FIFO Generator v13.2

LogiCORE IP Product Guide

Vivado Design Suite

PG057 October 4, 2017

XILINX
ALL PROGRAMMABLE™

Table of Contents

Introduction

The Xilinx LogiCORE™ IP FIFO Generator core 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, you can customize the width, depth, status flags, memory type, and the write/read port aspect ratios.

The FIFO Generator core supports Native interface FIFOs, AXI Memory Mapped interface FIFOs and AXI4-Stream interface FIFOs. Native interface FIFO cores are optimized for buffering, data width conversion and clock domain decoupling applications, providing ordered storage and retrieval.

AXI Memory Mapped and AXI4-Stream interface FIFOs are derived from the Native interface FIFO. Three AXI Memory Mapped interface styles are available: AXI4, AXI3 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)	UltraScale+™ Families, UltraScale™ Architecture, Zynq®-7000, 7 Series
Supported User Interfaces	Native, AXI4-Stream, AXI4, AXI3, AXI4-Lite
Resources	Performance and Resource Utilization web page
Provided with Core	
Design Files	Encrypted RTL
Example Design	VHDL
Test Bench	VHDL
Constraints File	XDC
Simulation Model	Verilog Behavioral(2)
Supported S/W Driver	N/A
Tested Design Flows(4)	
Design Entry	Vivado Design Suite
Simulation(3)	For other supported simulators, see the Xilinx Design Tools: Release Notes Guide.
Synthesis	Vivado Synthesis
Support	
Provided by Xilinx at the Xilinx Support web page	

Notes:

1. For a complete listing of supported devices, see the Vivado IP catalog.
2. Behavioral model does not model synchronization delay. See Simulation in Chapter 4 for details.
3. The FIFO Generator core supports the UniSim simulation model.
4. 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 ordered 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 using minimum resources.

This core supports Native interface FIFOs, AXI Memory Mapped interface FIFOs and AXI4-Stream interface FIFOs. AXI Memory Mapped and AXI4-Stream interface FIFOs are derived from the Native interface FIFO. Three AXI Memory Mapped interface styles are available: AXI4, AXI3 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.

Note: The Memory Mapped interface FIFO and AXI4-Stream interface FIFO are referred as "AXI FIFO" throughout this document.

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.

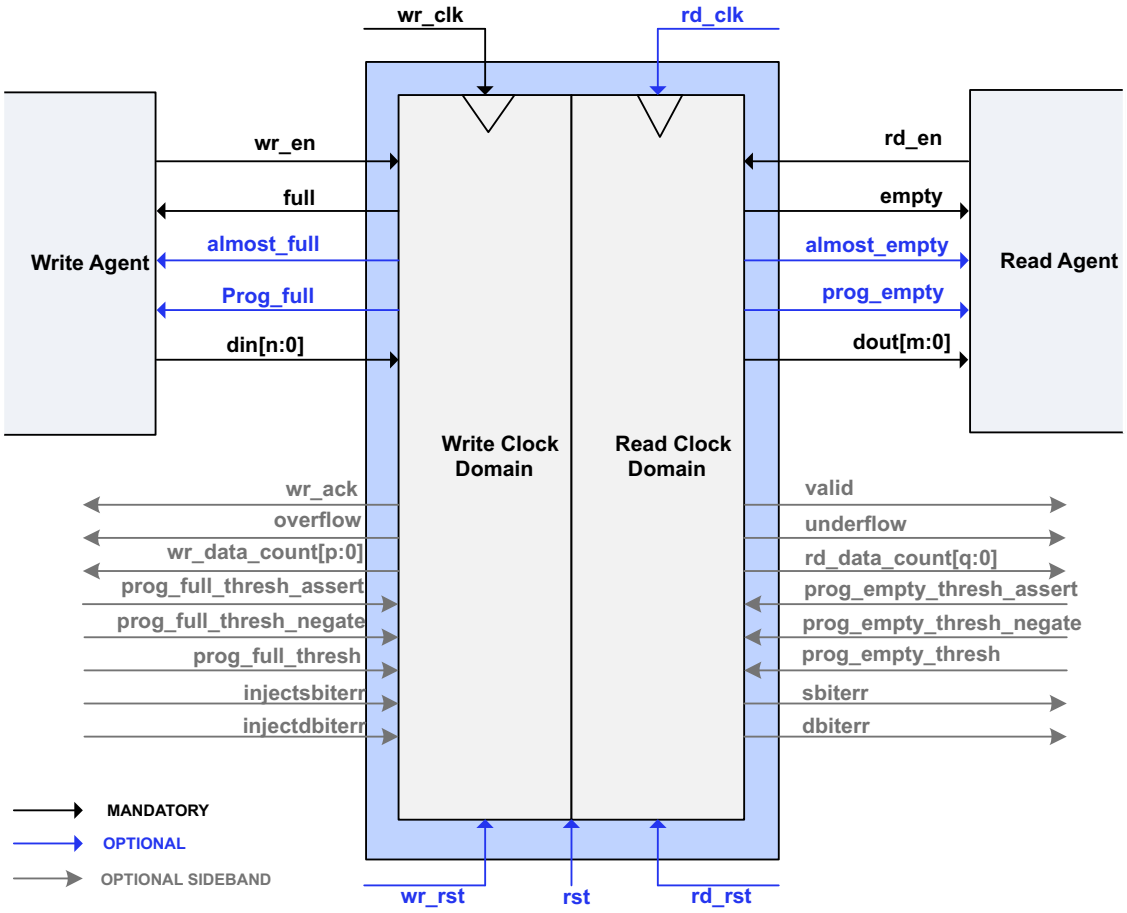


Figure 1-1: **Native Interface FIFOs Signal Diagram**

AXI Interface FIFOs

AXI interface FIFOs are derived from the Native interface FIFO, as shown in Figure 1-2. Three AXI memory mapped interface styles are available: AXI4, AXI3 and AXI4-Lite. In addition to applications supported by the Native interface FIFO, AXI FIFOs can also be used in AXI System Bus and Point-to-Point high speed applications.

AXI interface FIFOs do not support built-in FIFO and Shift Register FIFO configurations.

Use the AXI FIFOs in the same applications supported by the Native Interface FIFO when you need to connect to other AXI functions. AXI FIFOs can be integrated into a system by using the IP integrator. See the *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5] for more details.

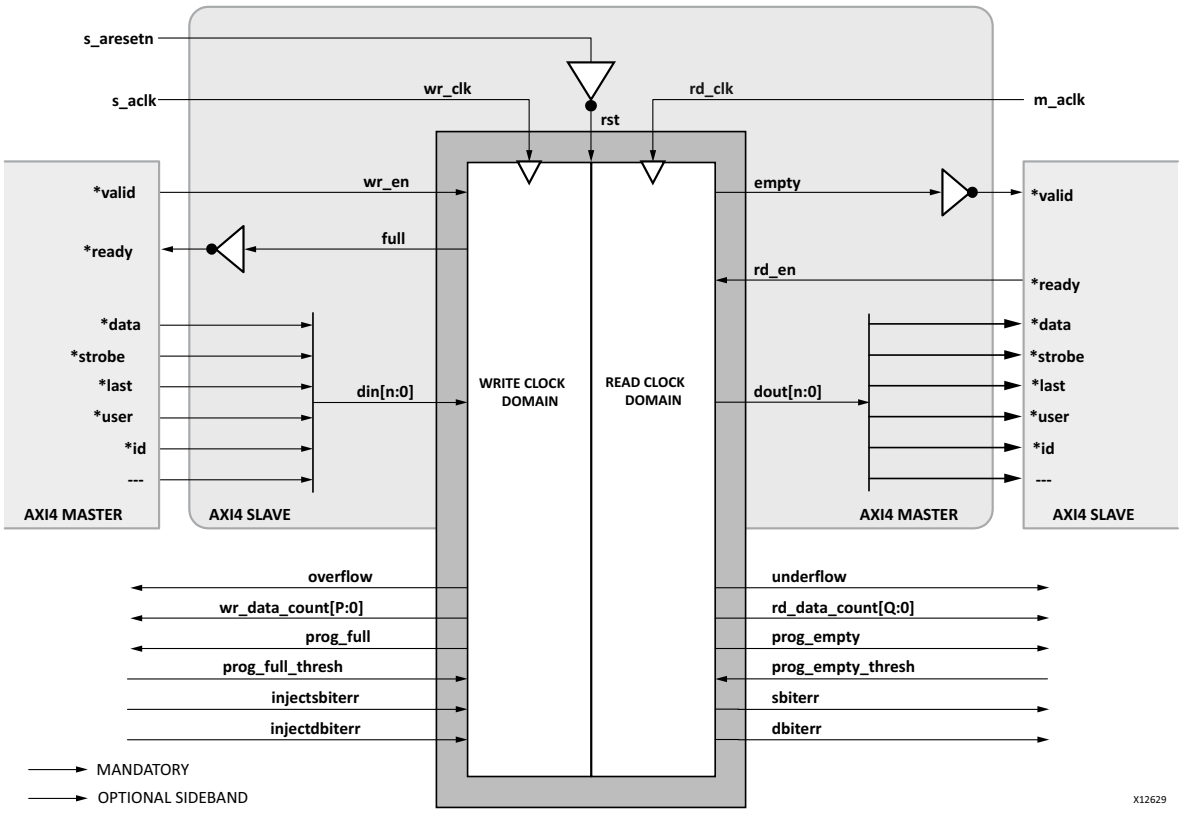


Figure 1-2: **AXI FIFO Derivation**

The AXI 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 AXI memory mapped interface FIFO.

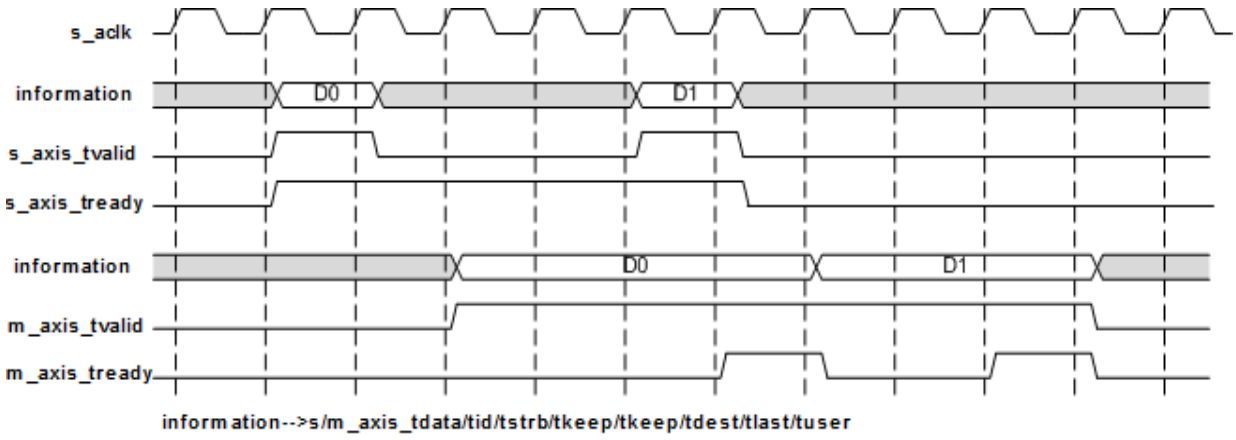


Figure 1-3: **AXI4-Stream FIFO Timing Diagram**

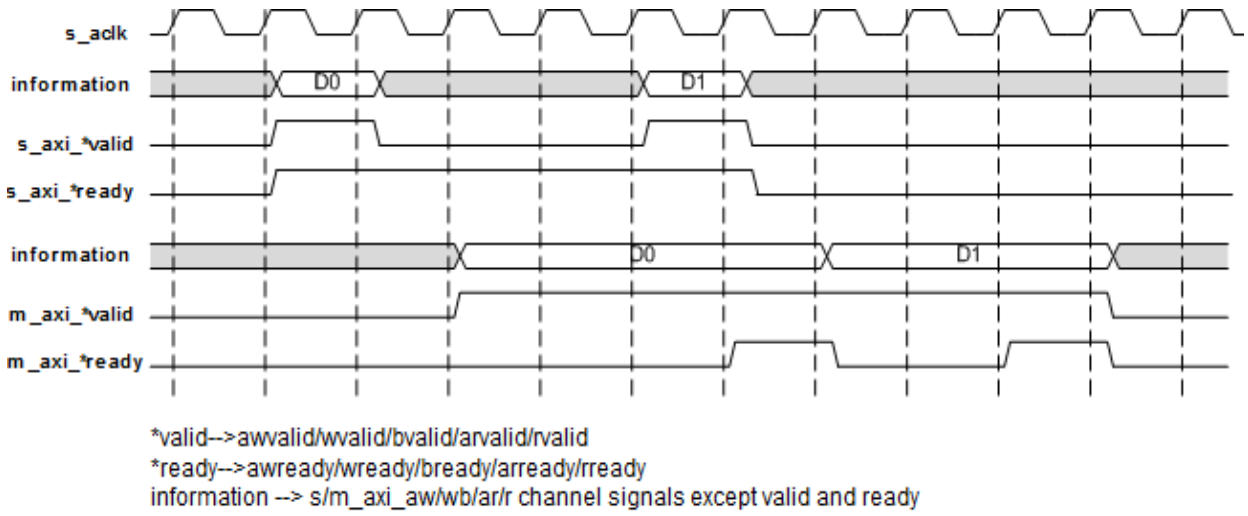


Figure 1-4: **AXI Memory Mapped Interface FIFO Timing Diagram**

In Figure 1-3 and Figure 1-4, 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 AXI 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 AXI FIFO is determined by concatenating all of the `information` signals of the AXI interface. The `information` signals include all AXI signals except for the `valid` and `ready` handshake signals.

AXI 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 data bus.

Note: For AXI interface, Safety Circuit is automatically enabled inside the core as the reset in is always asynchronous.

Feature Summary

Common Features

- Supports Native, AXI4-Stream, AXI4, AXI3 and AXI4-Lite interfaces
- FIFO depths up to 131,072 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
- Soft ECC support for block RAM FIFOs (upto 64-bit data widths)
- Embedded register option for block RAM and built-in FIFO configurations
- Dynamic Power Gating and ECC Pipeline Register support for UltraScale™ Architecture Built-in FIFO Configurations

AXI FIFO Features

- FIFO data widths:
  - AXI Stream: 1 to 4096 bits

  - AXI4/AXI3: 32, 64..... 1024 (multiples of 2) bits
  - AXI4-Lite: 32, 64 bits
- Supports AXI memory mapped and AXI4-Stream interface protocols - AXI4, AXI3, 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/independent clock AXI4-Stream FIFO and common clock AXI4/AXI3 FIFOs
- Operates in First-Word Fall-Through mode (FWFT)
- 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 core 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 core enables you to implement unique clock domains on the write and read ports. The FIFO Generator core 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 core can be used to generate a core optimized for that single clock.

Built-in FIFO Support

The FIFO Generator core supports the UltraScale, Zynq®-7000 and 7 series devices built-in FIFO macros, 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 using the FPGA logic to create optional status flags not implemented in the built-in FIFO macro. For example, programmable flags such as PROG_FULL and PROG_EMPTY are derived from ALMOSTFULL and ALMOSTEMPTY. The built-in Error Correction Checking (ECC) feature in

the built-in FIFO macro is also available. See the target device user guide for frequency requirements.

UltraScale architecture built-in FIFO supports only synchronous reset and comes with the following features:

- Non-symmetric aspect ratio
- Dynamic power gating
- ECC Pipeline register

Note: See the *Vivado Design Suite User Guide: Logic Simulation* for the limitation in simulation libraries for Built-in FIFO macros [Ref 9].

The FIFO Generator core offers a Low latency option for UltraScale devices to build a deeper FIFO where the latency of the FIFO is FIFO18E2/FIFO36E2 primitive latency if the **Output Register** option is not selected. The latency increases by one if **Output Register** is selected.

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 read data. 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 99).
- Programmable empty (see Non-symmetric Aspect Ratio and First-Word Fall-Through, page 116).
- Data counts (see First-Word Fall-Through Data Count, page 111 and Non-symmetric Aspect Ratio and First-Word Fall-Through, page 116).

Supported Memory Types

The FIFO Generator core 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 the following FIFO implementations:

- Common or Independent clock Block RAM FIFOs (UltraScale, Zynq-7000, and 7 series devices)
- Common or Independent clock Built-in FIFOs (UltraScale devices only)

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 117 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 Block RAM and 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	✓(2)	✓	✓	✓
Common	Distributed RAM		✓		

Table 1-2: **FIFO Configurations**

Clock Domain	Memory Type	Non-symmetric Aspect Ratios	First-word Fall-Through	ECC Support	Embedded Register Support
Common	Shift Register				
Common	Built-in FIFO	✓(2)	✓	✓	✓
Independent	Block RAM	✓	✓	✓	✓
Independent	Distributed RAM		✓		
Independent	Built-in FIFO	✓(2)	✓	✓	✓(1)

Notes:

1. Embedded register support for independent clock built-in FIFO is available only in Ultrascale family.
2. Xilinx supports Non-symmetric aspect ratio only for UltraScale devices. (The maximum depth is limited to 8192 in case of Built-In FIFOs)

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 non-symmetric aspect ratios (different write and read port widths), 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 option allows you to select the built-in FIFO and supports a common clock for write and read data accesses. The features supported for this configuration includes status flags (full and empty) and optional programmable full and empty flags with user-defined thresholds (for 7 series devices, programmable full/empty are directly connected to the ALMOSTFULL/ALMOSTEMPTY flags).

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 and UltraScale device-specific features such as non-symmetric aspect ratios (different write and read port widths), Dynamic Power Gating, and ECC Pipeline Register.

Independent Clocks: Block RAM and Distributed RAM

This implementation option 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 option 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 (for 7 series devices, programmable full/empty are directly connected to the ALMOSTFULL/ ALMOSTEMPTY flags). 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 and UltraScale device-specific features such as non-symmetric aspect ratios (different write and read port widths), Dynamic Power Gating, and ECC Pipeline Register.

Native FIFO Generator Feature Summary

FIFO Generator (from v13.1 onwards) is updated for UltraScale/UltraScale+ devices to utilize and improve the performance of the hardened cascading circuit for Built-in FIFO configurations. This update has significantly reduced the programmable full/empty threshold ranges of the Built-in FIFO structure. The valid range can be observed by opening up the XGUI and selecting the Built-in FIFO configurations. Programmable full calculations depend on the last primitive and programmable empty on the first primitive.

For example:

Until v13.0, FIFO Generator was using 2 FIFO macros as shown in Figure 1-5 for the configuration of 4096 deep and 18-bit wide. Thus, the programmable full/empty threshold ranges were as follows:

Programmable full= (1-4094)

Programmable empty= (2-4094) (both the primitives are 4kx9 primitives)

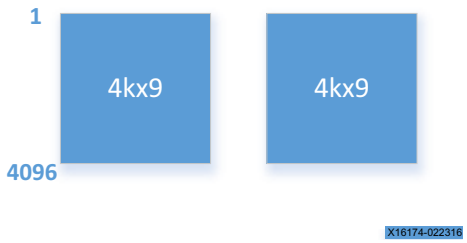


Figure 1-5: **Primitives Widthwise**

From v13.1 onwards, FIFO Generator uses 2 FIFO macros as shown in Figure 1-6 for the configuration of 4096 deep and 18-bit wide. Thus, the new programmable full/empty threshold ranges are as follows:

Programmable full= (2050-4094) (the last primitive)

Programmable empty=(2-2046) (the first primitive)

Figure 1-6: **Primitives Depth-wise**

Note: Due to the change mentioned above, there may be a change in the latency of write-to-empty and read-to-full de-assertion.

Table 1-3 summarizes the supported FIFO Generator core 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	✓		✓(1)	✓(1)		✓(1)
Symmetric Aspect Ratios	✓	✓	✓	✓	✓	✓
Almost Full	✓	✓		✓	✓	
Almost Empty	✓	✓		✓	✓	

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
Handshaking	✓	✓	✓	✓	✓	✓
Data Count	✓	✓		✓	✓	
Programmable Empty/Full Thresholds	✓	✓	✓(2)	✓	✓	✓(2)
First-Word Fall-Through	✓	✓	✓	✓	✓	✓
Synchronous Reset			✓(3)	✓	✓	✓(4)
Asynchronous Reset	✓(5)	✓(5)	✓(6)	✓(5)	✓(5)	✓(6)
dout Reset Value	✓	✓		✓	✓	✓(7)
ECC	✓		✓	✓		✓
Embedded or Interconnect Register	✓		✓(8)	✓		✓
Embedded and Interconnect Register	✓			✓		
ECC Pipeline Register			✓(9)			✓(9)
Dynamic Power Saving			✓(9)			✓(9)

Notes:

1. Xilinx supports Non-symmetric aspect ratio only for UltraScale devices. (The maximum depth is limited to 8192 in case of Built-In FIFOs).
2. For built-in FIFOs, the range of Programmable Empty/Full threshold is limited to take advantage of the logic internal to the macro.
3. Available only for UltraScale devices. The synchronous reset (srst) should be synchronous to wr_clk.
4. Available only for UltraScale devices. The synchronous reset (srst) should be synchronous to clk.
5. Asynchronous reset is optional for all FIFOs built using distributed and block RAM.
6. Asynchronous reset is not supported for UltraScale Built-in FIFOs.
7. dout Reset Value is supported only in common clock built-in FIFOs.
8. Embedded register option for independent clocks built-in FIFO is available only in UltraScale family. See Embedded Registers in Block RAM and FIFO Macros.
9. UltraScale devices only.

Using Block RAM FIFOs Versus Built-in FIFOs

The Built-In FIFO solutions are implemented to take advantage of logic internal to the Built-in FIFO macro. Several features, for example, almost full, almost empty, and so forth were not implemented because they are not native to the macro and require additional logic in the logic 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 1-7 illustrates these signals (both standard and optional ports) for a FIFO core that supports independent write and read clocks.

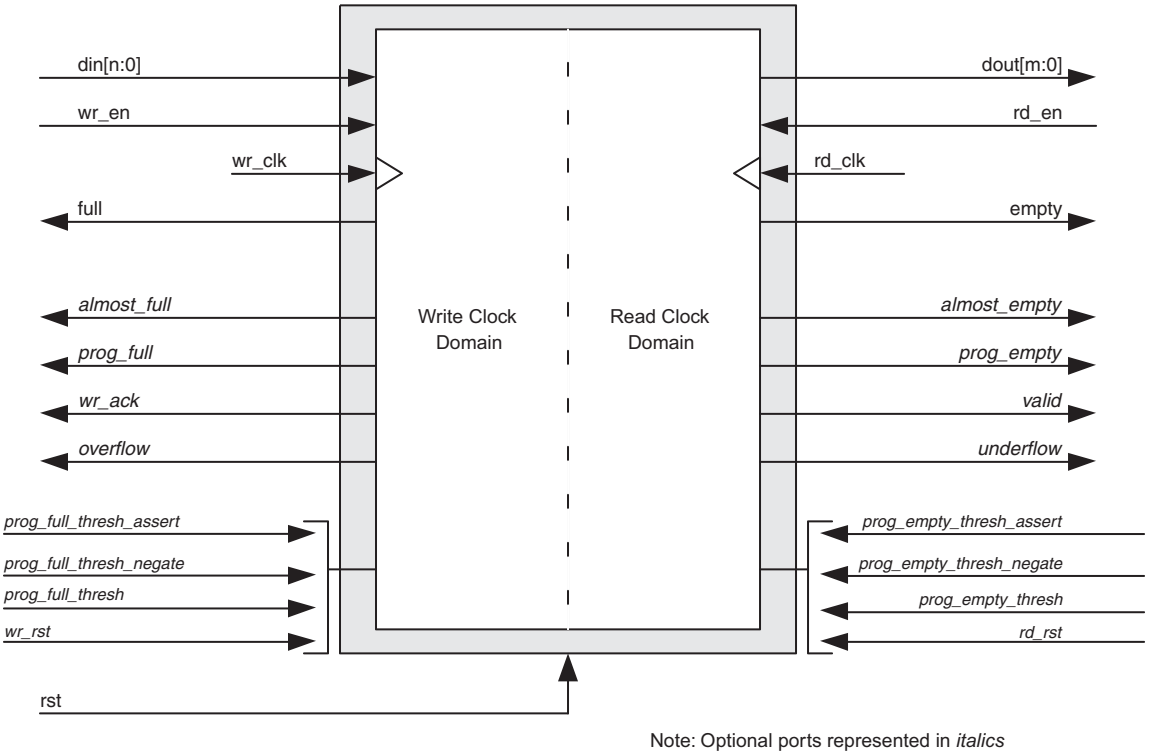


Figure 1-7: **FIFO with Independent Clocks: Interface Signals**

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 and Sleep Signals for FIFOs with Independent Clocks**

Name	Direction	Description
rst	Input	Reset: An asynchronous reset signal that initializes all internal pointers and output registers. Not available for UltraScale device built-in FIFOs.
sleep	Input	Dynamic power gating. If sleep is active, the FIFO is in power saving mode. ***Note:*** Only available for UltraScale device built-in FIFOs.

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.
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(1)	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.

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

Name	Direction	Description
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/clk, that write operation will only be reflected on wr_data_count at the next rising clock edge. If D is less than log2(FIFO depth)-1, the bus is truncated by removing the least-significant bits. ***Note:*** wr_data_count is also available for UltraScale devices using a common clock Block RAM-based FIFO when the Asymmetric Port Width option is enabled.
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. 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 Vivado IDE for the valid range of values(1).
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 Vivado IDE for the valid range of values(2).
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.
wr_rst_busy	Output	When asserted, this signal indicates that the write domain is in reset state. ***Note:*** Available only for UltraScale device built-in FIFOS.

Notes:

1. For 7 series devices using the Built-in FIFO configuration, this signal is connected to the almostfull signal of the FIFO18E1/FIFO36E1 primitive.
2. Valid range of values shown in the IDE 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(1)	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.
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/clk, that read operation is only reflected on rd_data_count at the next rising clock edge. If C is less than log2(FIFO depth)-1, the bus is truncated by removing the least-significant bits. ***Note:*** rd_data_count is also available for UltraScale devices using a common clock Block RAM-based FIFO when the Asymmetric Port Width option is enabled.
valid	Output	Valid: This signal indicates that valid data is available on the output bus (dout).

Table 1-6: **Read Interface Signals for FIFOs with Independent Clocks *(Cont'd)***

Name	Direction	Description
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. 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 Vivado IDE for the valid range of values(2).
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 Vivado IDE for the valid range of values(2).
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.
rd_rst_busy	Output	When asserted, this signal indicates that the read domain is in reset state. ***Note:*** Available only for UltraScale device built-in FIFOS.

Notes:

1. For 7 series devices using the Built-in FIFO configuration, this signal is connected to the almostfull signal of the FIFO18E1/FIFO36E1 primitive.
2. Valid range of values shown in the IDE 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. Not available for UltraScale device built-in FIFOs.
srst	Input	Synchronous Reset: A synchronous reset that initializes all internal pointers and output registers. The FIFO uses UltraScale device reset sequence mechanism for UltraScale devices.
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(1)	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.

Table 1-7: **Interface Signals for FIFOs with a Common Clock *(Cont'd)***

Name	Direction	Description
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. 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 Vivado IDE for the valid range of values(2).
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 Vivado IDE for the valid range of values(2).
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(3)	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.

Table 1-7: **Interface Signals for FIFOs with a Common Clock *(Cont'd)***

Name	Direction	Description
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. 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."
sleep	Input	Dynamic shutdown power saving. If sleep is active, the FIFO is in power saving mode. ***Note:*** Available only for UltraScale device built-in FIFOs.
wr_rst_busy	Output	When asserted, this signal indicates that the write domain is in reset state. ***Note:*** Available only for UltraScale device built-in FIFOs, and Common Clock Block RAM/Distributed RAM/Shift Register FIFOs with synchronous reset.

Table 1-7: **Interface Signals for FIFOs with a Common Clock *(Cont'd)***

Name	Direction	Description
rd_rst_busy	Output	When asserted, this signal indicates that the read domain is in reset state. ***Note:*** Available only for UltraScale device built-in FIFOs, and Common Clock Block RAM/Distributed RAM/Shift Register FIFOs with synchronous reset.

Notes:

1. For 7 series devices using the Built-in FIFO configuration, this signal is connected to the almostfull signal of the FIFO18E1/FIFO36E1 primitive.
2. Valid range of values shown in the IDE are the actual values even though they are grayed out for some selections.
3. For 7 series devices using the Built-in FIFO configuration, this signal is connected to the almostempty signal of the FIFO18E1/FIFO36E1 primitive.

AXI FIFO Feature Overview

Easy Integration of Independent FIFOs for Read and Write Channels

For AXI memory mapped interfaces, AXI 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 core provides the ability to generate either Write Channels or Read Channels, or both Write Channels and Read Channels for AXI memory mapped. 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 core integrates five independent FIFOs.

For AXI memory mapped interfaces, the FIFO Generator core provides the ability to implement independent FIFOs for each channel, as shown in Figure 1-8. For each channel, the core can be independently configured to generate a block RAM or distributed memory or built-in based FIFO. The depth of each FIFO can also be independently configured.

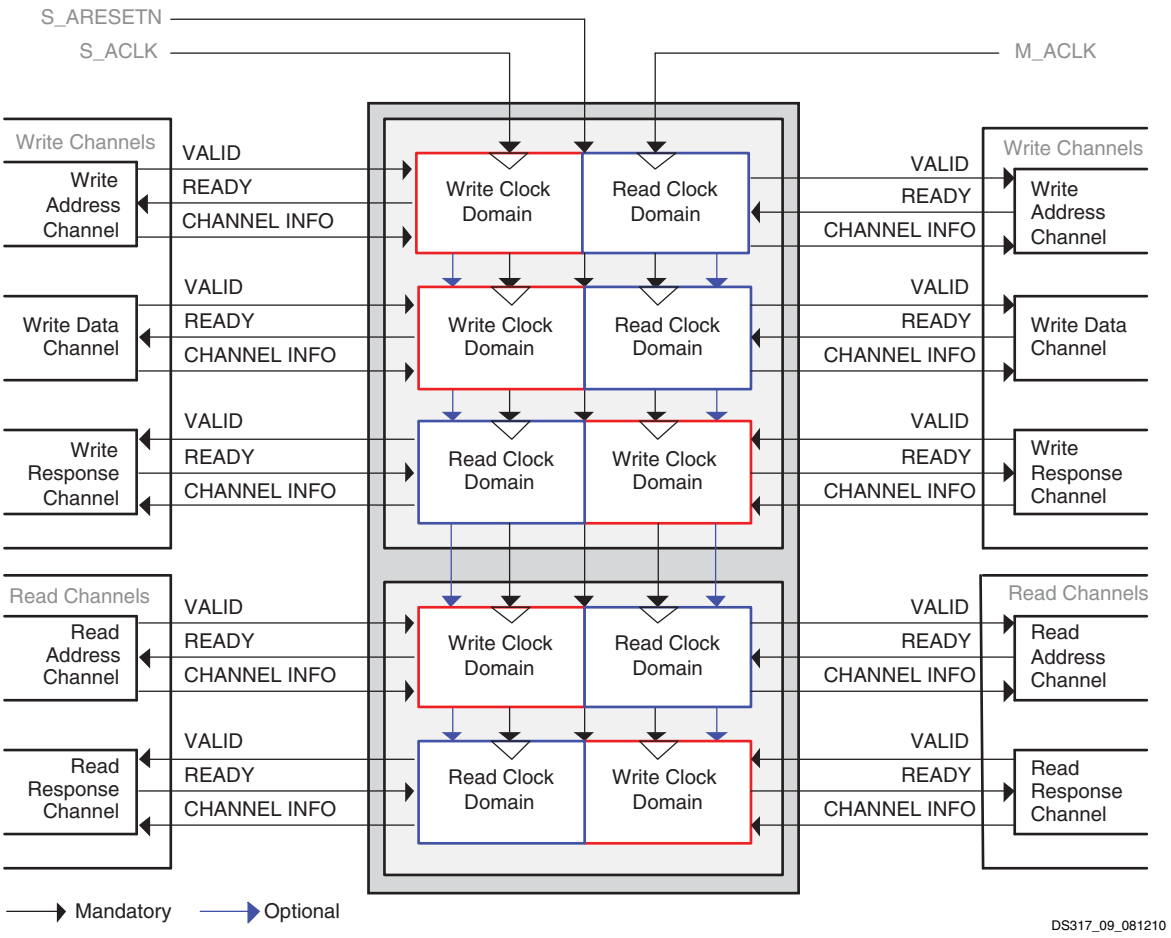


Figure 1-8: **AXI Memory Mapped FIFO Block Diagram**

Clock and Reset Implementation and Operation

For the AXI4-Stream and AXI memory mapped interfaces, all instantiated FIFOs share clock and asynchronous active-Low reset signals (Figure 1-8). In addition, all instantiated FIFOs can support either independent clock or common clock operation.

The independent clock configuration of the FIFO Generator core enables you to implement unique clock domains on the write and read ports. The FIFO Generator core 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 core can be used to generate a core optimized for a single clock by selecting the common clock option.

Automatic FIFO Width Calculation

AXI FIFOs support symmetric widths for the FIFO Read and Write ports. The FIFO width for the AXI FIFO is determined by the selected interface type (AXI4-Stream or AXI memory mapped) and user-selected signals and signal widths within the given interface. The AXI

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 core 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.

Register slices

Each AXI channel transfers information in only one direction, and there is no requirement for a fixed relationship between the various channels. This enables the insertion of a register slice in any channel, at the cost of an additional cycle of latency, but providing maximum frequency of operation.

The core provides two register slice options: fully registered (two stage pipeline register) and light weight (one stage pipeline register).

Pass Through Wire

The core offers the pass through wire option for the AXI memory mapped interface making all input signals pass through to output.

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/AXI3 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/AXI3 and Common/ Independent Clock AXI4-Stream configurations. It is not supported for AXI4-Lite configurations.

AXI4-Stream Packet FIFO

The FIFO Generator core uses AXI4-Stream Interface for the AXI4-Stream Packet FIFO feature. The FIFO Generator core 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.

AXI4/AXI3 Packet FIFO

The FIFO Generator core uses the AXI memory mapped interface for the AXI4/AXI3 Packet FIFO feature (for both write and read channels).

- **Packet FIFO on Write Channels**: The FIFO Generator core indicates an `awvalid` on the AXI AW channel Master side when a complete packet (marked by `wlast`) is received on the AXI 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 AXI Write Address Channel until all of the data needed for the requested transfer is received on the AXI 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 core indicates an `rvalid` on the AXI R channel Slave side when a complete packet (marked by `rlast`) is received on the AXI 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 AXI 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.

Low Latency FIFO

The core offers the Low Latency FIFO option for the memory mapped and AXI4-Stream interfaces in common clock mode of operation. In this mode, the latency is 1.

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.

AXI Slave Interface for Performing Writes

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

AXI Master Interface for Performing Reads

The AXI FIFO provides an AXI Master interface for performing Reads. In Figure 1-4, the AXI FIFO provides INFORMATION and VALID signals; upon detecting a READY signal asserted from the AXI Slave interface, the AXI FIFO will place the next INFORMATION on the bus. The VALID signal is de-asserted only when the FIFO is empty.

AXI FIFO Feature Summary

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

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

FIFO Options	Common Clock		Independent Clock	
	Block RAM/ Built-in	Distributed Memory	Block RAM/Built-in	Distributed Memory
Full(1)	✓	✓	✓	✓
Programmable Full(2)	✓	✓	✓	✓
Empty(3)	✓	✓	✓	✓
Programmable Empty(2)	✓	✓	✓	✓
Data Counts(4)	✓	✓	✓	✓
ECC	✓		✓	

Notes:

1. 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 IDE.
2. Provided as sideband signal depending on the IDE option.
3. 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 IDE.
4. Built-in FIFO does not support data counts feature.

AXI FIFO Interface Signals

The following sections define the AXI 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 outside **Reset Window**. To avoid unexpected behavior, do not perform any transactions during **Reset Window**.

Note: **Reset Window:** reset duration + 60 slowest clock cycles.

