LogiCORE IP FIFO Generator v9.3

Product Guide

PG057 December 18, 2012

Table of Contents

SECTION I: SUMMARY

Introduction

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

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

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

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

LogiCORE IP Facts Table	
Core Specifics	
Supported Device Family[1]	Zynq™-7000[2], Artix-7, Virtex®-7, Kintex®-7, Virtex-6, Virtex-5, Virtex-4, Spartan®-6, Spartan-3A/3AN/3A DSP, Spartan-3E, Spartan-3
Supported User Interfaces	Native, AXI4-Stream, AXI4, AXI4-Lite
Resources	See Table 2-1 through Table 2-7.
Provided with Core	
Design Files	NGC
Example Design	VHDL
Test Bench	VHDL
Constraints File	Vivado: XDC ISE: UCF
Simulation Model	Verilog and VHDL Behavioral[3] and Structural
Supported S/W Driver	N/A
Tested Design Flows[4]	
Design Entry	ISE Design Suite v14.4 Vivado Design Suite v2012.4[5]
Simulation	Mentor Graphics ModelSim Cadence Incisive Enterprise Simulator Vivado XSIM ISE ISIM
Synthesis	XST Vivado Synthesis
Support	
Provided by Xilinx @ www.xilinx.com/support	

Notes:

1. For a complete listing of supported devices, see Table 1-2, page 15, Table 1-9, page 29 and the release notes for this core.
2. Supported in ISE Design Suite implementations only.
3. Behavioral models do not model synchronization delay. See Simulating Your Design in Appendix E for details.
4. For the supported versions of the tools, see the Xilinx Design Tools: Release Notes Guide.
5. Vivado Design Suite supports only 7 series devices.

Overview

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

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

This core can be customized using either the Vivado IP customizers in the IP Catalog or the ISE CORE Generator system as a complete solution with control logic already implemented, including management of the read and write pointers and the generation of status flags.

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

Native Interface FIFOs

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

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

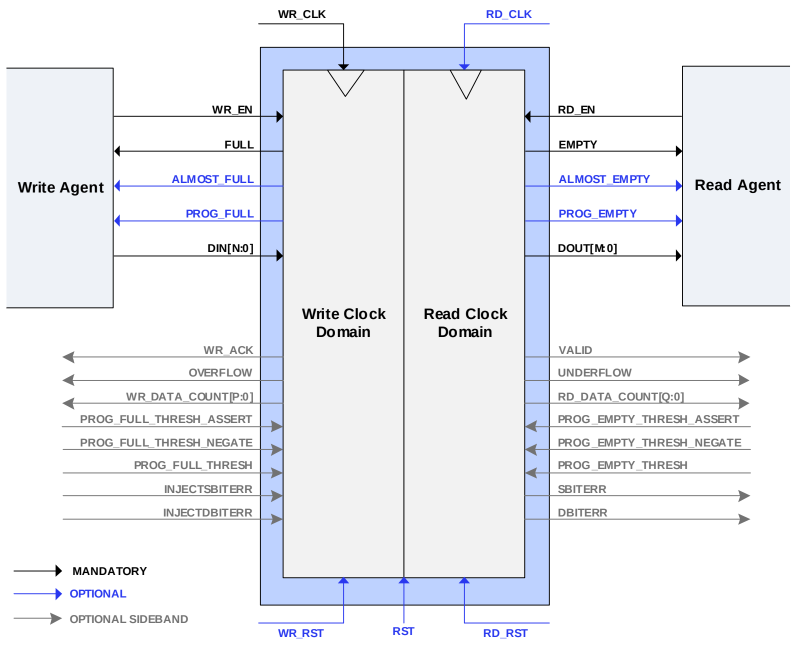


Figure 1-1: **Native Interface FIFOs Signal Diagram**

AXI4 Interface FIFOs

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

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

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

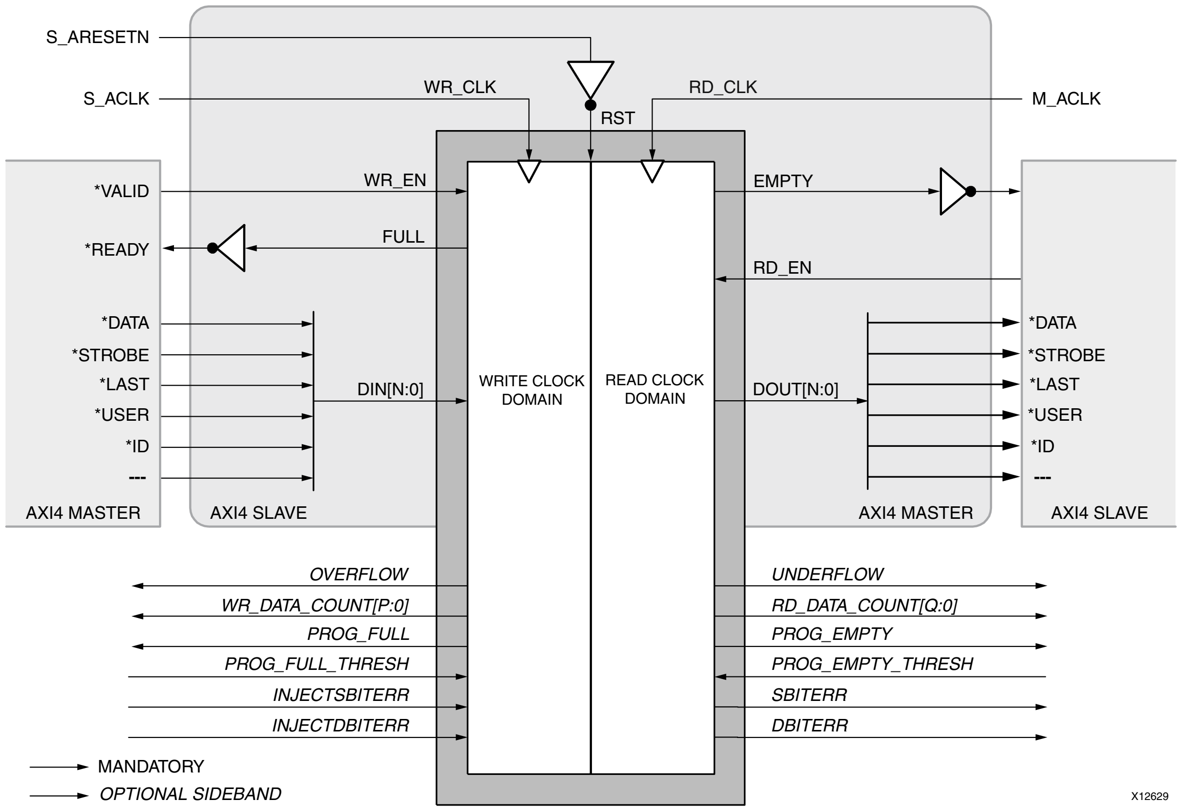


Figure 1-2: **AXI4 FIFO Derivation**

The AXI4 interface protocol uses a two-way VALID and READY handshake mechanism. The information source uses the VALID signal to show when valid data or control information is available on the channel. The information destination uses the READY signal to show when it can accept the data. Figure 1-3 shows an example timing diagram for write and read operations to the AXI4-Stream FIFO, and Figure 1-4 shows an example timing diagram for write and read operations to the AXI4/AXI4-Lite FIFO.

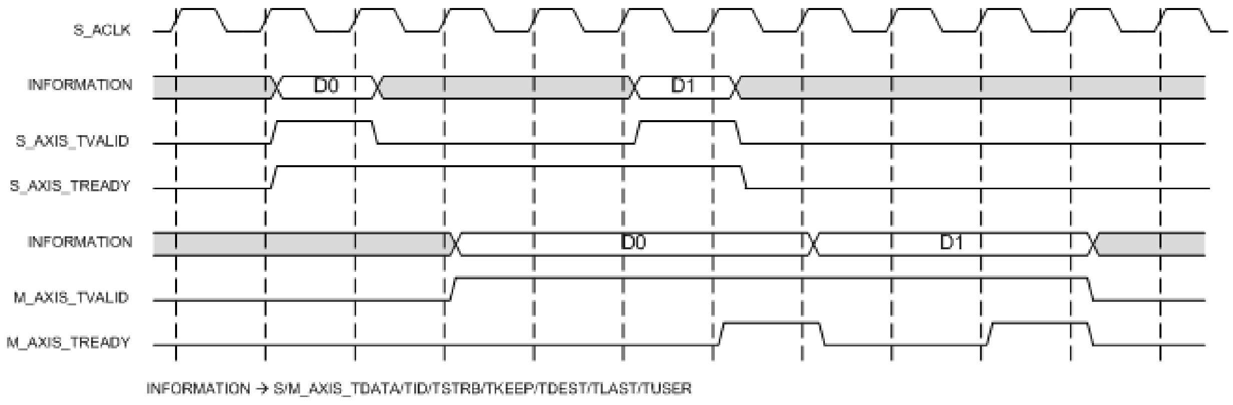


Figure 1-3: **AXI4-Stream FIFO Timing Diagram**

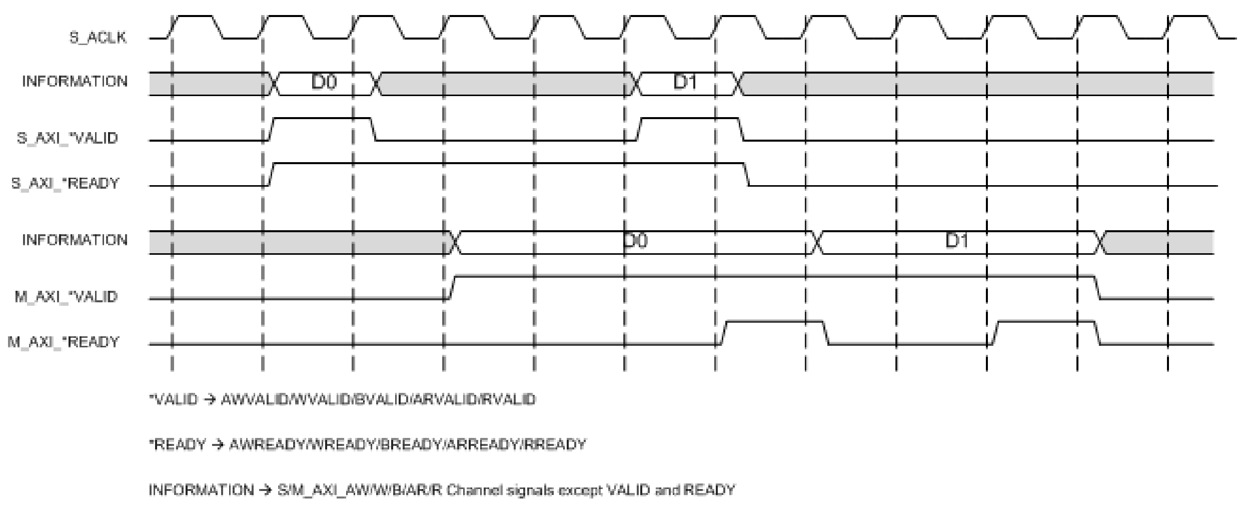

S_ACLK
INFORMATION
D0
D1
S_AXI_*VALID
S_AXI_*READY
INFORMATION
D0
D1
M_AXI_*VALID
M_AXI_*READY
*VALID → AWVALID/WVALID/BVALID/ARVALID/RVALID
*READY → AWREADY/WREADY/BREADY/ARREADY/RREADY
INFORMATION → S/M_AXI_AW/W/B/AR/R Channel signals except VALID and READY

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

In Figure 1-4 and Figure 1-3, the information source generates the VALID signal to indicate when the data is available. The destination generates the READY signal to indicate that it can accept the data, and transfer occurs only when both the VALID and READY signals are high.

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

AXI4 FIFOs operate only in First-Word Fall-Through mode. The First-Word Fall-Through (FWFT) feature provides the ability to look ahead to the next word available from the FIFO without issuing a read operation. When data is available in the FIFO, the first word falls through the FIFO and appears automatically on the output bus.

Feature Summary

Common Features

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

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

Native FIFO Specific Features

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

AXI4 FIFO Features

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

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

Native FIFO Feature Overview

Clock Implementation and Operation

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

Zynq-7000, 7 Series, Virtex-6 and Virtex-5 FPGA Built-in FIFO Support

The FIFO Generator supports the Zynq™-7000, Virtex®-6, Virtex-5, and 7 series (Artix™-7, Virtex-7, and Kintex™-7) FPGA built-in FIFO modules, enabling large FIFOs to be created by cascading the built-in FIFOs in both width and depth. The core expands the capabilities of the built-in FIFOs by utilizing the FPGA fabric to create optional status flags not implemented in the built-in FIFO macro. The built-in Error Correction Checking (ECC) feature in the built-in FIFO macro is also available to the user.

See the appropriate FPGA user guide for frequency requirements.

Virtex-4 FPGA Built-in FIFO Support

Support of the Virtex-4 FPGA built-in FIFO allows generation of a single FIFO primitive complete with fabric implemented flag patch, described in "Solution 1: Synchronous/ Asynchronous Clock Work-Arounds," in the *Virtex-4 FPGA User Guide* [Ref 4]. This patch is implemented in fabric. See Performance in Chapter 2 for resource utilization estimates.

First-Word Fall-Through (FWFT)

The first-word fall-through (FWFT) feature provides the ability to look-ahead to the next word available from the FIFO without issuing a read operation. When data is available in the FIFO, the first word falls through the FIFO and appears automatically on the output bus (DOUT). FWFT is useful in applications that require low-latency access to data and to applications that require throttling based on the contents of the data that are read. FWFT support is included in FIFOs created with block RAM, distributed RAM, or built-in FIFOs in the Zynq-7000, 7 series, Virtex-6 and Virtex-5 devices.

See Table 1-3 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 98).
- Programmable empty (see Non-symmetric Aspect Ratio and First-Word Fall-Through, page 115).
- Data counts (see First-Word Fall-Through Data Count, page 110 and Non-symmetric Aspect Ratio and First-Word Fall-Through, page 115).

Supported Memory Types

The FIFO Generator implements FIFOs built from block RAM, distributed RAM, shift registers, or the Zynq-7000, 7 series, Virtex-6 and Virtex-5 FPGA 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. The generator also creates single primitive Virtex-4 FPGA built-in FIFOs with the fabric implemented flag patch described in "Solution 1: Synchronous/Asynchronous Clock Work-Arounds," in the *Virtex-4 FPGA User Guide* [Ref 4].

Table 1-1: **Memory Configuration Benefits**

	Independent Clocks	Common Clock	Small Buffering	Medium-Large Buffering	High Performance	Minimal Resources
Zynq-7000, 7 Series, Virtex-6, and Virtex-5 FPGA with Built-in FIFO	✓	✓		✓	✓	✓
Block RAM	✓	✓		✓	✓	✓
Shift Register		✓	✓		✓	
Distributed RAM	✓	✓	✓		✓	

Non-Symmetric Aspect Ratio Support

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

Embedded Registers in block RAM and FIFO Macros

In Zynq-7000, 7 series, Virtex-6, Virtex-5 and Virtex-4 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 available in Zynq-7000, 7 series, and Virtex-6 FPGAs can be reset

(DOUT) to a default or user programmed value for common clock built-in FIFOs. See Embedded Registers in Block RAM and FIFO Macros (Zynq-7000, 7 Series, Virtex-6, Virtex-5 and Virtex-4 FPGAs), page 116 for more information.

Error Injection and Correction (ECC) Support

The block RAM and FIFO macros are equipped with built-in Error Correction Checking (ECC) in the Virtex-5 FPGA architecture and built-in Error Injection and Correction Checking in the Zynq-7000, 7 series, and Virtex-6 FPGA architectures. This feature is available for both the common and independent clock block RAM or built-in FIFOs.

Native FIFO Supported Devices

Table 1-2 shows the families and sub-families supported by the Native FIFO Generator. For more details about device support, see the Release Notes.

Table 1-2: **Supported FPGA Families and Sub-Families**

FPGA Family	Sub-Family
Zynq-7000	
Virtex-7	
Virtex-7 -2L	
Virtex-7 -2G	
Virtex-7	XT
Kintex-7	
Kintex-7-2L	
Artix-7	
Virtex-6 XC	CXT/LXT/SXT/HXT
Virtex-6 XQ	LXT/SXT
Virtex-6 -1L XC	LXT/SXT
Virtex-6 -1L XQ	LXT/SXT
Spartan-6 XC	LX/LXT
Spartan-6 XA	LX/LXT
Spartan-6 XQ	LX/LXT
Spartan-6 -1L	XC LX
Spartan-6 -1L	XQ LX
Virtex-5 XC	LX/LXT/SXT/TXT/FXT
Virtex-5 XQ	LX/LXT/SXT/FXT
Virtex-4 XC	LX/SX/FX
Virtex-4 XQ	LX/SX/FX
Virtex-4 XQR	LX/SX/FX

Table 1-2: **Supported FPGA Families and Sub-Families *(Cont'd)***

FPGA Family	Sub-Family
Spartan-3 XC	
Spartan-3 XA	
Spartan-3A XC	3A / 3A DSP / 3AN
Spartan-3A XA	3A / 3A DSP
Spartan-3E XC	
Spartan-3E XA	

Native FIFO Configuration and Implementation

Table 1-3 defines the supported memory and clock configurations.

Table 1-3: **FIFO Configurations**

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

a. Embedded register support is only available for Zynq-7000, 7 series, Virtex-6, Virtex-5 and Virtex-4 FPGA block RAM-based FIFOs, as well as Zynq-7000, 7 series, Virtex-6 and Virtex-5 FPGA common clock built-in FIFOs.

b. The built-in FIFO primitive is only available in the Virtex-6, Virtex-5 and Virtex-4 architectures.

c. FWFT is supported for Built-in FIFOs in Zynq-7000, 7 series, Virtex-6 and Virtex-5 devices only.

d. For non-symmetric aspect ratios, use the block RAM implementation (feature not supported in built-in FIFO primitive).

Common Clock: Block RAM, Distributed RAM, Shift Register

This implementation category allows you to select block RAM, distributed RAM, or shift register and supports a common clock for write and read data accesses. The feature set supported for this configuration includes status flags (full, almost full, empty, and almost empty) and programmable empty and full flags generated with user-defined thresholds.

In addition, optional handshaking and error flags are supported (write acknowledge, overflow, valid, and underflow), and an optional data count provides the number of words in the FIFO. In addition, for the block RAM and distributed RAM implementations, you have the option to select a synchronous or asynchronous reset for the core. For Zynq-7000, 7 series, Virtex-6 and Virtex-5 FPGA designs, the block RAM FIFO configuration also supports ECC.

Common Clock: Zynq-7000, 7 Series, Virtex-6, Virtex-5 or Virtex-4 FPGA Built-in FIFO

This implementation category allows you to select the built-in FIFO available in the Zynq-7000, 7 series, Virtex-6, Virtex-5 or Virtex-4 FPGA architecture and supports a common clock for write and read data accesses. The feature set supported for this configuration includes status flags (full and empty) and optional programmable full and empty flags with user-defined thresholds.

In addition, optional handshaking and error flags are available (write acknowledge, overflow, valid, and underflow). The Zynq-7000, 7 series, Virtex-6 and Virtex-5 FPGA built-in FIFO configuration also supports the built-in ECC feature.

Independent Clocks: Block RAM and Distributed RAM

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

The feature set supported for this type of FIFO includes non-symmetric aspect ratios (different write and read port widths), status flags (full, almost full, empty, and almost empty), as well as programmable full and empty flags generated with user-defined thresholds. Optional read data count and write data count indicators provide the number of words in the FIFO relative to their respective clock domains. In addition, optional handshaking and error flags are available (write acknowledge, overflow, valid, and underflow). For Zynq-7000, 7 series, Virtex-6 and Virtex-5 FPGA designs, the block RAM FIFO configuration also supports ECC.

Independent Clocks: Zynq-7000, 7 Series, Virtex-6, Virtex-5 or Virtex-4 FPGA Built-in FIFO

This implementation category allows you to select the built-in FIFO available in the Zynq-7000, 7 series, Virtex-6, Virtex-5 or Virtex-4 FPGA architecture. Operations in the read domain are synchronous to the read clock and operations in the write domain are synchronous to the write clock.

The feature set supported for this configuration includes status flags (full and empty) and programmable full and empty flags generated with user-defined thresholds. In addition, optional handshaking and error flags are available (write acknowledge, overflow, valid, and underflow). The Zynq-7000, 7 series, Virtex-6 and Virtex-5 FPGA built-in FIFO configuration also supports the built-in ECC feature.

Native FIFO Generator Feature Summary

Table 1-4 summarizes the supported FIFO Generator features for each clock configuration and memory type.

Table 1-4: **FIFO Configurations Summary**

FIFO Feature	Independent Clocks			Common Clock		
	Block RAM	Distributed RAM	Built-in FIFO	Block RAM	Distributed RAM, Shift Register	Built-in FIFO
Non-symmetric Aspect Ratios[a]	✓					
Symmetric Aspect Ratios	✓	✓	✓	✓	✓	✓
Almost Full	✓	✓		✓	✓	
Almost Empty	✓	✓		✓	✓	
Handshaking	✓	✓	✓	✓	✓	✓
Data Count	✓	✓		✓	✓	
Programmable Empty/Full Thresholds	✓	✓	✓[b]	✓	✓	✓(b)
First-Word Fall-Through(c)	✓	✓	✓	✓	✓	✓
Synchronous Reset				✓	✓	
Asynchronous Reset	✓[d]	✓(d)	✓	✓(d)	✓(d)	✓
DOUT Reset Value	✓	✓		✓	✓	✓(e)
ECC	✓(f)		✓[f]	✓(f)		✓(f)
Embedded Register	✓(g)			✓(g)		✓(g)

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

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

c. First-Word-Fall-Through is not supported for the shift RAM FIFOs and Virtex-4 built-in FIFOs.

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

e. DOUT Reset Value is supported only in Zynq-7000, 7 series, and Virtex-6 FPGA common clock built-in FIFOs.

f. ECC is only supported for the Zynq-7000, 7 series, Virtex-6 and Virtex-5 FPGAs and block RAM and built-in FIFOs.

g. Embedded register option is only supported in Zynq-7000, 7 series, Virtex-6, Virtex-5 and Virtex-4 FPGA block RAM FIFOs, as well as Zynq-7000, 7 series, Virtex-6 and Virtex-5 FPGA common clock built-in FIFOs. See <BL Blue>Embedded Registers in block RAM and FIFO Macros.

Using Block RAM FIFOs Versus Built-in FIFOs

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

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

Native FIFO Interface Signals

The following sections define the FIFO interface signals. Figure 1-5 illustrates these signals (both standard and optional ports) for a FIFO core that supports independent write and read clocks.

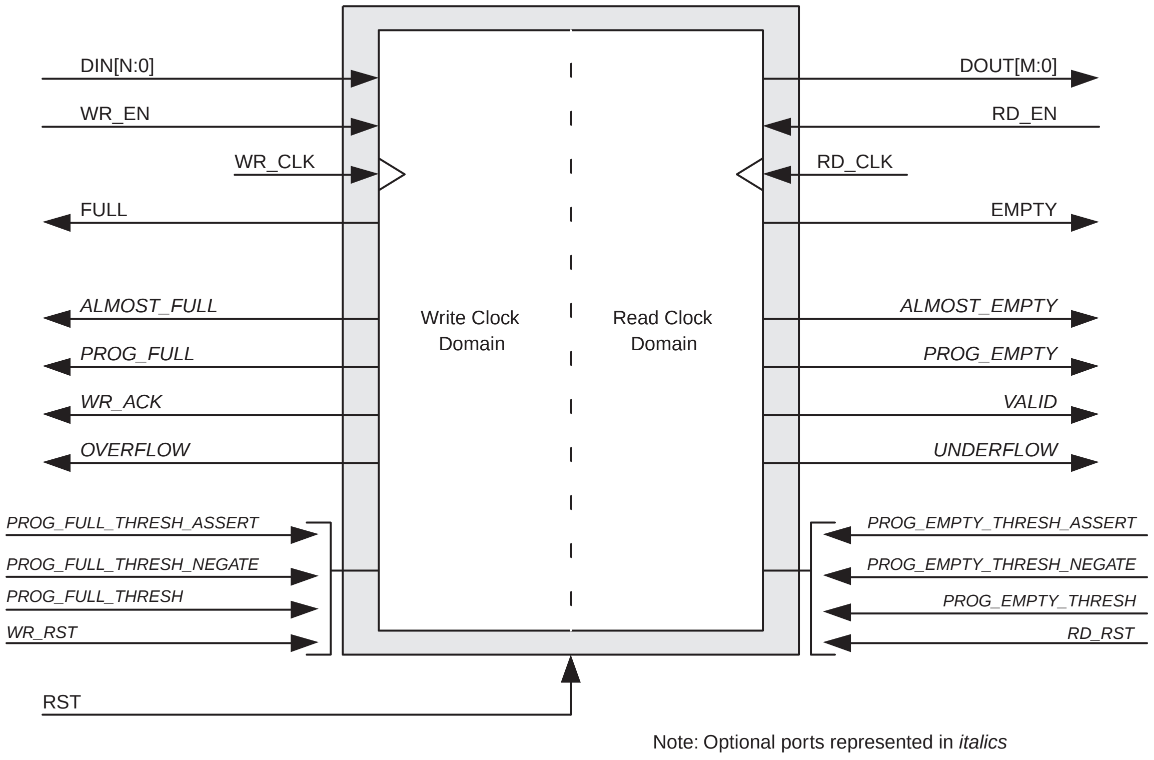


Figure 1-5: **FIFO with Independent Clocks: Interface Signals**

Interface Signals: FIFOs With Independent Clocks

The RST signal, as defined Table 1-5, 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 1-5: **Reset Signal for FIFOs with Independent Clocks**

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

Table 1-6 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 1-6: **Write Interface Signals for FIFOs with Independent Clocks**

Name	Direction	Description
	Required	
WR_CLK	Input	Write Clock: All signals on the write domain are synchronous to this clock.
DIN[N:0]	Input	Data Input: The input data bus used when writing the FIFO.
WR_EN	Input	Write Enable: If the FIFO is not full, asserting this signal causes data (on DIN) to be written to the FIFO.
FULL	Output	Full Flag: When asserted, this signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is not destructive to the contents of the FIFO.
	Optional	
WR_RST	Input	Write Reset: Synchronous to write clock. When asserted, initializes all internal pointers and flags of write clock domain.
ALMOST_FULL	Output	Almost Full: When asserted, this signal indicates that only one more write can be performed before the FIFO is full.
PROG_FULL	Output	Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the assert threshold. It is deasserted when the number of words in the FIFO is less than the negate threshold.
WR_DATA_COUNT [D:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never under-report the number of words in the FIFO, to ensure you never overflow the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of WR_CLK, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. If D is less than log2(FIFO depth)-1, the bus is truncated by removing the least-significant bits.

Table 1-6: **Write Interface Signals for FIFOs with Independent Clocks *(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 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 FIFO Generator GUI for the valid range of values[a].
PROG_FULL_THRESH_NEGATE	Input	Programmable Full Threshold Negate: This signal is used to set the lower threshold value for the programmable full flag, which defines when the signal is de-asserted. The threshold can be dynamically set in-circuit during reset. Refer to FIFO Generator GUI for the valid range of values[a].
INJECTSBITERR	Input	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, and Virtex-6 FPGA block RAMs or built-in FIFO macros.
INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, and Virtex-6 FPGA block RAMs or built-in FIFO macros.

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

Table 1-7 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 1-7: **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.

Table 1-7: **Read Interface Signals for FIFOs with Independent Clocks *(Cont'd)***

Name	Direction	Description
RD_EN	Input	Read Enable: If the FIFO is not empty, asserting this signal causes data to be read from the FIFO (output on DOUT).
EMPTY	Output	Empty Flag: When asserted, this signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is not destructive to the FIFO.
Optional		
ALMOST_EMPTY	Output	Almost Empty Flag: When asserted, this signal indicates that the FIFO is almost empty and one word remains in the FIFO.
PROG_EMPTY	Output	Programmable Empty: This signal is asserted when the number of words in the FIFO is less than or equal to the programmable threshold. It is de-asserted when the number of words in the FIFO exceeds the programmable threshold.
RD_DATA_COUNT [C:0]	Output	Read Data Count: This bus indicates the number of words available for reading in the FIFO. The count is guaranteed to never over-report the number of words available for reading, to ensure that you do not underflow the FIFO. The exception to this behavior is when the read operation occurs at the rising edge of RD_CLK, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. If C is less than log2(FIFO depth)-1, the bus is truncated by removing the least-significant bits.
VALID	Output	Valid: This signal indicates that valid data is available on the output bus (DOUT).
UNDERFLOW	Output	Underflow: Indicates that the read request (RD_EN) during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.
PROG_EMPTY_THRESH	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. You can either choose to set the assert and negate threshold to the same value (using PROG_EMPTY_THRESH), or you can control these values independently (using PROG_EMPTY_THRESH_ASSERT and PROG_EMPTY_THRESH_NEGATE).
PROG_EMPTY_THRESH_ASSERT	Input	Programmable Empty Threshold Assert: This signal is used to set the lower threshold value for the programmable empty flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset. Refer to the FIFO Generator GUI for the valid range of values[a].
PROG_EMPTY_THRESH_NEGATE	Input	Programmable Empty Threshold Negate: This signal is used to set the upper threshold value for the programmable empty flag, which defines when the signal is de-asserted. The threshold can be dynamically set in-circuit during reset. Refer to the FIFO Generator GUI for the valid range of values[a].

Table 1-7: **Read Interface Signals for FIFOs with Independent Clocks *(Cont'd)***

Name	Direction	Description
SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, Virtex-6 or Virtex-5 FPGA block RAM or built-in FIFO macro.
DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, Virtex-6 or Virtex-5 FPGA block RAM or built-in FIFO macro and data in the FIFO core is corrupted.

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

Interface Signals: FIFOs with Common Clock

Table 1-8 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 1-8: **Interface Signals for FIFOs with a Common Clock**

Name	Direction	Description
Required		
RST	Input	Reset: An asynchronous reset that initializes all internal pointers and output registers.
SRST	Input	Synchronous Reset: A synchronous reset that initializes all internal pointers and output registers.
CLK	Input	Clock: All signals on the write and read domains are synchronous to this clock.
DIN[N:0]	Input	Data Input: The input data bus used when writing the FIFO.
WR_EN	Input	Write Enable: If the FIFO is not full, asserting this signal causes data (on DIN) to be written to the FIFO.
FULL	Output	Full Flag: When asserted, this signal indicates that the FIFO is full. Write requests are ignored when the FIFO is full, initiating a write when the FIFO is full is not destructive to the contents of the FIFO.
DOUT[M:0]	Output	Data Output: The output data bus driven when reading the FIFO.
RD_EN	Input	Read Enable: If the FIFO is not empty, asserting this signal causes data to be read from the FIFO (output on DOUT).
EMPTY	Output	Empty Flag: When asserted, this signal indicates that the FIFO is empty. Read requests are ignored when the FIFO is empty, initiating a read while empty is not destructive to the FIFO.
Optional		
DATA_COUNT [C:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. If C is less than log2(FIFO depth)-1, the bus is truncated by removing the least-significant bits.

