 | 420 | 162 | 44 | 0 | 64 | 0 |
| | | Virtex-7 | 410 | 179 | 44 | 0 | 64 | 0 |

Table 2-2 provides results for FIFOs configured with multiple programmable thresholds. Benchmarks were performed using the following devices:

***Note:*** These benchmarks were obtained using ISE Design Suite. Results for use with Vivado Design Suite are expected to be similar.

- Artix-7 (XC7A350T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)

***Note:*** Zynq-7000 device benchmarks are similar to 7 series resource usage.

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

| FIFO Type | Depth x Width | FPGA Family | Performance (MHz) | Resources | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | LUTs | FFs | Block RAM | Shift Register | Distributed RAM |
| Common Clock FIFO (Block RAM) | 512 x 16 | Artix-7 | 245 | 76 | 72 | 1 | 0 | 0 |
| | | Kintex-7 | 325 | 130 | 72 | 1 | 0 | 0 |
| | | Virtex-7 | 325 | 139 | 72 | 1 | 0 | 0 |
| | 4096 x 16 | Artix-7 | 265 | 97 | 90 | 2 | 0 | 0 |
| | | Kintex-7 | 340 | 152 | 90 | 2 | 0 | 0 |
| | | Virtex-7 | 375 | 156 | 90 | 2 | 0 | 0 |
| Common Clock FIFO (Distributed RAM) | 512 x 16 | Artix-7 | 250 | 282 | 95 | 0 | 0 | 176 |
| | | Kintex-7 | 355 | 345 | 88 | 0 | 0 | 176 |
| | | Virtex-7 | 350 | 338 | 88 | 0 | 0 | 176 |
| | 64 x 16 | Artix-7 | 290 | 83 | 70 | 0 | 0 | 22 |
| | | Kintex-7 | 400 | 136 | 70 | 0 | 0 | 22 |
| | | Virtex-7 | 335 | 128 | 70 | 0 | 0 | 22 |

*Table 2-2:* **Benchmarks: FIFO Configured with Multiple Programmable Thresholds *(Cont'd)***

| FIFO Type | Depth x Width | FPGA Family | Performance (MHz) | Resources | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | LUTs | FFs | Block RAM | Shift Register | Distributed RAM |
| Independent Clock FIFO (Block RAM) | 512 x 16 | Artix-7 | 265 | 116 | 152 | 1 | 0 | 0 |
| | | Kintex-7 | 325 | 168 | 152 | 1 | 0 | 0 |
| | | Virtex-7 | 330 | 181 | 152 | 1 | 0 | 0 |
| | 4096 x 16 | Artix-7 | 285 | 144 | 197 | 2 | 0 | 0 |
| | | Kintex-7 | 350 | 202 | 197 | 2 | 0 | 0 |
| | | Virtex-7 | 320 | 221 | 197 | 2 | 0 | 0 |
| Independent Clock FIFO (Distributed RAM) | 512 x 16 | Artix-7 | 255 | 311 | 169 | 0 | 0 | 176 |
| | | Kintex-7 | 355 | 376 | 169 | 0 | 0 | 176 |
| | | Virtex-7 | 365 | 382 | 169 | 0 | 0 | 176 |
| | 64 x 16 | Artix-7 | 345 | 92 | 124 | 0 | 0 | 22 |
| | | Kintex-7 | 450 | 136 | 124 | 0 | 0 | 22 |
| | | Virtex-7 | 470 | 159 | 124 | 0 | 0 | 22 |
| Shifting Register FIFO | 512 x 16 | Artix-7 | 190 | 756 | 76 | 0 | 512 | 0 |
| | | Kintex-7 | 245 | 814 | 76 | 0 | 512 | 0 |
| | | Virtex-7 | 225 | 819 | 76 | 0 | 512 | 0 |
| | 64 x 16 | Artix-7 | 295 | 130 | 61 | 0 | 64 | 0 |
| | | Kintex-7 | 400 | 177 | 61 | 0 | 64 | 0 |
| | | Virtex-7 | 400 | 176 | 61 | 0 | 64 | 0 |

Table 2-3 provides results for FIFOs configured to use the built-in FIFO. The benchmarks were performed using the following devices:

***Note:*** These benchmarks were obtained using ISE Design Suite. Results for use with Vivado Design Suite are expected to be similar.

- Artix-7 (XC7A350T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)

***Note:*** Zynq-7000 device benchmarks are similar to 7 series resource usage.

*Table 2-3:* **Benchmarks: FIFO Configured with FIFO36E1 Resources**

| FIFO Type | Depth x Width | FPGA Family | Read Mode | Performance (MHz) | LUTs | FFs | FIFO36 |
|---|---|---|---|---|---|---|---|
| Common Clock FIFO36E1 (Basic) | 512 x 72 | Artix-7 | Standard | 265 | 3 | 7 | 1 |
| | | | FWFT | 255 | 3 | 9 | 1 |
| | | Kintex-7 | Standard | 320 | 2 | 7 | 1 |
| | | | FWFT | 310 | 3 | 9 | 1 |
| | | Virtex-7 | Standard | 215 | 2 | 7 | 1 |
| | | | FWFT | 290 | 4 | 9 | 1 |
| | 16k x 8 | Artix-7 | Standard | 225 | 8 | 11 | 4 |
| | | | FWFT | 220 | 9 | 15 | 4 |
| | | Kintex-7 | Standard | 265 | 8 | 11 | 4 |
| | | | FWFT | 270 | 8 | 15 | 4 |
| | | Virtex-7 | Standard | 205 | 7 | 11 | 4 |
| | | | FWFT | 235 | 8 | 15 | 4 |
| Common Clock FIFO36E1 (With Handshaking) | 512 x 72 | Artix-7 | Standard | 260 | 7 | 11 | 1 |
| | | | FWFT | 250 | 6 | 12 | 1 |
| | | Kintex-7 | Standard | 320 | 6 | 11 | 1 |
| | | | FWFT | 300 | 6 | 12 | 1 |
| | | Virtex-7 | Standard | 210 | 6 | 11 | 1 |
| | | | FWFT | 300 | 6 | 12 | 1 |
| | 16k x 8 | Artix-7 | Standard | 220 | 11 | 15 | 4 |
| | | | FWFT | 225 | 14 | 18 | 4 |
| | | Kintex-7 | Standard | 250 | 11 | 15 | 4 |
| | | | FWFT | 270 | 12 | 18 | 4 |
| | | Virtex-7 | Standard | 250 | 10 | 15 | 4 |
| | | | FWFT | 215 | 11 | 18 | 4 |

*Table 2-3:* **Benchmarks: FIFO Configured with FIFO36E1 Resources *(Cont'd)***

| FIFO Type | Depth x Width | FPGA Family | Read Mode | Performance (MHz) | LUTs | FFs | FIFO36 |
|---|---|---|---|---|---|---|---|
| Independent Clock FIFO36E1 (Basic) | 512 x 72 | Artix-7 | Standard | 300 | 3 | 7 | 1 |
| | | | FWFT | 305 | 3 | 7 | 1 |
| | | Kintex-7 | Standard | 385 | 2 | 7 | 1 |
| | | | FWFT | 385 | 2 | 7 | 1 |
| | | Virtex-7 | Standard | 315 | 2 | 7 | 1 |
| | | | FWFT | 315 | 2 | 7 | 1 |
| | 16k x 8 | Artix-7 | Standard | 255 | 6 | 7 | 4 |
| | | | FWFT | 245 | 5 | 7 | 4 |
| | | Kintex-7 | Standard | 335 | 5 | 7 | 4 |
| | | | FWFT | 345 | 5 | 7 | 4 |
| | | Virtex-7 | Standard | 250 | 5 | 7 | 4 |
| | | | FWFT | 320 | 5 | 7 | 4 |
| Independent Clock FIFO36E1 (With Handshaking) | 512 x 72 | Artix-7 | Standard | 280 | 7 | 18 | 1 |
| | | | FWFT | 345 | 6 | 10 | 1 |
| | | Kintex-7 | Standard | 410 | 8 | 18 | 1 |
| | | | FWFT | 410 | 5 | 10 | 1 |
| | | Virtex-7 | Standard | 330 | 7 | 18 | 1 |
| | | | FWFT | 400 | 5 | 10 | 1 |
| | 16k x 8 | Artix-7 | Standard | 255 | 10 | 18 | 4 |
| | | | FWFT | 265 | 8 | 10 | 4 |
| | | Kintex-7 | Standard | 315 | 10 | 18 | 4 |
| | | | FWFT | 315 | 8 | 10 | 4 |
| | | Virtex-7 | Standard | 220 | 9 | 18 | 4 |
| | | | FWFT | 210 | 8 | 10 | 4 |

## AXI4 FIFO Resource Utilization and Performance

Table 2-4 provides the default configuration settings for the benchmarks data. Table 2-5 shows benchmark information for AXI4 and AXI4-Lite configurations. The benchmarks were obtained using the following devices:

***Note:*** These benchmarks were obtained using ISE Design Suite. Results for use with Vivado Design Suite are expected to be similar.

- Artix-7 (XC7A350T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)

*Table 2-4:* **AXI4 and AXI4-Lite Default Configuration Settings**

| AXI Type | FIFO Type | Channel Type | ID, Address and Data Width | FIFO Depth x Width |
|---|---|---|---|---|
| AXI4 | Distributed RAM | Write Address | ID = 4<br>Address = 32<br>Data = 64[a] | 16 x 66 |
| | Block RAM | Write Data | | 1024 x 77 |
| | Distributed RAM | Write Response | | 16 x 6 |
| | Distributed RAM | Read Address | | 16 x 66 |
| | Block RAM | Read Data | | 1024 x 71 |
| AXI4-Lite | Distributed RAM | Write Address | ID = 4<br>Address = 32<br>Data = 32 | 16 x 35 |
| | Block RAM | Write Data | | 1024 x 36 |
| | Distributed RAM | Write Response | | 16 x 2 |
| | Distributed RAM | Read Address | | 16 x 35 |
| | Block RAM | Read Data | | 1024 x 34 |

*Table 2-5:* **AXI4 and AXI4-Lite Resource Utilization**

| FIFO Type | Clock Type | FPGA Family | Performance (MHz) | Resources | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | LUTs | FFs | Block RAM | Shift Register | Distributed RAM |
| AXI4 | Common Clock | Artix-7 | 260 | 344 | 601 | 5 | 0 | 92 |
| | | Kintex-7 | 315 | 231 | 601 | 2 | 0 | 92 |
| | | Virtex-7 | 179 | 326 | 601 | 5 | 0 | 92 |
| | Independent Clock | Artix-7 | 231 | 430 | 894 | 5 | 0 | 92 |
| | | Kintex-7 | 335 | 394 | 768 | 2 | 0 | 92 |
| | | Virtex-7 | 194 | 453 | 895 | 5 | 0 | 92 |
| AXI4-Lite | Common Clock | Artix-7 | 245 | 234 | 457 | 4 | 0 | 52 |
| | | Kintex-7 | 350 | 194 | 457 | 2 | 0 | 52 |
| | | Virtex-7 | 214 | 238 | 457 | 4 | 0 | 52 |
| | Independent Clock | Artix-7 | 240 | 343 | 752 | 4 | 0 | 52 |
| | | Kintex-7 | 350 | 273 | 650 | 2 | 0 | 52 |
| | | Virtex-7 | 190 | 394 | 752 | 4 | 0 | 52 |

Table 2-6 provides benchmarking results for AXI4-Stream FIFO configurations. The benchmarks were obtained using the following devices:

***Note:*** These benchmarks were obtained using ISE Design Suite. Results for use with Vivado Design Suite are expected to be similar.

- Artix-7 (XC7A350T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)

*Table 2-6:* **AXI4-Stream Resource Utilization**

| FIFO Type | FPGA Family | Depth x Width | Performance (MHz) | Resources | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | LUTs | FFs | Block RAM | Shift Register | Distributed RAM |
| Common Clock FIFO (Block RAM) | 512 x 16 | Artix-7 | 254 | 55 | 67 | 1 | 0 | 0 |
| | | Kintex-7 | 355 | 116 | 67 | 1 | 0 | 0 |
| | | Virtex-7 | 329 | 109 | 67 | 1 | 0 | 0 |
| | 4096 x 16 | Artix-7 | 260 | 79 | 79 | 2 | 0 | 0 |
| | | Kintex-7 | 325 | 123 | 79 | 2 | 0 | 0 |
| | | Virtex-7 | 325 | 117 | 79 | 2 | 0 | 0 |
| Common Clock FIFO (Distributed RAM) | 512 x 16 | Artix-7 | 259 | 262 | 87 | 0 | 0 | 176 |
| | | Kintex-7 | 378 | 318 | 83 | 0 | 0 | 176 |
| | | Virtex-7 | 349 | 321 | 83 | 0 | 0 | 176 |
| | 64 x16 | Artix-7 | 308 | 61 | 71 | 0 | 0 | 22 |
| | | Kintex-7 | 445 | 121 | 71 | 0 | 0 | 22 |
| | | Virtex-7 | 466 | 115 | 71 | 0 | 0 | 22 |
| Independent Clock FIFO (Block RAM) | 512 x 16 | Artix-7 | 266 | 87 | 151 | 1 | 0 | 0 |
| | | Kintex-7 | 355 | 151 | 151 | 1 | 0 | 0 |
| | | Virtex-7 | 325 | 159 | 151 | 1 | 0 | 0 |
| | 4096 x 16 | Artix-7 | 282 | 127 | 190 | 2 | 0 | 0 |
| | | Kintex-7 | 355 | 171 | 190 | 2 | 0 | 0 |
| | | Virtex-7 | 350 | 201 | 190 | 2 | 0 | 0 |
| Independent Clock FIFO (Distributed RAM) | 512 x 16 | Artix-7 | 110 | 283 | 167 | 0 | 0 | 176 |
| | | Kintex-7 | 375 | 351 | 167 | 0 | 0 | 176 |
| | | Virtex-7 | 395 | 363 | 167 | 0 | 0 | 176 |
| | 64 x 16 | Artix-7 | 340 | 103 | 128 | 0 | 0 | 22 |
| | | Kintex-7 | 485 | 154 | 128 | 0 | 0 | 22 |
| | | Virtex-7 | 495 | 173 | 128 | 0 | 0 | 22 |

# Port Descriptions

## Native FIFO Port Summary

Table 2-7 describes all the FIFO Generator ports.

*Table 2-7:* **FIFO Generator Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| RST | I | Yes | Yes | Yes |
| SRST | I | Yes | No | Yes |
| CLK | I | No | No | Yes |
| DATA_COUNT[C:0] | O | Yes | No | Yes |
| Write Interface Signals | | | | |
| WR_CLK | I | No | Yes | No |
| DIN[N:0] | I | No | Yes | Yes |
| WR_EN | I | No | Yes | Yes |
| FULL | O | No | Yes | Yes |
| ALMOST_FULL | O | Yes | Yes | Yes |
| PROG_FULL | O | Yes | Yes | Yes |
| WR_DATA_COUNT[D:0] | O | Yes | Yes | No |
| WR_ACK | O | Yes | Yes | Yes |
| OVERFLOW | O | Yes | Yes | Yes |
| PROG_FULL_THRESH | I | Yes | Yes | Yes |
| PROG_FULL_THRESH_ASSERT | I | Yes | Yes | Yes |
| PROG_FULL_THRESH_NEGATE | I | Yes | Yes | Yes |
| WR_RST | I | Yes | Yes | No |
| INJECTSBITERR | I | Yes | Yes | Yes |
| INJECTDBITERR | I | Yes | Yes | Yes |
| Read Interface Signals | | | | |
| RD_CLK | I | No | Yes | No |
| DOUT[M:0] | O | No | Yes | Yes |
| RD_EN | I | No | Yes | Yes |
| EMPTY | O | No | Yes | Yes |
| ALMOST_EMPTY | O | Yes | Yes | Yes |

*Table 2-7:* **FIFO Generator Ports *(Cont'd)***

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| PROG_EMPTY | O | Yes | Yes | Yes |
| RD_DATA_COUNT[C:0] | O | Yes | Yes | No |
| VALID | O | Yes | Yes | Yes |
| UNDERFLOW | O | Yes | Yes | Yes |
| PROG_EMPTY_THRESH | I | Yes | Yes | Yes |
| PROG_EMPTY_THRESH_ASSERT | I | Yes | Yes | Yes |
| PROG_EMPTY_THRESH_NEGATE | I | Yes | Yes | Yes |
| SBITERR | O | Yes | Yes | Yes |
| DBITERR | O | Yes | Yes | Yes |
| RD_RST | I | Yes | Yes | No |

# AXI4 FIFO Port Summary

## AXI4 Global Interface Ports

*Table 2-8:* **AXI4 FIFO - Global Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **Global Clock and Reset Signals Mapped to FIFO Clock and Reset Inputs** | | | | |
| M_ACLK | Input | Yes | Yes | No |
| S_ACLK | Input | No | Yes | Yes |
| S_ARESETN | Input | No | Yes | Yes |

## AXI4-Stream FIFO Interface Ports

*Table 2-9:* **AXI4-Stream FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4-Stream Interface: Handshake Signals for FIFO Read Interface** | | | | |
| M_AXIS_TVALID | Output | No | Yes | Yes |
| M_AXIS_TREADY | Input | No | Yes | Yes |
| **AXI4-Stream Interface: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| M_AXIS_TDATA[m-1:0] | Output | No | Yes | Yes |
| M_AXIS_TSTRB[m/8-1:0] | Output | Yes | Yes | Yes |

*Table 2-9:* **AXI4-Stream FIFO Interface Ports *(Cont'd)***

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| M_AXIS_TKEEP[m/8-1:0] | Output | Yes | Yes | Yes |
| M_AXIS_TLAST | Output | Yes | Yes | Yes |
| M_AXIS_TID[m:0] | Output | Yes | Yes | Yes |
| M_AXIS_TDEST[m:0] | Output | Yes | Yes | Yes |
| M_AXIS_TUSER[m:0] | Output | Yes | Yes | Yes |
| **AXI4-Stream Interface: Handshake Signals for FIFO Write Interface** | | | | |
| S_AXIS_TVALID | Input | No | Yes | Yes |
| S_AXIS_TREADY | Output | No | Yes | Yes |
| **AXI4-Stream Interface: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |
| S_AXIS_TDATA[m-1:0] | Input | No | Yes | Yes |
| S_AXIS_TSTRB[m/8-1:0] | Input | Yes | Yes | Yes |
| S_AXIS_TKEEP[m/8-1:0] | Input | Yes | Yes | Yes |
| S_AXIS_TLAST | Input | Yes | Yes | Yes |
| S_AXIS_TID[m:0] | Input | Yes | Yes | Yes |
| S_AXIS_TDEST[m:0] | Input | Yes | Yes | Yes |
| S_AXIS_TUSER[m:0] | Input | Yes | Yes | Yes |
| **AXI4-Stream FIFO: Optional Sideband Signals** | | | | |
| AXIS_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXIS_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXIS_INJECTSBITERR | Input | Yes | Yes | Yes |
| AXIS_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXIS_SBITERR | Output | Yes | Yes | Yes |
| AXIS_DBITERR | Output | Yes | Yes | Yes |
| AXIS_OVERFLOW | Output | Yes | Yes | Yes |
| AXIS_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXIS_UNDERFLOW | Output | Yes | Yes | Yes |
| AXIS_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXIS_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXIS_PROG_FULL | Output | Yes | Yes | Yes |
| AXIS_PROG_EMPTY | Output | Yes | Yes | Yes |

## AXI4 FIFO Interface Ports

### Write Channels

*Table 2-10:* **AXI4 Write Address Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4 Interface Write Address Channel: Information Signals Mapped to FIFO Data Input (DIN) bus** | | | | |
| S_AXI_AWID[m:0] | Input | No | Yes | Yes |
| S_AXI_AWADDR[m:0] | Input | No | Yes | Yes |
| S_AXI_AWLEN[7:0] | Input | No | Yes | Yes |
| S_AXI_AWSIZE[2:0] | Input | No | Yes | Yes |
| S_AXI_AWBURST[1:0] | Input | No | Yes | Yes |
| S_AXI_AWLOCK[2:0] | Input | No | Yes | Yes |
| S_AXI_AWCACHE[4:0] | Input | No | Yes | Yes |
| S_AXI_AWPROT[3:0] | Input | No | Yes | Yes |
| S_AXI_AWQOS[3:0] | Input | No | Yes | Yes |
| S_AXI_AWREGION[3:0] | Input | No | Yes | Yes |
| S_AXI_AWUSER[m:0] | Input | Yes | Yes | Yes |
| **AXI4 Interface Write Address Channel: Handshake Signals for FIFO Write Interface** | | | | |
| S_AXI_AWVALID | Input | No | Yes | Yes |
| S_AXI_AWREADY | Output | No | Yes | Yes |
| **AXI4 Interface Write Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| M_AXI_AWID[m:0] | Output | No | Yes | Yes |
| M_AXI_AWADDR[m:0] | Output | No | Yes | Yes |
| M_AXI_AWLEN[7:0] | Output | No | Yes | Yes |
| M_AXI_AWSIZE[2:0] | Output | No | Yes | Yes |
| M_AXI_AWBURST[1:0] | Output | No | Yes | Yes |
| M_AXI_AWLOCK[2:0] | Output | No | Yes | Yes |
| M_AXI_AWCACHE[4:0] | Output | No | Yes | Yes |
| M_AXI_AWPROT[3:0] | Output | No | Yes | Yes |
| M_AXI_AWQOS[3:0] | Output | No | Yes | Yes |
| M_AXI_AWREGION[3:0] | Output | No | Yes | Yes |
| M_AXI_AWUSER[m:0] | Output | Yes | Yes | Yes |
| **AXI4 Interface Write Address Channel: Handshake Signals for FIFO Read Interface** | | | | |
| M_AXI_AWVALID | Output | No | Yes | Yes |

*Table 2-10:* **AXI4 Write Address Channel FIFO Interface Ports *(Cont'd)***

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| M_AXI_AWREADY | Input | No | Yes | Yes |
| **AXI4 Write Address Channel FIFO: Optional Sideband Signals** | | | | |
| AXI_AW_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_AW_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_AW_INJECTSBITERR | Input | Yes | Yes | Yes |
| AXI_AW_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_AW_SBITERR | Output | Yes | Yes | Yes |
| AXI_AW_DBITERR | Output | Yes | Yes | Yes |
| AXI_AW_OVERFLOW | Output | Yes | Yes | Yes |
| AXI_AW_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_AW_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_AW_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_AW_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_AW_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_AW_PROG_EMPTY | Output | Yes | Yes | Yes |

*Table 2-11:* **AXI4 Write Data Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4 Interface Write Data Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |
| S_AXI_WID[m:0] | Input | No | Yes | Yes |
| S_AXI_WDATA[m-1:0] | Input | No | Yes | Yes |
| S_AXI_WSTRB[m/8-1:0] | Input | No | Yes | Yes |
| S_AXI_WLAST | Input | No | Yes | Yes |
| S_AXI_WUSER[m:0] | Input | Yes | Yes | Yes |
| **AXI4 Interface Write Data Channel: Handshake Signals for FIFO Write Interface** | | | | |
| S_AXI_WVALID | Input | No | Yes | Yes |
| S_AXI_WREADY | Output | No | Yes | Yes |
| **AXI4 Interface Write Data Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| M_AXI_WID[m:0] | Output | No | Yes | Yes |
| M_AXI_WDATA[m-1:0] | Output | No | Yes | Yes |
| M_AXI_WSTRB[m/8-1:0] | Output | No | Yes | Yes |

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

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| M_AXI_WLAST | Output | No | Yes | Yes |
| M_AXI_WUSER[m:0] | Output | Yes | Yes | Yes |
| **AXI4 Interface Write Data Channel: Handshake Signals for FIFO Read Interface** | | | | |
| M_AXI_WVALID | Output | No | Yes | Yes |
| M_AXI_WREADY | Input | No | Yes | Yes |
| **AXI4 Write Data Channel FIFO: Optional Sideband Signals** | | | | |
| AXI_W_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_W_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_W_INJECTSBITERR | Input | Yes | Yes | Yes |
| AXI_W_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_W_SBITERR | Output | Yes | Yes | Yes |
| AXI_W_DBITERR | Output | Yes | Yes | Yes |
| AXI_W_OVERFLOW | Output | Yes | Yes | Yes |
| AXI_W_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_W_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_W_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_W_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_W_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_W_PROG_EMPTY | Output | Yes | Yes | Yes |

*Table 2-12:* **AXI4 Write Response Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4 Interface Write Response Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| S_AXI_BID[m:0] | Output | No | Yes | Yes |
| S_AXI_BRESP[1:0] | Output | No | Yes | Yes |
| S_AXI_BUSER[m:0] | Output | Yes | Yes | Yes |
| **AXI4 Interface Write Response Channel: Handshake Signals for FIFO Read Interface** | | | | |
| S_AXI_BVALID | Output | No | Yes | Yes |
| S_AXI_BREADY | Input | No | Yes | Yes |
| **AXI4 Interface Write Response Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |

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

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| M_AXI_BID[m:0] | Input | No | Yes | Yes |
| M_AXI_BRESP[1:0] | Input | No | Yes | Yes |
| M_AXI_BUSER[m:0] | Input | Yes | Yes | Yes |
| **AXI4 Interface Write Response Channel: Handshake Signals for FIFO Write Interface** | | | | |
| M_AXI_BVALID | Input | No | Yes | Yes |
| M_AXI_BREADY | Output | No | Yes | Yes |
| **AXI4 Write Response Channel FIFO: Optional Sideband Signals** | | | | |
| AXI_B_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_B_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_B_INJECTSBITERR | Input | Yes | Yes | Yes |
| AXI_B_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_B_SBITERR | Output | Yes | Yes | Yes |
| AXI_B_DBITERR | Output | Yes | Yes | Yes |
| AXI_B_OVERFLOW | Output | Yes | Yes | Yes |
| AXI_B_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_B_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_B_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_B_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_B_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_B_PROG_EMPTY | Output | Yes | Yes | Yes |

## Read Channels

*Table 2-13:* **AXI4 Read Address Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4 Interface Read Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |
| S_AXI_ARID[m:0] | Input | No | Yes | Yes |
| S_AXI_ARADDR[m:0] | Input | No | Yes | Yes |
| S_AXI_ARLEN[7:0] | Input | No | Yes | Yes |
| S_AXI_ARSIZE[2:0] | Input | No | Yes | Yes |
| S_AXI_ARBURST[1:0] | Input | No | Yes | Yes |