Global Signals

Table 1-9 defines the global interface signals for AXI 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: **AXI 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 AXI 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		
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		

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

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

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

Name	Direction	Description
m_axis_tkeep[m/8-1:0]	Output	TKEEP: The byte qualifier that indicates whether the content of the associated byte of TDATA is processed as part of the data stream. Associated bytes that have the TKEEP byte qualifier deasserted are null bytes and can be removed from the data stream. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • KEEP[0] = 1b, DATA[7:0] is a NULL byte • KEEP [7] = 0b, DATA[63:56] is not a NULL byte
m_axis_tlast	Output	TLAST: Indicates the boundary of a packet.
m_axis_tid[m:0]	Output	TID: The data stream identifier that indicates different streams of data.
m_axis_tdest[m:0]	Output	TDEST. Provides routing information for the data stream.
m_axis_tuser[m:0]	Output	TUSER: The user-defined sideband information that can be transmitted alongside the data stream.
AXI4-Stream FIFO: Optional Sideband Signals		
axis_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. *D* = log2(FIFO depth)-1
axis_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. *D* = log2(FIFO depth)-1
axis_injectsbiterr	Input	Inject Single-Bit Error: Injects a single-bit error if the ECC feature is used.
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. **Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full.

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

Name	Direction	Description
axis_wr_data_count[*d*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure 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. $D = \log_2(\text{FIFO depth})+1$
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. **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. $D = \log_2(\text{FIFO depth})+1$
axis_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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/AXI3 FIFO Interface Signals

Write Channels

Table 1-11 defines the AXI4/AXI3 FIFO interface signals for Write Address Channel.

Table 1-11: **AXI4/AXI3 Write Address Channel FIFO Interface Signals**

Name	Direction	Description
AXI4/AXI3 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](1)	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[1:0](2)	Input	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.
s_axi_awcache[3:0]	Input	Cache Type: Indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction.
s_axi_awprot[2:0]	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](3)	Input	Region Identifier: Sent on the write address channel for each write transaction.
s_axi_awuser[m:0]	Input	Write Address Channel User
AXI4/AXI3 Interface Write Address Channel: Handshake Signals for FIFO Write Interface		

Table 1-11: **AXI4/AXI3 Write Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_awvalid	Input	Write Address Valid: Indicates that valid write address and control information are available: • 1 = Address and control information available. • 0 = Address and control information not available. The address and control information remain stable until the address acknowledge signal, awready, goes High.
s_axi_awready	Output	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4/AXI3 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](1)	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[1:0](2)	Output	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.
m_axi_awcache[3:0]	Output	Cache Type: This signal indicates the bufferable, cacheable, write-through, write-back, and allocate attributes of the transaction.
m_axi_awprot[2:0]	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](3)	Output	Region Identifier: Sent on the write address channel for each write transaction.
m_axi_awuser[m:0]	Output	Write Address Channel User

Table 1-11: **AXI4/AXI3 Write Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI4/AXI3 Interface Write Address Channel: Handshake Signals for FIFO Read Interface		
m_axi_awvalid	Output	Write Address Valid: Indicates that valid write address and control information are available: • 1 = address and control information available • 0 = address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes high.
m_axi_awready	Input	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4/AXI3 Write Address Channel FIFO: Optional Sideband Signals		
axi_aw_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (prog_full) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
axi_aw_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
axi_aw_injectsbiterr	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used.
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. **Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full.

Table 1-11: **AXI4/AXI3 Write Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
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. $D = \log_2(\text{FIFO depth})+1$
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. **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. $D = \log_2(\text{FIFO depth})+1$
axi_aw_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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.

Notes:

1. *_awlen port width is 8 for AXI4 and 4 for AXI3.
2. *_awlock port width is 1 for AXI4 and 2 for AXI3.
3. Port not available for AXI3.

Table 1-12 defines the AXI4/AXI3 FIFO interface signals for Write Data Channel.

Table 1-12: **AXI4/AXI3 Write Data Channel FIFO Interface Signals**

Name	Direction	Description
AXI4/AXI3 Interface Write Data Channel: Information Signals mapped to FIFO Data Input (din) Bus		
s_axi_wid[m:0](1)	Input	Write ID Tag: This signal is the ID tag of the write data transfer. The WID value must match the AWID value of the write transaction.
s_axi_wdata[m-1:0]	Input	Write Data: The write data bus can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
s_axi_wstrb[m/8-1:0]	Input	Write Strobes: Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)]. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid
s_axi_wlast	Input	Write Last: Indicates the last transfer in a write burst.
s_axi_wuser[m:0]	Input	Write Data Channel User
AXI4/AXI3 Interface Write Data Channel: Handshake Signals for FIFO Write Interface		
s_axi_wvalid	Input	Write Valid: Indicates that valid write data and strobes are available: • 1 = Write data and strobes available. • 0 = Write data and strobes not available.
s_axi_wready	Output	Write Ready: Indicates that the slave can accept the write data: • 1 = Slave ready. • 0 = Slave not ready.
AXI4/AXI3 Interface Write Data Channel: Information Signals Derived from FIFO Data Output (dout) Bus		
m_axi_wid[m:0](1)	Output	Write ID Tag: This signal is the ID tag of the write data transfer. The WID value must match the AWID value of the write transaction.
m_axi_wdata[m-1:0]	Output	Write Data: The write data bus can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
m_axi_wstrb[m/8-1:0]	Output	Write Strobes: Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)]. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid

Table 1-12: **AXI4/AXI3 Write Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
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/AXI3 Interface Write Data Channel: Handshake Signals for FIFO Read Interface		
m_axi_wvalid	Output	Write valid: Indicates that valid write data and strobes are available: • 1 = Write data and strobes available . • 0 = Write data and strobes not available.
m_axi_wready	Input	Write ready: Indicates that the slave can accept the write data: • 1 = Slave ready. • 0 = Slave not ready.
AXI4/AXI3 Write Data Channel FIFO: Optional Sideband Signals		
axi_w_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (prog_full) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
axi_w_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
axi_w_injectsbiterr	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used.
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. **Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full.

Table 1-12: **AXI4/AXI3 Write Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
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. $D = \log_2(\text{FIFO depth})+1$
axi_w_underflow	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO
axi_w_rd_data_count[*d*:0]	Output	Read Data Count: This bus indicates the number of words available for reading in the FIFO. The count is guaranteed to never over-report the number of words available for reading, to ensure that 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. $D = \log_2(\text{FIFO depth})+1$
axi_w_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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.

Notes:

1. Port not available for AXI4.

Table 1-13 defines the AXI4/AXI3 FIFO interface signals for Write Response Channel.

Table 1-13: **AXI4/AXI3 Write Response Channel FIFO Interface Signals**

Name	Direction	Description
AXI4/AXI3 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/AXI3 Interface Write Response Channel: Handshake Signals for FIFO Read Interface		
s_axi_bvalid	Output	Write Response Valid: Indicates that a valid write response is available: • 1 = Write response available. • 0 = Write response not available.
s_axi_bready	Input	Response Ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.
AXI4/AXI3 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/AXI3 Interface Write Response Channel: Handshake Signals for FIFO Write Interface		
m_axi_bvalid	Input	Write Response Valid: Indicates that a valid write response is available: • 1 = Write response available. • 0 = Write response not available.
m_axi_bready	Output	Response Ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.
AXI4/AXI3 Write Response Channel FIFO: Optional Sideband Signals		

Table 1-13: **AXI4/AXI3 Write Response Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
axi_b_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (prog_full) flag. The threshold can be dynamically set in-circuit during reset. *D* = $\log_2$(FIFO depth)-1
axi_b_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. *D* = $\log_2$(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. **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. *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. **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/AXI3 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. $D = \log_2(\text{FIFO depth})+1$
axi_b_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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/AXI3 FIFO interface signals for Read Address Channel.

Table 1-14: **AXI4/AXI3 Read Address Channel FIFO Interface Signals**

Name	Direction	Description
AXI4/AXI3 Interface Read Address Channel: Information Signals Mapped to FIFO Data Input (din) Bus		
s_axi_arid[m:0]	Input	Read Address ID: This signal is the identification tag for the read address group of signals.
s_axi_araddr[m:0]	Input	Read Address: The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
s_axi_arlen[7:0](1)	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.

Table 1-14: **AXI4/AXI3 Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_arburst[1:0]	Input	Burst Type: The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.
s_axi_arlock[1:0](2)	Input	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.
s_axi_arcache[3:0]	Input	Cache Type: This signal provides additional information about the cacheable characteristics of the transfer.
s_axi_arprot[2:0]	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](3)	Input	Region Identifier: Sent on the read address channel for each read transaction.
s_axi_aruser[m:0]	Input	Read Address Channel User
AXI4/AXI3 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, aready, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
s_axi_arready	Output	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4/AXI3 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. 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](1)	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.

Table 1-14: **AXI4/AXI3 Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
m_axi_arlock[1:0](2)	Output	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.
m_axi_arcache[3:0]	Output	Cache Type: This signal provides additional information about the cacheable characteristics of the transfer.
m_axi_arprot[2:0]	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](3)	Output	Region Identifier: Sent on the read address channel for each read transaction.
m_axi_aruser[m:0]	Output	Read Address Channel User
AXI4/AXI3 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, aready, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
m_axi_arready	Input	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4/AXI3 Read Address Channel FIFO: Optional Sideband Signals		
axi_ar_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (prog_full) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
axi_ar_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
axi_ar_injectsbiterr	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used.
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.

Table 1-14: **AXI4/AXI3 Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
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. **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. $D = \log_2(\text{FIFO depth})+1$
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. **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. $D = \log_2(\text{FIFO depth})+1$
axi_ar_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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.

Table 1-14: **AXI4/AXI3 Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
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.

Notes:

1. *_arlen port width is 8 for AXI4 and 4 for AXI3.
2. *_arlock port width is 1 for AXI4 and 2 for AXI3.
3. Port not available for AXI3.

Table 1-15 defines the AXI4/AXI3 FIFO interface signals for Read Data Channel.

Table 1-15: **AXI4/AXI3 Read Data Channel FIFO Interface Signals**

Name	Direction	Description
AXI4/AXI3 Interface Read Data Channel: Information Signals Mapped to FIFO Data Output (dout) Bus		
s_axi_rid[m:0]	Output	Read ID Tag: ID tag of the read data group of signals. The RID value is generated by the slave and must match the ARID value of the read transaction to which it is responding.
s_axi_rdata[m-1:0]	Output	Read Data: Can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
s_axi_rresp[1:0]	Output	Read Response: Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
s_axi_rlast	Output	Read Last: Indicates the last transfer in a read burst.
s_axi_ruser[m:0]	Output	Read Data Channel User
AXI4/AXI3 Interface Read Data Channel: Handshake Signals for FIFO Read Interface		
s_axi_rvalid	Output	Read Valid: Indicates that the required read data is available and the read transfer can complete: • 1 = Read data available. • 0 = Read data not available.
s_axi_rready	Input	Read Ready: Indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.
AXI4/AXI3 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.

Table 1-15: **AXI4/AXI3 Read Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
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/AXI3 Interface Read Data Channel: Handshake Signals for FIFO Write Interface		
m_axi_rvalid	Input	Read Valid: Indicates that the required read data is available and the read transfer can complete: • 1 = Read data available. • 0 = Read data not available.
m_axi_rready	Output	Read Ready: Indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.
AXI4/AXI3 Read Data Channel FIFO: Optional Sideband Signals		
axi_r_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. *D* = $\log_2$(FIFO depth)-1
axi_r_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. *D* = $\log_2$(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. **Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full.

Table 1-15: **AXI4/AXI3 Read Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
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. $D = \log_2(\text{FIFO depth})+1$
axi_r_underflow	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO. **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. $D = \log_2(\text{FIFO depth})+1$
axi_r_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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: • 1 = Address and control information available. • 0 = Address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes high.
s_axi_awready	Output	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Interface Write Address Channel: Information Signals Derived from FIFO Data Output (dout) Bus		
m_axi_awaddr[m:0]	Output	Write Address: Gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
m_axi_awprot[3:0]	Output	Protection Type: This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
AXI4-Lite Interface Write Address Channel: Handshake Signals for FIFO Read Interface		

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

Name	Direction	Description
m_axi_awvalid	Output	Write Address Valid: Indicates that valid write address and control information are available: • 1 = Address and control information available. • 0 = Address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes high.
m_axi_awready	Input	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Write Address Channel FIFO: Optional Sideband Signals		
axi_aw_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (prog_full) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
axi_aw_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
axi_aw_injectsbiterr	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used.
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. **Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full.

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

Name	Direction	Description
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. $D = \log_2(\text{FIFO depth})+1$
axi_aw_underflow	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO. **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. $D = \log_2(\text{FIFO depth})+1$
axi_aw_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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.

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

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

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

Name	Direction	Description
axi_w_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. D = $\log_2$(FIFO depth)-1
axi_w_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. D = $\log_2$(FIFO depth)-1
axi_w_injectsbiterr	Input	Injects a single bit error if the ECC feature is used.
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.
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. **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. 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. **Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty.

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

Name	Direction	Description
axi_w_rd_data_count[*d*:0]	Output	Read Data Count: This bus indicates the number of words available for reading in the FIFO. The count is guaranteed to never over-report the number of words available for reading, to ensure that 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. $D = \log_2(\text{FIFO depth})+1$
axi_w_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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: • 1 = Write response available. • 0 = Write response not available.
s_axi_bready	Input	Response Ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.

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

Name	Direction	Description
AXI4-Lite Interface Write Response Channel: Information Signals Derived from FIFO Data Input (din) Bus		
m_axi_bresp[1:0]	Input	Write response: Indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
AXI4-Lite Interface Write Response Channel: Handshake Signals for FIFO Write Interface		
m_axi_bvalid	Input	Write response valid: Indicates that a valid write response is available: • 1 = Write response available. • 0 = Write response not available.
m_axi_bready	Output	Response ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.
AXI4-Lite Write Response Channel FIFO: Optional Sideband Signals		
axi_b_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (prog_full) flag. The threshold can be dynamically set in-circuit during reset. *D* = $\log_2$(FIFO depth)-1
axi_b_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. D is than log2(FIFO depth)-1
axi_b_injectsbiterr	Input	Injects a single bit error if the ECC feature is used.
axi_b_injectdbiterr	Input	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. **Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full.

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

Name	Direction	Description
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. $D = \log_2(\text{FIFO depth})+1$
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. **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. $D = \log_2(\text{FIFO depth})+1$
axi_b_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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, aready, is High. • 1 = Address and control information valid. • 0 = Address and control information not valid.
s_axi_aready	Output	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Interface Read Address Channel: Information Signals Derived from FIFO Data Output (dout) Bus		
m_axi_araddr[m:0]	Output	Read Address: The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
m_axi_arprot[3:0]	Output	Protection Type: This signal provides protection unit information for the transaction.
AXI4-Lite Interface Read Address Channel: Handshake Signals for FIFO Read Interface		
m_axi_arvalid	Output	Read Address Valid: WHen high, indicates that the read address and control information is valid and will remain stable until the address acknowledge signal, aready, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.

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

Name	Direction	Description
m_axi_arready	Input	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Read Address Channel FIFO: Optional Sideband Signals		
axi_ar_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (prog_full) flag. The threshold can be dynamically set in-circuit during reset. *D* = $\log_2$(FIFO depth)-1
axi_ar_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. *D* = $\log_2$(FIFO depth)-1
axi_ar_injectsbiterr	Input	Inject Single-Bit Error: Injects a single-bit error if the ECC feature is used.
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. **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. *D* = $\log_2$(FIFO depth)+1

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

Name	Direction	Description
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. **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. $D = \log_2(\text{FIFO depth})+1$
axi_ar_data_count[d:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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: • 1 = Read data available. • 0 = Read data not available.

Table 1-20: **AXI4-Lite Read Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
s_axi_rready	Input	Read Ready: indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.
AXI4-Lite Interface Read Data Channel: Information Signals Derived from FIFO Data Input (din) Bus		
m_axi_rdata[m-1:0]	Input	Read Data: The read data bus can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
m_axi_ rresp[1:0]	Input	Read Response: Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
AXI4-Lite Interface Read Data Channel: Handshake Signals for FIFO Write Interface		
m_axi_rvalid	Input	Read Valid: Indicates that the required read data is available and the read transfer can complete: • 1 = Read data available. • 0 = Read data not available.
m_axi_rready	Output	Read ready: Indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.
AXI4-Lite Read Data Channel FIFO: Optional Sideband Signals		
axi_r_prog_full_thresh[*d*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (prog_full) flag. The threshold can be dynamically set in-circuit during reset. D = $\log_2$(FIFO depth)-1
axi_r_prog_empty_thresh[*d*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (prog_empty) flag. The threshold can be dynamically set in-circuit during reset. D = $\log_2$(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.

Table 1-20: **AXI4-Lite Read Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
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. **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. $D = \log_2(\text{FIFO depth})+1$
axi_r_underflow	Output	Underflow: Indicates that read request during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO. **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. $D = \log_2(\text{FIFO depth})+1$
axi_r_data_count[*d*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. $D = \log_2(\text{FIFO depth})+1$
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 1-9 highlights just one of many configurations that the FIFO Generator core 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 core, 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 devices.

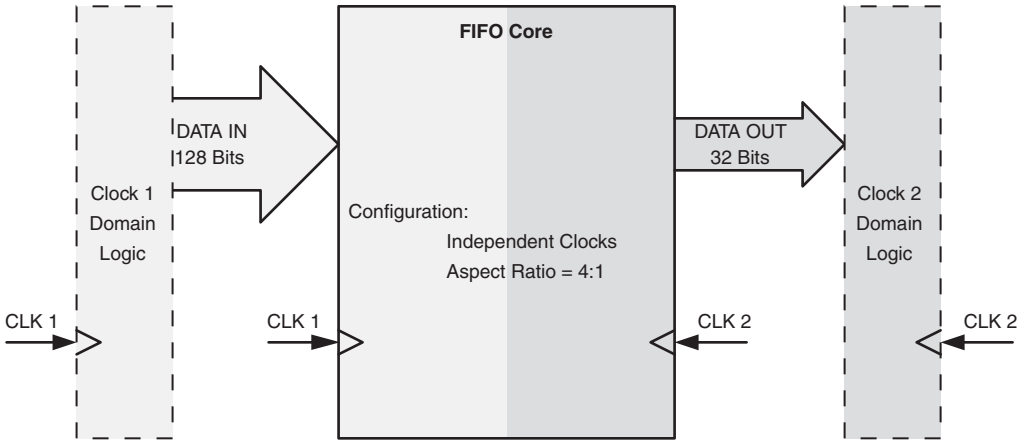


Figure 1-9: **FIFO Generator core Application Example**

AXI 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).

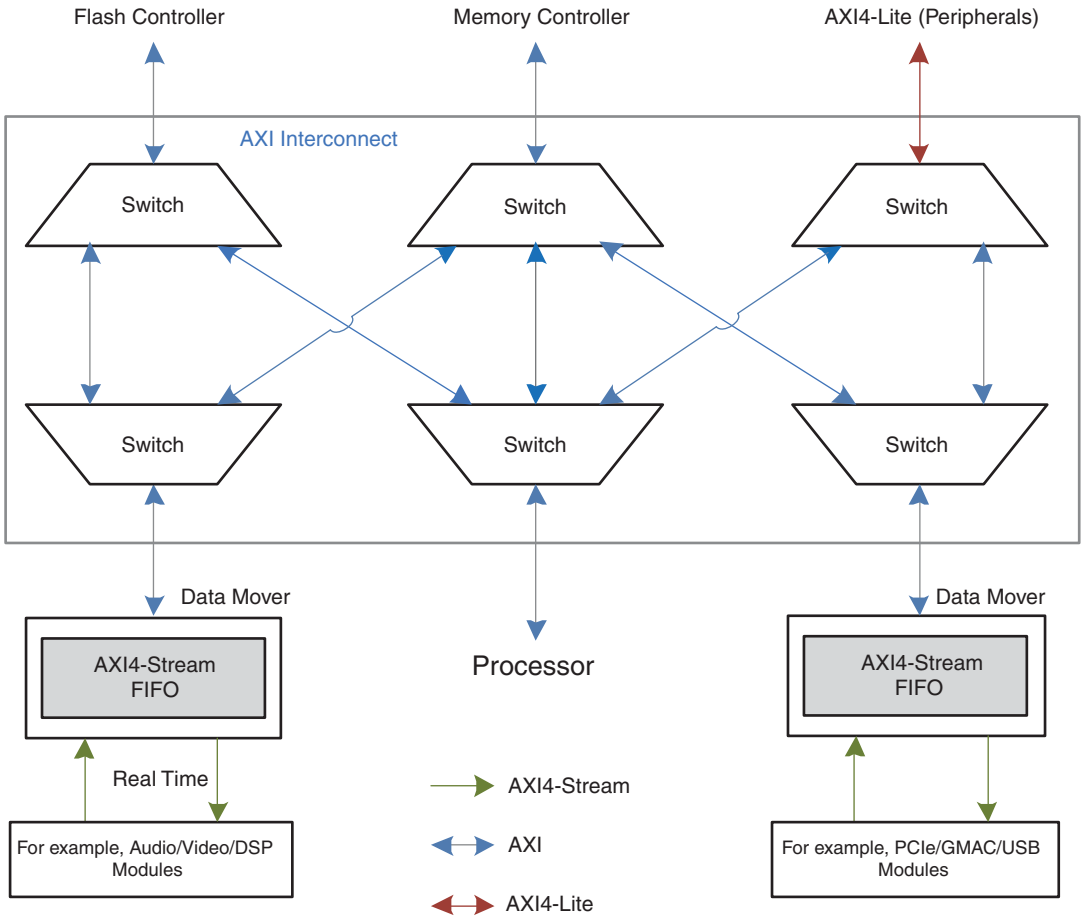


Figure 1-10: **AXI4-Stream Application Diagram**

Figure 1-10 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 AXI System Bus. The AXI Interconnect and Data Mover blocks shown in Figure 1-10 are which are available in the Vivado IP catalog.

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 AXI LogiCORE IP functions easier.

AXI4/AXI3 Memory Mapped FIFOs

The full version of the AXI4/AXI3 interface is referred to as AXI4/AXI3. It may also be referred to as AXI Memory Mapped. Use AXI4/AXI3 FIFOs in memory mapped system bus designs such as bridging applications requiring a memory mapped interface to connect to other AXI4/AXI3 blocks.

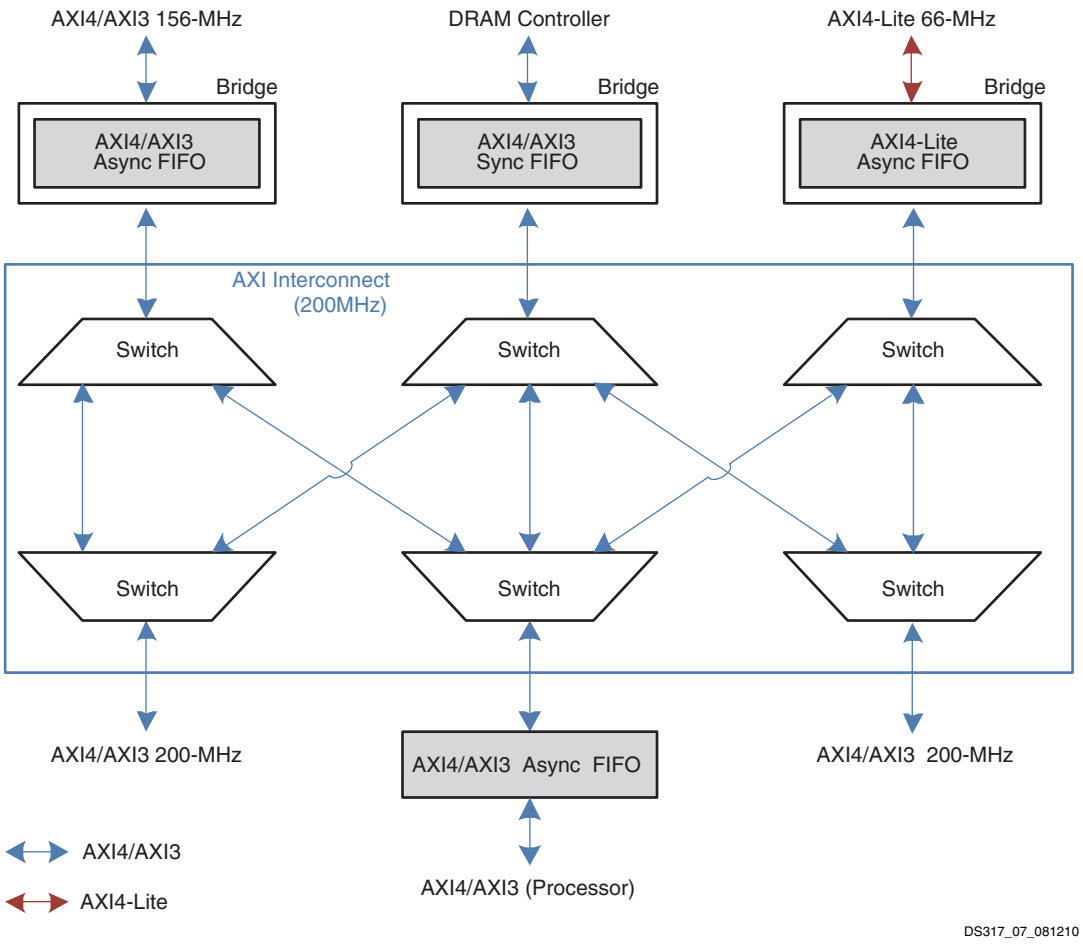


Figure 1-11: **AXI4/AXI3 Application Diagram**

Figure 1-11 shows an example application for AXI4/AXI3 FIFOs where they are used in AXI4/AXI3-to-AXI4/AXI3 bridging applications enabling different AXI4/AXI3 clock domains running at 200, 100, 66, and 156 MHz to communicate with each other. The AXI4/AXI3-to-AXI4-Lite bridging is another pertinent application for AXI4/AXI3 FIFO (for example, for performing protocol conversion). The AXI4/AXI3 FIFOs can also used inside an IP core to buffer data or transactions (for example, a DRAM Controller). The AXI Interconnect block shown in Figure 1-11 is an IP core available in the Vivado IP catalog.

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 1-12 shows an AXI4-Lite FIFO being used in an AXI4/AXI3 to AXI4-Lite bridging application to perform protocol conversion. The AXI4-Lite Interconnect in Figure 1-12 is also available as an IP core in the Vivado IP catalog.

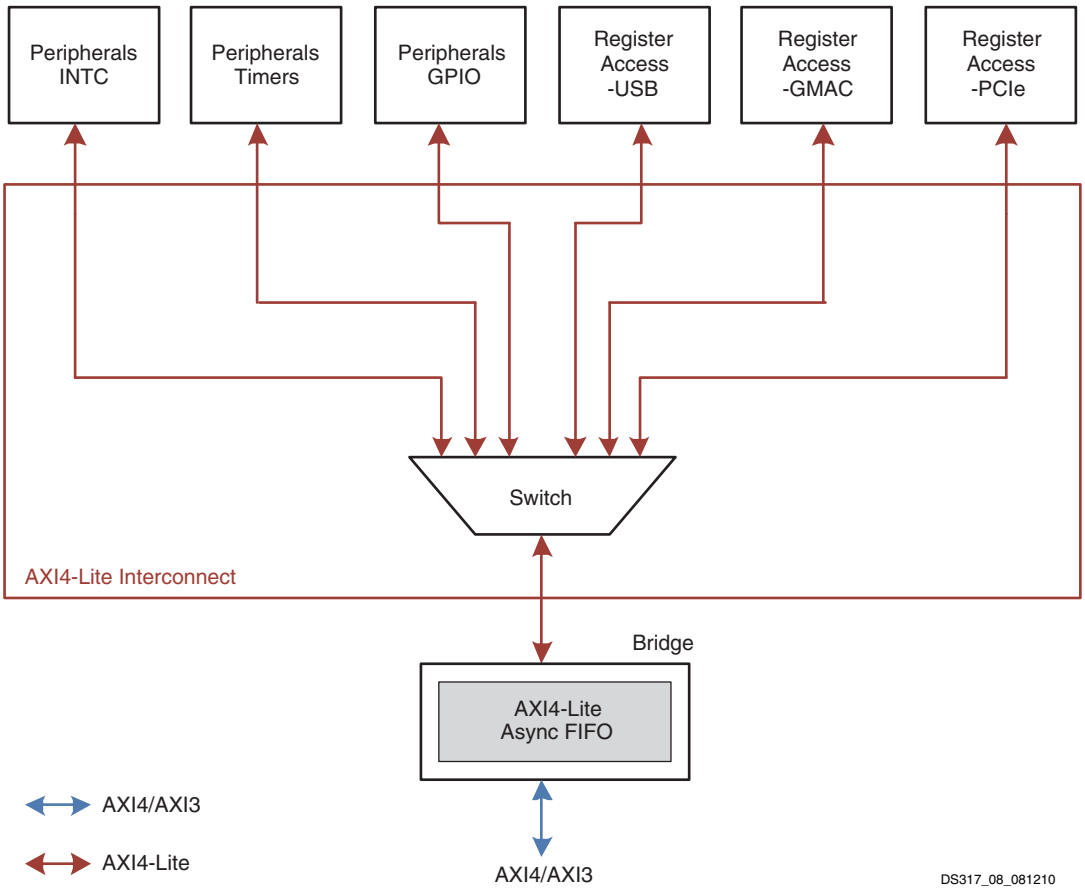


Figure 1-12: **AXI4-Lite Application Diagram**

Licensing and Ordering

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.

Native FIFO Performance

Performance for a Native interface FIFO varies depending on the configuration and features selected during core customization. Table 2-1 through Table 2-2 show 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 Vivado Design Suite.

- Artix®-7 (XC7A200T- FFG1156-1)
- Virtex®-7 (XC7V2000T-FLG1925-1)
- Kintex®-7 (XC7K480T-FFG1156-1)
- Virtex® UltraScale™ (XCVU125-FLVA2104-1-I-ES2)
- Kintex® UltraScale™ (XCKU115-FLVD1924-1-C-ES2)

Table 2-1: **Benchmarks: FIFO Configured without Optional Features for 7 Series Family**

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)
Common Clock FIFO (Block RAM)	512 x 16	Artix-7	270
		Kintex-7	325
		Virtex-7	325
	4096 x 16	Artix-7	265
		Kintex-7	350
		Virtex-7	355
Common Clock FIFO (Distributed RAM)	512 x 16	Artix-7	250
		Kintex-7	345
		Virtex-7	350
	64 x 16	Artix-7	325
		Kintex-7	420
		Virtex-7	440
Independent Clock FIFO (Block RAM)	512 x 16	Artix-7	265
		Kintex-7	335
		Virtex-7	335
	4096 x 16	Artix-7	275
		Kintex-7	340
		Virtex-7	350
Independent Clock FIFO (Distributed RAM)	512 x 16	Artix-7	275
		Kintex-7	355
		Virtex-7	370
	64 x 16	Artix-7	365
		Kintex-7	445
		Virtex-7	475
Shifting Register FIFO	512 x 16	Artix-7	195
		Kintex-7	250
		Virtex-7	240
	64 x 16	Artix-7	300
		Kintex-7	420
		Virtex-7	410

Table 2-2: **Benchmarks: FIFO Configured without Optional Features for UltraScale Family**

FIFO Type	Depth x Width	FPGA Family	Performance (Fmax)
Common Clock Block RAM	512 x 16	Virtex UltraScale	498
		Kintex UltraScale	506
	4096 x 16	Virtex UltraScale	521
		Kintex UltraScale	513
Common Clock Distributed RAM	512 x 16	Virtex UltraScale	513
		Kintex UltraScale	510
	64x16	Virtex UltraScale	583
		Kintex UltraScale	581
Independent Clock Block RAM	512x16	Virtex UltraScale	521
		Kintex UltraScale	506
	4096x12	Virtex UltraScale	521
		Kintex UltraScale	506
Independent Clock Distributed RAM	512x16	Virtex UltraScale	552
		Kintex UltraScale	544
	64x16	Virtex UltraScale	631
		Kintex UltraScale	629
Shift Register FIFO	512x16	Virtex UltraScale	338
		Kintex UltraScale	333
	64x16	Virtex UltraScale	583
		Kintex UltraScale	581

Table 2-3 and Table 2-4 provides the results for FIFOs configured with multiple programmable thresholds. Benchmarks were performed using the following devices:

Note: These benchmarks were obtained using Vivado Design Suite.

- Artix-7 (XC7A200T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)
- Virtex® UltraScale™ (XCVU125-FLVA2104-1-I-ES2)
- Kintex® UltraScale™ (XCKU115-FLVD1924-1-C-ES2)

Table 2-3: **Benchmarks: FIFO Configured with Multiple Programmable Thresholds for 7 Series**

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)
Common Clock FIFO (Block RAM)	512 x 16	Artix-7	245
		Kintex-7	325
		Virtex-7	325
	4096 x 16	Artix-7	265
		Kintex-7	340
		Virtex-7	375
Common Clock FIFO (Distributed RAM)	512 x 16	Artix-7	250
		Kintex-7	355
		Virtex-7	350
	64 x 16	Artix-7	290
		Kintex-7	400
		Virtex-7	335
Independent Clock FIFO (Block RAM)	512 x 16	Artix-7	265
		Kintex-7	325
		Virtex-7	330
	4096 x 16	Artix-7	285
		Kintex-7	350
		Virtex-7	320
Independent Clock FIFO (Distributed RAM)	512 x 16	Artix-7	255
		Kintex-7	355
		Virtex-7	365
	64 x 16	Artix-7	345
		Kintex-7	450
		Virtex-7	470
Shifting Register FIFO	512 x 16	Artix-7	190
		Kintex-7	245
		Virtex-7	225
	64 x 16	Artix-7	295
		Kintex-7	400
		Virtex-7	400

Table 2-4: **Benchmarks: FIFO Configured with Multiple Programmable Thresholds for UltraScale Family**

FIFO Type	Depth x Width	FPGA Family	Performance (Fmax)
Common Clock Block RAM	512 x 16	Virtex UltraScale	506
		Kintex UltraScale	506
	4096 x 16	Virtex UltraScale	521
		Kintex UltraScale	513
Common Clock Distributed RAM	512 x 16	Virtex UltraScale	498
		Kintex UltraScale	513
	64 x 16	Virtex UltraScale	615
		Kintex UltraScale	611
Independent Clock Block RAM	512 x 16	Virtex UltraScale	521
		Kintex UltraScale	498
	4096 x 16	Virtex UltraScale	521
		Kintex UltraScale	521
Independent Clock Distributed RAM	512 x 16	Virtex UltraScale	490
		Kintex UltraScale	552
	64 x 16	Virtex UltraScale	631
		Kintex UltraScale	631
Shifting Register FIFO	512 x 16	Virtex UltraScale	310
		Kintex UltraScale	307
	64 x 16	Virtex UltraScale	607
		Kintex UltraScale	601

Table 2-5 and Table 2-6 provides the results for FIFOs configured to use the built-in FIFO. The benchmarks were performed using the following devices:

Note: These benchmarks were obtained using Vivado Design Suite.