Table 1-8: **Interface Signals for FIFOs with a Common Clock *(Cont'd)***

Name	Direction	Description
ALMOST_FULL	Output	Almost Full: When asserted, this signal indicates that only one more write can be performed before the FIFO is full.
PROG_FULL	Output	Programmable Full: This signal is asserted when the number of words in the FIFO is greater than or equal to the assert threshold. It is deasserted when the number of words in the FIFO is less than the negate threshold.
WR_ACK	Output	Write Acknowledge: This signal indicates that a write request (WR_EN) during the prior clock cycle succeeded.
OVERFLOW	Output	Overflow: This signal indicates that a write request (WR_EN) during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO.
PROG_FULL_THRESH	Input	Programmable Full Threshold: This signal is used to set the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. You can either choose to set the assert and negate threshold to the same value (using PROG_FULL_THRESH), or you can control these values independently (using PROG_FULL_THRESH_ASSERT and PROG_FULL_THRESH_NEGATE).
PROG_FULL_THRESH_ASSERT	Input	Programmable Full Threshold Assert: This signal is used to set the upper threshold value for the programmable full flag, which defines when the signal is asserted. The threshold can be dynamically set in-circuit during reset. Refer to the FIFO Generator GUI for the valid range of values[a].
PROG_FULL_THRESH_NEGATE	Input	Programmable Full Threshold Negate: This signal is used to set the lower threshold value for the programmable full flag, which defines when the signal is de-asserted. The threshold can be dynamically set in-circuit during reset. Refer to the FIFO Generator GUI for the valid range of values[a].
ALMOST_EMPTY	Output	Almost Empty Flag: When asserted, this signal indicates that the FIFO is almost empty and one word remains in the FIFO.
PROG_EMPTY	Output	Programmable Empty: This signal is asserted after the number of words in the FIFO is less than or equal to the programmable threshold. It is de-asserted when the number of words in the FIFO exceeds the programmable threshold.
VALID	Output	Valid: This signal indicates that valid data is available on the output bus (DOUT).
UNDERFLOW	Output	Underflow: Indicates that read request (RD_EN) during the previous clock cycle was rejected because the FIFO is empty. Underflowing the FIFO is not destructive to the FIFO.

Table 1-8: **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 on a Kintex-7, Virtex-7, Virtex-6 or Virtex-5 FPGA block RAM or built-in FIFO macro.
DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, Virtex-6 or Virtex-5 FPGA block RAM or built-in FIFO macro and data in the FIFO core is corrupted.
INJECTSBITERR	Input	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM or built-in FIFO macro. For detailed information, see Chapter 3, "Designing with the Core."
INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM or built-in FIFO macro. For detailed information, see Chapter 3, "Designing with the Core."

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

AXI4 FIFO Feature Overview

Easy Integration of Independent FIFOs for Read and Write Channels

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

For AXI4 and AXI4-Lite interfaces, the FIFO Generator provides the ability to implement independent FIFOs for each channel, as shown in Figure 1-6. For each channel, the core can be independently configured to generate a block RAM or distributed memory-based FIFO. The depth of each FIFO can also be independently configured.

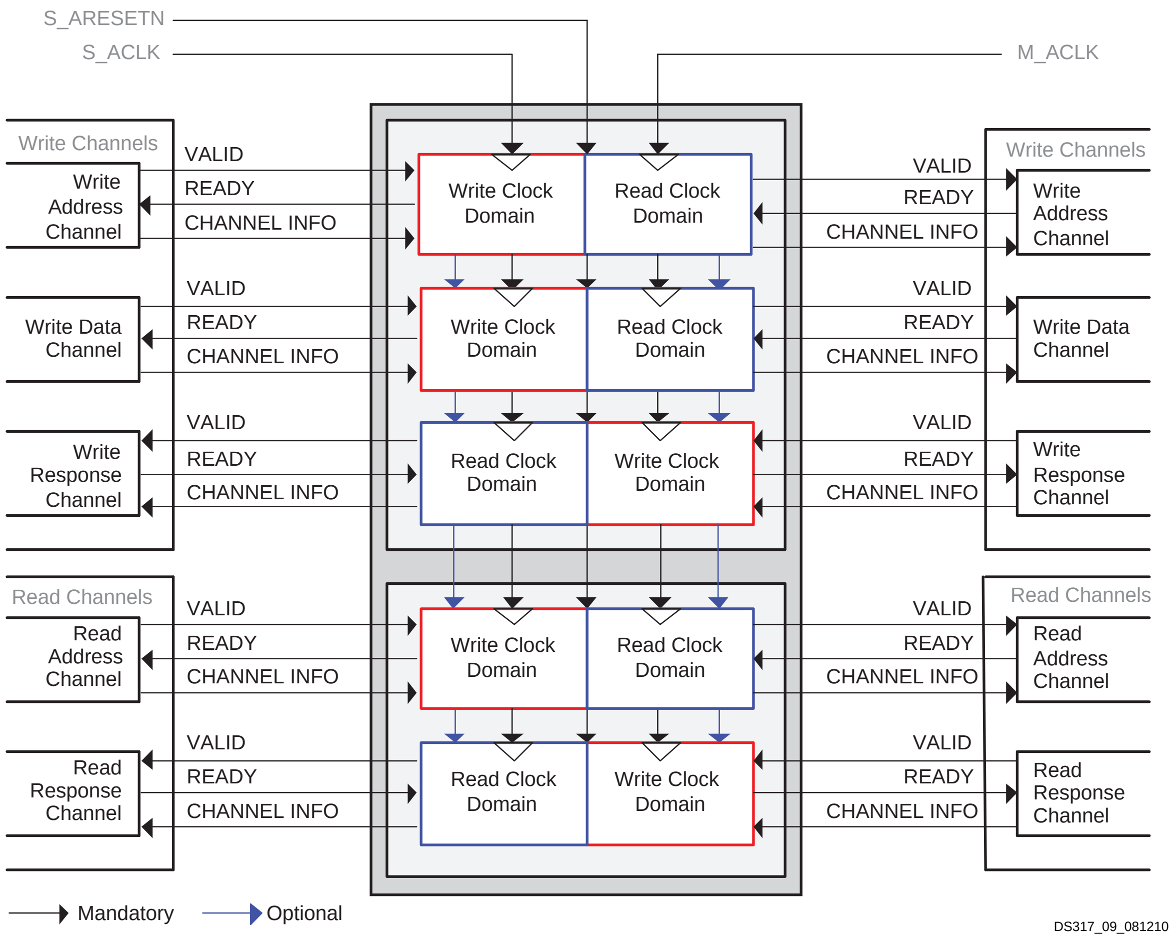


Figure 1-6: **AXI4 Block Diagram**

Clock and Reset Implementation and Operation

For the AXI4-Stream, AXI4 and AXI4-Lite interfaces, all instantiated FIFOs share clock and asynchronous active low reset signals (as shown Figure 1-6). In addition, all instantiated FIFOs can support either independent clock or common clock operation.

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

Automatic FIFO Width Calculation

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

Supported Configuration, Memory and Application Types

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

Packet FIFO

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

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

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

AXI4-Stream Packet FIFO

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

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

AXI4 Packet FIFO

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

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

Error Injection and Correction (ECC) Support

The block RAM macros are equipped with built-in Error Injection and Correction Checking in the Zynq-7000, 7 series, and Virtex-6 FPGA architectures. This feature is available for both the common and independent clock block RAM FIFOs.

For more details on Error Injection and Correction, see Built-in Error Correction Checking in Chapter 3.

AXI4 Slave Interface for Performing Writes

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

AXI4 Master Interface for Performing Reads

The AXI4 FIFO provides an AXI4 Master interface for performing Reads. In Figure 1-4, the AXI4 FIFO provides INFORMATION and VALID signals; upon detecting a READY signal asserted from the AXI4 Slave interface, the AXI4 FIFO will place the next INFORMATION on the bus. The VALID signal will be de-asserted only when the FIFO is empty.

AXI4 FIFO Supported Devices

Table 1-9 shows the families and sub-families supported by the FIFO Generator. For more details about device support, see the Release Notes.

Table 1-9: **Supported FPGA Families and Sub-Families**

FPGA Family	Sub-Family
Zynq-7000[a]	
Virtex-7	
Virtex-7 -2L	
Virtex-7 -2G	
Virtex-7 XT	
Kintex-7	
Kintex-7 -2L	
Artix-7	
Virtex-6 XC	CXT/LXT/SXT/HXT
Virtex-6 XQ	LXT/SXT
Virtex-6 -1L XC	LXT/SXT
Virtex-6 -1L XQ	LXT/SXT
Spartan-6 XQ	LX/LXT
Spartan-6 -1L XC	LX
Spartan-6 -1L XQ	LX

a. ISE Design Suite implementations only.

AXI4 FIFO Feature Summary

Table 1-10 summarizes the supported FIFO Generator features for each clock configuration and memory type.

Table 1-10: **AXI4 FIFO Configuration Summary**

FIFO Options	Common Clock		Independent Clock	
	Block RAM	Distributed Memory	Block RAM	Distributed Memory
Full[(a)]	✓	✓	✓	✓
Programable Full[b]	✓	✓	✓	✓
Empty[(c)]	✓	✓	✓	✓
Programmable Empty[(b)]	✓	✓	✓	✓
Data Counts	✓	✓	✓	✓

Table 1-10: **AXI4 FIFO Configuration Summary**

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

a. Mapped to S_AXIS_TREADY/S_AXI_AWREADY/S_AXI_WREADY/M_AXI_BREADY/S_AXI_ARREADY/M_AXI_RREADY depending on the Handshake Flag Options in the GUI.
b. Provided as sideband signal depending on the GUI option.
c. Mapped to M_AXIS_TVALID/M_AXI_AWVALID/M_AXI_WVALID/S_AXI_BVALID/M_AXI_ARVALID/S_AXI_RVALID depending on the Handshake Flag Options in the GUI.

AXI4 FIFO Interface Signals

The following sections define the AXI4 FIFO interface signals.

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

Global Signals

Table 1-11 defines the global interface signals for AXI4 FIFO.

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

Table 1-11: **AXI4 FIFO - Global Interface Signals**

Name	Direction	Description
Global Clock and Reset Signals Mapped to FIFO Clock and Reset Inputs		
M_ACLK	Input	Global Master Interface Clock: All signals on Master Interface of AXI4 FIFO are synchronous to M_ACLK
S_ACLK	Input	Global Slave Interface Clock: All signals are sampled on the rising edge of this clock.
S_ARESETN	Input	Global reset: This signal is active low.
Clock Enable Signals Gated with FIFO's WR_EN and RD_EN Inputs		
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 1-12 defines the AXI4-Stream FIFO interface signals.

Table 1-12: **AXI4-Stream FIFO Interface Signals**

Name	Direction	Description
AXI4-Stream Interface: Handshake Signals for FIFO Write Interface		
S_AXIS_TVALID	Input	TVALID: Indicates that the master is driving a valid transfer. A transfer takes place when both TVALID and TREADY are asserted.
S_AXIS_TREADY	Output	TREADY: Indicates that the slave can accept a transfer in the current cycle.
AXI4-Stream Interface: Information Signals Mapped to FIFO Data Input (DIN) Bus		
S_AXIS_TDATA[m-1:0]	Input	TDATA: The primary payload that is used to provide the data that is passing across the interface. The width of the data payload is an integer number of bytes.
S_AXIS_TSTRB[m/8-1:0]	Input	TSTRB: The byte qualifier that indicates whether the content of the associated byte of TDATA is processed as a data byte or a position byte. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • 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		

Table 1-12: **AXI4-Stream FIFO Interface Signals *(Cont'd)***

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

Table 1-12: **AXI4-Stream FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXIS_DBITERR	Output	Double-Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
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.
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 FIFO Interface Signals

Write Channels

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

Table 1-13: **AXI4 Write Address Channel FIFO Interface Signals**

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

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

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

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

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

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

Name	Direction	Description
AXI_AW_RD_DATA_COUNT[*D*:0]	Output	Read Data Count: This bus indicates the number of words available for reading in the FIFO. The count is guaranteed to never over-report the number of words available for reading, to ensure that you do not underflow the FIFO. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
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 1-14 defines the AXI4 FIFO interface signals for Write Data Channel.

Table 1-14: **AXI4 Write Data Channel FIFO Interface Signals**

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

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

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

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

Name	Direction	Description
AXI_W_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_W_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_W_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_W_SBITERR	Output	Single-Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_W_DBITERR	Output	Double-Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_W_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO. **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. $D = \log_2(\text{FIFO depth})+1$
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. $D = \log_2(\text{FIFO depth})+1$

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

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

Table 1-15 defines the AXI4 FIFO interface signals for Write Response Channel.

Table 1-15: **AXI4 Write Response Channel FIFO Interface Signals**

Name	Direction	Description
AXI4 Interface Write Response Channel: Information Signals Mapped to FIFO Data Output (DOUT) Bus		
S_AXI_BID[m:0]	Output	Response ID: The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.
S_AXI_BRESP[1:0]	Output	Write Response: Indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
S_AXI_BUSER[m:0]	Output	Write Response Channel User
AXI4 Interface Write Response Channel: Handshake Signals for FIFO Read Interface		
S_AXI_BVALID	Output	Write Response Valid: Indicates that a valid write response is available: • 1 = Write response available. • 0 = Write response not available.
S_AXI_BREADY	Input	Response Ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.
AXI4 Interface Write Response Channel: Information Signals Derived from FIFO Data Input (DIN) Bus		
M_AXI_BID[m:0]	Input	Response ID: The identification tag of the write response. The BID value must match the AWID value of the write transaction to which the slave is responding.
M_AXI_BRESP[1:0]	Input	Write Response: Indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
M_AXI_BUSER[m:0]	Input	Write Response Channel User
AXI4 Interface Write Response Channel: Handshake Signals for FIFO Write Interface		

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

Name	Direction	Description
M_AXI_BVALID	Input	Write Response Valid: Indicates that a valid write response is available: • 1 = Write response available. • 0 = Write response not available.
M_AXI_BREADY	Output	Response Ready: Indicates that the master can accept the response information. • 1 = Master ready. • 0 = Master not ready.
AXI4 Write Response Channel FIFO: Optional Sideband Signals		
AXI_B_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_B_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_B_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_B_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_B_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_B_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_B_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO. **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. $D = \log_2(\text{FIFO depth})+1$

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

Name	Direction	Description
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 1-16 defines the AXI4 FIFO interface signals for Read Address Channel.

Table 1-16: **AXI4 Read Address Channel FIFO Interface Signals**

Name	Direction	Description
AXI4 Interface Read Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus		
S_AXI_ARID[m:0]	Input	Read Address ID: This signal is the identification tag for the read address group of signals.
S_AXI_ARADDR[m:0]	Input	Read Address: The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
S_AXI_ARLEN[7:0]	Input	Burst Length: The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
S_AXI_ARSIZE[2:0]	Input	Burst Size: This signal indicates the size of each transfer in the burst.

Table 1-16: **AXI4 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[2:0]	Input	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.
S_AXI_ARCACHE[4:0]	Input	Cache Type: This signal provides additional information about the cacheable characteristics of the transfer.
S_AXI_ARPROT[3:0]	Input	Protection Type: This signal provides protection unit information for the transaction.
S_AXI_ARQOS[3:0]	Input	Quality of Service (QoS): Sent on the read address channel for each read transaction.
S_AXI_ARREGION[3:0]	Input	Region Identifier: Sent on the read address channel for each read transaction.
S_AXI_ARUSER[m:0]	Input	Read Address Channel User
AXI4 Interface Read Address Channel: Handshake Signals for FIFO Write Interface		
S_AXI_ARVALID	Input	Read Address Valid: When high, indicates that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
S_AXI_ARREADY	Output	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4 Interface Read Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_ARID[m:0]	Output	Read Address ID. This signal is the identification tag for the read address group of signals.
M_AXI_ARADDR[m:0]	Output	Read Address: The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
M_AXI_ARLEN[7:0]	Output	Burst Length: The burst length gives the exact number of transfers in a burst. This information determines the number of data transfers associated with the address.
M_AXI_ARSIZE[2:0]	Output	Burst Size: This signal indicates the size of each transfer in the burst.
M_AXI_ARBURST[1:0]	Output	Burst Type: The burst type, coupled with the size information, details how the address for each transfer within the burst is calculated.

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

Name	Direction	Description
M_AXI_ARLOCK[2:0]	Output	Lock Type: This signal provides additional information about the atomic characteristics of the transfer.
M_AXI_ARCACHE[4:0]	Output	Cache Type: This signal provides additional information about the cacheable characteristics of the transfer.
M_AXI_ARPROT[3:0]	Output	Protection Type: This signal provides protection unit information for the transaction.
M_AXI_ARQOS[3:0]	Output	Quality of Service (QoS) signaling, sent on the read address channel for each read transaction.
M_AXI_ARREGION[3:0]	Output	Region Identifier: Sent on the read address channel for each read transaction.
M_AXI_ARUSER[m:0]	Output	Read Address Channel User
AXI4 Interface Read Address Channel: Handshake Signals for FIFO Read Interface		
M_AXI_ARVALID	Output	Read Address Valid: Indicates, when HIGH, that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
M_AXI_ARREADY	Input	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4 Read Address Channel FIFO: Optional Sideband Signals		
AXI_AR_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_AR_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_AR_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_AR_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AR_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.

Table 1-16: **AXI4 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 on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_AR_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO. **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.
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 1-17 defines the AXI4 FIFO interface signals for Read Data Channel.

Table 1-17: **AXI4 Read Data Channel FIFO Interface Signals**

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

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

Name	Direction	Description
AXI4 Read Data Channel FIFO: Optional Sideband Signals		
AXI_R_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_R_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_R_INJECTSBITERR	Input	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_R_INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_R_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_R_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_R_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO. **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.

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

Name	Direction	Description
AXI_R_RD_DATA_COUNT[*D*:0]	Output	Read Data Count: This bus indicates the number of words available for reading in the FIFO. The count is guaranteed to never over-report the number of words available for reading, to ensure that you do not underflow the FIFO. The exception to this behavior is when the read operation occurs at the rising edge of read clock, that read operation is only reflected on RD_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
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 1-18 defines the AXI4-Lite FIFO interface signals for Write Address Channel.

Table 1-18: **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.

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

Name	Direction	Description
S_AXI_AWREADY	Output	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Interface Write Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_AWADDR[m:0]	Output	Write Address: Gives the address of the first transfer in a write burst transaction. The associated control signals are used to determine the addresses of the remaining transfers in the burst.
M_AXI_AWPROT[3:0]	Output	Protection Type: This signal indicates the normal, privileged, or secure protection level of the transaction and whether the transaction is a data access or an instruction access.
AXI4-Lite Interface Write Address Channel: Handshake Signals for FIFO Read Interface		
M_AXI_AWVALID	Output	Write Address Valid: Indicates that valid write address and control information are available: • 1 = Address and control information available. • 0 = Address and control information not available. The address and control information remain stable until the address acknowledge signal, AWREADY, goes high.
M_AXI_AWREADY	Input	Write Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Write Address Channel FIFO: Optional Sideband Signals		
AXI_AW_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_AW_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_AW_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_AW_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AW_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.

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

Name	Direction	Description
AXI_AW_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_AW_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO. **Note**: This signal may have a constant value of 0 because the core does not allow additional writes when the FIFO is full.
AXI_AW_WR_DATA_COUNT[*D*:0]	Output	Write Data Count: This bus indicates the number of words written into the FIFO. The count is guaranteed to never underreport the number of words in the FIFO, to ensure you never overflow the FIFO. The exception to this behavior is when a write operation occurs at the rising edge of write clock, that write operation will only be reflected on WR_DATA_COUNT at the next rising clock edge. $D = \log_2(\text{FIFO depth})+1$
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 1-19 defines the AXI4-Lite FIFO interface signals for Write Data Channel.

Table 1-19: **AXI4-Lite Write Data Channel FIFO Interface Signals**

Name	Direction	Description
AXI4-Lite Interface Write Data Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus		
S_AXI_WDATA[m-1:0]	Input	Write Data: Can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
S_AXI_WSTRB[m/8-1:0]	Input	Write Strobes: Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)]. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid
AXI4-Lite Interface Write Data Channel: Handshake Signals for FIFO Write Interface		
S_AXI_WVALID	Input	Write Valid: Indicates that valid write data and strobes are available: • 1 = Write data and strobes available. • 0 = Write data and strobes not available.
S_AXI_WREADY	Output	Write Ready: Indicates that the slave can accept the write data: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Interface Write Data Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_WDATA[m-1:0]	Output	Write Data: Can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
M_AXI_WSTRB[m/8-1:0]	Output	Write Strobes: Indicates which byte lanes to update in memory. There is one write strobe for each eight bits of the write data bus. Therefore, WSTRB[n] corresponds to WDATA[(8 × n) + 7:(8 × n)]. For a 64-bit DATA, bit 0 corresponds to the least significant byte on DATA, and bit 7 corresponds to the most significant byte. For example: • STROBE[0] = 1b, DATA[7:0] is valid • STROBE[7] = 0b, DATA[63:56] is not valid
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 1-19: **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. $D = \log_2(\text{FIFO depth})-1$
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	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_W_INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_W_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_W_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_W_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO. **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. $D = \log_2(\text{FIFO depth})+1$
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. **Note**: This signal may have a constant value of 0 because the core does not allow additional reads when the FIFO is empty.

Table 1-19: **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 1-20 defines the AXI4-Lite FIFO interface signals for Write Response Channel.

Table 1-20: **AXI4-Lite Write Response Channel FIFO Interface Signals**

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

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

Name	Direction	Description
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. $D = \log_2(\text{FIFO depth})-1$
AXI_B_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. D is than log2(FIFO depth)-1
AXI_B_INJECTSBITERR	Input	Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_B_INJECTDBITERR	Input	Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_B_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_B_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_B_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO. **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. $D = \log_2(\text{FIFO depth})+1$

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

Name	Direction	Description
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 1-21 defines the AXI4-Lite FIFO interface signals for Read Address Channel.

Table 1-21: **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		

Table 1-21: **AXI4-Lite Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
S_AXI_ARVALID	Input	Read Address Valid: When high, indicates that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
S_AXI_ARREADY	Output	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Interface Read Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus		
M_AXI_ARADDR[m:0]	Output	Read Address: The read address bus gives the initial address of a read burst transaction. Only the start address of the burst is provided and the control signals that are issued alongside the address detail how the address is calculated for the remaining transfers in the burst.
M_AXI_ARPROT[3:0]	Output	Protection Type: This signal provides protection unit information for the transaction.
AXI4-Lite Interface Read Address Channel: Handshake Signals for FIFO Read Interface		
M_AXI_ARVALID	Output	Read Address Valid: WHen high, indicates that the read address and control information is valid and will remain stable until the address acknowledge signal, ARREADY, is high. • 1 = Address and control information valid. • 0 = Address and control information not valid.
M_AXI_ARREADY	Input	Read Address Ready: Indicates that the slave is ready to accept an address and associated control signals: • 1 = Slave ready. • 0 = Slave not ready.
AXI4-Lite Read Address Channel FIFO: Optional Sideband Signals		
AXI_AR_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2$(FIFO depth)-1
AXI_AR_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2$(FIFO depth)-1
AXI_AR_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single-bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.

Table 1-21: **AXI4-Lite Read Address Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_AR_INJECTDBITERR	Input	Inject Double-Bit Error: Injects a double-bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AR_SBITERR	Output	Single Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_AR_DBITERR	Output	Double Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_AR_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO. **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.
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 1-22 defines the AXI4-Lite FIFO interface signals for Write Data Channel.

Table 1-22: **AXI4-Lite Read Data Channel FIFO Interface Signals**

Name	Direction	Description
AXI4-Lite Interface Read Data Channel: Information Signals Mapped to FIFO Data Output (DOUT) Bus		
S_AXI_RDATA[m-1:0]	Output	Read Data: The read data bus can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
S_AXI_RRESP[1:0]	Output	Read Response: Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
AXI4-Lite Interface Read Data Channel: Handshake Signals for FIFO Read Interface		
S_AXI_RVALID	Output	Read Valid: Indicates that the required read data is available and the read transfer can complete: • 1 = Read data available. • 0 = Read data not available.
S_AXI_RREADY	Input	Read Ready: indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.
AXI4-Lite Interface Read Data Channel: Information Signals Derived from FIFO Data Input (DIN) Bus		
M_AXI_RDATA[m-1:0]	Input	Read Data: The read data bus can be 8, 16, 32, 64, 128, 256 or 512 bits wide.
M_AXI_ RRESP[1:0]	Input	Read Response: Indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.
AXI4-Lite Interface Read Data Channel: Handshake Signals for FIFO Write Interface		
M_AXI_RVALID	Input	Read Valid: Indicates that the required read data is available and the read transfer can complete: • 1 = Read data available. • 0 = Read data not available.
M_AXI_RREADY	Output	Read ready: Indicates that the master can accept the read data and response information: • 1= Master ready. • 0 = Master not ready.
AXI4-Lite Read Data Channel FIFO: Optional Sideband Signals		
AXI_R_PROG_FULL_THRESH[*D*:0]	Input	Programmable Full Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable full (PROG_FULL) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$
AXI_R_PROG_EMPTY_THRESH[*D*:0]	Input	Programmable Empty Threshold: This signal is used to input the threshold value for the assertion and de-assertion of the programmable empty (PROG_EMPTY) flag. The threshold can be dynamically set in-circuit during reset. $D = \log_2(\text{FIFO depth})-1$

Table 1-22: **AXI4-Lite Read Data Channel FIFO Interface Signals *(Cont'd)***

Name	Direction	Description
AXI_R_INJECTSBITERR	Input	Inject Single-Bit Error: Injects a single bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 FPGA block RAM FIFO.
AXI_R_INJECTDBITERR	Input	Inject DOuble-Bit Error. Injects a double bit error if the ECC feature is used on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_R_SBITERR	Output	Single-Bit Error: Indicates that the ECC decoder detected and fixed a single-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO.
AXI_R_DBITERR	Output	Double-Bit Error: Indicates that the ECC decoder detected a double-bit error on a Kintex-7, Virtex-7, or Virtex-6 block RAM FIFO and data in the FIFO core is corrupted.
AXI_R_OVERFLOW	Output	Overflow: This signal indicates that a write request during the prior clock cycle was rejected, because the FIFO is full. Overflowing the FIFO is not destructive to the FIFO. **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 = $\log_2$(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 = $\log_2$(FIFO depth)+1
AXI_R_DATA_COUNT[*D*:0]	Output	Data Count: This bus indicates the number of words stored in the FIFO. D = $\log_2$(FIFO depth)+1

Table 1-22: **AXI4-Lite Read Data Channel FIFO Interface Signals *(Cont'd)***

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

Applications

Native FIFO Applications

In digital designs, FIFOs are ubiquitous constructs required for data manipulation tasks such as clock domain crossing, low-latency memory buffering, and bus width conversion. Figure 1-7 highlights just one of many configurations that the FIFO Generator supports. In this example, the design has two independent clock domains and the width of the write data bus is four times wider than the read data bus. Using the FIFO Generator, you are able to rapidly generate solutions such as this one, that is customized for their specific requirements and provides a solution fully optimized for Xilinx FPGAs.