*Table 2-13:* **AXI4 Read Address Channel FIFO Interface Ports *(Cont'd)***

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| S_AXI_ARLOCK[2:0] | Input | No | Yes | Yes |
| S_AXI_ARCACHE[4:0] | Input | No | Yes | Yes |
| S_AXI_ARPROT[3:0] | Input | No | Yes | Yes |
| S_AXI_ARQOS[3:0] | Input | No | Yes | Yes |
| S_AXI_ARREGION[3:0] | Input | No | Yes | Yes |
| S_AXI_ARUSER[m:0] | Input | Yes | Yes | Yes |
| **AXI4 Interface Read Address Channel: Handshake Signals for FIFO Write Interface** | | | | |
| S_AXI_ARVALID | Input | No | Yes | Yes |
| S_AXI_ARREADY | Output | No | Yes | Yes |
| **AXI4 Interface, Read Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| M_AXI_ARID[m:0] | Output | No | Yes | Yes |
| M_AXI_ARADDR[m:0] | Output | No | Yes | Yes |
| M_AXI_ARLEN[7:0] | Output | No | Yes | Yes |
| M_AXI_ARSIZE[2:0] | Output | No | Yes | Yes |
| M_AXI_ARBURST[1:0] | Output | No | Yes | Yes |
| M_AXI_ARLOCK[2:0] | Output | No | Yes | Yes |
| M_AXI_ARCACHE[4:0] | Output | No | Yes | Yes |
| M_AXI_ARPROT[3:0] | Output | No | Yes | Yes |
| M_AXI_ARQOS[3:0] | Output | No | Yes | Yes |
| M_AXI_ARREGION[3:0] | Output | No | Yes | Yes |
| M_AXI_ARUSER[m:0] | Output | Yes | Yes | Yes |
| **AXI4 Interface Read Address Channel: Handshake Signals for FIFO Read Interface** | | | | |
| M_AXI_ARVALID | Output | No | Yes | Yes |
| M_AXI_ARREADY | Input | No | Yes | Yes |
| **AXI4 Read Address Channel FIFO: Optional Sideband Signals** | | | | |
| AXI_AR_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_AR_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_AR_INJECTSBITERR | Input | Yes | Yes | Yes |
| AXI_AR_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_AR_SBITERR | Output | Yes | Yes | Yes |
| AXI_AR_DBITERR | Output | Yes | Yes | Yes |
| AXI_AR_OVERFLOW | Output | Yes | Yes | Yes |

*Table 2-13:* **AXI4 Read Address Channel FIFO Interface Ports *(Cont'd)***

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| AXI_AR_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_AR_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_AR_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_AR_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_AR_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_AR_PROG_EMPTY | Output | Yes | Yes | Yes |

*Table 2-14:* **AXI4 Read Data Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Common Clock | Independent Clocks |
| **AXI4 Interface Read Data Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| S_AXI_RID[m:0] | Output | No | Yes | Yes |
| S_AXI_RDATA[m-1:0] | Output | No | Yes | Yes |
| S_AXI_RRESP[1:0] | Output | No | Yes | Yes |
| S_AXI_RLAST | Output | No | Yes | Yes |
| S_AXI_RUSER[m:0] | Output | Yes | Yes | Yes |
| **AXI4 Interface Read Data Channel: Handshake Signals for FIFO Read Interface** | | | | |
| S_AXI_RVALID | Output | No | Yes | Yes |
| S_AXI_RREADY | Input | No | Yes | Yes |
| **AXI4 Interface Read Data Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |
| M_AXI_RID[m:0] | Input | No | Yes | Yes |
| M_AXI_RDATA[m-1:0] | Input | No | Yes | Yes |
| M_AXI_ RRESP[1:0] | Input | No | Yes | Yes |
| M_AXI_RLAST | Input | No | Yes | Yes |
| M_AXI_RUSER[m:0] | Input | Yes | Yes | Yes |
| **AXI4 Interface, Read Data Channel: Handshake Signals for FIFO Read Interface** | | | | |
| M_AXI_RVALID | Input | No | Yes | Yes |
| M_AXI_RREADY | Output | No | Yes | Yes |
| **AXI4 Read Data Channel FIFO: Optional Sideband Signals** | | | | |
| AXI_R_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_R_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |

*Table 2-14:* **AXI4 Read Data Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Common Clock | Independent Clocks |
| AXI_R_INJECTSBITERR | Input | Yes | Yes | Yes |
| AXI_R_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_R_SBITERR | Output | Yes | Yes | Yes |
| AXI_R_DBITERR | Output | Yes | Yes | Yes |
| AXI_R_OVERFLOW | Output | Yes | Yes | Yes |
| AXI_R_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_R_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_R_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_R_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_R_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_R_PROG_EMPTY | Output | Yes | Yes | Yes |

## AXI4-Lite FIFO Interface Ports

### Write Channels

*Table 2-15:* **AXI4-Lite Write Address Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4-Lite Interface Write Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |
| S_AXI_AWADDR[m:0] | Input | No | Yes | Yes |
| S_AXI_AWPROT[3:0] | Input | No | Yes | Yes |
| **AXI4-Lite Interface Write Address Channel: Handshake Signals for FIFO Write Interface** | | | | |
| S_AXI_AWVALID | Input | No | Yes | Yes |
| S_AXI_AWREADY | Output | No | Yes | Yes |
| **AXI4-Lite Interface Write Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| M_AXI_AWADDR[m:0] | Output | No | Yes | Yes |
| M_AXI_AWPROT[3:0] | Output | No | Yes | Yes |
| **AXI4-Lite Interface Write Address Channel: Handshake Signals for FIFO Read Interface** | | | | |
| M_AXI_AWVALID | Output | No | Yes | Yes |
| M_AXI_AWREADY | Input | No | Yes | Yes |

*Table 2-15:* **AXI4-Lite Write Address Channel FIFO Interface Ports *(Cont'd)***

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4-Lite Write Address Channel FIFO: Optional Sideband Signals** | | | | |
| AXI_AW_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_AW_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_AW_INJECTSBITERR | Input | Yes | Yes | Yes |
| AXI_AW_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_AW_SBITERR | Output | Yes | Yes | Yes |
| AXI_AW_DBITERR | Output | Yes | Yes | Yes |
| AXI_AW_OVERFLOW | Output | Yes | Yes | Yes |
| AXI_AW_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_AW_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_AW_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_AW_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_AW_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_AW_PROG_EMPTY | Output | Yes | Yes | Yes |

*Table 2-16:* **AXI4-Lite Write Data Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4-Lite Interface Write Data Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |
| S_AXI_WDATA[m-1:0] | Input | No | Yes | Yes |
| S_AXI_WSTRB[m/8-1:0] | Input | No | Yes | Yes |
| **AXI4-Lite Interface Write Data Channel: Handshake Signals for FIFO Write Interface** | | | | |
| S_AXI_WVALID | Input | No | Yes | Yes |
| S_AXI_WREADY | Output | No | Yes | Yes |
| **AXI4-Lite Interface Write Data Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| M_AXI_WDATA[m-1:0] | Output | No | Yes | Yes |
| M_AXI_WSTRB[m/8-1:0] | Output | No | Yes | Yes |
| **AXI4-Lite Interface Write Data Channel: Handshake Signals for FIFO Read Interface** | | | | |
| M_AXI_WVALID | Output | No | Yes | Yes |
| M_AXI_WREADY | Input | No | Yes | Yes |
| **AXI4-Lite Write Data Channel FIFO: Optional Sideband Signals** | | | | |

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

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| AXI_W_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_W_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_W_INJECTSBITERR | Input | Yes | Yes | Yes |
| AXI_W_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_W_SBITERR | Output | Yes | Yes | Yes |
| AXI_W_DBITERR | Output | Yes | Yes | Yes |
| AXI_W_OVERFLOW | Output | Yes | Yes | Yes |
| AXI_W_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_W_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_W_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_W_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_W_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_W_PROG_EMPTY | Output | Yes | Yes | Yes |

*Table 2-17:* **AXI4-Lite Write Response Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4-Lite Interface Write Response Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| S_AXI_BRESP[1:0] | Output | No | Yes | Yes |
| **AXI4-Lite Interface Write Response Channel: Handshake Signals for FIFO Read Interface** | | | | |
| S_AXI_BVALID | Output | No | Yes | Yes |
| S_AXI_BREADY | Input | No | Yes | Yes |
| **AXI4-Lite Interface Write Response Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |
| M_AXI_BRESP[1:0] | Input | No | Yes | Yes |
| **AXI4-Lite Interface Write Response Channel: Handshake Signals for FIFO Write Interface** | | | | |
| M_AXI_BVALID | Input | No | Yes | Yes |
| M_AXI_BREADY | Output | No | Yes | Yes |
| **AXI4-Lite Write Response Channel FIFO: Optional Sideband Signals** | | | | |
| AXI_B_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_B_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_B_INJECTSBITERR | Input | Yes | Yes | Yes |

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

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| AXI_B_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_B_SBITERR | Output | Yes | Yes | Yes |
| AXI_B_DBITERR | Output | Yes | Yes | Yes |
| AXI_B_OVERFLOW | Output | Yes | Yes | Yes |
| AXI_B_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_B_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_B_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_B_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_B_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_B_PROG_EMPTY | Output | Yes | Yes | Yes |

## Read Channels

*Table 2-18:* **AXI4-Lite Read Address Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4-Lite Interface Read Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |
| S_AXI_ARADDR[m:0] | Input | No | Yes | Yes |
| S_AXI_ARPROT[3:0] | Input | No | Yes | Yes |
| **AXI4-Lite Interface Read Address Channel: Handshake Signals for FIFO Write Interface** | | | | |
| S_AXI_ARVALID | Input | No | Yes | Yes |
| S_AXI_ARREADY | Output | No | Yes | Yes |
| **AXI4-Lite Interface Read Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| M_AXI_ARADDR[m:0] | Output | No | Yes | Yes |
| M_AXI_ARPROT[3:0] | Output | No | Yes | Yes |
| **AXI4-Lite Interface Read Address Channel: Handshake Signals for FIFO Read Interface** | | | | |
| M_AXI_ARVALID | Output | No | Yes | Yes |
| M_AXI_ARREADY | Input | No | Yes | Yes |
| **AXI4-Lite Read Address Channel FIFO: Optional Sideband Signals** | | | | |
| AXI_AR_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_AR_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |

*Table 2-18:* **AXI4-Lite Read Address Channel FIFO Interface Ports *(Cont'd)***

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| AXI_AR_INJECTSBITERR | Input | Yes | Yes | Yes |
| AXI_AR_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_AR_SBITERR | Output | Yes | Yes | Yes |
| AXI_AR_DBITERR | Output | Yes | Yes | Yes |
| AXI_AR_OVERFLOW | Output | Yes | Yes | Yes |
| AXI_AR_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_AR_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_AR_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_AR_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_AR_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_AR_PROG_EMPTY | Output | Yes | Yes | Yes |

*Table 2-19:* **AXI4-Lite Read Data Channel FIFO Interface Ports**

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| **AXI4-Lite Interface Read Data Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus** | | | | |
| S_AXI_RDATA[m-1:0] | Output | No | Yes | Yes |
| S_AXI_RRESP[1:0] | Output | No | Yes | Yes |
| **AXI4-Lite Interface Read Data Channel: Handshake Signals for FIFO Read Interface** | | | | |
| S_AXI_RVALID | Output | No | Yes | Yes |
| S_AXI_RREADY | Input | No | Yes | Yes |
| **AXI4-Lite Interface Read Data Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus** | | | | |
| M_AXI_RDATA[m-1:0] | Input | No | Yes | Yes |
| M_AXI_ RRESP[1:0] | Input | No | Yes | Yes |
| **AXI4-Lite Interface Read Data Channel: Handshake Signals for FIFO Write Interface** | | | | |
| M_AXI_RVALID | Input | No | Yes | Yes |
| M_AXI_RREADY | Output | No | Yes | Yes |
| **AXI4-Lite Read Data Channel FIFO: Optional Sideband Signals** | | | | |
| AXI_R_PROG_FULL_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_R_PROG_EMPTY_THRESH[m:0] | Input | Yes | Yes | Yes |
| AXI_R_INJECTSBITERR | Input | Yes | Yes | Yes |

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

| Port Name | Input or Output | Optional Port | Port Available | |
|---|---|---|---|---|
| | | | Independent Clocks | Common Clock |
| AXI_R_INJECTDBITERR | Input | Yes | Yes | Yes |
| AXI_R_SBITERR | Output | Yes | Yes | Yes |
| AXI_R_DBITERR | Output | Yes | Yes | Yes |
| AXI_R_OVERFLOW | Output | Yes | Yes | Yes |
| AXI_R_WR_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_R_UNDERFLOW | Output | Yes | Yes | Yes |
| AXI_R_RD_DATA_COUNT[m:0] | Output | Yes | Yes | No |
| AXI_R_DATA_COUNT[m:0] | Output | Yes | No | Yes |
| AXI_R_PROG_FULL | Output | Yes | Yes | Yes |
| AXI_R_PROG_EMPTY | Output | Yes | Yes | Yes |

# 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.

**IMPORTANT:** *Depending on the configuration of the FIFO core, only a subset of the implementation details provided are applicable. For successful use of a FIFO core, the design guidelines discussed in this chapter must be observed.*

## General Design Guidelines

### Know the Degree of Difficulty

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

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

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

### Understand Signal Pipelining 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 you 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 3-1 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.

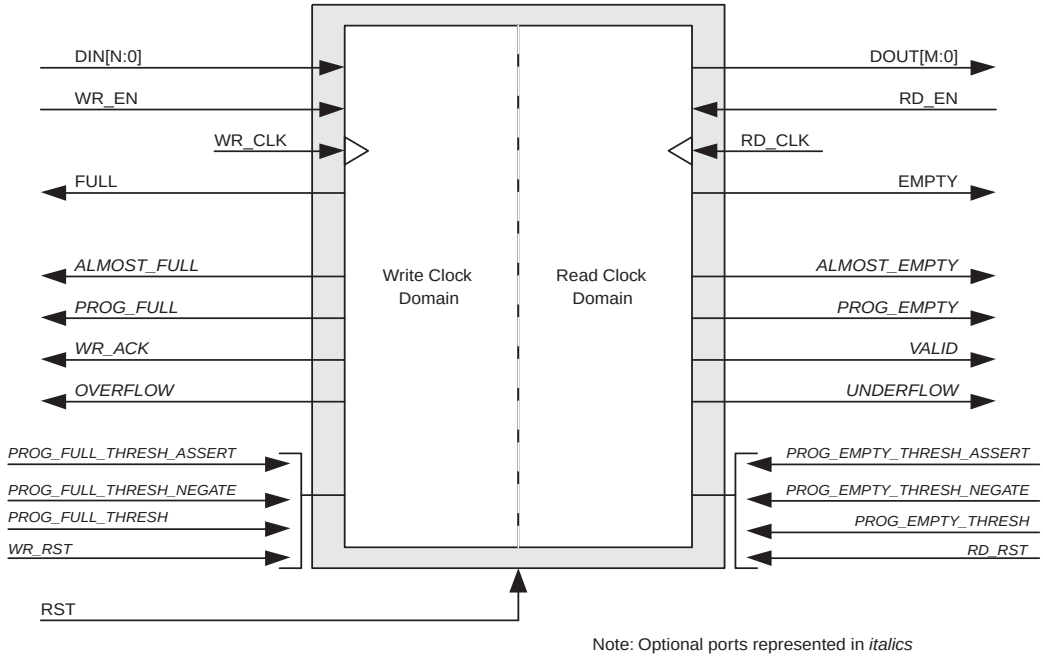


*Figure 3-1:* **FIFO with Independent Clocks: Write and Read Clock Domains**

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 Resets, page 115.

# 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 does 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.

### Example Operation

Figure 3-2 shows a typical write operation. The user asserts WR_EN, causing a write operation to occur on the next rising edge of the WR_CLK. 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 you perform one or more read operations, the FIFO deasserts FULL, and data can successfully be written to the FIFO, as is indicated by the assertion of WR_ACK and deassertion of OVERFLOW.

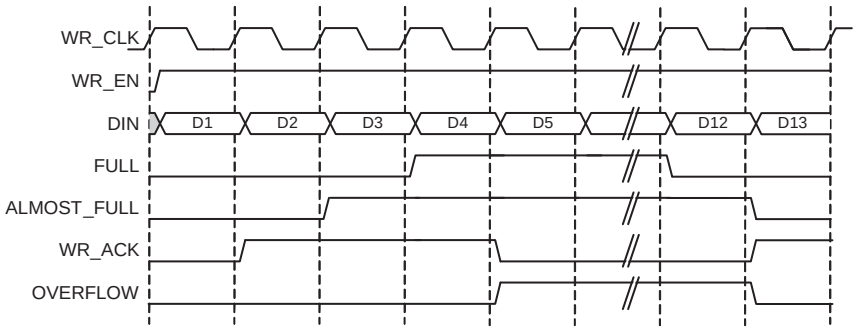


*Figure 3-2:* **Write Operation for a FIFO with Independent Clocks**

## Read Operation

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

### ALMOST_EMPTY and EMPTY Flags

***Note:*** The 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 3-1 details the supported implementations for FWFT.

*Table 3-1:* **Implementation-Specific Support for First-Word Fall-Through**

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

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

to assert RD_EN while EMPTY is asserted, the read request is ignored, VALID is deasserted, and UNDERFLOW is asserted. Once the user performs a write operation, the FIFO deasserts EMPTY, allowing the user to resume valid read operations, as indicated by the assertion of VALID and deassertion of UNDERFLOW.

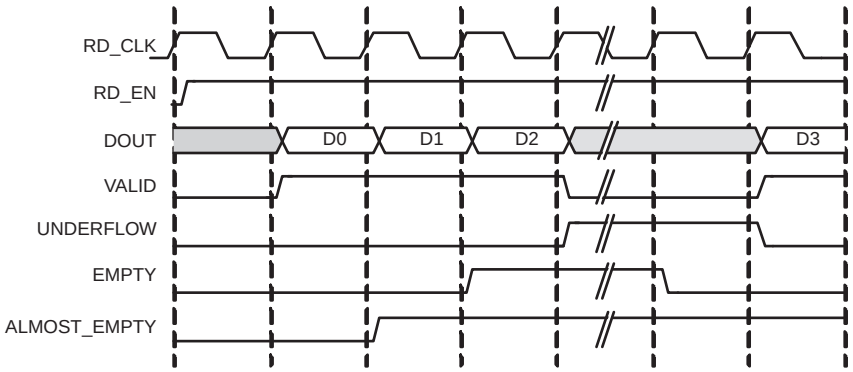


*Figure 3-3:* **Standard Read Operation for a FIFO with Independent Clocks**

### First-Word Fall-Through FIFO Read Operation

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

Figure 3-4 shows a FWFT read access. Initially, the FIFO is not empty, the next available data word is placed on the output bus (DOUT), and VALID is asserted. When you assert 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 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.

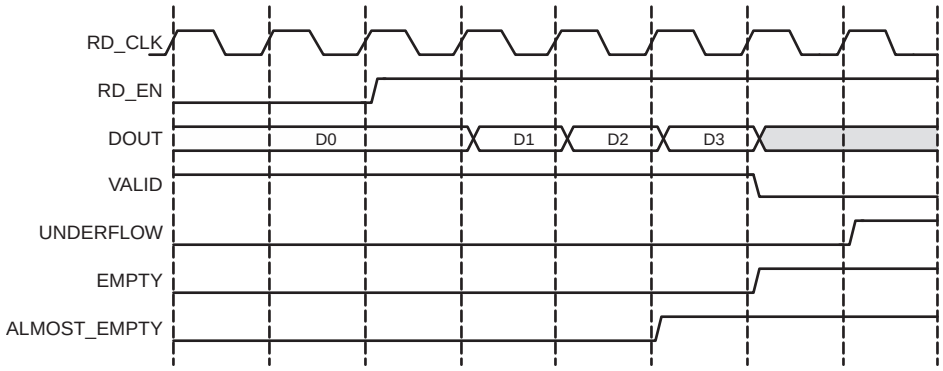


*Figure 3-4:* **FWFT Read Operation for a FIFO with Independent Clocks**

### Common Clock FIFO, Simultaneous Read and Write Operation

Figure 3-5 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.

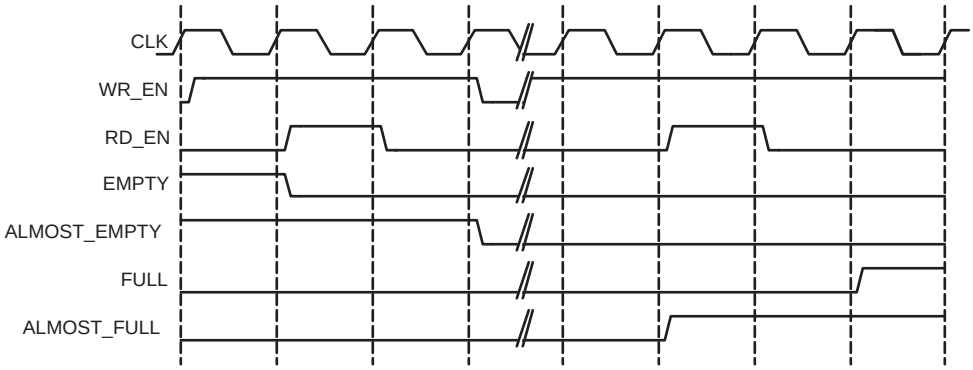


*Figure 3-5:* **Write and Read Operation for a FIFO with Common Clocks**

## 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 FIFO Generator GUI. These flags (configured as active high) are illustrated in Figure 3-6.

## 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 3-6 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.

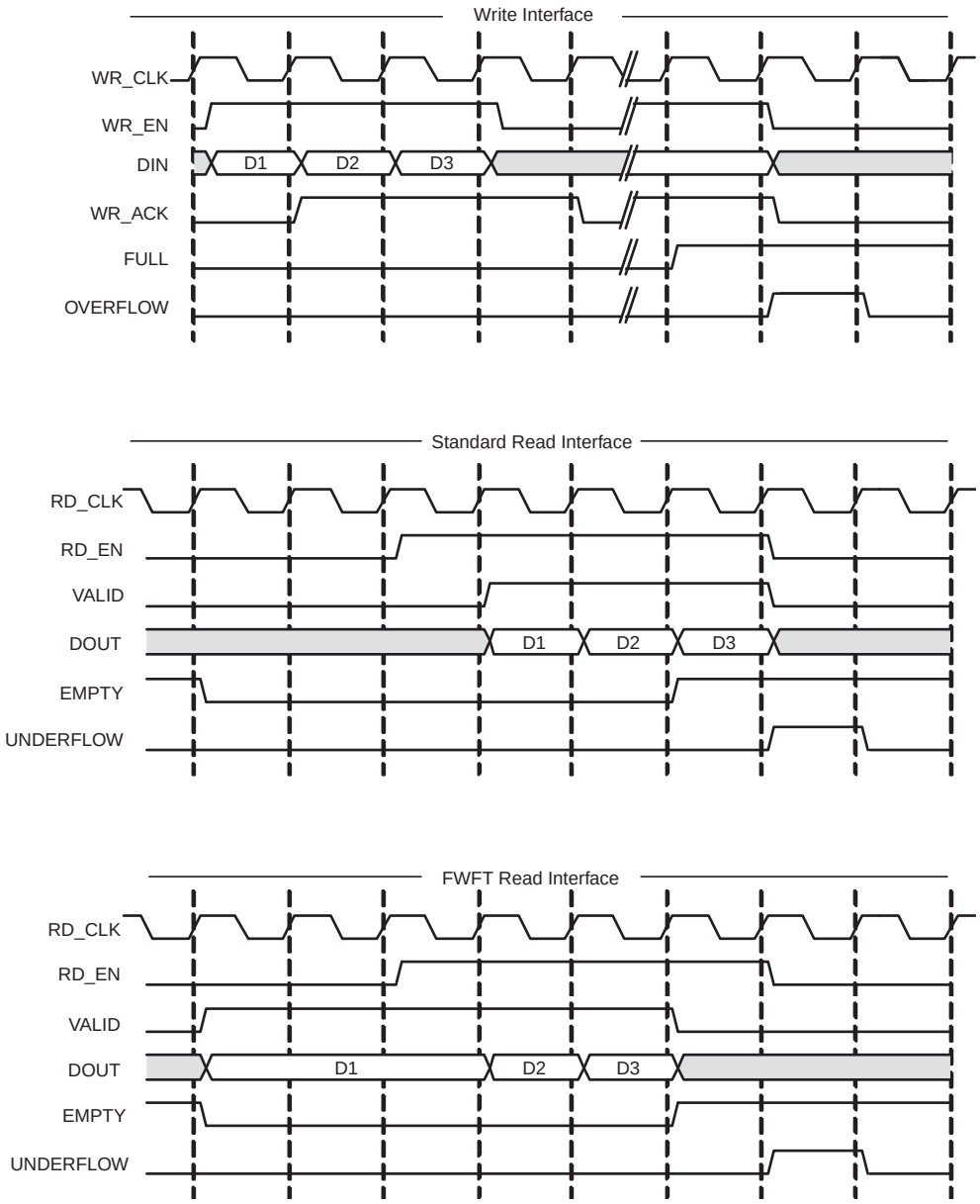


*Figure 3-6:* **Handshaking Signals for a FIFO with Independent Clocks**

## 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 3-7 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.

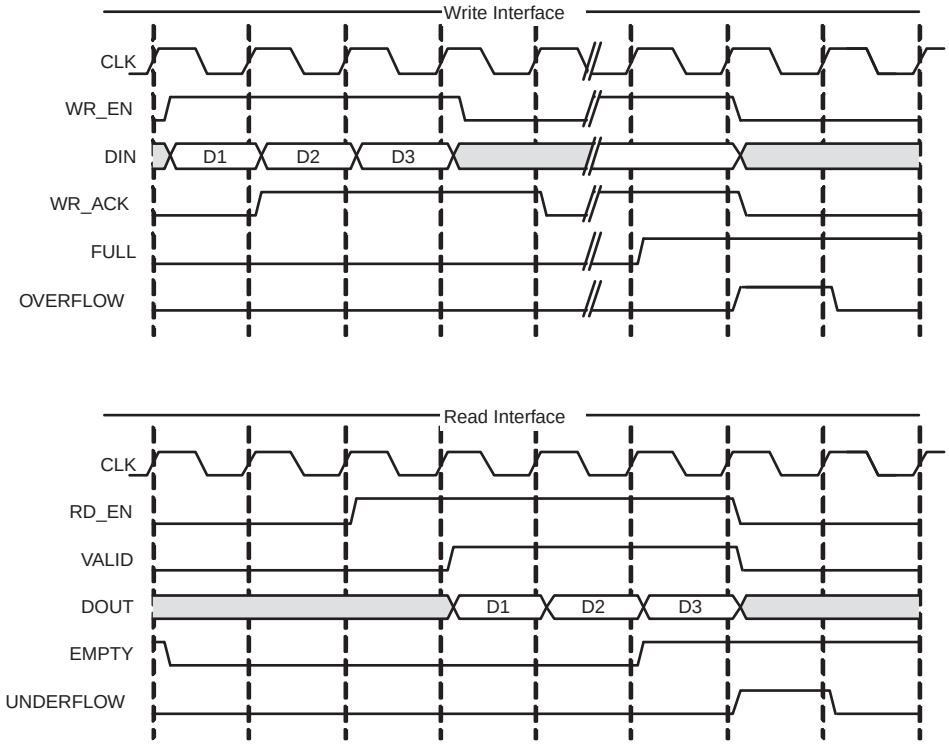


*Figure 3-7:* **Handshaking Signals for a FIFO with Common Clocks**

## Programmable Flags

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

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

For these thresholds, you can set a constant value or choose to have dedicated input ports, enabling the thresholds to change dynamically in circuit. Hysteresis is also optionally supported, by providing unique assert and negate values for each flag. Detailed information about these options are provided below. For information about the latency behavior of the programmable flags, see Latency, page 125.

## 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 FIFO Generator GUI and accessed within the programmable flags window (Figure 9-5, page 201).

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 you 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 FIFO 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 you the flexibility to change the programmable full threshold in-circuit without re-generating the core.

***Note:*** See the FIFO Generator GUI screen for valid ranges for each threshold.

Figure 3-8 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.