- Artix-7 (XC7A200T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)
- Virtex® UltraScale™ (XCVU125-FLVA2104-1-I-ES2)
- Kintex® UltraScale™ (XCKU115-FLVD1924-1-C-ES2)

Table 2-5: **Benchmarks: FIFO Configured with FIFO36E1 Resources for 7 Series Family**

FIFO Type	Depth x Width	FPGA Family	Read Mode	Performance (MHz)
Common Clock FIFO36E1 (Basic)	512 x 72	Artix-7	Standard	265
			FWFT	255
		Kintex-7	Standard	320
			FWFT	310
		Virtex-7	Standard	215
			FWFT	290
	16k x 8	Artix-7	Standard	225
			FWFT	220
		Kintex-7	Standard	265
			FWFT	270
		Virtex-7	Standard	205
			FWFT	235
Common Clock FIFO36E1 (With Handshaking)	512 x 72	Artix-7	Standard	260
			FWFT	250
		Kintex-7	Standard	320
			FWFT	300
		Virtex-7	Standard	210
			FWFT	300
	16k x 8	Artix-7	Standard	220
			FWFT	225
		Kintex-7	Standard	250
			FWFT	270
		Virtex-7	Standard	250
			FWFT	215

Table 2-5: **Benchmarks: FIFO Configured with FIFO36E1 Resources for 7 Series Family *(Cont'd)***

FIFO Type	Depth x Width	FPGA Family	Read Mode	Performance (MHz)
Independent Clock FIFO36E1 (Basic)	512 x 72	Artix-7	Standard	300
			FWFT	305
		Kintex-7	Standard	385
			FWFT	385
		Virtex-7	Standard	315
			FWFT	315
	16k x 8	Artix-7	Standard	255
			FWFT	245
		Kintex-7	Standard	335
			FWFT	345
		Virtex-7	Standard	250
			FWFT	320
Independent Clock FIFO36E1 (With Handshaking)	512 x 72	Artix-7	Standard	280
			FWFT	345
		Kintex-7	Standard	410
			FWFT	410
		Virtex-7	Standard	330
			FWFT	400
	16k x 8	Artix-7	Standard	255
			FWFT	265
		Kintex-7	Standard	315
			FWFT	315
		Virtex-7	Standard	220
			FWFT	210

Table 2-6: **Benchmarks: FIFO Configured with FIFO36E1 Resources for UltraScale Family**

FIFO Type	Depth x Width	FPGA Family	Read Mode	Performance (Fmax)
Common Clock Built-in FIFO	512x72	Virtex UltraScale	Standard	521
			FWFT	521
		Kintex UltraScale	Standard	521
			FWFT	521
	16kx8	Virtex UltraScale	Standard	521
			FWFT	521
		Kintex UltraScale	Standard	521
			FWFT	521
Common Clock Built-in FIFO (with Handshaking)	512x72	Virtex UltraScale	Standard	521
			FWFT	521
		Kintex UltraScale	Standard	521
			FWFT	521
	16kx8	Virtex UltraScale	Standard	521
			FWFT	521
		Kintex UltraScale	Standard	521
			FWFT	521
Independent Clock Built-in FIFO	512x72	Virtex UltraScale	Standard	521
			FWFT	521
		Kintex UltraScale	Standard	521
			FWFT	521
	16kx8	Virtex UltraScale	Standard	521
			FWFT	521
		Kintex UltraScale	Standard	521
			FWFT	521
Independent Clock Built-in FIFO(with handshaking)	512x72	Virtex UltraScale	Standard	521
			FWFT	521
		Kintex UltraScale	Standard	521
			FWFT	521
	16kx8	Virtex UltraScale	Standard	521
			FWFT	521
		Kintex UltraScale	Standard	521
			FWFT	521

AXI Memory Mapped FIFO Performance

Table 2-7 provides the default configuration settings for the benchmarks data. Table 2-8 shows benchmark information for AXI4/AXI3 and AXI4-Lite configurations. The benchmarks were obtained using the following devices:

Note: These benchmarks were obtained using Vivado Design Suite.

- Artix-7 (XC7A200T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)
- Virtex® UltraScale™ (XCVU125-FLVA2104-1-I-ES2)
- Kintex® UltraScale™ (XCKU115-FLVD1924-1-C-ES2)

Table 2-7: **AXI4/AXI3 and AXI4-Lite Default Configuration Settings**

AXI Type	FIFO Type	Channel Type	ID, Address and Data Width	FIFO Depth x Width
AXI4/AXI3 for 7 series Family	Distributed RAM	Write Address	ID = 4 Address = 32 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 for 7 series Family	Distributed RAM	Write Address	ID = 4 Address = 32 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
AXI4/AXI3 for UltraScale Family	Distributed RAM	Write Address	ID=0 Address=32 Data=64	16x61
	Built-in	Write Data		512x73
	Distributed RAM	Write Response		16x2
	Distributed RAM	Read Address		16x61
	Built-in	Read Data		512x67
AXI4-Lite for UltraScale Family	Distributed RAM	Write Address	Address=32 Data=64	16x35
	Built-in	Write Data		512x72
	Distributed RAM	Write Response		16x2
	Distributed RAM	Read Address		16x35
	Built-in	Read Data		512x66

Table 2-8: **AXI4/AXI3 and AXI4-Lite Performance**

FIFO Type	Clock Type	FPGA Family	Performance (MHz)
AXI4/AXI3 for 7 series	Common Clock	Artix-7	260
		Kintex-7	315
		Virtex-7	179
	Independent Clock	Artix-7	231
		Kintex-7	335
		Virtex-7	194
AXI4-Lite for 7 series	Common Clock	Artix-7	245
		Kintex-7	350
		Virtex-7	214
	Independent Clock	Artix-7	240
		Kintex-7	350
		Virtex-7	190
AXI4/AXI3 for UltraScale	Common Clock	Virtex UltraScale	521
		Kintex UltraScale	521
	Independent Clock	Virtex UltraScale	365
		Kintex UltraScale	341
AXI4-Lite for UltraScale	Common Clock	Virtex UltraScale	521
		Kintex UltraScale	521
	Independent Clock	Virtex UltraScale	521
		Kintex UltraScale	521

Table 2-9 and Table 2-10 provides the benchmarking results for AXI4-Stream FIFO configurations. The benchmarks were obtained using the following devices:

Note: These benchmarks were obtained using the Vivado Design Suite.

- Artix-7 (XC7A200T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)
- Virtex® UltraScale™ (XCVU125-FLVA2104-1-I-ES2)
- Kintex® UltraScale™ (XCKU115-FLVD1924-1-C-ES2)

Table 2-9: **AXI4-Stream Performance for 7 series Family**

FIFO Type	FPGA Family	Depth x Width	Performance (MHz)
Common Clock FIFO (Block RAM)	512 x 16	Artix-7	254
		Kintex-7	355
		Virtex-7	329
	4096 x 16	Artix-7	260
		Kintex-7	325
		Virtex-7	325
Common Clock FIFO (Distributed RAM)	512 x 16	Artix-7	259
		Kintex-7	378
		Virtex-7	349
	64 x16	Artix-7	308
		Kintex-7	445
		Virtex-7	466
Independent Clock FIFO (Block RAM)	512 x 16	Artix-7	266
		Kintex-7	355
		Virtex-7	325
	4096 x 16	Artix-7	282
		Kintex-7	355
		Virtex-7	350
Independent Clock FIFO (Distributed RAM)	512 x 16	Artix-7	110
		Kintex-7	375
		Virtex-7	395
	64 x 16	Artix-7	340
		Kintex-7	485
		Virtex-7	495

Table 2-10: **AXI4-Stream Performance for UltraScale Family**

FIFO Type	Depth x Family	FPGA Family	Performance (F_{max})
Common Clock FIFO (Block RAM)	512x16	Virtex UltraScale	470
		Kintex UltraScale	458
	4096x16	Virtex UltraScale	458
		Kintex UltraScale	435
Common Clock FIFO (Distributed RAM)	512x16	Virtex UltraScale	498
		Kintex UltraScale	472
	64x16	Virtex UltraScale	631
		Kintex UltraScale	610
Independent Clock FIFO (Block RAM)	512x16	Virtex UltraScale	521
		Kintex UltraScale	513
	4096x16	Virtex UltraScale	521
		Kintex UltraScale	482
Independent Clock FIFO (Distributed RAM)	512x16	Virtex UltraScale	521
		Kintex UltraScale	490
	64x16	Virtex UltraScale	631
		Kintex UltraScale	607

Latency

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

Resource Utilization

For details about Resource Utilization, visit Performance and Resources Utilization web page

Port Descriptions

Native FIFO Port Summary

Table 2-11 describes all the FIFO Generator ports.

Table 2-11: **FIFO Generator Ports**

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
rst (Not available for UltraScale architecture built-in FIFOs)	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	Yes
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
prog_empty	O	Yes	Yes	Yes
rd_data_count[c:0](1)	O	Yes	Yes	Yes
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

Table 2-11: **FIFO Generator Ports *(Cont'd)***

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
prog_empty_thresh_negate	I	Yes	Yes	Yes
sbiterr	O	Yes	Yes	Yes
dbiterr	O	Yes	Yes	Yes
rd_rst	I	Yes	Yes	No
sleep[2]				
wr_rst_busy[3]				
rd_rst_busyb				

Notes:

1. wr_data_count/rd_data_count is also available for UltraScale devices using a common clock Block RAM-based FIFO when the Asymmetric Port Width option is enabled.
2. Available only for UltraScale architecture built-in FIFOs.
3. Available for UltraScale architecture built-in FIFOs and UltraScale architecture non-built-in FIFOs with synchronous reset.

AXI FIFO Port Summary

AXI Global Interface Ports

Table 2-12: **AXI 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 2-13: **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

Table 2-13: **AXI4-Stream FIFO Interface Ports *(Cont'd)***

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
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
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/AXI3 FIFO Interface Ports

Write Channels

Table 2-14: **AXI4/AXI3 Write Address Channel FIFO Interface Ports**

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI4/AXI3 Interface Write Address Channel: Information Signals Mapped to FIFO Data Input (din) bus				
s_axi_awid[m:0]	Input	Yes	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/AXI3 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/AXI3 Interface Write Address Channel: Information Signals Derived from FIFO Data Output (dout) Bus				
m_axi_awid[m:0]	Output	Yes	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/AXI3 Interface Write Address Channel: Handshake Signals for FIFO Read Interface				
m_axi_awvalid	Output	No	Yes	Yes

Table 2-14: **AXI4/AXI3 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/AXI3 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 2-15: **AXI4/AXI3 Write Data Channel FIFO Interface Ports**

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI4/AXI3 Interface Write Data Channel: Information Signals Mapped to FIFO Data Input (din) Bus				
s_axi_wid[m:0]	Input	Yes	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/AXI3 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/AXI3 Interface Write Data Channel: Information Signals Derived from FIFO Data Output (dout) Bus				
m_axi_wid[m:0]	Output	Yes	Yes	Yes
m_axi_wdata[m-1:0]	Output	No	Yes	Yes
m_axi_wstrb[m/8-1:0]	Output	No	Yes	Yes

Table 2-15: **AXI4/AXI3 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/AXI3 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/AXI3 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 2-16: **AXI4/AXI3 Write Response Channel FIFO Interface Ports**

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI4/AXI3 Interface Write Response Channel: Information Signals Derived from FIFO Data Output (dout) Bus				
s_axi_bid[m:0]	Output	Yes	Yes	Yes
s_axi_bresp[1:0]	Output	No	Yes	Yes
s_axi_buser[m:0]	Output	Yes	Yes	Yes
AXI4/AXI3 Interface Write Response Channel: Handshake Signals for FIFO Read Interface				
s_axi_bvalid	Output	No	Yes	Yes
s_axi_bready	Input	No	Yes	Yes

Table 2-16: **AXI4/AXI3 Write Response Channel FIFO Interface Ports *(Cont'd)***

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI4/AXI3 Interface Write Response Channel: Information Signals Mapped to FIFO Data Input (din) Bus				
m_axi_bid[m:0]	Input	Yes	Yes	Yes
m_axi_bresp[1:0]	Input	No	Yes	Yes
m_axi_buser[m:0]	Input	Yes	Yes	Yes
AXI4/AXI3 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/AXI3 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 2-17: **AXI4/AXI3 Read Address Channel FIFO Interface Ports**

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI4/AXI3 Interface Read Address Channel: Information Signals Mapped to FIFO Data Input (din) Bus				
s_axi_arid[m:0]	Input	Yes	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

Table 2-17: **AXI4/AXI3 Read Address Channel FIFO Interface Ports *(Cont'd)***

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
s_axi_arburst[1:0]	Input	No	Yes	Yes
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/AXI3 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/AXI3 Interface, Read Address Channel: Information Signals Derived from FIFO Data Output (dout) Bus				
m_axi_arid[m:0]	Output	Yes	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/AXI3 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/AXI3 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

Table 2-17: **AXI4/AXI3 Read Address Channel FIFO Interface Ports *(Cont'd)***

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
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 2-18: **AXI4/AXI3 Read Data Channel FIFO Interface Ports**

Port Name	Input or Output	Optional Port	Port Available	
			Common Clock	Independent Clocks
AXI4/AXI3 Interface Read Data Channel: Information Signals Derived from FIFO Data Output (dout) Bus				
s_axi_rid[m:0]	Output	Yes	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/AXI3 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/AXI3 Interface Read Data Channel: Information Signals Mapped to FIFO Data Input (din) Bus				
m_axi_rid[m:0]	Input	Yes	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/AXI3 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/AXI3 Read Data Channel FIFO: Optional Sideband Signals				
axi_r_prog_full_thresh[m:0]	Input	Yes	Yes	Yes

Table 2-18: **AXI4/AXI3 Read Data Channel FIFO Interface Ports**

Port Name	Input or Output	Optional Port	Port Available	
			Common Clock	Independent Clocks
axi_r_prog_empty_thresh[m:0]	Input	Yes	Yes	Yes
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 2-19: **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 2-19: **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 2-20: **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 2-20: **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 2-21: **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 2-21: **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 2-22: **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 2-22: **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 2-23: **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 2-23: **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.

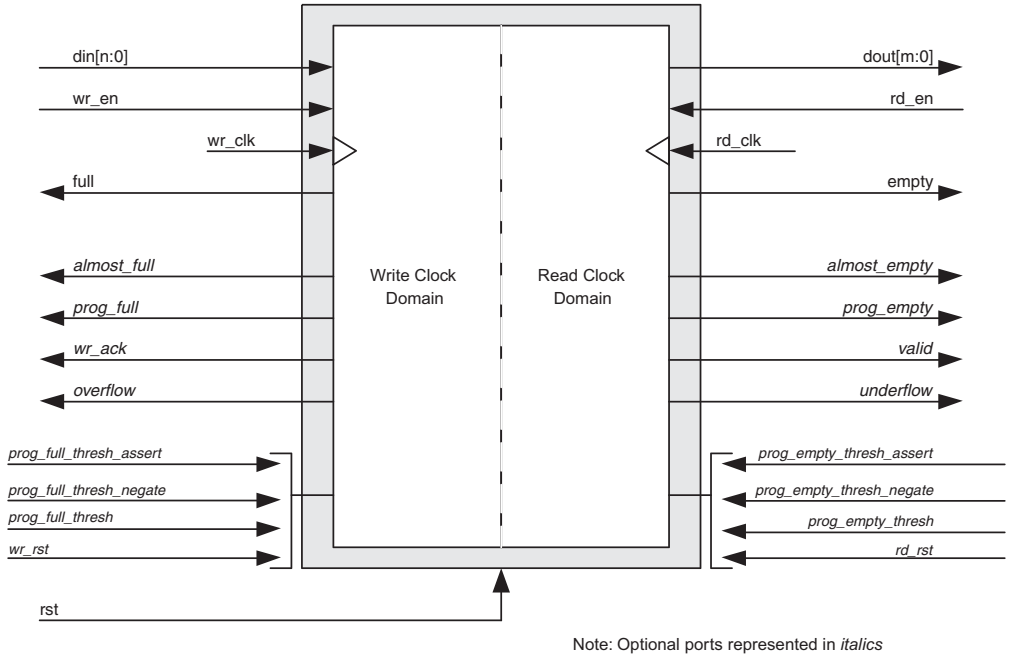


Figure 3-1: **FIFO with Independent Clocks: Write and Read Clock Domains**

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 for shift register FIFOs and 7 series built-in FIFOs. This reset is an asynchronous reset that clears the internal counters and output registers. For UltraScale architecture built-in FIFO implementation, the reset pin (`srst`) is synchronous to `clk`/`wr_clk` that clears the internal counters and output registers. UltraScale architecture built-in FIFO provides `wr_rst_busy` and `rd_rst_busy` output signals to indicate if the FIFO is ready for write or read operations.

For FIFOs implemented with block RAM or distributed RAM, a reset is not required, and the input pin is optional. For common clock configurations, you have the option of asynchronous or synchronous reset. For independent clock configurations, you 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 considered to be synchronous 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 127.

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. When you assert `wr_en`, it causes 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`.

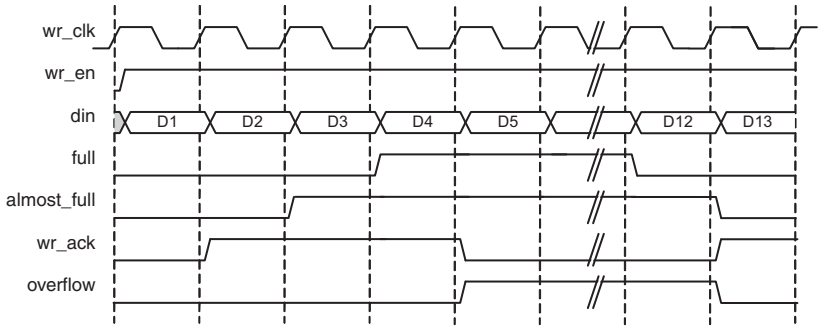


Figure 3-2: **Write Operation for a FIFO with Independent Clocks**

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 core 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. When you write at least one word into the FIFO, `empty` is deasserted — indicating that the data is available to be read. When you assert

rd_en, a read operation occurs 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 you continue to assert rd_en while empty is asserted, the read request is ignored, VALID is deasserted, and underflow is asserted. When you perform a write operation, the FIFO deasserts empty, allowing you to resume valid read operations, as indicated by the assertion of VALID and deassertion of underflow.

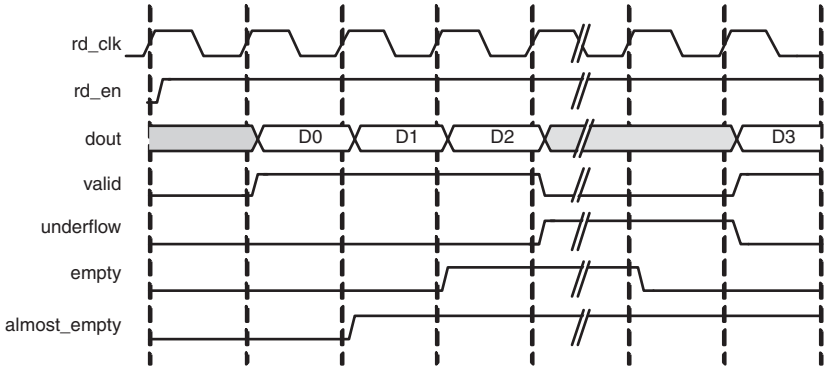


Figure 3-3: **Standard Read Operation for a FIFO with Independent Clocks**

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.

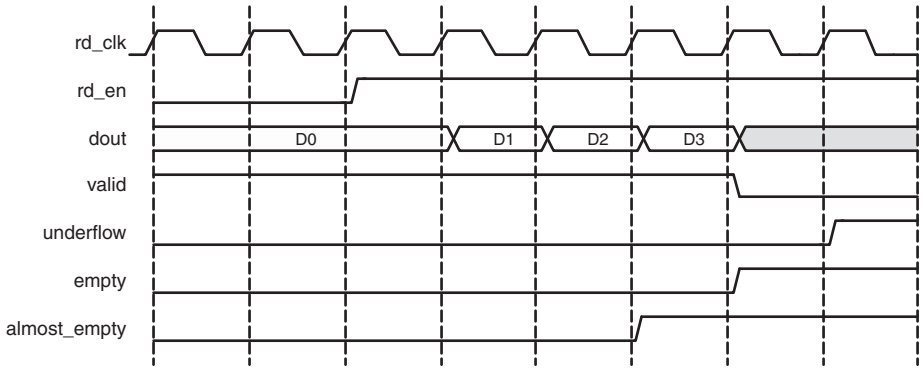


Figure 3-4: **FWFT Read Operation for a FIFO with Independent Clocks**

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.

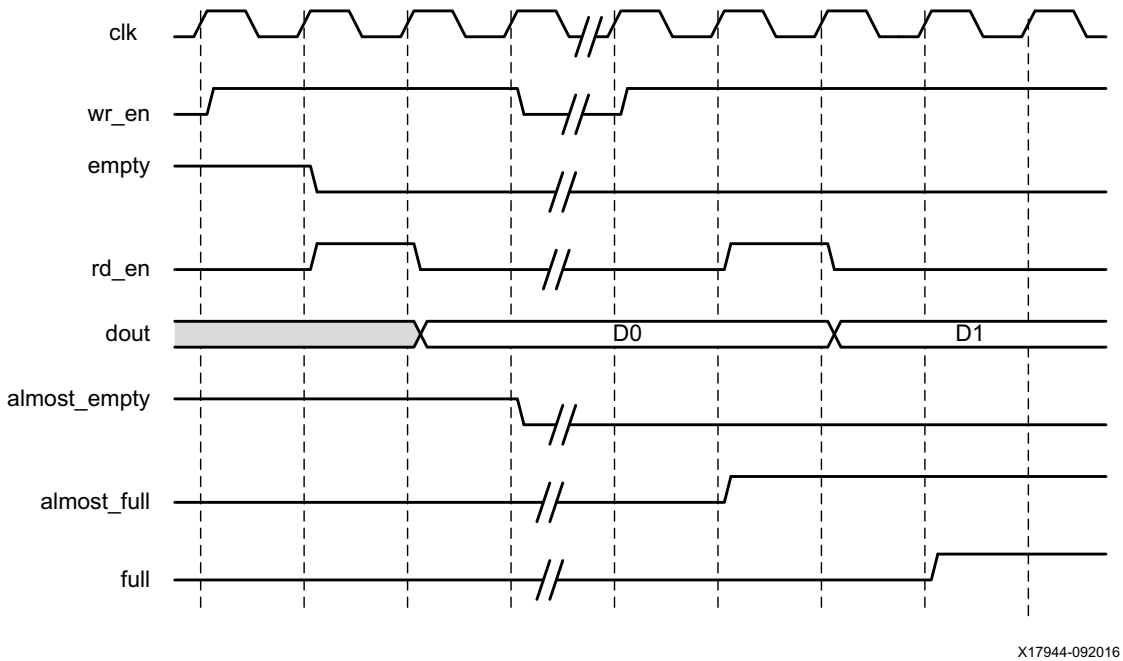


Figure 3-5: **Write and Read Operation for a FIFO with Common Clocks**

Handshaking Flags

Handshaking flags (VALID, underflow, wr_ack 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 Vivado IDE. 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.

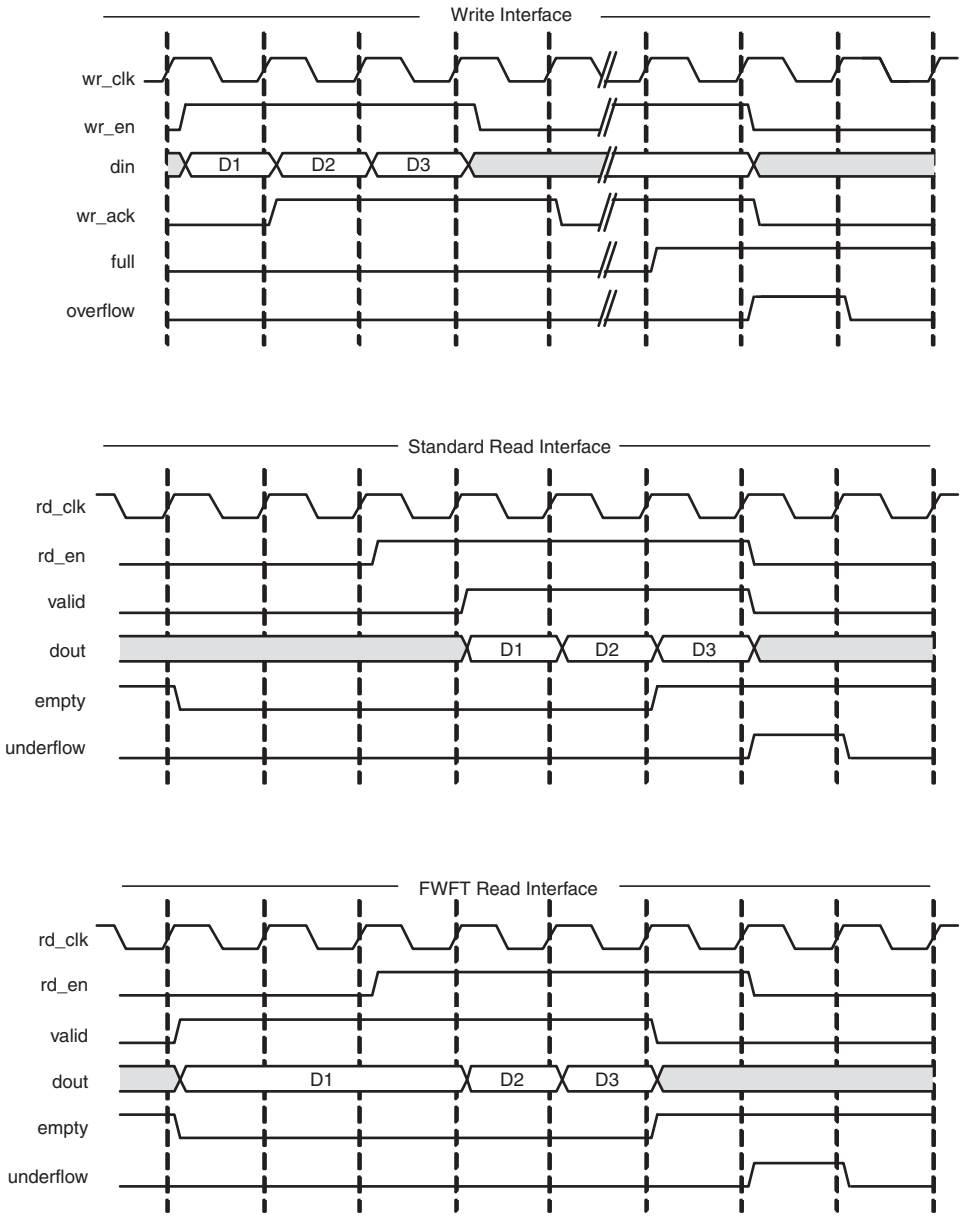


Figure 3-6: **Handshaking Signals for a FIFO with Independent Clocks**

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.

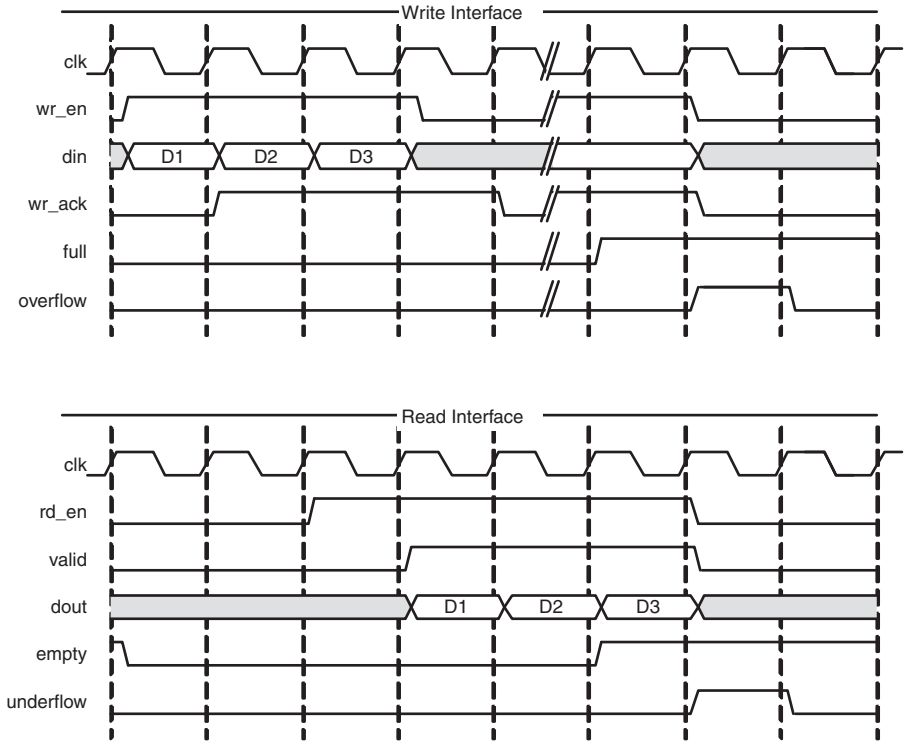


Figure 3-7: **Handshaking Signals for a FIFO with Common Clocks**

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 139.

Programmable Full

The FIFO Generator core 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 Vivado IDE and accessed within the programmable flags window (Status Flags Tab, page 162).

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**. You can specify the threshold value through the FIFO Generator Vivado IDE. 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). You can specify the threshold value through an input port (`prog_full_thresh`) on the core. This input can be changed while the FIFO is in reset, providing you the flexibility to change the programmable full threshold in-circuit without re-generating the core.

Note: See the Vivado IDE 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. You can write 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. When the FIFO has six or fewer words in the FIFO, prog_full is deasserted.*

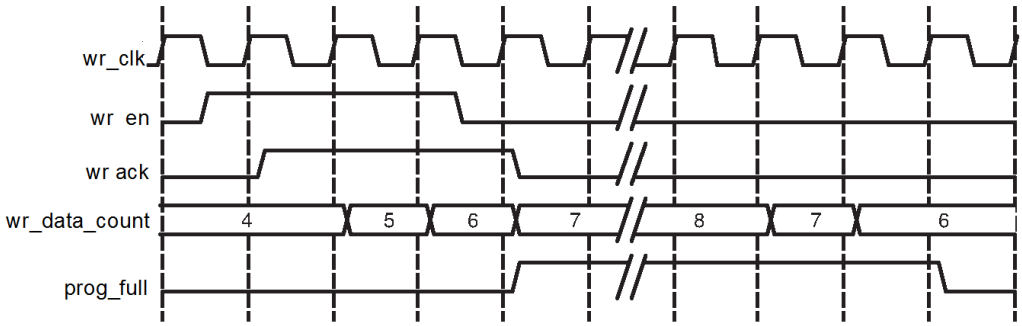


Figure 3-8: **Programmable Full Single Threshold: Threshold Set to 7**

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: You can specify the threshold values through the FIFO Generator Vivado IDE. 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: You can specify 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 Vivado IDE for valid ranges for each threshold.

Figure 3-9 shows the programmable full flag with assert and negate thresholds. You can write 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.

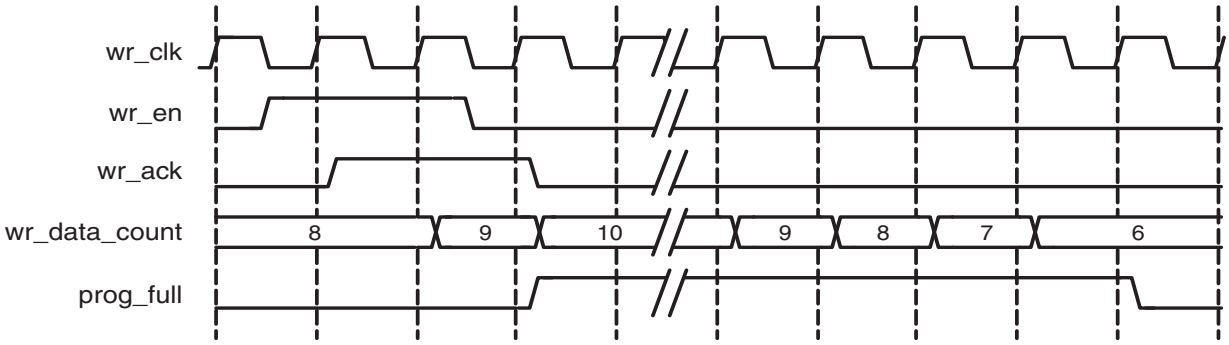


Figure 3-9: **Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7**

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 Vivado IDE 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 Vivado IDE 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 core 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 Vivado IDE and accessed within the programmable flags window (Status Flags Tab, page 162).

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**: You can specify the threshold value through the Vivado IDE. 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**: You can specify the threshold value through an input port (`prog_empty_thresh`) on the core. This input can be changed while the FIFO is in reset, providing the flexibility to change the programmable empty threshold in-circuit without re-generating the core.

Note: See the Vivado IDE for valid ranges for each threshold.

Figure 3-10 shows the programmable empty flag with a single threshold for a non-built-in FIFO. You can write 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.

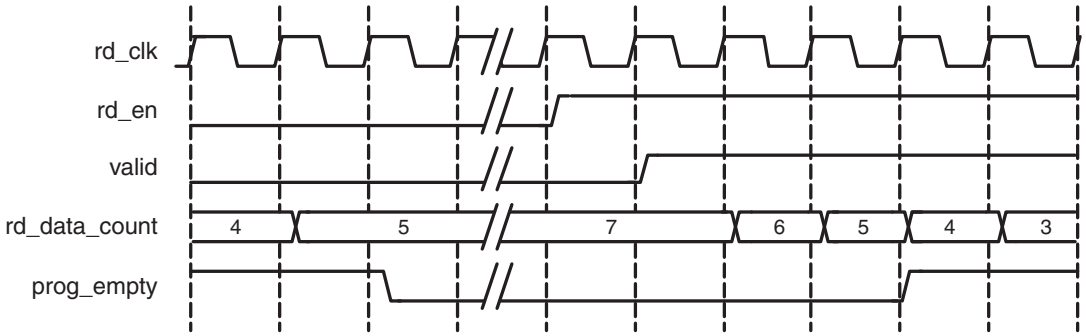


Figure 3-10: **Programmable Empty with Single Threshold: Threshold Set to 4**

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 Vivado IDE. 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 Vivado IDE for valid ranges for each threshold.

Figure 3-11 shows the programmable empty flag with assert and negate thresholds. You can write 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.

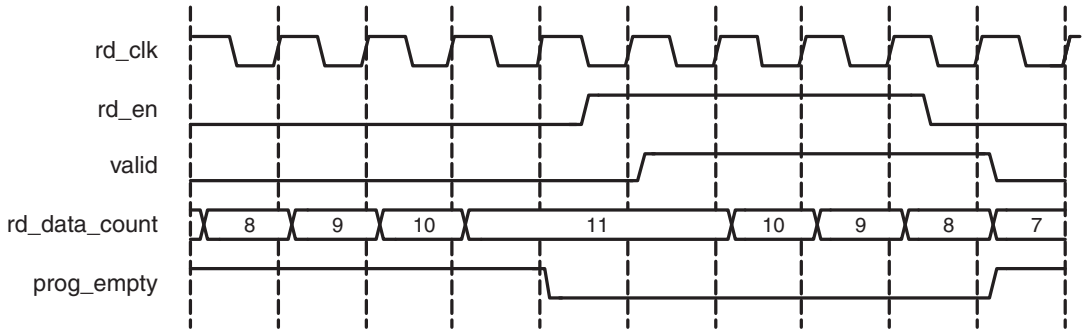


Figure 3-11: **Programmable Empty with Assert and Negate Thresholds: Assert Set to 7 and Negate Set to 10**

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 Vivado IDE 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 Vivado IDE 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 core, and are enabled through the Vivado IDE. Table 3-2 identifies data count support for each FIFO implementation. For information about the latency behavior of data count flags, see Latency, page 139.

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

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`/`clk`, that read is reflected on the `rd_data_count` signal following the next rising clock edge. A write operation on the `wr_clk`/`clk` clock domain may take a number of clock cycles before being reflected in the `rd_data_count`.

Write Data Count

Write data count (wr_data_count) pessimistically reports the number of words written into the FIFO. The count is guaranteed to never under-report the number of words in the FIFO (although it may temporarily over-report the number of words present) to ensure that 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/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/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 core 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 Vivado IDE 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 Vivado IDE. 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 3-18). 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 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.

IMPORTANT: *Use Extra Logic is always true for asymmetric Common Clock BRAM FIFOs when rd_data_count or wr_data_count is enabled.*

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)
1:2	0	1(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

Notes:

1. This is an expected value. However, it may over-report up to two words near empty or until both empty & almost_empty deasserts.

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.