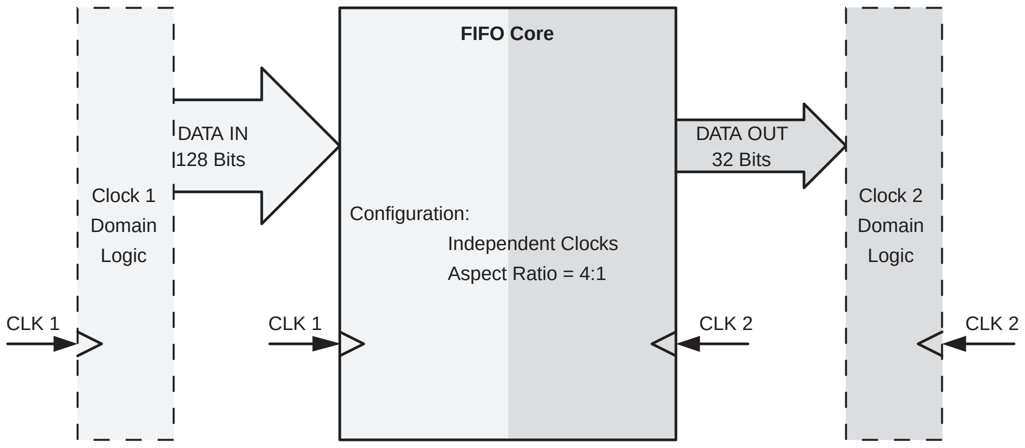


Figure 1-7: **FIFO Generator Application Example**

AXI4 FIFO Applications

AXI4-Stream FIFOs

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

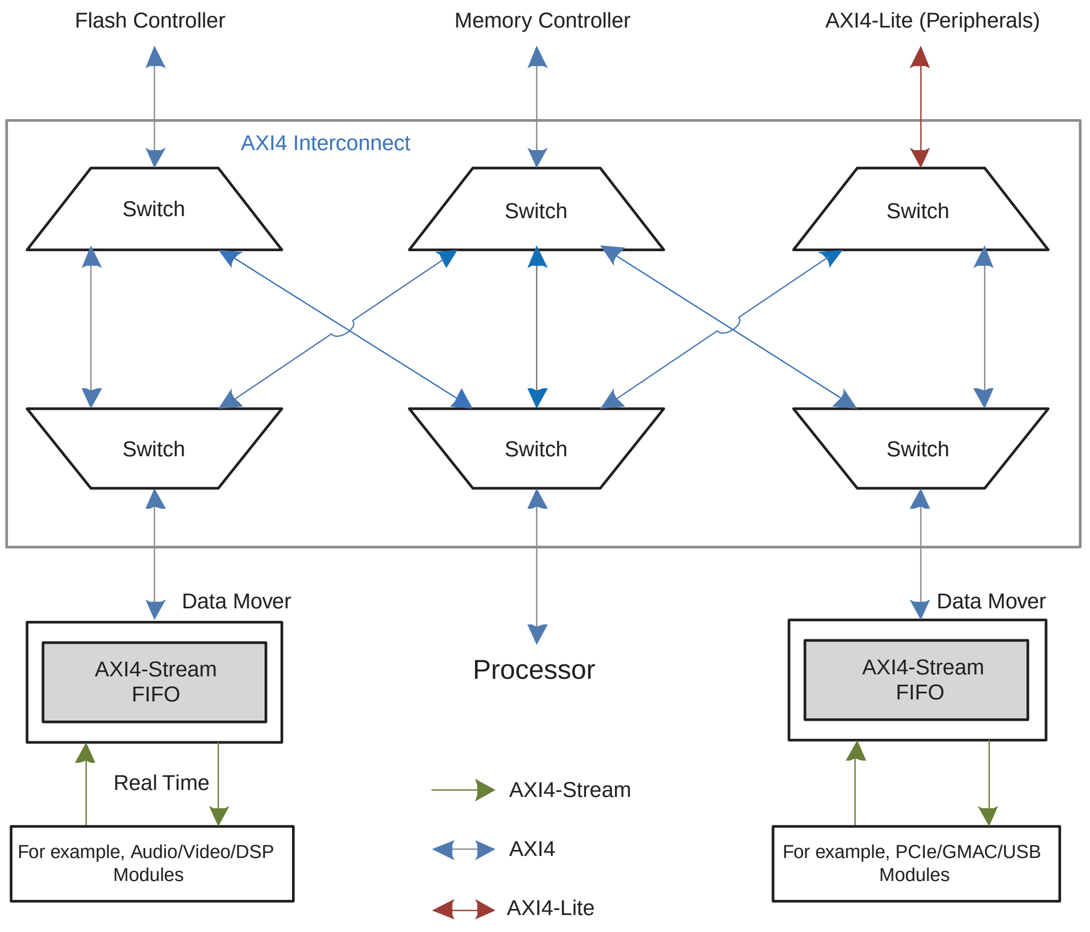


Figure 1-8: **AXI4-Stream Application Diagram**

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

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

AXI4 FIFOs (Memory Mapped)

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

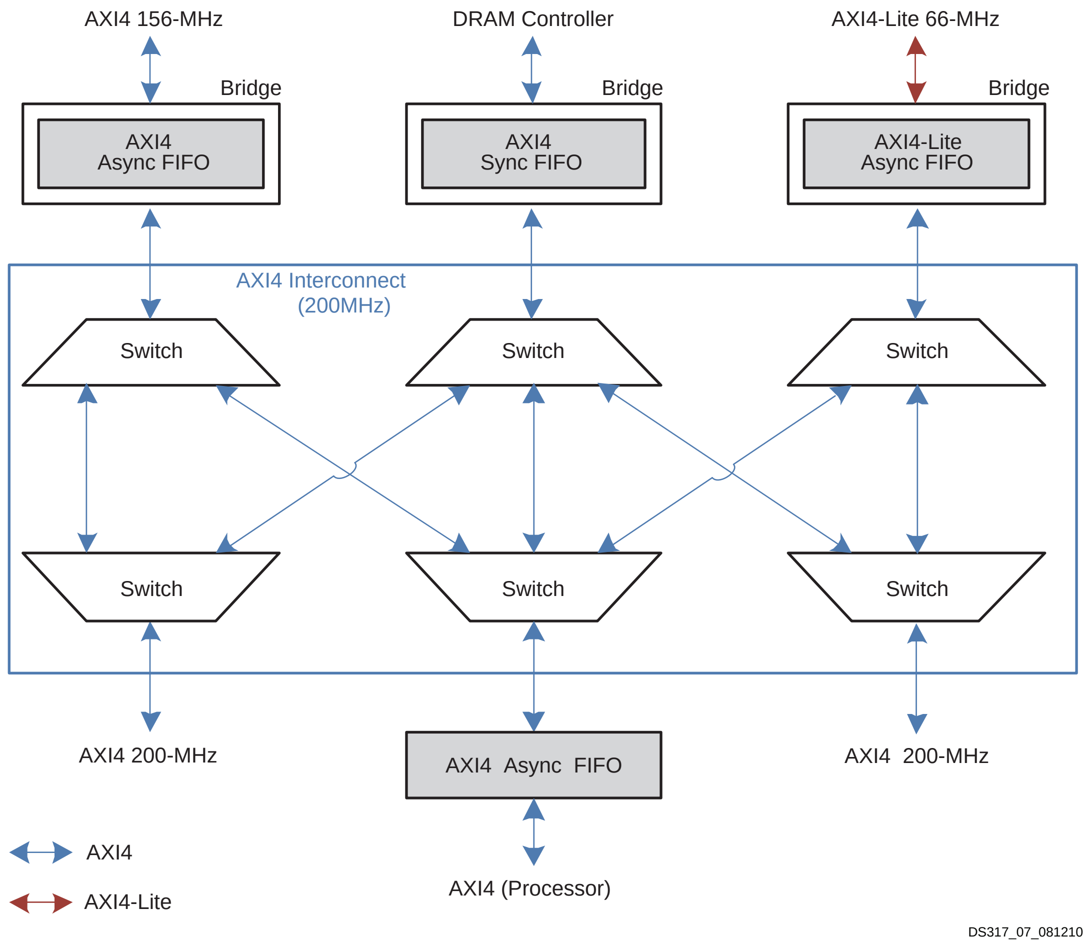


Figure 1-9: **AXI4 Application Diagram**

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

AXI4-Lite FIFOs

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

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

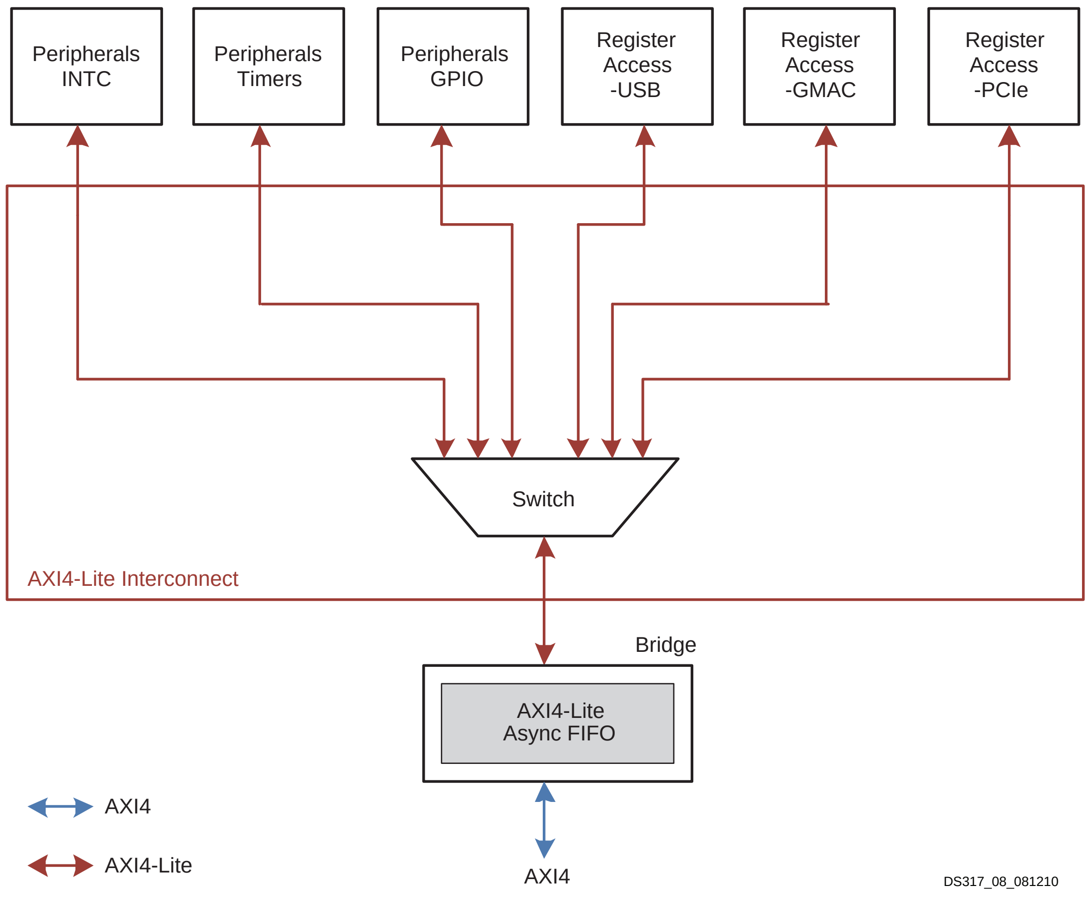


Figure 1-10: **AXI4-Lite Application Diagram**

Licensing and Ordering Information

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

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

Product Specification

This chapter includes details on performance and latency.

Performance

Performance and resource utilization for a FIFO varies depending on the configuration and features selected during core customization. The following tables show resource utilization data and maximum performance values for a variety of sample FIFO configurations.

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

Latency

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

Resource Utilization

Native FIFO Resource Utilization and Performance

Performance and resource utilization for a Native interface FIFO varies depending on the configuration and features selected during core customization. Table 2-1 through Table 2-4 show resource utilization data and maximum performance values for a variety of sample FIFO configurations.

The benchmarks were performed while adding two levels of registers on all inputs (except clock) and outputs having only the period constraints in the UCF. 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 with the ISE Design Suite using the following devices:

- Artix-7 (XC7A350T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)
- Virtex-6 (XC6VLX760-FF1760-1)
- Virtex-5 (XC5VLX330T-FF1738-1)
- Virtex-4 (XC4VLX200-FF1513-10)
- Spartan-6 (XC6SLX150T-FGG900-2)

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

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

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Common Clock FIFO (Block RAM)	512 x 16	Artix-7	270	47	48	1	0	0
		Kintex-7	325	114	48	1	0	0
		Virtex-7	325	112	48	1	0	0
		Virtex-6	335	39	48	1	0	0
		Virtex-5	320	45	53	1	0	0
		Virtex-4	335	39	48	1	0	0
		Spartan-6	275	79	48	1	0	0
	4096 x 16	Artix-7	265	66	60	2	0	0
		Kintex-7	350	121	60	2	0	0
		Virtex-7	355	127	60	2	0	0
		Virtex-6	325	61	60	2	0	0
		Virtex-5	305	55	65	2	0	0
		Virtex-4	325	61	60	2	0	0
		Spartan-6	245	91	60	4	0	0

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

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Common Clock FIFO (Distributed RAM)	512 x 16	Artix-7	250	254	68	0	0	176
		Kintex-7	345	309	65	0	0	176
		Virtex-7	350	324	65	0	0	176
		Virtex-6	380	252	68	0	0	176
		Virtex-5	305	260	69	0	0	176
		Virtex-4	380	252	68	0	0	176
		Spartan-6	230	257	73	0	0	176
	64 x 16	Artix-7	325	66	52	0	0	22
		Kintex-7	420	109	52	0	0	22
		Virtex-7	440	110	52	0	0	22
		Virtex-6	465	47	52	0	0	22
		Virtex-5	380	49	57	0	0	22
		Virtex-4	465	47	52	0	0	22
		Spartan-6	265	55	53	0	0	22
Independent Clock FIFO (Block RAM)	512 x 16	Artix-7	265	82	132	1	0	0
		Kintex-7	335	141	132	1	0	0
		Virtex-7	335	150	132	1	0	0
		Virtex-6	330	73	132	1	0	0
		Virtex-5	320	77	136	1	0	0
		Virtex-4	330	73	132	1	0	0
		Spartan-6	277	114	132	1	0	0
	4096 x 16	Artix-7	275	100	172	2	0	0
		Kintex-7	340	157	172	2	0	0
		Virtex-7	350	187	172	2	0	0
		Virtex-6	325	61	60	2	0	0
		Virtex-5	300	55	65	2	0	0
		Virtex-4	325	61	60	2	0	0
		Spartan-6	245	91	60	4	0	0

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

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Independent Clock FIFO (Distributed RAM)	512 x 16	Artix-7	275	279	148	0	0	176
		Kintex-7	355	338	148	0	0	176
		Virtex-7	370	354	148	0	0	176
		Virtex-6	350	293	148	0	0	176
		Virtex-5	320	292	152	0	0	176
		Virtex-4	400	281	148	0	0	176
		Spartan-6	245	300	154	0	0	176
	64 x 16	Artix-7	365	67	110	0	0	22
		Kintex-7	445	124	110	0	0	22
		Virtex-7	475	145	110	0	0	22
		Virtex-6	485	64	110	0	0	22
		Virtex-5	395	69	113	0	0	22
		Virtex-4	475	65	110	0	0	22
		Spartan-6	315	82	110	0	0	22
Shifting Register FIFO	512 x 16	Artix-7	195	711	53	0	496	22
		Kintex-7	250	768	53	0	497	0
		Virtex-7	240	768	53	0	496	0
		Virtex-6	325	375	53	0	256	0
		Virtex-5	290	389	56	0	256	0
		Virtex-4	325	375	53	0	256	0
		Spartan-6	190	422	67	0	256	0
	64 x 16	Artix-7	300	126	44	0	64	0
		Kintex-7	420	162	44	0	64	0
		Virtex-7	410	179	44	0	64	0
		Virtex-6	465	70	44	0	32	0
		Virtex-5	425	83	47	0	32	0
		Virtex-4	465	70	44	0	32	0
		Spartan-6	275	89	45	0	32	0

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

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

- Kintex-7 (XC7K480T-FFG1156-1)
- Virtex-6 (XC6VLX760-FF1760-1)
- Virtex-5 (XC5VLX330T-FF1738-1)
- Virtex-4 (XC4VLX200-FF1513-10)
- Spartan-6 (XC6SLX150T-FGG900-2)

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

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

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Common Clock FIFO (Block RAM)	512 x 16	Artix-7	245	76	72	1	0	0
		Kintex-7	325	130	72	1	0	0
		Virtex-7	325	139	72	1	0	0
		Virtex-6	335	75	72	1	0	0
		Virtex-5	320	74	77	1	0	0
		Virtex-4	325	72	76	1	0	0
		Spartan-6	270	99	72	1	0	0
	4096 x 16	Artix-7	265	97	90	2	0	0
		Kintex-7	340	152	90	2	0	0
		Virtex-7	375	156	90	2	0	0
		Virtex-6	330	91	90	2	0	0
		Virtex-5	305	92	95	2	0	0
		Virtex-4	330	91	90	2	0	0
		Spartan-6	255	126	90	4	0	0

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

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Common Clock FIFO (Distributed RAM)	512 x 16	Artix-7	250	282	95	0	0	176
		Kintex-7	355	345	88	0	0	176
		Virtex-7	350	338	88	0	0	176
		Virtex-6	370	278	88	0	0	176
		Virtex-5	300	289	93	0	0	176
		Virtex-4	325	288	88	0	0	176
		Spartan-6	230	288	97	0	0	176
	64 x 16	Artix-7	290	83	70	0	0	22
		Kintex-7	400	136	70	0	0	22
		Virtex-7	335	128	70	0	0	22
		Virtex-6	455	69	70	0	0	22
		Virtex-5	380	69	75	0	0	22
		Virtex-4	425	63	70	0	0	22
		Spartan-6	260	77	71	0	0	22
Independent Clock FIFO (Block RAM)	512 x 16	Artix-7	265	116	152	1	0	0
		Kintex-7	325	168	152	1	0	0
		Virtex-7	330	181	152	1	0	0
		Virtex-6	335	119	152	1	0	0
		Virtex-5	320	105	156	1	0	0
		Virtex-4	335	119	152	1	0	0
		Spartan-6	255	145	152	1	0	0
	4096 x 16	Artix-7	285	144	197	2	0	0
		Kintex-7	350	202	197	2	0	0
		Virtex-7	320	221	197	2	0	0
		Virtex-6	330	91	90	2	0	0
		Virtex-5	305	92	95	2	0	0
		Virtex-4	320	153	197	2	0	0
		Spartan-6	260	187	197	0	0	0

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

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Independent Clock FIFO (Distributed RAM)	512 x 16	Artix-7	255	311	169	0	0	176
		Kintex-7	355	376	169	0	0	176
		Virtex-7	365	382	169	0	0	176
		Virtex-6	355	315	169	0	0	176
		Virtex-5	320	320	172	0	0	176
		Virtex-4	355	315	169	0	0	176
		Spartan-6	255	324	174	0	0	176
	64 x 16	Artix-7	345	92	124	0	0	22
		Kintex-7	450	136	124	0	0	22
		Virtex-7	470	159	124	0	0	22
		Virtex-6	485	90	124	0	0	22
		Virtex-5	400	90	127	0	0	22
		Virtex-4	485	90	124	0	0	22
		Spartan-6	315	101	124	0	0	22
Shifting Register FIFO	512 x 16	Artix-7	190	756	76	0	512	0
		Kintex-7	245	814	76	0	512	0
		Virtex-7	225	819	76	0	512	0
		Virtex-6	305	399	75	0	256	0
		Virtex-5	285	416	78	0	256	0
		Virtex-4	300	400	75	0	256	0
		Spartan-6	215	433	81	0	256	0
	64 x 16	Artix-7	295	130	61	0	64	0
		Kintex-7	400	177	61	0	64	0
		Virtex-7	400	176	61	0	64	0
		Virtex-6	435	88	60	0	32	0
		Virtex-5	425	105	63	0	32	0
		Virtex-4	435	88	60	0	32	0
		Spartan-6	255	124	60	0	32	0

Table 2-3 provides results for FIFOs configured to use the Virtex-5 FPGA built-in FIFO. The benchmarks were performed using the following devices:

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

- Kintex-7 (XC7K480T-FFG1156-1)
- Virtex-6 (XC6VLX760-FF1760-1)
- Virtex-5 (XC5VLX330T-FF1738-1)

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

Table 2-3: **Benchmarks: FIFO Configured with Virtex-5 and Virtex-6 FIFO36 Resources**

FIFO Type	Depth x Width	FPGA Family	Read Mode	Performance (MHz)	LUTs	FFs	FIFO36
Common Clock FIFO36 (Basic)	512 x 72	Artix-7	Standard	265	3	7	1
			FWFT	255	3	9	1
		Kintex-7	Standard	320	2	7	1
			FWFT	310	3	9	1
		Virtex-7	Standard	215	2	7	1
			FWFT	290	4	9	1
		Virtex-6	Standard	325	2	7	1
			FWFT	325	3	9	1
		Virtex-5	Standard	305	2	7	1
			FWFT	305	4	4	1
	16k x 8	Artix-7	Standard	225	8	11	4
			FWFT	220	9	15	4
		Kintex-7	Standard	265	8	11	4
			FWFT	270	8	15	4
		Virtex-7	Standard	205	7	11	4
			FWFT	235	8	15	4
		Virtex-6	Standard	270	7	11	4
			FWFT	275	7	15	4
		Virtex-5	Standard	300	12	11	4
			FWFT	280	15	15	4

Table 2-3: **Benchmarks: FIFO Configured with Virtex-5 and Virtex-6 FIFO36 Resources *(Cont'd)***

FIFO Type	Depth x Width	FPGA Family	Read Mode	Performance (MHz)	LUTs	FFs	FIFO36
Common Clock FIFO36 (With Handshaking)	512 x 72	Artix-7	Standard	260	7	11	1
			FWFT	250	6	12	1
		Kintex-7	Standard	320	6	11	1
			FWFT	300	6	12	1
		Virtex-7	Standard	210	6	11	1
			FWFT	300	6	12	1
		Virtex-6	Standard	320	6	11	1
			FWFT	325	7	12	1
		Virtex-5	Standard	305	6	11	1
			FWFT	305	8	12	1
	16k x 8	Artix-7	Standard	220	11	15	4
			FWFT	225	14	18	4
		Kintex-7	Standard	250	11	15	4
			FWFT	270	12	18	4
		Virtex-7	Standard	250	10	15	4
			FWFT	215	11	18	4
		Virtex-6	Standard	260	10	15	4
			FWFT	280	9	18	4
		Virtex-5	Standard	250	14	15	4
			FWFT	295	18	18	4

Table 2-3: **Benchmarks: FIFO Configured with Virtex-5 and Virtex-6 FIFO36 Resources *(Cont'd)***

FIFO Type	Depth x Width	FPGA Family	Read Mode	Performance (MHz)	LUTs	FFs	FIFO36
Independent Clock FIFO36 (Basic)	512 x 72	Artix-7	Standard	300	3	7	1
			FWFT	305	3	7	1
		Kintex-7	Standard	385	2	7	1
			FWFT	385	2	7	1
		Virtex-7	Standard	315	2	7	1
			FWFT	315	2	7	1
		Virtex-6	Standard	325	3	7	1
			FWFT	325	3	9	1
		Virtex-5	Standard	305	7	2	1
			FWFT	305	9	4	1
	16k x 8	Artix-7	Standard	255	6	7	4
			FWFT	245	5	7	4
		Kintex-7	Standard	335	5	7	4
			FWFT	345	5	7	4
		Virtex-7	Standard	250	5	7	4
			FWFT	320	5	7	4
		Virtex-6	Standard	305	8	7	4
			FWFT	320	6	7	4
		Virtex-5	Standard	295	8	7	4
			FWFT	295	8	7	4

Table 2-3: **Benchmarks: FIFO Configured with Virtex-5 and Virtex-6 FIFO36 Resources *(Cont'd)***

FIFO Type	Depth x Width	FPGA Family	Read Mode	Performance (MHz)	LUTs	FFs	FIFO36
Independent Clock FIFO36 (With Handshaking)	512 x 72	Artix-7	Standard	280	7	18	1
			FWFT	345	6	10	1
		Kintex-7	Standard	410	8	18	1
			FWFT	410	5	10	1
		Virtex-7	Standard	330	7	18	1
			FWFT	400	5	10	1
		Virtex-6	Standard	405	8	18	1
			FWFT	325	5	12	1
		Virtex-5	Standard	450	8	18	1
			FWFT	450	6	10	1
	16k x 8	Artix-7	Standard	255	10	18	4
			FWFT	265	8	10	4
		Kintex-7	Standard	315	10	18	4
			FWFT	315	8	10	4
		Virtex-7	Standard	220	9	18	4
			FWFT	210	8	10	4
		Virtex-6	Standard	335	14	18	4
			FWFT	355	13	10	4
		Virtex-5	Standard	305	12	18	4
			FWFT	310	11	10	4

Table 2-4 provides results for FIFOs configured to use the Virtex-4 built-in FIFO with patch. The benchmarks were performed using a Virtex-4 (XC4VLX200-FF1513-10) FPGA.

Table 2-4: **Benchmarks: FIFO Configured with Virtex-4 FIFO16 Patch**

FIFO Type	Depth x Width	Clock Ratios	Performance (MHz)	LUTs	FFs	FIFO16s
Built-in FIFO (basic)	512x36	WR_CLK ≥ RD_CLK	210	118	114	0
		RD_CLK > WR_CLK	210	115	110	0
Built-in FIFO (Handshaking)	512x36	WR_CLK ≥ RD_CLK	210	121	119	0
		RD_CLK > WR_CLK	210	117	115	0

AXI4 FIFO Resource Utilization and Performance

Table 2-5 provides the default configuration settings for the benchmarks data. Table 2-6 shows benchmark information for AXI4 and AXI4-Lite configurations. The benchmarks were obtained using the following devices:

- Artix-7 (XC7A350T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)
- Virtex-6 (XC6VLX760-FF1760-1)
- Spartan-6 (XC6SLX150T-FGG900-2)

Table 2-5: **AXI4 and AXI4-Lite Default Configuration Settings**

AXI Type	FIFO Type	Channel Type	ID, Address and Data Width	FIFO Depth x Width

AXI4	Distributed RAM	Write Address	ID = 4
			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	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

Table 2-6: **AXI4 and AXI4-Lite Resource Utilization**

FIFO Type	Clock Type	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
AXI4	Common Clock	Artix-7	260	344	601	5	0	92
		Kintex-7	315	231	601	2	0	92
		Virtex-7	179	326	601	5	0	92
		Virtex-6	310	326	601	5	0	92
		Spartan-6	240	482	481	4	0	92
	Independent Clock	Artix-7	231	430	894	5	0	92
		Kintex-7	335	394	768	2	0	92
		Virtex-7	194	453	895	5	0	92
		Virtex-6	290	411	895	5	0	92
		Spartan-6	245	535	768	4	0	92

Table 2-6: **AXI4 and AXI4-Lite Resource Utilization *(Cont'd)***

FIFO Type	Clock Type	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
AXI4-Lite	Common Clock	Artix-7	245	234	457	4	0	52
		Kintex-7	350	194	457	2	0	52
		Virtex-7	214	238	457	4	0	52
		Virtex-6	324	288	457	4	0	52
		Spartan-6	230	509	465	8	0	52
	Independent Clock	Artix-7	240	343	752	4	0	52
		Kintex-7	350	273	650	2	0	52
		Virtex-7	190	394	752	4	0	52
		Virtex-6	325	358	752	4	0	52
		Spartan-6	252	635	756	8	0	52

Table 2-7 provides benchmarking results for AXI4-Stream FIFO configurations. The benchmarks were obtained using the following devices:

- Artix-7 (XC7A350T- FFG1156-1)
- Virtex-7 (XC7V2000T-FLG1925-1)
- Kintex-7 (XC7K480T-FFG1156-1)
- Virtex-6 (XC6VLX760-FF1760-1)
- Spartan-6 (XC6SLX150T-FGG900-2)

Table 2-7: **AXI4-Stream Resource Utilization**

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Common Clock FIFO (Block RAM)	512 x 16	Artix-7	254	55	67	1	0	0
		Kintex-7	355	116	67	1	0	0
		Virtex-7	329	109	67	1	0	0
		Virtex-6	334	55	67	1	0	0
		Spartan-6	277	86	67	1	0	0
	4096 x 16	Artix-7	260	79	79	2	0	0
		Kintex-7	325	123	79	2	0	0
		Virtex-7	325	117	79	2	0	0
		Virtex-6	335	73	79	2	0	0
		Spartan-6	276	124	79	2	0	0
Common Clock FIFO (Distributed RAM)	512 x 16	Artix-7	259	262	87	0	0	176
		Kintex-7	378	318	83	0	0	176
		Virtex-7	349	321	83	0	0	176
		Virtex-6	300	258	87	0	0	176
		Spartan-6	220	268	92	0	0	176
	64 x16	Artix-7	308	61	71	0	0	22
		Kintex-7	445	121	71	0	0	22
		Virtex-7	466	115	71	0	0	22
		Virtex-6	475	53	71	0	0	22
		Spartan-6	301	63	72	0	0	22

Table 2-7: **AXI4-Stream Resource Utilization *(Cont'd)***

FIFO Type	FPGA Family	Depth x Width	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Independent Clock FIFO (Block RAM)	512 x 16	Artix-7	266	87	151	1	0	0
		Kintex-7	355	151	151	1	0	0
		Virtex-7	325	159	151	1	0	0
		Virtex-6	330	108	151	1	0	0
		Spartan-6	274	120	151	1	0	0
	4096 x 16	Artix-7	282	127	190	2	0	0
		Kintex-7	355	171	190	2	0	0
		Virtex-7	350	201	190	2	0	0
		Virtex-6	325	109	190	2	0	0
		Spartan-6	244	171	190	4	0	0
Independent Clock FIFO (Distributed RAM)	512 x 16	Artix-7	110	283	167	0	0	176
		Kintex-7	375	351	167	0	0	176
		Virtex-7	395	363	167	0	0	176
		Virtex-6	415	293	167	0	0	176
		Spartan-6	239	310	171	0	0	176
	64 x 16	Artix-7	340	103	128	0	0	22
		Kintex-7	485	154	128	0	0	22
		Virtex-7	495	173	128	0	0	22
		Virtex-6	476	95	128	0	0	22
		Spartan-6	280	127	128	0	0	22

Port Descriptions

Native FIFO Port Summary

Table 2-8 describes all the FIFO Generator ports.

Table 2-8: **FIFO Generator Ports**

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

Table 2-8: **FIFO Generator Ports *(Cont'd)***

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
SBITERR	O	Yes	Yes	Yes
DBITERR	O	Yes	Yes	Yes
RD_RST	I	Yes	Yes	No

AXI4 FIFO Port Summary

AXI4 Global Interface Ports

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

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

AXI4-Stream FIFO Interface Ports

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

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

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

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

AXI4 FIFO Interface Ports

Write Channels

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

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI4 Interface Write Address Channel: Information Signals Mapped to FIFO Data Input (DIN) bus				
S_AXI_AWID[m:0]	Input	No	Yes	Yes
S_AXI_AWADDR[m:0]	Input	No	Yes	Yes

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

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

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

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
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 2-12: **AXI4 Write Data Channel FIFO Interface Ports**

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

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

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
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 2-13: **AXI4 Write Response Channel FIFO Interface Ports**

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

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

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
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 2-14: **AXI4 Read Address Channel FIFO Interface Ports**

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI4 Interface Read Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus				
S_AXI_ARID[m:0]	Input	No	Yes	Yes
S_AXI_ARADDR[m:0]	Input	No	Yes	Yes
S_AXI_ARLEN[7:0]	Input	No	Yes	Yes
S_AXI_ARSIZE[2:0]	Input	No	Yes	Yes
S_AXI_ARBURST[1:0]	Input	No	Yes	Yes
S_AXI_ARLOCK[2:0]	Input	No	Yes	Yes
S_AXI_ARCACHE[4:0]	Input	No	Yes	Yes
S_AXI_ARPROT[3:0]	Input	No	Yes	Yes
S_AXI_ARQOS[3:0]	Input	No	Yes	Yes
S_AXI_ARREGION[3:0]	Input	No	Yes	Yes
S_AXI_ARUSER[m:0]	Input	Yes	Yes	Yes
AXI4 Interface Read Address Channel: Handshake Signals for FIFO Write Interface				

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

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
S_AXI_ARVALID	Input	No	Yes	Yes
S_AXI_ARREADY	Output	No	Yes	Yes
AXI4 Interface, Read Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus				
M_AXI_ARID[m:0]	Output	No	Yes	Yes
M_AXI_ARADDR[m:0]	Output	No	Yes	Yes
M_AXI_ARLEN[7:0]	Output	No	Yes	Yes
M_AXI_ARSIZE[2:0]	Output	No	Yes	Yes
M_AXI_ARBURST[1:0]	Output	No	Yes	Yes
M_AXI_ARLOCK[2:0]	Output	No	Yes	Yes
M_AXI_ARCACHE[4:0]	Output	No	Yes	Yes
M_AXI_ARPROT[3:0]	Output	No	Yes	Yes
M_AXI_ARQOS[3:0]	Output	No	Yes	Yes
M_AXI_ARREGION[3:0]	Output	No	Yes	Yes
M_AXI_ARUSER[m:0]	Output	Yes	Yes	Yes
AXI4 Interface Read Address Channel: Handshake Signals for FIFO Read Interface				
M_AXI_ARVALID	Output	No	Yes	Yes
M_AXI_ARREADY	Input	No	Yes	Yes
AXI4 Read Address Channel FIFO: Optional Sideband Signals				
AXI_AR_PROG_FULL_THRESH[m:0]	Input	Yes	Yes	Yes
AXI_AR_PROG_EMPTY_THRESH[m:0]	Input	Yes	Yes	Yes
AXI_AR_INJECTSBITERR	Input	Yes	Yes	Yes
AXI_AR_INJECTDBITERR	Input	Yes	Yes	Yes
AXI_AR_SBITERR	Output	Yes	Yes	Yes
AXI_AR_DBITERR	Output	Yes	Yes	Yes
AXI_AR_OVERFLOW	Output	Yes	Yes	Yes
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 2-15: **AXI4 Read Data Channel FIFO Interface Ports**

Port Name	Input or Output	Optional Port	Port Available	
			Common Clock	Independent Clocks
AXI4 Interface Read Data Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus				
S_AXI_RID[m:0]	Output	No	Yes	Yes
S_AXI_RDATA[m-1:0]	Output	No	Yes	Yes
S_AXI_RRESP[1:0]	Output	No	Yes	Yes
S_AXI_RLAST	Output	No	Yes	Yes
S_AXI_RUSER[m:0]	Output	Yes	Yes	Yes
AXI4 Interface Read Data Channel: Handshake Signals for FIFO Read Interface				
S_AXI_RVALID	Output	No	Yes	Yes
S_AXI_RREADY	Input	No	Yes	Yes
AXI4 Interface Read Data Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus				
M_AXI_RID[m:0]	Input	No	Yes	Yes
M_AXI_RDATA[m-1:0]	Input	No	Yes	Yes
M_AXI_ RRESP[1:0]	Input	No	Yes	Yes
M_AXI_RLAST	Input	No	Yes	Yes
M_AXI_RUSER[m:0]	Input	Yes	Yes	Yes
AXI4 Interface, Read Data Channel: Handshake Signals for FIFO Read Interface				
M_AXI_RVALID	Input	No	Yes	Yes
M_AXI_RREADY	Output	No	Yes	Yes
AXI4 Read Data Channel FIFO: Optional Sideband Signals				
AXI_R_PROG_FULL_THRESH[m:0]	Input	Yes	Yes	Yes
AXI_R_PROG_EMPTY_THRESH[m:0]	Input	Yes	Yes	Yes
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 2-16: **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
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 2-17: **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				
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 2-18: **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
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 2-19: **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
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 2-20: **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
AXI_R_INJECTDBITERR	Input	Yes	Yes	Yes
AXI_R_SBITERR	Output	Yes	Yes	Yes
AXI_R_DBITERR	Output	Yes	Yes	Yes
AXI_R_OVERFLOW	Output	Yes	Yes	Yes
AXI_R_WR_DATA_COUNT[m:0]	Output	Yes	Yes	No
AXI_R_UNDERFLOW	Output	Yes	Yes	Yes
AXI_R_RD_DATA_COUNT[m:0]	Output	Yes	Yes	No
AXI_R_DATA_COUNT[m:0]	Output	Yes	No	Yes
AXI_R_PROG_FULL	Output	Yes	Yes	Yes
AXI_R_PROG_EMPTY	Output	Yes	Yes	Yes

Designing with the Core

This chapter describes the steps required to turn a FIFO Generator core into a fully functioning design integrated with the user application logic.