**TIP:** *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.*

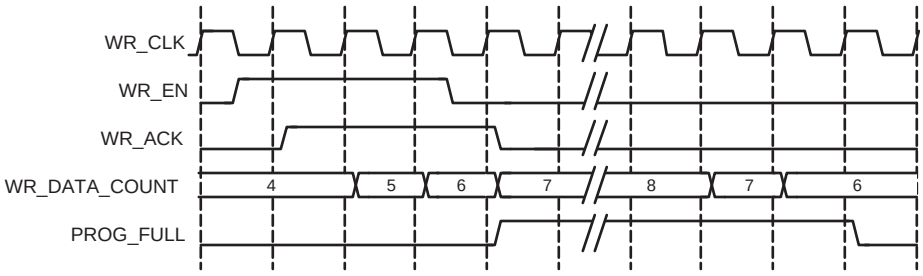


*Figure 3-8:* **Programmable Full Single Threshold: Threshold Set to 7**

### Programmable Full: Assert and Negate Thresholds

This option enables you 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.

**IMPORTANT:** *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 FIFO 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 you 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 FIFO Generator GUI for valid ranges for each threshold.

Figure 3-9 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.

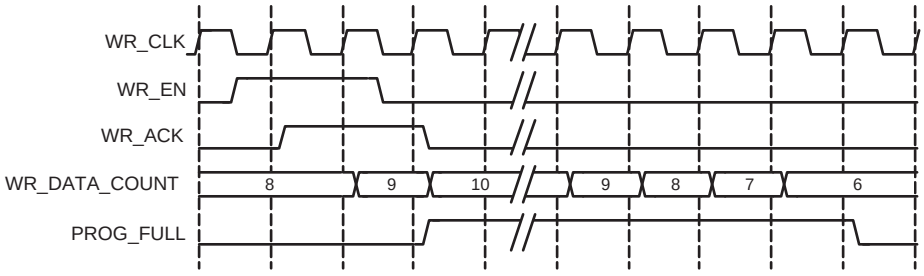


*Figure 3-9:* **Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7**

### 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 (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 FIFO 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 FIFO Generator GUI and accessed within the programmable flags window (Figure 9-5, page 201).

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 FIFO 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 FIFO Generator GUI for valid ranges for each threshold.

Figure 3-10 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.

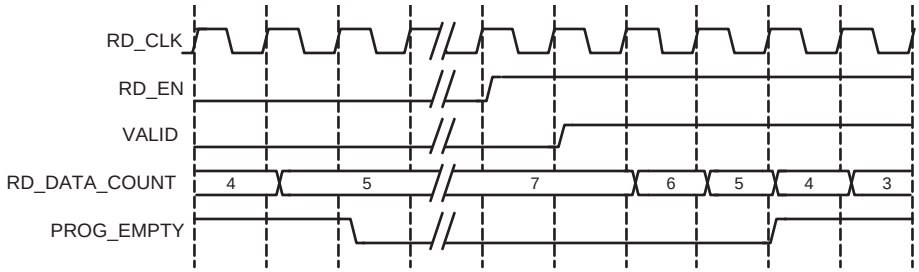


*Figure 3-10:* **Programmable Empty with Single Threshold: Threshold Set to 4**

### Programmable Empty: Assert and Negate Thresholds

This option lets you 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 FIFO 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 you 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 FIFO Generator GUI for valid ranges for each threshold.

Figure 3-11 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.

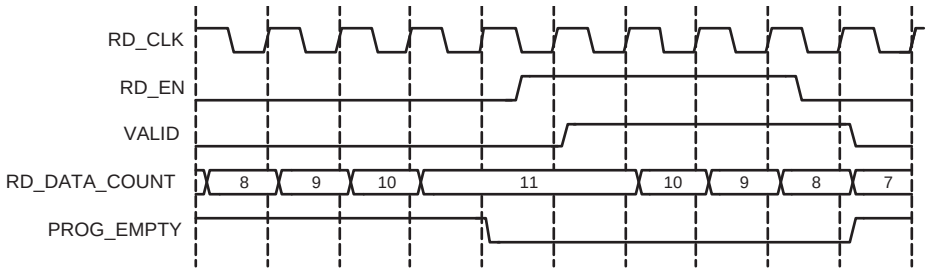


*Figure 3-11:* **Programmable Empty with Assert and Negate Thresholds: Assert Set to 7 and Negate Set to 10**

### Programmable Empty Threshold Range Restrictions

The programmable empty threshold ranges depend on several features that dictate the way the FIFO is implemented, described as follows:

- 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 (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.

**IMPORTANT:** *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 FIFO Generator GUI for valid ranges for each threshold. To avoid unexpected behavior, do not use 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 FIFO Generator GUI. Table 3-2 identifies data count support for each FIFO implementation. For information about the latency behavior of data count flags, see Latency, page 125.

*Table 3-2:* **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. You 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, 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. 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 you never overflow the FIFO. You 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 you 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 FIFO 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 FIFO 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.

**IMPORTANT:** *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 3-3 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 3-17). 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 3-3 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 3-3:* **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 3-12 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 3-12, 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.

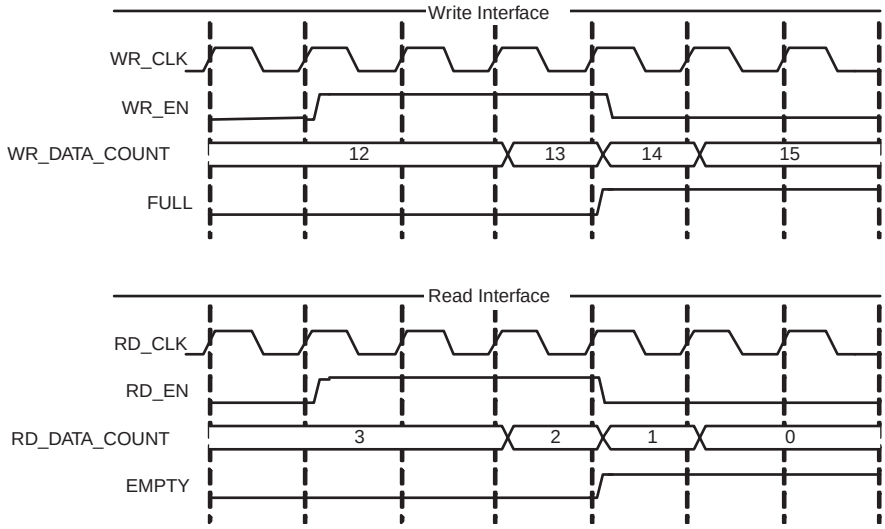


*Figure 3-12:* **Write and Read Data Counts for FIFO with Independent Clocks**

## Non-symmetric Aspect Ratios

Table 3-4 identifies support for non-symmetric aspect ratios.

*Table 3-4:* **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 Vivado IP Catalog. 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, you 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 3-13 and Figure 3-14).

Figure 3-13 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.

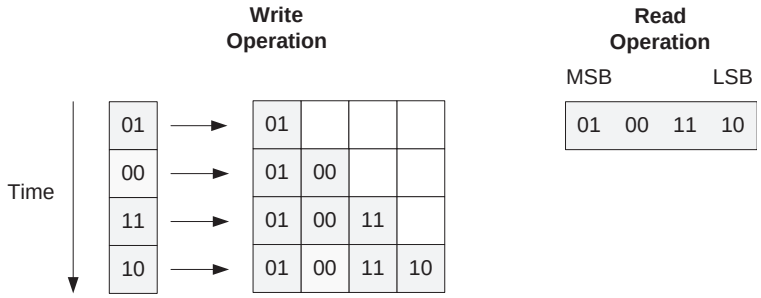


*Figure 3-13:* **1:4 Aspect Ratio: Data Ordering**

Figure 3-14 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.

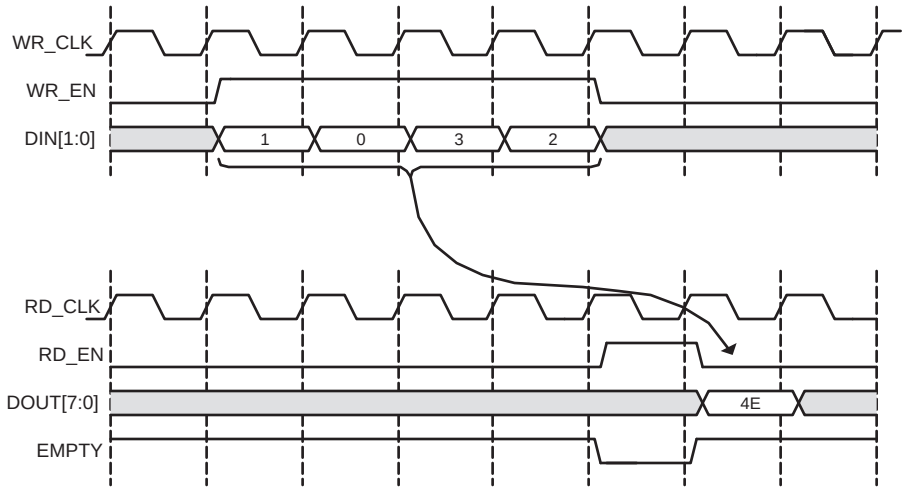


*Figure 3-14:* **1:4 Aspect Ratio: Status Flag Behavior**

Figure 3-15 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.

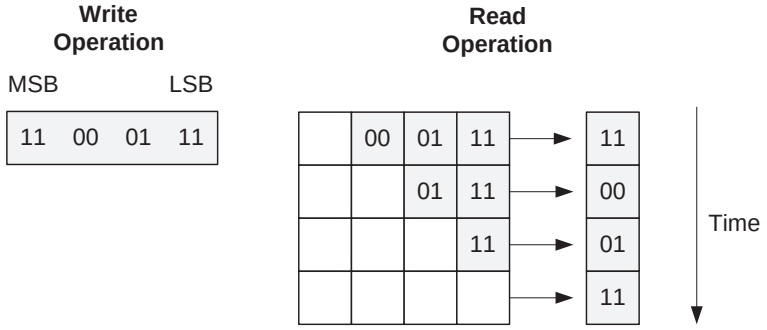


*Figure 3-15:* **4:1 Aspect Ratio: Data Ordering**

Figure 3-16 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.

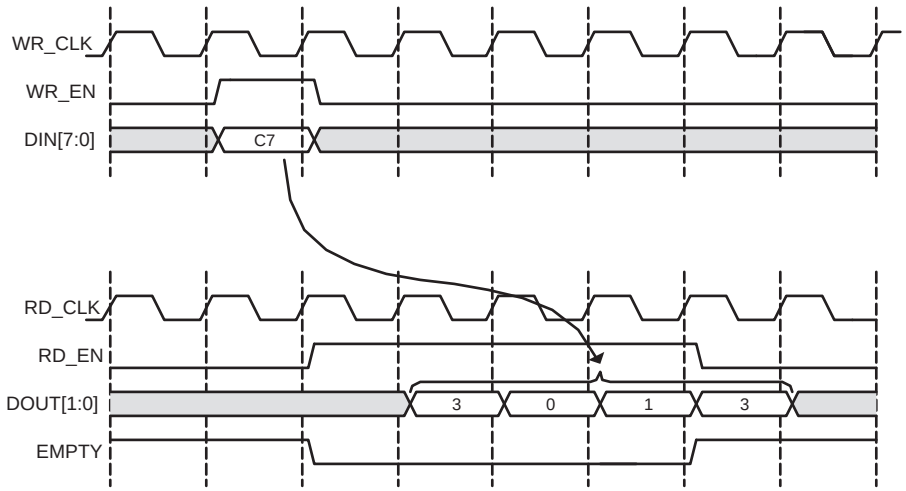


*Figure 3-16:* **4:1 Aspect Ratio: Status Flag Behavior**

## 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, 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, 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 you never overflow 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

The block RAM macros and built-in FIFO macros 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 one clock cycle during a read operation. These additional pipeline registers are always enabled, as illustrated in Figure 3-17.

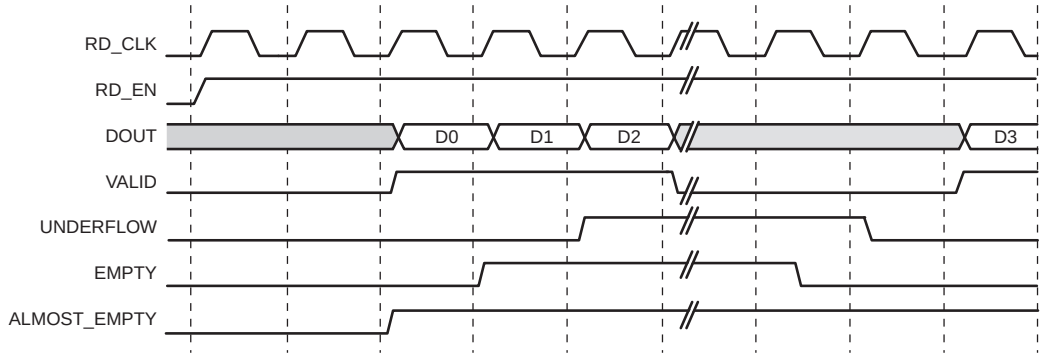


*Figure 3-17:* **Standard Read Operation for a Block RAM or built-in FIFO with Use Embedded Registers Enabled**

## 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, as illustrated in Figure 3-18. For this configuration, the embedded output register feature is only available for FIFOs that use only one FIFO macro in depth.

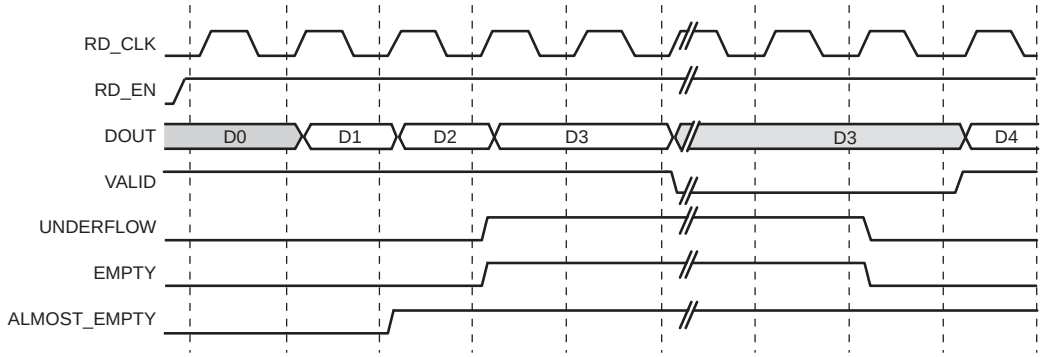


*Figure 3-18:* **FWFT Read Operation for a Synchronous Built-in FIFO with User Embedded Registers Enabled**

When using the embedded output registers with a common clock built-in FIFO, the DOUT reset value feature is supported, as illustrated in Figure 3-19.

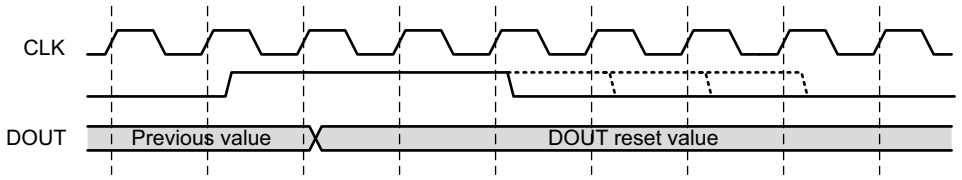


*Figure 3-19:* **DOUT Reset Value Common Clock Built-in FIFO Embedded Register**

## Built-in Error Correction Checking

Built-in ECC is supported for FIFOs configured with independent or common clock block RAM and built-in FIFOs. 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 3-20 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, 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.

**TIP:** *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.*

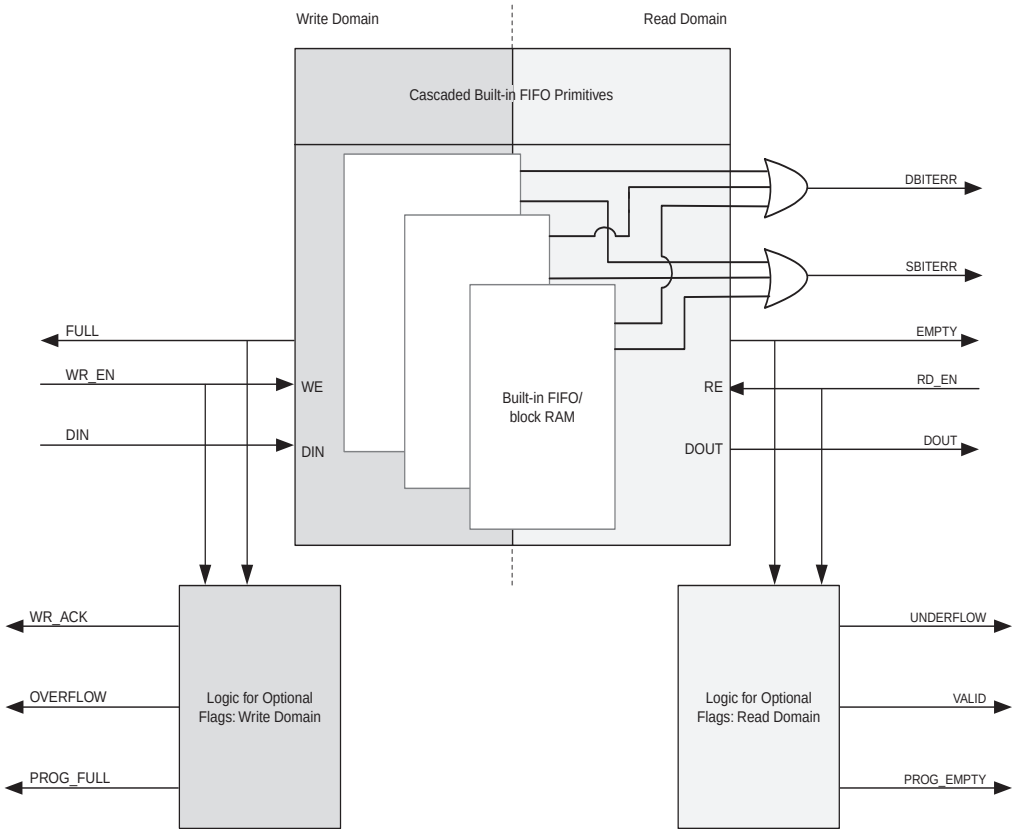


*Figure 3-20:* **SBITERR and DBITERR Outputs in the FIFO Generator Core**

## Built-in Error Injection

Built-in Error Injection is supported for FIFOs configured with independent or common clock block RAM and built-in FIFOs. 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 ECC primitives. If the data width chosen 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. In this case, the accuracy of the `DBITERR` signal cannot be guaranteed. For example, if the data width is set to 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 3-21 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 Resets, page 115 for more details.

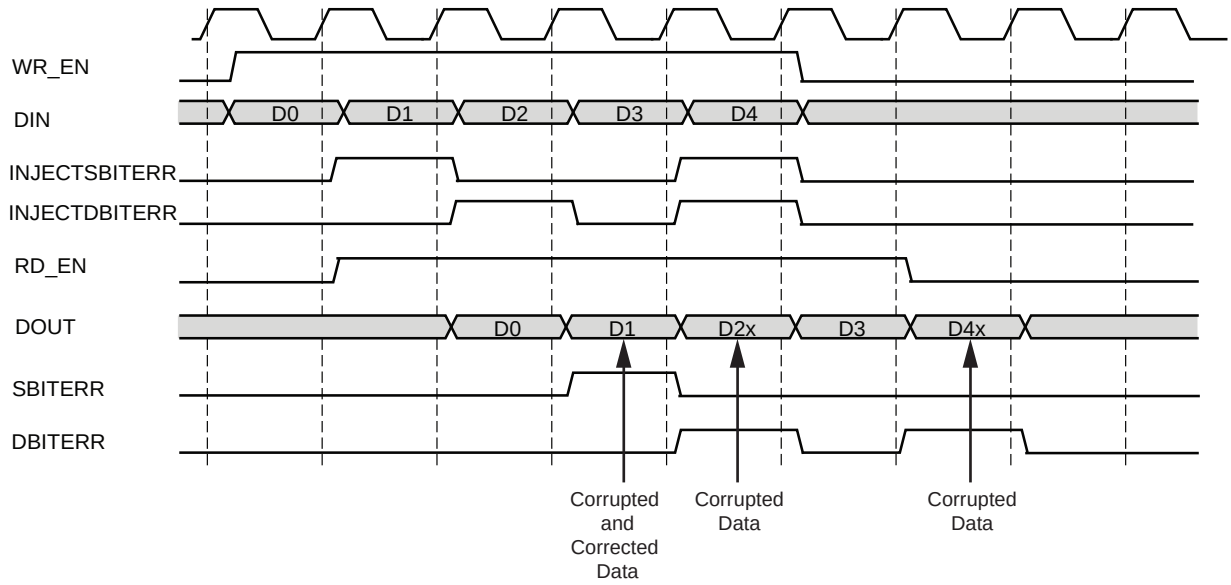


*Figure 3-21:* **Error Injection and Correction**

# Clocking

Each FIFO configuration has a set of allowable features, as defined in Table 1-3, page 14.

## Independent Clocks: Block RAM and Distributed RAM

Figure 3-22 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.

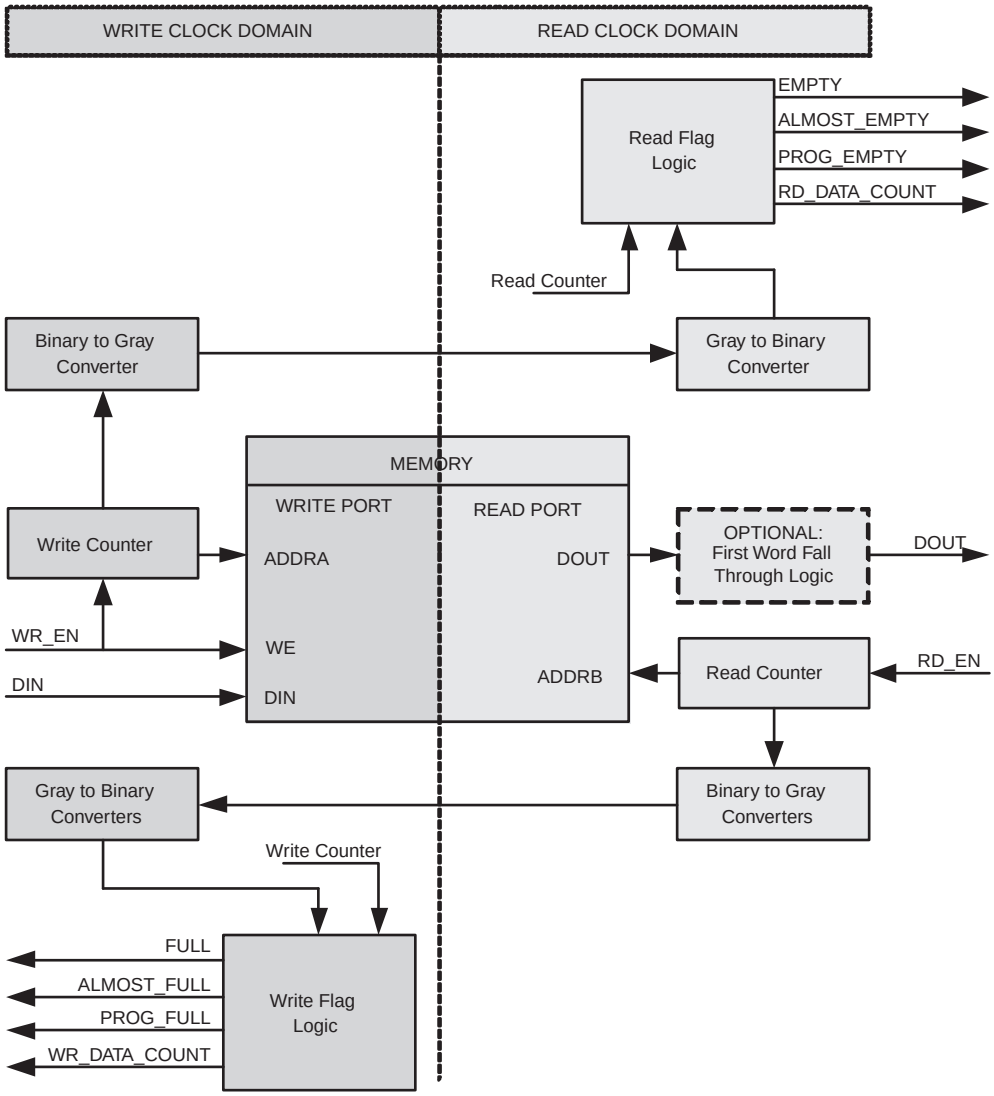


*Figure 3-22:* **Functional Implementation of a FIFO with Independent Clock Domains**

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 3-5 summarizes the FIFO interface signals, which are only valid in their respective clock domains.

*Table 3-5:* **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 |

*Table 3-5:* **Interface Signals and Corresponding Clock Domains *(Cont'd)***

| | |
|---|---|
| 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 83.

## Independent Clocks: Built-in FIFO

Figure 3-23 illustrates the functional implementation of FIFO configured with independent clocks using the built-in FIFO primitive. This design implementation consists of cascaded built-in FIFO primitives and handshaking logic. The number of built-in primitives depends on the FIFO width and depth requested.

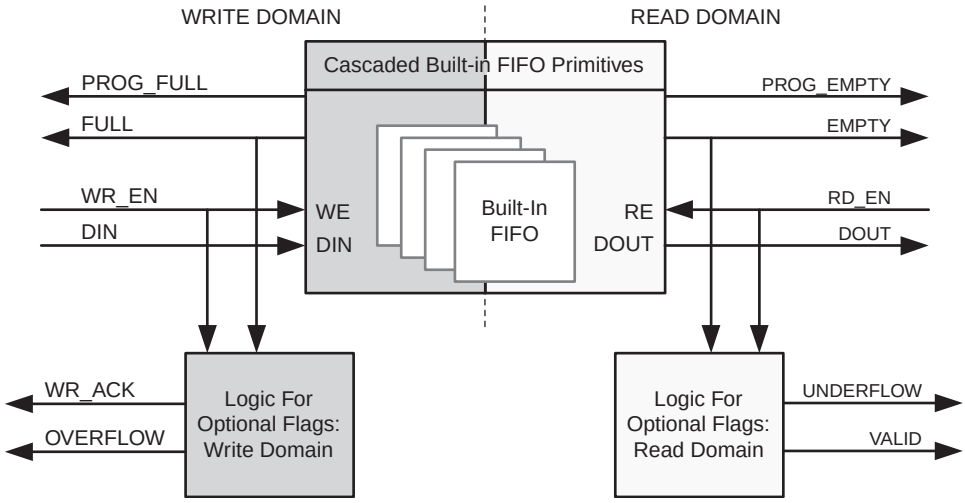


*Figure 3-23:* **Functional Implementation of Built-in FIFO**

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 3-6 summarizes the FIFO interface signals, which are only valid in their respective clock domains.

*Table 3-6:* **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 83.

For 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 108.

***Note:*** When the ECC option is selected, the number of Built-in FIFO primitives in depth and all the output latency will be different. For more information on latency, see Latency, page 125.

For example, if user depth is 4096, user width is 9 and ECC is not selected, then the number of Built-in FIFO primitives in depth is 1. However, if ECC is selected for the same configuration, then the number of Built-in FIFO primitives in depth is 4092/512 = 8.

## 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 112, for more information.

## Common Clock FIFO: Block RAM and Distributed RAM

Figure 3-24 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.