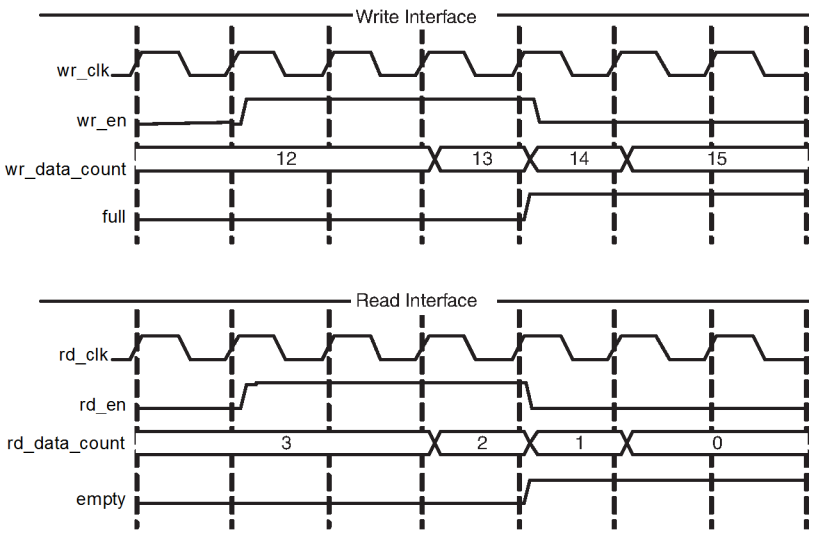


Figure 3-12: **Write and Read Data Counts for FIFO with Independent Clocks**

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 (UltraScale Only)	✓
Common Clock	Block RAM	✓
	Distributed RAM	
	Shift Register	
	Built-in (UltraScale Only)	✓

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.

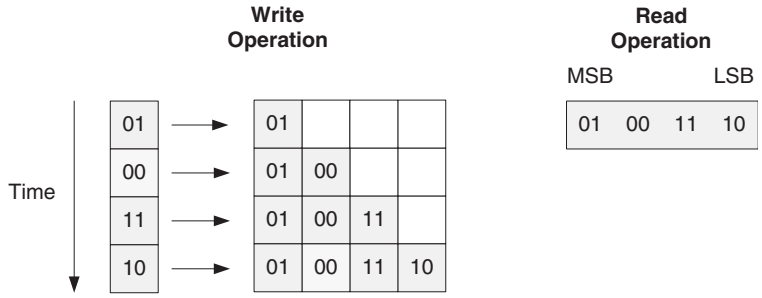


Figure 3-13: **1:4 Aspect Ratio: Data Ordering**

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.

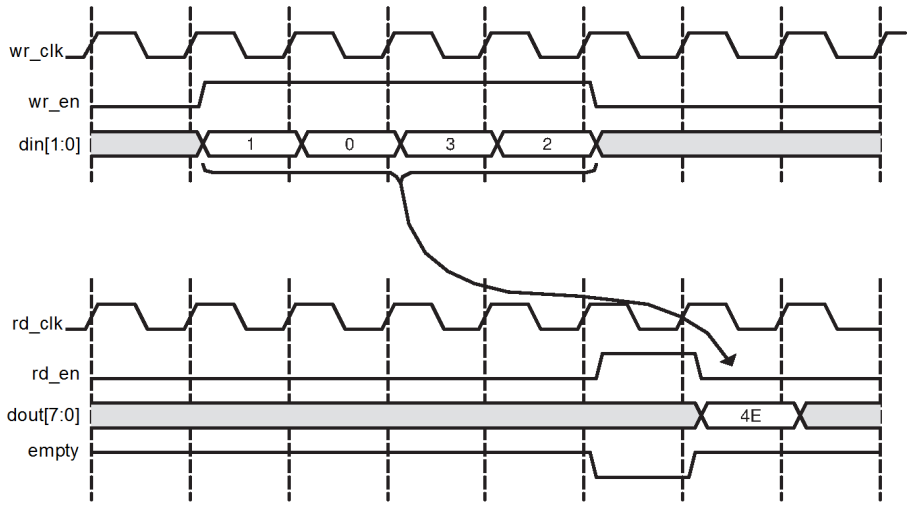


Figure 3-14: **1:4 Aspect Ratio: Status Flag Behavior**

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.

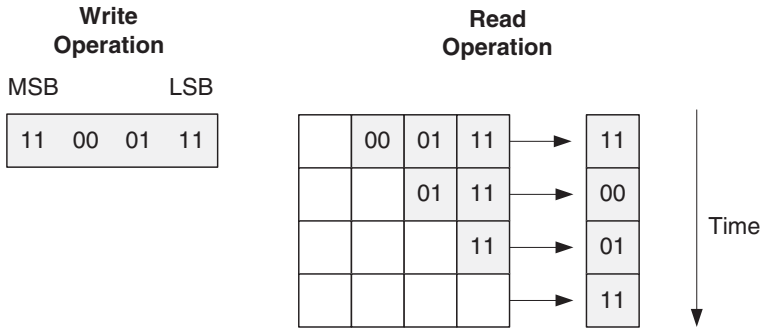


Figure 3-15: **4:1 Aspect Ratio: Data Ordering**

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.

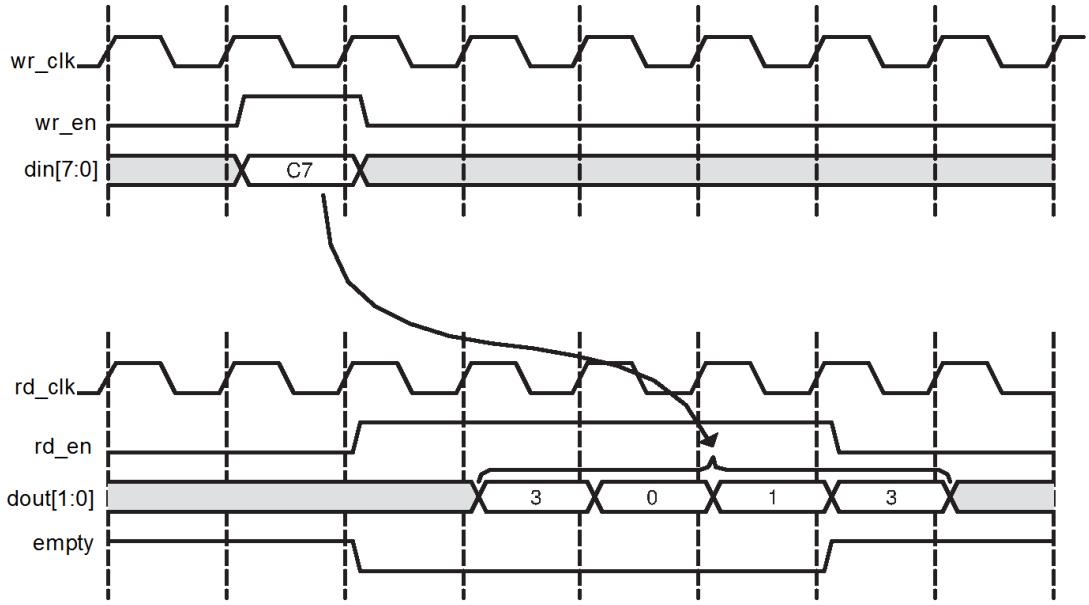


Figure 3-16: **4:1 Aspect Ratio: Status Flag Behavior**

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 Block RAM configuration, you can add an extra output register instead of embedded register from the general interconnect to improve the timing. In built-in FIFO, you have embedded register option.In block RAM FIFO, you have the choice to select either the primitive embedded register or an output register from the general interconnect.

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.

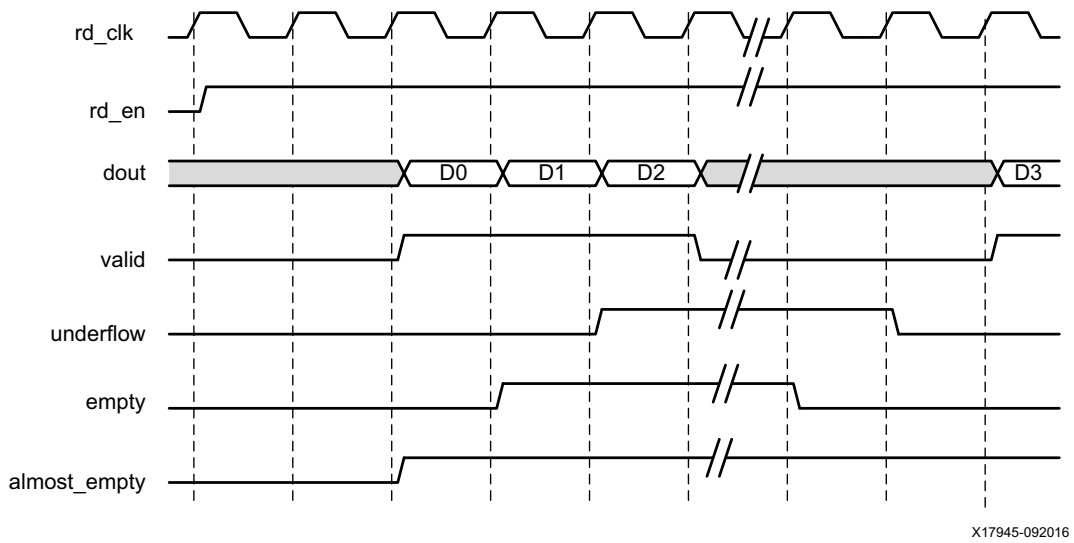


Figure 3-17: **Standard Read Operation for a Block RAM or built-in FIFO with Use Embedded Registers Enabled**

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 in 7 Series. These pipeline registers are always enabled, as shown 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.

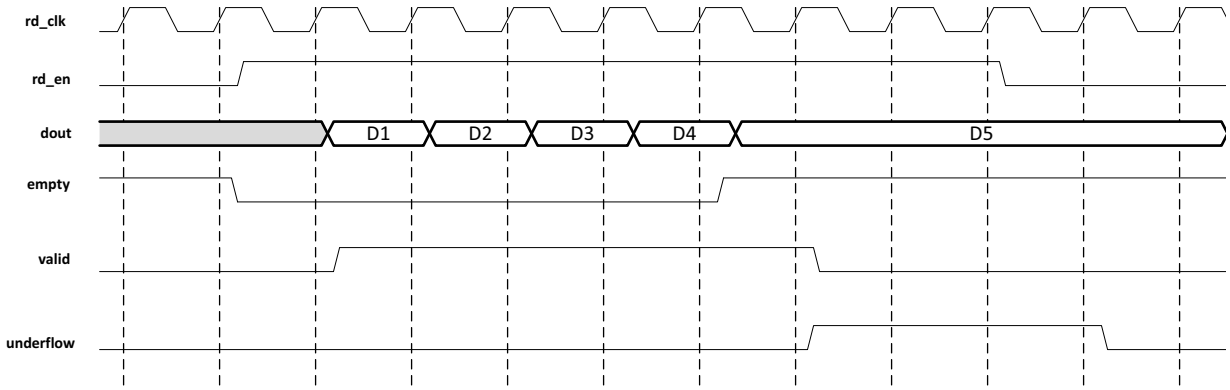


Figure 3-18: **FWFT Read Operation for a Synchronous Built-in FIFO with User Embedded Registers Enabled in 7 Series**

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.

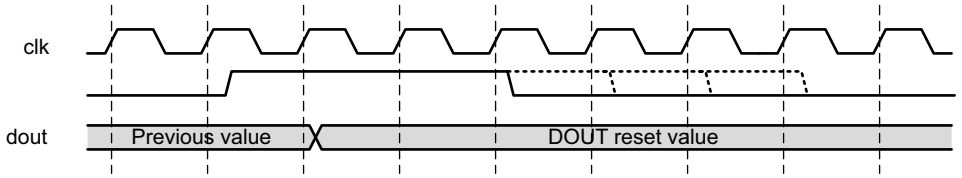


Figure 3-19: **dout Reset Value Common Clock Built-in FIFO Embedded Register for 7 Series**

For UltraScale designs, the read behavior with and without embedded register enabled is same. The dout value is obtained at the same clock edge of read operation due to primitive nature.Figure 3-20 illustrates the dout generation in UltraScale devices with FWFT feature enabled.

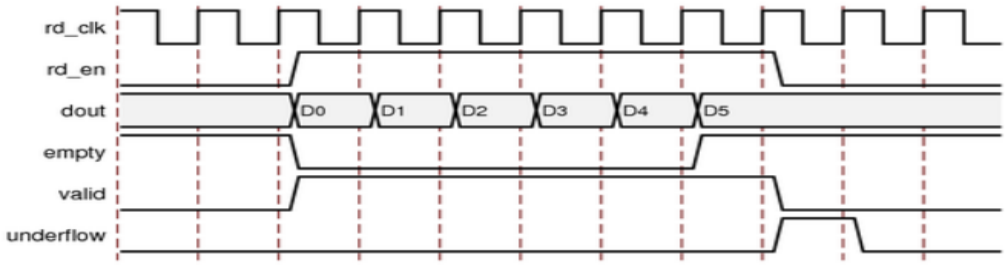


Figure 3-20: **FWFT Read Operation for a Synchronous Built-in FIFO with User Embedded Registers Enabled in UltraScale devices**

Embedded Registers and Interconnect Registers in Block RAM and FIFO Macros

FIFO Generator provides an option to use both embedded and interconnect register for Block RAM based FIFOs (common/independent) to improve timing. The chosen interface type decides the latency added at the output (dout). For standard BRAM FIFOs, a latency of two cycles is added to the output when both registers are chosen as shown in Figure 3-21.

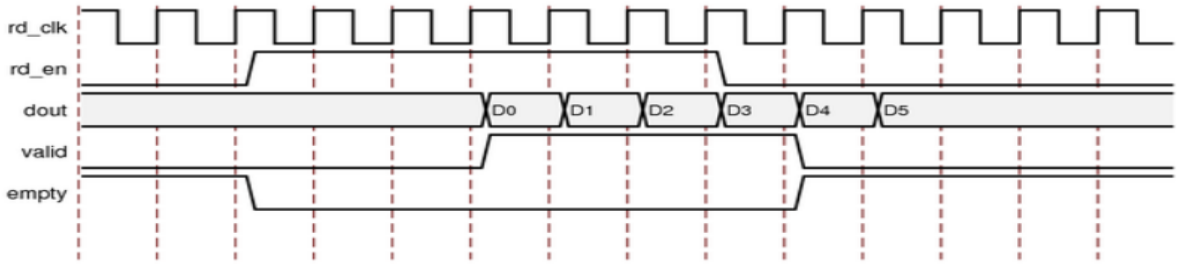


Figure 3-21: **Standard FIFO Behavior with Embedded and Interconnect Registers**

The first word fall through maintains a similar behavior with embedded/interconnect register. The empty gives a latency of one more cycle as compared to selecting only one

register, before rd_en signal is initiated. The next output is latched when rd_en initiates without any additional latency for BRAM as shown in Figure 3-22.

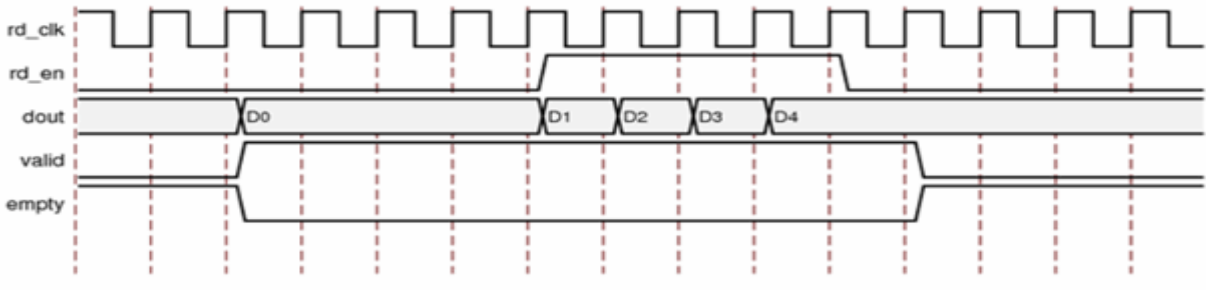


Figure 3-22: **First Word Fall Through with Embedded and Interconnect Registers**

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.

Note: In Block RAM based FIFO configurations with widths lesser than 64, you can select a soft ECC option where the general interconnect is used to build the ECC logic. The functionality remains unchanged between the HARD and Soft ECC.

Figure 3-23 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.*

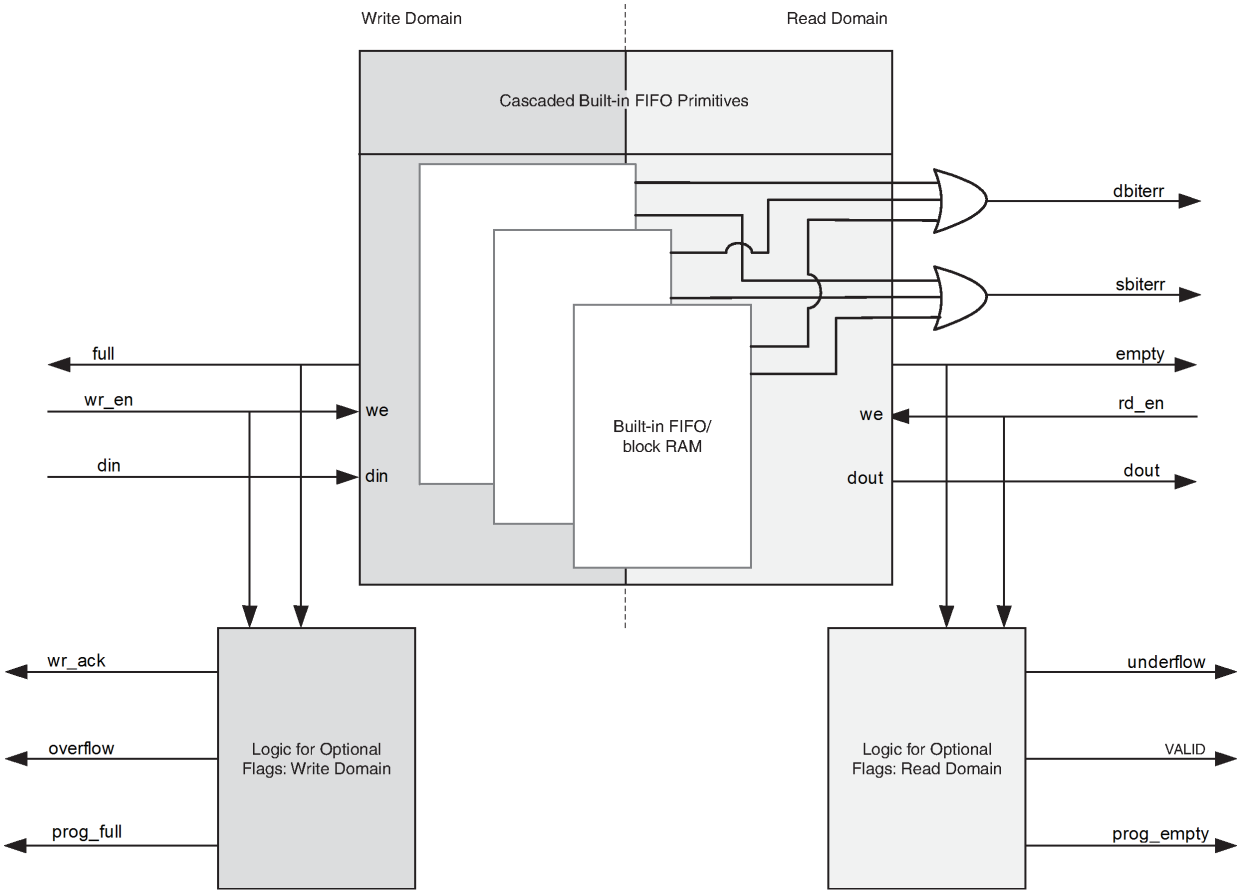


Figure 3-23: **sbiterr and dbiterr Outputs in the FIFO Generator Core**

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 3-24 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 127 for more details.

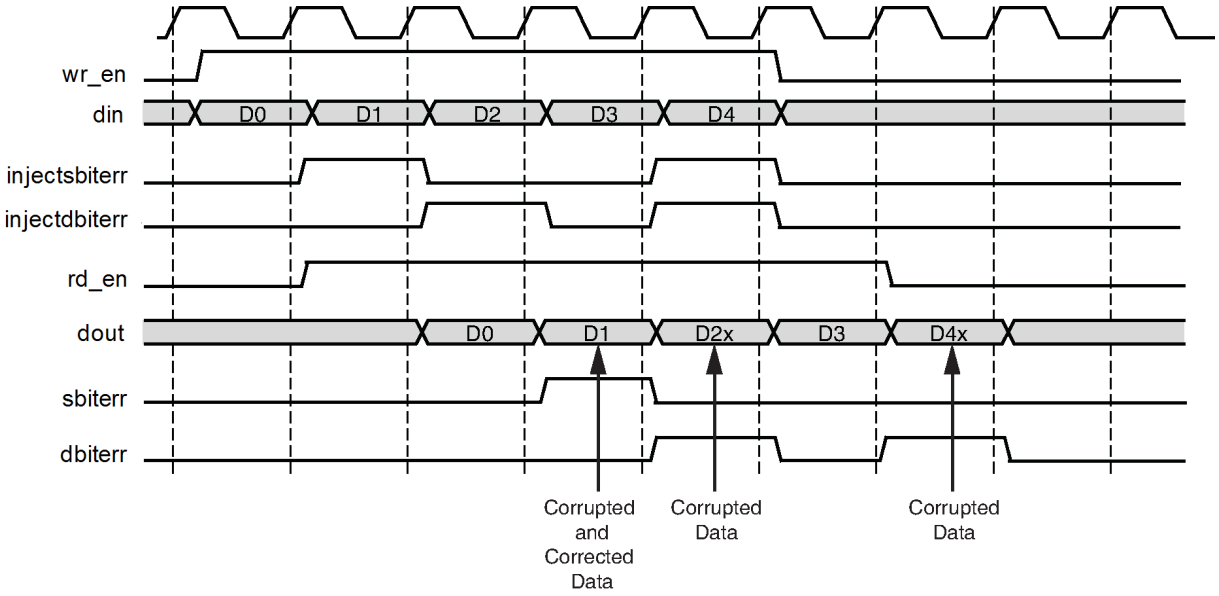


Figure 3-24: **Error Injection and Correction**

Clocking

Each FIFO configuration has a set of allowable features, as defined in Table 1-3, page 15.

Independent Clocks: Block RAM and Distributed RAM

Figure 3-25 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.

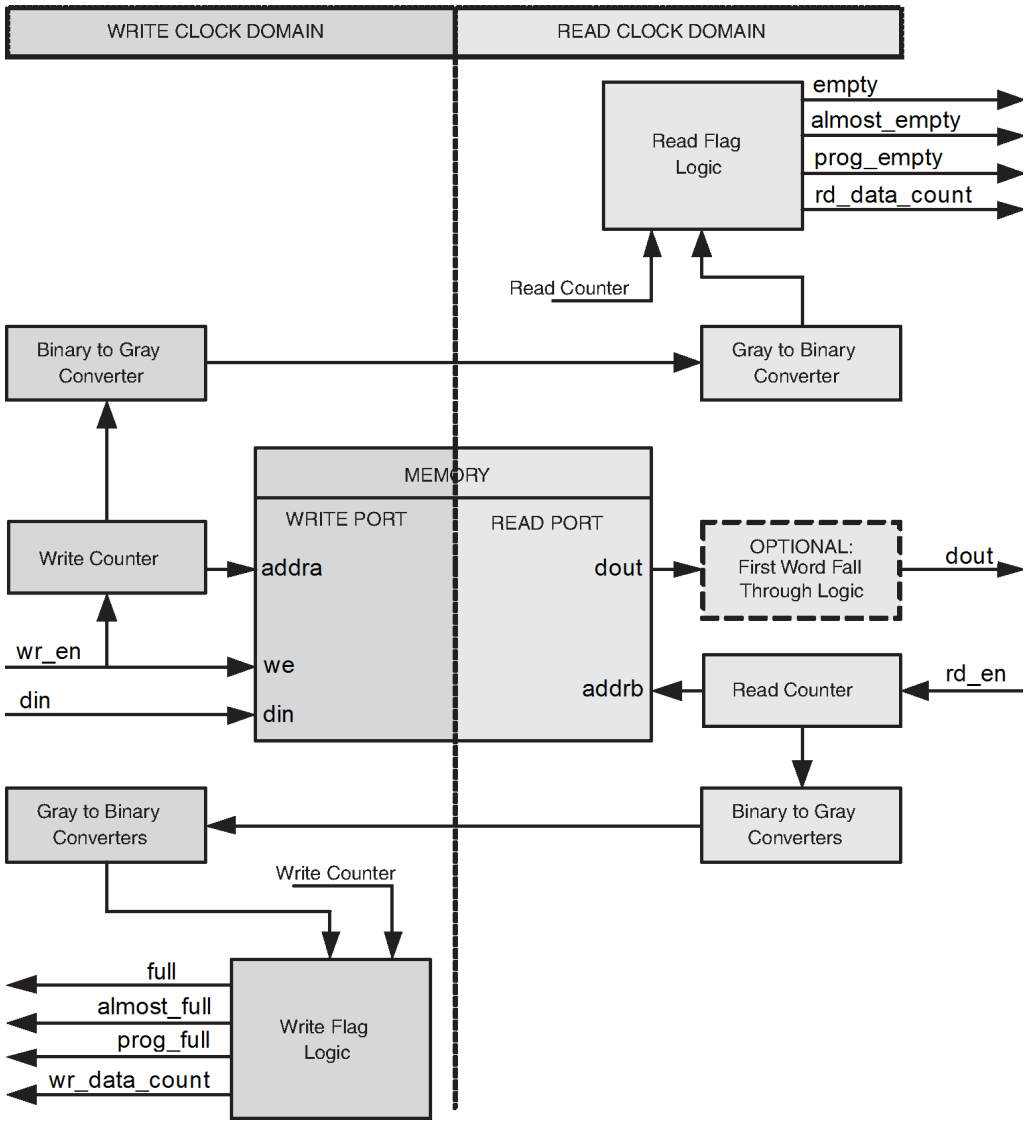


Figure 3-25: **Functional Implementation of a FIFO with Independent Clock Domains**

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_clk	rd_clk
wr_ack	valid
overflow	underflow
wr_data_count	rd_data_count
wr_rst	rd_rst
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 refer to the Synchronization Considerations, page 94.

Independent Clocks: Built-in FIFO

Figure 3-26 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.

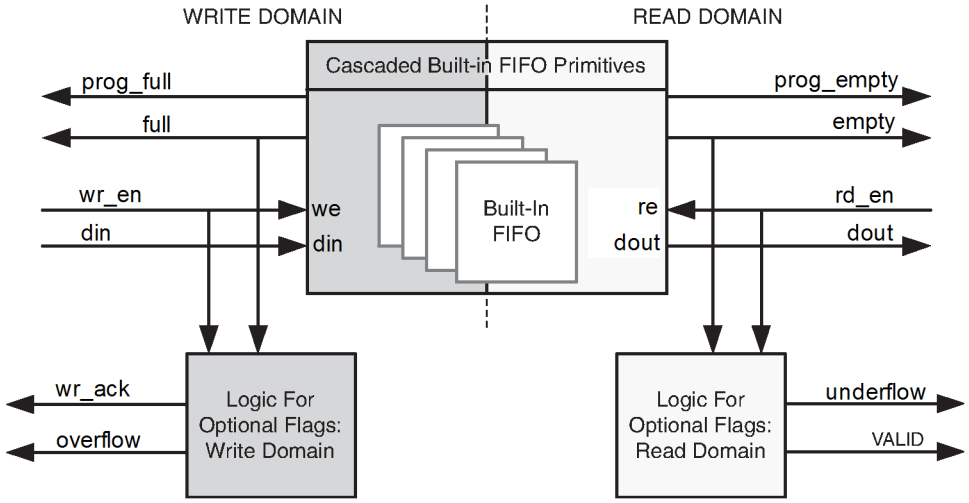


Figure 3-26: **Functional Implementation of Built-in FIFO**

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 94.

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 120.

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 139.

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 core supports FIFO cores using the built-in FIFO primitive with a common clock. This provides you 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 124, for more information.

Common Clock FIFO: Block RAM and Distributed RAM

Figure 3-27 illustrates the functional implementation of a FIFO configured with a common clock using block RAM or distributed RAM for memory. All signals are synchronous to a single clock input (`clk`). This design implements counters for write and read pointers and logic for calculating the status flags. An optional synchronous (`srst`) or asynchronous (`rst`) reset signal is also available.

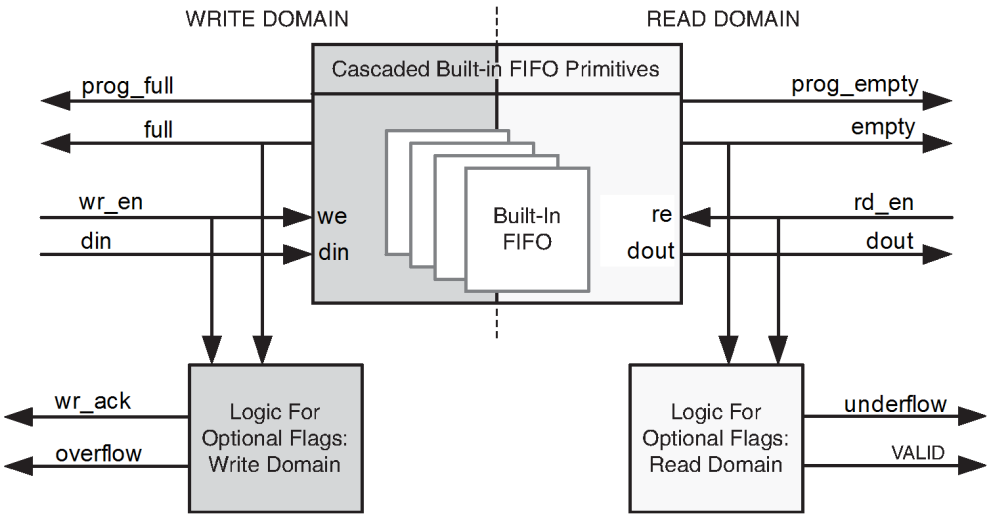


Figure 3-27: **Functional Implementation of a Common Clock FIFO using Block RAM or Distributed RAM**

Common Clock FIFO: Shift Registers

Figure 3-28 illustrates the functional implementation of a FIFO configured with a common clock using shift registers for memory. All operations are synchronous to the same clock input (`clk`). This design implements a single up/down counter for both the write and read pointers and logic for calculating the status flags.

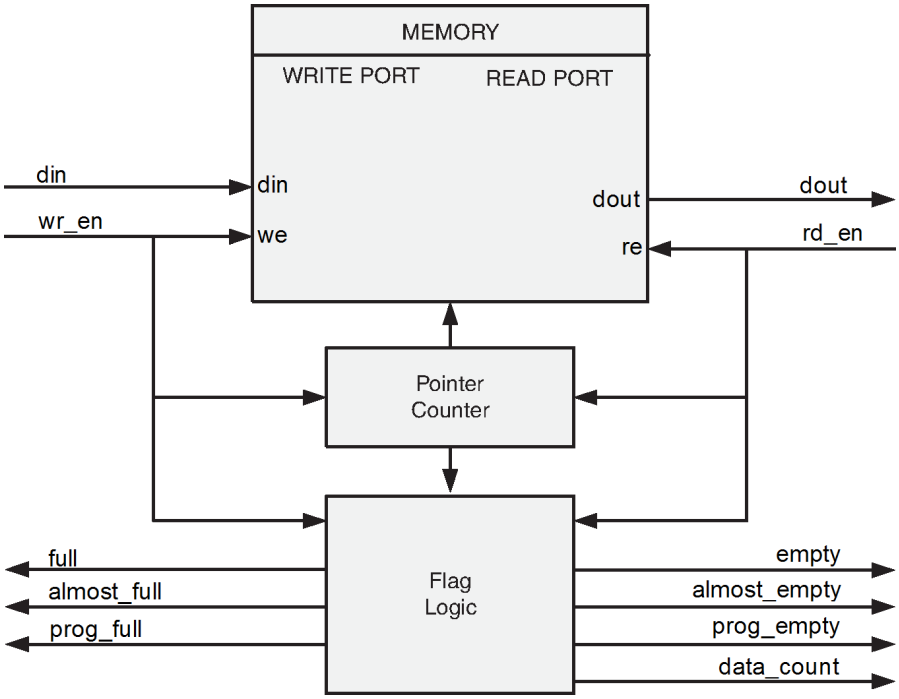


Figure 3-28: **Functional Implementation of a Common Clock FIFO using Shift Registers**

Resets

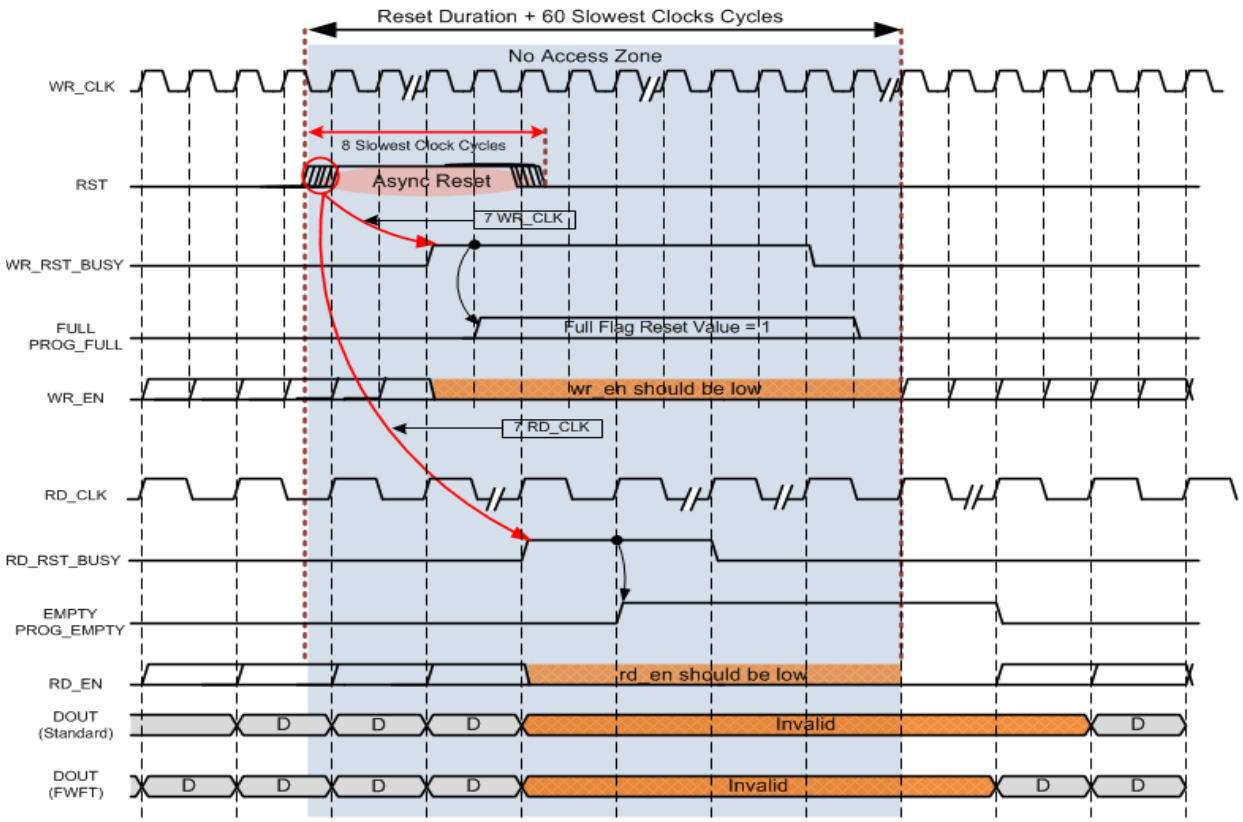
The FIFO Generator core provides a reset input that resets all counters and output registers when asserted. For block RAM or distributed RAM implementations, resetting the FIFO is not required, and the reset pin can be disabled in the FIFO. There are two reset options: asynchronous and synchronous.

Asynchronous Reset

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.