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

General Design Guidelines

Know the Degree of Difficulty

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

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

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

Understand Signal Pipelining and Synchronization

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

Synchronization Considerations

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

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

Figure 3-1 shows the signals with respect to their clock domains. All signals are synchronous to a specific clock, with the exception of RST, which performs an asynchronous reset of the entire FIFO.

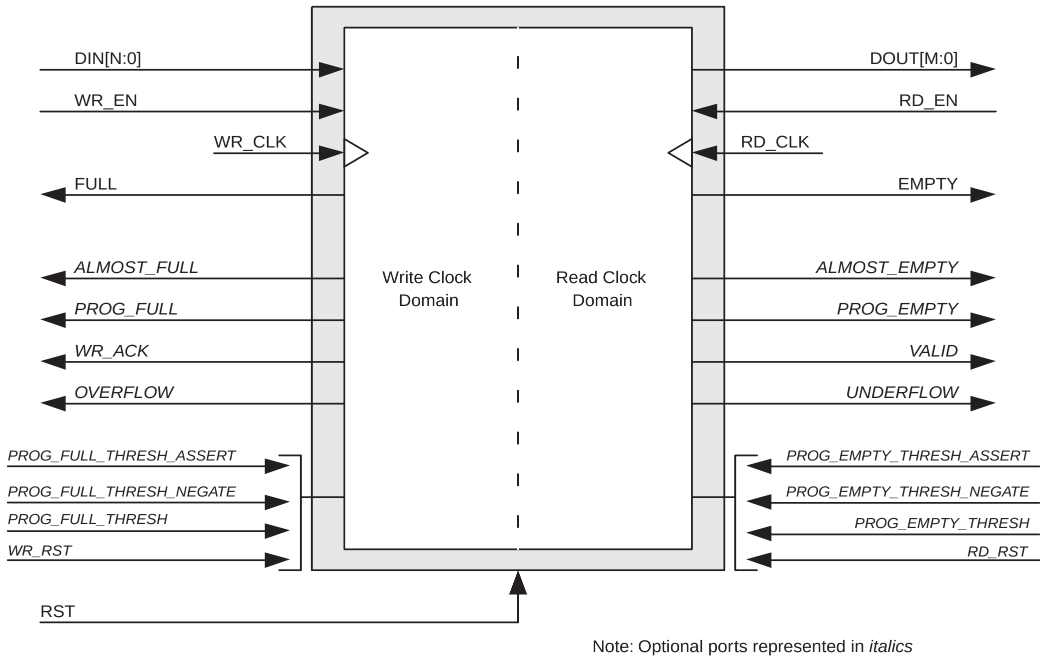


Figure 3-1: **FIFO with Independent Clocks: Write and Read Clock Domains**

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

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

Initializing the FIFO Generator

When designing with the built-in FIFO or common clock shift register FIFO, the FIFO must be reset after the FPGA is configured and before operation begins. An asynchronous reset pin (RST) is provided, which is an asynchronous reset that clears the internal counters and output registers.

For FIFOs implemented with block RAM or distributed RAM, a reset is not required, and the input pin is optional. For common clock configurations, users have the option of asynchronous or synchronous reset. For independent clock configurations, users have the option of asynchronous reset (RST) or synchronous reset (WR_RST/RD_RST) with respect to respective clock domains.

When asynchronous reset is implemented (Enable Reset Synchronization option is selected), it is synchronized to the clock domain in which it is used to ensure that the FIFO initializes to a known state. This synchronization logic allows for proper reset timing of the core logic, avoiding glitches and metastable behavior. The reset pulse and synchronization delay requirements are dependent on the FIFO implementation types.

When WR_RST/RD_RST is implemented (Enable Reset Synchronization option is not selected), the WR_RST/RD_RST is treated as a synchronous reset to the respective clock domain. The write clock domain remains in reset state as long as WR_RST is asserted, and the read clock domain remains in reset state as long as RD_RST is asserted. See Resets, page 125.

FIFO Usage and Control

Write Operation

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

ALMOST_FULL and FULL Flags

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

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

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

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

Example Operation

Figure 3-2 shows a typical write operation. The user asserts WR_EN, causing a write operation to occur on the next rising edge of the WR_CLK. Because the FIFO is not full, WR_ACK is asserted, acknowledging a successful write operation. When only one additional word can be written into the FIFO, the FIFO asserts the ALMOST_FULL flag. When ALMOST_FULL is asserted, one additional write causes the FIFO to assert FULL. When a write occurs after FULL is asserted, WR_ACK is deasserted and OVERFLOW is asserted, indicating an overflow condition. Once you perform one or more read operations, the FIFO deasserts FULL, and data can successfully be written to the FIFO, as is indicated by the assertion of WR_ACK and deassertion of OVERFLOW.

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

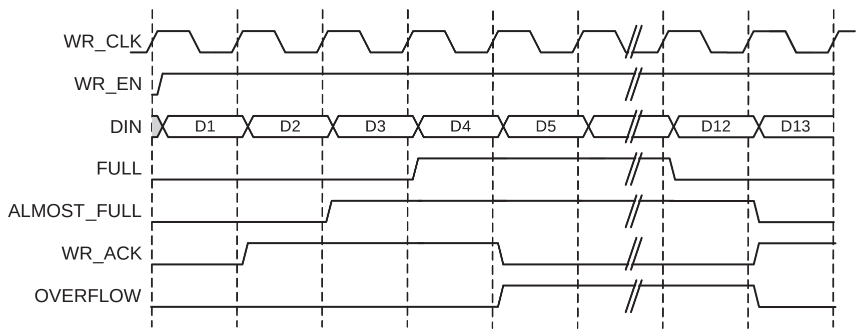


Figure 3-2: **Write Operation for a FIFO with Independent Clocks**

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

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

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

Common Clock Note

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

Modes of Read Operation

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

Table 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	✓(1)
Common Clock	Block RAM	✓
	Distributed RAM	✓
	Shift Register	
	Built-in	✓(1)

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

Standard FIFO Read Operation

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

Figure 3-3 shows a standard read access. Once the user writes at least one word into the FIFO, EMPTY is deasserted — indicating data is available to be read. The user asserts RD_EN, causing a read operation to occur on the next rising edge of RD_CLK. The FIFO outputs the next available word on DOUT and asserts VALID, indicating a successful read operation. When the last data word is read from the FIFO, the FIFO asserts EMPTY. If the user continues to assert RD_EN while EMPTY is asserted, the read request is ignored, VALID is deasserted, and UNDERFLOW is asserted. Once the user performs a write operation, the FIFO deasserts EMPTY, allowing the user to resume valid read operations, as indicated by the assertion of VALID and deassertion of UNDERFLOW.

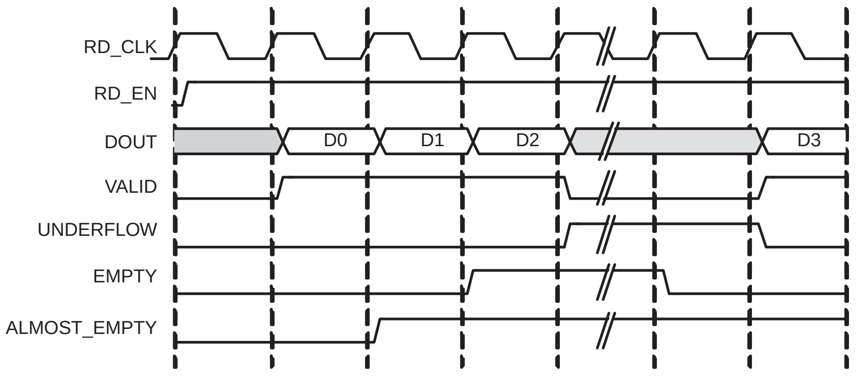


Figure 3-3: **Standard Read Operation for a FIFO with Independent Clocks**

First-Word Fall-Through FIFO Read Operation

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

Figure 3-4 shows a FWFT read access. Initially, the FIFO is not empty, the next available data word is placed on the output bus (DOUT), and VALID is asserted. When you assert RD_EN, the next rising clock edge of RD_CLK places the next data word onto DOUT. After the last data word has been placed on DOUT, an additional read request causes the data on DOUT to become invalid, as indicated by the deassertion of VALID and the assertion of EMPTY. Any further attempts to read from the FIFO results in an underflow condition.

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

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

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

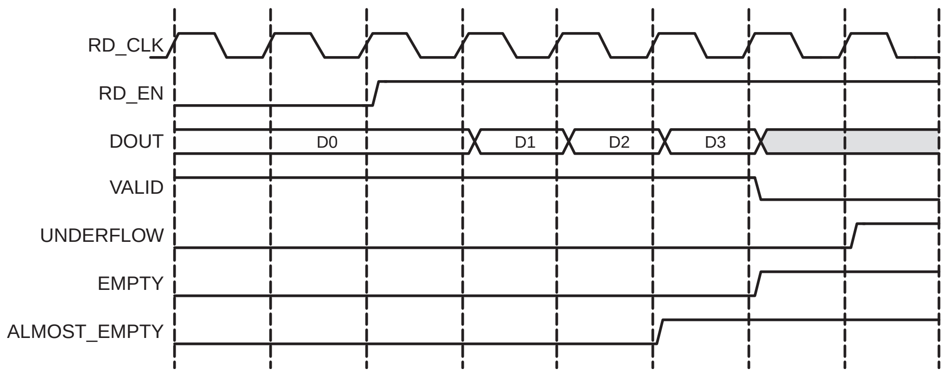


Figure 3-4: **FWFT Read Operation for a FIFO with Independent Clocks**

Common Clock FIFO, Simultaneous Read and Write Operation

Figure 3-5 shows a typical write and read operation. A write is issued to the FIFO, resulting in the deassertion of the EMPTY flag. A simultaneous write and read is then issued, resulting in no change in the status flags. Once two or more words are present in the FIFO, the ALMOST_EMPTY flag is deasserted. Write requests are then issued to the FIFO, resulting in the assertion of ALMOST_FULL when the FIFO can only accept one more write (without a read). A simultaneous write and read is then issued, resulting in no change in the status flags. Finally one additional write without a read results in the FIFO asserting FULL, indicating no further data can be written until a read request is issued.

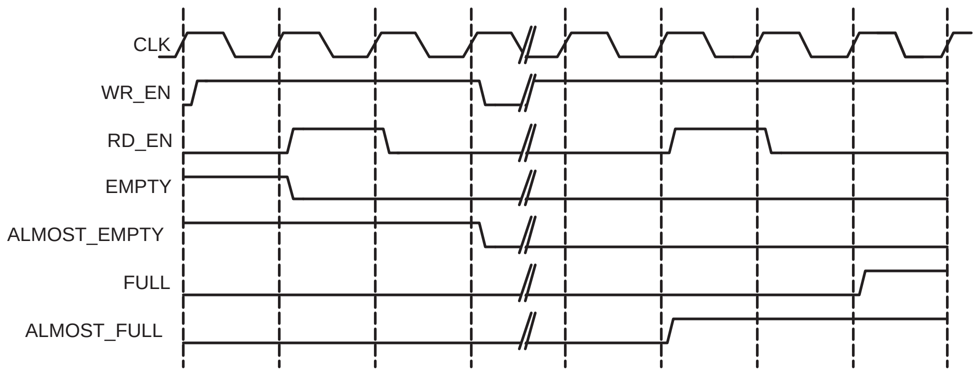


Figure 3-5: **Write and Read Operation for a FIFO with Common Clocks**

Handshaking Flags

Handshaking flags (valid, underflow, write acknowledge and overflow) are supported to provide additional information regarding the status of the write and read operations. The handshaking flags are optional, and can be configured as active high or active low through the FIFO Generator GUI. These flags (configured as active high) are illustrated in Figure 3-6.

Write Acknowledge

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

Valid

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

Standard FIFO Read Operation

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

FWFT FIFO Read Operation

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

Example Operation

Figure 3-6 illustrates the behavior of the FIFO flags. On the write interface, FULL is not asserted and writes to the FIFO are successful (as indicated by the assertion of WR_ACK). When a write occurs after FULL is asserted, WR_ACK is deasserted and OVERFLOW is asserted, indicating an overflow condition. On the read interface, once the FIFO is not EMPTY, the FIFO accepts read requests. In standard FIFO operation, VALID is asserted and DOUT is updated on the clock cycle following the read request. In FWFT operation, VALID is asserted and DOUT is updated prior to a read request being issued. When a read request is issued while EMPTY is asserted, VALID is deasserted and UNDERFLOW is asserted, indicating an underflow condition.

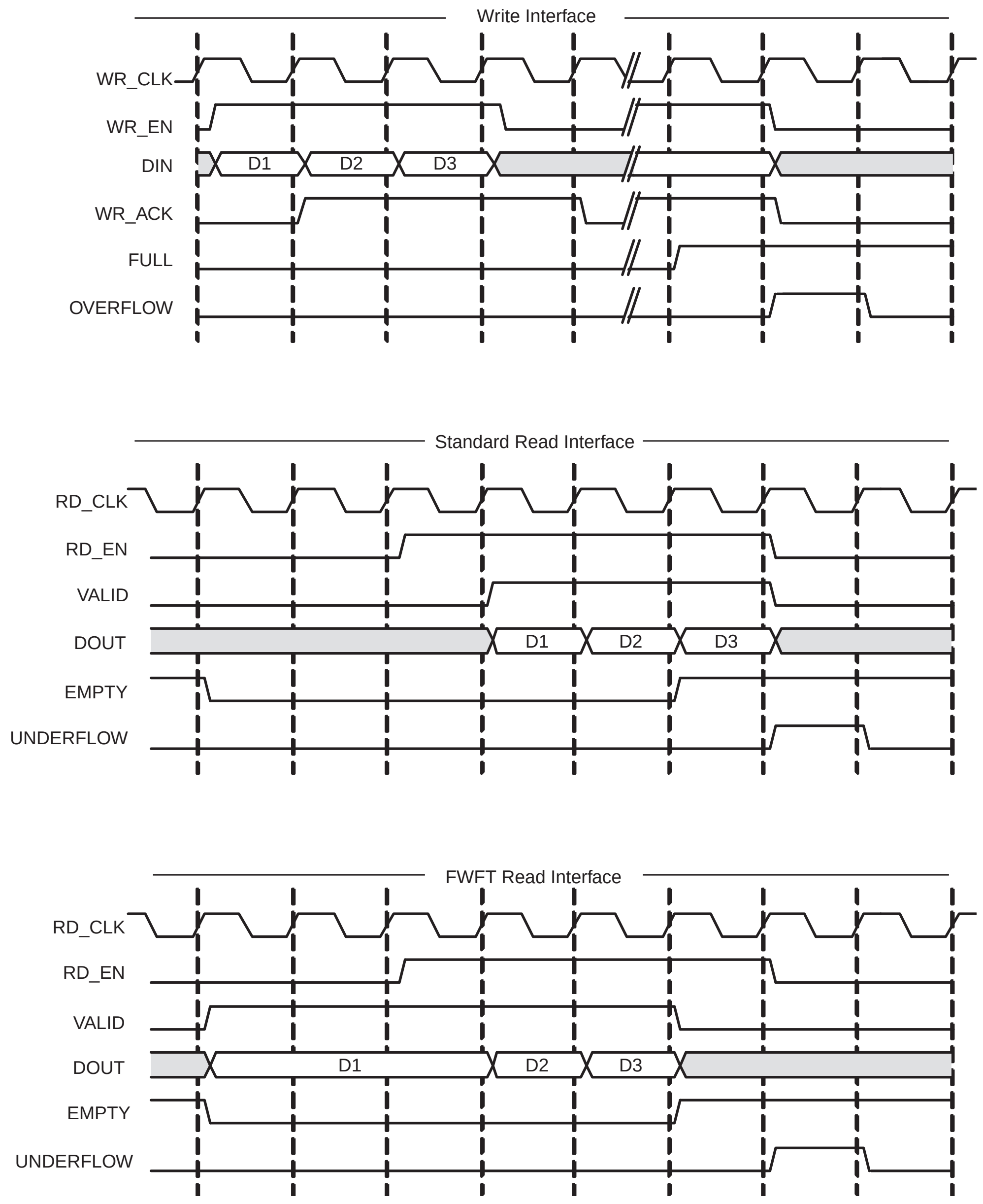


Figure 3-6: **Handshaking Signals for a FIFO with Independent Clocks**

Underflow

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

Overflow

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

Example Operation

Figure 3-7 illustrates the Handshaking flags. On the write interface, FULL is deasserted and therefore writes to the FIFO are successful (indicated by the assertion of WR_ACK). When a write occurs after FULL is asserted, WR_ACK is deasserted and OVERFLOW is asserted, indicating an overflow condition. On the read interface, once the FIFO is not EMPTY, the FIFO accepts read requests. Following a read request, VALID is asserted and DOUT is updated. When a read request is issued while EMPTY is asserted, VALID is deasserted and UNDERFLOW is asserted, indicating an underflow condition.

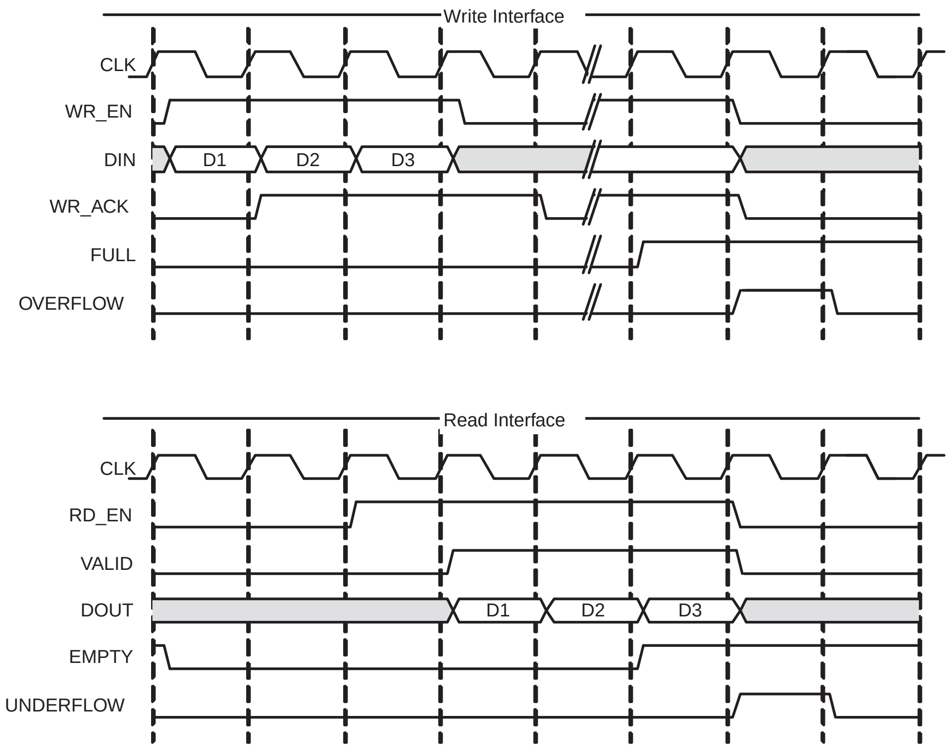


Figure 3-7: **Handshaking Signals for a FIFO with Common Clocks**

Programmable Flags

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

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

For these thresholds, you can set a constant value or choose to have dedicated input ports, enabling the thresholds to change dynamically in circuit. Hysteresis is also optionally supported, by providing unique assert and negate values for each flag. Detailed information about these options are provided below. For information about the latency behavior of the programmable flags, see Latency, page 135.

Programmable Full

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

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

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

These options are available in the FIFO Generator GUI and accessed within the programmable flags window (Figure 9-5, page 201).

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

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

Programmable Full: Single Threshold

This option enables you to set a single threshold value for the assertion and deassertion of `PROG_FULL`. When the number of entries in the FIFO is greater than or equal to the threshold value, `PROG_FULL` is asserted. The deassertion behavior differs between built-in and non built-in FIFOs (block RAM, distributed RAM, and so forth).

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

Two options are available to implement this threshold:

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

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

Figure 3-8 shows the programmable full flag with a single threshold for a non-built-in FIFO. The user writes to the FIFO until there are seven words in the FIFO. Because the programmable full threshold is set to seven, the FIFO asserts PROG_FULL once seven words are written into the FIFO.

TIP: *Both write data count (WR_DATA_COUNT) and PROG_FULL have one clock cycle of delay. Once the FIFO has six or fewer words in the FIFO, PROG_FULL is deasserted.*

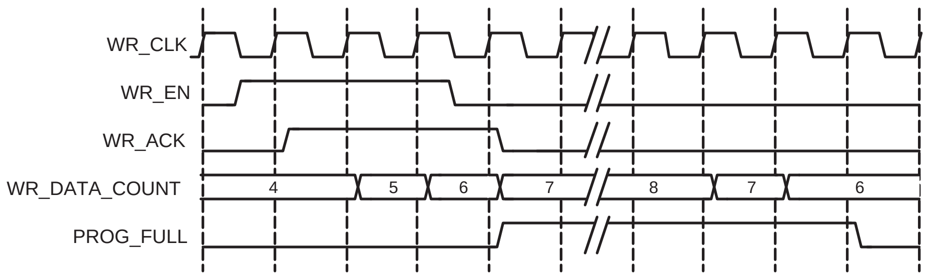


Figure 3-8: **Programmable Full Single Threshold: Threshold Set to 7**

Programmable Full: Assert and Negate Thresholds

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

IMPORTANT: *This feature is not available for built-in FIFOs.*

Two options are available to implement these thresholds:

- Assert and negate threshold constants: User specifies the threshold values through the FIFO Generator GUI. Once the core is generated, these values can only be changed by re-generating the core. This option consumes fewer resources than the assert and negate thresholds with dedicated input ports.

- Assert and negate thresholds with dedicated input ports: User specifies the threshold values through input ports on the core. These input ports can be changed while the FIFO is in reset, providing you the flexibility to change the values of the programmable full assert (PROG_FULL_THRESH_ASSERT) and negate (PROG_FULL_THRESH_NEGATE) thresholds in-circuit without re-generating the core.

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

Figure 3-9 shows the programmable full flag with assert and negate thresholds. The user writes to the FIFO until there are 10 words in the FIFO. Because the assert threshold is set to 10, the FIFO then asserts PROG_FULL. The negate threshold is set to seven, and the FIFO deasserts PROG_FULL once six words or fewer are in the FIFO. Both write data count (WR_DATA_COUNT) and PROG_FULL have one clock cycle of delay.

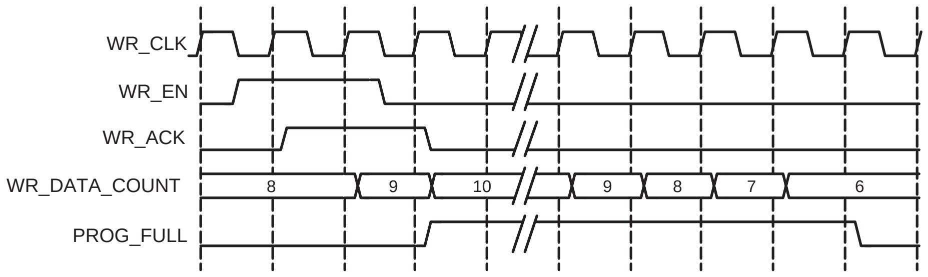


Figure 3-9: **Programmable Full with Assert and Negate Thresholds: Assert Set to 10 and Negate Set to 7**

Programmable Full Threshold Range Restrictions

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

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

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

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

Programmable Empty

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

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

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

These options are available in the FIFO Generator GUI and accessed within the programmable flags window (Figure 9-5, page 201).

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

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

Programmable Empty: Single Threshold

This option enables you to set a single threshold value for the assertion and deassertion of PROG_EMPTY. When the number of entries in the FIFO is less than or equal to the threshold value, PROG_EMPTY is asserted. The deassertion behavior differs between built-in and non built-in FIFOs (block RAM, distributed RAM, and so forth).

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

Two options are available to implement this threshold:

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

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

Figure 3-10 shows the programmable empty flag with a single threshold for a non-built-in FIFO. The user writes to the FIFO until there are five words in the FIFO. Because the programmable empty threshold is set to four, PROG_EMPTY is asserted until more than four words are present in the FIFO. Once five words (or more) are present in the FIFO, PROG_EMPTY is deasserted. Both read data count (RD_DATA_COUNT) and PROG_EMPTY have one clock cycle of delay.

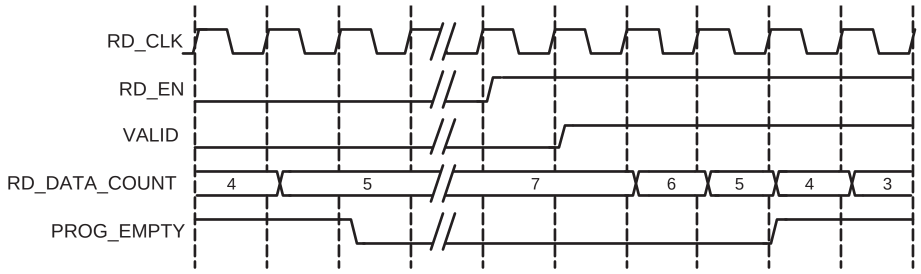


Figure 3-10: **Programmable Empty with Single Threshold: Threshold Set to 4**

Programmable Empty: Assert and Negate Thresholds

This option lets you set separate values for the assertion and deassertion of PROG_EMPTY. When the number of entries in the FIFO is less than or equal to the assert value, PROG_EMPTY is asserted. When the number of entries in the FIFO is greater than the negate value, PROG_EMPTY is deasserted. This feature is not available for built-in FIFOs.

Two options are available to implement these thresholds.

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

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

Figure 3-11 shows the programmable empty flag with assert and negate thresholds. The user writes to the FIFO until there are eleven words in the FIFO; because the programmable empty deassert value is set to ten, PROG_EMPTY is deasserted when more than ten words are in the FIFO. Once the FIFO contains less than or equal to the programmable empty negate value (set to seven), PROG_EMPTY is asserted. Both read data count (RD_DATA_COUNT) and PROG_EMPTY have one clock cycle of delay.

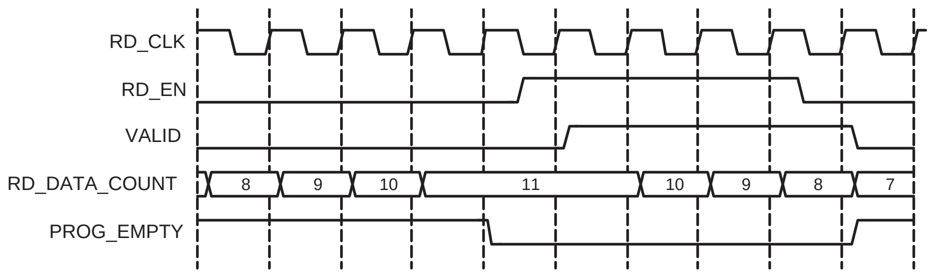


Figure 3-11: **Programmable Empty with Assert and Negate Thresholds: Assert Set to 7 and Negate Set to 10**

Programmable Empty Threshold Range Restrictions

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

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

The FIFO Generator GUI automatically parameterizes the threshold ranges based on these features, allowing you to choose only within the valid ranges.

IMPORTANT: *For Common or Independent Clock Built-in FIFO implementation type, you can only choose a threshold range within 1 primitive deep of the FIFO depth due to the core implementation. If a wider threshold range is needed, use the Common or Independent Clock Block RAM implementation type.*

Note: Refer to the FIFO Generator GUI for valid ranges for each threshold. To avoid unexpected behavior, do not use out-of-range threshold values.

Data Counts

DATA_COUNT tracks the number of words in the FIFO. You can specify the width of the data count bus with a maximum width of log2 (FIFO depth). If the width specified is smaller than the maximum allowable width, the bus is truncated by removing the lower bits. These signals are optional outputs of the FIFO Generator, and are enabled through the FIFO Generator GUI. Table 3-2 identifies data count support for each FIFO implementation. For information about the latency behavior of data count flags, see Latency, page 135.

Table 3-2: **Implementation-specific Support for Data Counts**

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

Data Count (Common Clock FIFO Only)

Data Count output (`DATA_COUNT`) accurately reports the number of words available in a Common Clock FIFO. You can specify the width of the data count bus with a maximum width of log2(depth). If the width specified is smaller than the maximum allowable width, the bus is truncated with the lower bits removed.

For example, you can specify to use two bits out of a maximum allowable three bits (provided a FIFO depth of eight). These two bits indicate the number of words in the FIFO with a quarter resolution, providing the status of the contents of the FIFO for read and write operations.

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

Read Data Count (Independent Clock FIFO Only)

Read data count (`RD_DATA_COUNT`) pessimistically reports the number of words available for reading. The count is guaranteed to never over-report the number of words available in the FIFO (although it may temporarily under-report the number of words available) to ensure that the user design never underflows the FIFO. You can specify the width of the read data count bus with a maximum width of log2 (read depth). If the width specified is smaller than the maximum allowable width, the bus is truncated with the lower bits removed.

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

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

Write Data Count (Independent Clock FIFO Only)

Write data count (WR_DATA_COUNT) pessimistically reports the number of words written into the FIFO. The count is guaranteed to never under-report the number of words in the FIFO (although it may temporarily over-report the number of words present) to ensure that you never overflow the FIFO. You can specify the width of the write data count bus with a maximum width of log2 (write depth). If the width specified is smaller than the maximum allowable width, the bus is truncated with the lower bits removed.

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

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

First-Word Fall-Through Data Count

By providing the capability to read the next data word before requesting it, first-word fall-through (FWFT) implementations increase the depth of the FIFO by 2 read words. Using this configuration, the FIFO Generator enables you to generate data count in two ways:

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

Approximate Data Count

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

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

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

More Accurate Data Count (Use Extra Logic)

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

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

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

Data Count Behavior

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

Table 3-3 defines the value of DATA_COUNT when FIFO is empty.