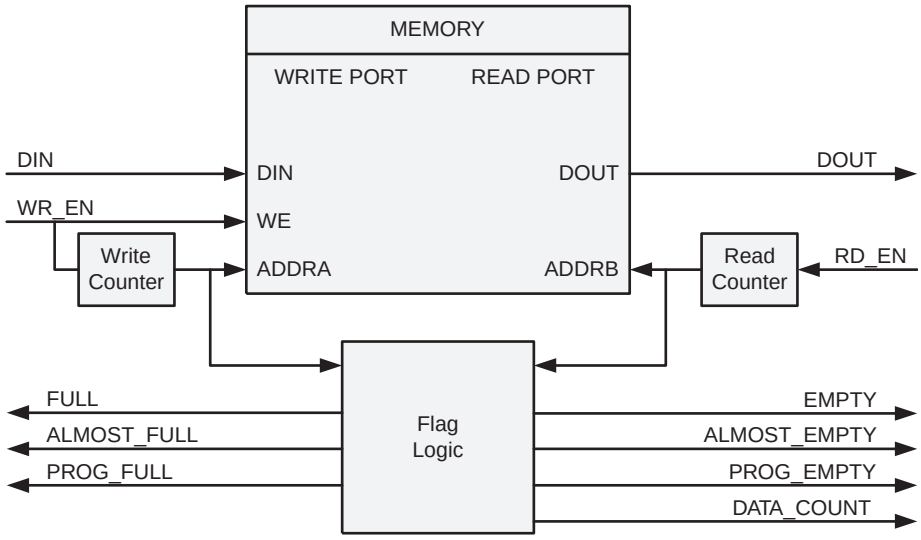


*Figure 3-24:* **Functional Implementation of a Common Clock FIFO using Block RAM or Distributed RAM**

## Common Clock FIFO: Shift Registers

Figure 3-25 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.

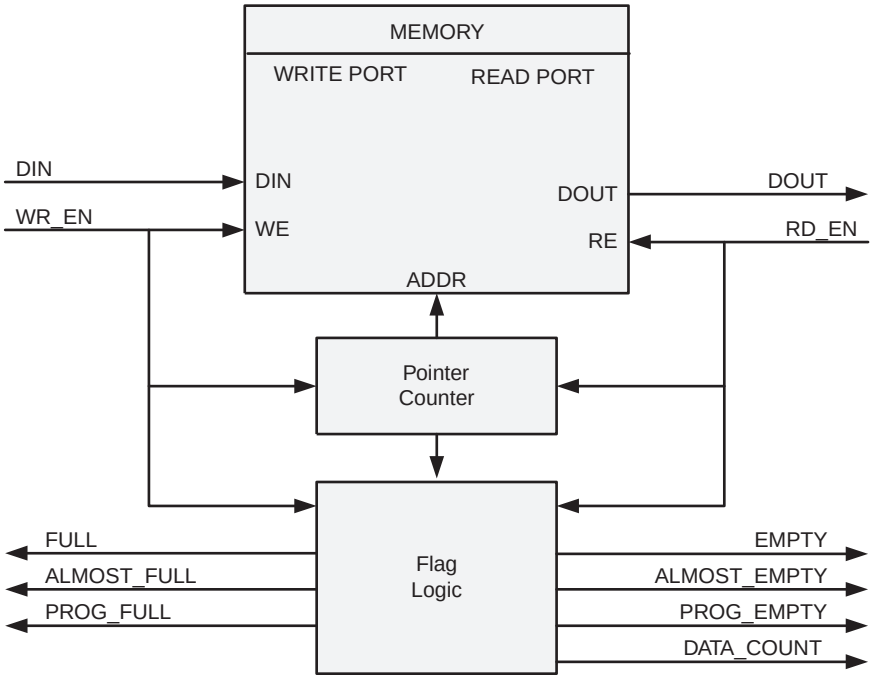


*Figure 3-25:* **Functional Implementation of a Common Clock FIFO using Shift Registers**

# Resets

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 3-7 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.

**IMPORTANT:** *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 3-7:* **Asynchronous Reset Values for Block, Distributed, 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 |

*Table 3-7:* **Asynchronous Reset Values for Block, Distributed, and Shift RAM FIFOs**

| | | | |
|---|---|---|---|
| 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 three clock periods (WR_CLK/CLK) and the FIFO can then accept write operations.

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 3-26 and Figure 3-27 for example behaviors. Note that the power-up values for this configuration are different from the reset state value.

Figure 3-26 shows an example timing diagram for when the reset pulse is one clock cycle.

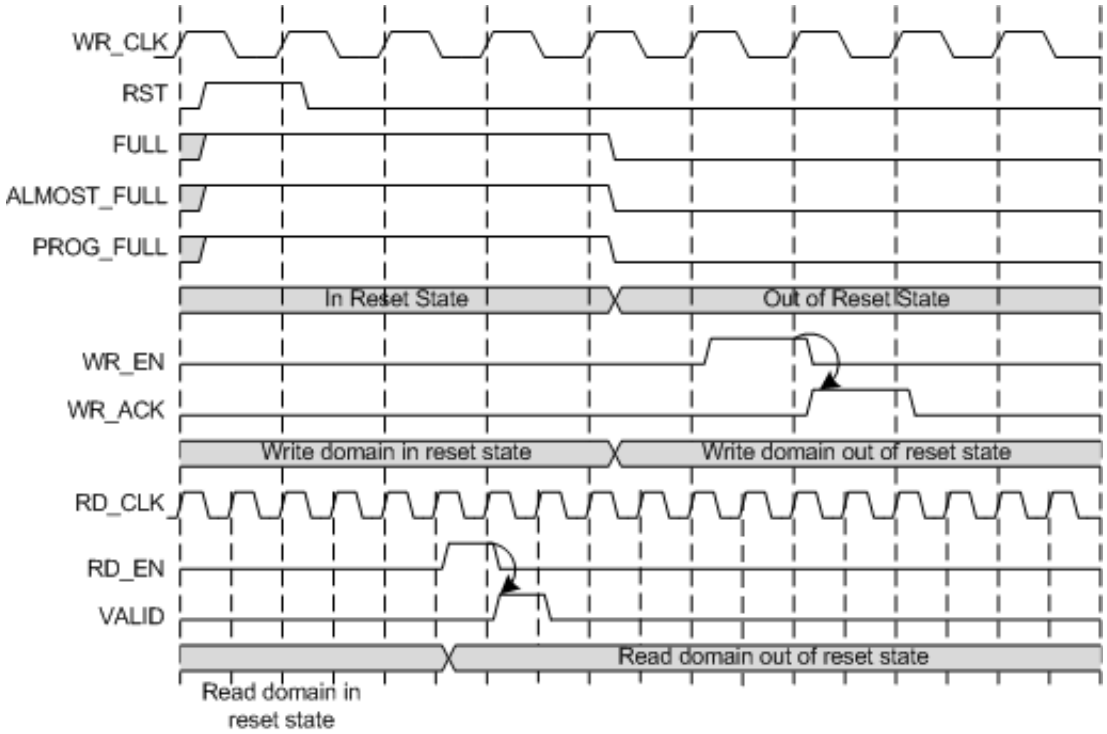


*Figure 3-26:* **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 1 for the Reset Pulse of One Clock**

Figure 3-27 shows an example timing diagram for when the reset pulse is longer than one clock cycle.

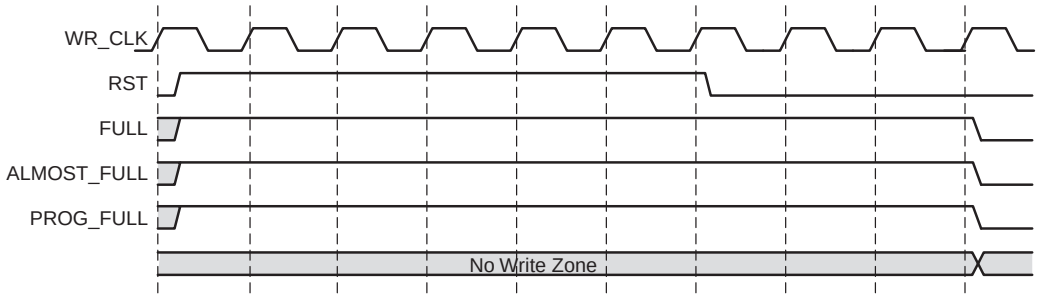


*Figure 3-27:* **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 1 for the Reset Pulse of More Than One Clock**

### Full Flags Reset Value of 0

In this configuration, the FIFO requires a minimum asynchronous reset pulse of one 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 3-28 for example behavior.

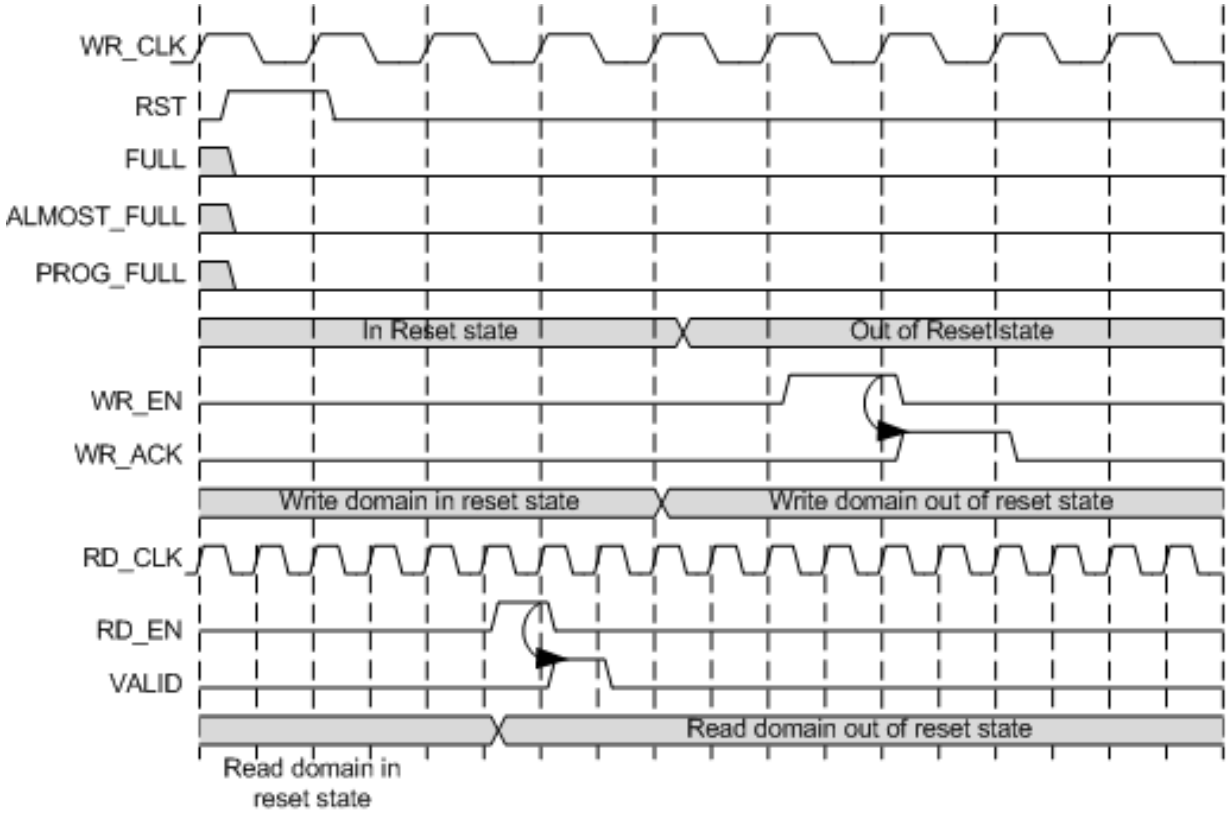


*Figure 3-28:* **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 0**

### Common/Independent Clock: Built-in

Table 3-7 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 common clock Built-In FIFOs with the embedded register option selected. The built-In FIFOs require an asynchronous reset pulse of at least five read and write clock cycles. To be consistent across all built-in FIFO configurations, it is recommended to give an asynchronous reset pulse of at least 5 read and write clock cycles for built-in FIFOs. However, the FIFO Generator core has a built-in mechanism ensuring the reset pulse is high for five read and write clock cycles for all 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 3-29 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 3-8:* **Asynchronous 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) |
| PROG_FULL | 0 | 0 |
| PROG_EMPTY | 1 | 1 |

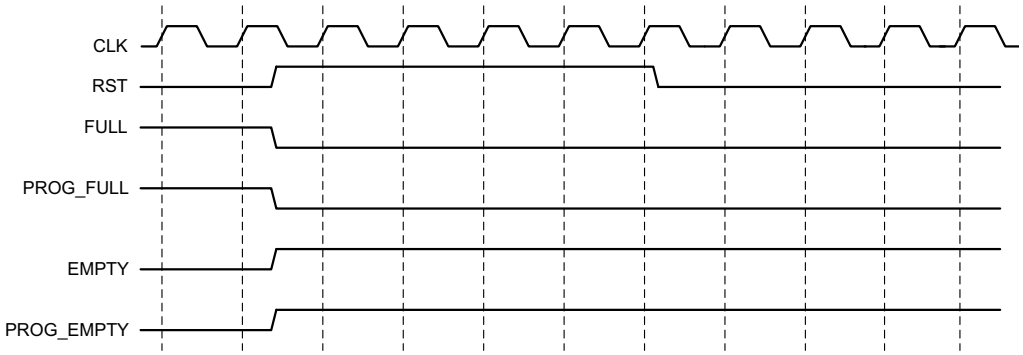


*Figure 3-29:* **Built-in FIFO, Asynchronous Reset Behavior**

## 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 3-32 illustrates the flags following the release of SRST.

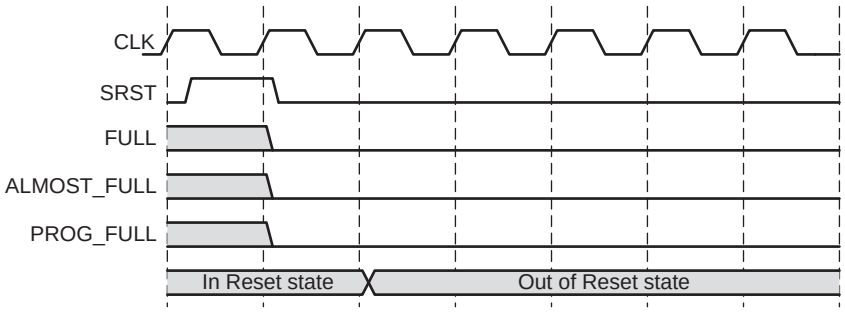


*Figure 3-32:* **Synchronous Reset: FIFO with a Common Clock**

### 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, do not 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 3-33 and Figure 3-34 detail the resets.

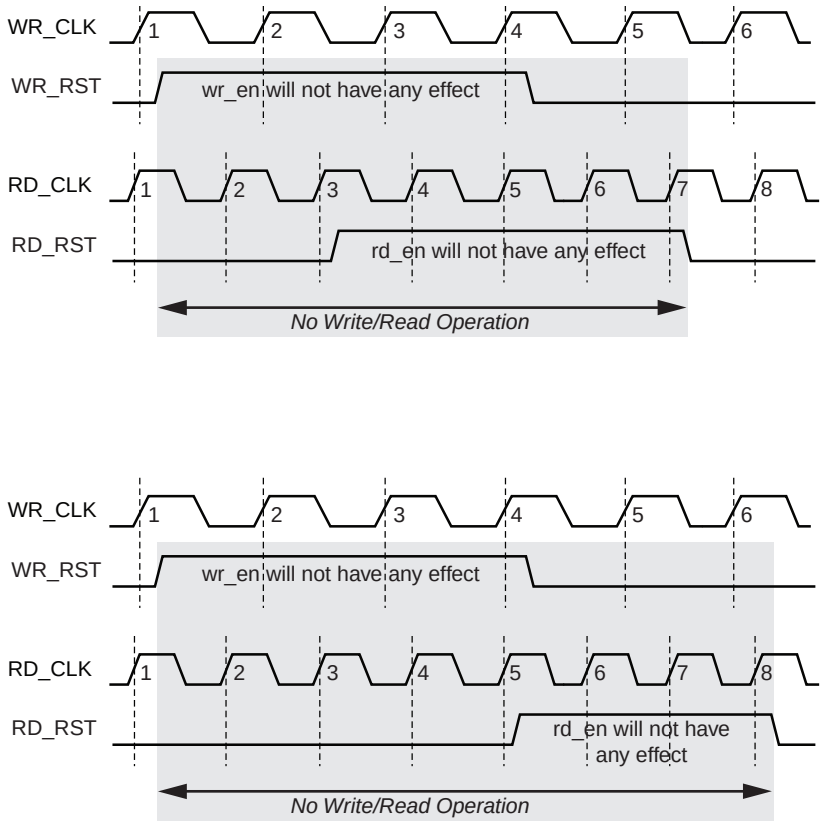


*Figure 3-33:* **Synchronous Reset: FIFO with Independent Clock - WR_RST then RD_RST**

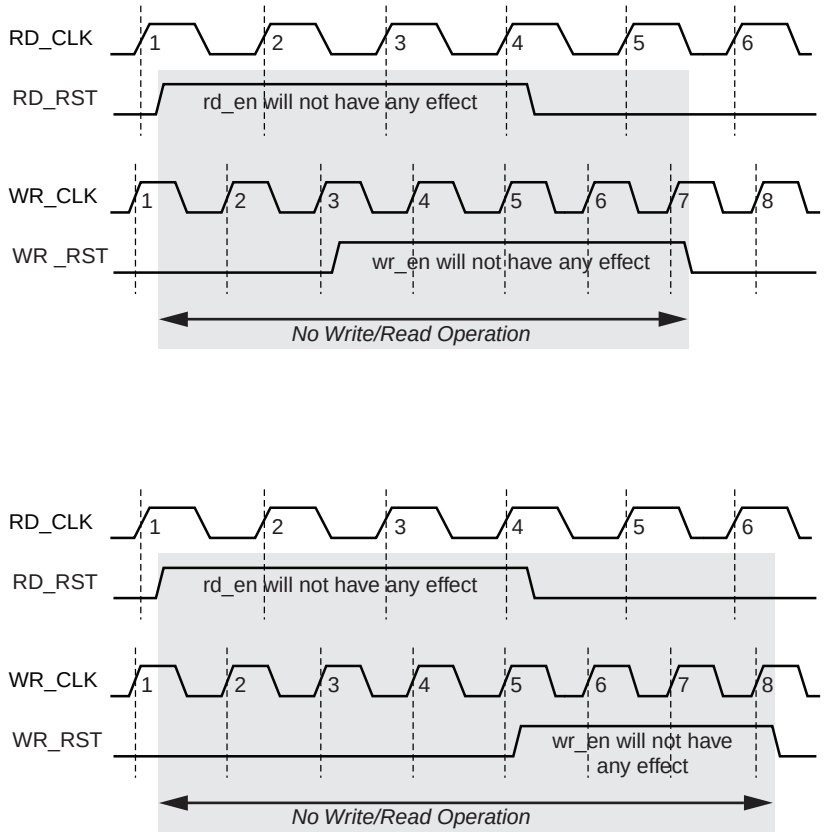


*Figure 3-34:* **Synchronous Reset: FIFO with Independent Clock - RD_RST then WR_RST**

Table 3-9 defines the values of the output ports during power-up and the reset state. If you do 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 3-9:* **Synchronous 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 3-9:* **Synchronous 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.

## Built-In FIFOs

The actual FIFO depths 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)

# 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 3-35 in which WR_ACK is getting updated in which WR_EN is high.

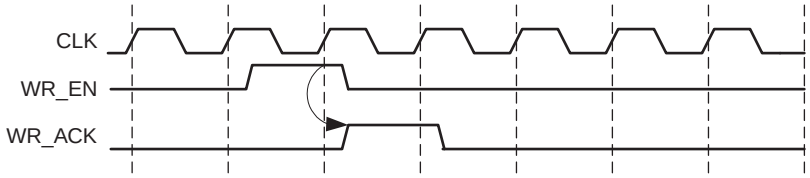


*Figure 3-35:* **Latency 0 Timing**

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

Table 3-10 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 3-10:* **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 3-11 defines the read port flags update latency due to a read operation.

*Table 3-11:* **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 |

*Table 3-11:* **Non-Built-in FIFOs, Common Clock and Standard Read Mode Implementations: Read Port Flags Update Latency Due to Read Operation**

| | |
|---|---|
| UNDERFLOW | 0 |
| DATA_COUNT | 0 |

Table 3-12 defines the write port flags update latency due to a read operation.

*Table 3-12:* **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 3-13 defines the read port flags update latency due to a write operation.

*Table 3-13:* **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 3-14 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 3-14:* **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 3-15 defines the read port flags update latency due to a read operation.

*Table 3-15:* **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 3-16 defines the write port flags update latency due to a read operation.

*Table 3-16:* **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 3-17 defines the read port flags update latency due to a write operation.

*Table 3-17:* **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 3-18 defines the write port flags update latency due to a write operation.

*Table 3-18:* **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 3-19 defines the read port flags update latency due to a read operation.

*Table 3-19:* **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 3-20 defines the write port flags update latency due to a read operation.

*Table 3-20:* **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 3-21 defines the read port flags update latency due to a write operation.

*Table 3-21:* **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 3-22 defines the write port flags update latency due to a write operation.

*Table 3-22:* **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 3-23 defines the read port flags update latency due to a read operation.

*Table 3-23:* **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 3-24 defines the write port flags update latency due to a read operation.

*Table 3-24:* **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 3-25 defines the read port flags update latency due to a write operation.

*Table 3-25:* **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. For ECC, the primitive depth is 512. The term "Built-in FIFOs" refers to the hard FIFO macros of FPGAs.

For more details for the write and read port flags update latency for a single primitive, see UG473, *7 Series FPGAs Memory Resources User Guide*.

Table 3-26 defines the write port flags update latency due to a write operation.

*Table 3-26:* **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 3-27 defines the read port flags update latency due to a read operation.

*Table 3-27:* **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 3-28 defines the write port flags update latency due to a read operation.

*Table 3-28:* **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 3-29 defines the read port flags update latency due to a write operation.

*Table 3-29:* **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 |

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. For ECC, the primitive depth is 512. The term "Built-in FIFOs" refers to the hard FIFO macros of FPGAs.

For more details for the write and read port flags update latency for a single primitive, see UG473, *7 Series FPGAs Memory Resources User Guide*.

Table 3-30 defines the write port flags update latency due to a write operation.

*Table 3-30:* **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 3-31 defines the read port flags update latency due to a read operation.

*Table 3-31:* **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 3-32 defines the write port flags update latency due to a read operation.

*Table 3-32:* **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 |
| WR_ACK[a] | N/A |
| OVERFLOW | N/A |

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

Table 3-33 defines the read port flags update latency due to a write operation.

*Table 3-33:* **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. For ECC, the primitive depth is 512. `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 FPGAs.

For more details for the write and read port flags update latency for a single primitive, see UG473, *7 Series FPGAs Memory Resources User Guide*.

Table 3-34 defines the write port flags update latency due to a write operation.

*Table 3-34:* **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 3-35 defines the read port flags update latency due to a read operation.

*Table 3-35:* **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 3-36 defines the write port flags update latency due to a read operation.

*Table 3-36:* **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

| Signals | Latency |
|---|---|
| FULL[a] | L1[b] RD_CLK + (N-1)*L2[b] faster_clk + L3[b] WR_CLK |
| PROG_FULL[a] | L4[b] RD_CLK + (N-1)*(L2[b] -1) faster_clk + L5[b] WR_CLK |
| WR_ACK[c] | N/A |
| OVERFLOW[c] | N/A |

a. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
b. L1 = 1, L2 = 4, L3 = 3, L4 = 1 and L5 = 4
c. Write handshaking signals are only impacted by a Write operation.

Table 3-37 defines the read port flags update latency due to a write operation.

*Table 3-37:* **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

| Signals | Latency |
|---|---|
| EMPTY[a] | L1[b] WR_CLK + (N-1)*L2[b] faster_clk + L3[b] RD_CLK |
| PROG_EMPTY[a] | L4[b] WR_CLK + (N-1)*(L5[b] -1) faster_clk + L6[b] RD_CLK |
| VALID[c] | N/A |
| UNDERFLOW[c] | N/A |

a. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
b. L1 = 1, L2 = 4, L3 = 4, L4 = 1, L5 = 5 and L6 = 4.
c. 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. For ECC, the primitive depth is 512. `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 FPGAs.

For more details for the write and read port flags update latency for a single primitive, see UG473, *7 Series FPGAs Memory Resources User Guide*.

Table 3-38 defines the write port flags update latency due to a write operation.

*Table 3-38:* **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 3-39 defines the read port flags update latency due to a read operation.

*Table 3-39:* **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 3-40 defines the write port flags update latency due to a read operation.

*Table 3-40:* **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

| Signals | Latency |
|---|---|
| FULL[a] | $L1^{b}$ RD_CLK + (N-1)*$L2^{b}$ faster_clk + $L3^{b}$ WR_CLK |
| PROG_FULL[a] | $L4^{b}$ RD_CLK + (N-1)*($L2^{b}$ -1) faster_clk + $L5^{b}$ WR_CLK |
| WR_ACK[c] | N/A |
| OVERFLOW[c] | N/A |

a. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
b. L1 = 1, L2 = 4, L3 = 3, L4 = 1 and L5 = 4.
c. Write handshaking signals are only impacted by a Write operation.

Table 3-41 defines the read port flags update latency due to a write operation.

*Table 3-41:* **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

| Signals | Latency |
|---|---|
| EMPTY[a] | $L1^{b}$ WR_CLK + (N-1)*$L2^{b}$ faster_clk + $L3^{b}$ RD_CLK |
| PROG_EMPTY[a] | $L4^{b}$ WR_CLK + (N-1)*($L5^{b}$ -1) faster_clk + $L6^{b}$ RD_CLK |

*Table 3-41:* **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation *(Cont'd)***

| Signals | Latency |
|---|---|
| VALID[c] | N/A |
| UNDERFLOW[c] | N/A |

a. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
b. L1 = 1, L2 = 5, L3 = 4, L4 = 1, L5 = 5 and L6 = 4.
c. Read handshaking signals are only impacted by a Read operation.

# 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 Resets in Chapter 3 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 3, "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 3 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 3)

**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 3-20, page 129.

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 3-21, page 130.

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 3 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.

The FIFO Generator core provides an IP-level constraint that applies a MAXDELAY constraint to avoid setup and hold violations on the cross-clock domain logic. In addition to the

IP-level constraint, the FIFO Generator also provides an example design constraint that applies a FALSE_PATH on the reset path.

# Customizing and Generating the Native Core

This chapter includes information about using Xilinx tools to customize and generate the FIFO Generator for Native FIFO Interfaces in the Vivado Design Suite.

## GUI

The Native FIFO Interface GUI includes seven configuration screens.