IMPORTANT: *The clock(s) must be available when the reset is applied. If for any reason, the clock(s) is/are lost at the time of reset, you must apply the reset again when the clock(s) is/are available. Violating this requirement may cause an unexpected behavior. Sometimes, the busy signals may be stuck and might need reconfiguration of FPGA.*

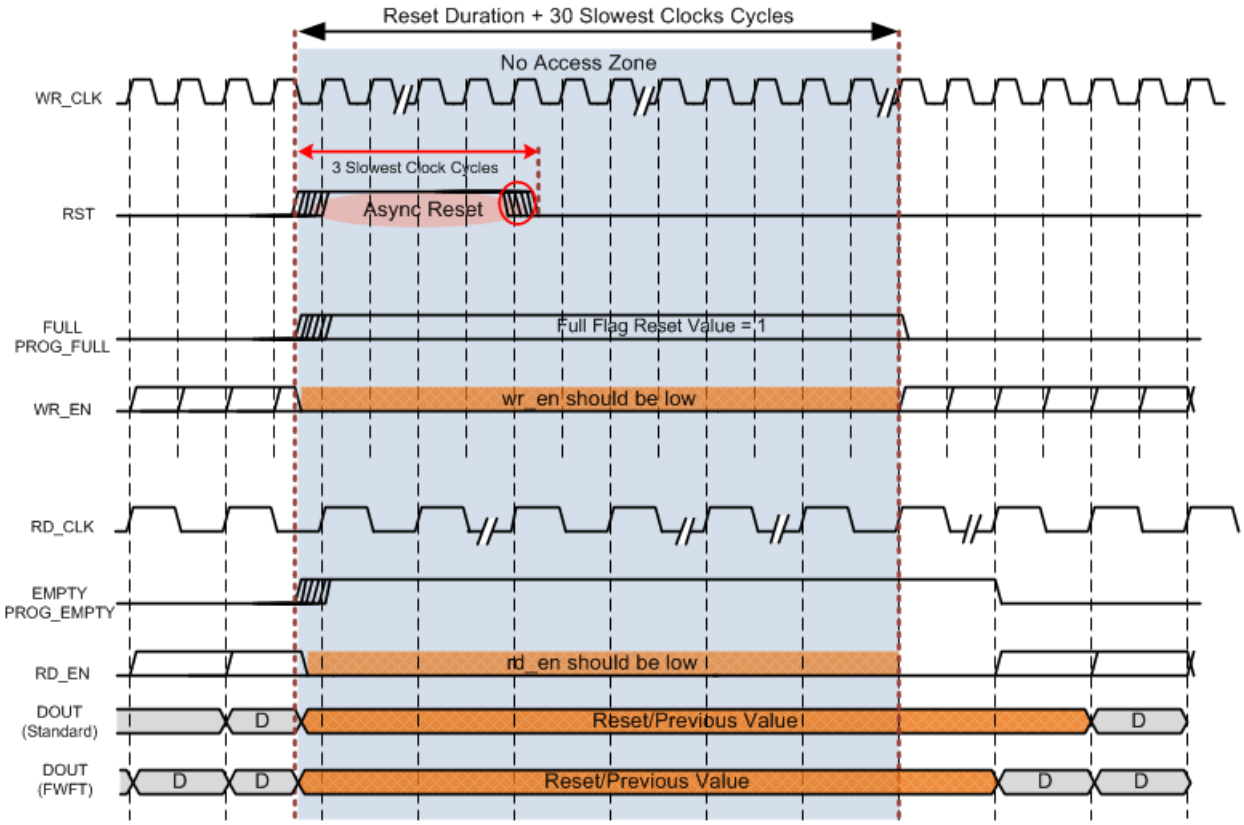
Note: If the asynchronous reset is one slowest clock wide and the assertion happens very close to the rising edge of slowest clock, then the reset detection may not happen properly causing unexpected behavior. To avoid such situations, it is always recommended to have the asynchronous reset asserted for at least 3 or C_SYNCHRONIZER_STAGE (whichever is maximum) slowest clock cycles, though this guide talks about one clock wide reset at some places.



For AXI Interface, the *_axi_**valid/ready must be used outside the *No Access Zone* window only
*_axi_**valid: * --> s_axi_/m_axi_; ** --> tvalid/awvalid/wvalid/bvalid/arvalid/rvalid
*_axi_**ready: * -->s_axi_/m_axi_; ** --> tready/awready/wready/bready/arready/rready

Figure 3-29: **FIFO Asynchronous Reset Timing With Safety Circuit**

Note: All FIFO outputs during *No Access Zone* should be considered as invalid.



For AXI Interface, the *_axi_**valid/ready must be used outside the *No Access Zone* window only
*_axi_**valid: * --> s_axi_/m_axi_; ** --> tvalid/awvalid/wvalid/bvalid/arvalid/rvalid
*_axi_**ready: * -->s_axi_/m_axi_; ** --> tready/awready/wready/bready/arready/rready

Figure 3-30: **FIFO Asynchronous Reset Timing Without Safety Circuit**

Note: All FIFO outputs during *No Access Zone* should be considered as invalid.

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

Ensure that there is a minimum gap of 6 clocks (slower clock in case of independent clock) between 2 consecutive resets when you use Asynchronous reset. In BRAM cases with asynchronous reset, an additional safety circuit option is provided to ensure that the assertion and deassertion of BRAM input signals happen synchronously. When you use a safety circuit option, you need to wait for `wr_rst_busy` signal to transition from 1 to 0 before either applying next reset or initiating any write operations. A DRC warning, if any, on the BRAM can be considered as a false positive warning for asynchronous reset with safety circuit. If you select FIFO Generator's safety circuit option, you need to ensure that the reset (`rst`) signal is asserted High (logic 1) for at least **3 or C_SYNCHRONIZER_STAGE** (whichever is maximum) write/read clock cycles (whichever is slower). For AXI interface, `wr_rst_busy` is asserted inside the core and the transactions are based on `*_axi_**valid/*_axi_**ready` protocol.

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 `wr_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 recommended to follow the reset pulse requirement and it is not recommended to drive/toggle `wr_en`/`rd_en` when `rst` is asserted/High.*

Note: The embedded register option must be selected for enabling safety circuit.

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
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/read clock cycle (whichever is slower). 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 five 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 five clock cycles after the deassertion of asynchronous reset.

See Figure 3-31 and Figure 3-32 for example behaviors. Note that the power-up values for this configuration are different from the reset state value.

Figure 3-31 shows an example timing diagram for when the reset pulse is one clock cycle.

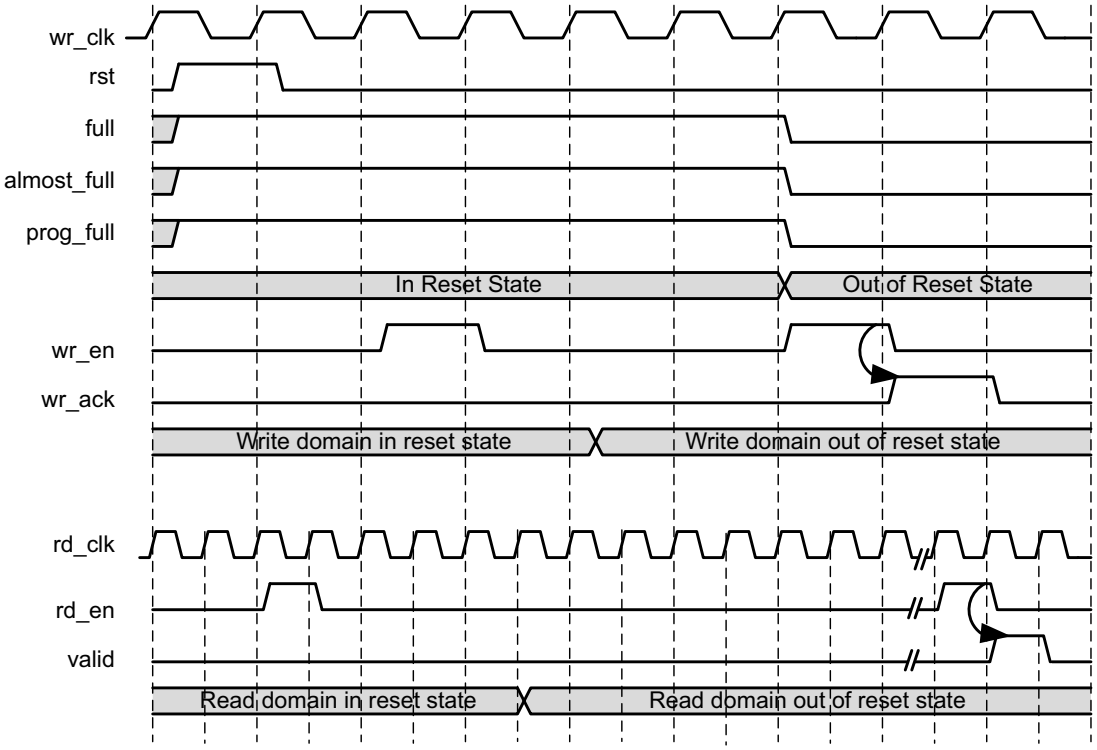


Figure 3-31: **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 1 for the Reset Pulse of One Clock**

Figure 3-32 shows an example timing diagram for when the reset pulse is longer than one clock cycle.

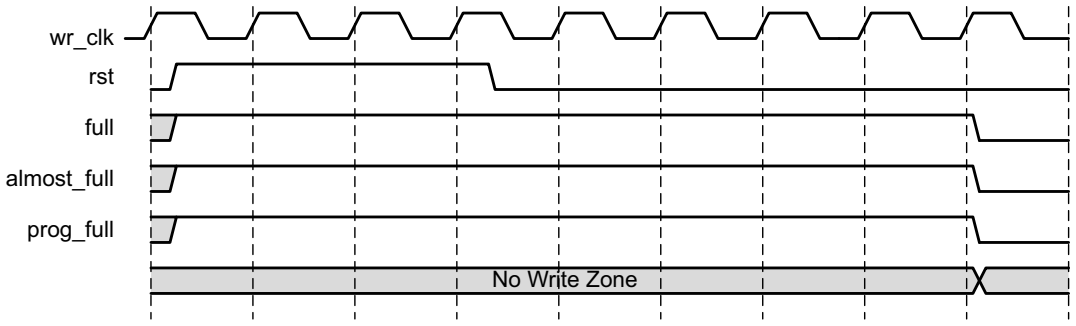


Figure 3-32: **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 1 for the Reset Pulse of More Than One Clock**

Full Flags Reset Value of 0

In this configuration, the FIFO requires a minimum asynchronous reset pulse of one write/read clock cycle (whichever is slower) 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 five clock cycles after the assertion of asynchronous reset. See Figure 3-33 for example behavior.

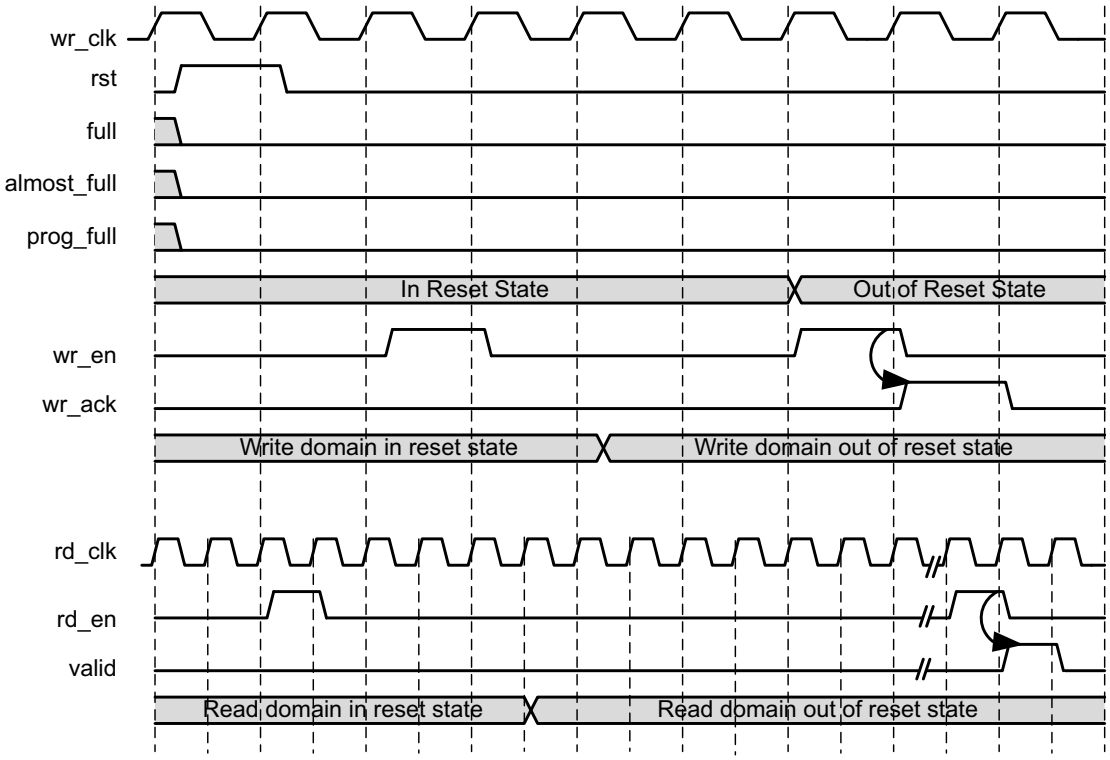


Figure 3-33: **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 0**

Common/Independent Clock: 7 Series Built-in FIFOs

Table 3-8 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 3-34 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 wr_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

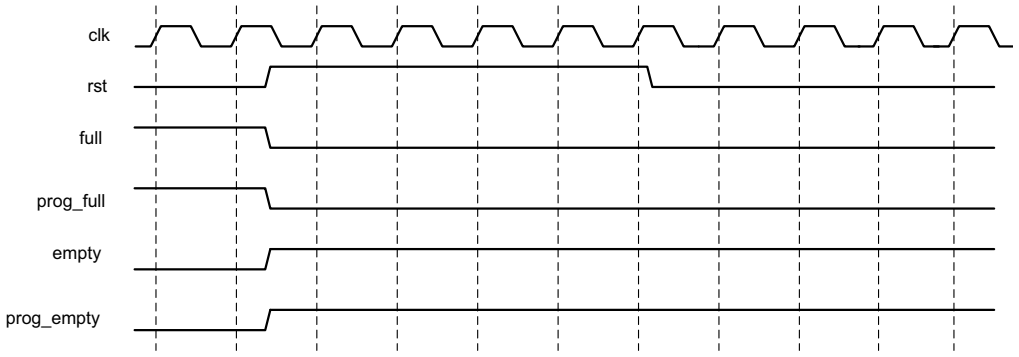


Figure 3-34: **Built-in FIFO, Asynchronous Reset Behavior**

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, shift RAM, or built-in FIFO 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 3-37 illustrates the flags following the release of `srst`.

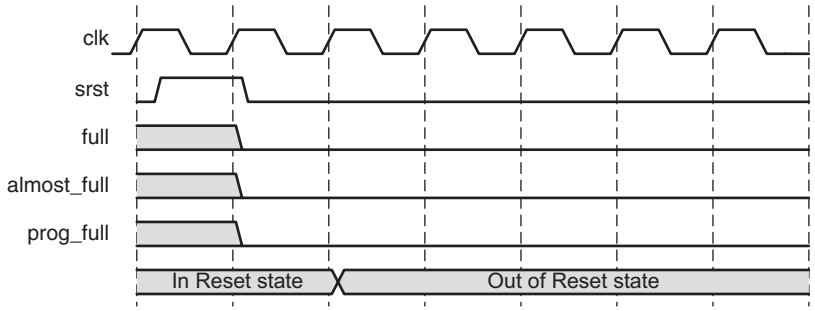


Figure 3-37: **Synchronous Reset: FIFO with a Common Clock**

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.

Assert synchronous resets (`wr_rst/rd_rst`) together, at least for one clock cycle as shown in Figure 3-38. 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 3-38 and Figure 3-39 detail the resets.

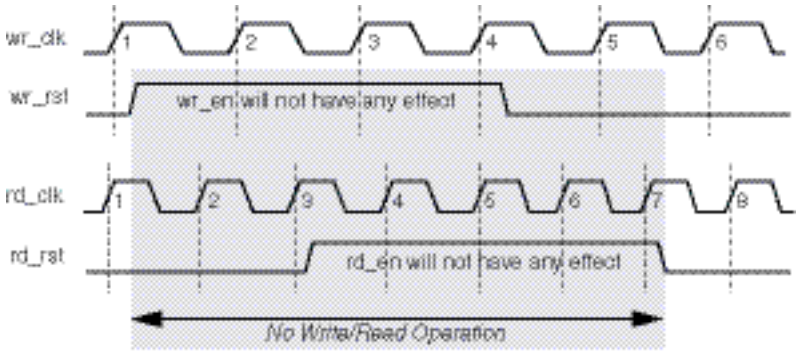


Figure 3-38: **Synchronous Reset: FIFO with Independent Clock - wr_rst then rd_rst**

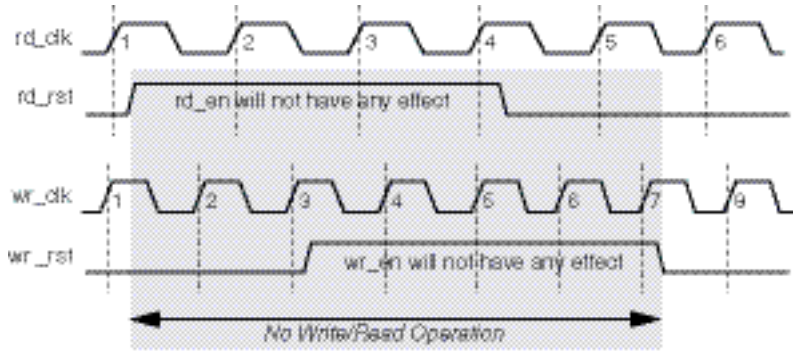


Figure 3-39: **Synchronous Reset: FIFO with Independent Clock - rd_rst then wr_rst**

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 `wr_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

Table 3-9: **Synchronous Reset and Power-up Values *(Cont'd)***

Signal	Block Memory and Distributed Memory Values of Output Ports During Reset and Power-up
valid	0 (active-High) or 1 (active-Low)
wr_ack	0 (active-High) or 1 (active-Low)
prog_full	0
prog_empty	0
rd_data_count	0
wr_data_count	0

Common/Independent Clock: UltraScale Built-in FIFOs

UltraScale architecture-based built-in FIFO supports only the synchronous reset (`srst`). The reset must always be synchronous to write clock (`clk`/`wr_clk`). The built-in FIFOs require a synchronous reset pulse of at least one write clock cycle. The built-in FIFO provides `wr_rst_busy` and `rd_rst_busy` output signals

If `srst` is asserted, the `wr_rst_busy` output asserts immediately after the rising edge of `wr_clk` and remains asserted until the reset operation is complete. Following the assertion of `wr_rst_busy`, the internal reset is synchronized to the `rd_clk` domain. Upon arrival in the `rd_clk` domain, the `rd_rst_busy` is asserted, and is held asserted until the resetting of all `rd_clk` domain signals is complete. At this time, `rd_rst_busy` is deasserted. In common-clock mode, this logic is simplified because the clock domain crossing is not required.

During the reset state, 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 the `empty` and `prog_empty` flags are asserted. After `wr_rst_busy` and `rd_rst_busy` are released, the core exits the reset state and is ready for writing.

For more information, see the *UltraScale Architecture Memory Resources: Advance Specification User Guide* (UG573) [Ref 4].

IMPORTANT: *The underflow and overflow signals are directly connected to the FIFO18E2/FIFO36E2 primitive. If `rd_en` is asserted and the FIFO is empty, underflow is asserted. If `wr_en` is asserted and the FIFO is full, overflow is asserted.*

Note: Ensure that the FIFO Generator's input clocks are free-running while applying reset. Violating this might result in reset hang condition, which requires hard restart or reboot. For more information, see AR 67912.

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 Vivado IDE, 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.
4. When you select Embedded and Interconnect Registers, the actual write depth and actual read depth increases by one as compared to selecting embedded/interconnect register in FWFT mode.

Built-In FIFOs

The actual FIFO depths of 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.

Table 3-10: **Built-in FIFO Primitives**

			Common_Clock		Independent_Clock	
			STD	**FWFT**	**STD**	**FWFT**
7 Series			((primitive_depth+1) *N)-1	(primitive_depth+ 1)*N	((primitive_depth+1) *N)-1	(primitive_depth +1)*N

Table 3-10: **Built-in FIFO Primitives**

			Common_Clock		Independent_Clock	
			STD	**FWFT**	**STD**	**FWFT**
UltraScale/ UltraScale+	**With Low Latency & with low latency output register**	With embedded register	(primitive_depth+1)*N	primitive_depth*N		
		Without embedded register	(primitive_depth+1)*N	primitive_depth*N	(primitive_depth+1)*N	primitive_depth*N
	With Low Latency & without low latency output register	With embedded register	primitive_depth*N	primitive_depth*N		
		Without embedded register	primitive_depth*N	primitive_depth*N	primitive_depth*N	primitive_depth*N
	Without Low Latency	With embedded register	((primitive_depth+1)*N)-1	primitive_depth*N	((primitive_depth+2)*N)-2	((primitive_depth+1)*N)-1
		Without embedded register	(primitive_depth*N)-1	primitive_depth*N	((primitive_depth+2)*N)-2	((primitive_depth+1)*N)-1

Notes:

1. primitive_depth = depth of the primitive used to implement the FIFO. This is displayed in the Summary tab of FIFO GUI. For detailed information on the primitive_depth for various width and depth configurations, see 7 Series FPGAs Memory Resources User Guide (UG473)[Ref 3] and UltraScale Architecture Memory Resources (UG573)[Ref 4]
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 3-40 in which `wr_ack` is getting updated in which `wr_en` is high.

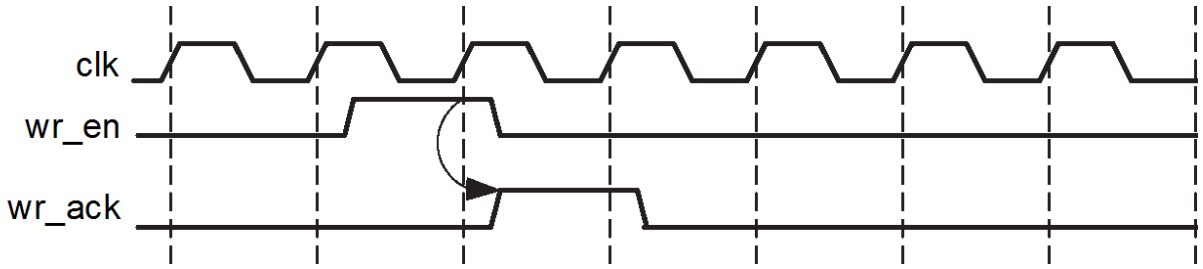


Figure 3-40: **Latency 0 Timing**

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

Table 3-11 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 3-11: **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 3-12 defines the read port flags update latency due to a read operation.

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

Table 3-13: **Write Port Flags Update Latency Due to Read Operation**

Signals	Latency (clk)
full	0
almost_full	0
prog_full	1

Table 3-13: **Write Port Flags Update Latency Due to Read Operation *(Cont'd)***

wr_ack[1]	N/A
overflow[1]	N/A

Notes:
1. Write handshaking signals are only impacted by a write operation.

Table 3-14 defines the read port flags update latency due to a write operation.

Table 3-14: **Read Port Flags Update Latency Due to Write Operation**

Signals	Latency (clk)
empty	0
almost_empty	0
prog_empty	0
valid[1]	N/A
underflow[1]	N/A
data_count	0

Notes:
1. Read handshaking signals are only impacted by a read operation.

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

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

Table 3-16: **Read Port Flags Update Latency due to Read Operation**

Signals	Latency (clk)
empty	0
almost_empty	0

Table 3-16: **Read Port Flags Update Latency due to Read Operation**

prog_empty	1
valid	0
underflow	0
data_count	0

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

Table 3-17: **Write Port Flags Update Latency Due to Read Operation**

Signals	Latency (clk)
full	0
almost_full	0
prog_full	1
wr_ack(1)	N/A
overflow(1)	N/A

Notes:
1. Write handshaking signals are only impacted by a write operation.

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

Table 3-18: **Read Port Flags Update Latency Due to Write Operation**

Signals	No Register	Embedded/ Interconnect Register	Embedded/ Interconnect Registers
	Latency (clk)	Latency (clk)	Latency (clk)
empty	2	2	3
almost_empty	1	1	2
prog_empty	1	1	2
valid(1)	N/A	N/A	N/A
underflow(1)	N/A	N/A	N/A
data_count	0	0	0

Notes:
1. Read handshaking signals are only impacted by a read operation.

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

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

Table 3-19: **Write Port Flags Update Latency Due to a Write Operation**

Signals	Latency (wr_clk)
full	0

Table 3-19: **Write Port Flags Update Latency Due to a Write Operation**

almost_full	0
prog_full	1
wr_ack	0
overflow	0
wr_data_count	1

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

Table 3-20: **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 3-21 defines the write port flags update latency due to a read operation. *N* is the number of synchronization stages. In this example, *N* is 2.

Table 3-21: **Write Port Flags Update Latency Due to a Read Operation**

Signals	Latency
full	1 rd_clk + (N + 2) wr_clk (+1 wr_clk)(1)
almost_full	1 rd_clk + (N + 2) wr_clk (+1 wr_clk)(1)
prog_full	1 rd_clk + (N + 3) wr_clk (+1 wr_clk)(1)
wr_ack(2)	N/A
overflow(2)	N/A
wr_data_count	1 rd_clk + (N + 2) wr_clk (+1 wr_clk)(1)

Notes:

1. The crossing clock domain logic in independent clock FIFOs introduces a 1 wr_clk uncertainty to the latency calculation.
2. Write handshaking signals are only impacted by a write operation.

Table 3-22 defines the read port flags update latency due to a write operation. *N* is the number of synchronization stages. In this example, *N* is 2.

Table 3-22: **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 + (N + 2) rd_clk (+1 rd_clk)(1)
almost_empty	1 wr_clk + (N + 2) rd_clk (+1 rd_clk)(1)
prog_empty	1 wr_clk + (N + 3) rd_clk (+1 rd_clk)(1)
valid(2)	N/A
underflow(2)	N/A
rd_data_count	1 wr_clk + (N + 2) rd_clk (+1 rd_clk)(1)

Notes:

1. The crossing clock domain logic in independent clock FIFOs introduces a 1 rd_clk uncertainty to the latency calculation.
2. Read handshaking signals are only impacted by a read operation.

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

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

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

Table 3-24: **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 3-25 defines the write port flags update latency due to a read operation. *N* is the number of synchronization stages. In this example, *N* is 2.

Table 3-25: **Write Port Flags Update Latency Due to a Read Operation**

Signals	Latency
full	1 rd_clk + (N + 2) wr_clk (+1 wr_clk)(1)
almost_full	1 rd_clk + (N + 2) wr_clk (+1 wr_clk)(1)
prog_full	1 rd_clk + (N + 3) wr_clk (+1 wr_clk)(1)
wr_ack(2)	N/A
overflow(2)	N/A
wr_data_count	1 rd_clk + (N + 2) wr_clk (+1 wr_clk)(1)

Notes:

1. The crossing clock domain logic in independent clock FIFOs introduces a 1 wr_clk uncertainty to the latency calculation.
2. Write handshaking signals are only impacted by a write operation.

Figure 3-26 defines the read port flags update latency due to a write operation. *N* is the number of synchronization stages. In this example, *N* is 2.

Table 3-26: **Independent Clock FWFT**

Signals	No Register	Embedded/Interconnect Register	Embedded and Interconnect Registers
	Latency(clk)	Latency(clk)	Latency(clk)
empty	1 wr_clk + (N + 4) rd_clk (+1 rd_clk)(1)	1 wr_clk + (N + 4) rd_clk (+1 rd_clk)(1)	1 wr_clk + (N + 5) rd_clk (+1 rd_clk)(1)
almost_empty	1 wr_clk + (N + 4) rd_clk (+1 rd_clk)(1)	1 wr_clk + (N + 4) rd_clk (+1 rd_clk)(1)	1 wr_clk + (N + 5) rd_clk (+1 rd_clk)(1)
prog_empty	1 wr_clk + (N + 3) rd_clk (+1 rd_clk)(1)	1 wr_clk + (N + 3) rd_clk (+1 rd_clk)(1)	1 wr_clk + (N + 4) rd_clk (+1 rd_clk)(1)
valid(2)	N/A	N/A	N/A
underflow(2)	N/A	N/A	N/A
rd_data_count	1 wr_clk + (N + 2) rd_clk (+1 rd_clk)(1) + [N rd_clk (+1 rd_clk)](3)	1 wr_clk + (N + 2) rd_clk (+1 rd_clk)(1) + [N rd_clk (+1 rd_clk)](3)	1 wr_clk + (N + 2) rd_clk (+1 rd_clk)(1) + [N rd_clk (+1 rd_clk)](3)

Notes:

1. The crossing clock domain logic in independent clock FIFOs introduces a 1 rd_clk uncertainty to the latency calculation.
2. Read handshaking signals are only impacted by a read operation.
3. 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 *7 Series FPGAs Memory Resources User Guide* (UG473) [Ref 3].

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

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

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

Table 3-29: **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)(2)
prog_full	N(2)
wr_ack(1)	N/A
overflow(1)	N/A

Notes:

1. Write handshaking signals are only impacted by a write operation.
2. Use N=1 for UltraScale devices with number of primitives in depth <=16

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

Table 3-30: **Read Port Flags Update Latency Due to Write Operation**

Signals	Latency (clk)
empty	(N[2]-1)*2
prog_empty	(N[2]-1)*2+1
valid[1]	N/A
underflow[1]	N/A

Notes:
1. Read handshaking signals are only impacted by a read operation.
2. Use N=1 for UltraScale devices with number of primitives in depth <=16

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 *7 Series FPGAs Memory Resources User Guide* (UG473) [Ref 3].

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

Table 3-31: **Write Port Flags Update Latency Due to Write Operation**

Signals	Latency (clk)
full	0
prog_full	1
wr_ack	0
overflow	0

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

Table 3-32: **Read Port Flags Update Latency Due to a Read Operation**

Signals	Latency (clk)
empty	0
prog_empty	1
valid	0
underflow	0

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

Table 3-33: **Write Port Flags Update Latency Due to a Read Operation**

Signals	Latency (clk)
full	(N-1)[2]
prog_full[1]	N[2]
wr_ack[1]	N/A
overflow	N/A

Notes:
1. Write handshaking signals are only impacted by a write operation.
2. Use N=1 for UltraScale devices with number of primitives in depth <=16

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

Table 3-34: **Read Port Flags Update Latency Due to a Write Operation**

Signals	Latency (clk)
empty	((N[2]-1)*2+1)
prog_empty	((N[2]-1)*2+1)
valid[1]	N/A
underflow[1]	N/A

Notes:
1. Read handshaking signals are only impacted by a read operation.
2. Use N=1 for UltraScale devices with number of primitives in depth <=16

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 *7 Series FPGAs Memory Resources User Guide* (UG473) [Ref 3].

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

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

Table 3-36: **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 3-37 defines the write port flags update latency due to a read operation.

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

Signals	Latency
full(1)	L1(2)rd_clk + (N(4)-1)*L2(2) faster_clk + L3(2)wr_clk
prog_full(1)	L4(2) rd_clk + (N(4)-1)*(L2(2)-1) faster_clk + L5(2)wr_clk
wr_ack(3)	N/A
overflow(3)	N/A

Notes:

1. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
2. L1 = 1, L2 = 4, L3 = 3, L4 = 1 and L5 = 4
3. Write handshaking signals are only impacted by a Write operation.
4. Use N=1 for UltraScale devices with number of primitives in depth <=16

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

Table 3-38: **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

Signals	Latency
empty(1)	L1(2) wr_clk + (N(4)-1)*L2(2) faster_clk + L3(2)rd_clk
prog_empty(1)	L4(2)wr_clk + (N(4)-1)*(L5(2)-1) faster_clk + L6(2)rd_clk
valid(3)	N/A

Table 3-38: **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

underflow[3]	N/A

Notes:
1. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
2. L1 = 1, L2 = 4, L3 = 4, L4 = 1, L5 = 5 and L6 = 4.
3. Read handshaking signals are only impacted by a read operation.
4. Use N=1 for UltraScale devices with number of primitives in depth <=16

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 *7 Series FPGAs Memory Resources User Guide* (UG473) [Ref 3].

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

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

Table 3-40: **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 3-41 defines the write port flags update latency due to a read operation.

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

Signals	Latency
full[1]	L1[2] rd_clk + (N[4]-1)*L2[2] faster_clk + L3[2] wr_clk
prog_full[1]	L4[2] rd_clk + (N[4]-1)*(L2[2]-1) faster_clk + L5[2]wr_clk
wr_ack[3]	N/A
overflow[3]	N/A

Notes:
1. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
2. L1 = 1, L2 = 4, L3 = 3, L4 = 1 and L5 = 4.
3. Write handshaking signals are only impacted by a Write operation.
4. Use N=1 for UltraScale devices with number of primitives in depth <=16

Table 3-42 defines the read port flags update latency due to a write operation.

Table 3-42: **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

Signals	Latency
empty[1]	L1[2] wr_clk + (N[4]-1)*L2[2] faster_clk + L3[2]rd_clk
prog_empty[1]	L4[2] wr_clk + (N[4]-1)*(L5[2]-1) faster_clk + L6[2]rd_clk
valid[3]	N/A
underflow[3]	N/A

Notes:
1. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
2. L1 = 1, L2 = 5, L3 = 4, L4 = 1, L5 = 5 and L6 = 4.
3. Read handshaking signals are only impacted by a read operation.
4. Use N=1 for UltraScale devices with number of primitives in depth <=16

Special Design Considerations

Resetting the FIFO

The FIFO Generator core 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 7 series 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.
- For UltraScale common clock block RAM, distributed RAM and Shift Register FIFO with synchronous reset, the FIFO Generator core uses the UltraScale architecture built-in FIFO's reset mechanism. For more information on reset mechanism, see the *UltraScale Architecture Memory Resources: Advance Specification User Guide* (UG573) [Ref 4].
- For UltraScale built-in FIFO implementations, the reset (`srst`) is synchronous to `clk`/`wr_clk`. If `srst` is asserted, the `wr_rst_busy` output asserts immediately after the rising edge of `wr_clk`, and remains asserted until the reset operation is complete. Following the assertion of `wr_rst_busy`, the internal reset is synchronized to the `rd_clk` domain. Upon arrival in the `rd_clk` domain, the `rd_rst_busy` is asserted and is held asserted until the resetting of all `rd_clk` domain signals is complete. At this time, `rd_rst_busy` is deasserted. In common-clock mode, this logic is simplified because the clock domain crossing is not required. For more information on reset mechanism, see the *UltraScale Architecture Memory Resources: Advance Specification User Guide* (UG573) [Ref 4].

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

Continuous Clocks

The FIFO Generator core 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.

Note: If the clock(s) is/are lost during the normal operation, use `BUFGCE` to make clock line flat to ensure that the FIFO Generator core is not operating on a perturbed clock. It is highly recommended to apply reset once the clock(s) is/are stable.

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

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 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)
- N = number of synchronization stages. In this example, N = 2

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 + (3 + N)*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 3-21, page 143.

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 + (3 + N)*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 3-22, page 144.

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 or common clock non-symmetric Block RAM FIFO (UltraScale architecture only) is 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 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.

Design Flow Steps

This chapter describes customizing and generating the core, constraining the core, and the simulation, synthesis and implementation steps that are specific to this IP core. More detailed information about the standard Vivado® design flows and the IP integrator can be found in the following Vivado Design Suite user guides:

- *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994) [Ref 5]
- *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7]
- *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 8]
- *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 9]

Customizing and Generating the Native Core

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu .

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 8].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

The Native FIFO Interface IDE includes the following configuration tabs:

- Basic Tab
- Native Ports Tab
- Status Flags Tab
- Data Counts Tab
- Summary Tab

Basic Tab

The Basic tab defines the component name and provides the interface options and configuration options for the core.