From the point-of-view of the write interface, DATA_COUNT is always accurate, reporting the first word immediately once its written to the FIFO. However, from the point-of-view of the read interface, the DATA_COUNT output may over-report by up to two words until ALMOST_EMPTY and EMPTY have both deasserted. This is due to the latency of EMPTY deassertion in the first-word-fall-through FIFO (see Table 3-17). This latency allows DATA_COUNT to reflect written words which may not yet be available for reading.

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

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

Write Data Count Behavior

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

Table 3-3 defines the value of WR_DATA_COUNT when the FIFO is empty.

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

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

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

Table 3-3: **Empty FIFO WR_DATA_COUNT/DATA_COUNT Value**

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

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

Example Operation

Figure 3-12 shows write and read data counts. When WR_EN is asserted and FULL is deasserted, WR_DATA_COUNT increments. Similarly, when RD_EN is asserted and EMPTY is deasserted, RD_DATA_COUNT decrements.

Note: In the first part of Figure 3-12, a successful write operation occurs on the third rising clock edge, and is not reflected on WR_DATA_COUNT until the next full clock cycle is complete. Similarly, RD_DATA_COUNT transitions one full clock cycle after a successful read operation.

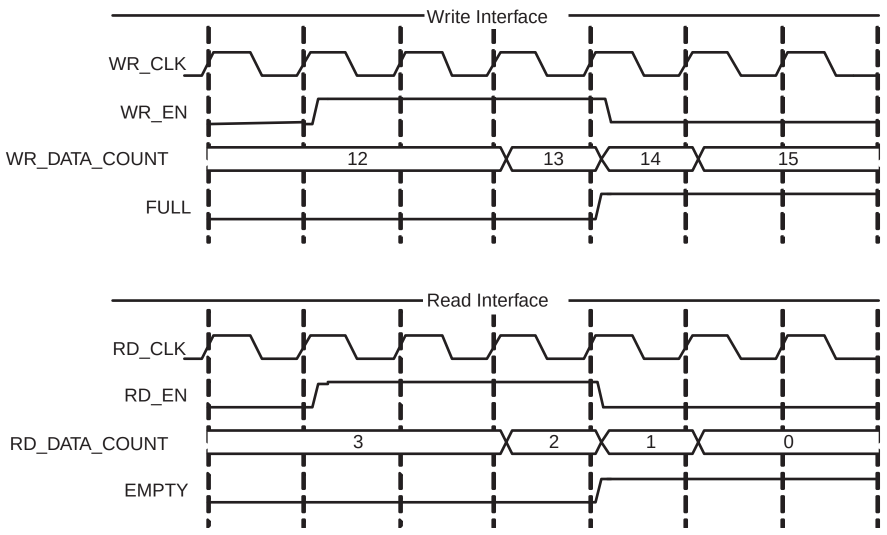


Figure 3-12: **Write and Read Data Counts for FIFO with Independent Clocks**

Non-symmetric Aspect Ratios

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

Table 3-4: **Implementation-specific Support for Non-symmetric Aspect Ratios**

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

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

For non-symmetric aspect ratios, the full and empty flags are active only when one complete word can be written or read. The FIFO does not allow partial words to be accessed. For example, assuming a full FIFO, if the write width is 8 bits and read width is 2 bits, 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.

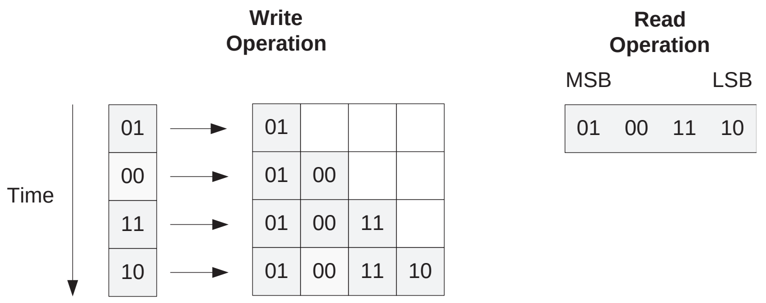


Figure 3-13: **1:4 Aspect Ratio: Data Ordering**

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.

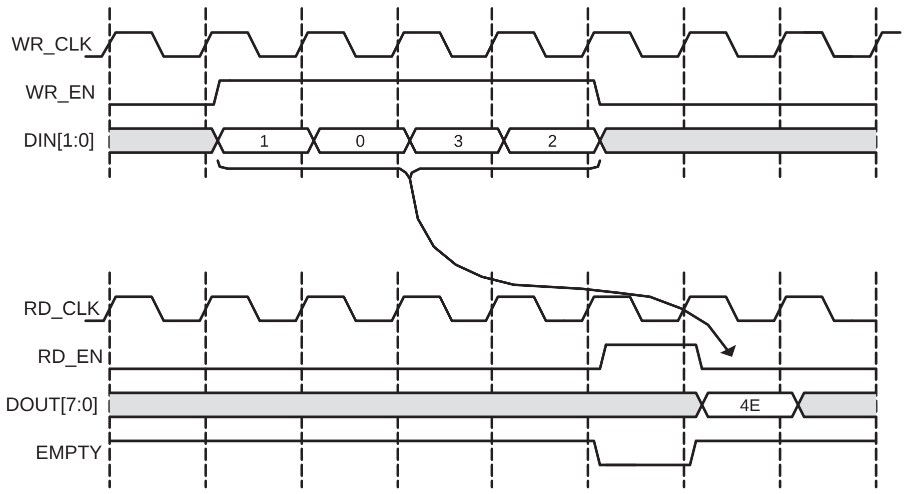


Figure 3-14: **1:4 Aspect Ratio: Status Flag Behavior**

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.

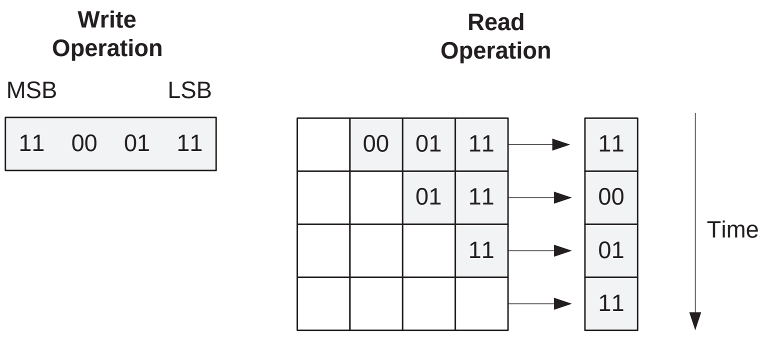


Figure 3-15: **4:1 Aspect Ratio: Data Ordering**

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.

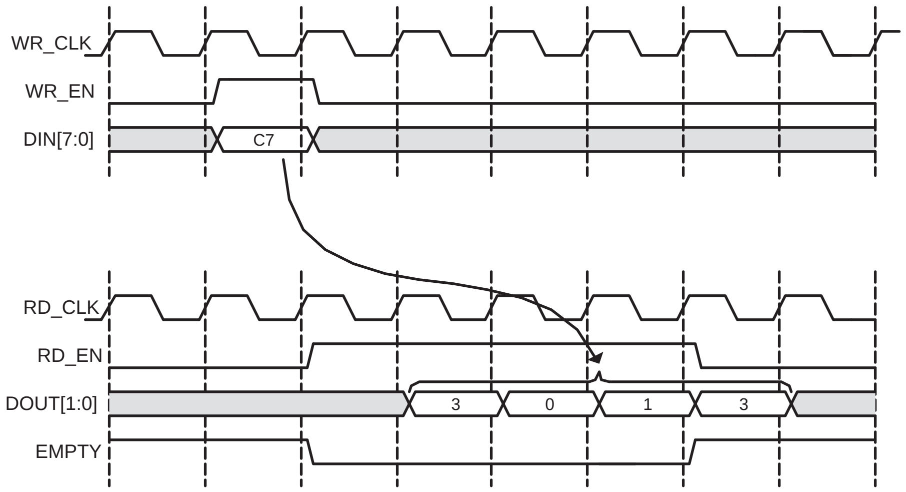


Figure 3-16: **4:1 Aspect Ratio: Status Flag Behavior**

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 (Zynq-7000, 7 Series, Virtex-6, Virtex-5 and Virtex-4 FPGAs)

The block RAM macros available in Zynq-7000, Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 devices, as well as built-in FIFO macros available in Zynq-7000, Kintex-7, Virtex-7, Virtex-6 and Virtex-5 devices, have built-in embedded registers that can be used to pipeline data and improve macro timing. Depending on the configuration, this feature can be leveraged to add one additional latency to the FIFO core (DOUT bus and VALID outputs) or implement the output registers for FWFT FIFOs. For built-in FIFOs configuration, this feature is only available for common clock FIFOs.

Standard FIFOs

When using the embedded registers to add an output pipeline register to the standard FIFOs, only the DOUT and VALID output ports are delayed by one clock cycle during a read operation. These additional pipeline registers are always enabled, as illustrated in Figure 3-17.

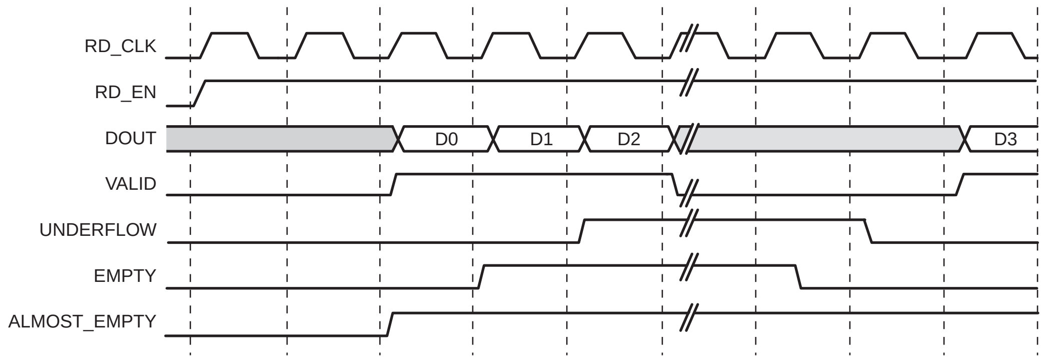


Figure 3-17: **Standard Read Operation for a Block RAM or built-in FIFO with Use Embedded Registers Enabled**

Block RAM Based FWFT FIFOs

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

Built-in Based FWFT FIFOs (Common Clock Only)

When using the embedded output registers with a common clock built-in based FIFO with FWFT, the embedded registers add an output pipeline register to the FWFT FIFO. The DOUT and VALID output ports are delayed by 1 clock cycle during a read operation. These pipeline registers are always enabled, as illustrated in Figure 3-18, The DOUT reset value feature is not supported in Virtex-4 and Virtex-5 FPGAs. For this configuration, the embedded output register feature is only available for FIFOs that use only one FIFO macro in depth.

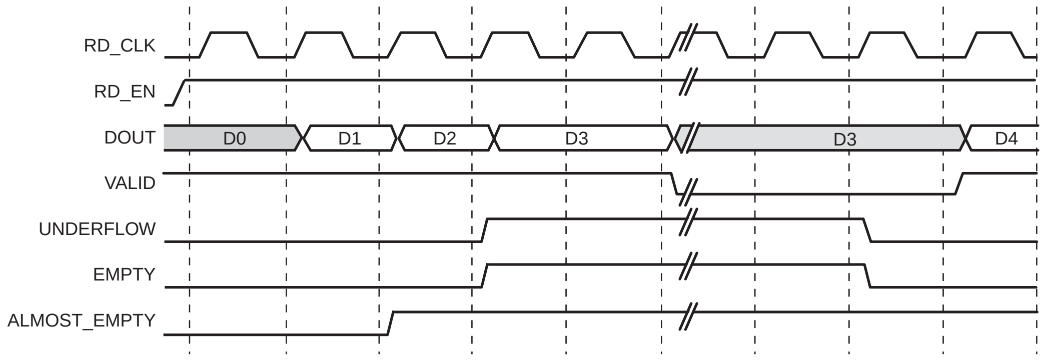


Figure 3-18: **FWFT Read Operation for a Synchronous Built-in FIFO with User Embedded Registers Enabled**

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

When using the embedded output registers with a common clock built-in FIFO in Kintex-7, Virtex-7, and Virtex-6 FPGAs, the DOUT reset value feature is supported, as illustrated in Figure 3-19.

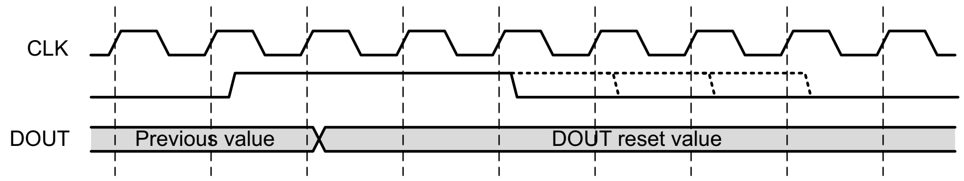


Figure 3-19: **DOUT Reset Value for Kintex-7, Virtex-7, and Virtex-6 Common Clock Built-in FIFO Embedded Register**

Built-in Error Correction Checking

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

Figure 3-20 shows how the SBITERR and DBITERR outputs are generated in the FIFO Generator core. The output signals are created by combining all the SBITERR and DBITERR signals from the FIFO or block RAM primitives using an OR gate. Because the FIFO primitives may be cascaded in depth, when SBITERR or DBITERR is asserted, the error may have occurred in any of the built-in FIFO macros chained in depth or block RAM macros. For this reason, these flags are not correlated to the data currently being read from the FIFO Generator core or to a read operation. For this reason, when the DBITERR is flagged, assume that the data in the entire FIFO has been corrupted and the user logic needs to take the appropriate action. As an example, when DBITERR is flagged, an appropriate action for the user logic is to halt all FIFO operation, reset the FIFO, and restart the data transfer.

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

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

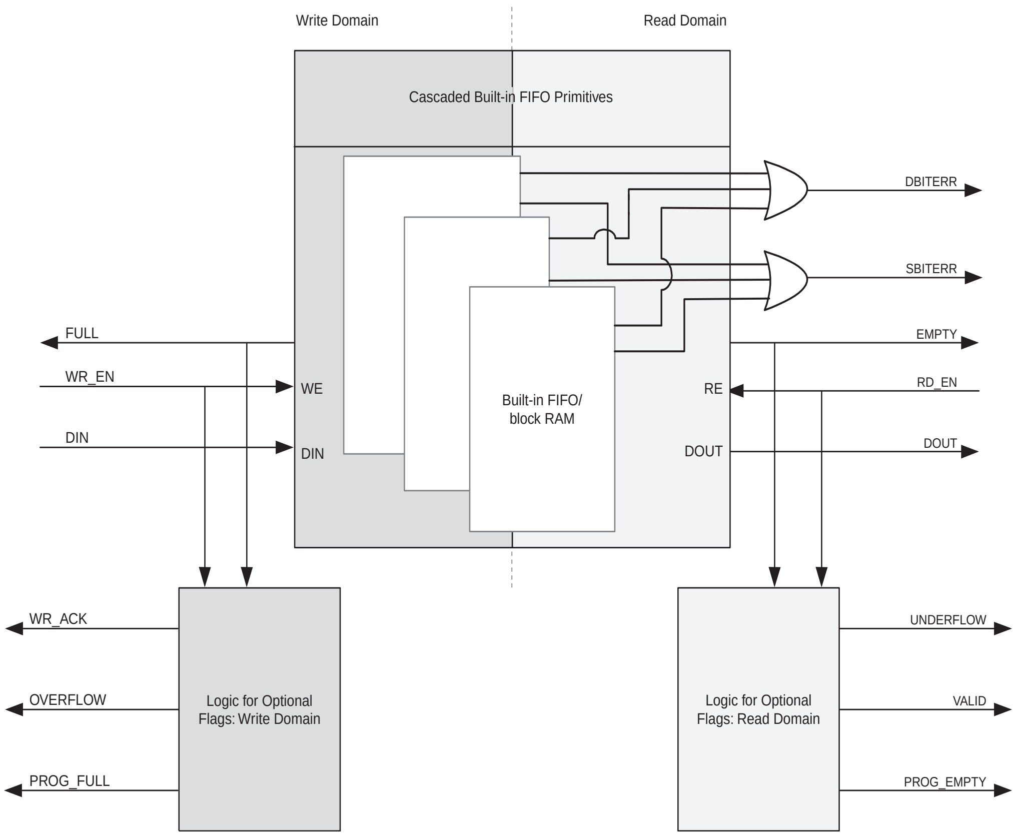


Figure 3-20: **SBITERR and DBITERR Outputs in the FIFO Generator Core**

Built-in Error Injection

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

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

When `INJECTSBITERR` is asserted on a write operation, a single bit error is injected and `SBITERR` is asserted upon read operation of a specific write. When `INJECTDBITERR` is asserted on a write operation, a double bit error is injected and `DBITERR` is asserted upon read operation of a specific write. When both INJECTSBITERR and INJECTDBITERR are asserted on a write operation, a double bit error is injected and DBITERR is asserted upon read operation of a specific write. Figure 3-21 shows how the SBITERR and DBITERR outputs are generated in the FIFO Generator core.

Note: Reset is not supported by the FIFO/BRAM macros when using the ECC option. Therefore, outputs of the FIFO core (DOUT, DBITERR and SBITERR) will not be affected by reset, and they hold their previous values. See Resets, page 125 for more details.

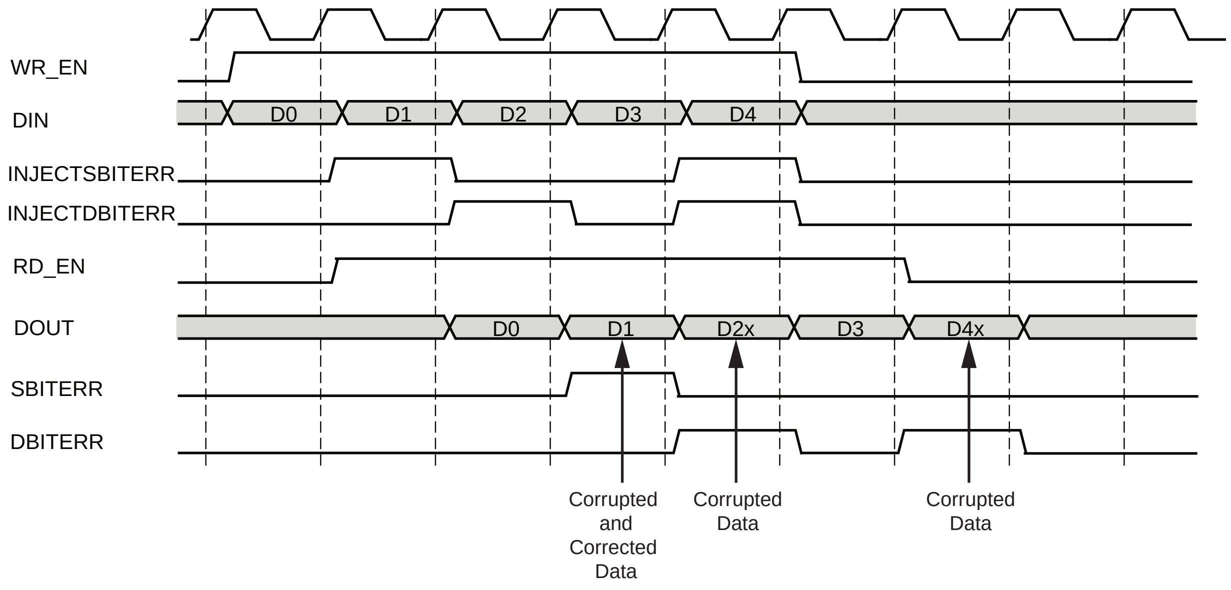


Figure 3-21: **Error Injection and Correction in Kintex-7, Virtex-7, and Virtex-6 FPGAs**

Clocking

Each FIFO configuration has a set of allowable features, as defined in Table 1-4, page 18.

Independent Clocks: Block RAM and Distributed RAM

Figure 3-22 illustrates the functional implementation of a FIFO configured with independent clocks. This implementation uses block RAM or distributed RAM for memory, counters for write and read pointers, conversions between binary and Gray code for synchronization across clock domains, and logic for calculating the status flags.

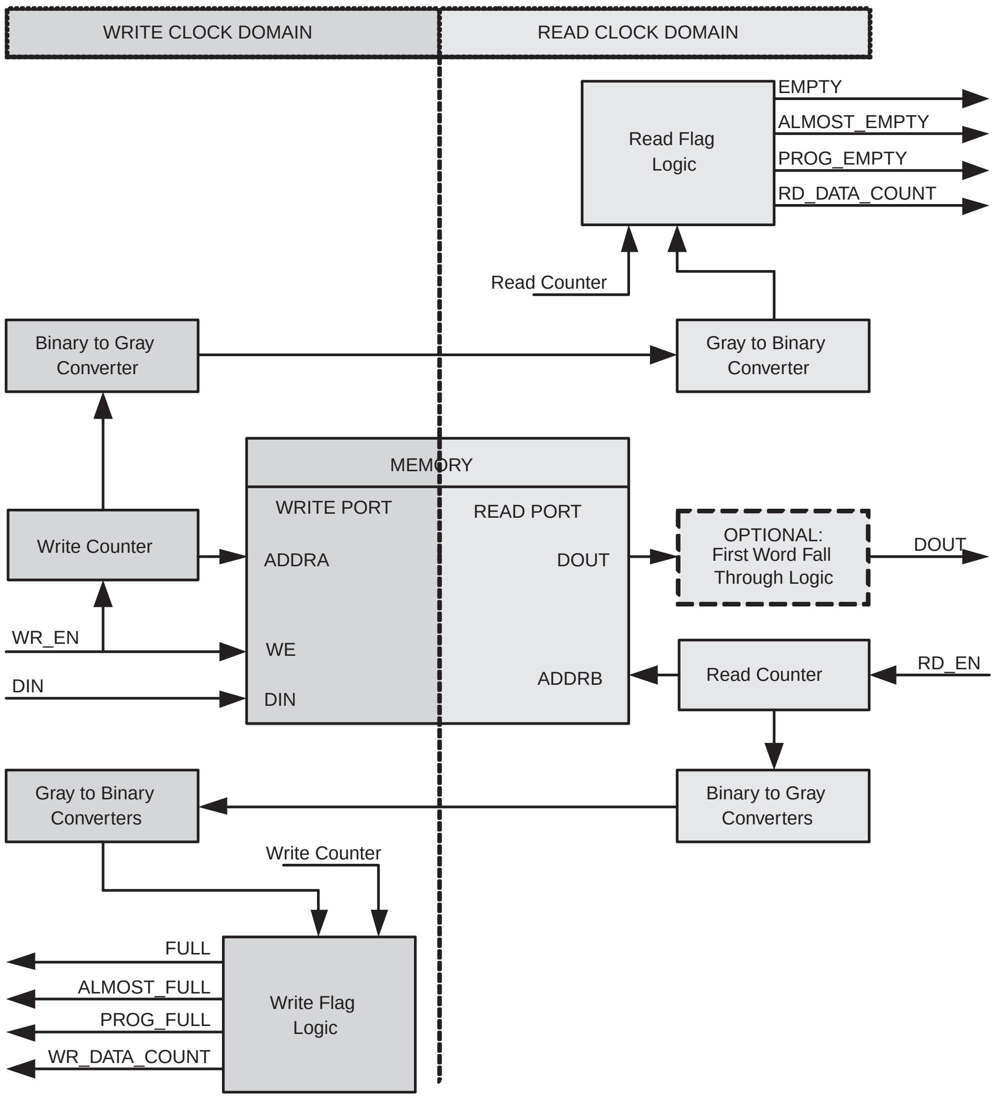


Figure 3-22: **Functional Implementation of a FIFO with Independent Clock Domains**

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

Table 3-5: **Interface Signals and Corresponding Clock Domains**

WR_CLK	RD_CLK
DIN	DOUT
WR_EN	RD_EN
FULL	EMPTY
ALMOST_FULL	ALMOST_EMPTY
PROG_FULL	PROG_EMPTY

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

WR_ACK	VALID
OVERFLOW	UNDERFLOW
WR_DATA_COUNT	RD_DATA_COUNT
WR_RST	SBITERR
INJECTSBITERR	DBITERR
INJECTDBITERR	RD_RST

For FIFO cores using independent clocks, the timing relationship between the write and read operations and the status flags is affected by the relationship of the two clocks. For example, the timing between writing to an empty FIFO and the deassertion of EMPTY is determined by the phase and frequency relationship between the write and read clocks. For additional information refer to the Synchronization Considerations, page 93.

Independent Clocks: Built-in FIFO

Figure 3-23 illustrates the functional implementation of FIFO configured with independent clocks using the Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA built-in FIFO primitive. This design implementation consists of cascaded built-in FIFO primitives and handshaking logic. The number of built-in primitives depends on the FIFO width and depth requested.

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

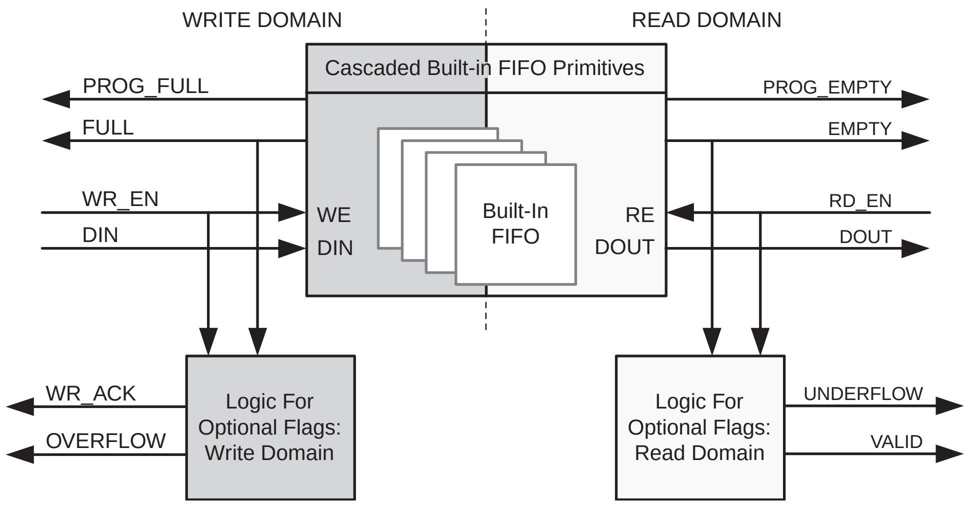


Figure 3-23: **Functional Implementation of Built-in FIFO**

This FIFO is designed to support an independent read clock (`RD_CLK`) and write clock (`WR_CLK`); in other words, there is no required relationship between `RD_CLK` and `WR_CLK` with regard to frequency or phase. Table 3-6 summarizes the FIFO interface signals, which are only valid in their respective clock domains.

Table 3-6: **Interface Signals and Corresponding Clock Domains**

WR_CLK	RD_CLK
DIN	DOUT
WR_EN	RD_EN
FULL	EMPTY
PROG_FULL	PROG_EMPTY
WR_ACK	VALID
OVERFLOW	UNDERFLOW
INJECTSBITERR	SBITERR
INJECTDBITERR	DBITERR

For FIFO cores using independent clocks, the timing relationship between the write and read operations and the status flags is affected by the relationship of the two clocks. For example, the timing between writing to an empty FIFO and the deassertion of EMPTY is determined by the phase and frequency relationship between the write and read clocks. For additional information, see Synchronization Considerations, page 93.

For Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA built-in FIFO configurations, the built-in ECC feature in the FIFO macro is provided. For more information, see "Built-in Error Correction Checking," page 118.

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 135.

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

Common Clock: Built-in FIFO

The FIFO Generator supports FIFO cores using the built-in FIFO primitive with a common clock. This provides users the ability to use the built-in FIFO, while requiring only a single clock interface. The behavior of the common clock configuration with built-in FIFO is identical to the independent clock configuration with built-in FIFO, except all operations are in relation to the common clock (CLK). See Independent Clocks: Built-in FIFO, page 122, for more information.

Common Clock FIFO: Block RAM and Distributed RAM

Figure 3-24 illustrates the functional implementation of a FIFO configured with a common clock using block RAM or distributed RAM for memory. All signals are synchronous to a single clock input (CLK). This design implements counters for write and read pointers and logic for calculating the status flags. An optional synchronous (SRST) or asynchronous (RST) reset signal is also available.

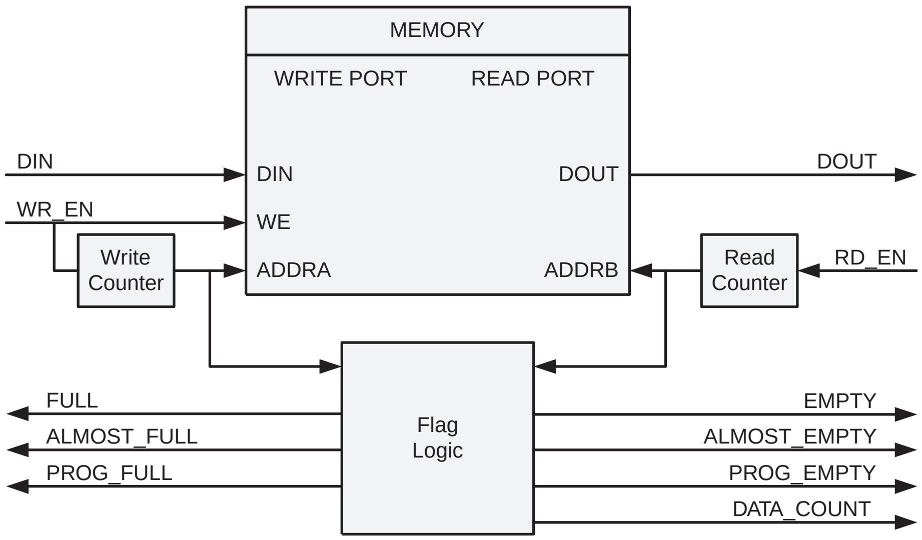


Figure 3-24: **Functional Implementation of a Common Clock FIFO using Block RAM or Distributed RAM**

Common Clock FIFO: Shift Registers

Figure 3-25 illustrates the functional implementation of a FIFO configured with a common clock using shift registers for memory. All operations are synchronous to the same clock input (CLK). This design implements a single up/down counter for both the write and read pointers and logic for calculating the status flags.