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

### Interface Type and Implementation Options

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

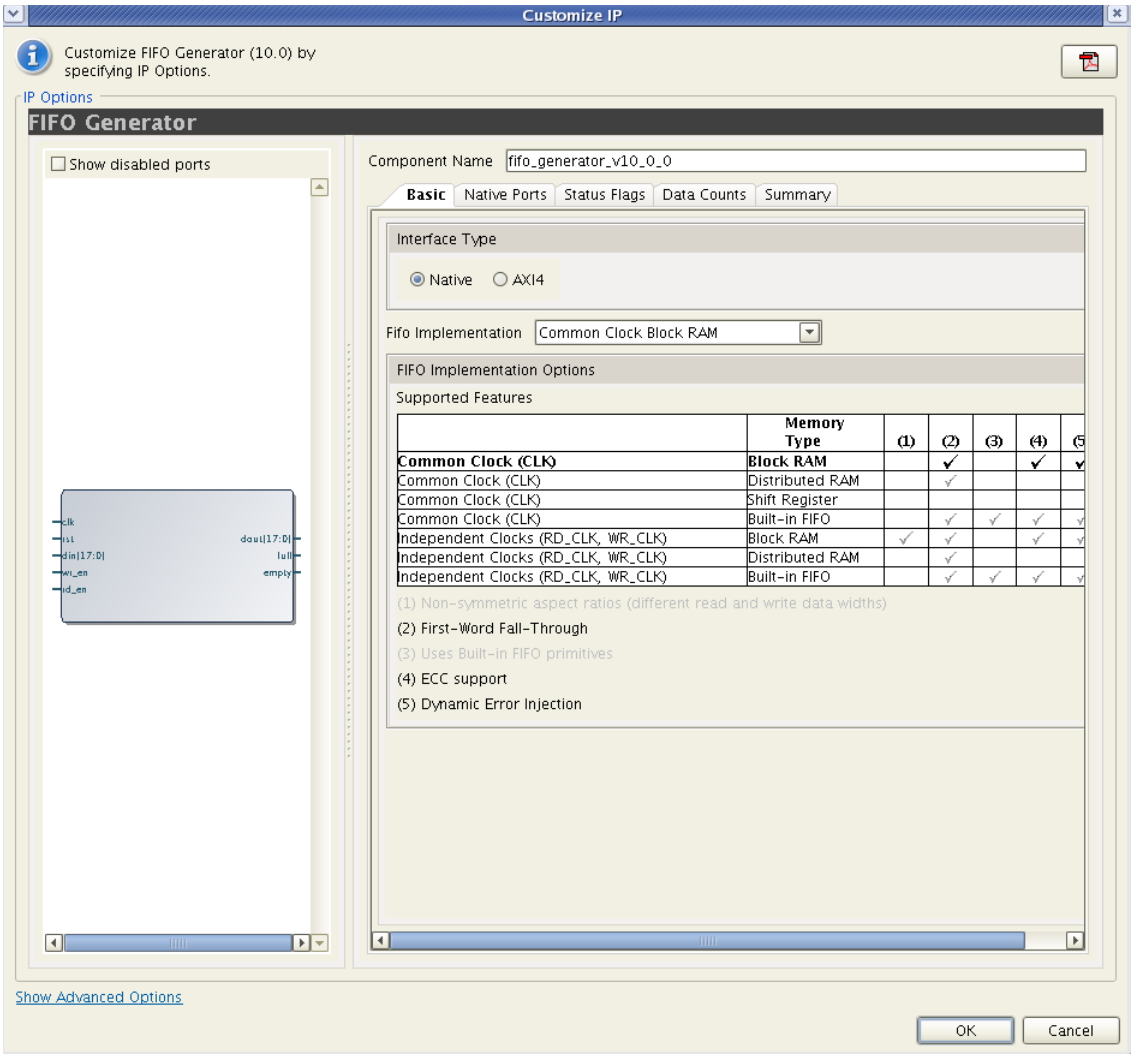


*Figure 5-1:* **Interface Type and FIFO Implementation Screen**

- **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.
- **Common Clock (CLK), Block RAM**: See Common Clock FIFO: Block RAM and Distributed RAM in Chapter 3 for details. This implementation optionally supports first-word-fall-through (selectable in the second GUI screen, Figure 5-3).

- **Common Clock (CLK), Distributed RAM**: For details, see Common Clock FIFO: Block RAM and Distributed RAM in Chapter 3. This implementation optionally supports first-word-fall-through (selectable in the second GUI screen, Figure 5-3).
- **Common Clock (CLK), Shift Register**: For details, see Common Clock FIFO: Shift Registers in Chapter 3.
- **Common Clock (CLK), Built-in FIFO**: For details, see Common Clock: Built-in FIFO in Chapter 3. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, Figure 5-3).
- **Independent Clocks (RD_CLK, WR_CLK), Block RAM**: For details, see Independent Clocks: Block RAM and Distributed RAM in Chapter 3. This implementation optionally supports asymmetric read/write ports and first-word fall-through (selectable in the second GUI screen, Figure 5-3).
- **Independent Clocks (RD_CLK, WR_CLK), Distributed RAM**: For more information, see Independent Clocks: Block RAM and Distributed RAM in Chapter 3. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, Figure 5-3).
- **Independent Clocks (RD_CLK, WR_CLK), Built-in FIFO**: For more information, see Independent Clocks: Built-in FIFO in Chapter 3. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, Figure 5-3).

## Performance Options, Data Port Parameters, ECC and Initialization

This screen provides performance options (Read Mode), data port parameters, ECC and initialization options for the core.

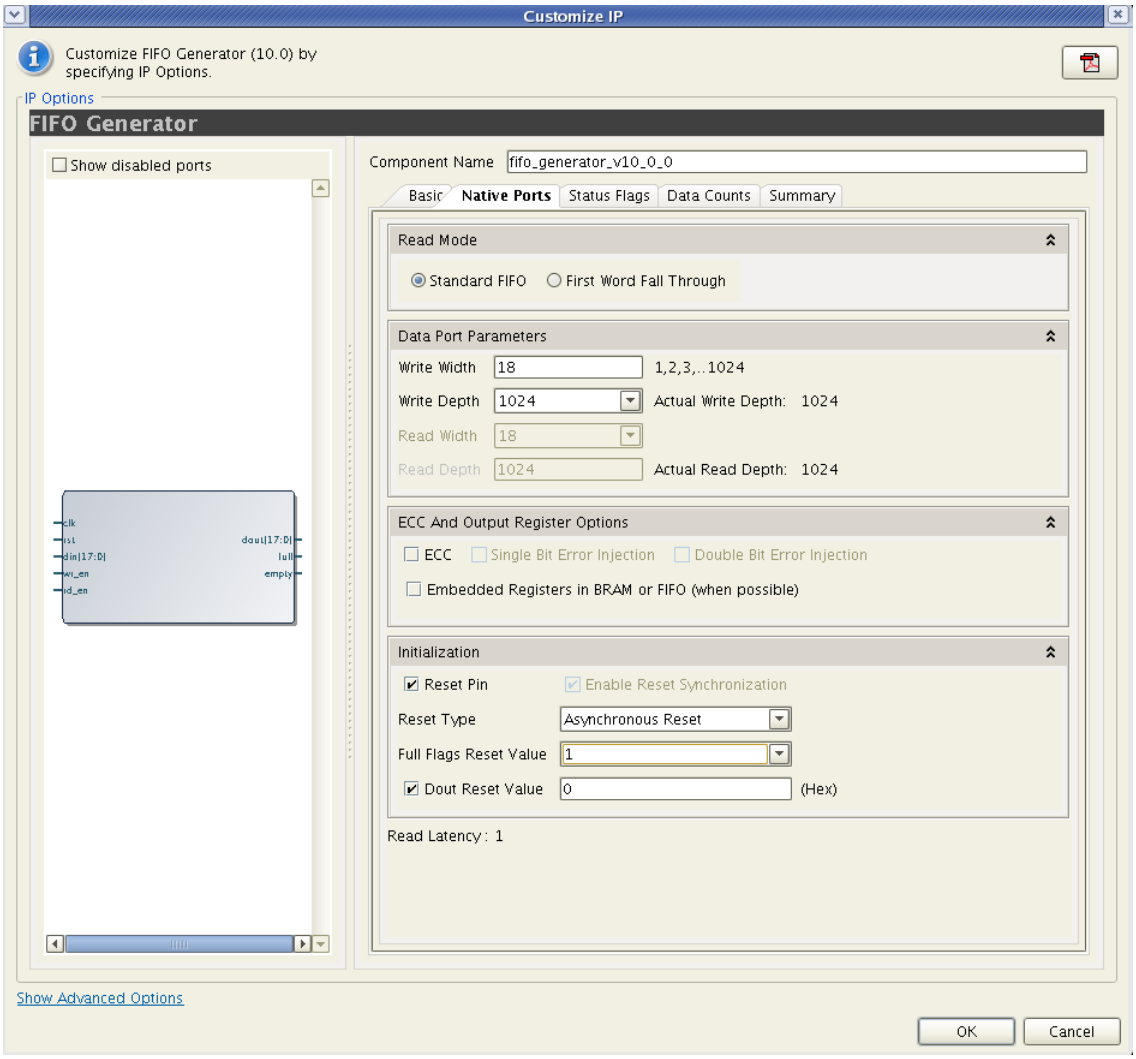


*Figure 5-2:* **Read Mode, Data Port Parameters, ECC and Initialization Screen**

- **Read Mode**: Available only when block RAM or distributed RAM FIFOs are selected.
- **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 in Chapter 3.
- **Data Port Parameters**
  - Write Width: The valid range of write data width is 1 to 1024.
  - Write Depth: The valid range of write depth is 16 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 in Chapter 3.

- Read Depth: Automatically calculated based on write width, write depth, and read width.

- **ECC and Output Register Options**
  - Error Correction Checking (ECC): The Error Correction Checking (ECC) feature enables built-in error correction in the 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.
  - Single Bit Error Injection: Available 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 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.
  - Use Embedded Registers in Block RAM or FIFO: The block RAM macros 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. 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 in Chapter 3.
- **Initialization**
  - Reset Pin: For FIFOs implemented with block RAM or distributed RAM, a reset pin is not required, and the input pin is optional.
  - Reset Type:
    - 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 Resets in Chapter 3for details.
    - Asynchronous Reset: Optional selection for a common-clock FIFO implemented using distributed or block RAM.
    - Synchronous Reset: Optional selection for 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 for all implementations using block RAM, distributed RAM, shift register or 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.

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

## Optional, Handshaking, and Programmable Flags

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

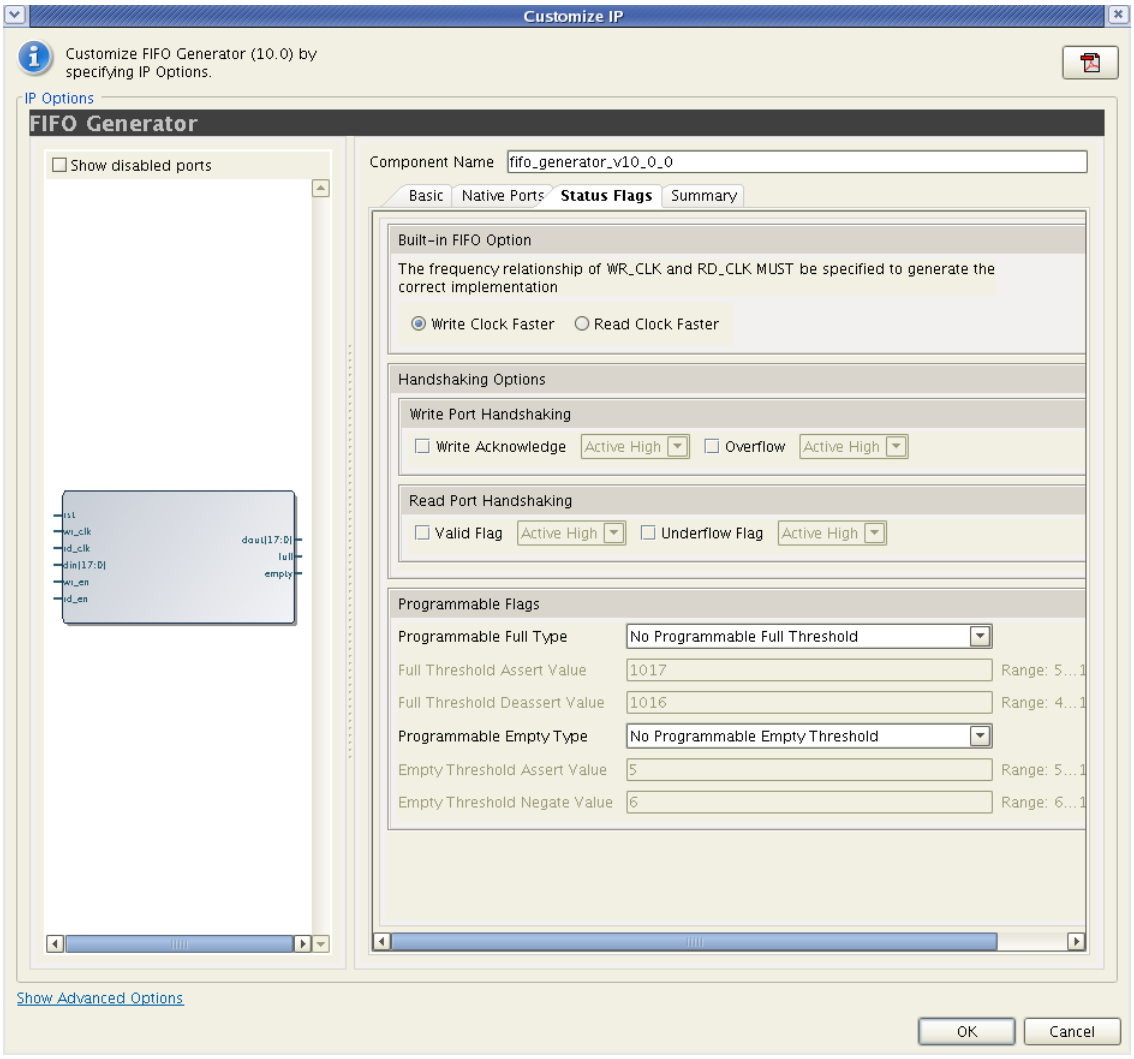


*Figure 5-3:* **Status Flags Screen**

- **Built-in FIFO Options**

- Read/Write Clock Faster: The Read Clock Faster and Write Clock Faster 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.

**IMPORTANT:** *It is critical that Read Clock Faster and Write Clock Faster information is accurate. If this information is inaccurate, it can result in a sub-optimal solution with incorrect core behavior.*

- **Optional Flags**: See Latency in Chapter 3 for the latency of the Almost Full/Empty flags due to write/read operation.
  - Almost Full Flag: Available in all FIFO implementations except those using built-in FIFOs. Generates an output port that indicates the FIFO is almost full (only one more word can be written).
  - Almost Empty Flag: Available in all FIFO implementations except in those using built-in FIFOs. Generates an output port that indicates the FIFO is almost empty (only one more word can be read).
- **Handshaking Options**: See Latency in Chapter 3 for the latency of the handshaking flags due to write/read operation.
  - 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).
- **Programmable Flags**: See Latency in Chapter 3 for the latency of the programmable flags due to write/read operation.
  - 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.

***Note:*** Valid range of values shown in the GUI is the actual values even though they are grayed out for some selection.

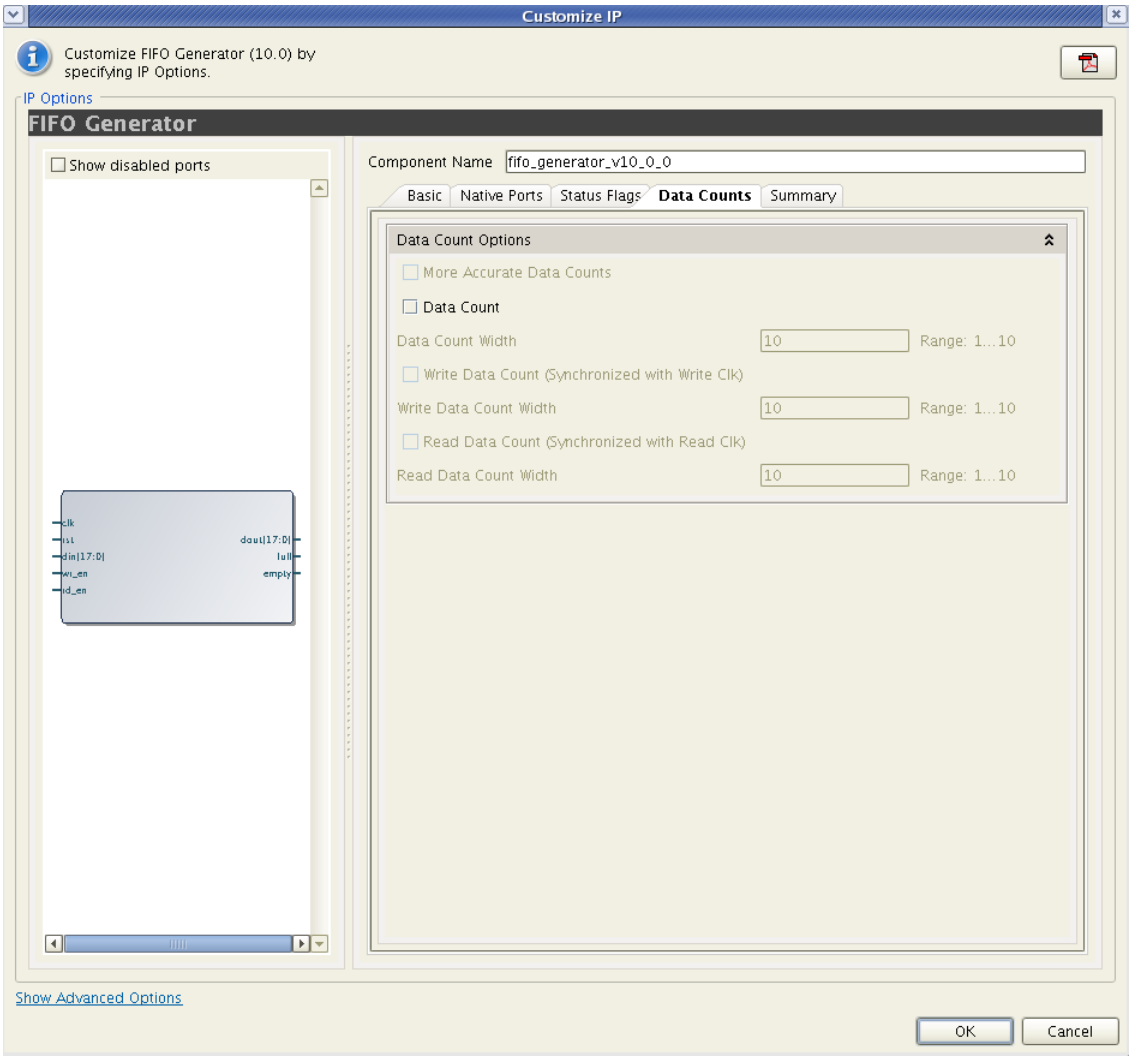


*Figure 5-4:* **Data Count Screen**

- **Data Count Options**: See Latency in Chapter 3 for the latency of the data counts due to write/read operation.
  - 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 in Chapter 3 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 log2 (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 log2 (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 log2 (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.

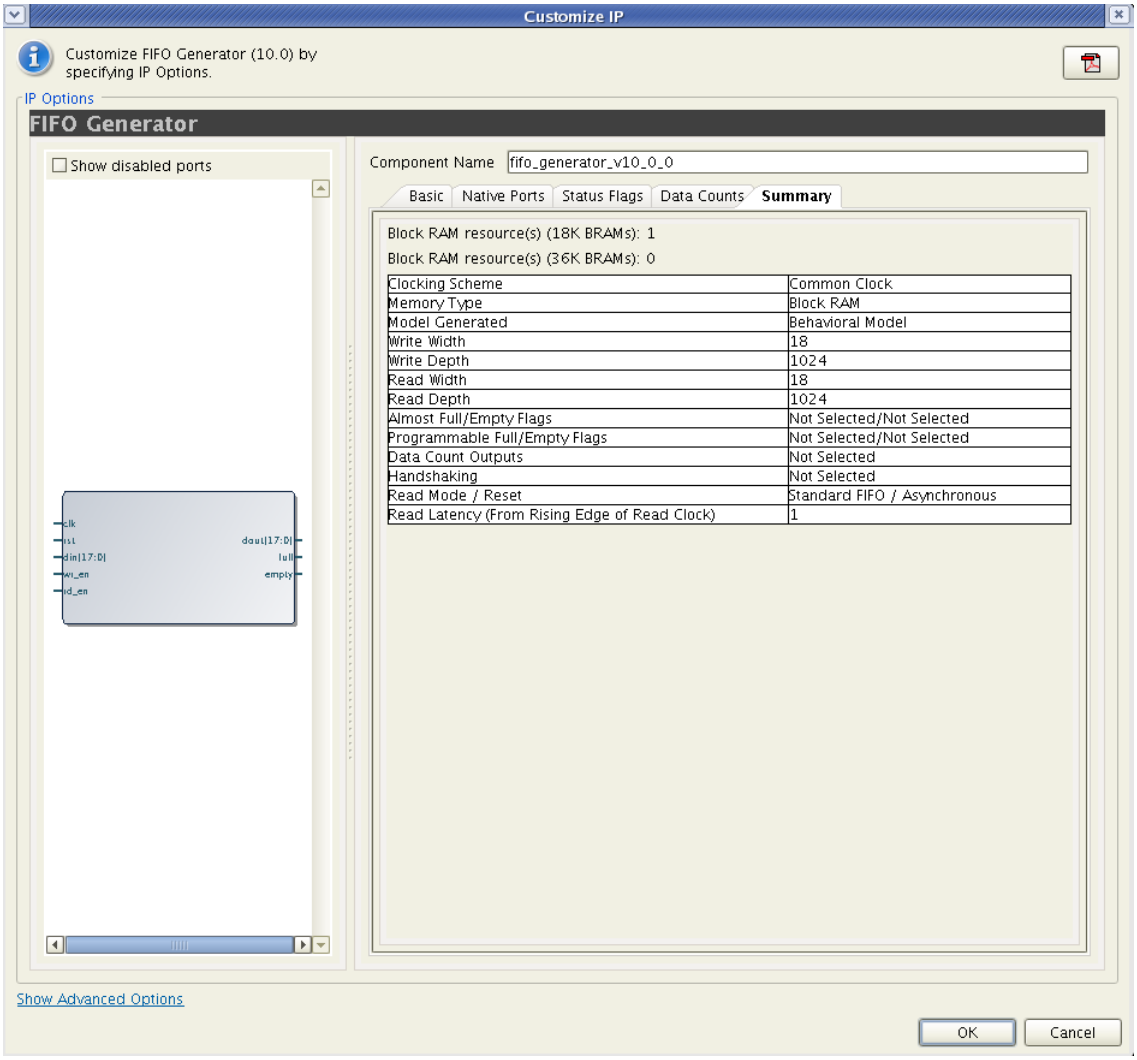


*Figure 5-5:* **Summary Screen**

# Output Generation

The output files generated from the Xilinx Vivado Design Suite are placed in the <project_directory> top-level directory. Depending on the settings, the file output list may include some or all of the following files:

- <project_directory>/<project_name.data>
  Contains constraints and file set details.
- <project directory>/<project_name>.src/sources_1/ip/<component name>
  Contains the sources like XCI, XDC, TCL and document files.
- synth
  Contains the source file necessary to synthesize the FIFO Generator

sim
Contains the source file necessary to synthesize the FIFO Generator

<component name>/example_design
Contains the source file necessary to synthesize the example design

<component name>/simulation
Contains the source file necessary to simulate the example design

The FIFO Generator core directories and their associated files are defined in the following sections.

## <project directory>/<project_name>.src/sources_1/ip/<component name>

This directory contains templates for instantiation of the core, example design, synth, XML and the XCI files.

*Table 5-1:* **Component Name Directory**

| Name | Description |
|---|---|
| <component_name>.xci | Log file from VIVADO software describing which options were used to generate the FIFO Generator core. An XCI file can also be used as an input to the Vivado Design Suite. |
| <component_name>.{veo\|vho} | VHDL or Verilog instantiation template. |

## synth

The synth directory contains the FIFO Generator synthesis file.

*Table 5-2:* **Synth Directory**

| Name | Description |
|---|---|
| <component_name>.vhd | VHDL file from Vivado Design Suite used to synthesize the FIFO Generator core. |

## sim

The sim directory contains the FIFO Generator simulation wrapper file.

*Table 5-3:* **Sim Directory**

| Name | Description |
|---|---|
| <component_name>.vhd | A VHDL file from VIVADO software to simulate the FIFO Generator. |

## <component name>/example_design

The example design directory contains the example design files provided with the core.

*Table 5-4:* **Example Design Directory**

| Name | Description |
|---|---|
| <component_name>_exdes.vhd | The VHDL top-level file for the example design. It instantiates the FIFO Generator core. This file contains entity with the IO's required for the core configuration. |
| <component_name>_exdes.xdc | Provides an example clock constraint for processing the FIFO Generator core using the Vivado Design Suite implementation tools. |

## <component name>/simulation

The simulation directory contains the simulation files provided with the core.

*Table 5-5:* **Simulation Directory**

| Name | Description |
|---|---|
| <component_name>_dverif.vhd | This VHDL file verifies the output data against the input data. |
| <component_name>_pctrl.vhd | This VHDL file generates the control signals to the core. |
| <component_name>_dgen.vhd | This VHDL file generates the random input data to the FIFO Generator core. |
| <component_name>_tb_pkg.vhd | This VHDL file has all the common functions used in stimulus generation. |
| <component_name>_tb_synth.vhd | This VHDL file instantiates the example design. |
| <component_name>_tb_rng.vhd | This VHDL file has the random number generation used to generate input data for FIFO Writes. |
| <component_name>_tb.vhd | This VHDL file is the top-level test bench File. |

# Customizing and Generating the AXI4 Core

This chapter includes information about using Xilinx tools to customize and generate the FIFO Generator for AXI4 FIFO Interfaces in the Vivado Design Suite.

## GUI

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 1.

- 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 6-1 shows the AXI4 interface selection screen.