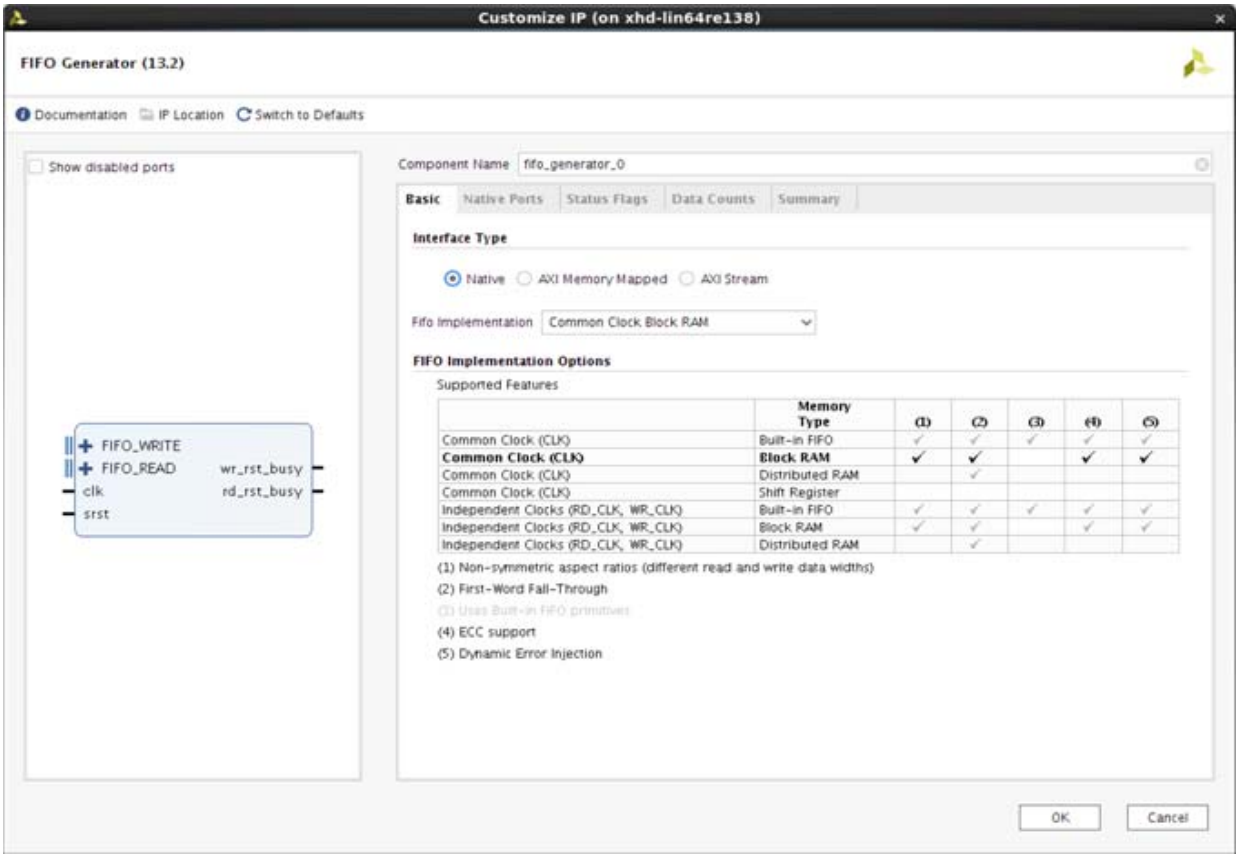


Figure 4-1: **Basic Tab**

- **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.
  - AXI Memory Mapped: Implements an AXI4, AXI3 and AXI4-Lite FIFOs in First-Word-Fall-Through mode.
  - AXI Stream: Implements an AXI4-Stream FIFO in First-Word-Fall-Through mode.
- **FIFO Implementation**
  - 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 Status Flags tab, Figure 4-3).

- 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 Status Flags tab, Figure 4-3).
- 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 Status Flags tab, Figure 4-3).
- 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 Status Flags tab, Figure 4-3).
- 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 Status Flags tab, Figure 4-3).
- 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 Status Flags tab, Figure 4-3).

- **Synchronization stages:** Defines the number of synchronizers stages across the cross clock domain logic. Table 4-1 shows the examples of synchronization stages with the frequency of operation.

Table 4-1: **Examples of Synchronization Stages**

Architecture	Frequency (MHz)	Speed grade	Number of synchronization stages
Virtex-7/Kintex-7	200	3	3
	250		3
	300		3
	400		4
artix-7/zynq 7000	200	3	2
	250		2
	300		3

Native Ports Tab

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

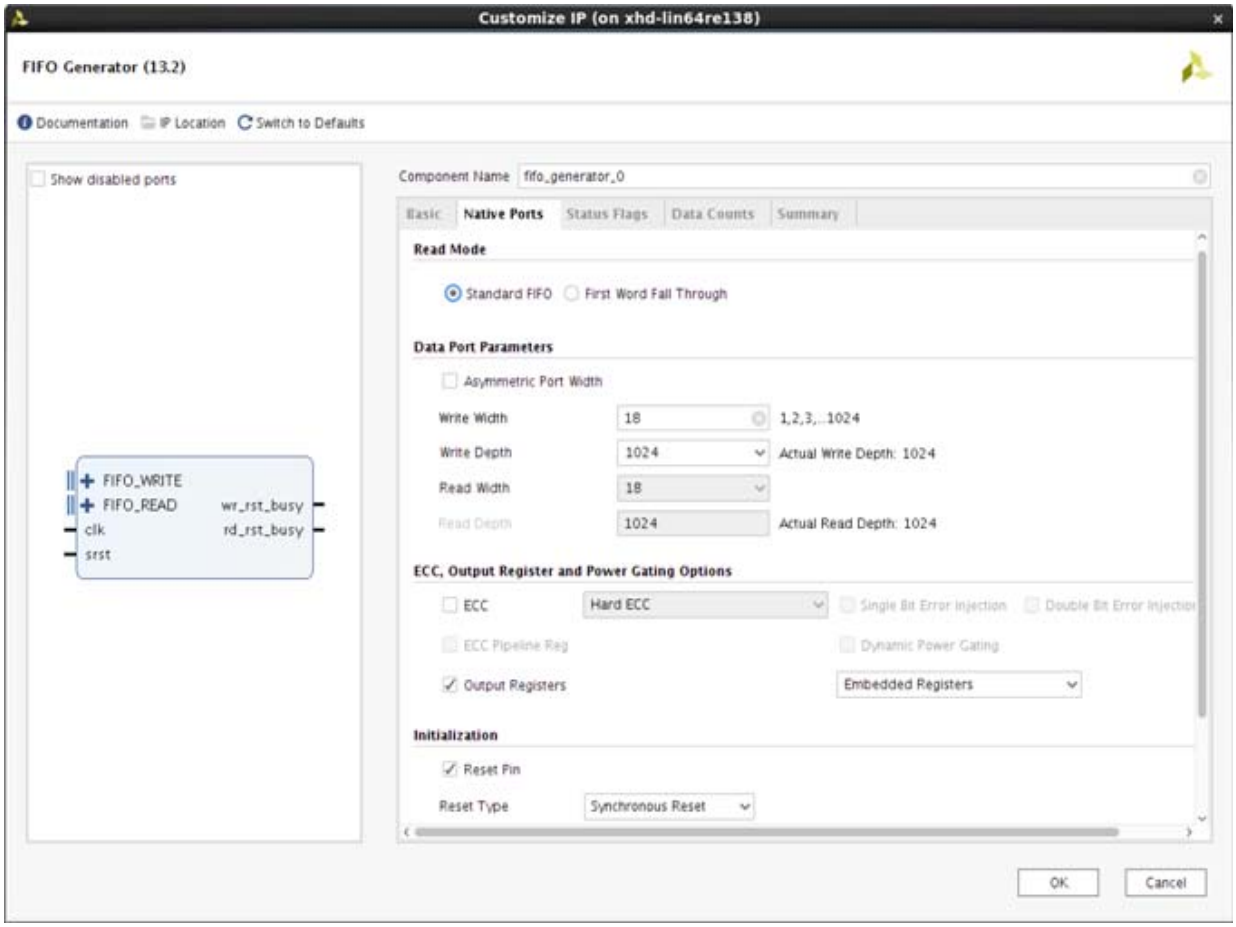


Figure 4-2: **Native Ports Tab**

- **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**
  - Asymmetric Port Width: Available only for UltraScale devices for Block RAM/Built-in FIFOs.
  - Write Width: The valid range of write data width is 1 to 1024.

- Write Depth: The valid range of write depth is 16 to 131072. Only depths with powers of 2 are allowed.
- Read Width: Available only if configuration with block RAM or built-in FIFO 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.
- Low Latency: Available only for UltraScale device built-in FIFO configurations where the number of primitives in depth is more than one.
- Low Latency Output Register: Available only when **Low Latency** option is selected.

- **ECC, Output Register and Power Gating Options**
  - ECC: When the Error Correction Checking (ECC) feature is enabled with Hard ECC, 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. Soft ECC option with ECC logic using the general interconnect is available in Block RAM based FIFO builds. 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.
  - ECC Pipeline Reg: The built-in FIFO macro has an optional pipeline register on the ECC decoder and encoder path. When the ECC pipeline register is selected, it improves the FIFO macro timing and adds one clock additional latency on the DOUT. The ECC pipeline register holds the previously read data in the pipeline when a reset is applied to the FIFO. For more information on ECC Pipeline Register, see *UltraScale Architecture Memory Resources: Advance Specification User Guide* (UG573) [Ref 4].
  - Dynamic Power Gating: The dynamic power saving capability (controlled by the sleep pin) keeps the built-in FIFO macro in sleep mode while preserving the data content. Any FIFO access prior to the wake-up time requirement is not guaranteed and might cause memory content corruption. While sleep is active (High), the `wr_en` and `rd_en` pins must be held Low. The data content of the memory is preserved during this mode. For more information on Dynamic Power Gating, see *UltraScale Architecture Memory Resources: Advance Specification User Guide* (UG573) [Ref 4].
  - Output Registers: The block RAM macros have built-in embedded or interconnect 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.

FIFO Generator also gives the option to choose both embedded and interconnect registers to have a smooth latency in meeting the timing.

- **Initialization**
  - Reset Pin: For FIFOs implemented with block RAM or distributed RAM, a reset pin is not required, and the input pin is optional.
  - Enable Reset Synchronization: Optional selection available only for independent clock block RAM or distributed RAM FIFOs. When unchecked, `wr_rst/rd_rst` is available. See Resets in Chapter 3 for details.
  - Enable safety Circuit: Optional selection is available only for BRAM based FIFOs with asynchronous reset (enabled by default). For AXI interface, Safety Circuit is automatically enabled inside the core as the reset in AXI interface is always asynchronous. When you select the Enable Safety Circuit option, the additional logic is enabled to ensure that the synchronous flops drive the FIFO Generator's outputs and the control signals into the BRAM primitives. See *Answer Record(AR#42571)*[Ref 11] for more information.
  - Reset Type:
    - 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, you can choose the reset value of the full flags (`prog_full`, `almost_full`, and `full`) during reset.
  - Dout Reset Value: Indicates the hexadecimal value asserted on the output of the FIFO when the reset is asserted. Available for all implementations using block RAM, distributed RAM, shift register or common clock 7 series devices' 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 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 with a limitation for UltraScale architecture-based devices using Built-In FIFOs.

IMPORTANT: *For UltraScale Built-in FIFOs, the `dout` output of the FIFO resets to the defined **Dout Reset Value** when the reset is asserted irrespective of the **Use Dout Reset** selection.*

See Appendix D, dout Reset Value Timing for timing diagrams for different configurations.

Status Flags Tab

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

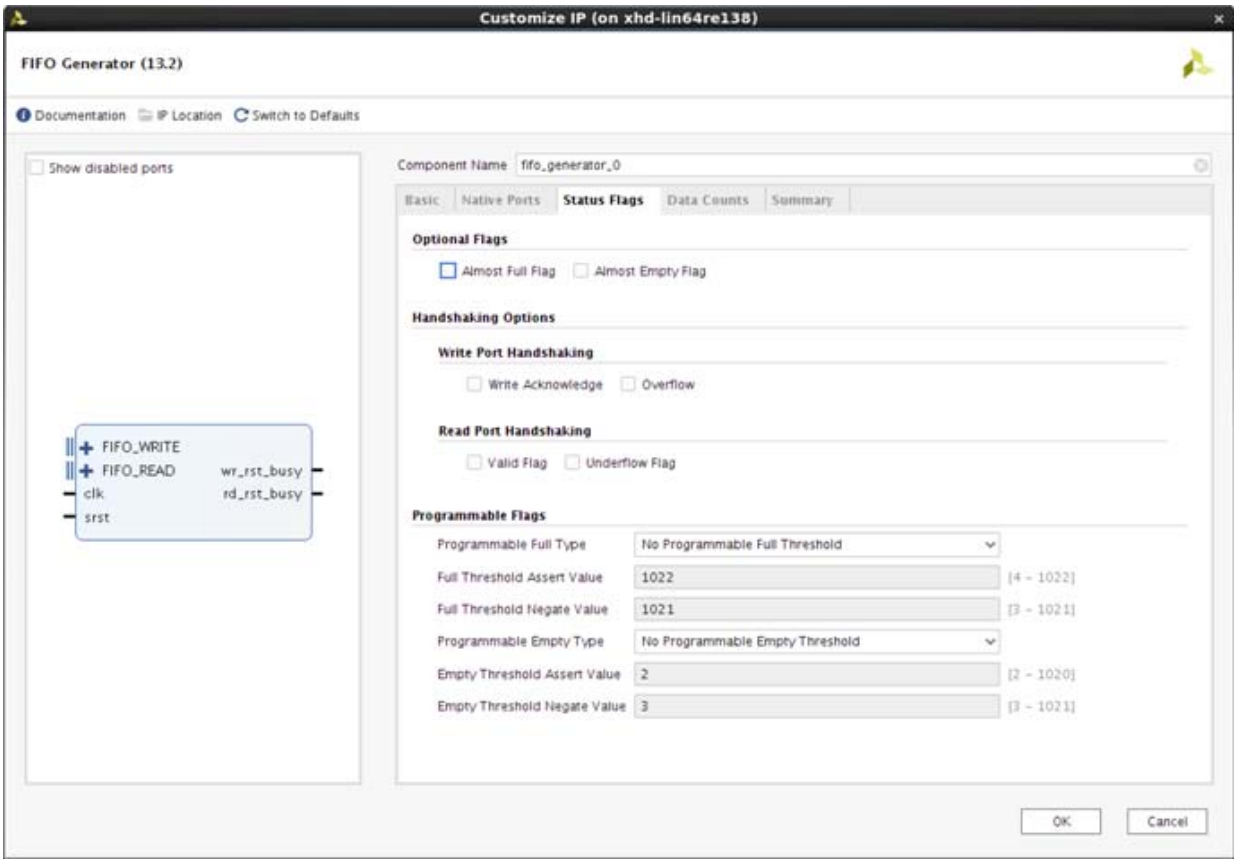


Figure 4-3: **Status Flags Tab**

- **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).
  - Read Port Handshaking
    - Valid (Read Acknowledge): Generates valid flag that indicates when the data on the output bus is valid. This signal can be configured to be active-High or Low (default active-High).

      Note: Active-Low option is available only for 7 series devices.
    - Underflow (Read Error): Generates underflow flag to indicate that the previous read request was not successful. This signal can be configured to be active-High or Low (default active-High).

      Note: Active-Low option is available only for 7 series devices.

- **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 IDE.
  - 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 IDE. 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 IDE.
  - 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 IDE.
  - 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 IDE. 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 IDE.

TIP: *For 7 series devices with Built-in FIFO configurations, `prog_full` and `prog_empty` signals are connected to `almostfull` and `almostempty` of the FIFO18E1/FIFO36E1 primitive.*

Data Counts Tab

Use this tab to set data count options.

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

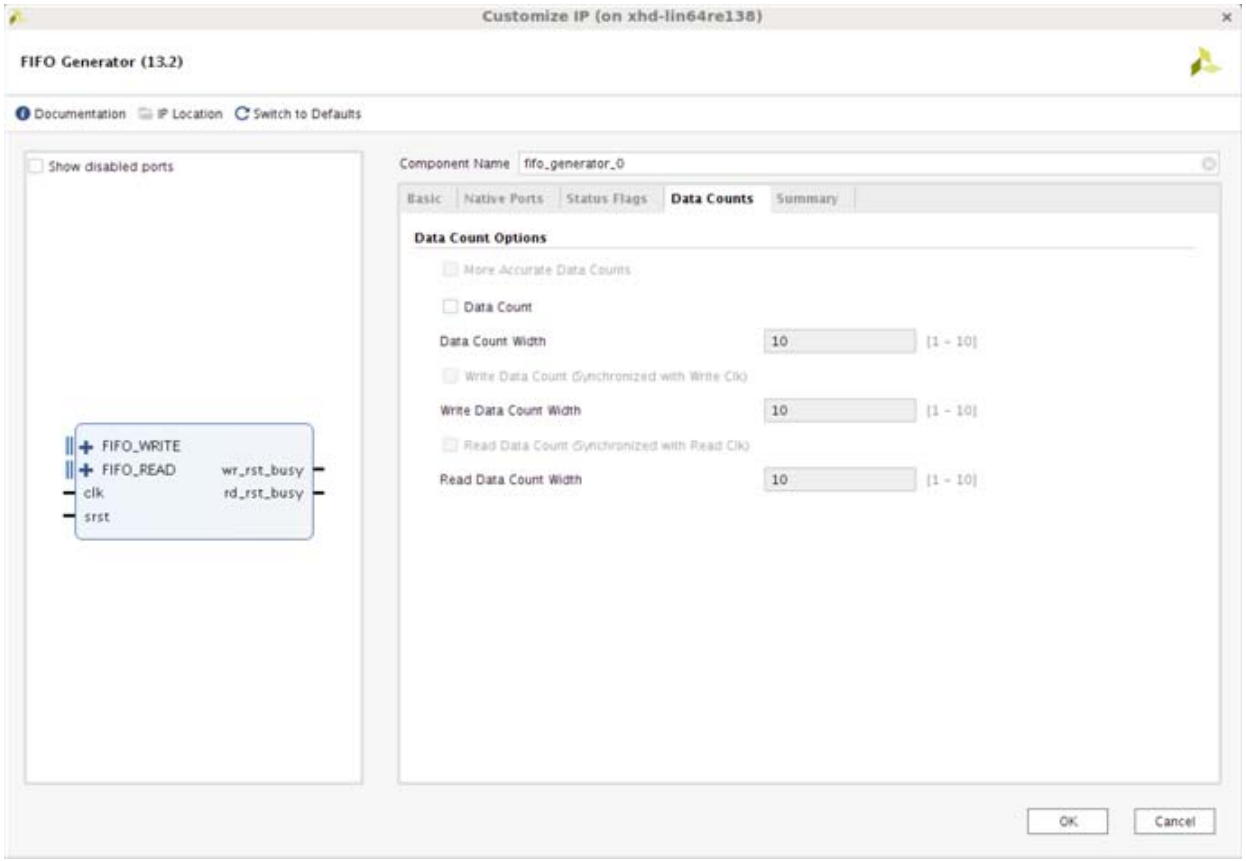


Figure 4-4: **Data Count Tab**

- **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 Tab

This tab 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 IDE.

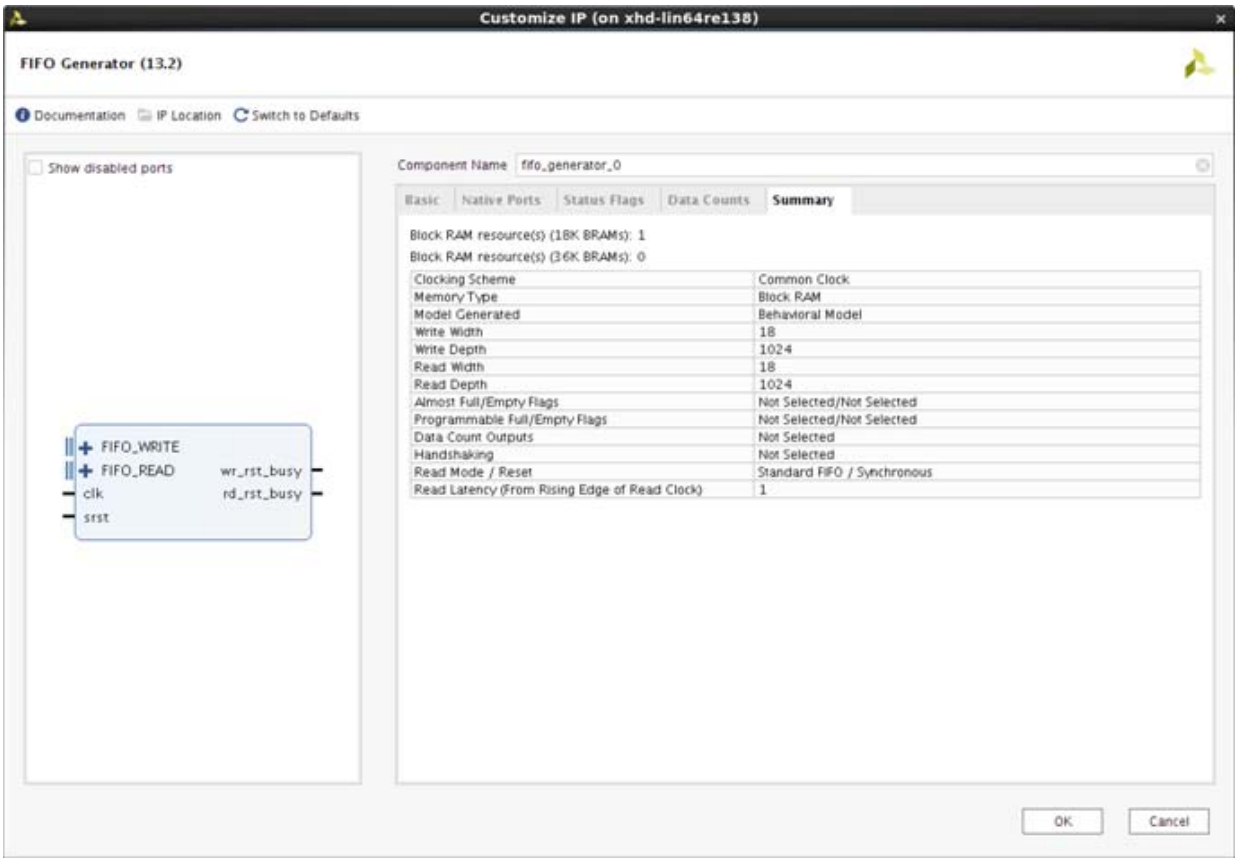


Figure 4-5: **Summary Tab**

User Parameters

Table 4-2 shows the relationship between the Vivado IDE and the User Parameters (which can be viewed in the Tcl console).

Table 4-2: **Vivado IDE Parameter to User Parameter Relationship**

Vivado IDE Parameter	User Parameter	Default Value
Interface Type Native AXI Memory Mapped AXI Stream	interface_type native axi_memory_mapped axi_stream	native
FIFO Implementation Common Clock Block RAM Common Clock Distributed RAM Common Clock Shift Register Common Clock Builtin FIFO Independent Clock Block RAM Independent Clock Distributed RAM Independent Clock Builtin FIFO	fifo_implementation common_clock_block_ram common_clock_distributed_ram common_clock_shift_register common_clock_builtin independent_clock_block_ram independent_clock_distributed_ram independent_clock_builtin	common_clock_block_ram (7 series devices) common_clock_builtin (UltraScale devices)

Table 4-2: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter	User Parameter	Default Value
Synchronization Option 2 to 8	synchronization_stages 2 to 8	2
Read Mode Standard FIFO First Word Fall Through	performance_options standard_fifo first_word_fall_through	standard_fifo
Asymmetric Port Width true, false	asymmetric_port_width true, false	false
Write Width 1 to 1024	input_data_width 1 to 1024	18
Write Depth 16 to 131072	input_depth 16 to 131072	1024
Read Width 1 to 1024	output_data_width 1 to 1024	18
Read Depth 16 to 131072	output_depth 16 to 131072	1024
Low Latency true, false	enable_low_latency true, false	false
Output Register true, false	use_dout_register true, false	false
Enable ECC true, false	enable_ecc true, false	false
Enable ECC Type Hard ECC Soft ECC	Enable ECC Type Hard ECC Soft ECC	Hard ECC
Single Bit Error Injection true, false	inject_sbit_error true, false	false
Double Bit Error Injection true, false	inject_dbit_error true, false	false
ECC Pipeline Register true, false	ecc_pipeline_reg true, false	false
Output Registers true, false	use_embedded_registers true, false	false
Output Register Type Embedded Register Fabric Register Embedded and Fabric Register	Output_Regsiter_Type Embedded_Reg Fabric_Reg Both	Embedded Register
Dynamic Power Gating true, false	dynamic_power_saving true, false	false
Reset Pin true, false	reset_pin true, false	true

Table 4-2: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter	User Parameter	Default Value
Enable Reset Synchronization true, false	enable_reset_synchronization true, false	true
Enable Safety Circuit true, false	enable_safety_circuit true,false	true
Reset Type Asynchronous Reset Synchronous Reset	reset_type asynchronous_reset synchronous_reset	Asynchronous Reset
Full Flags Reset Value 0, 1	full_flags_reset_value 0, 1	0
Dout Reset Value true, false	use_dout_reset true, false	true
	dout_reset_value	0
Almost Full Flag true, false	almost_full_flag true, false	false
Almost Empty Flag true, false	almost_empty_flag true, false	false
Read Clock Frequency 1 to 1000	read_clock_frequency 1 to 1000	1
Write Clock Frequency 1 to 1000	write_clock_frequency 1 to 1000	1
Write Acknowledge true, false	write_acknowledge_flag true, false	Active High
Write Acknowledge Sensitivity(1) Active High Active Low	write_acknowledge_sense active_high active_low	Active High
Overflow Flag true, false	overflow_flag true, false	false
Overflow Flag Sensitivity(1) Active High Active Low	overflow_sense active_high active_low	Active High
Valid Flag true, false	valid_flag true, false	false
Valid Flag Sensitivity(1) Active High Active Low	valid_sense active_high active_low	Active High
Underflow Flag true, false	underflow_flag true, false	false
Underflow Flag Sensitivity(1) Active High Active Low	underflow_sense active_high active_low	Active High

Table 4-2: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter	User Parameter	Default Value
Programmable Full Type No Programmable Full Threshold Single Programmable Full Threshold Constant Multiple programmable Full Threshold Constants Single Programmable Full Threshold Input Port Multiple Programmable Full Threshold Input Ports	programmable_full_type No_Programmable_Full_Threshold Single_Programmable_Full_Threshold_Constant Multiple_programmable_Full_Threshold_Constants Single_Programmable_Full_Threshold_Input_Port Multiple_Programmable_Full_Threshold_Input_Ports	No_Programmable_Full_Threshold
Full Threshold Assert Value 4 to 1022 (7 series devices) 5 to 1023 (UltraScale devices)	full_threshold_assert_value	1022 (7 series devices) 1023(UltraScale devices)
Full Threshold Negate Value 3 to 1021 (7 series devices) 1 to 1022 (UltraScale devices)	full_threshold_negate_value	1021 (7 series devices) 1022 (UltraScale devices)
Programmable Empty Type No Programmable Empty Threshold Single Programmable Empty Threshold Constant Multiple programmable Empty Threshold Constants Single Programmable Empty Threshold Input Port Multiple Programmable Empty Threshold Input Ports	programmable_empty_type No_programmable_empty_threshold Single_Programmable_empty_Threshold_Constant Multiple_programmable_empty_Threshold_Constants Single_Programmable_empty_Threshold_Input_Port Multiple_Programmable_empty_Threshold_Input_Ports	No_Programmable_Empty_Threshold
Empty Threshold Assert Value 2 to 1020 (7 series devices) 2 to 1022 (UltraScale devices)	empty_threshold_assert_value	2
Empty Threshold Negate Value 3 to 1021 (7 series devices) 3 to 1023 (UltraScale devices)	empty_threshold_negate_value	3
More Accurate Data Counts true, false	use_extra_logic true, false	false
Data Count true, false	data_count true, false	false
Data Count Width 1 - $\log_2$(Write Depth)+1	Data_count_width 1 - $\log_2$(Write Depth)+1	10
Write Data Count true, false	wr_data_count true, false	false
Write Data Count Width 1 - $\log_2$(Write Depth)+1	wr_data_count_width 1 - $\log_2$(Write Depth)+1	10

Table 4-2: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter	User Parameter	Default Value
Read Data Count true, false	rd_data_count true, false	false
Read Data Count Width 1 - $\log_2$(Read Depth)+1	rd_data_count_width 1 - $\log_2$(Read Depth)+1	10
Disable timing violations on cross clock domain register true, false	Disable_timing_violations true, false	false
PROTOCOL AXI4 AXI3 AXI4 Lite	axi_type AXI4 AXI3 AXI4_Lite	AXI4
Read Write Mode Read Write Read Only Write Only	read_write_mode read_write read_only write_only	Read Write
Clock Type AXI Common Clock Independent Clock	clock_type_axi common_clock independent_clock	Common Clock
Synchronization Stages across Cross Clock Domain Logic 2 to 8	synchronization_stages_(2) 2 to 8	2
ID Width 0 to 32	Id_width 0 to 32	0
Address Width 1 to 64	address_width 1 to 64	32
Data Width 32 to 1024 (power of 2)	data_width 32 to 1024	64
AWUSER Width 1 to 256	awuser_width 1 to 256	1
WUSER Width 1 to 256	wuser_width 1 to 256	1
BUSER Width 1 to 256	buser_width 1 to 256	1
ARUSER Width 1 to 256	aruser_width 1 to 256	1
RUSER Width 1 to 256	ruser_width 1 to 256	1
Configuration Options FIFO Register Slice Pass Through Wire	_type(2) FIFO Register Slice Pass Through Wire	FIFO

Table 4-2: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter	User Parameter	Default Value
FIFO Implementation Type Common Clock Block RAM Common Clock Distributed RAM	Fifo_implementation_[2] Common Clock Block RAM Common Clock Distributed RAM	Common Clock Distributed RAM
FIFO Application type Data FIFO Packet FIFO Low Latency Data FIFO	Fifo_application_type_[2] Data FIFO Packet FIFO Low Latency Data FIFO	Data FIFO
FIFO Depth 16 to 4194304 (multiples of 2 from 16 to 2^22)	input_depth_[2]	Write Address Channel: 16 Write Data Channel: 512 Write Response Channel: 16 Read Address Channel: 16 Read Data Channel: 512 If AXI Type = AXI Stream, AXI Stream: 512
Enable ECC true, false	enable_ecc_[2] true, false	false
Single Bit Error Injection true, false	inject_sbit_error_[2] true, false	false
Double Bit Error Injection true, false	inject_sbit_error_[2] true, false	false
Provide FIFO Occupancy Data Counts true, false	enable_data_Counts_[2] true, false	false
Programmable Full Type No Programmable Full Threshold Single Programmable Full Threshold Constant Multiple programmable Full Threshold Constants Single Programmable Full Threshold Input Port Multiple Programmable Full Threshold Input Ports	programmable_full_type_[2] No Programmable Full Threshold Single Programmable Full Threshold Constant Multiple programmable Full Threshold Constants Single Programmable Full Threshold Input Port Multiple Programmable Full Threshold Input Ports	No Programmable Full Threshold
Full Threshold Assert Value 4 to 1022 (7 series devices) 5 to 1023 (UltraScale devices)	full_threshold_assert_value_[2]	1022 (7 series devices) 1023 (UltraScale devices)

Table 4-2: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter	User Parameter	Default Value
Programmable Empty Type No Programmable Empty Threshold Single Programmable Empty Threshold Constant Multiple programmable Empty Threshold Constants Single Programmable Empty Threshold Input Port Multiple Programmable Empty Threshold Input Ports	programmable_empty_type_ (2) No Programmable Empty Threshold Single Programmable Empty Threshold Constant Multiple programmable Empty Threshold Constants Single Programmable Empty Threshold Input Port Multiple Programmable Empty Threshold Input Ports	No Programmable Empty Threshold
Empty Threshold Assert Value 2 to 1020 (7 series devices) 2 to 1022 (UltraScale devices)	empty_threshold_assert_value_ (2)	2
Underflow Flag true, false	underflow_flag_axi true, false	false
Underflow Flag Active High Active Low	underflow_sense_axi Active High Active Low	Active High
HAS ACLKEN true, false	has_acklen true, false	false
Clock Enable Type Slave Interface Clock Enable Master Interface Clock Enable	clock_enable_type slave_interface_clock_enable master_interface_clock_enable	Slave Interface Clock Enable
TDATA NUM BYTES 0 to 512 ***Note:*** Range includes the values raised to the power of 2.	tdata_num_bytes 0 to 512 ***Note:*** Range includes the values raised to the power of 2.	1
TID WIDTH 0 to 32	tid_width 0 to 32	0
TDEST WIDTH 0 to 32	tdest_width 0 to 32	0
TUSER WIDTH 0 to 4096	tuser_width 0 to 4096	4
HAS TSTRB true, false	has_tstrb true, false	false
HAS TKEEP true, false	has_tkeep true, false	false
TREADY true, false	enable_tready true, false	true
TLAST true, false	enable_tlast true, false	false

Table 4-2: **Vivado IDE Parameter to User Parameter Relationship *(Cont'd)***

Vivado IDE Parameter	User Parameter	Default Value
Register Slice Fully Registered Light Weight	register_slice_mode_(2) fully_registered light_weight	Fully Registered
Disable timing violations on cross clock domain register true, false	Disable_timing_violations_axi true, false	false

Notes:

1. Not available for UltraScale devices.
2. This user parameter is suffixed with axis, wach, wdch, wrch, rach and rdch. For example, enable_ecc_axis, enable_ecc_wach, enable_ecc_wdch, enable_ecc_wrch, enable_ecc_rach and enable_ecc_rdch.
3. This user parameter is prefixed with axis, wach, wdch, wrch, rach and rdch. For example, axis_type, wach_type, wdch_type, wrch_type, rach_type and rdch_type.

Note: Independent clock distributed RAM based Constraints are updated to reduce the time taken during the implementation stage. This may trigger a false positive CDC-1 warning and can be ignored.

Note: When you use FIFO Generator IP in Memory Mapped Interface, the Vivado tool triggers CDC-11 as the reset synchronization inside the IP is repeated multiple times. You can safely ignore this CDC-11 as the FIFO Generator itself ensures that both the domains are out of reset before it de-asserts `wr_rst_busy` signal.

Output Generation

For details about files created when generating the core, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Customizing and Generating the AXI Core

You can customize the IP for use in your design by specifying values for the various parameters associated with the IP core using the following steps:

1. Select the IP from the IP catalog.
2. Double-click the selected IP or select the Customize IP command from the toolbar or right-click menu.

For details, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7] and the *Vivado Design Suite User Guide: Getting Started* (UG910) [Ref 8].

Note: Figures in this chapter are illustrations of the Vivado Integrated Design Environment (IDE). This layout might vary from the current version.

For AXI memory mapped, the FIFO Generator core includes the following configuration tabs:

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

  For AXI4/AXI3 and AXI4-Lite interfaces, FIFO Generator core 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 AXI4 Ports tab of the IDE is applied to all selected Channels of the AXI4/AXI3 or AXI4-Lite interfaces

More details on these customization IDE tabs are provided in the following sections.

Basic Tab

Figure 4-6 shows the Basic tab that includes AXI interface selection options.