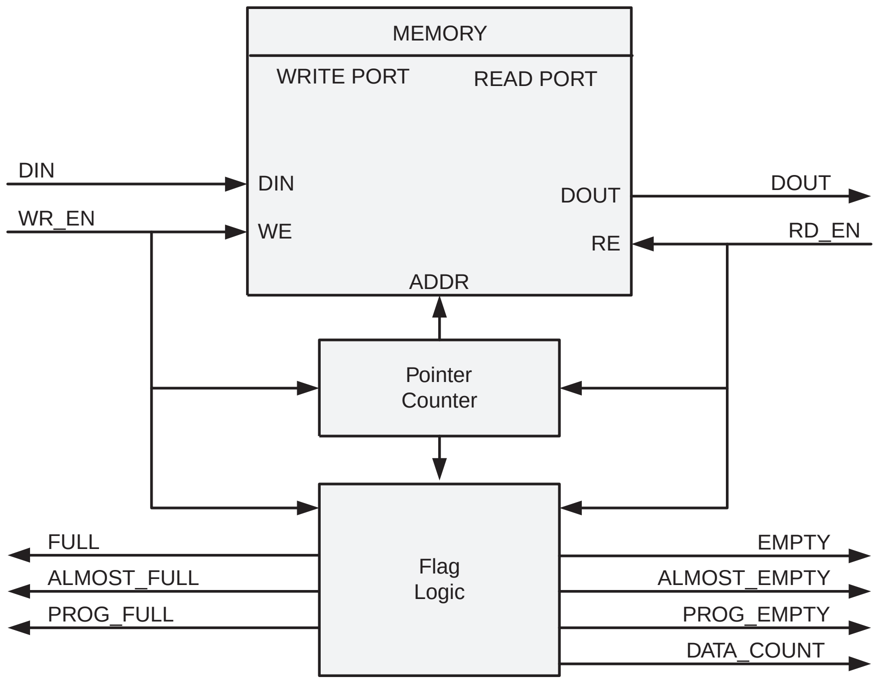


Figure 3-25: **Functional Implementation of a Common Clock FIFO using Shift Registers**

Resets

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

Asynchronous Reset (Enable Reset Synchronization Option is Selected)

The asynchronous reset (RST) input asynchronously resets all counters, output registers, and memories when asserted. When reset is implemented, it is synchronized internally to the core with each respective clock domain for setting the internal logic of the FIFO to a known state. This synchronization logic allows for proper timing of the reset logic within the core to avoid glitches and metastable behavior.

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

Table 3-7 defines the values of the output ports during power-up and reset state for block RAM, distributed RAM, and shift RAM FIFOs. Note that the underflow signal is dependent on RD_EN. If RD_EN is asserted and the FIFO is empty, underflow is asserted. The overflow signal is dependent on WR_EN. If WE_EN is asserted and the FIFO is full, overflow is asserted.

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

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

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

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

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

Full Flags Reset Value of 1

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

The FULL and ALMOST_FULL flags are asserted to ensure that no write operations occur when the FIFO core is in the reset state. After the FIFO exits the reset state and is ready for writing, the FULL and ALMOST_FULL flags deassert; this occurs approximately three clock cycles after the deassertion of asynchronous reset. See Figure 3-26 and Figure 3-27 for example behaviors. Note that the power-up values for this configuration are different from the reset state value.

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

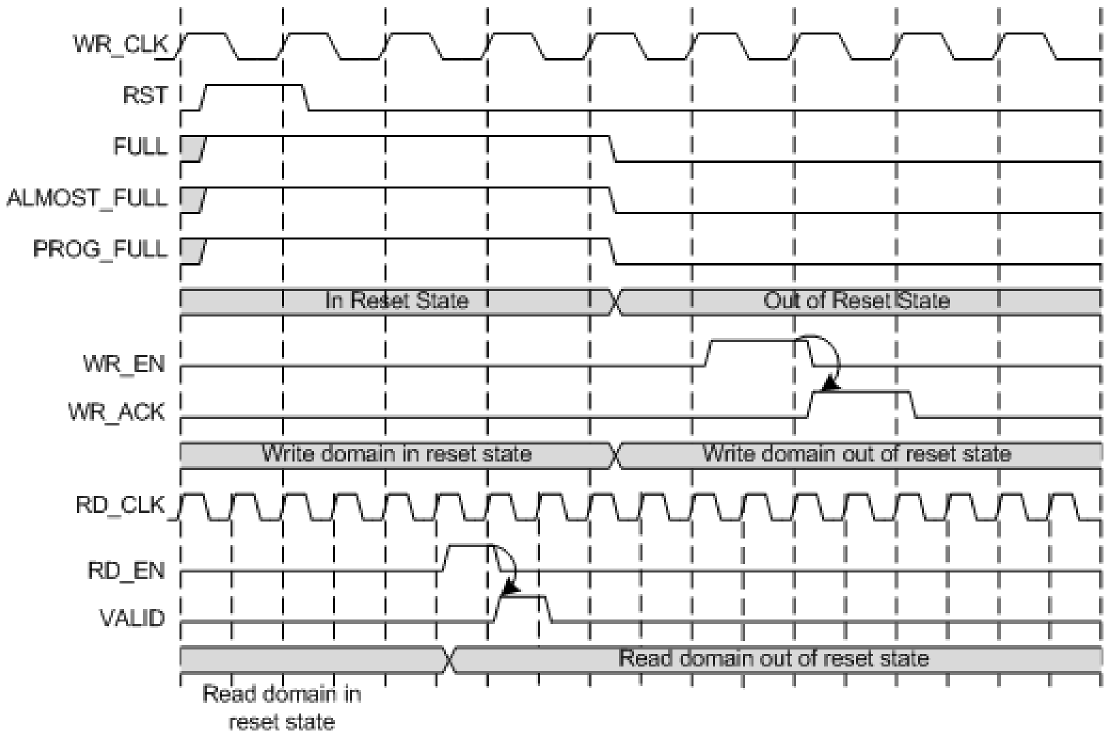

WR_CLK
RST
FULL
ALMOST_FULL
PROG_FULL
In Reset State
Out of Reset State
WR_EN
WR_ACK
Write domain in reset state
Write domain out of reset state
RD_CLK
RD_EN
VALID
Read domain out of reset state
Read domain in reset state

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

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

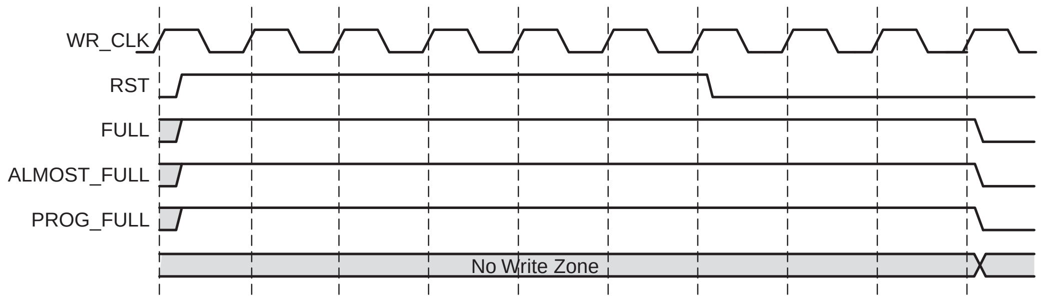


Figure 3-27: **Block RAM, Distributed RAM, Shift RAM with Full Flags Reset Value of 1 for the Reset Pulse of More Than One Clock**

Full Flags Reset Value of 0

In this configuration, the FIFO requires a minimum asynchronous reset pulse of one write clock cycle to complete the proper reset synchronization. At reset, FULL, ALMOST_FULL and PROG_FULL flags are deasserted. After the FIFO exits the reset synchronization state, the FIFO is ready for writing; this occurs approximately three clock cycles after the assertion of asynchronous reset. See Figure 3-28 for example behavior.

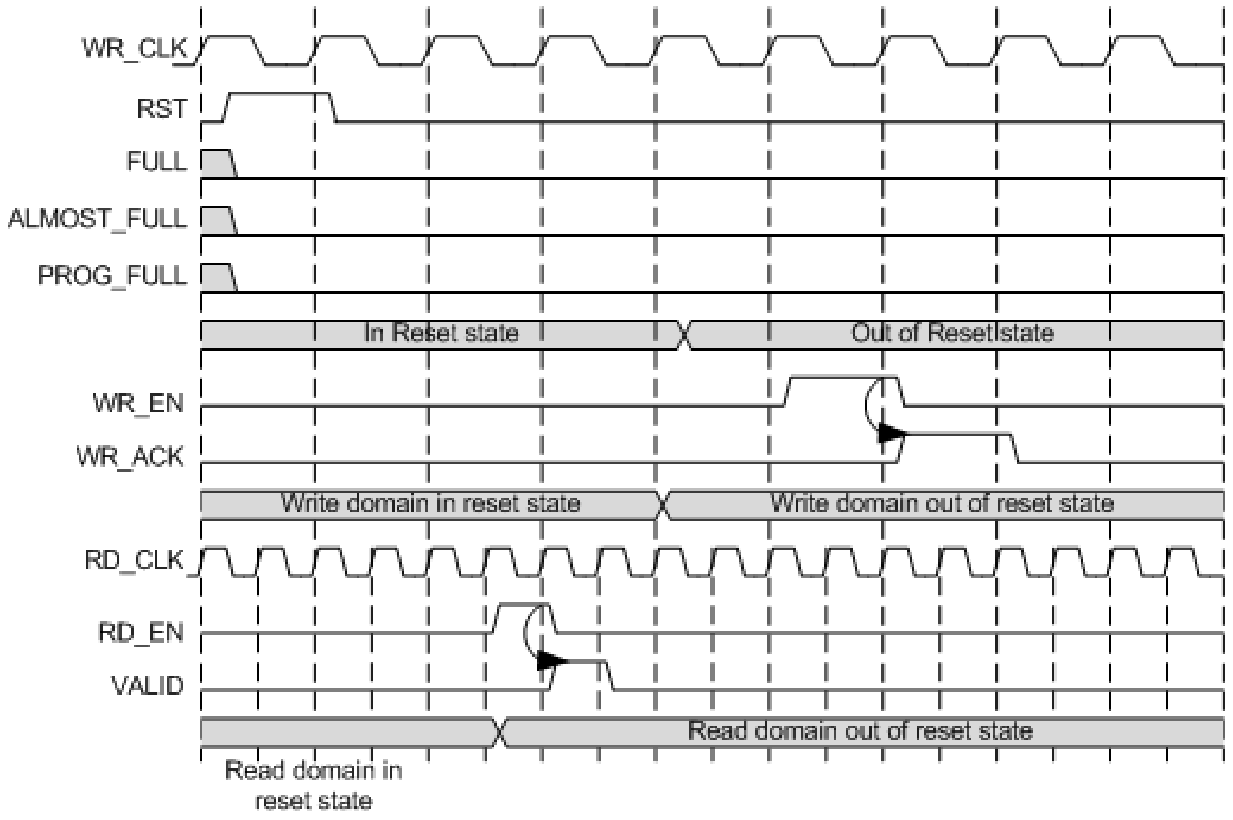

WR_CLK
RST
FULL
ALMOST_FULL
PROG_FULL
In Reset state
Out of Reset state
WR_EN
WR_ACK
Write domain in reset state
Write domain out of reset state
RD_CLK
RD_EN
VALID
Read domain out of reset state
Read domain in reset state

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

Common/Independent Clock: Built-in

Table 3-7 defines the values of the output ports during power-up and reset state for Built-in FIFOs. The DOUT reset value is supported only for Kintex-7, Virtex-7 and Virtex-6 common clock Built-In FIFOs with the embedded register option selected. The Kintex-7 and Virtex-7 FPGA Built-In FIFOs require an asynchronous reset pulse of at least 5 read and write clock cycles. To be consistent across all built-in FIFO configurations, it is recommended to give an asynchronous reset pulse of at least 5 read and write clock cycles for Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA Built-in FIFOs. However, the FIFO Generator core has a built-in mechanism ensuring the reset pulse is high for five read and write clock cycles for all Built-in FIFOs.

During reset, the RD_EN and WR_EN ports are required to be deasserted (no read or write operation can be performed). Assertion of reset causes the FULL and PROG_FULL flags to deassert and EMPTY and PROG_EMPTY flags to assert. After asynchronous reset is released, the core exits the reset state and is ready for writing. See Figure 3-29 for example behavior.

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

Table 3-8: **Asynchronous Reset Values for Built-in FIFO**

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

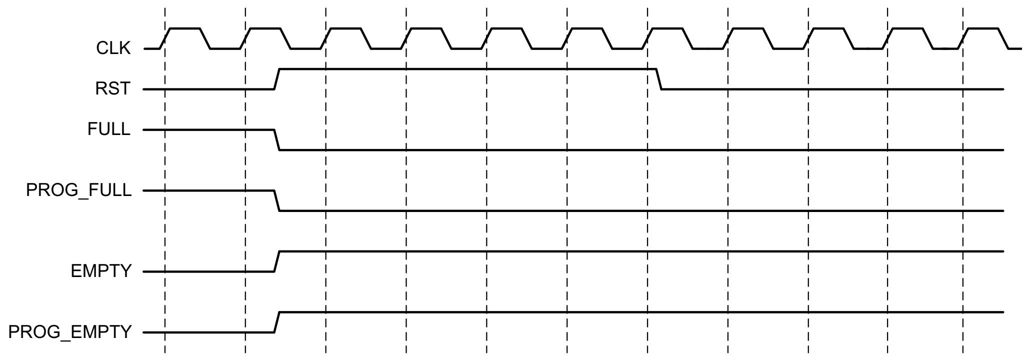


Figure 3-29: **Built-in FIFO, Asynchronous Reset Behavior**

Synchronous Reset

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

Common Clock Block, Distributed, or Shift RAM FIFOs

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

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

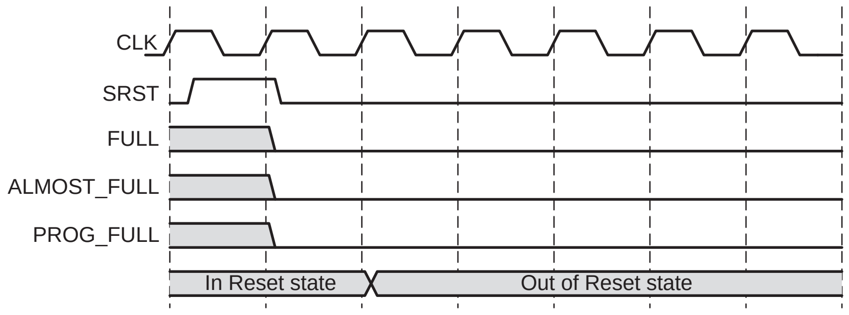


Figure 3-32: **Synchronous Reset: FIFO with a Common Clock**

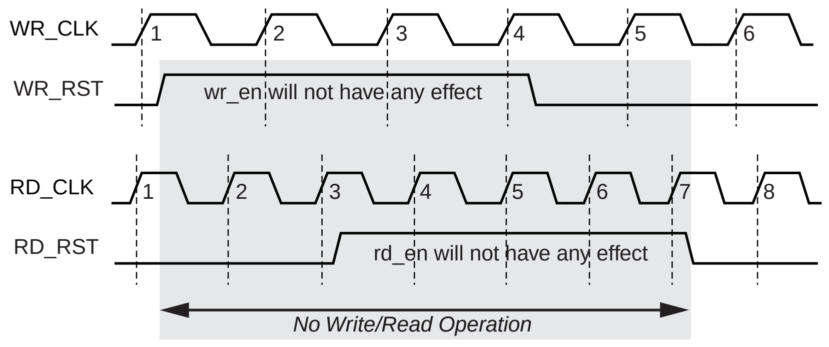
Independent Clock Block and Distributed RAM FIFOs (Enable Reset Synchronization Option not Selected)

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

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

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

Figure 3-33 and Figure 3-34 detail the resets.



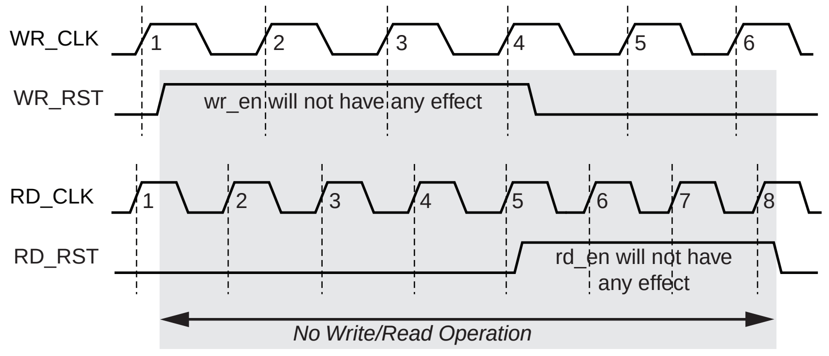
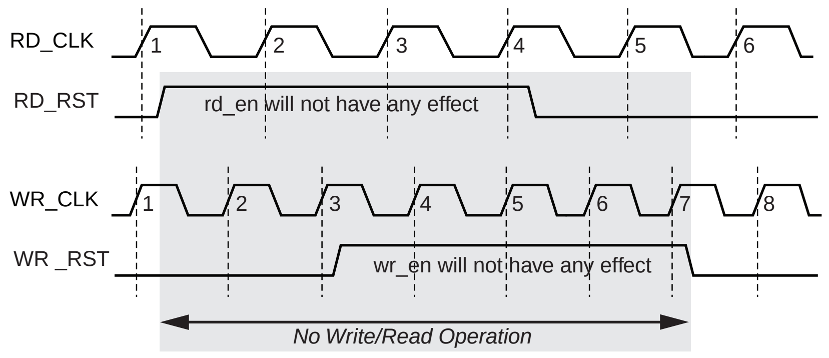


Figure 3-33: **Synchronous Reset: FIFO with Independent Clock - WR_RST then RD_RST**


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

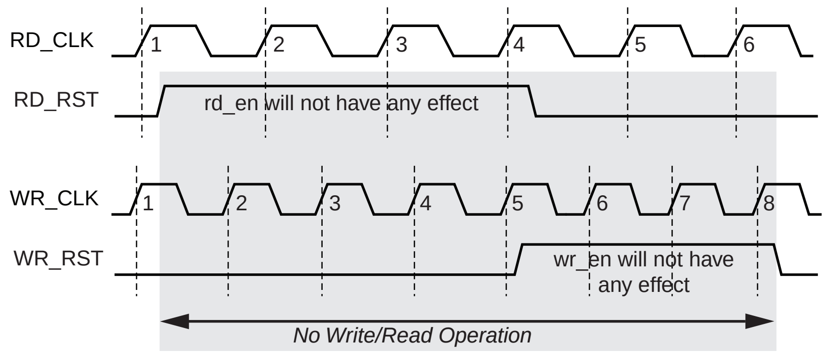

RD_CLK
RD_RST
rd_en will not have any effect
WR_CLK
WR_RST
wr_en will not have any effect
No Write/Read Operation

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

Table 3-9 defines the values of the output ports during power-up and the reset state. If you do not specify a DOUT reset value, it defaults to 0. The FIFO requires a reset pulse of only 1 clock cycle. The FIFOs are available for transaction on the clock cycle after the reset is released. The power-up values for the synchronous reset are the same as the reset state.

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

Table 3-9: **Synchronous Reset and Power-up Values**

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

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

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

Actual FIFO Depth

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

Block RAM, Distributed RAM and Shift RAM FIFOs

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

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

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

- Common Clock FIFO in Standard Read Mode

  `actual_write_depth = gui_write_depth`

  `actual_read_depth = gui_read_depth`

- Common Clock FIFO in FWFT Read Mode

  `actual_write_depth = gui_write_depth +2`

  `actual_read_depth = gui_read_depth +2`

- Independent Clock FIFO in Standard Read Mode

  `actual_write_depth = gui_write_depth - 1`

  `actual_read_depth = gui_read_depth - 1`

- Independent Clock FIFO in FWFT Read Mode

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

  actual_read_depth = gui_read_depth + 1

Notes

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

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

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

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

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

- Independent Clock FIFO in Standard Read Mode

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

- Independent Clock FIFO in FWFT Read Mode

actual_write_depth = (primitive_depth+2)*N

- Common Clock FIFO in Standard Read Mode

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

- Common Clock FIFO in FWFT Read Mode

actual_write_depth = (primitive_depth+1)*N

Notes

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

Virtex-4 FPGA Built-In FIFOs

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

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

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

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

actual_write_depth = primitive_depth+17

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

actual_write_depth = primitive_depth+17

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

Latency

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

Note: Latency is defined as the number of clock edges after a read or write operation occur before the signal is updated. Example: if latency is 0, that means that the signal is updated at the clock edge in which the operation occurred, as shown in Figure 3-35 in which `WR_ACK` is getting updated in which `WR_EN` is high.

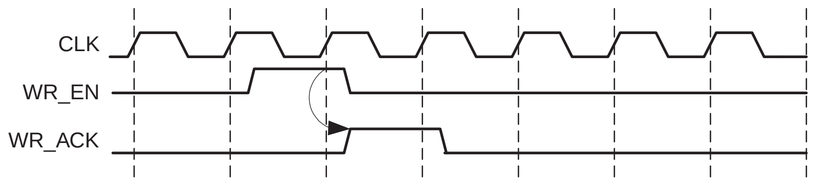


Figure 3-35: **Latency 0 Timing**

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

Table 3-10 defines the write port flags update latency due to a write operation for non-Built-in FIFOs such as block RAM, distributed RAM, and shift RAM FIFOs.

Table 3-10: **Non-Built-in FIFOs, Common Clock and Standard Read Mode Implementations: Write Port Flags Update Latency Due to Write Operation**

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

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

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

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

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

Table 3-12: **Non-Built-in FIFOs, Common Clock and Standard Read Mode Implementations: Write Port Flags Update Latency Due to Read Operation**

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

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

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

Table 3-13: **Non-Built-in FIFOs, Common Clock and Standard Read Mode Implementations: Read Port Flags Update Latency Due to Write Operation**

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

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

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

Table 3-14 defines the write port flags update latency due to a write operation for non-Built-in FIFOs such as block RAM, distributed RAM, and shift RAM FIFOs.

Table 3-14: **Non-Built-in FIFOs, Common Clock and FWFT Read Mode Implementations: Write Port Flags Update Latency due to Write Operation**

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

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

Table 3-15: **Non-Built-in FIFOs, Common Clock and FWFT Read Mode Implementations: Read Port Flags Update Latency due to Read Operation**

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

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

Table 3-16: **Non-Built-in FIFOs, Common Clock and FWFT Read Mode Implementations: Write Port Flags Update Latency Due to Read Operation**

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

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

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

Table 3-17: **Non-Built-in FIFOs, Common Clock and FWFT Read Mode Implementations: Read Port Flags Update Latency Due to Write Operation**

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

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

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

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

Table 3-18: **Non-Built-in FIFOs, Independent Clock and Standard Read Mode Implementations: Write Port Flags Update Latency Due to a Write Operation**

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

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

Table 3-19: **Non-Built-in FIFOs, Independent Clock and Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

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

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

Table 3-20: **Non-Built-in FIFOs, Independent Clock and Standard Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

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

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

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

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

Table 3-21: **Non-Built-in FIFOs, Independent Clock and Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

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

Note: Read handshaking signals only impacted by read operation.

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

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

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

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

Table 3-22: **Non-Built-in FIFOs, Independent Clock and FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Write Operation**

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

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

Table 3-23: **Non-Built-in FIFOs, Independent Clock and FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

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

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

Table 3-24: **Non-Built-in FIFOs, Independent Clock and FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

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

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

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

Table 3-25: **Non-Built-in FIFOs, Independent Clock and FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

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

Note: Read handshaking signals only impacted by read operation.

a. The crossing clock domain logic in independent clock FIFOs introduces a 1 RD_CLK uncertainty to the latency calculation.
b. Read handshaking signals are only impacted by a read operation.
c. This latency is the worst-case latency. The addition of the [2 RD_CLK (+1 RD_CLK)] latency depends on the status of the EMPTY and ALMOST_EMPTY flags.

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

Note: N is the number of primitives cascaded in depth. This can be calculated by dividing the GUI depth by the primitive depth. For ECC, the primitive depth is 512. The term "Built-in FIFOs" refers to the hard FIFO macros of Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs.

For more details for the write and read port flags update latency for a single primitive, see UG363, *Virtex-6 FPGA Memory Resources User Guide*, and UG473, *7 Series FPGAs Memory Resources User Guide*.

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

Table 3-26: **Common Clock Built-in FIFOs with Standard Read Mode Implementations: Write Port Flags Update Latency Due to Write Operation**

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

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

Table 3-27: **Common Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to Read Operation**

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

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

Table 3-28: **Common Clock Built-in FIFOs with Standard Read Mode Implementations: Write Port Flags Update Latency Due to Read Operation**

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

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

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

Table 3-29: **Common Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to Write Operation**

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

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

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

Note: N is the number of primitives cascaded in depth. This can be calculated by dividing the GUI depth by the primitive depth. For ECC, the primitive depth is 512. The term "Built-in FIFOs" refers to the hard FIFO macros of Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs.

For more details for the write and read port flags update latency for a single primitive, see UG363, *Virtex-6 FPGA Memory Resources User Guide*, and UG473, *7 Series FPGAs Memory Resources User Guide*.

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

Table 3-30: **Common Clock Built-in FIFOs with FWFT Read Mode Implementations: Write Port Flags Update Latency Due to Write Operation**

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

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

Table 3-31: **Common Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

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

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

Table 3-32: **Common Clock Built-in FIFOs with FWFT Read Mode Implementations: Write Port Flags Update Latency Due to a Read Operation**

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

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

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

Table 3-33: **Common Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Write Operation**

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

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

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

Note: N is the number of primitives cascaded in depth. This can be calculated by dividing the GUI depth by the primitive. For ECC, the primitive depth is 512. `Faster_Clk` is the clock domain, either `RD_CLK` or `WR_CLK`, that has a larger frequency. The term "Built-in FIFOs" refers to the hard FIFO macros of Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs.

For more details for the write and read port flags update latency for a single primitive, see UG363, *Virtex-6 FPGA Memory Resources User Guide*, and UG473, *7 Series FPGAs Memory Resources User Guide*.

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

Table 3-34: **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Write Port Flags Update Latency Due to a Write Operation**

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

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

Table 3-35: **Independent Clock Built-in FIFOs with Standard Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

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

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

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

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

a. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
b. L1 = 2 for Virtex-5 and Virtex-6, and L1 = 1 for 7 series devices.
c. L2 = 5 for Virtex-5, Virtex-6, and L2 = 4 for 7 series devices.
d. L3 = 3 for Virtex-5, Virtex-6 and 7 series devices.
e. L4 = 1 for Virtex-5, Virtex-6 and 7 series devices.
f. L5 = 3 for Virtex-5, Virtex-6, and L5 = 4 for 7 series devices.
g. Write handshaking signals are only impacted by a Write operation.

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

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

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

a. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
b. L1 = 2 for Virtex-5 and Virtex-6, and L1 = 1 for 7 series devices.
c. L2 = 4 for Virtex-5, Virtex-6 and 7 series devices.
d. L3 = 4 for Virtex-5, Virtex-6 and 7 series devices.
e. L4 = 1 for Virtex-5, Virtex-6 and 7 series devices.
f. L5 = 4 for Virtex-5, Virtex-6; and L5 = 5 for 7 series devices.
g. L6 = 3 for Virtex-5, Virtex-6; and L6 = 4 for 7 series devices.
h. Read handshaking signals are only impacted by a Read operation.

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

Note: N is the number of primitives cascaded in depth, which can be calculated by dividing the GUI depth by the primitive depth. For ECC, the primitive depth is 512. `Faster_Clk` is the clock domain, either `RD_CLK` or `WR_CLK`, that has a larger frequency. The term "Built-in FIFOs" refers to the hard FIFO macros of Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGAs.

For more details for the write and read port flags update latency for a single primitive, see UG363, *Virtex-6 FPGA Memory Resources User Guide*, and UG473, *7 Series FPGAs Memory Resources User Guide*.

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

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

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

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

Table 3-39: **Independent Clock Built-in FIFOs with FWFT Read Mode Implementations: Read Port Flags Update Latency Due to a Read Operation**

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

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

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

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

a. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
b. L1 = 2 for Virtex-5 and Virtex-6, and L1 = 1 for 7 series devices.
c. L2 = 5 for Virtex-5, Virtex-6, and L2 = 4 for 7 series devices.

d. L3 = 3 for Virtex-5, Virtex-6 and 7 series devices.
e. L4 = 1 for Virtex-5, Virtex-6 and 7 series devices.
f. L5 = 3 for Virtex-5, Virtex-6, and L5 = 4 for 7 series devices.
g. Write handshaking signals are only impacted by a Write operation.

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

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

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

a. Depending on the offset between read and write clock edges, the Empty and Full flags can deassert one cycle later.
b. L1 = 2 for Virtex-5 and Virtex-6, and L1 = 1 for 7 series devices.
c. L2 = 5 for Virtex-5, Virtex-6 and 7 series devices.
d. L3 = 4 for Virtex-5, Virtex-6 and 7 series devices.
e. L4 = 1 for Virtex-5, Virtex-6 and 7 series devices.
f. L5 = 4 for Virtex-5, Virtex-6, and L5 = 5 for 7 series devices.
g. L6 = 3 for Virtex-5, Virtex-6, and L6 = 4 for 7 series devices.
h. Read handshaking signals are only impacted by a Read operation.

Virtex-4 FPGA Built-in FIFO

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

Special Design Considerations

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

Resetting the FIFO

The FIFO Generator must be reset after the FPGA is configured and before operation begins. Two reset pins are available, asynchronous (RST) and synchronous (SRST), and both clear the internal counters and output registers.

- For asynchronous reset, internal to the core, RST is synchronized to the clock domain in which it is used, to ensure that the FIFO initializes to a known state. This synchronization logic allows for proper reset timing of the core logic, avoiding glitches and metastable behavior. To avoid unexpected behavior, it is not recommended to drive/toggle `WR_EN/RD_EN` when RST is asserted/high.
- For common clock block and distributed RAM synchronous reset, because the reset pin is synchronous to the input clock and there is only one clock domain in the FIFO, no additional synchronization logic is needed.
- For independent clock block and distributed RAM synchronous reset, because the reset pin (`WR_RST/RD_RST`) is synchronous to the respective clock domain, no additional synchronization logic is needed. However, it is recommended to follow these rules to avoid unexpected behavior:
    - If `WR_RST` is applied, then `RD_RST` must also be applied and vice versa.
    - No write or read operations should be performed until both clock domains are reset.

The generated FIFO core will be initialized after reset to a known state. For details about reset values and behavior, see Resets in Chapter 3 of this guide.

Continuous Clocks

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

Pessimistic Full and Empty

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

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

The EMPTY flags are considered pessimistic flags because they assume that no write operations have taken place in the write clock domain. ALMOST_EMPTY is guaranteed to be asserted on the rising edge of RD_CLK when there is only one more word in the FIFO, and EMPTY is guaranteed to be asserted on the rising edge of RD_CLK when the FIFO is empty. There may be a number of clock cycles between a write operation and the deassertion of EMPTY. The precise number of clock cycles for EMPTY to deassert is not predictable due to the crossing of clock domains and synchronization logic. For more information see Simultaneous Assertion of Full and Empty Flag

See Chapter 3, "Designing with the Core," for detailed information about the latency and behavior of the full and empty flags.

Programmable Full and Empty

The programmable full (PROG_FULL) and programmable empty (PROG_EMPTY) flags provide the user flexibility in specifying when the programmable flags assert and deassert. These flags can be set either by constant value(s) or by input port(s). These signals differ from the full and empty flags because they assert one (or more) clock cycle *after* the assert threshold has been reached. These signals are deasserted some time after the negate threshold has been passed. In this way, PROG_EMPTY and PROG_FULL are also considered pessimistic flags. See Programmable Flags in Chapter 3 of this guide for more information about the latency and behavior of the programmable flags.

Simultaneous Assertion of Full and Empty Flag

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

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

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

For example, assume the following configurations:

- Independent clock (non built-in), standard FIFO
- write clock frequency = 3MHz, wr_clk_period = 333 ns
- read clock frequency = 148 MHz, rd_clk_period = 6.75 ns
- write depth = read depth = 20
- actual_wr_depth = actual_rd_depth = 19 (as mentioned in Actual FIFO Depth in Chapter 3)

Apply equation A:

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

1*6.75 + 5*333 < 19*6.75

1671.75 ns < 128.5 ns --> Equation VIOLATED!