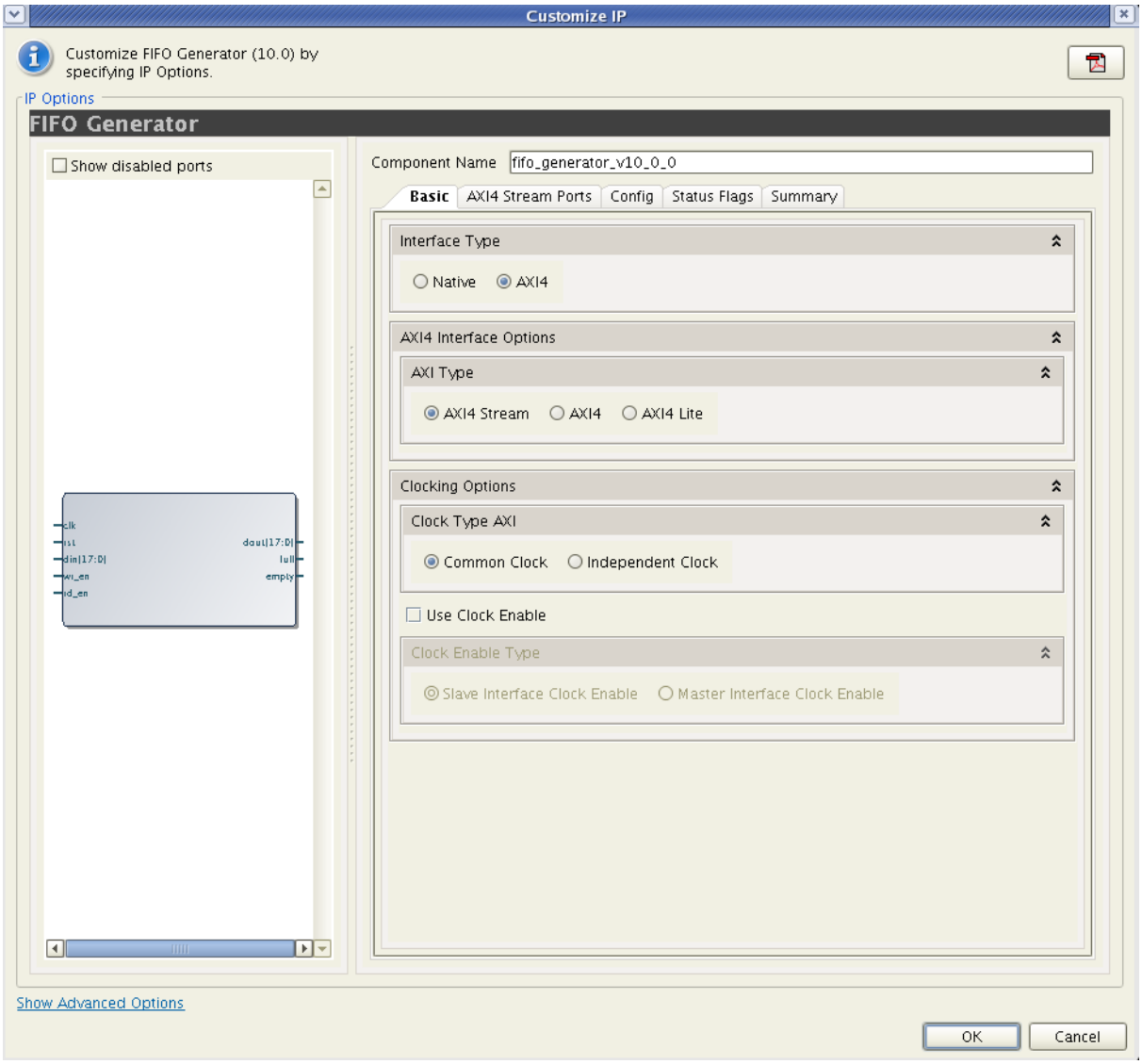


*Figure 6-1:* **AXI4 Interface Selection Screen**

- **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 3.

For more details on Independent Clock FIFO, see Independent Clocks: Block RAM and Distributed RAM in Chapter 3.

**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.

## 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 6-1 and Table 6-2. GUI screens are available for configuring:

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

## AXI4-Stream Width Calculation

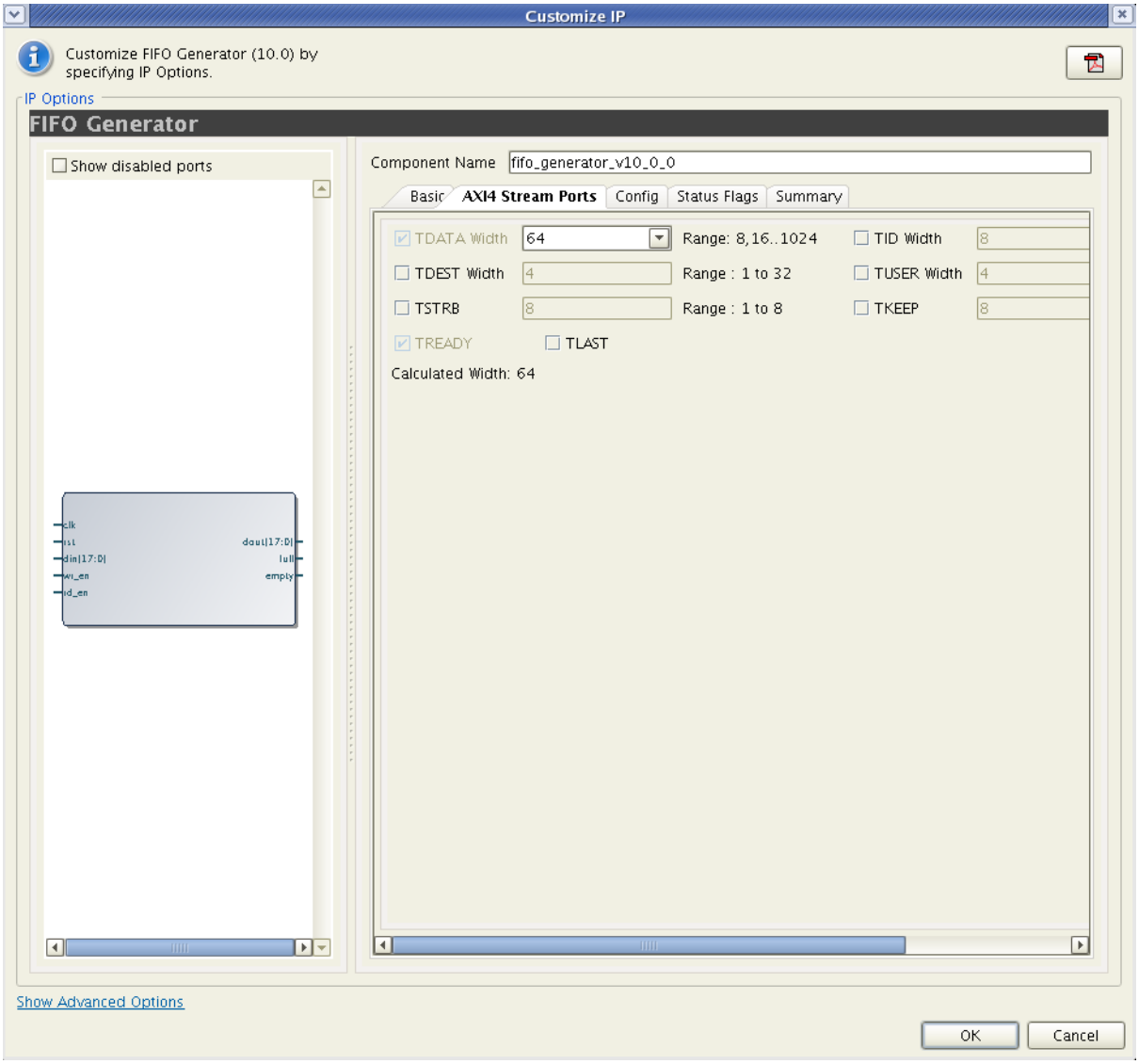


*Figure 6-2:* **AXI4-Stream Width Calculation Screen**

The AXI4-Stream FIFO allows the 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

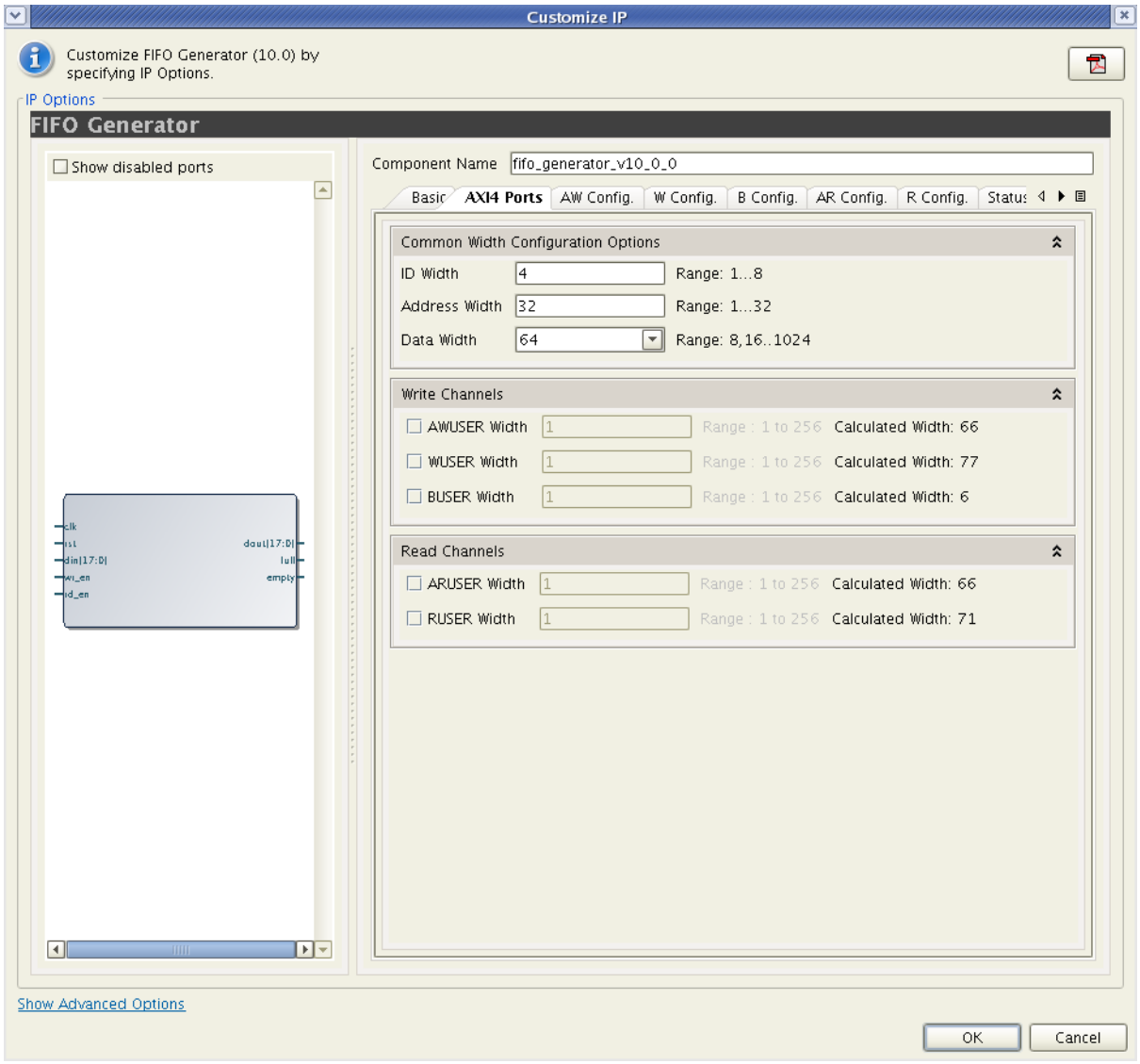


*Figure 6-3:* **AXI4 Width Calculation Screen**

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 6-1.

*Table 6-1:* **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] | |
| 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

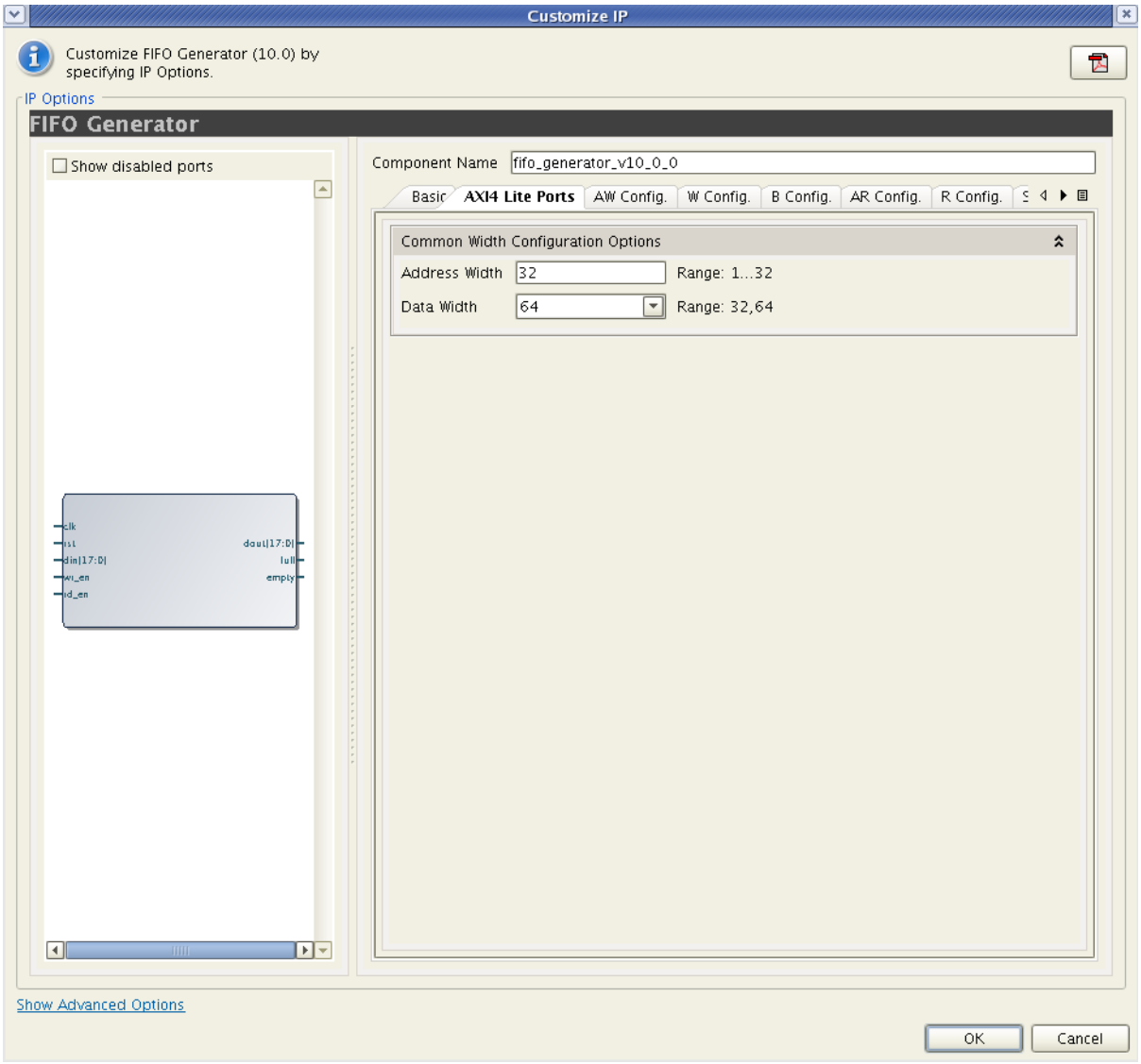


*Figure 6-4:* **AXI4-Lite Width Calculation Screen**

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 6-2.

*Table 6-2:* **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 6-3 shows the default settings for each AXI4 interface type.

*Table 6-3:* **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

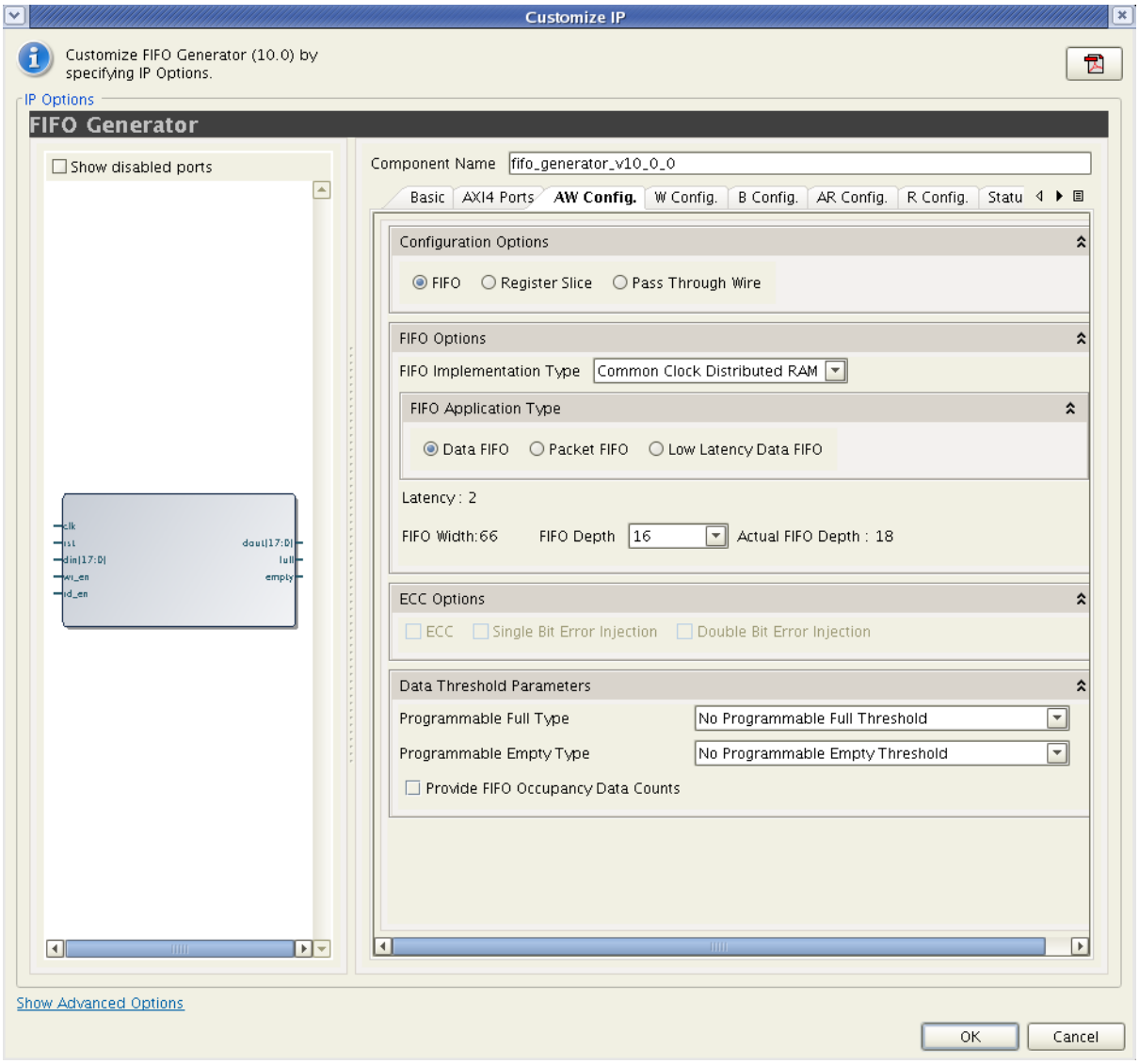


*Figure 6-5:* **AXI4 FIFO Configuration Screen**

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), Programmable 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 3.

### 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 7 series 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 3.

### 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 157.

### FIFO Depth

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

## Programmable Flags

This section includes details about the available 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 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 Threshold Constants is selected. Enter a user-defined value. The valid range for this threshold is provided in the GUI.

## Data Threshold Parameters

This section includes details about data threshold parameters.

### 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 3 and More Accurate Data Count (Use Extra Logic) in Chapter 3.

#### 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

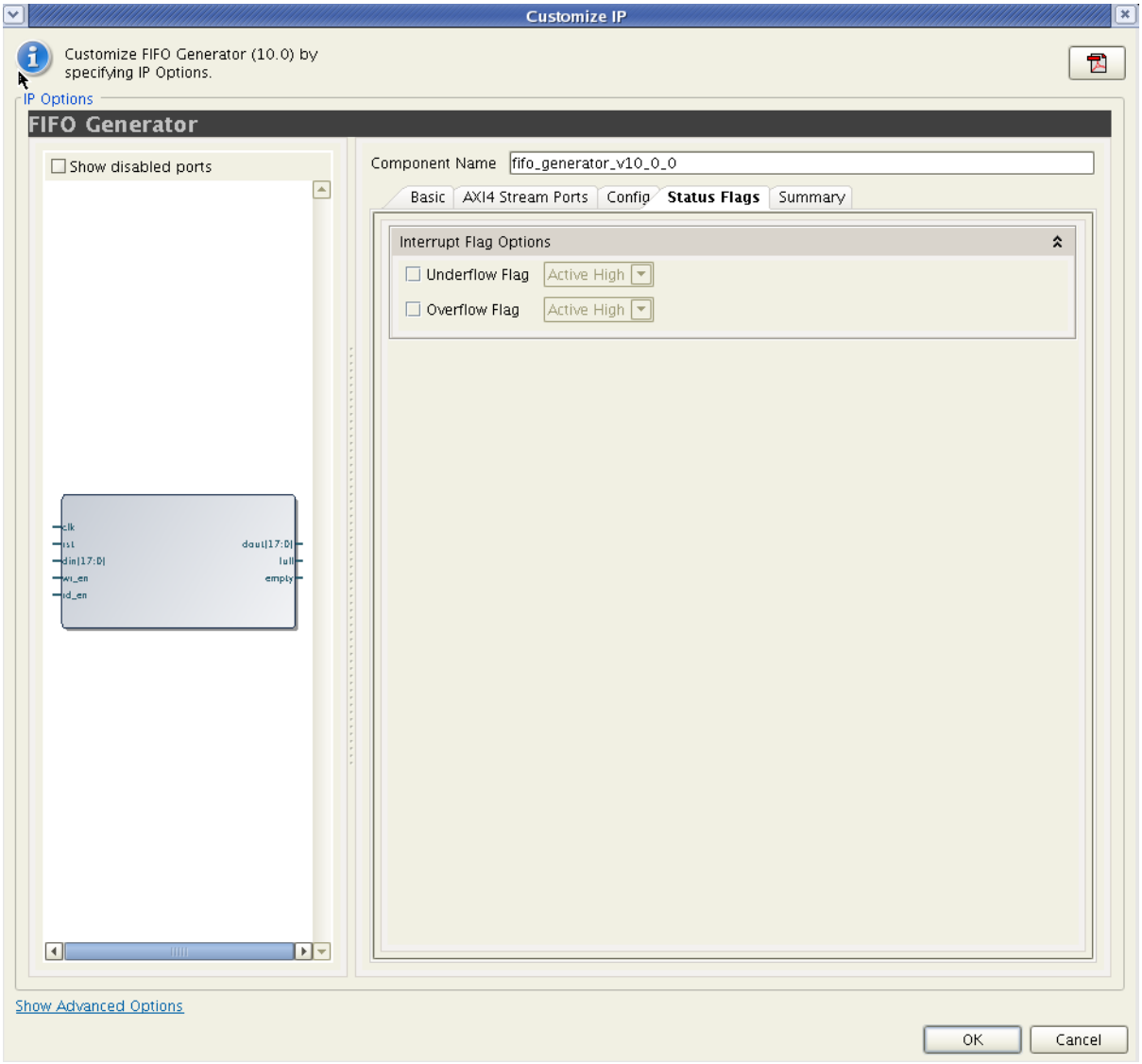


*Figure 6-6:* **AXI4 FIFO Common Configurations Screen**

### 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 3 and Overflow in Chapter 3.

## 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

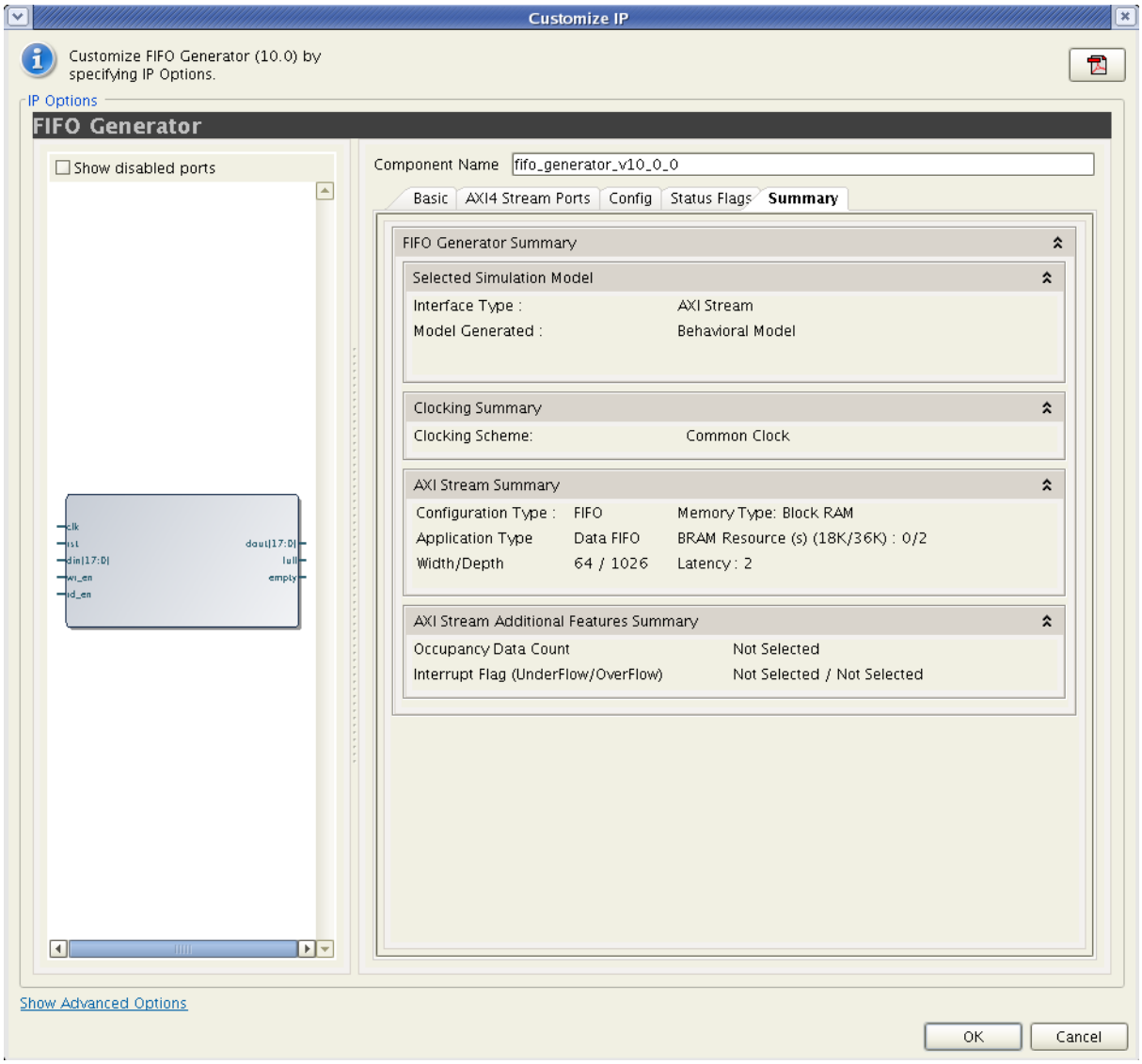


*Figure 6-7:* **AXI4-Stream Summary Screen**

## AXI4 and AXI4-Lite Summary

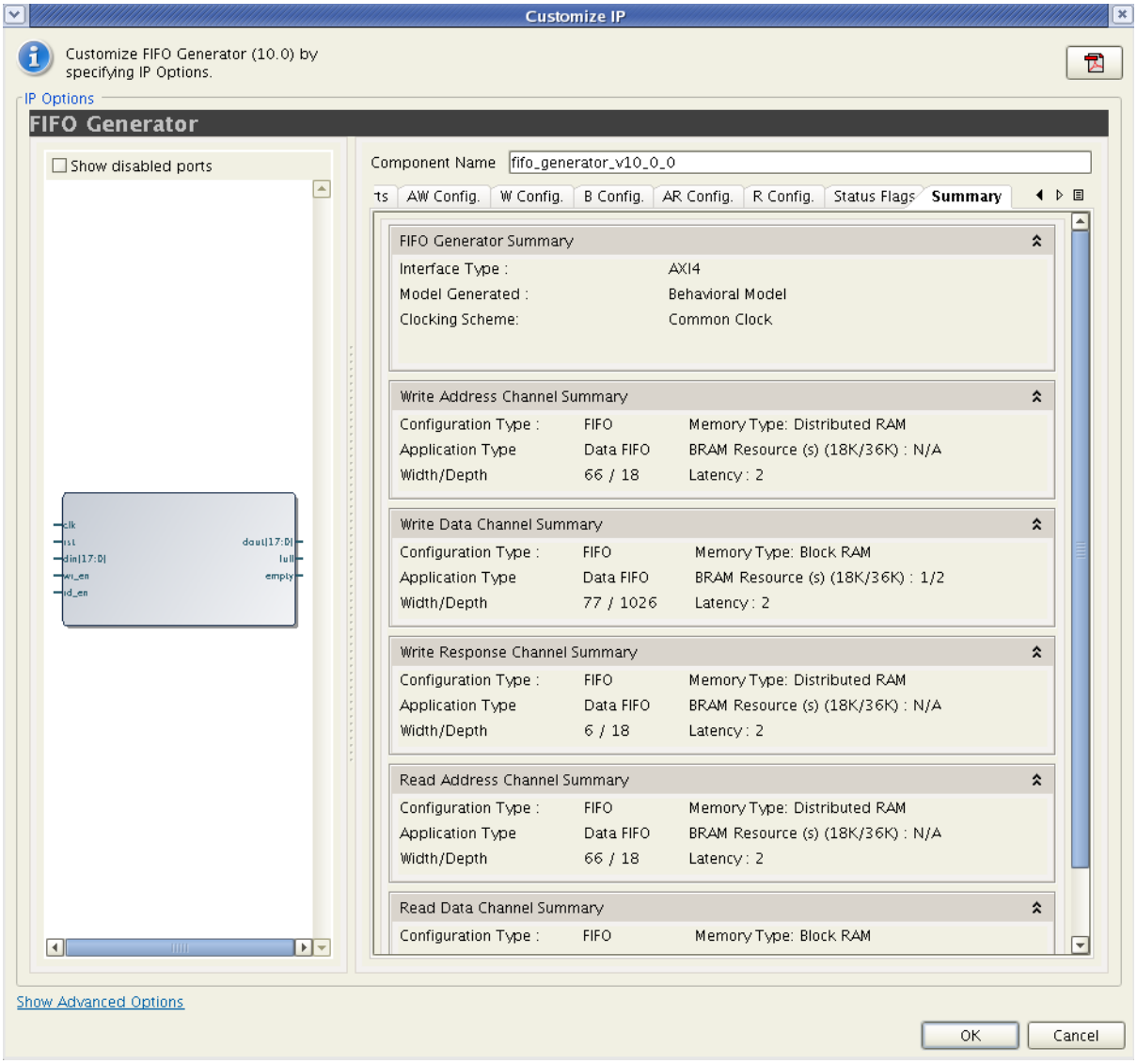


*Figure 6-8:* **AXI4 / AXI4-Lite Summary Screen**

# Output Generation

See Output Generation in Chapter 5 for details about the files generated with the core.