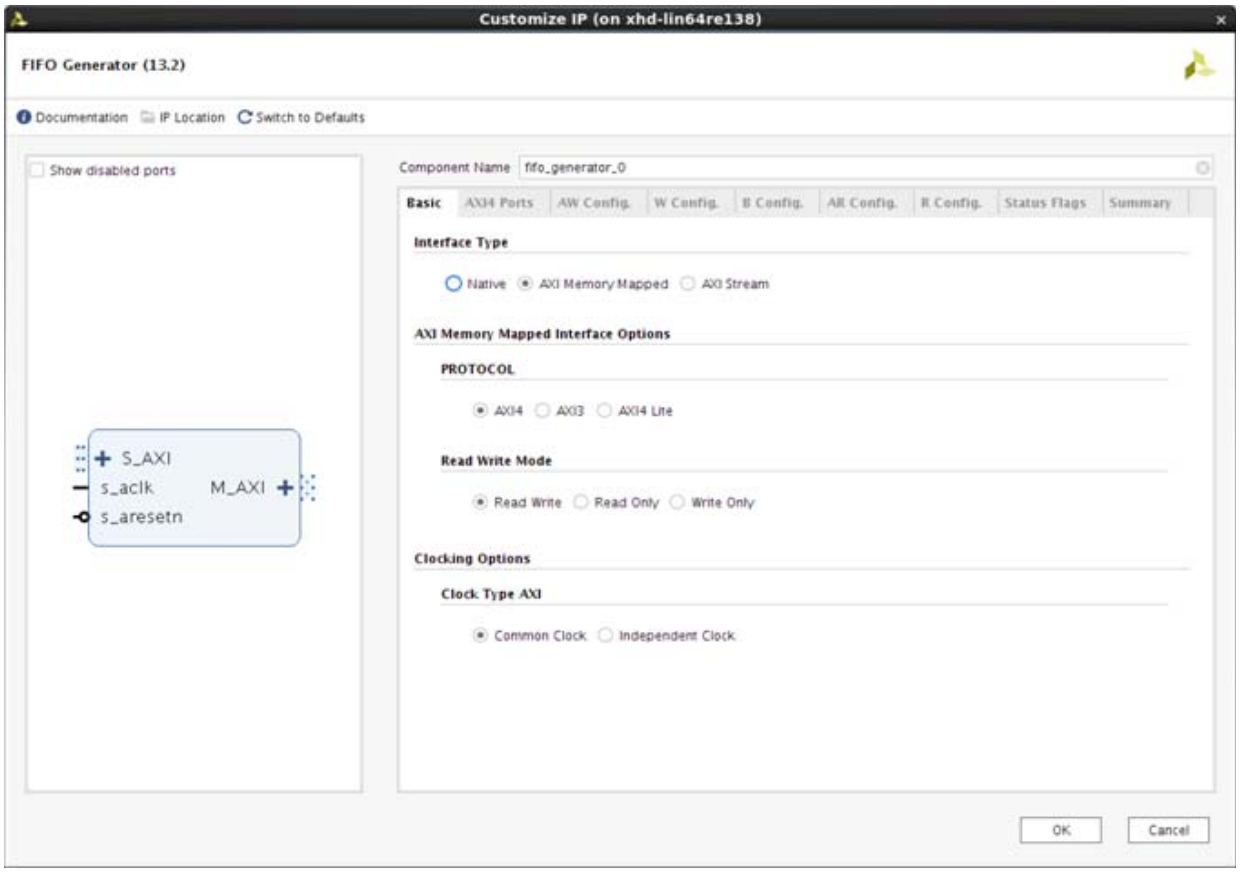


Figure 4-6: **Basic Tab**

- **AXI Interface Options:** Four AXI interface styles are available: AXI4-Stream, AXI4, AXI3 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 core 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 core 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

Note: This option is available only for AXI Stream Interface.

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

Performing Reads with Master Clock Enable

Note: This option is available only for AXI Stream Interface.

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

Ports Tabs

The AXI FIFO Width is determined by aggregating all of the channel information signals in a channel. The channel information signals for AXI4-Stream, AXI4, AXI3 and AXI4-Lite interfaces are listed in Table 4-3 and Table 4-4. IDE tabs are available for configuring:

- AXI4-Stream Ports Tab
- AXI4/AXI3 Ports Tab
- AXI4-Lite Ports Tab

AXI4-Stream Ports Tab

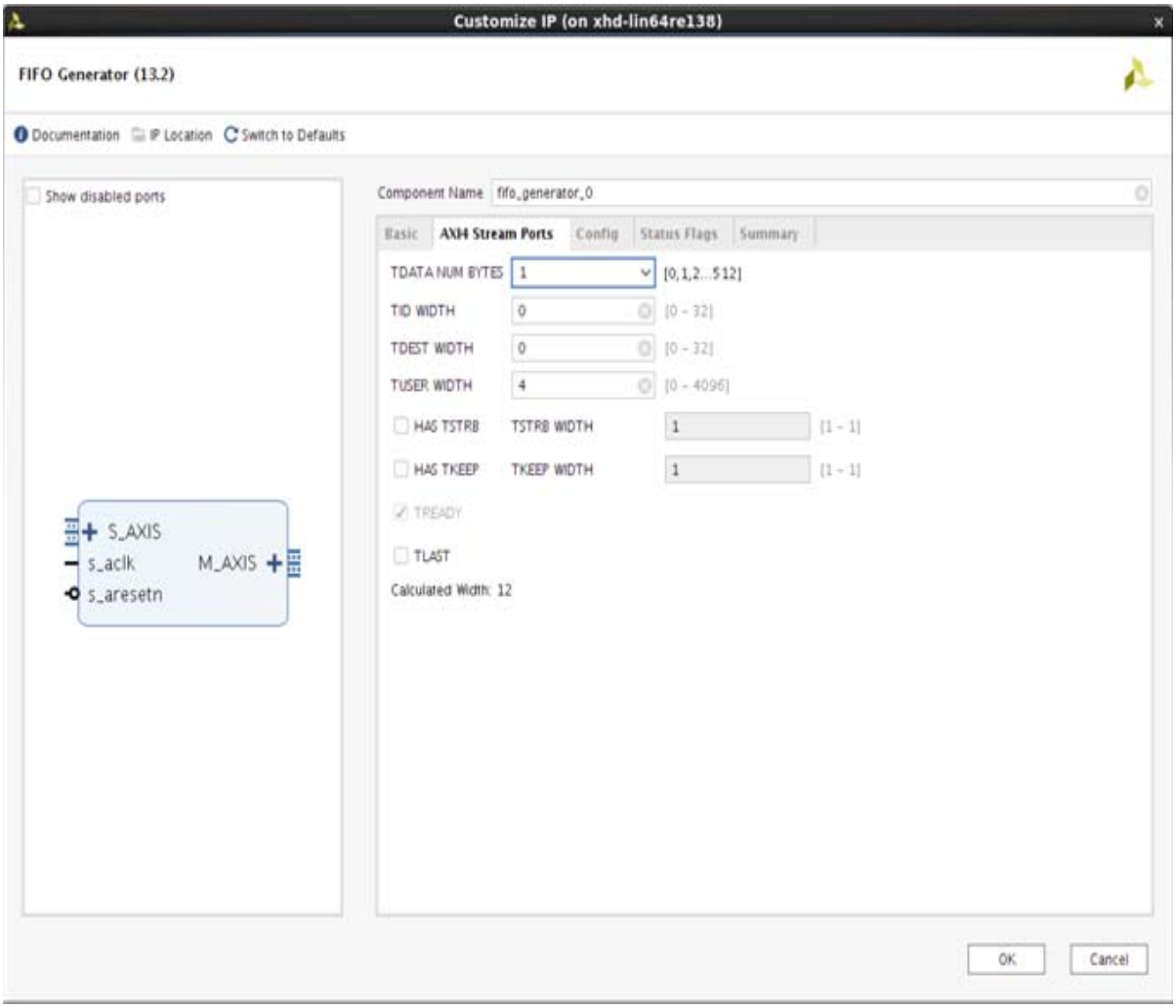


Figure 4-7: **AXI4-Stream Ports Tab**

The AXI4-Stream FIFO allows you to configure the 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/AXI3 Ports Tab

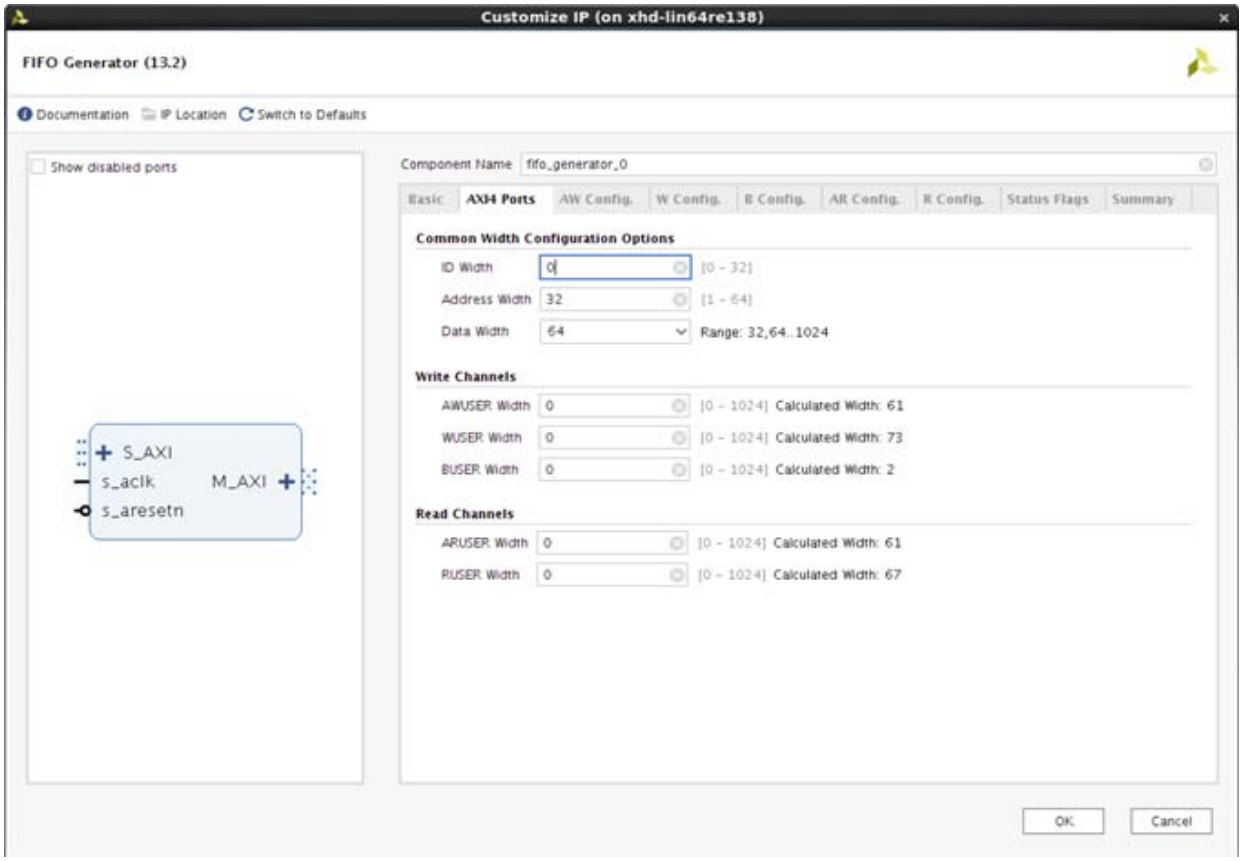


Figure 4-8: **AXI4/AXI3 Ports Tab**

The AXI4/AXI3 FIFO widths can be configured for ID, ADDR, DATA and USER signals. ID Width is applied to all channels in the AXI3 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 and ID signals are the only optional signals for the AXI4/AXI3 FIFO and can be independently configured for each channel.

For all the selected signals, the AXI4/AXI3 FIFO width for the respective channel is determined by summing up the widths of signals in the particular channel, as shown in Table 4-3.

Table 4-3: **AXI4/AXI3 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]	

Table 4-3: **AXI4/AXI3 Signals used in AXI FIFO Width Calculation *(Cont'd)***

Write Address Channel	Read Address Channel	Write Data Channel	Read Data Channel	Write Resp Channel
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 Ports Tab

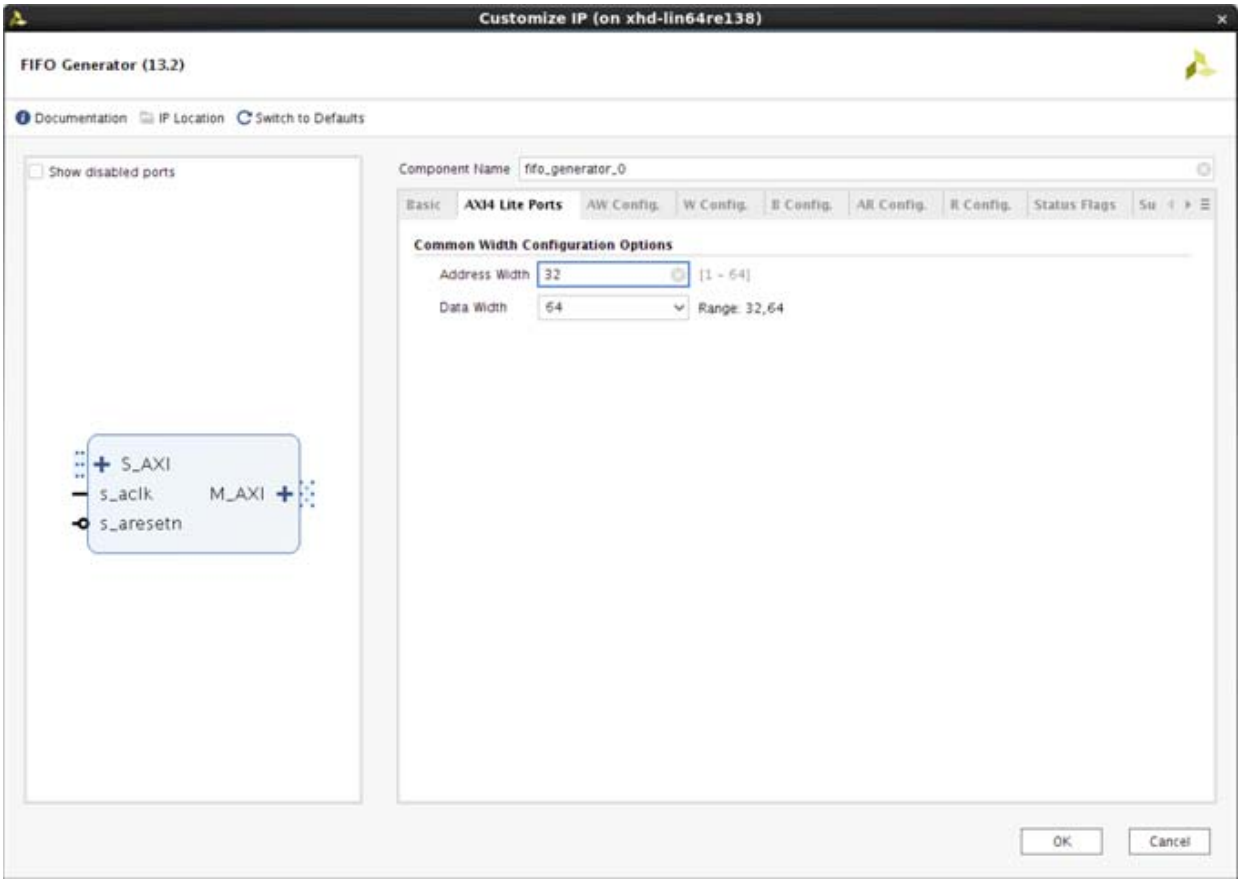


Figure 4-9: **AXI4-Lite Ports Tab**

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

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

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

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

Default Settings

Table 4-5 and Table 4-6 show the default settings for each Memory Mapped and AXI4 Stream interface type.

Table 4-5: **AXI FIFO Default Settings for 7 Series Devices**

Interface Type	Channels	Memory Type	FIFO Depth
AXI4 Stream	NA	Block Memory	1024
AXI4/AXI3	Write Address, Read Address, Write Response	Distributed Memory	16
AXI4/AXI3	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

Table 4-6: **AXI FIFO Default Settings for UltraScale Architecture**

Interface Type	Channels	Memory Type	FIFO Depth
AXI4 Stream	NA	Built-In	1024
AXI4/AXI3	Write Address, Read Address, Write Response	Distributed Memory	16
AXI4/AXI3	Write Data, Read Data	Built-In	1024
AXI4-Lite	Write Address, Read Address, Write Response	Distributed Memory	16
AXI4-Lite	Write Data, Read Data	Distributed Memory	16

AW/W/B/AR/R/AXI4-Stream Configuration Tab

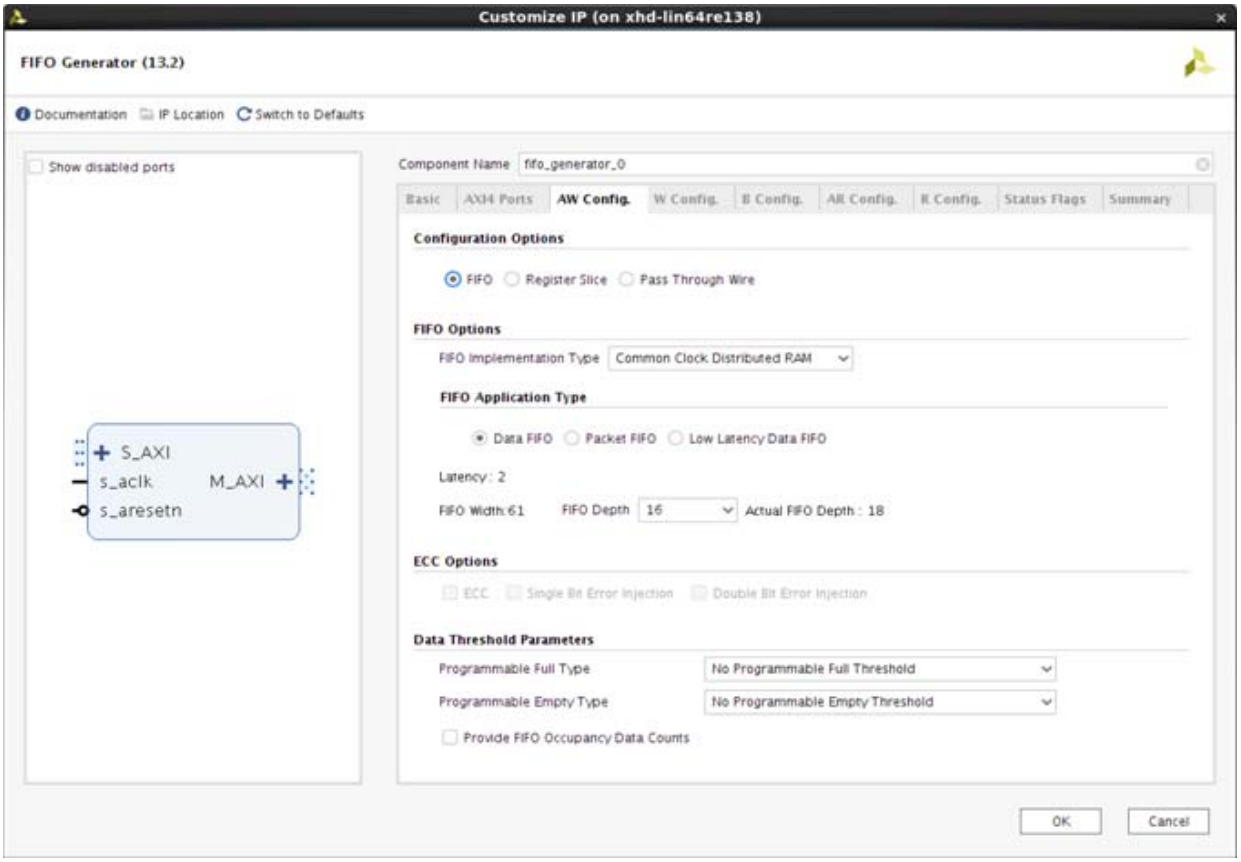


Figure 4-10: **AXI FIFO Configuration Tab**

The functionality of AXI 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).

For more details on first-word fall-through mode, see First-Word Fall-Through FIFO Read Operation in Chapter 3.

- **Configuration Options**: FIFO, Register Slice or Pass Through Wire.
- **FIFO Options**:
  - FIFO Implementation Type: The FIFO Generator core 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.
  - FIFO Application Type: Data FIFO, PAcket FIFO or Low Latency Data FIFO.
  - FIFO Width: AXI FIFOs support symmetric Write and Read widths. The width of the AXI FIFO is determined based on the selected Interface Type (AXI4-Stream, AXI4, AXI3 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 Ports Tabs, page 175.

- FIFO Depth: AXI FIFOs allow ranging from 16 to 131072. Only depths with powers of 2 are allowed.

- **ECC Options**: The block RAM and FIFO macros are equipped with built-in or general interconnect, error injection and correction checking. 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.

- **Data Threshold Parameters**:
  - 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 IDE.
    - 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 IDE.
  - 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 IDE.
    - 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 IDE.
  - Provide FIFO Occupancy Data Counts: 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.

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 log2(FIFO depth)+1. In common clock mode, the AXI 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 AXI transfer based on empty threshold

- Data Counts can be used to determine number of Transactions in the FIFO

Status Flags Tab

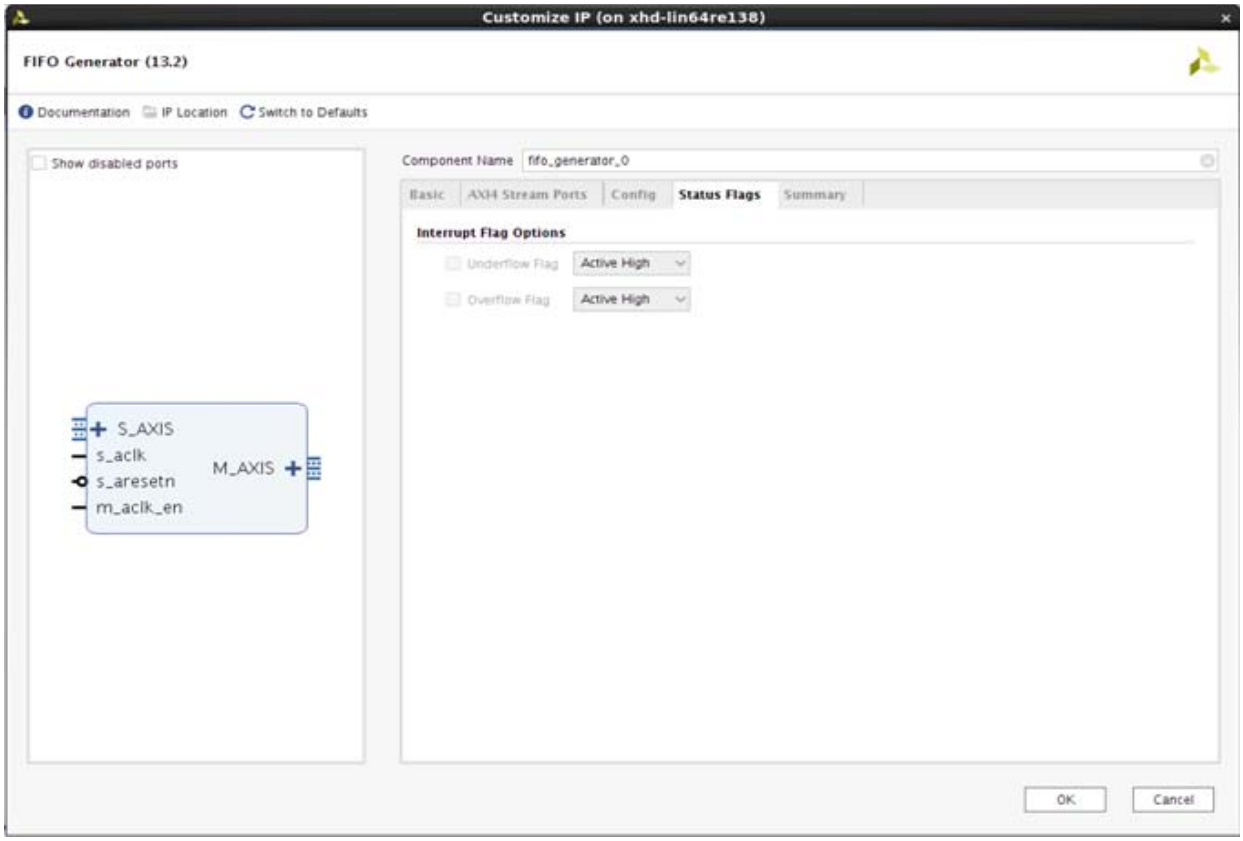


Figure 4-11: **Status Flags Tab**

- **Interrupt Flag Options**:
  - Underflow Flag: 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 Flag: 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 Tab

The Summary tab displays a summary of the AXI 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 AXI FIFO IDE.

Summary Tab (AXI4-Stream)

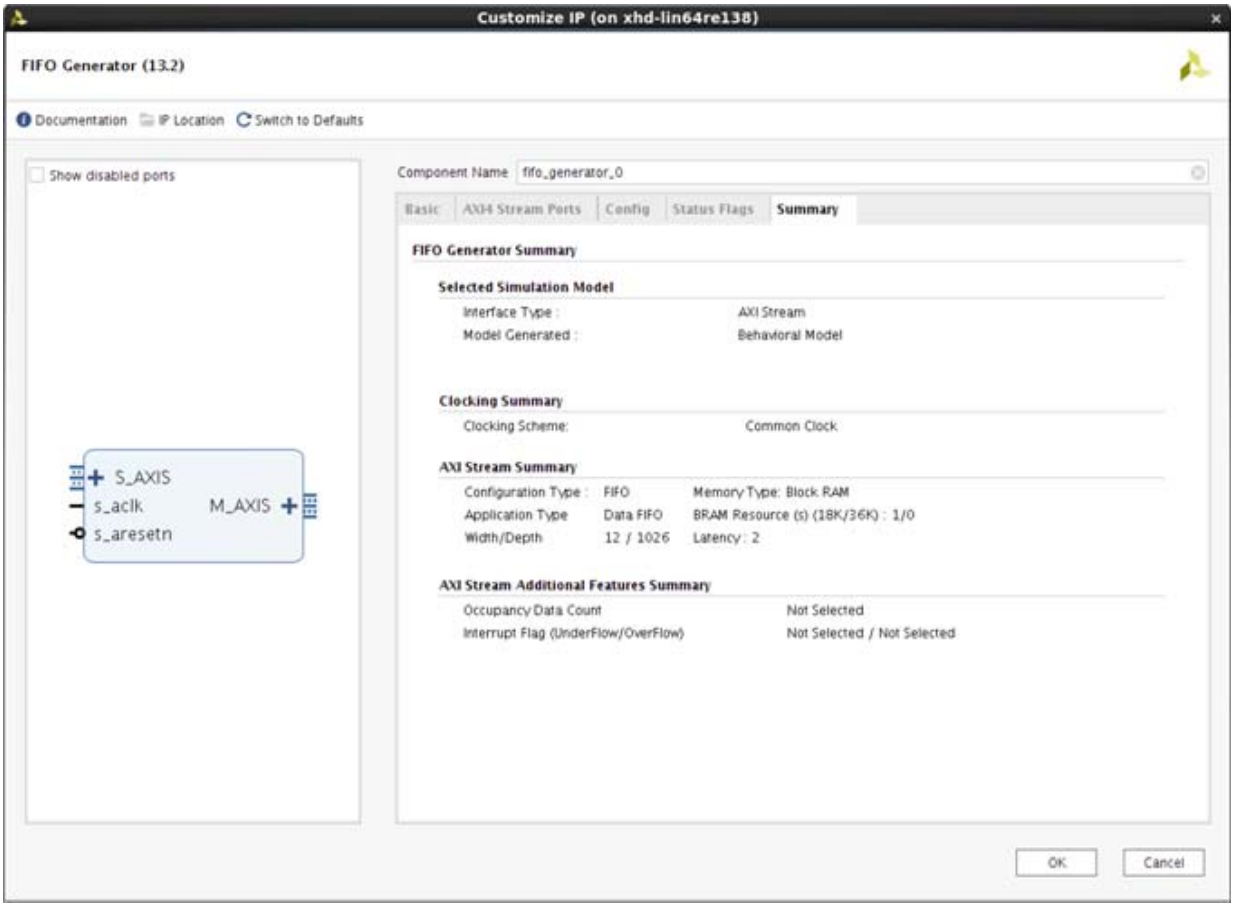


Figure 4-12: **AXI4-Stream Summary Tab**

Summary Tab (AXI Memory Mapped)

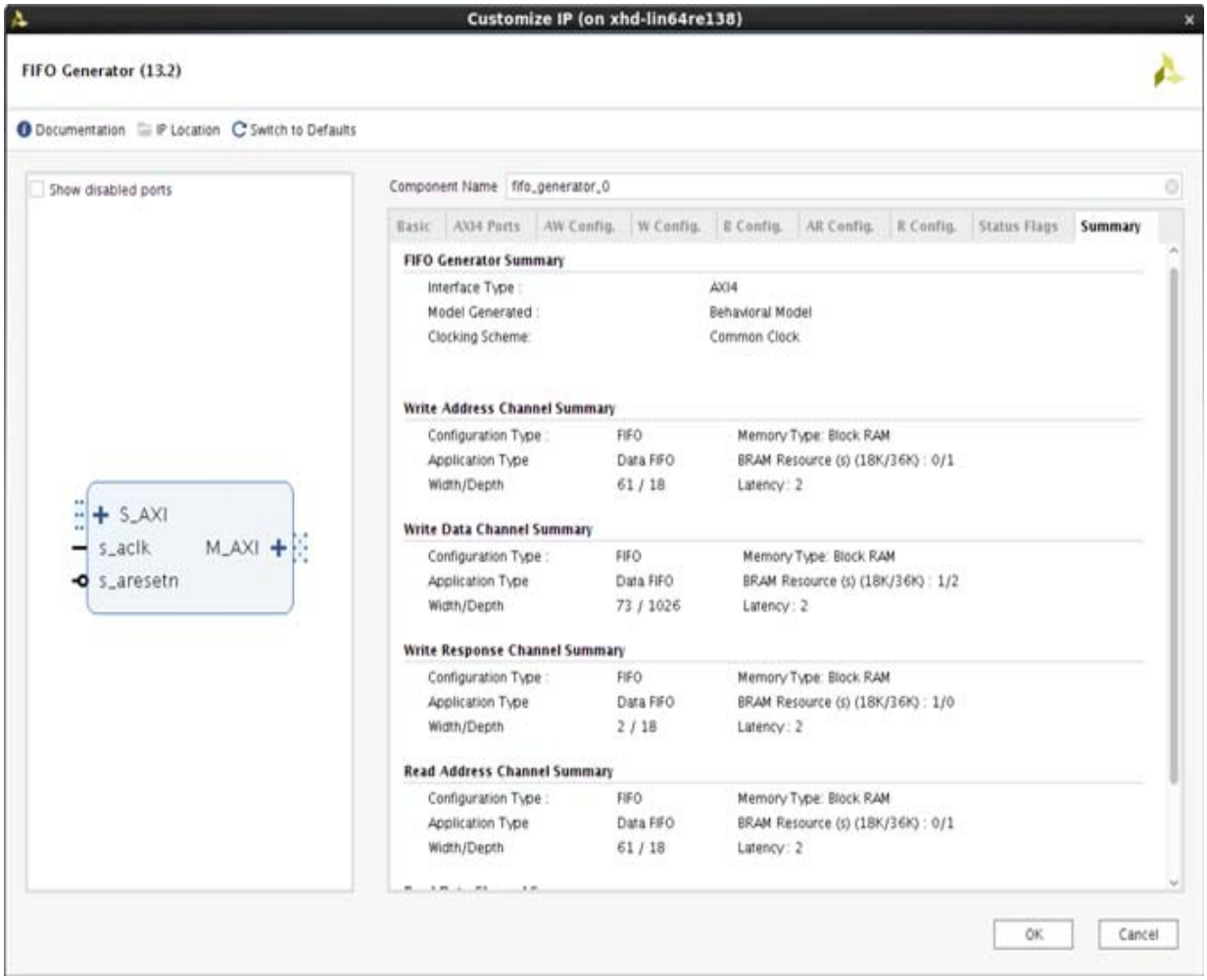


Figure 4-13: **AXI Memory Mapped Summary Tab**

User Parameters

For details about the relationship between the GUI fields in the Vivado IDE and the User Parameters, see Table 4-2.

Output Generation

For details about files created when generating the core, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Constraining the Core

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

Required Constraints

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

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.

Simulation

This section contains information about simulating IP in the Vivado® Design Suite.For details, see *Vivado Design Suite User Guide: Logic Simulation* (UG900) [Ref 9].

The FIFO Generator core supports Verilog simulation model.

IMPORTANT: *For cores targeting 7 series or Zynq-7000 devices, UNIFAST libraries are not supported. Xilinx IP is tested and qualified with UNISIM libraries only.*

Behavioral Model

IMPORTANT: *The behavioral model provided does not model synchronization delay, and is designed to reproduce the behavior and functionality of the FIFO Generator core. The model maintains the assertion/deassertion of the output signals to match the FIFO Generator core for the write/read operation (outside reset window). There may be one clock cycle (`clk/wr_clk/rd_clk`) difference between behavioral model and the core, if the asynchronous reset assertion/deassertion happens exactly at the rising edge of the clock (`clk/wr_clk/rd_clk`). It is highly recommended to maintain proper setup/hold window of all input signals in order to match with structural simulation as all input signals (except `clk`, `wr_clk`, `rd_clk`) are delayed by 100ps inside the model. If the core is configured without reset, it is recommended to wait for 15 slowest clock cycles at the beginning of simulation (from time 0).*

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

The following considerations apply to the behavioral model:

- Write operations always occur relative to the write clock (`wr_clk`) or common clock (`clk`) domain, as do the corresponding handshaking signals.
- Read operations always occur relative to the read clock (`rd_clk`) or common clock (`clk`) domain, as do the corresponding handshaking signals.
- The delay through the FIFO (write-to-read latency) will match the VHDL model, Verilog model, and core.
- The deassertion of the status flags (full, almost full, programmable full, empty, almost empty, programmable empty) will match the VHDL model, Verilog model, and core.

Note: If independent clocks or common clocks with built-in FIFO is selected, encrypted RTL file is used for simulation. For detailed information on file delivery structure, see Appendix E.

Structural (Post Synthesis/Implementation) Models

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

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

Synthesis and Implementation

For details about synthesis and implementation, see the *Vivado Design Suite User Guide: Designing with IP* (UG896) [Ref 7].

Detailed Example Design

Figure 5-1 shows the configuration of the example design.

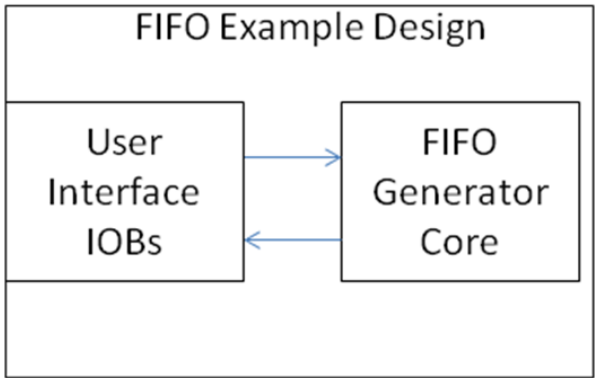


Figure 5-1: **Example Design Configuration**

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.

Implementing the Example Design

After generating a core, right click on the generated core and click **Open IP Example Design**. In the opened example project, click **Run Synthesis** and **Run Implementation** options to implement the example design.

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 (behavioral models) provided are either in VHDL or Verilog, depending on the CORE project settings in the Vivado tools.

Test Bench

This chapter contains information about the test bench provided in the Vivado® Design Suite.

Figure 6-1 shows a block diagram of the demonstration test bench.