Note: Left side equation is the latency of full flag updating due to read operation as mentioned in Table 3-20, page 139.

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

Apply Equation B:

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

1*wr_clk_period + 5*rd_clk_period < actual_wr_depth*wr_clk_period

1*333 + 5*6.75 < 19*333

366.75 ns < 6327 ns --> Equation MET!

Note: Left side equation is the latency of empty flag updating due to write operation as mentioned in Table 3-21, page 140.

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

Write Data Count and Read Data Count

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

Consider the following when using the WR_DATA_COUNT or RD_DATA_COUNT ports.

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

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

See Data Counts in Chapter 3 of this guide for details about the latency and behavior of the data count flags.

Setup and Hold Time Violations

When generating a FIFO with independent clock domains (whether a DCM is used to derive the write/read clocks or not), the core internally synchronizes the write and read clock domains. For this reason, setup and hold time violations are expected on certain registers within the core. In simulation, warning messages may be issued indicating these violations. If these warning messages are from the FIFO Generator core, they can be safely ignored. The core is designed to properly handle these conditions, regardless of the phase or frequency relationship between the write and read clocks.

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

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

SECTION II: VIVADO DESIGN SUITE

Customizing and Generating the Native Core

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

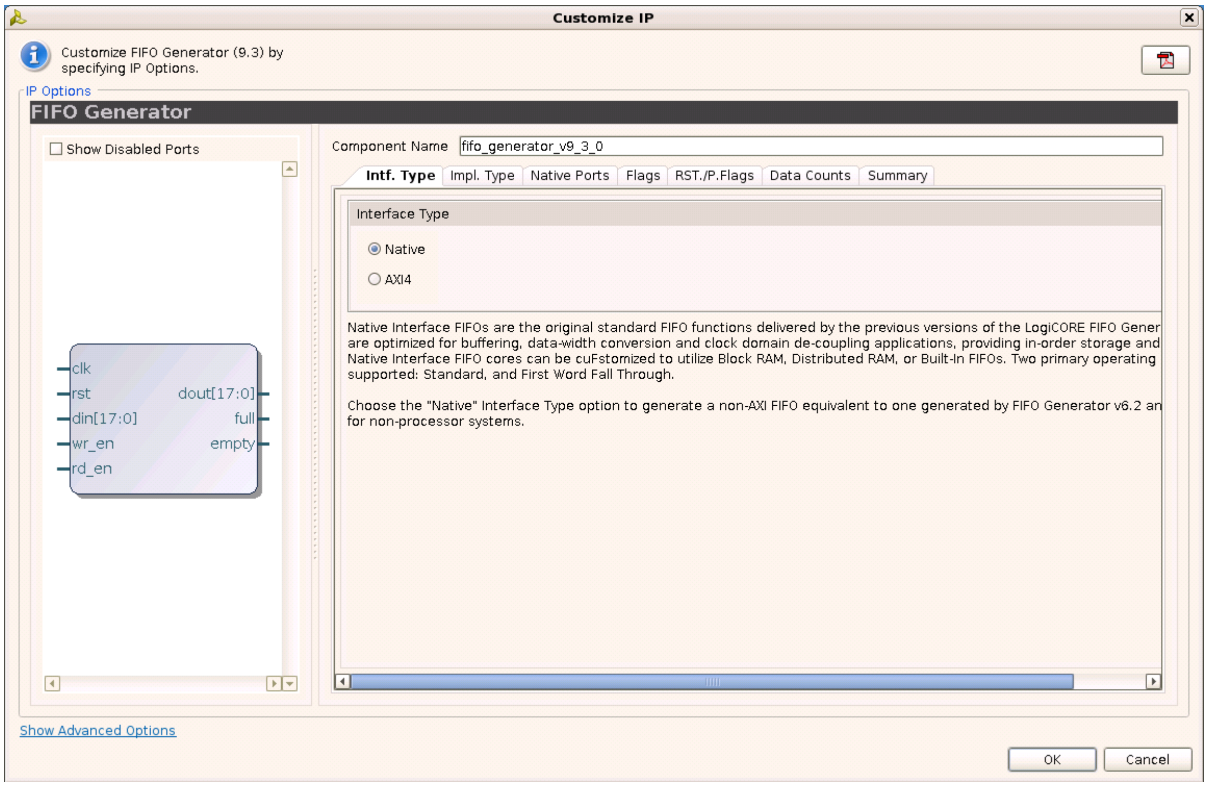
GUI

The Native FIFO Interface GUI includes seven configuration screens.

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

Interface Type

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


Customize IP

Customize FIFO Generator (9.3) by specifying IP Options.

IP Options

FIFO Generator

Show Disabled Ports

Component Name fifo_generator_v9_3_0

Interface Type

[✓ Native]

AXI4

Native Interface FIFOs are the original standard FIFO functions delivered by the previous versions of the LogiCORE FIFO Gener
are optimized for buffering, data-width conversion and clock domain de-coupling applications, providing in-order storage and
Native Interface FIFO cores can be cuFstomized to utilize Block RAM, Distributed RAM, or Built-In FIFOs. Two primary operating
supported: Standard, and First Word Fall Through.

Choose the "Native" Interface Type option to generate a non-AXI FIFO equivalent to one generated by FIFO Generator v6.2 an
for non-processor systems.

Figure 5-1: **Main FIFO Generator Screen**

- **Component Name**

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

- **Interface Type**
  - **Native**

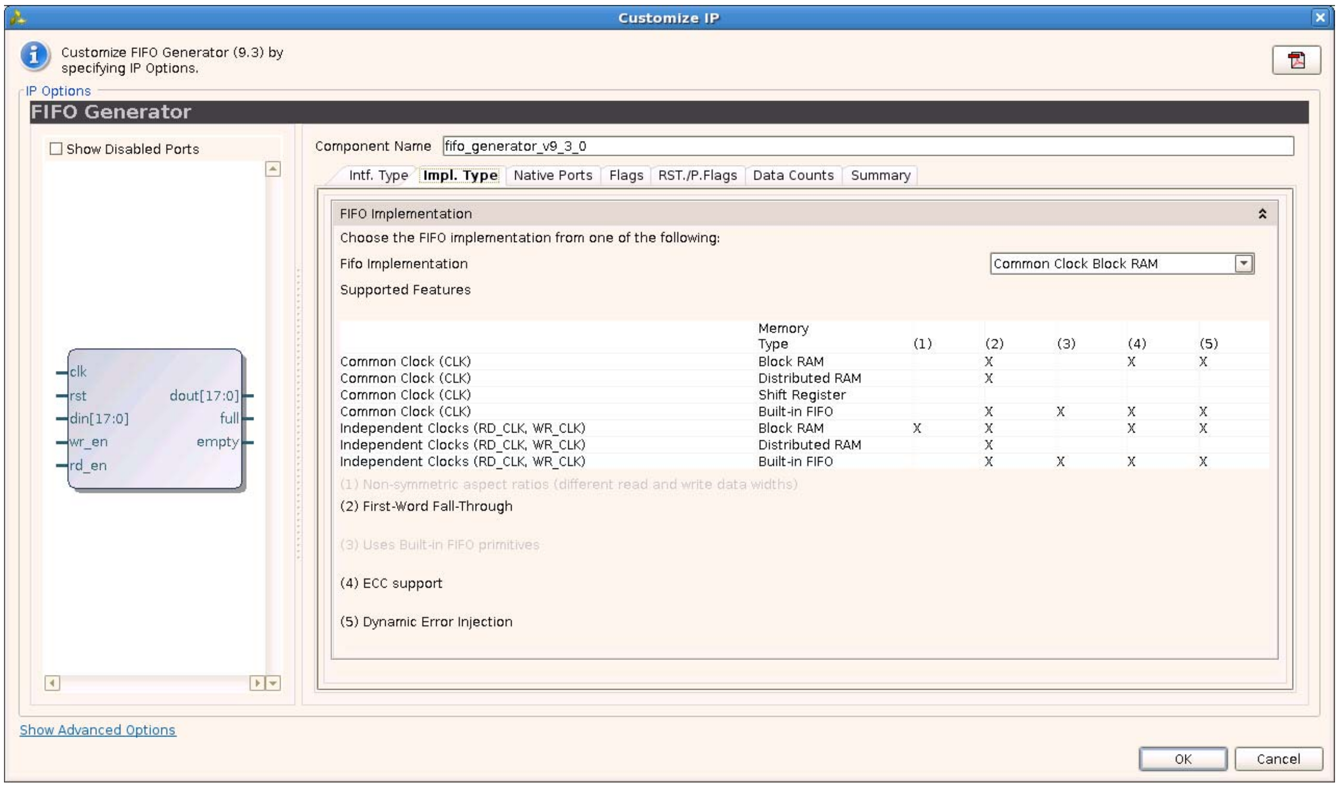
    Implements a Native FIFO.

  - **AXI4**

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

FIFO Implementation

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



Component Name fifo_generator_v9_3_0

FIFO Implementation

Choose the FIFO implementation from one of the following:

Fifo Implementation: Common Clock Block RAM

Supported Features

	Memory Type	(1)	(2)	(3)	(4)	(5)
Common Clock (CLK)	Block RAM		X		X	X
Common Clock (CLK)	Distributed RAM		X			
Common Clock (CLK)	Shift Register					
Common Clock (CLK)	Built-in FIFO		X	X	X	X
Independent Clocks (RD_CLK, WR_CLK)	Block RAM	X	X		X	X
Independent Clocks (RD_CLK, WR_CLK)	Distributed RAM		X			
Independent Clocks (RD_CLK, WR_CLK)	Built-in FIFO		X	X	X	X

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

(2) First-Word Fall-Through

(3) Uses Built-in FIFO primitives

(4) ECC support

(5) Dynamic Error Injection

Figure 5-2: **FIFO Implementation Screen**

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

- **Common Clock (CLK), Block RAM**

  For details, see Common Clock FIFO: Block RAM and Distributed RAM, page 123. This implementation optionally supports first-word-fall-through (selectable in the second GUI screen, shown in Figure 5-3).

- **Common Clock (CLK), Distributed RAM**

  For details, see Common Clock FIFO: Block RAM and Distributed RAM, page 123. This implementation optionally supports first-word-fall-through (selectable in the second GUI screen, shown in Figure 5-3).

- **Common Clock (CLK), Shift Register**

For details, see Common Clock FIFO: Shift Registers, page 124. This implementation is only available in Virtex-4 FPGA and newer architectures.

- **Common Clock (CLK), Built-in FIFO**

  For details, see Common Clock: Built-in FIFO, page 123. This implementation is only available when using the Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA architectures. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, shown in Figure 5-3).

- **Independent Clocks (RD_CLK, WR_CLK), Block RAM**

  For details, see Independent Clocks: Block RAM and Distributed RAM, page 120. This implementation optionally supports asymmetric read/write ports and first-word fall-through (selectable in the second GUI screen, shown in Figure 5-3).

- **Independent Clocks (RD_CLK, WR_CLK), Distributed RAM**

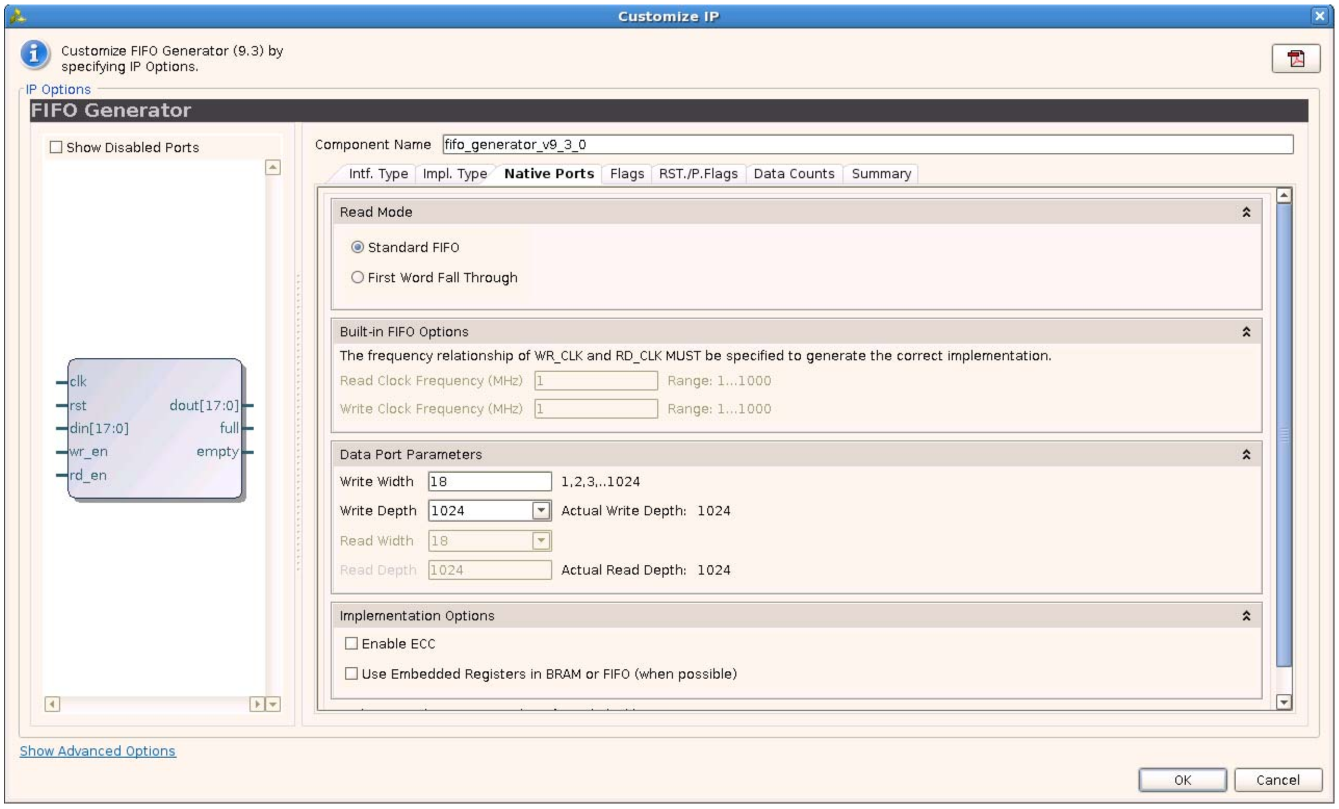
  For more information, see Independent Clocks: Block RAM and Distributed RAM, page 120. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, shown in Figure 5-3).

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

  For more information, see Independent Clocks: Built-in FIFO, page 122. This implementation is only available when using Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA architectures. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, shown in Figure 5-3).

Performance Options and Data Port Parameters

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



Component Name: fifo_generator_v9_3_0

Read Mode
- (●) Standard FIFO
- ( ) First Word Fall Through

Built-in FIFO Options

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

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

Data Port Parameters

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

Implementation Options
- [ ] Enable ECC
- [ ] Use Embedded Registers in BRAM or FIFO (when possible)

Figure 5-3: **Performance Options and Data Port Parameters Screen**

- **Read Mode**

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

  - **Standard FIFO**

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

  - **First-Word Fall-Through FIFO**

    Implements a FIFO with registered outputs. For more information about FWFT functionality, see First-Word Fall-Through FIFO Read Operation, page 98.

- **Built-in FIFO Options**

  - **Read/Write Clock Frequencies**

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

IMPORTANT: *It is critical that Read Clock and Write Clock Frequency data is entered and accurate. If this information is not provided, it can result in a sub-optimal solution with incorrect core behavior.*

- **Data Port Parameters**
  - **Write Width**

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

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

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

    Available only if independent clocks configuration with block RAM is selected. Valid range must comply with asymmetric port rules. See Non-symmetric Aspect Ratios, page 113.
  - **Read Depth**

    Automatically calculated based on Write Width, Write Depth, and Read Width.
- **Implementation Options**
  - **Error Correction Checking in Block RAM or Built-in FIFO**

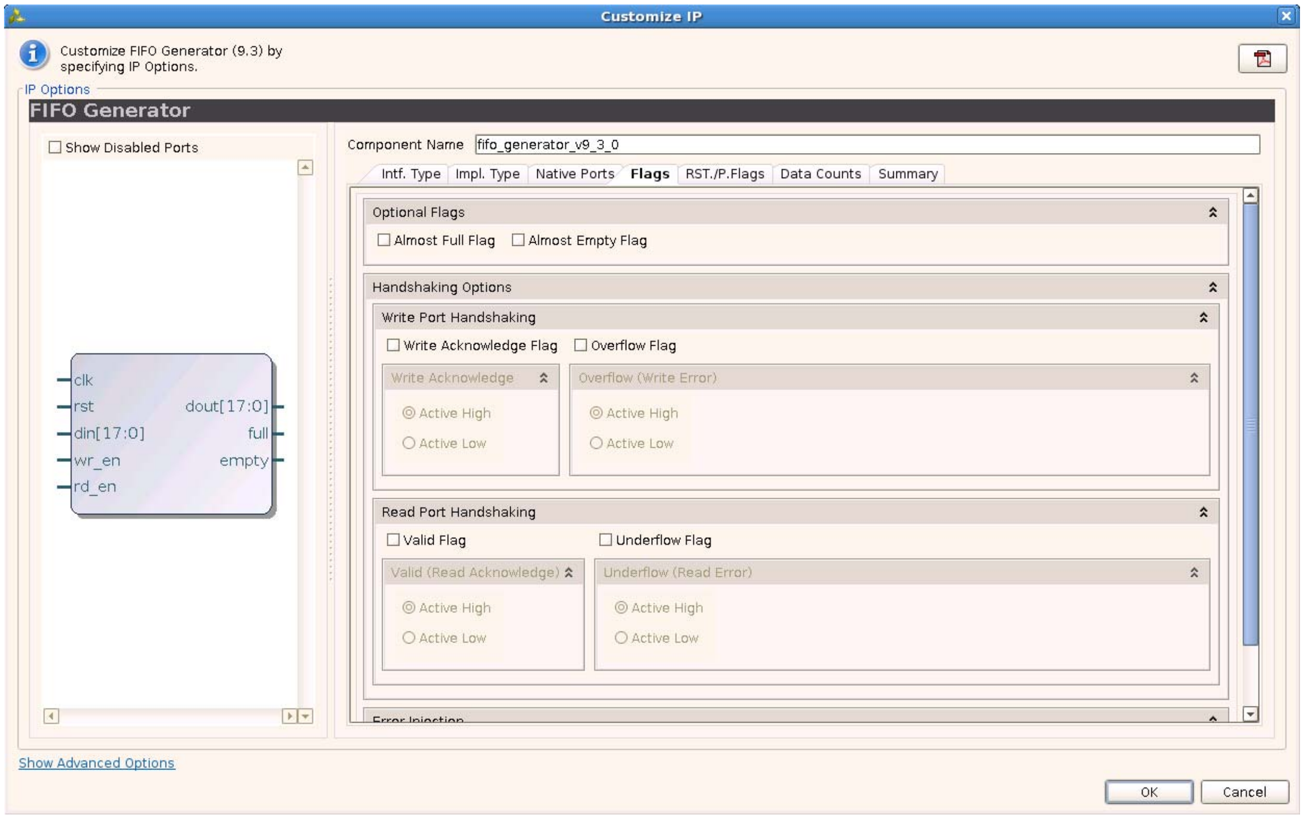
    The Error Correction Checking (ECC) feature enables built-in error correction in the Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA block RAM and built-in FIFO macros. When this feature is enabled, the block RAM or built-in FIFO is set to the full ECC mode, where both the encoder and decoder are enabled.

- **Use Embedded Registers in Block RAM or FIFO**

  The block RAM macros available in Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA, as well as built-in FIFO macros available in Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA, have built-in embedded registers that can be used to pipeline data and improve macro timing. This option enables users to add one pipeline stage to the output of the FIFO and take advantage of the available embedded registers; however, the ability to reset the data output of the Virtex-5 FPGA built-in FIFO is disabled when this feature is used. For built-in FIFOs, this feature is only supported for synchronous FIFO configurations that have only 1 FIFO macro in depth. See Embedded Registers in Block RAM and FIFO Macros (Zynq-7000, 7 Series, Virtex-6, Virtex-5 and Virtex-4 FPGAs), page 116.

Optional Flags, Handshaking, and Initialization

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



Component Name: fifo_generator_v9_3_0

Optional Flags
☐ Almost Full Flag ☐ Almost Empty Flag

Handshaking Options

Write Port Handshaking
☐ Write Acknowledge Flag ☐ Overflow Flag
Write Acknowledge: [✓ Active High] Active Low
Overflow (Write Error): [✓ Active High] Active Low

Read Port Handshaking
☐ Valid Flag ☐ Underflow Flag
Valid (Read Acknowledge): [✓ Active High] Active Low
Underflow (Read Error): [✓ Active High] Active Low

Figure 5-4: **Optional Flags, Handshaking, and Error Injection Options Screen**

- **Optional Flags**

  Refer to 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 Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA built-in FIFOs. Generates an output port that indicates the FIFO is almost full (only one more word can be written).

- **Almost Empty Flag**

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

- **Handshaking Options**

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

    - **Underflow (Read Error)**

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

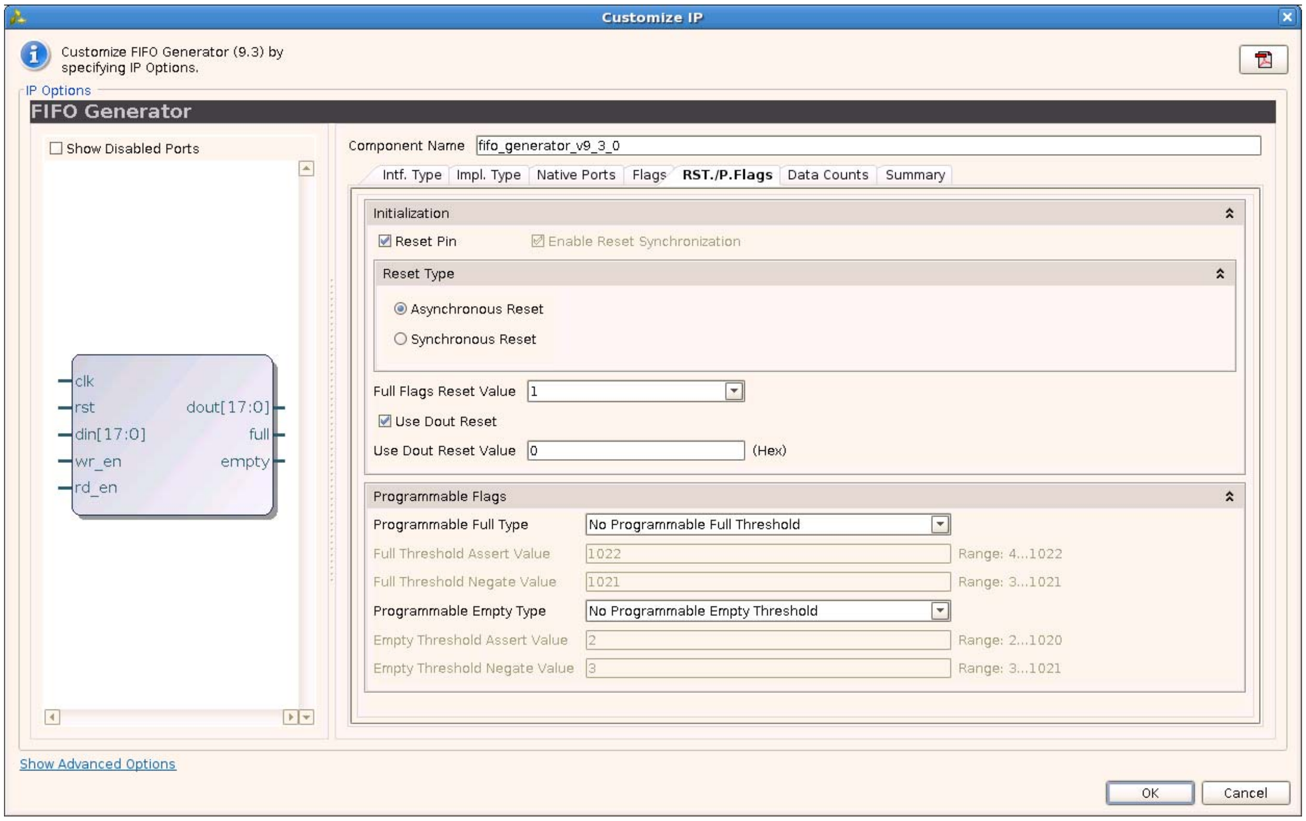
- **Error Injection**
  - **Single Bit Error Injection**

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

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

Initialization and Programmable Flags

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



Component Name: fifo_generator_v9_3_0

Initialization
- [✓] Reset Pin
- [✓] Enable Reset Synchronization

Reset Type
- (●) Asynchronous Reset
- ( ) Synchronous Reset

Full Flags Reset Value: 1
- [✓] Use Dout Reset

Use Dout Reset Value: 0 (Hex)

Programmable Flags

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

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

- **Initialization**
  - **Reset Pin**

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

    - **Reset Type**

      - **Enable Reset Synchronization**

        Optional selection only available for independent clock block RAM or distributed RAM FIFOs. When unchecked, WR_RST/RD_RST is available. See Resets in Chapter 3 for details.

      - **Asynchronous Reset**

        Optional selection for a common-clock FIFO implemented using distributed or block RAM.

- **Synchronous Reset**

  Optional selection for a a common-clock FIFO implemented using distributed or block RAM.

- **Full Flags Reset Value**

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

◦ **Use Dout Reset**

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

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

  - **Use Dout Reset Value**

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

• **Programmable Flags**

  Refer to Latency in Chapter 3 for the latency of the programmable flags due to write/ read operation.

  ◦ **Programmable Full Type**

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

    - **Full Threshold Assert Value**

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

- **Full Threshold Negate Value**

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

◦ **Programmable Empty Type**

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

- **Empty Threshold Assert Value**

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

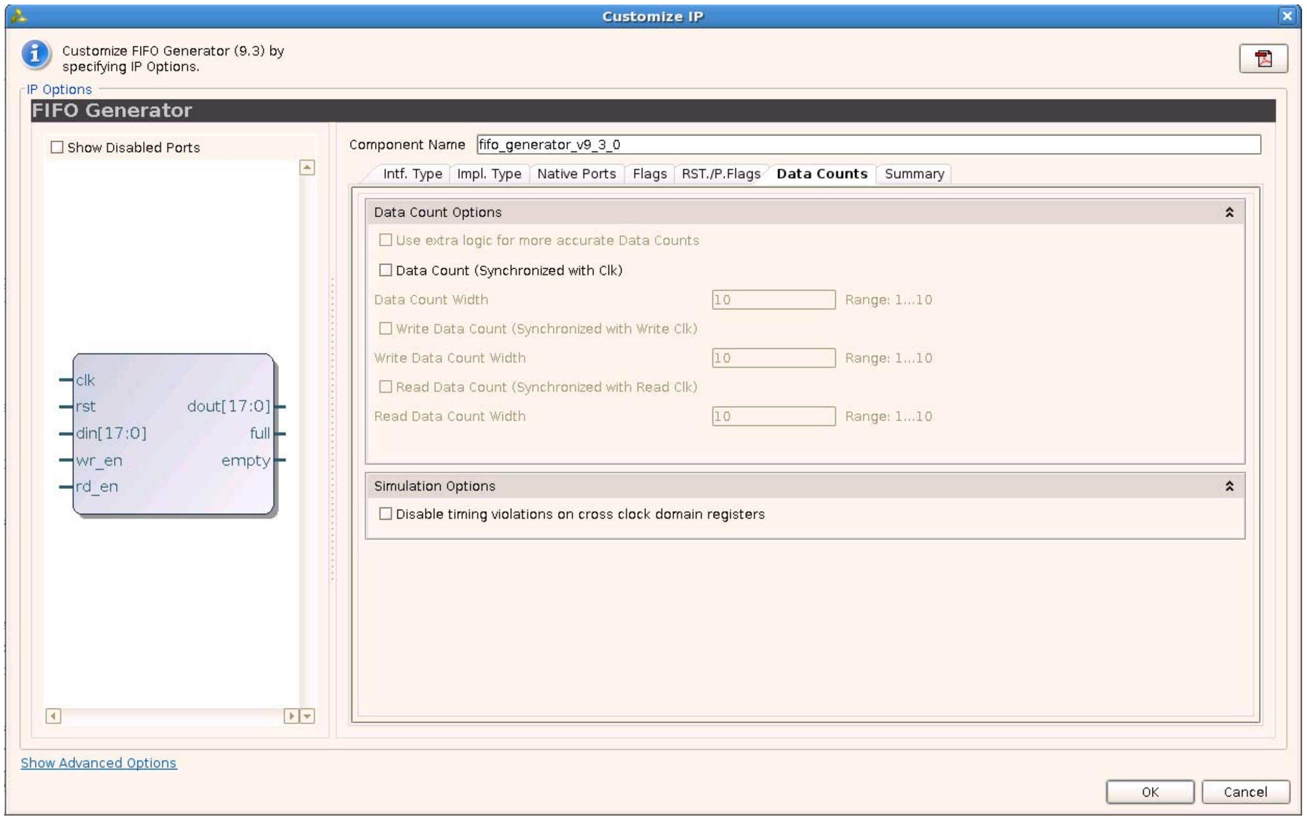
- **Empty Threshold Negate Value**

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

Data Count

Use this screen to set data count options.

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



Customize IP

Customize FIFO Generator (9.3) by specifying IP Options.

IP Options

FIFO Generator

☐ Show Disabled Ports

Component Name fifo_generator_v9_3_0

Intf. Type | Impl. Type | Native Ports | Flags | RST./P.Flags | Data Counts | Summary

Data Count Options

☐ Use extra logic for more accurate Data Counts

☐ Data Count (Synchronized with Clk)

Data Count Width 10 Range: 1...10

☐ Write Data Count (Synchronized with Write Clk)

Write Data Count Width 10 Range: 1...10

☐ Read Data Count (Synchronized with Read Clk)

Read Data Count Width 10 Range: 1...10

Simulation Options

☐ Disable timing violations on cross clock domain registers

Show Advanced Options

OK Cancel

Figure 5-6: **Data Count Screen**

- **Data Count Options**

  Refer to Latency in Chapter 3 for the latency of the data counts due to write/read operation.

  - **Use Extra Logic For More Accurate Data Counts**

    Only available for independent clocks FIFO with block RAM or distributed RAM, and when using first-word fall-through. This option uses additional external logic to generate a more accurate data count. This feature is always enabled for common clock FIFOs with block RAM or distributed RAM and when using first-word-fall-through. See First-Word Fall-Through Data Count, page 110 for details.