# Constraining the Core

This chapter contains details about any constraints for the FIFO Generator when implemented with the Vivado Design Suite.

## Required Constraints

The FIFO Generator core provides a sample clock constraint for synchronous and asynchronous FIFOs, and TIG constraints for asynchronous FIFOs. These sample constraints can be added to the user's design constraint file.

## Device, Package, and Speed Grade Selections

See IP Facts for details about supported devices.

## Clock Frequencies

There are no clock frequency constraints.

## Clock Management

There are no additional clock management constraints for this core.

## Clock Placement

There are no additional clock placement constraints for this core.

# Detailed Example Design

This chapter provides detailed information about the example design, including the purpose and contents of the provided scripts, the contents of the example HDL wrappers, and the operation of the demonstration test bench.

## Directory and File Contents

See Output Generation in Chapter 9 for output directory and file details.

## Example Design

Figure 8-1 shows the configuration of the example design.

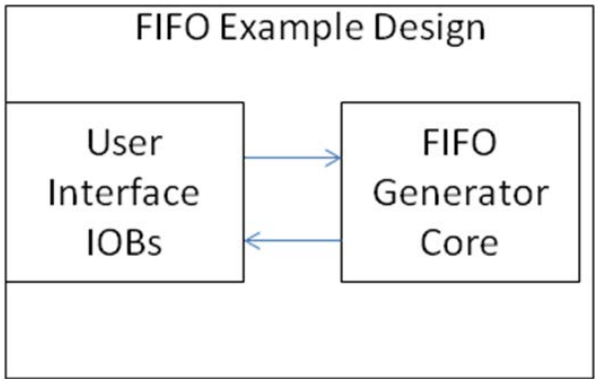


*Figure 8-1:* **Example Design Configuration**

The example design contains the following:

- An instance of the FIFO Generator core. During simulation, the FIFO Generator core is instantiated as a black box and replaced during implementation with the structural netlist model generated by the Vivado IP Catalog IP customizer for timing simulation or a behavioral model for the functional simulation.
- Global clock buffers for top-level port clock signals.

# Demonstration Test Bench

Figure 8-2 shows a block diagram of the demonstration test bench.

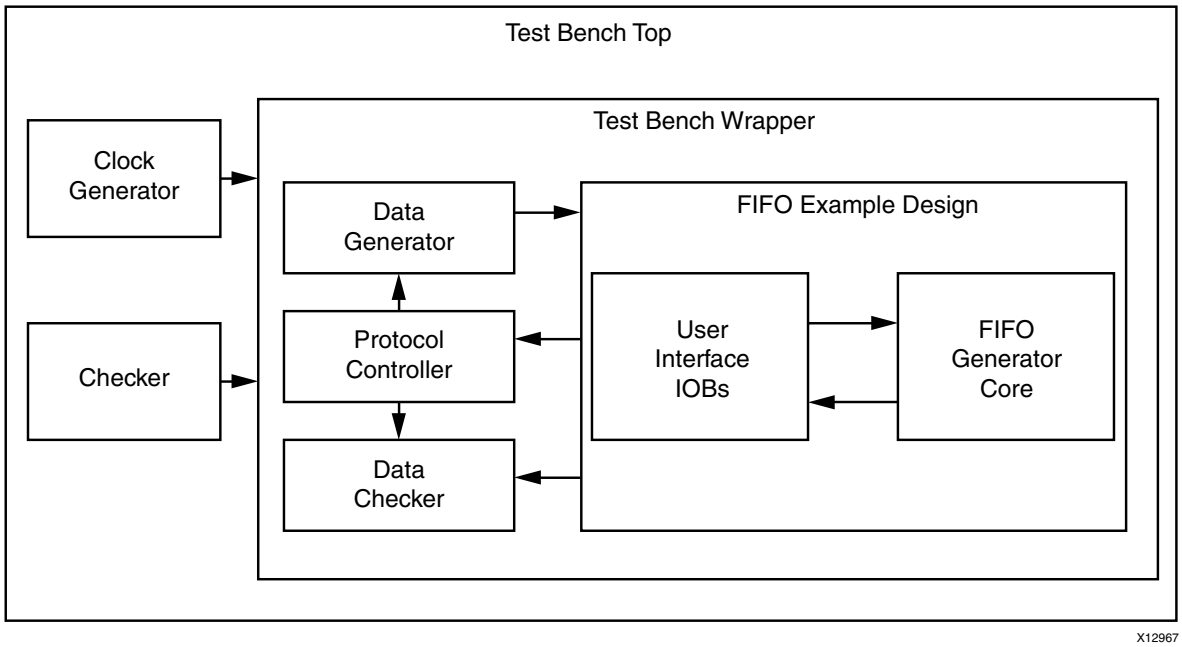


*Figure 8-2:* **Demonstration Test Bench**

## Test Bench Functionality

The demonstration test bench is a straightforward VHDL file that can be used to exercise the example design and the core itself. The test bench consists of the following:

- Clock Generators
- Data generator module
- Data verifier module
- Module to control data generator and verifier

### Core with Native Interface

The demonstration test bench in a core with a Native interface performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- Pseudo random data is generated and given as input to FIFO data input port.

- Data on DOUT port of the FIFO generator core is cross checked using another pseudo random generator with same seed as data input generator.
- Core is exercised for two full and empty conditions.
- Full/almost_full and empty/almost_empty flags are checked.

### Core with AXI4 Interface

The demonstration test bench in a core with an AXI4 interface performs the following tasks:

- Input clock signals are generated.
- A reset is applied to the example design.
- Pseudo random data is generated and given as input to FIFO AXI4 Interface input signals. Each channel is independently checked for Valid-Ready handshake protocol.
- AXI4 output signals on read side are combined and cross checked with the pseudo random generator data.
- For AXI4 Full/Lite interface five instances of data generator, data verifier and protocol controller are used.
- For AXI4 Full Packet FIFO write address and read address channels valid/ready signals are not checked.

## Customizing the Demonstration Test Bench

This section describes the variety of demonstration test bench customization options that can be used for individual system requirements.

### Changing the Data/Stimulus

The random data/stimulus can be altered by changing the seed passed to FIFO generator test bench wrapper module in test bench top file (`fg_tb_top.vhd`).

### Changing the Test Bench Run Time

The test bench iteration count (number of full/empty conditions before finish) can be altered by changing the value passed to TB_STOP_CNT parameter. A '0' to this parameter runs the test bench until the test bench timeout value set in test bench top file (`fg_tb_top.vhd`).

It is also possible to decide whether to stop the simulation on error or on reaching the count set by TB_STOP_CNT by using FREEZEON_ERROR parameter value (1(TRUE), 0(FALSE)) of test bench wrapper file (`fg_tb_synth.vhd`).

# Messages and Warnings

When the functional or timing simulation has completed successfully, the test bench displays the following message, and it is safe to ignore this message.

```
Failure: Test Completed Successfully
```

XILINX

*Appendix A*

# Verification, Compliance, and Interoperability

Xilinx has verified the FIFO Generator core in a proprietary test environment, using an internally developed bus functional model. Tens of thousands of test vectors were generated and verified, including both valid and invalid write and read data accesses.

## Simulation

The FIFO Generator has been tested with Xilinx Vivado Design Suite, Xilinx ISIM/XSIM, Cadence Incisive Enterprise Simulator (IES), Synopsys VCS and VCS MX and Mentor Graphics Questa SIM simulator.

# Debugging

This appendix provides information for using the resources available on the Xilinx Support website, debug tools, and other step-by-step processes for debugging designs that use the FIFO Generator.

The following topics are included in this appendix:



## Finding Help on Xilinx.com

To help in the design and debug process when using the FIFO Generator, the Xilinx Support web page (www.xilinx.com/support) contains key resources such as product documentation, answer records, information about known issues, and links for opening a Technical Support Web Case.

### Documentation

This product guide is the main document associated with the FIFO Generator. This guide, along with documentation related to all products that aid in the design process, can be found on the Xilinx Support web page (www.xilinx.com/support) or by using the Xilinx Documentation Navigator.

Download the Xilinx Documentation Navigator from the Design Tools tab on the Downloads page (www.xilinx.com/download). For more information about this tool and the features available, open the online help after installation.

## Known Issues

Answer Records include information about commonly encountered problems, helpful information on how to resolve these problems, and any known issues with a Xilinx product. Answer Records are created and maintained daily ensuring that users have access to the most accurate information available.

Answer Records for this core are listed below, and can also be located by using the Search Support box on the main Xilinx support web page. To maximize your search results, use proper keywords such as:

- Product name
- Tool message(s)
- Summary of the issue encountered

A filter search is available after results are returned to further target the results.

**Answer Records for the FIFO Generator**

- AR50917, FIFO Generator Release Notes

## Contacting Technical Support

Xilinx provides technical support at www.xilinx.com/support for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support of product if implemented in devices that are not defined in the documentation, if customized beyond that allowed in the product documentation, or if changes are made to any section of the design labeled DO NOT MODIFY.

Xilinx provides premier technical support for customers encountering issues that require additional assistance.

To contact Xilinx Technical Support:

1. Navigate to www.xilinx.com/support.
2. Open a WebCase by selecting the WebCase link located under Support Quick Links.

When opening a WebCase, include:

- Target FPGA including package and speed grade.
- All applicable Xilinx Design Tools and simulator software versions.
- Additional files based on the specific issue might also be required. See the relevant sections in this debug guide for guidelines about which file(s) to include with the WebCase.

# Debug Tools

There are many tools available to address FIFO Generator design issues. It is important to know which tools are useful for debugging various situations.

## Example Design

The FIFO Generator is delivered with an example design that can be synthesized, complete with functional test benches. Information about the example design can be found inChapter 8, Detailed Example Design for the Vivado Design Suite.

## Vivado Lab Tools

Vivado inserts logic analyzer and virtual I/O cores directly into your design. Vivado Lab Tools allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed. This feature represents the functionality in the Vivado IDE that is used for logic debugging and validation of a design running in Xilinx FPGA devices in hardware.

The Vivado logic analyzer is used to interact with the logic debug LogiCORE IP cores, including:

- ILA 2.0 (and later versions)
- VIO 2.0 (and later versions)

# Simulation Debug

For details about simulating a design in the Vivado Design Suite, see the *Vivado Logic Simulation User Guide* (UG900).

# Hardware Debug

Hardware issues can range from system start to problems seen after hours of testing. This section provides debug steps for common issues.

## General Checks

Ensure that all the timing constraints for the core were properly incorporated from the FIFO Generator and that all constraints were met during implementation.

- Does it work in post-place and route timing simulation? If problems are seen in hardware but not in timing simulation, this could indicate a PCB issue. Ensure that all clock sources are active and clean.
- If using MMCMs in the design, ensure that all MMCMs have obtained lock by monitoring the LOCKED port.
- Ensure WR_EN and RD_EN are not toggling during reset
- If Built-in FIFO is used, ensure reset guideline is followed. See Common/Independent Clock: Built-in in Chapter 3.
- If independent clock FIFO is used, ensure WR_EN is coming from the write clock domain and RD_EN is coming from the read clock domain.
- If Enable Reset Synchronization options are not selected, ensure WR_RST and RD_RST are synchronized using WR_CLK and RD_CLK before passing to FIFO Generator.
- If your outputs go to 0, check your licensing.

# Interface Debug

## Native Interface

If the data is not being written, check the following conditions:

- If FULL is High, the core cannot write the data
- Check if the core is in reset state.
- Check if WR_EN is synchronous to write domain clock.

If the data is not being read, check the following conditions:

- If EMPTY is High, the core cannot read the data
- Check if the core is in reset state.
- Check if RD_EN is synchronous to read domain clock.

# Quick Start Example Design

This chapter provides instructions to generate a FIFO generator core quickly, run the design through implementation with the Xilinx tools, and simulate the example design using the provided demonstration test bench. See the example design in Chapter 12, Detailed Example Design.

Figure C-1 shows the example design and demonstration test bench block diagram.

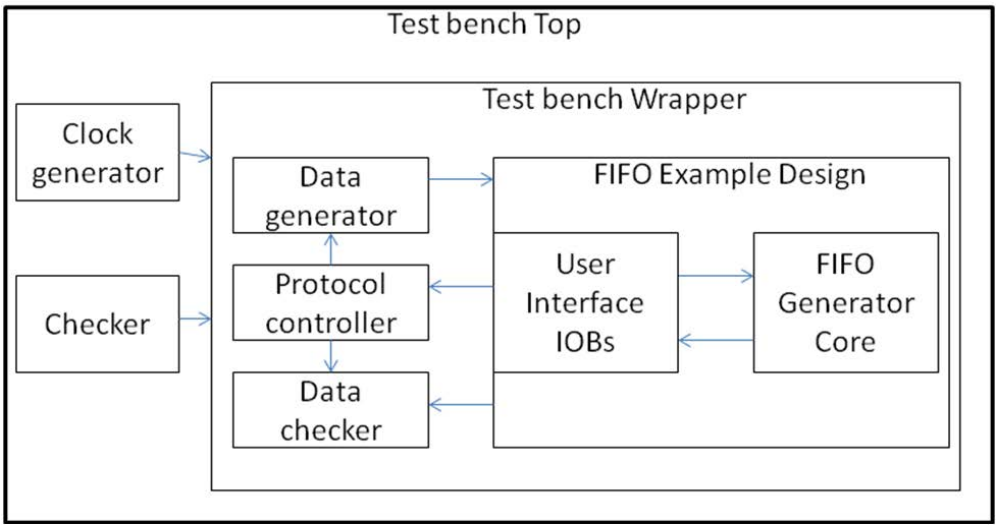


*Figure C-1:* **Example Design and Demonstration Test Bench**

The FIFO generator example design consists of the following:

- FIFO generator netlist/Behavioral model
- HDL wrapper which instantiates the FIFO generator netlist/Behavioral model
- Customizable demonstration test bench to simulate the example design

The FIFO generator example design has been tested with Xilinx Vivado Design Suite, Xilinx ISim, Cadence Incisive Enterprise Simulator (IES) and Mentor Graphics Questa SIM simulator.

# Implementing the Example Design

After generating a core the netlist and example design can be processed by the Xilinx implementation tools. To implement the FIFO Generator example design, open a command prompt or terminal window and type these commands:

**For Windows**:

```
ms-dos> cd <proj_dir>\<component_name>\implement
ms-dos> implement.bat
```

**For Linux**:

```
Linux-shell% cd <proj_dir>/<component_name>/implement
Linux-shell% ./implement.sh
```

These commands execute a script that synthesizes, builds, maps, and places-and-routes the example design. The script then generates a post-par simulation model for use in timing simulation. The resulting files are placed in the results directory.

# Simulating the Example Design

The FIFO Generator core provides a quick way to simulate and observe the behavior of the core by using the provided example design. There are five different simulation types:

- Behavioral
- Post-Synthesis Functional
- Post-Synthesis Timing
- Post-Implementation Functional
- Post-Implementation Timing

The simulation models provided are either in VHDL or Verilog, depending on the CORE project settings in the Vivado tools.

# DOUT Reset Value Timing

Figure D-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).

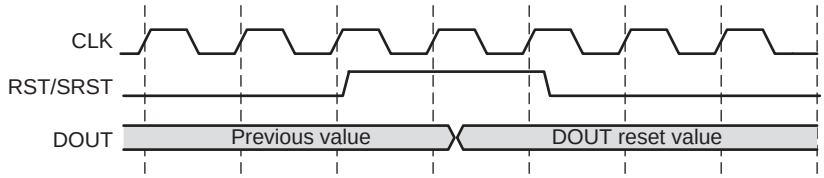


*Figure D-1:* **DOUT Reset Value for Synchronous Reset (SRST) and for Asynchronous Reset (RST) for Common Clock Block RAM Based FIFO**

Figure D-2 shows the DOUT reset value for common clock distributed RAM and Shift Register based FIFOs for asynchronous reset (RST).

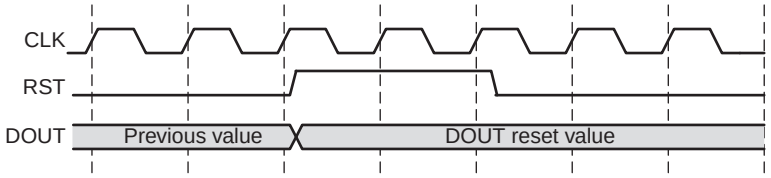


*Figure D-2:* **DOUT Reset Value for Asynchronous Reset (RST) for Common Clock Distributed/ Shift RAM Based FIFO**

Figure D-3 shows the DOUT reset value for the common clock built-in FIFOs with embedded register for asynchronous reset (RST).

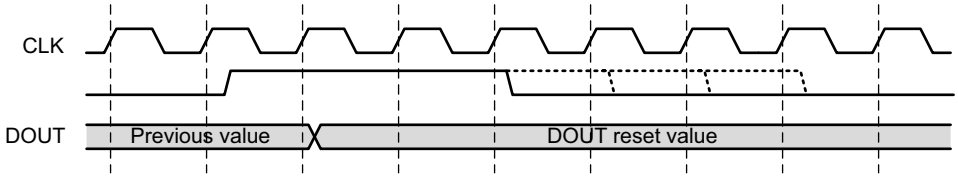


*Figure D-3:* **DOUT Reset Value for Common Clock Built-in FIFO**

Figure D-4 shows the DOUT reset value for independent clock block RAM based FIFOs (RD_RST).

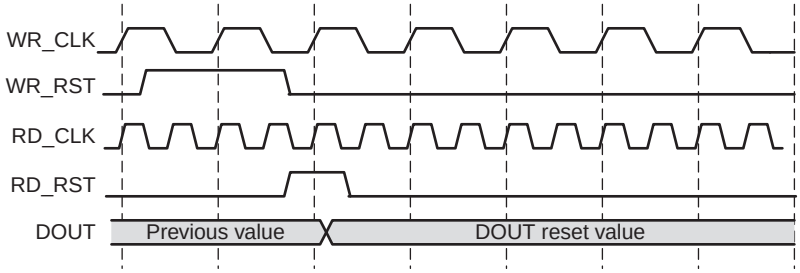


*Figure D-4:* **DOUT Reset Value for Independent Clock Block RAM Based FIFO**

Figure D-5 shows the DOUT reset value for independent clock distributed RAM based FIFOs (RD_RST).

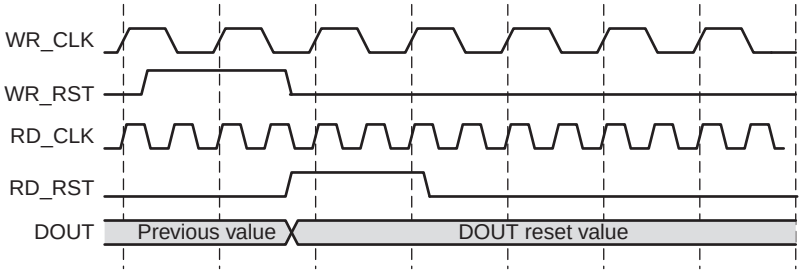


*Figure D-5:* **DOUT Reset Value for Independent Clock Distributed RAM Based FIFO**

# Supplemental Information

The following sections provide additional information about working with the FIFO Generator core.

## Auto-Upgrade Feature

The FIFO Generator core has an auto-upgrade feature for updating older versions of the FIFO Generator core to the latest version. The auto-upgrade feature can be seen by right clicking any pre-existing FIFO Generator core in your project in the Project IP tab of the Vivado IP Catalog.

## Native FIFO SIM Parameters

Table E-1 defines the Native FIFO SIM parameters used to specify the configuration of the core. These parameters are only used while instantiating the core in HDL manually or while calling the core dynamically. This parameter list does not apply to a core generated using the IP Catalog GUI.

*Table E-1:* **Native FIFO SIM Parameters**

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 1 | C_COMMON_CLOCK | Integer | • 0: Independent Clock<br>• 1: Common Clock |
| 2 | C_DATA_COUNT_WIDTH | Integer | Width of DATA_COUNT bus (1 – 23) |
| 3 | C_DIN_WIDTH | Integer | Width of DIN bus (1 – 1024)<br>Width must be > 1 for ECC with Double bit error injection |
| 4 | C_DOUT_RST_VAL | String | Reset value of DOUT<br>Hexadecimal value, 0 - 'F's equal to C_DOUT_WIDTH |
| 5 | C_DOUT_WIDTH | Integer | Width of DOUT bus (1 – 1024)<br>Width must be > 1 for ECC with Double bit error injection |
| 6 | C_ENABLE_RST_SYNC | Integer | • 0: Do not synchronize the reset (WR_RST/ RD_RST is directly used, available only for independent clock)<br>• 1: Synchronize the reset |

*Table E-1:* **Native FIFO SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 7 | C_ERROR_INJECTION_TYPE | Integer | • 0: No error injection<br>• 1: Single bit error injection<br>• 2: Double bit error injection<br>• 3: Single and double bit error injection |
| 8 | C_FAMILY | String | Device family (for example, Virtex-7 or Kintex-7) |
| 9 | C_FULL_FLAGS_RST_VAL | Integer | Full flags rst val (0 or 1) |
| 10 | C_HAS_ALMOST_EMPTY | Integer | • 0: Core does not have ALMOST_EMPTY flag<br>• 1: Core has ALMOST_EMPTY flag |
| 11 | C_HAS_ALMOST_FULL | Integer | • 0: Core does not have ALMOST_FULL flag<br>• 1: Core has ALMOST_ FULL flag |
| 12 | C_HAS_DATA_COUNT | Integer | • 0: Core does not have DATA_COUNT bus<br>• 1: Core has DATA_COUNT bus |
| 13 | C_HAS_OVERFLOW | Integer | • 0: Core does not have OVERFLOW flag<br>• 1: Core has OVERFLOW flag |
| 14 | C_HAS_RD_DATA_COUNT | Integer | • 0: Core does not have RD_DATA_COUNT bus<br>• 1: Core has RD_DATA_COUNT bus |
| 15 | C_HAS_RST | Integer | • 0: Core does not have asynchronous reset (RST)<br>• 1: Core has asynchronous reset (RST) |
| 16 | C_HAS_SRST | Integer | • 0: Core does not have synchronous reset (SRST)<br>• 1: Core has synchronous reset (SRST) |
| 17 | C_HAS_UNDERFLOW | Integer | • 0: Core does not have UNDERFLOW flag<br>• 1: Core has UNDERFLOW flag |
| 18 | C_HAS_VALID | Integer | • 0: Core does not have VALID flag<br>• 1: Core has VALID flag |
| 19 | C_HAS_WR_ACK | Integer | • 0: Core does not have WR_ACK flag<br>• 1: Core has WR_ACK flag |
| 20 | C_HAS_WR_DATA_COUNT | Integer | • 0: Core does not have WR_DATA_COUNT bus<br>• 1: Core has WR_DATA_COUNT bus |
| 21 | C_IMPLEMENTATION_TYPE | Integer | • 0: Common-Clock Block RAM/Distributed RAM FIFO<br>• 1: Common-Clock Shift RAM FIFO<br>• 2: Independent Clocks Block RAM/ Distributed RAM FIFO<br>• 6: 7 Series Built-in FIFO |

*Table E-1:* **Native FIFO SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 22 | C_MEMORY_TYPE | Integer | • 1: Block RAM<br>• 2: Distributed RAM<br>• 3: Shift RAM<br>• 4: Built-in FIFO |
| 23 | C_MSGON_VAL | Integer | • 0: Disables timing violation on cross clock domain registers<br>• 1: Enables timing violation on cross clock domain registers |
| 24 | C_OVERFLOW_LOW | Integer | • 0: OVERFLOW active high<br>• 1: OVERFLOW active low |
| 25 | C_PRELOAD_LATENCY | Integer | • 0: First-Word Fall-Through with or without Embedded Register<br>• 1: Standard FIFO without Embedded Register<br>• 2: Standard FIFO with Embedded Register |
| 26 | C_PRELOAD_REGS | Integer | • 0: Standard FIFO without Embedded Register<br>• 1: Standard FIFO with Embedded Register or First-Word Fall-Through with or without Embedded Register |
| 27 | C_PRIM_FIFO_TYPE | String | Primitive used to build a FIFO<br>(Ex. "512x36") |
| 28 | C_PROG_EMPTY_THRESH_ASSERT_VAL | Integer | PROG_EMPTY assert threshold[a] |
| 29 | C_PROG_EMPTY_THRESH_NEGATE_VAL | Integer | PROG_EMPTY negate threshold (a) |
| 30 | C_PROG_EMPTY_TYPE | Integer | • 0: No programmable empty<br>• 1: Single programmable empty thresh constant<br>• 2: Multiple programmable empty thresh constants<br>• 3: Single programmable empty thresh input<br>• 4: Multiple programmable empty thresh inputs |
| 31 | C_PROG_FULL_THRESH_ASSERT_VAL | Integer | PROG_FULL assert threshold (a) |
| 32 | C_PROG_FULL_THRESH_NEGATE_VAL | Integer | PROG_FULL negate threshold (a) |
| 33 | C_PROG_FULL_TYPE | Integer | • 0: No programmable full<br>• 1: Single programmable full thresh constant<br>• 2: Multiple programmable full thresh constants<br>• 3: Single programmable full thresh input<br>• 4: Multiple programmable full thresh inputs |
| 34 | C_RD_DATA_COUNT_WIDTH | Integer | Width of RD_DATA_COUNT bus (1 - 23) |
| 35 | C_RD_DEPTH | Integer | Depth of read interface (16 – 4194305) |

*Table E-1:* **Native FIFO SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 36 | C_RD_FREQ | Integer | Read clock frequency (1 MHz - 1000 MHz) |
| 37 | C_RD_PNTR_WIDTH | Integer | log2(C_RD_DEPTH) |
| 38 | C_UNDERFLOW_LOW | Integer | • 0: UNDERFLOW active high<br>• 1: UNDERFLOW active low |
| 39 | C_USE_DOUT_RST | Integer | • 0: Does not reset DOUT on RST<br>• 1: Resets DOUT on RST |
| 40 | C_USE_ECC | Integer | • 0: Does not use ECC feature<br>• 1: Uses ECC feature |
| 41 | C_USE_EMBEDDED_REG | Integer | • 0: Does not use BRAM embedded output register<br>• 1: Uses BRAM embedded output register |
| 42 | C_USE_FWFT_DATA_COUNT | Integer | • 0: Does not use extra logic for FWFT data count<br>• 1: Uses extra logic for FWFT data count |
| 43 | C_VALID_LOW | Integer | • 0: VALID active high<br>• 1: VALID active low |
| 44 | C_WR_ACK_LOW | Integer | • 0: WR_ACK active high<br>• 1: WR_ACK active low |
| 45 | C_WR_DATA_COUNT_WIDTH | Integer | Width of WR_DATA_COUNT bus (1 – 23) |
| 46 | C_WR_DEPTH | Integer | Depth of write interface (16 – 4194305) |
| 47 | C_WR_FREQ | Integer | Write clock frequency (1 MHz - 1000 MHz) |
| 48 | C_WR_PNTR_WIDTH | Integer | log2(C_WR_DEPTH) |

a. See the FIFO Generator GUI for the allowable range of values.