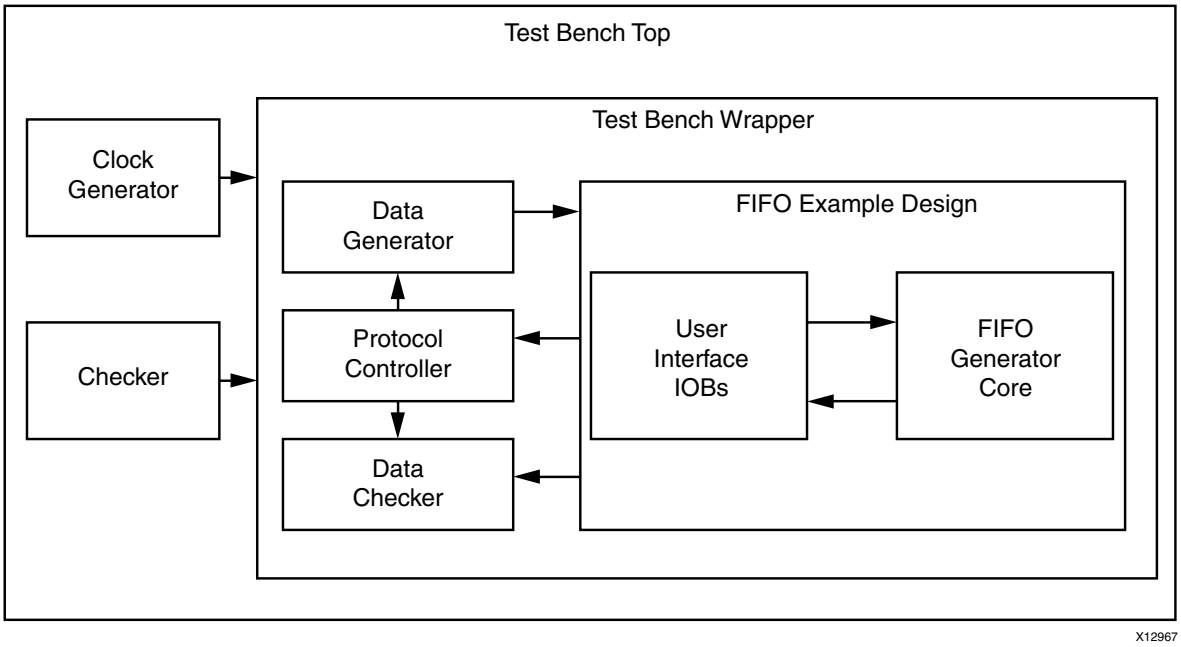


Figure 6-1: **Demonstration Test Bench**

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 AXI Interface

The demonstration test bench in a core with an AXI 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 AXI Interface input signals. Each channel is independently checked for Valid-Ready handshake protocol.
- AXI output signals on read side are combined and cross checked with the pseudo random generator data.
- For AXI memory mapped interface five instances of data generator, data verifier and protocol controller are used.
- For AXI4/AXI3 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
```

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 core 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 core.

Finding Help on Xilinx.com

To help in the design and debug process when using the FIFO Generator, the Xilinx Support web page contains key resources such as product documentation, release notes, answer records, information about known issues, and links for obtaining further product support.

Documentation

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

Download the Xilinx Documentation Navigator from the Downloads page. For more information about this tool and the features available, open the online help after installation.

Answer Records

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 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.

Master Answer Record for the FIFO Generator

AR: 54663

Technical Support

Xilinx provides technical support in the Xilinx Support web page for this LogiCORE™ IP product when used as described in the product documentation. Xilinx cannot guarantee timing, functionality, or support if you do any of the following:

- Implement the solution in devices that are not defined in the documentation.
- Customize the solution beyond that allowed in the product documentation.
- Change 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, navigate to the Xilinx Support web page.

Debug Tools

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

Vivado Design Suite Debug Feature

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

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)

See *Vivado Design Suite User Guide: Programming and Debugging* (UG908) [Ref 10].

Simulation Debug

For details about simulating a design in the Vivado Design Suite, see the *Vivado Logic Simulation User Guide* (UG900) [Ref 9].

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: 7 Series Built-in FIFOs 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 core.
- 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.

Upgrading

This appendix contains information about migrating a design from ISE® to the Vivado® Design Suite, and for upgrading to a more recent version of the IP core. For customers upgrading in the Vivado Design Suite, important details (where applicable) about any port changes and other impact to user logic are included.

Migrating to the Vivado Design Suite

For information about migrating to the Vivado Design Suite, see the *ISE to Vivado Design Suite Migration Guide* (UG911) [Ref 6].

Upgrading in the Vivado Design Suite

This section provides information about any changes to the user logic or port designations that take place when you upgrade to a more current version of this IP core in the Vivado Design Suite.

IMPORTANT: *When upgrading from a 7 series design to a design using UltraScale architecture, the behavior of the core during reset and after reset will not match for about 20 slowest clocks. There are UltraScale architecture specific user parameters (ex: Asymmetric Port Width, Low Latency, Output Register, etc). When upgrading 7 series design with asymmetric port width to UltraScale, the upgrade does not work and user intervention is required.*

Parameter Changes

There were no parameter changes in this version of the core,

Port Changes

There were no port changes in this version of the core

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`).

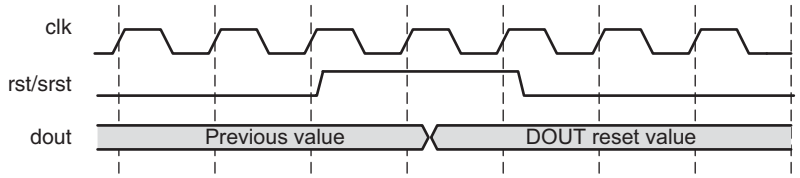


Figure D-1: **dout Reset Value for Synchronous Reset (srst) and for Asynchronous Reset (rst) for Common Clock Block RAM Based FIFO**

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

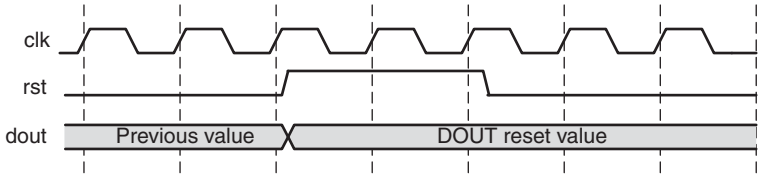


Figure D-2: **dout Reset Value for Asynchronous Reset (rst) for Common Clock Distributed/Shift RAM Based FIFO**

Figure D-3 shows the `dout` reset value for the 7 series device common clock built-in FIFOs with embedded register for asynchronous reset (`rst`).

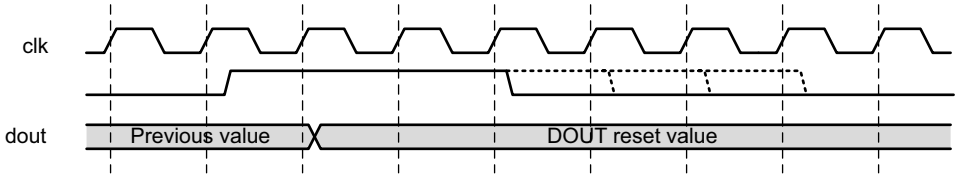


Figure D-3: **dout Reset Value for Common Clock Built-in FIFO**

Figure D-4 shows the `dout` reset value for UltraScale device common clock built-in FIFOs with embedded register for asynchronous reset (`rst`).

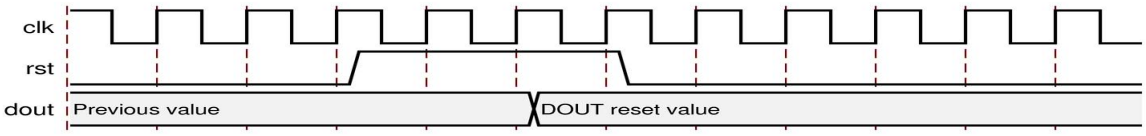


Figure D-4: **dout reset value for UltraScale device common clock built-in FIFO**

Figure D-5 shows the `dout` reset value for independent clock block RAM based FIFOs (`rd_rst`).

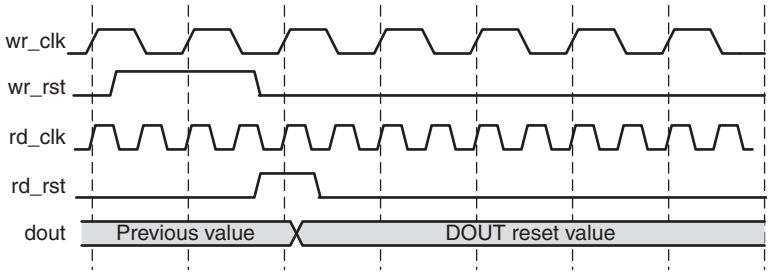


Figure D-5: **dout Reset Value for Independent Clock Block RAM Based FIFO**

Figure D-6 shows the `dout` reset value for independent clock distributed RAM based FIFOs (`rd_rst`)

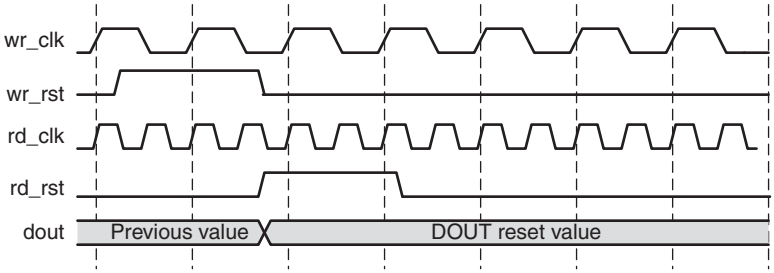


Figure D-6: **dout Reset Value for Independent Clock Distributed RAM Based FIFO**

FIFO Generator Files

FIFO Generator delivers many files while generating the core. Some are used for synthesis and some are used for behavioral simulation.

Target Language = VHDL/Verilog

Project Area:

<project_name>/<project_name>.srcs/sources_1/ip/<component_name>/
fifo_generator_<core_version>/hdl

1. fifo_generator_<core_version>.vhd
   - FIFO Generator's top level RTL file used for synthesis
2. fifo_generator_<core_version>_vhsyn_rfs.vhd
   - The encrypted RTL file used in fifo_generator_<core_version>.vhd
3. fifo_generator_<core_version>_rfs.v
   - Top file is used for behavioral simulation only. This file provides the choice of using behavioral model or encrypted RTL. The behavioral model is delivered at "<project_name>/<project_name>.srcs/sources_1/ip/<component_name>/ fifo_generator_<core_version>/simulation" directory.
4. fifo_generator_<core_version>_rfs.vhd
   - This file is used for behavioral simulation only. This file includes the encrypted RTL for Built-In FIFO configuration

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 F-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 IDE.

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

	SIM Parameter	Type	Description
1	C_COMMON_CLOCK	Integer	• 0: Independent Clock • 1: Common Clock
2	C_DATA_COUNT_WIDTH	Integer	Width of data_count bus (1 – 18)
3	C_DIN_WIDTH	Integer	Width of din bus (1 – 1024) Width must be > 1 for ECC with Double bit error injection
4	C_DOUT_RST_VAL	String	Reset value of dout Hexadecimal value, 0 – F's equal to C_DOUT_WIDTH
5	C_DOUT_WIDTH	Integer	Width of dout bus (1 – 1024) 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) • 1: Synchronize the reset

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

	SIM Parameter	Type	Description
7	C_ERROR_INJECTION_TYPE	Integer	• 0: No error injection • 1: Single bit error injection • 2: Double bit error injection • 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 • 1: Core has almost_empty flag
11	C_HAS_ALMOST_FULL	Integer	• 0: Core does not have ALMOST_FULL flag • 1: Core has ALMOST_ FULL flag
12	C_HAS_DATA_COUNT	Integer	• 0: Core does not have data_count bus • 1: Core has data_count bus
13	C_HAS_OVERFLOW	Integer	• 0: Core does not have OVERFLOW flag • 1: Core has OVERFLOW flag
14	C_HAS_RD_DATA_COUNT	Integer	• 0: Core does not have rd_data_count bus • 1: Core has rd_data_count bus
15	C_HAS_RST	Integer	• 0: Core does not have asynchronous reset (rst) • 1: Core has asynchronous reset (rst)
16	C_HAS_SRST	Integer	• 0: Core does not have synchronous reset (srst) • 1: Core has synchronous reset (srst)
17	C_HAS_UNDERFLOW	Integer	• 0: Core does not have underflow flag • 1: Core has underflow flag
18	C_HAS_VALID	Integer	• 0: Core does not have valid flag • 1: Core has valid flag
19	C_HAS_WR_ACK	Integer	• 0: Core does not have wr_ack flag • 1: Core has wr_ack flag
20	C_HAS_WR_DATA_COUNT	Integer	• 0: Core does not have wr_data_count bus • 1: Core has wr_data_count bus
21	C_IMPLEMENTATION_TYPE	Integer	• 0: Common-Clock Block RAM/Distributed RAM FIFO • 1: Common-Clock Shift RAM FIFO • 2: Independent Clocks Block RAM/ Distributed RAM FIFO • 6: 7 Series and Ultrascale/Ultrascale+ Built-in FIFO • 8: UltraScale Low Latency Built-in FIFO without Output Register • 9: UltraScale Low Latency Built-in FIFO with Output Register

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

	SIM Parameter	Type	Description
22	C_MEMORY_TYPE	Integer	• 1: Block RAM • 2: Distributed RAM • 3: Shift RAM • 4: Built-in FIFO
23	C_MSGON_VAL	Integer	• 0: Disables timing violation on cross clock domain registers • 1: Enables timing violation on cross clock domain registers
24	C_OVERFLOW_LOW(1)	Integer	• 0: OVERFLOW active-High • 1: OVERFLOW active-Low
25	C_PRELOAD_LATENCY	Integer	• 0: First-Word Fall-Through with or without Embedded Register • 1: Standard FIFO without Embedded Register • 2: Standard FIFO with Embedded Register
26	C_PRELOAD_REGS	Integer	• 0: Standard FIFO without Embedded Register • 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 (Ex. 512x36)
28	C_PROG_EMPTY_THRESH_ASSERT_VAL	Integer	prog_empty assert threshold(2)
29	C_PROG_EMPTY_THRESH_NEGATE_VAL	Integer	prog_empty negate threshold(2)
30	C_PROG_EMPTY_TYPE	Integer	• 0: No programmable empty • 1: Single programmable empty thresh constant • 2: Multiple programmable empty thresh constants • 3: Single programmable empty thresh input • 4: Multiple programmable empty thresh inputs
31	C_PROG_FULL_THRESH_ASSERT_VAL	Integer	PROG_FULL assert threshold(2)
32	C_PROG_FULL_THRESH_NEGATE_VAL	Integer	PROG_FULL negate threshold(2)
33	C_PROG_FULL_TYPE	Integer	• 0: No programmable full • 1: Single programmable full thresh constant • 2: Multiple programmable full thresh constants • 3: Single programmable full thresh input • 4: Multiple programmable full thresh inputs
34	C_RD_DATA_COUNT_WIDTH	Integer	Width of rd_data_count bus (1 - 18)
35	C_RD_DEPTH	Integer	Depth of read interface (16 – 131072)

Table F-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(1)	Integer	• 0: underflow active-High • 1: underflow active-Low
39	C_USE_DOUT_RST	Integer	• 0: Does not reset dout on rst • 1: Resets dout on rst
40	C_USE_ECC	Integer	• 0: Does not use ECC feature • 1: Uses Hard ECC • 2: Uses Soft ECC
41	C_USE_EMBEDDED_REG	Integer	• 0: Does not use BRAM/Built-in embedded output register • 1: Uses BRAM/Built-in embedded output register • 2: Uses Interconnect output register • 3: Uses BRAM Embedded and Interconnect output registers
42	C_USE_FWFT_DATA_COUNT	Integer	• 0: Does not use extra logic for FWFT data count • 1: Uses extra logic for FWFT data count • 2: Use general interconnect output register in BRAM FIFO
43	C_VALID_LOW(1)	Integer	• 0: valid active-High • 1: valid active-Low
44	C_WR_ACK_LOW(1)	Integer	• 0: wr_ack active-High • 1: wr_ack active-Low
45	C_WR_DATA_COUNT_WIDTH	Integer	Width of wr_data_count bus (1 – 18)
46	C_WR_DEPTH	Integer	Depth of write interface (16 – 131072)
47	C_WR_FREQ	Integer	Write clock frequency (1 MHz - 1000 MHz)
48	C_WR_PNTR_WIDTH	Integer	log2(C_WR_DEPTH)
49	C_USE_PIPELINE_REG(3)	Integer	• 0: Does not use ECC pipeline register • 1: Uses ECC pipeline register
50	C_POWER_SAVING_MODE(3)	Integer	• 0: Does not use dynamic power gating feature • 1: Uses dynamic power gating feature
51	C_EN_SAFETY_CKT	Integer	• 0: Does not use the additional safety circuit • 1: Use additional safety circuit in asynchronous reset BRAM based FIFOs

Notes:

1. Not available for UltraScale™ architecture-based devices.
2. See the Vivado IDE for the allowable range of values.
3. Available only for UltraScale architecture-based devices with built-in FIFO configurations.

AXI FIFO SIM Parameters

Table F-2 defines the AXI 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 Vivado IP Catalog.

Table F-2: **AXI SIM Parameters**

	SIM Parameter	Type	Description
1	C_INTERFACE_TYPE	Integer	• 0: Native FIFO • 1: AXI4-Stream • 2: AXI Memory Mapped
2	C_AXI_TYPE	Integer	• 1: AXI4 • 2: AXI4-Lite • 3: AXI3
3	C_HAS_AXI_WR_CHANNEL	Integer	• 0: Core does not have Write Channel(1) • 1: Core has Write Channel(1)
4	C_HAS_AXI_RD_CHANNEL	Integer	• 0: Core does not have Read Channel(2) • 1: Core has Read Channel(2)
5	C_HAS_SLAVE_CE(3)	Integer	• 0: Core does not have Slave Interface Clock Enable • 1: Core has Slave Interface Clock Enable
6	C_HAS_MASTER_CE(3)	Integer	• 0: Core does not have Master Interface Clock Enable • 1: Core has Master Interface Clock Enable
7	C_ADD_NGC_CONSTRAINT(3)	Integer	• 0: Core does not add NGC constraint • 1: Core adds NGC constraint
8	C_USE_COMMON_UNDERFLOW(3)	Integer	• 0: Core does not have common underflow flag • 1: Core has common underflow flag
9	C_USE_COMMON_OVERFLOW(3)	Integer	• 0: Core does not have common OVERFLOW flag • 1: Core has common OVERFLOW flag
10	C_USE_DEFAULT_SETTINGS(3)	Integer	• 0: Core does not use default settings • 1: Core uses default settings
11	C_AXI_ID_WIDTH	Integer	ID Width (Range is 1 to 32)
12	C_AXI_ADDR_WIDTH	Integer	Address Width (Range is 1 to 64)
13	C_AXI_DATA_WIDTH	Integer	Data Width (32, 64, 128, 256, 512 or 1024)
14	C_AXI_LEN_WIDTH	Integer	AWLEN/ARLEN Width: • 8: Width is for AXI4 • 4: Width is for AXI3

Table F-2: **AXI SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
15	C_AXI_LOCK_WIDTH	Integer	AWLOCK/ARLOCK Width: • 1: Width is for AXI4 • 2: Width is for AXI3
16	C_HAS_AXI_ID	Integer	• 0: Core does not have AWID/WID/BID/ARID/RID ports • 1: Core has AWID/WID/BID/ARID/RID ports
17	C_HAS_AXI_AWUSER	Integer	• 0: Core does not have AWUSER • 1: Core has AWUSER
18	C_HAS_AXI_WUSER	Integer	• 0: Core does not have WUSER • 1: Core has WUSER
19	C_HAS_AXI_BUSER	Integer	• 0: Core does not have BUSER • 1: Core has BUSER
20	C_HAS_AXI_ARUSER	Integer	• 0: Core does not have ARUSER • 1: Core has ARUSER
21	C_HAS_AXI_RUSER	Integer	• 0: Core does not have RUSER • 1: Core has RUSER
22	C_AXI_AWUSER_WIDTH	Integer	AWUSER Width
23	C_AXI_WUSER_WIDTH	Integer	WUSER Width
24	C_AXI_BUSER_WIDTH	Integer	BUSER Width
25	C_AXI_ARUSER_WIDTH	Integer	ARUSER Width
26	C_AXI_RUSER_WIDTH	Integer	RUSER Width
27	C_HAS_AXIS_TDATA	Integer	• 0: AXI4 Stream does not have TDATA • 1: AXI4 Stream has TDATA
28	C_HAS_AXIS_TID	Integer	• 0: AXI4 Stream does not have TID • 1: AXI4 Stream has TID
29	C_HAS_AXIS_TDEST	Integer	• 0: AXI4 Stream does not have TDEST • 1: AXI4 Stream has TDEST
30	C_HAS_AXIS_TUSER	Integer	• 0: AXI4 Stream does not have TUSER • 1: AXI4 Stream has TUSER
31	C_HAS_AXIS_TREADY	Integer	• 0: AXI4 Stream does not have TREADY • 1: AXI4 Stream has TREADY
32	C_HAS_AXIS_TLAST	Integer	• 0: AXI4 Stream does not have TLAST • 1: AXI4 Stream has TLAST
33	C_HAS_AXIS_TSTRB	Integer	• 0: AXI4 Stream does not have TSTRB • 1: AXI4 Stream has TSTRB
34	C_HAS_AXIS_TKEEP	Integer	• 0: AXI4 Stream does not have TKEEP • 1: AXI4 Stream has TKEEP
35	C_AXIS_TDATA_WIDTH	Integer	AXI4 Stream TDATA Width
36	C_AXIS_TID_WIDTH	Integer	AXI4 Stream TID Width

Table F-2: **AXI SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
37	C_AXIS_TDEST_WIDTH	Integer	AXI4 Stream TDEST Width
38	C_AXIS_TUSER_WIDTH	Integer	AXI4 Stream TUSER Width
39	C_AXIS_TSTRB_WIDTH	Integer	AXI4 Stream TSTRB Width
40	C_AXIS_TKEEP_WIDTH	Integer	AXI4 Stream TKEEP Width
41	C_WACH_TYPE	Integer	Write Address Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
42	C_WDCH_TYPE	Integer	Write Data Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
43	C_WRCH_TYPE	Integer	Write Response Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
44	C_RACH_TYPE	Integer	Read Address Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
45	C_RDCH_TYPE	Integer	Read Data Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
46	C_AXIS_TYPE	Integer	AXI4 Stream type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
47	C_REG_SLICE_MODE_WACH	Integer	Write Address Channel configuration type • 0: Fully Registered • 1: Light Weight
48	C_REG_SLICE_MODE_WDCH	Integer	Write Data Channel configuration type • 0: Fully Registered • 1: Light Weight
49	C_REG_SLICE_MODE_WRCH	Integer	Write Response Channel configuration type • 0: Fully Registered • 1: Light Weight
50	C_REG_SLICE_MODE_RACH	Integer	Read Address Channel configuration type • 0: Fully Registered • 1: Light Weight

Table F-2: **AXI SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
51	C_REG_SLICE_MODE_RDCH	Integer	Read Data Channel configuration type • 0: Fully Registered • 1: Light Weight
52	C_REG_SLICE_MODE_AXIS	Integer	AXI4 Stream configuration type • 0: Fully Registered • 1: Light Weight
53	C_IMPLEMENTATION_TYPE_WACH	Integer	Write Address Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 5: Common Clock Built-in FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO • 13: Independent Clock Built-in FIFO
54	C_IMPLEMENTATION_TYPE_WDCH	Integer	Write Data Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 5: Common Clock Built-in FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO • 13: Independent Clock Built-in FIFO
55	C_IMPLEMENTATION_TYPE_WRCH	Integer	Write Response Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 5: Common Clock Built-in FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO • 13: Independent Clock Built-in FIFO
56	C_IMPLEMENTATION_TYPE_RACH	Integer	Read Address Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 5: Common Clock Built-in FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO • 13: Independent Clock Built-in FIFO

Table F-2: **AXI SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
57	C_IMPLEMENTATION_TYPE_RDCH	Integer	Read Data Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 5: Common Clock Built-in FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO • 13: Independent Clock Built-in FIFO
58	C_IMPLEMENTATION_TYPE_AXIS	Integer	AXI4 Stream Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 5: Common Clock Built-in FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO • 13: Independent Clock Built-in FIFO
59	C_APPLICATION_TYPE_WACH	Integer	Write Address Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
60	C_APPLICATION_TYPE_WDCH	Integer	Write Data Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
61	C_APPLICATION_TYPE_WRCH	Integer	Write Response Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
62	C_APPLICATION_TYPE_RACH	Integer	Read Address Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
63	C_APPLICATION_TYPE_RDCH	Integer	Read Data Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
64	C_APPLICATION_TYPE_AXIS	Integer	AXI4 Stream Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO

Table F-2: **AXI SIM Parameters *(Cont'd)***

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

Table F-2: **AXI SIM Parameters *(Cont'd)***

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

Table F-2: **AXI SIM Parameters *(Cont'd)***

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

Table F-2: **AXI SIM Parameters *(Cont'd)***

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

Table F-2: **AXI SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
108	C_PROG_FULL_TYPE_WDCH	Integer	Write Data Channel • 1 or 3: PROG_FULL is mapped to READY • 5: FULL is mapped to READY • 6: ALMOST_FULL is mapped to READY
109	C_PROG_FULL_TYPE_WRCH	Integer	Write Response Channel • 1 or 3: PROG_FULL is mapped to READY • 5: FULL is mapped to READY • 6: ALMOST_FULL is mapped to READY
110	C_PROG_FULL_TYPE_RACH	Integer	Read Address Channel • 1 or 3: PROG_FULL is mapped to READY • 5: FULL is mapped to READY • 6: ALMOST_FULL is mapped to READY
111	C_PROG_FULL_TYPE_RDCH	Integer	Read Data Channel • 1 or 3: PROG_FULL is mapped to READY • 5: FULL is mapped to READY • 6: ALMOST_FULL is mapped to READY
112	C_PROG_FULL_TYPE_AXIS	Integer	AXI4 Stream • 1 or 3: PROG_FULL is mapped to READY • 5: FULL is mapped to READY • 6: ALMOST_FULL is mapped to READY
113	C_PROG_FULL_THRESH_ASSERT_VAL_WACH	Integer	PROG_FULL assert threshold(4) for Write Address Channel
114	C_PROG_FULL_THRESH_ASSERT_VAL_WDCH	Integer	PROG_FULL assert threshold(4) for Write Data Channel
115	C_PROG_FULL_THRESH_ASSERT_VAL_WRCH	Integer	PROG_FULL assert threshold(4) for Write Response Channel
116	C_PROG_FULL_THRESH_ASSERT_VAL_RACH	Integer	PROG_FULL assert threshold(4) for Read Address Channel
117	C_PROG_FULL_THRESH_ASSERT_VAL_RDCH	Integer	PROG_FULL assert threshold(4) for Read Data Channel
118	C_PROG_FULL_THRESH_ASSERT_VAL_AXIS	Integer	PROG_FULL assert threshold(4) for AXI4 Stream

Table F-2: **AXI SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
119	C_PROG_EMPTY_TYPE_WACH	Integer	Write Address Channel • 1 or 3: prog_empty is mapped to valid • 5: empty is mapped to valid • 6: almost_empty is mapped to valid
120	C_PROG_EMPTY_TYPE_WDCH	Integer	Write Data Channel • 1 or 3: prog_empty is mapped to valid • 5: empty is mapped to valid • 6: almost_empty is mapped to valid
121	C_PROG_EMPTY_TYPE_WRCH	Integer	Write Response Channel • 1 or 3: prog_empty is mapped to valid • 5: empty is mapped to valid • 6: almost_empty is mapped to valid
122	C_PROG_EMPTY_TYPE_RACH	Integer	Read Address Channel • 1 or 3: prog_empty is mapped to valid • 5: empty is mapped to valid • 6: almost_empty is mapped to valid
123	C_PROG_EMPTY_TYPE_RDCH	Integer	Read Data Channel • 1 or 3: prog_empty is mapped to valid • 5: empty is mapped to valid • 6: almost_empty is mapped to valid
124	C_PROG_EMPTY_TYPE_AXIS	Integer	AXI4 Stream • 1 or 3: prog_empty is mapped to valid • 5: empty is mapped to valid • 6: almost_empty is mapped to valid
125	C_PROG_EMPTY_THRESH_ASSERT_VAL_WACH	Integer	prog_empty assert threshold for Write Address Channel(4).
126	C_PROG_EMPTY_THRESH_ASSERT_VAL_WDCH	Integer	prog_empty assert threshold for Write Data Channel.(4)
127	C_PROG_EMPTY_THRESH_ASSERT_VAL_WRCH	Integer	prog_empty assert threshold for Write Response Channel.(4)
128	C_PROG_EMPTY_THRESH_ASSERT_VAL_RACH	Integer	prog_empty assert threshold for Read Address Channel.(4)
129	C_PROG_EMPTY_THRESH_ASSERT_VAL_RDCH	Integer	prog_empty assert threshold for Read Data Channel.(4)
130	C_PROG_EMPTY_THRESH_ASSERT_VAL_AXIS	Integer	prog_empty assert threshold for AXI4 Stream.(4)
131	C_PRIM_FIFO_TYPE_WACH	String	Primitive used to build a FIFO (for example, 512x36)
132	C_PRIM_FIFO_TYPE_WDCH	String	Primitive used to build a FIFO (for example, 512x36)
133	C_PRIM_FIFO_TYPE_WRCH	String	Primitive used to build a FIFO (for example, 512x36)

Table F-2: **AXI SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
134	C_PRIM_FIFO_TYPE_RACH	String	Primitive used to build a FIFO (for example, 512x36)
135	C_PRIM_FIFO_TYPE_RDCH	String	Primitive used to build a FIFO (for example, 512x36)
136	C_PRIM_FIFO_TYPE_AXIS	String	Primitive used to build a FIFO (for example, 512x36)

Notes:

1. Includes Write Address Channel, Write Data Channel and Write Response Channel.
2. Includes Read Address Channel, Read Data Channel.
3. This feature is supported for 7 series device common clock AXI4/AXI3 and AXI4-Stream FIFOs only.
4. See the Vivado IDE for the allowable range of values.

Additional Resources and Legal Notices

Xilinx Resources

For support resources such as Answers, Documentation, Downloads, and Forums, see Xilinx Support.

Documentation Navigator and Design Hubs

Xilinx® Documentation Navigator provides access to Xilinx documents, videos, and support resources, which you can filter and search to find information. To open the Xilinx Documentation Navigator (DocNav):

- From the Vivado® IDE, select **Help > Documentation and Tutorials**.
- On Windows, select **Start > All Programs > Xilinx Design Tools > DocNav**.
- At the Linux command prompt, enter `docnav`.

Xilinx Design Hubs provide links to documentation organized by design tasks and other topics, which you can use to learn key concepts and address frequently asked questions. To access the Design Hubs:

- In the Xilinx Documentation Navigator, click the **Design Hubs View** tab.
- On the Xilinx website, see the Design Hubs page.

Note: For more information on Documentation Navigator, see the Documentation Navigator page on the Xilinx website.

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. *7 Series FPGA Memory Resources* (UG473)
4. *UltraScale Architecture Memory Resources: Advance Specification User Guide* (UG573)
5. *Vivado Design Suite User Guide: Designing IP Subsystems using IP Integrator* (UG994)
6. *ISE to Vivado Design Suite Migration Methodology Guide* (UG911)
7. *Vivado Design Suite User Guide: Designing with IP* (UG896)
8. *Vivado Design Suite User Guide: Getting Started* (UG910)
9. *Vivado Design Suite User Guide: Logic Simulation* (UG900)
10. Vivado Design Suite User Guide: Programming and Debugging (UG908)
11. Answer Record 42571 (AR#42571)

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
10/04/2017	13.2	• Updated for core v13.2. Updated the FIFO generator IP figures to reflect the latest core version. • Updated timing diagrams for reset behavior with and without safety circuit.
04/05/2017	13.1	• Updated the Actual FIFO Depth calculation section for Built-in FIFOs. • Updated the timing diagrams for independent clocks synchronous reset (wr_rst and rd_rst).
10/05/2016	13.1	• Added wr_rst_busy signal assertion for AXI interface. • Updated required constraints in Constraining the Core in Chapter 4.
06/08/2016	13.1	• Added more details on the corruption issue described in AR#42571. • updated a note regarding wr_en/rd_en assertion in Resets section of Chapter 3, Designing with the Core
04/06/2016	13.1	• Updated for core v13.1. Updated the FIFO generator IP figures to reflect the latest core version. • Added an option for BRAM to select both embedded and interconnect registers.

Date	Version	Revision
11/18/2015	13.0	• Added support for UltraScale+ architecture-based devices. • Updated Table 3-21, Table 3-22, Table 3-25, and Table 3-26in Chapter 3, Designing with the Core.
09/30/2015	13.0	• Updated for core v13.0. Updated the FIFO generator IP figures to reflect the latest core version. • Updated the UltraScale support for Built in FIFOs and Block RAM.
06/24/2015	12.0	• Updated the figures to correctly depict the signal and port names.
04/01/2015	12.0	• Updated the UltraScale support for Non-symmetric aspect ratio.
10/01/2014	12.0	• Added a Low latency option for UltraScale devices with Built-In FIFOs. • Added Asymmetric Port Width for UltraScale devices with Common Clock BRAM FIFO. • Added Asynchronous AXI4-Stream Packet FIFO. • Added GUI Option and User Parameter mapping table.
04/02/2014	12.0	Added new ports and parameters for UltraScale architecture. Added ECC Pipeline Register and Dynamic Power Gating support.
12/18/2013	11.0	Added support for UltraScale™ architecture. Added new FIFO primitive parameters.
10/02/2013	11.0	• Document revision number advanced to 11.0 to align with core version number. • All ports/signal names were changed to all lower case. • AXI4-Stream FIFO's depth requirement for packet FIFO was removed.
03/20/2013	4.0	• Updated for core v10.0. • Removed support for ISE Design Suite. • Added Embedded register support for AXI4-Stream packet FIFO. • Updated the parameters and settings in Chapter 4, Customizing and Generating the Native Core.
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.
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.
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*.

Please Read: Important Legal Notices

The information disclosed to you hereunder (the "Materials") is provided solely for the selection and use of Xilinx products. To the maximum extent permitted by applicable law: (1) Materials are made available "AS IS" and with all faults, Xilinx hereby DISCLAIMS ALL WARRANTIES AND CONDITIONS, EXPRESS, IMPLIED, OR STATUTORY, INCLUDING BUT NOT LIMITED TO WARRANTIES OF MERCHANTABILITY, NON-INFRINGEMENT, OR FITNESS FOR ANY PARTICULAR PURPOSE; and (2) Xilinx shall not be liable (whether in contract or tort, including negligence, or under any other theory of liability) for any loss or damage of any kind or nature related to, arising under, or in connection with, the Materials (including your use of the Materials), including for any direct, indirect, special, incidental, or consequential loss or damage (including loss of data, profits, goodwill, or any type of loss or damage suffered as a result of any action brought by a third party) even if such damage or loss was reasonably foreseeable or Xilinx had been advised of the possibility of the same. Xilinx assumes no obligation to correct any errors contained in the Materials or to notify you of updates to the Materials or to product specifications. You may not reproduce, modify, distribute, or publicly display the Materials without prior written consent. Certain products are subject to the terms and conditions of Xilinx's limited warranty, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos; IP cores may be subject to warranty and support terms contained in a license issued to you by Xilinx. Xilinx products are not designed or intended to be fail-safe or for use in any application requiring fail-safe performance; you assume sole risk and liability for use of Xilinx products in such critical applications, please refer to Xilinx's Terms of Sale which can be viewed at https://www.xilinx.com/legal.htm#tos.

AUTOMOTIVE APPLICATIONS DISCLAIMER

AUTOMOTIVE PRODUCTS (IDENTIFIED AS "XA" IN THE PART NUMBER) ARE NOT WARRANTED FOR USE IN THE DEPLOYMENT OF AIRBAGS OR FOR USE IN APPLICATIONS THAT AFFECT CONTROL OF A VEHICLE ("SAFETY APPLICATION") UNLESS THERE IS A SAFETY CONCEPT OR REDUNDANCY FEATURE CONSISTENT WITH THE ISO 26262 AUTOMOTIVE SAFETY STANDARD ("SAFETY DESIGN"). CUSTOMER SHALL, PRIOR TO USING OR DISTRIBUTING ANY SYSTEMS THAT INCORPORATE PRODUCTS, THOROUGHLY TEST SUCH SYSTEMS FOR SAFETY PURPOSES. USE OF PRODUCTS IN A SAFETY APPLICATION WITHOUT A SAFETY DESIGN IS FULLY AT THE RISK OF CUSTOMER, SUBJECT ONLY TO APPLICABLE LAWS AND REGULATIONS GOVERNING LIMITATIONS ON PRODUCT LIABILITY.