## AXI4 FIFO SIM Parameters

Table E-2 defines the AXI4 SIM parameters used to specify the configuration of the core. These parameters are only used while instantiating the core in HDL manually or while calling the core dynamically. This parameter list does not apply to a core generated using the IP Catalog GUI.

*Table E-2:* **AXI4 SIM Parameters**

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 1 | C_INTERFACE_TYPE | Integer | • 0: Native FIFO<br>• 1: AXI4 FIFO |
| 2 | C_AXI_TYPE | Integer | • 0: AXI4-Stream<br>• 1: AXI4<br>• 2: AXI4-Lite |
| 3 | C_HAS_AXI_WR_CHANNEL | Integer | • 0: Core does not have Write Channel [a]<br>• 1: Core has Write Channel [a] |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 4 | C_HAS_AXI_RD_CHANNEL | Integer | • 0: Core does not have Read Channel[b]<br>• 1: Core has Read Channel[b] |
| 5 | C_HAS_SLAVE_CE[c] | Integer | • 0: Core does not have Slave Interface Clock Enable<br>• 1: Core has Slave Interface Clock Enable |
| 6 | C_HAS_MASTER_CE[c] | Integer | • 0: Core does not have Master Interface Clock Enable<br>• 1: Core has Master Interface Clock Enable |
| 7 | C_ADD_NGC_CONSTRAINT[c] | Integer | • 0: Core does not add NGC constraint<br>• 1: Core adds NGC constraint |
| 8 | C_USE_COMMON_UNDERFLOW[c] | Integer | • 0: Core does not have common UNDERFLOW flag<br>• 1: Core has common UNDERFLOW flag |
| 9 | C_USE_COMMON_OVERFLOW[c] | Integer | • 0: Core does not have common OVERFLOW flag<br>• 1: Core has common OVERFLOW flag |
| 10 | C_USE_DEFAULT_SETTINGS[c] | Integer | • 0: Core does not use default settings<br>• 1: Core uses default settings |
| 11 | C_AXI_ID_WIDTH | Integer | ID Width |
| 12 | C_AXI_ADDR_WIDTH | Integer | Address Width |
| 13 | C_AXI_DATA_WIDTH | Integer | Data Width |
| 14 | C_HAS_AXI_AWUSER | Integer | • 0: Core does not have AWUSER<br>• 1: Core has AWUSER |
| 15 | C_HAS_AXI_WUSER | Integer | • 0: Core does not have WUSER<br>• 1: Core has WUSER |
| 16 | C_HAS_AXI_BUSER | Integer | • 0: Core does not have BUSER<br>• 1: Core has BUSER |
| 17 | C_HAS_AXI_ARUSER | Integer | • 0: Core does not have ARUSER<br>• 1: Core has ARUSER |
| 18 | C_HAS_AXI_RUSER | Integer | • 0: Core does not have RUSER<br>• 1: Core has RUSER |
| 19 | C_AXI_AWUSER_WIDTH | Integer | AWUSER Width |
| 20 | C_AXI_WUSER_WIDTH | Integer | WUSER Width |
| 21 | C_AXI_BUSER_WIDTH | Integer | BUSER Width |
| 22 | C_AXI_ARUSER_WIDTH | Integer | ARUSER Width |
| 23 | C_AXI_RUSER_WIDTH | Integer | RUSER Width |
| 24 | C_HAS_AXIS_TDATA | Integer | • 0: AXI4 Stream does not have TDATA<br>• 1: AXI4 Stream has TDATA |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 25 | C_HAS_AXIS_TID | Integer | • 0: AXI4 Stream does not have TID<br>• 1: AXI4 Stream has TID |
| 26 | C_HAS_AXIS_TDEST | Integer | • 0: AXI4 Stream does not have TDEST<br>• 1: AXI4 Stream has TDEST |
| 27 | C_HAS_AXIS_TUSER | Integer | • 0: AXI4 Stream does not have TUSER<br>• 1: AXI4 Stream has TUSER |
| 28 | C_HAS_AXIS_TREADY | Integer | • 0: AXI4 Stream does not have TREADY<br>• 1: AXI4 Stream has TREADY |
| 29 | C_HAS_AXIS_TLAST | Integer | • 0: AXI4 Stream does not have TLAST<br>• 1: AXI4 Stream has TLAST |
| 30 | C_HAS_AXIS_TSTRB | Integer | • 0: AXI4 Stream does not have TSTRB<br>• 1: AXI4 Stream has TSTRB |
| 31 | C_HAS_AXIS_TKEEP | Integer | • 0: AXI4 Stream does not have TKEEP<br>• 1: AXI4 Stream has TKEEP |
| 32 | C_AXIS_TDATA_WIDTH | Integer | AXI4 Stream TDATA Width |
| 33 | C_AXIS_TID_WIDTH | Integer | AXI4 Stream TID Width |
| 34 | C_AXIS_TDEST_WIDTH | Integer | AXI4 Stream TDEST Width |
| 35 | C_AXIS_TUSER_WIDTH | Integer | AXI4 Stream TUSER Width |
| 36 | C_AXIS_TSTRB_WIDTH | Integer | AXI4 Stream TSTRB Width |
| 37 | C_AXIS_TKEEP_WIDTH | Integer | AXI4 Stream TKEEP Width |
| 38 | C_WACH_TYPE | Integer | Write Address Channel type<br>• 0: FIFO<br>• 1: Register Slice<br>• 2: Pass Through Logic |
| 39 | C_WDCH_TYPE | Integer | Write Data Channel type<br>• 0: FIFO<br>• 1: Register Slice<br>• 2: Pass Through Logic |
| 40 | C_WRCH_TYPE | Integer | Write Response Channel type<br>• 0: FIFO<br>• 1: Register Slice<br>• 2: Pass Through Logic |
| 41 | C_RACH_TYPE | Integer | Read Address Channel type<br>• 0: FIFO<br>• 1: Register Slice<br>• 2: Pass Through Logic |
| 42 | C_RDCH_TYPE | Integer | Read Data Channel type<br>• 0: FIFO<br>• 1: Register Slice<br>• 2: Pass Through Logic |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 43 | C_AXIS_TYPE | Integer | AXI4 Stream type<br>• 0: FIFO<br>• 1: Register Slice<br>• 2: Pass Through Logic |
| 44 | C_REG_SLICE_MODE_WACH | Integer | Write Address Channel configuration type<br>• 0: Fully Registered<br>• 1: Light Weight |
| 45 | C_REG_SLICE_MODE_WDCH | Integer | Write Data Channel configuration type<br>• 0: Fully Registered<br>• 1: Light Weight |
| 46 | C_REG_SLICE_MODE_WRCH | Integer | Write Response Channel configuration type<br>• 0: Fully Registered<br>• 1: Light Weight |
| 47 | C_REG_SLICE_MODE_RACH | Integer | Read Address Channel configuration type<br>• 0: Fully Registered<br>• 1: Light Weight |
| 48 | C_REG_SLICE_MODE_RDCH | Integer | Read Data Channel configuration type<br>• 0: Fully Registered<br>• 1: Light Weight |
| 49 | C_REG_SLICE_MODE_AXIS | Integer | AXI4 Stream configuration type<br>• 0: Fully Registered<br>• 1: Light Weight |
| 50 | C_IMPLEMENTATION_TYPE_WACH | Integer | Write Address Channel Implementation type<br>• 1: Common Clock Block RAM FIFO<br>• 2: Common Clock Distributed RAM FIFO<br>• 11: Independent Clock Block RAM FIFO<br>• 12: Independent Clock Distributed RAM FIFO |
| 51 | C_IMPLEMENTATION_TYPE_WDCH | Integer | Write Data Channel Implementation type<br>• 1: Common Clock Block RAM FIFO<br>• 2: Common Clock Distributed RAM FIFO<br>• 11: Independent Clock Block RAM FIFO<br>• 12: Independent Clock Distributed RAM FIFO |
| 52 | C_IMPLEMENTATION_TYPE_WRCH | Integer | Write Response Channel Implementation type<br>• 1: Common Clock Block RAM FIFO<br>• 2: Common Clock Distributed RAM FIFO<br>• 11: Independent Clock Block RAM FIFO<br>• 12: Independent Clock Distributed RAM FIFO |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 53 | C_IMPLEMENTATION_TYPE_RACH | Integer | Read Address Channel Implementation type<br>• 1: Common Clock Block RAM FIFO<br>• 2: Common Clock Distributed RAM FIFO<br>• 11: Independent Clock Block RAM FIFO<br>• 12: Independent Clock Distributed RAM FIFO |
| 54 | C_IMPLEMENTATION_TYPE_RDCH | Integer | Read Data Channel Implementation type<br>• 1: Common Clock Block RAM FIFO<br>• 2: Common Clock Distributed RAM FIFO<br>• 11: Independent Clock Block RAM FIFO<br>• 12: Independent Clock Distributed RAM FIFO |
| 55 | C_IMPLEMENTATION_TYPE_AXIS | Integer | AXI4 Stream Implementation type<br>• 1: Common Clock Block RAM FIFO<br>• 2: Common Clock Distributed RAM FIFO<br>• 11: Independent Clock Block RAM FIFO<br>• 12: Independent Clock Distributed RAM FIFO |
| 56 | C_APPLICATION_TYPE_WACH | Integer | Write Address Channel Application type<br>• 0: Data FIFO<br>• 1: Packet FIFO (c)<br>• 2: Low Latency Data FIFO |
| 57 | C_APPLICATION_TYPE_WDCH | Integer | Write Data Channel Application type<br>• 0: Data FIFO<br>• 1: Packet FIFO (c)<br>• 2: Low Latency Data FIFO |
| 58 | C_APPLICATION_TYPE_WRCH | Integer | Write Response Channel Application type<br>• 0: Data FIFO<br>• 1: Packet FIFO (c)<br>• 2: Low Latency Data FIFO |
| 59 | C_APPLICATION_TYPE_RACH | Integer | Read Address Channel Application type<br>• 0: Data FIFO<br>• 1: Packet FIFO (c)<br>• 2: Low Latency Data FIFO |
| 60 | C_APPLICATION_TYPE_RDCH | Integer | Read Data Channel Application type<br>• 0: Data FIFO<br>• 1: Packet FIFO (c)<br>• 2: Low Latency Data FIFO |
| 61 | C_APPLICATION_TYPE_AXIS | Integer | AXI4 Stream Application type<br>• 0: Data FIFO<br>• 1: Packet FIFO (c)<br>• 2: Low Latency Data FIFO |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 62 | C_USE_ECC_WACH | Integer | • 0: ECC option not used for Write Address Channel<br>• 1: ECC option used for Write Address Channel |
| 63 | C_USE_ECC_WDCH | Integer | • 0: ECC option not used for Write Data Channel<br>• 1: ECC option used for Write Data Channel |
| 64 | C_USE_ECC_WRCH | Integer | • 0: ECC option not used for Write Response Channel<br>• 1: ECC option used for Write Response Channel |
| 65 | C_USE_ECC_RACH | Integer | • 0: ECC option not used for Read Address Channel<br>• 1: ECC option used for Read Address Channel |
| 66 | C_USE_ECC_RDCH | Integer | • 0: ECC option not used for Read Data Channel<br>• 1: ECC option used for Read Data Channel |
| 67 | C_USE_ECC_AXIS | Integer | • 0: ECC option not used for AXI4 Stream<br>• 1: ECC option used for AXI4 Stream |
| 68 | C_ERROR_INJECTION_TYPE_WACH | Integer | ECC Error Injection type for Write Address Channel<br>• 0: No Error Injection<br>• 1: Single Bit Error Injection<br>• 2: Double Bit Error Injection<br>• 3: Single Bit and Double Bit Error Injection |
| 69 | C_ERROR_INJECTION_TYPE_WDCH | Integer | ECC Error Injection type for Write Data Channel<br>• 0: No Error Injection<br>• 1: Single Bit Error Injection<br>• 2: Double Bit Error Injection<br>• 3: Single Bit and Double Bit Error Injection |
| 70 | C_ERROR_INJECTION_TYPE_WRCH | Integer | ECC Error Injection type for Write Response Channel<br>• 0: No Error Injection<br>• 1: Single Bit Error Injection<br>• 2: Double Bit Error Injection<br>• 3: Single Bit and Double Bit Error Injection |
| 71 | C_ERROR_INJECTION_TYPE_RACH | Integer | ECC Error Injection type for Read Address Channel<br>• 0: No Error Injection<br>• 1: Single Bit Error Injection<br>• 2: Double Bit Error Injection<br>• 3: Single Bit and Double Bit Error Injection |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 72 | C_ERROR_INJECTION_TYPE_RDCH | Integer | ECC Error Injection type for Read Data Channel<br>• 0: No Error Injection<br>• 1: Single Bit Error Injection<br>• 2: Double Bit Error Injection<br>• 3: Single Bit and Double Bit Error Injection |
| 73 | C_ERROR_INJECTION_TYPE_AXIS | Integer | ECC Error Injection type for AXI4 Stream<br>• 0: No Error Injection<br>• 1: Single Bit Error Injection<br>• 2: Double Bit Error Injection<br>• 3: Single Bit and Double Bit Error Injection |
| 74 | C_DIN_WIDTH_WACH | Integer | DIN Width of Write Address Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID). |
| 75 | C_DIN_WIDTH_WDCH | Integer | DIN Width of Write Data Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID). |
| 76 | C_DIN_WIDTH_WRCH | Integer | DIN Width of Write Response Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID). |
| 77 | C_DIN_WIDTH_RACH | Integer | DIN Width of Read Address Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID). |
| 78 | C_DIN_WIDTH_RDCH | Integer | DIN Width of Read Data Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID). |
| 79 | C_DIN_WIDTH_AXIS | Integer | DIN Width of AXI4 Stream bus (1 - 1024)<br>Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID). |
| 80 | C_WR_DEPTH_WACH | Integer | FIFO Depth of Write Address Channel |
| 81 | C_WR_DEPTH_WDCH | Integer | FIFO Depth of Write Data Channel |
| 82 | C_WR_DEPTH_WRCH | Integer | FIFO Depth of Write Response Channel |
| 83 | C_WR_DEPTH_RACH | Integer | FIFO Depth of Read Address Channel |
| 84 | C_WR_DEPTH_RDCH | Integer | FIFO Depth of Read Data Channel |
| 85 | C_WR_DEPTH_AXIS | Integer | FIFO Depth of AXI4 Stream |
| 86 | C_WR_PNTR_WIDTH_WACH | Integer | $Log_2$(C_WR_DEPTH_WACH) |
| 87 | C_WR_PNTR_WIDTH_WDCH | Integer | $Log_2$(C_WR_DEPTH_WDCH) |
| 88 | C_WR_PNTR_WIDTH_WRCH | Integer | $Log_2$(C_WR_DEPTH_WRCH) |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 89 | C_WR_PNTR_WIDTH_RACH | Integer | $Log_2$(C_WR_DEPTH_RACH) |
| 90 | C_WR_PNTR_WIDTH_RDCH | Integer | $Log_2$(C_WR_DEPTH_RDCH) |
| 91 | C_WR_PNTR_WIDTH_AXIS | Integer | $Log_2$(C_WR_DEPTH_AXIS) |
| 92 | C_HAS_DATA_COUNTS_WACH | Integer | Write Address Channel<br>• 0: FIFO does not have Data Counts<br>• 1: FIFO has Data Count if C_COMMON_CLOCK = 1<br>Write/Read Data Count if C_COMMON_CLOCK = 0 |
| 93 | C_HAS_DATA_COUNTS_WDCH | Integer | Write Data Channel<br>• 0: FIFO does not have Data Counts<br>• 1: FIFO has Data Count if C_COMMON_CLOCK = 1<br>Write/Read Data Count if C_COMMON_CLOCK = 0 |
| 94 | C_HAS_DATA_COUNTS_WRCH | Integer | Write Response Channel<br>• 0: FIFO does not have Data Counts<br>• 1: FIFO has Data Count if C_COMMON_CLOCK = 1<br>Write/Read Data Count if C_COMMON_CLOCK = 0 |
| 95 | C_HAS_DATA_COUNTS_RACH | Integer | Read Address Channel<br>• 0: FIFO does not have Data Counts<br>• 1: FIFO has Data Count if C_COMMON_CLOCK = 1,<br>Write/Read Data Count if C_COMMON_CLOCK = 0 |
| 96 | C_HAS_DATA_COUNTS_RDCH | Integer | Read Data Channel<br>• 0: FIFO does not have Data Counts<br>• 1: FIFO has Data Count if C_COMMON_CLOCK = 1,<br>Write/Read Data Count if C_COMMON_CLOCK = 0 |
| 97 | C_HAS_DATA_COUNTS_AXIS | Integer | AXI4 Stream<br>• 0: FIFO does not have Data Counts<br>• 1: FIFO has Data Count if C_COMMON_CLOCK = 1,<br>Write/Read Data Count if C_COMMON_CLOCK = 0 |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 98 | C_HAS_PROG_FLAGS_WACH | Integer | Write Address Channel<br>• 0: FIFO does not have the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID<br>• 1: FIFO has the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID |
| 99 | C_HAS_PROG_FLAGS_WDCH | Integer | Write Data Channel<br>• 0: FIFO does not have the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID<br>• 1: FIFO has the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID |
| 100 | C_HAS_PROG_FLAGS_WRCH | Integer | Write Response Channel<br>• 0: FIFO does not have the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID<br>• 1: FIFO has the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID |
| 101 | C_HAS_PROG_FLAGS_RACH | Integer | Read Address Channel<br>• 0: FIFO does not have the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID<br>• 1: FIFO has the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID |
| 102 | C_HAS_PROG_FLAGS_RDCH | Integer | Read Data Channel<br>• 0: FIFO does not have the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID<br>• 1: FIFO has the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID |
| 103 | C_HAS_PROG_FLAGS_AXIS | Integer | AXI4 Stream<br>• 0: FIFO does not have the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID<br>• 1: FIFO has the option to map Almost Full/Empty or Programmable Full/Empty to READY/VALID |
| 104 | C_PROG_FULL_TYPE_WACH | Integer | Write Address Channel<br>• 1 or 3: PROG_FULL is mapped to READY<br>• 5: FULL is mapped to READY<br>• 6: ALMOST_FULL is mapped to READY |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 105 | C_PROG_FULL_TYPE_WDCH | Integer | Write Data Channel<br>• 1 or 3: PROG_FULL is mapped to READY<br>• 5: FULL is mapped to READY<br>• 6: ALMOST_FULL is mapped to READY |
| 106 | C_PROG_FULL_TYPE_WRCH | Integer | Write Response Channel<br>• 1 or 3: PROG_FULL is mapped to READY<br>• 5: FULL is mapped to READY<br>• 6: ALMOST_FULL is mapped to READY |
| 107 | C_PROG_FULL_TYPE_RACH | Integer | Read Address Channel<br>• 1 or 3: PROG_FULL is mapped to READY<br>• 5: FULL is mapped to READY<br>• 6: ALMOST_FULL is mapped to READY |
| 108 | C_PROG_FULL_TYPE_RDCH | Integer | Read Data Channel<br>• 1 or 3: PROG_FULL is mapped to READY<br>• 5: FULL is mapped to READY<br>• 6: ALMOST_FULL is mapped to READY |
| 109 | C_PROG_FULL_TYPE_AXIS | Integer | AXI4 Stream<br>• 1 or 3: PROG_FULL is mapped to READY<br>• 5: FULL is mapped to READY<br>• 6: ALMOST_FULL is mapped to READY |
| 110 | C_PROG_FULL_THRESH_ASSERT_VAL_WACH | Integer | PROG_FULL assert threshold[d] for Write Address Channel |
| 111 | C_PROG_FULL_THRESH_ASSERT_VAL_WDCH | Integer | PROG_FULL assert threshold[(d)] for Write Data Channel |
| 112 | C_PROG_FULL_THRESH_ASSERT_VAL_WRCH | Integer | PROG_FULL assert threshold[(d)] for Write Response Channel |
| 113 | C_PROG_FULL_THRESH_ASSERT_VAL_RACH | Integer | PROG_FULL assert threshold[(d)] for Read Address Channel |

*Table E-2:* **AXI4 SIM Parameters *(Cont'd)***

| | SIM Parameter | Type | Description |
|---|---|---|---|
| 114 | C_PROG_FULL_THRESH_ASSERT_VAL_RDCH | Integer | PROG_FULL assert threshold (d) for Read Data Channel |
| 115 | C_PROG_FULL_THRESH_ASSERT_VAL_AXIS | Integer | PROG_FULL assert threshold (d) for AXI4 Stream |
| 116 | C_PROG_EMPTY_TYPE_WACH | Integer | Write Address Channel<br>• 1 or 3: PROG_EMPTY is mapped to VALID<br>• 5: EMPTY is mapped to VALID<br>• 6: ALMOST_EMPTY is mapped to VALID |
| 117 | C_PROG_EMPTY_TYPE_WDCH | Integer | Write Data Channel<br>• 1 or 3: PROG_EMPTY is mapped to VALID<br>• 5: EMPTY is mapped to VALID<br>• 6: ALMOST_EMPTY is mapped to VALID |
| 118 | C_PROG_EMPTY_TYPE_WRCH | Integer | Write Response Channel<br>• 1 or 3: PROG_EMPTY is mapped to VALID<br>• 5: EMPTY is mapped to VALID<br>• 6: ALMOST_EMPTY is mapped to VALID |
| 119 | C_PROG_EMPTY_TYPE_RACH | Integer | Read Address Channel<br>• 1 or 3: PROG_EMPTY is mapped to VALID<br>• 5: EMPTY is mapped to VALID<br>• 6: ALMOST_EMPTY is mapped to VALID |
| 120 | C_PROG_EMPTY_TYPE_RDCH | Integer | Read Data Channel<br>• 1 or 3: PROG_EMPTY is mapped to VALID<br>• 5: EMPTY is mapped to VALID<br>• 6: ALMOST_EMPTY is mapped to VALID |
| 121 | C_PROG_EMPTY_TYPE_AXIS | Integer | AXI4 Stream<br>• 1 or 3: PROG_EMPTY is mapped to VALID<br>• 5: EMPTY is mapped to VALID<br>• 6: ALMOST_EMPTY is mapped to VALID |
| 122 | C_PROG_EMPTY_THRESH_ASSERT_VAL_WACH | Integer | PROG_EMPTY assert threshold for Write Address Channel (d). |
| 123 | C_PROG_EMPTY_THRESH_ASSERT_VAL_WDCH | Integer | PROG_EMPTY assert threshold for Write Data Channel (d). |
| 124 | C_PROG_EMPTY_THRESH_ASSERT_VAL_WRCH | Integer | PROG_EMPTY assert threshold for Write Response Channel (d). |
| 125 | C_PROG_EMPTY_THRESH_ASSERT_VAL_RACH | Integer | PROG_EMPTY assert threshold for Read Address Channel (d). |
| 126 | C_PROG_EMPTY_THRESH_ASSERT_VAL_RDCH | Integer | PROG_EMPTY assert threshold for Read Data Channel (d). |
| 127 | C_PROG_EMPTY_THRESH_ASSERT_VAL_AXIS | Integer | PROG_EMPTY assert threshold for AXI4 Stream (d). |

a. Includes Write Address Channel, Write Data Channel and Write Response Channel.
b. Includes Read Address Channel, Read Data Channel.

c. This feature is supported for Common Clock AXI4 and AXI4-Stream FIFOs only.
d. See the FIFO Generator GUI for the allowable range of values.

# Additional Resources

## Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see the Xilinx Support website at:

www.xilinx.com/support.

For a glossary of technical terms used in Xilinx documentation, see:

www.xilinx.com/company/terms.htm.

## References

These documents provide supplemental material useful with this product guide:

1. *AMBA® AXI4-Stream Protocol Specification*
2. *AXI4 AMBA® AXI Protocol Version: 2.0 Specification*
3. UG911, *Vivado Design Suite Migration Methodology Guide*
4. Vivado™ Design Suite Documentation

## Revision History

The following table shows the revision history for this document.

| Date | Version | Revision |
|---|---|---|
| 07/25/2012 | 1.0 | Initial release of this document as a product guide. This document replaces DS317, *FIFO Generator Data Sheet*, UG175, *FIFO Generator User Guide*, and XAPP992, *FIFO Generator Migration Guide*. |
| 10/16/2012 | 2.0 | Updated for core v9.3, Vivado Design Suite v2012.3, and ISE Design Suite v14.3. Clock Enable ports added for AXI4-Stream interface. |

| Date | Version | Revision |
|---|---|---|
| 12/18/2012 | 3.0 | Updated for core v9.3, Vivado Design Suite v2012.3, and ISE Design Suite v14.3. Added Appendix B, Debugging. |
| 03/20/2013 | 4.0 | • Updated for core v10.0.<br>• Removed support for ISE Design Suite.<br>• Added Embedded register support for AXI4-Stream packet FIFO.<br>• Updated the parameters and settings in Chapter 5, Customizing and Generating the Native Core. |

# Notice of Disclaimer