- **Data Count (Synchronized With Clk)**

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

  - **Data Count Width**

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

- **Write Data Count (Synchronized with Write Clk)**

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

  - **Write Data Count Width**

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

- **Read Data Count (Synchronized with Read Clk)**

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

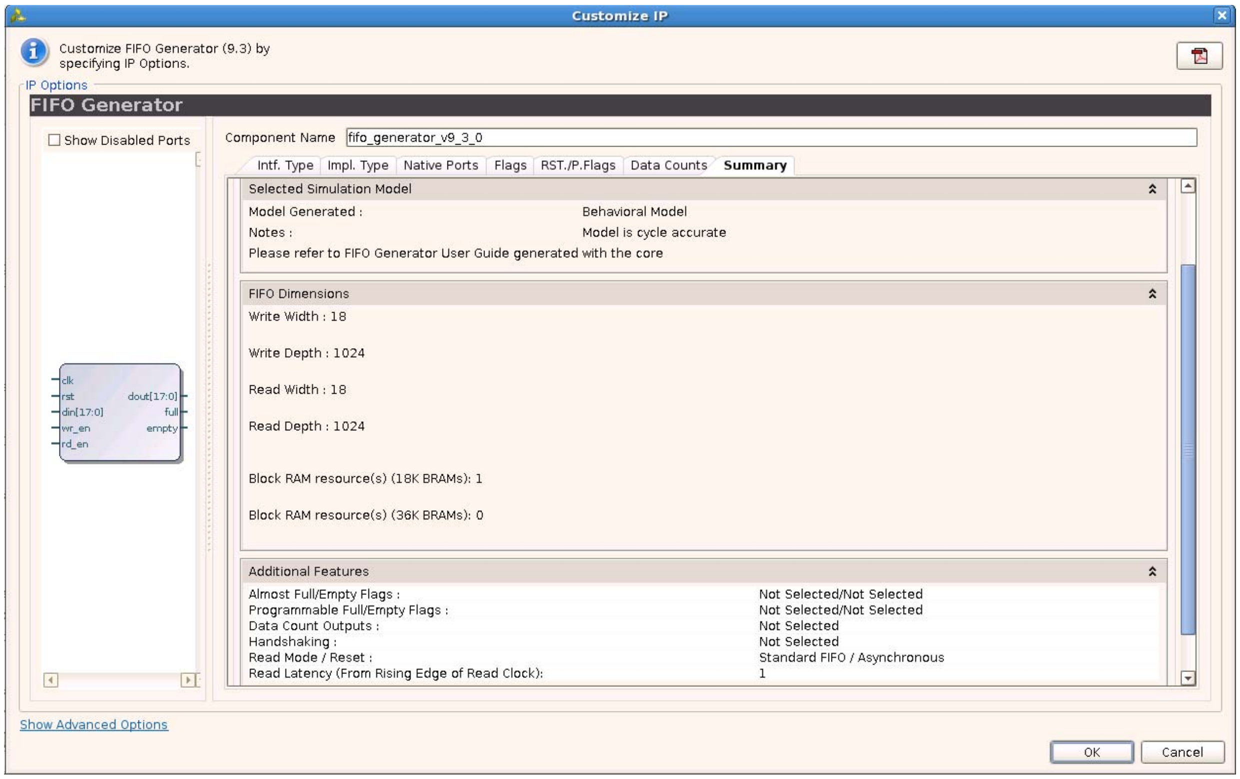
  - **Read Data Count Width**

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

Summary

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

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



Component Name fifo_generator_v9_3_0

Selected Simulation Model

Model Generated : Behavioral Model
Notes : Model is cycle accurate
Please refer to FIFO Generator User Guide generated with the core

FIFO Dimensions

Write Width : 18

Write Depth : 1024

Read Width : 18

Read Depth : 1024

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

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

Additional Features

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

Figure 5-7: **Summary Screen**

Output Generation

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

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

- <component name>/sim
  Contains the source file necessary to synthesize the FIFO Generator
- <component name>/example_design
  Contains the source file necessary to synthesize the example design
- <component name>/simulation
  Contains the source file necessary to simulate the example design
  - <component name>/simulation/functional
    Contains the scripts necessary to run functional simulation on the core including example design
  - <component name>/simulation/timing
    Contains the scripts necessary to run timing simulation on the core including example design

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

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

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

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

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

<component name>/synth

The synth directory contains the FIFO Generator synthesis file.

Table 5-2: **Synth Directory**

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

<component name>/sim

The sim directory contains the FIFO Generator simulation wrapper file.

Table 5-3: **Sim Directory**

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

<component name>/example_design

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

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

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

<component name>/simulation

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

Table 5-5: **Simulation Directory**

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

<component name>/simulation/functional

The functional directory contains the scripts to launch XSIM/MTI Simulation with simulation test bench set as top entity.

Table 5-6: **Functional Directory**

Name	Description
Simulate_xsim.sh	XSim macro file for Linux machines that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.
Simulate_xsim.bat	XSim macro file for Windows that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.
wave_xsim.tcl	XSim macro file that opens a Wave window with top-level signals.
simulate_mti.sh	Linux shell script that executes the ModelSim macro file.
simulate_mti.bat	Windows batch script that executes the ModelSim macro file.
simulate_mti.do	A ModelSim macro file that compiles the HDL sources and runs the simulation.

<component name>/simulation/timing

The timing directory contains the scripts to launch XSIM/MTI Simulation with simulation test bench set as top entity.

Table 5-7: **Timing Directory**

Name	Description
Simulate_xsim.sh	XSim macro file for Linux machines that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.
Simulate_xsim.bat	XSim macro file for Windows that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.
wave_xsim.tcl	XSim macro file that opens a Wave window with top-level signals.
simulate_mti.sh	Linux shell script that executes the ModelSim macro file.
simulate_mti.bat	Windows batch script that executes the ModelSim macro file.
simulate_mti.do	A ModelSim macro file that compiles the HDL sources and runs the simulation.

Customizing and Generating the AXI4 Core

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

GUI

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

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

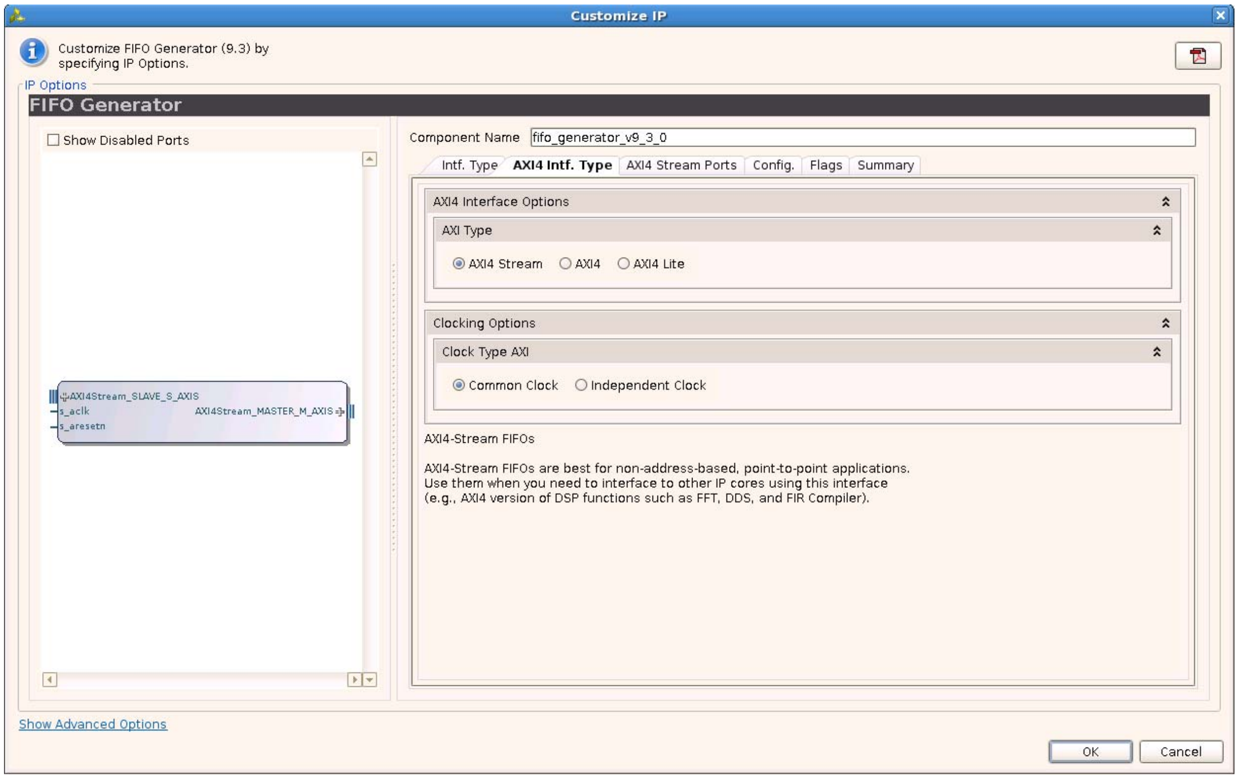
  For AXI4 and AXI4-Lite interfaces, FIFO Generator provides a separate page to configure each FIFO channel. For more details, see Easy Integration of Independent FIFOs for Read and Write Channels in Chapter 1.
- Summary

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

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

AXI4 Interface Selection

Figure 6-1 shows the AXI4 interface selection screen.



Component Name fifo_generator_v9_3_0

AXI4 Interface Options

AXI Type

[✓ AXI4 Stream] AXI4 AXI4 Lite

Clocking Options

Clock Type AXI

[✓ Common Clock] Independent Clock

AXI4-Stream FIFOs

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

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

- **AXI4 Interface Options**

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

- **Clocking Options**

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

  The Independent Clock configuration enables the user to implement unique clock domains on the Write and Read ports. The FIFO Generator handles the synchronization between clock domains, placing no requirements on phase and frequency. When data buffering in a single clock domain is required, the FIFO Generator can be used to generate a core optimized for a single clock by selecting the Common Clocks option.

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

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

  Performing Writes with Slave Clock Enable

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

Performing Reads with Master Clock Enable

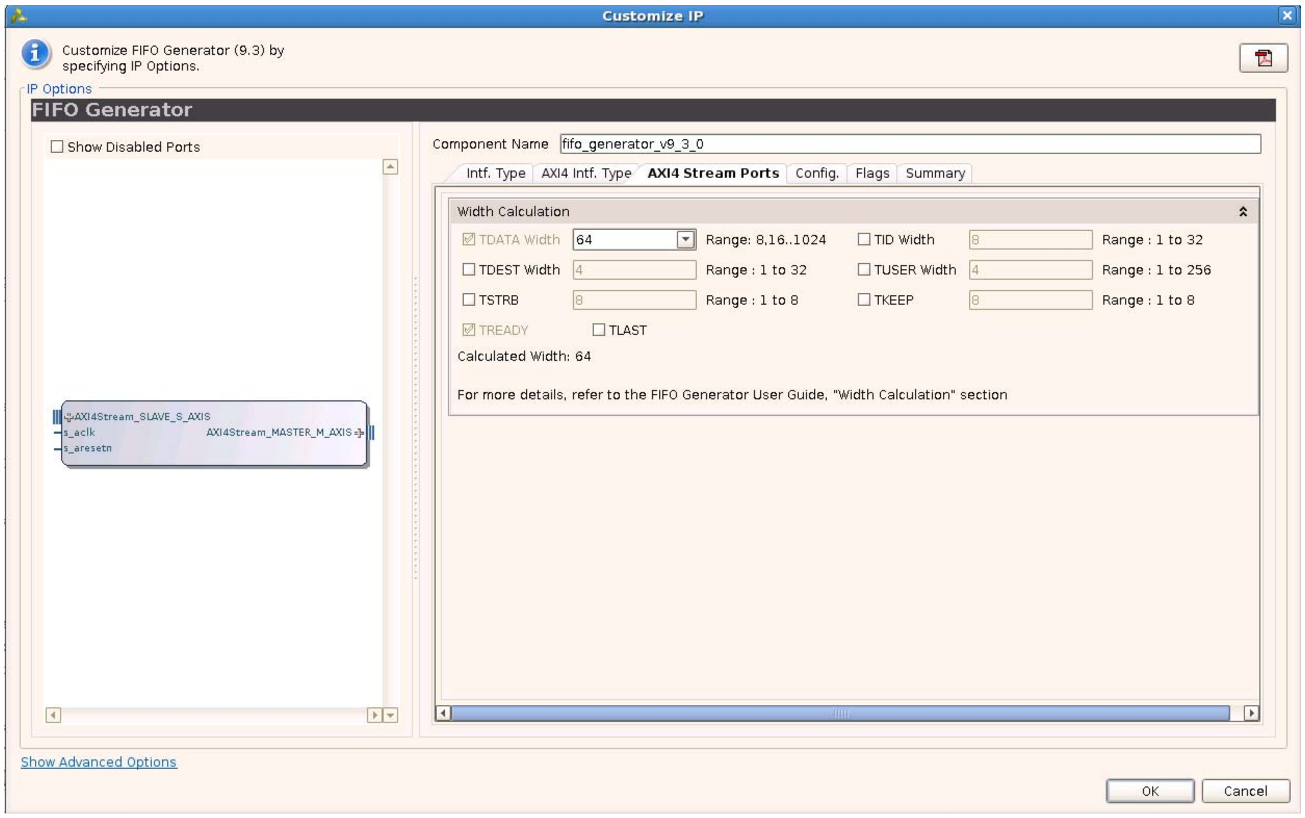
The Master Interface Clock Enable allows AXI4 Slave to operate at fractional rates of AXI4 Master Interface (or Read side) of the FIFO. The above timing diagram shows the AXI4 Slave operating at half the frequency of the FIFO AXI4 Master Interface. The Clock Enable in this case is Single Clock Wide,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

Width Calculation

The AXI4 FIFO Width is determined by aggregating all of the channel information signals in a channel. The channel information signals for AXI4-Stream, AXI4 and AXI4-Lite interfaces are listed in Table 6-1 and Table 6-2. GUI screens are available for configuring:

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

AXI4-Stream Width Calculation



Component Name fifo_generator_v9_3_0

Width Calculation

[✓] TDATA Width	64	Range: 8,16..1024	☐ TID Width	8	Range : 1 to 32
☐ TDEST Width	4	Range : 1 to 32	☐ TUSER Width	4	Range : 1 to 256
☐ TSTRB	8	Range : 1 to 8	☐ TKEEP	8	Range : 1 to 8
[✓] TREADY	☐ TLAST				

Calculated Width: 64

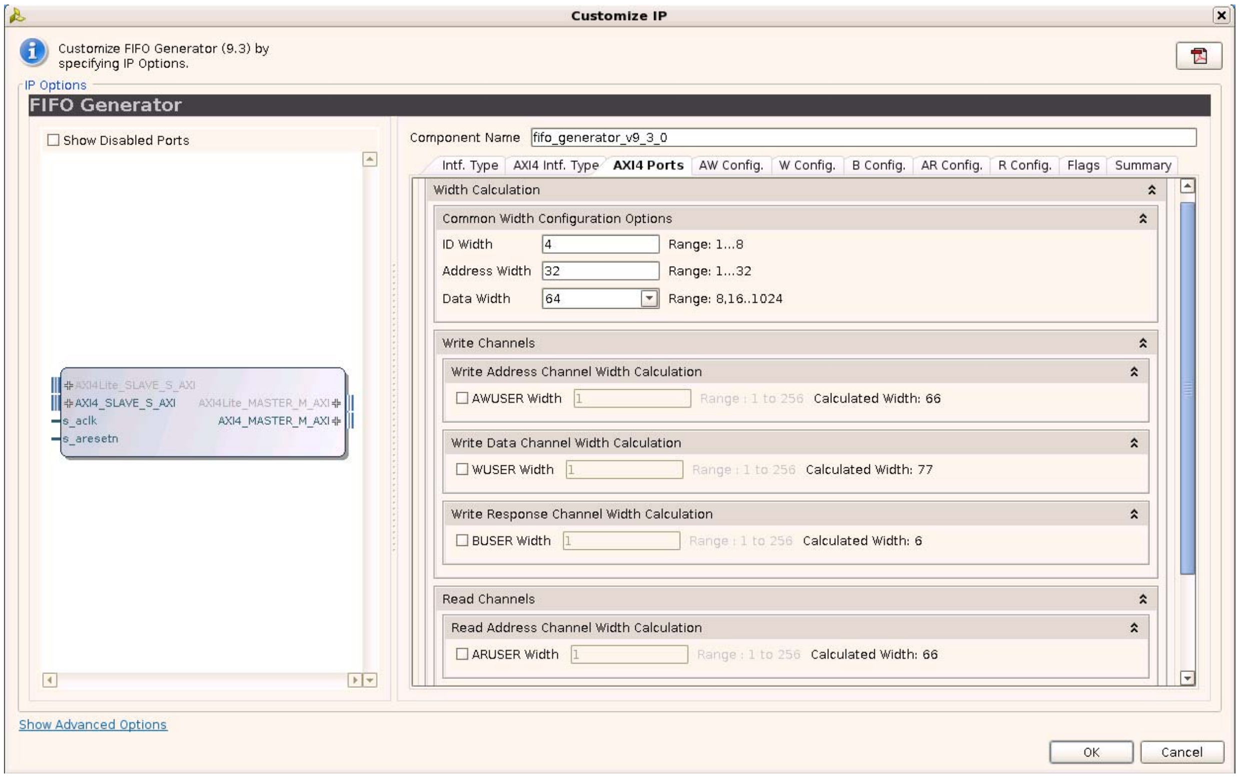
For more details, refer to the FIFO Generator User Guide, "Width Calculation" section

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

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

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

AXI4 Width Calculation



Customize FIFO Generator (9.3) by specifying IP Options.

☐ Show Disabled Ports

Component Name: fifo_generator_v9_3_0

Width Calculation

Common Width Configuration Options

ID Width	4	Range: 1...8
Address Width	32	Range: 1...32
Data Width	64	Range: 8,16..1024

Write Channels

Write Address Channel Width Calculation

☐ AWUSER Width 1 Range : 1 to 256 Calculated Width: 66

Write Data Channel Width Calculation

☐ WUSER Width 1 Range : 1 to 256 Calculated Width: 77

Write Response Channel Width Calculation

☐ BUSER Width 1 Range : 1 to 256 Calculated Width: 6

Read Channels

Read Address Channel Width Calculation

☐ ARUSER Width 1 Range : 1 to 256 Calculated Width: 66

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

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

For all the selected signals, the AXI4 FIFO width for the respective channel is determined by summing up the widths of signals in the particular channel, as shown in Table 6-1.

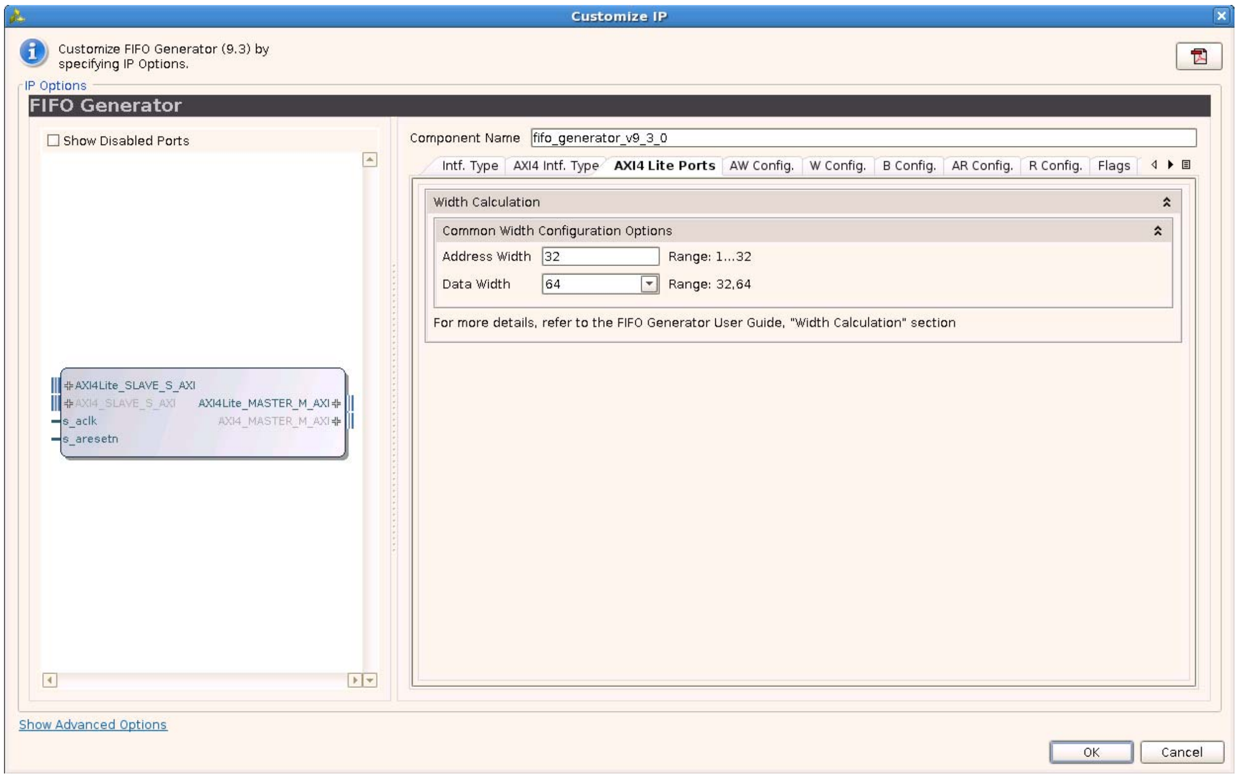
Table 6-1: **AXI4 Signals used in AXI FIFO Width Calculation**

Write Address Channel	Read Address Channel	Write Data Channel	Read Data Channel	Write Resp Channel
AWID[m:0]	ARID[m:0]	WID[m:0]	RID[m:0]	BID[m:0]
AWADDR[m:0]	ARADDR[m:0]	WDATA[m-1:0]	RDATA[m-1:0]	BRESP[1:0]
AWLEN[7:0]	ARLEN[7:0]	WLAST	RLAST	BUSER[m:0]
AWSIZE[2:0]	ARSIZE[2:0]	WSTRB[m/8-1:0]	RRESP[1:0]	
AWBURST[1:0]	ARBURST[1:0]	WUSER[m:0]	RUSER[m:0]	
AWLOCK[2:0]	ARLOCK[2:0]			
AWCACHE[4:0]	ARCACHE[4:0]			

Table 6-1: **AXI4 Signals used in AXI FIFO Width Calculation *(Cont'd)***

Write Address Channel	Read Address Channel	Write Data Channel	Read Data Channel	Write Resp Channel
AWPROT[3:0]	ARPROT[3:0]			
AWREGION[3:0]	ARREGION[3:0]			
AWQOS[3:0]	ARQOS[3:0]			
AWUSER[m:0]	ARUSER[m:0]			

AXI4-Lite Width Calculation



Component Name: fifo_generator_v9_3_0

Width Calculation

Common Width Configuration Options

Address Width: 32 Range: 1...32

Data Width: 64 Range: 32,64

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

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

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

AXI4-Lite FIFO width for the respective channel is determined by summing up the widths of all the signals in the particular channel, as shown in Table 6-2.

Table 6-2: **AXI4-Lite Width Calculation**

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

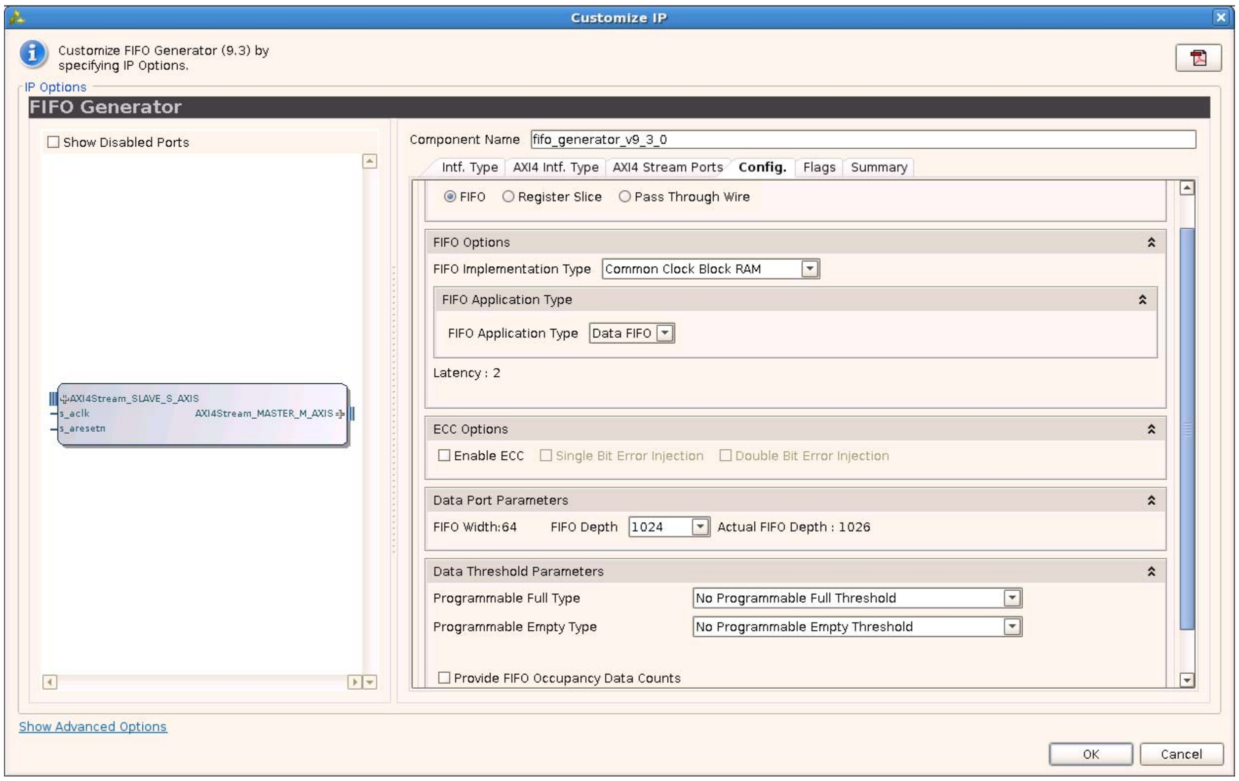
Default Settings

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

Table 6-3: **AXI4 FIFO Default Settings**

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

FIFO Configurations


Customize IP

Customize FIFO Generator (9.3) by specifying IP Options.

IP Options

FIFO Generator

☐ Show Disabled Ports

Component Name: fifo_generator_v9_3_0

Intf. Type | AXI4 Intf. Type | AXI4 Stream Ports | Config. | Flags | Summary

[✓ FIFO] Register Slice Pass Through Wire

FIFO Options

FIFO Implementation Type: Common Clock Block RAM

FIFO Application Type

FIFO Application Type: Data FIFO

Latency : 2

ECC Options

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

Data Port Parameters

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

Data Threshold Parameters

Programmable Full Type: No Programmable Full Threshold

Programmable Empty Type: No Programmable Empty Threshold

☐ Provide FIFO Occupancy Data Counts

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

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

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

Memory Types

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

Error Injection and Correction (ECC)

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

For more details on Error Injection and Correction, see Built-in Error Correction Checking in Chapter 3.

FIFO Width

AXI4 FIFOs support symmetric Write and Read widths. The width of the AXI4 FIFO is determined based on the selected Interface Type (AXI4-Stream, AXI4 or AXI4-Lite), and the selected signals and configured signal widths within the given interface. The calculation of the FIFO Write Width is defined in Width Calculation, page 174.

FIFO Depth

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

Programmable Flags

This section includes details about the available programmable flags.

Programmable Full Type

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

Full Threshold Assert Value

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

Programmable Empty Type

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

Empty Threshold Assert Value

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

Data Threshold Parameters

This section includes details about data threshold parameters.

Occupancy Data Counts

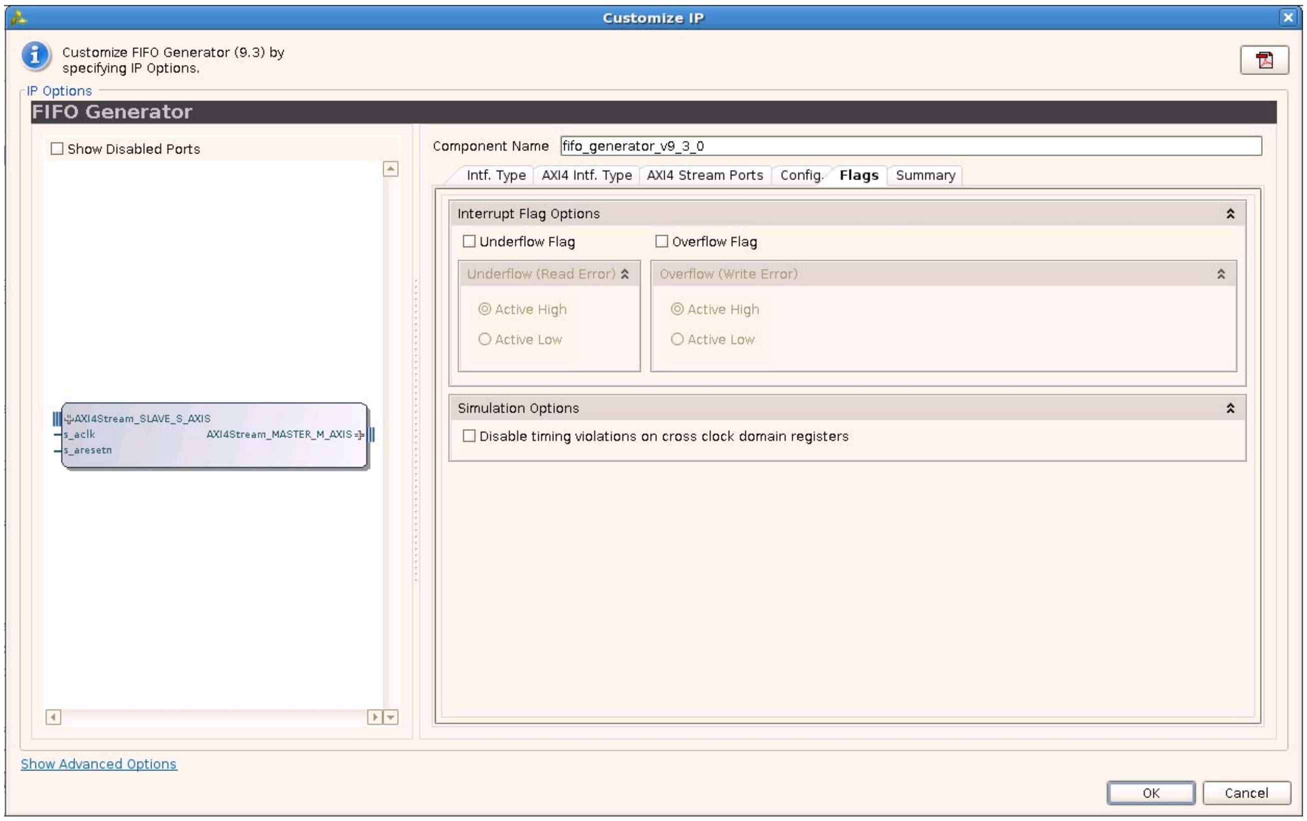
DATA_COUNT tracks the number of words in the FIFO. The width of the data count bus will be always be set to $\log_2$(FIFO depth)+1. In common clock mode, the AXI4 FIFO provides a single "Data Count" output. In independent clock mode, it provides Read Data Count and Write Data Count outputs.

For more details on Occupancy Data Counts, see First-Word Fall-Through Data Count in Chapter 3 and More Accurate Data Count (Use Extra Logic) in Chapter 3.

Examples for Data Threshold Parameters

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

Common Configurations



Component Name: fifo_generator_v9_3_0

Interrupt Flag Options
- ☐ Underflow Flag
- ☐ Overflow Flag

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

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

Simulation Options
- ☐ Disable timing violations on cross clock domain registers

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

Interrupt Flags

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

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

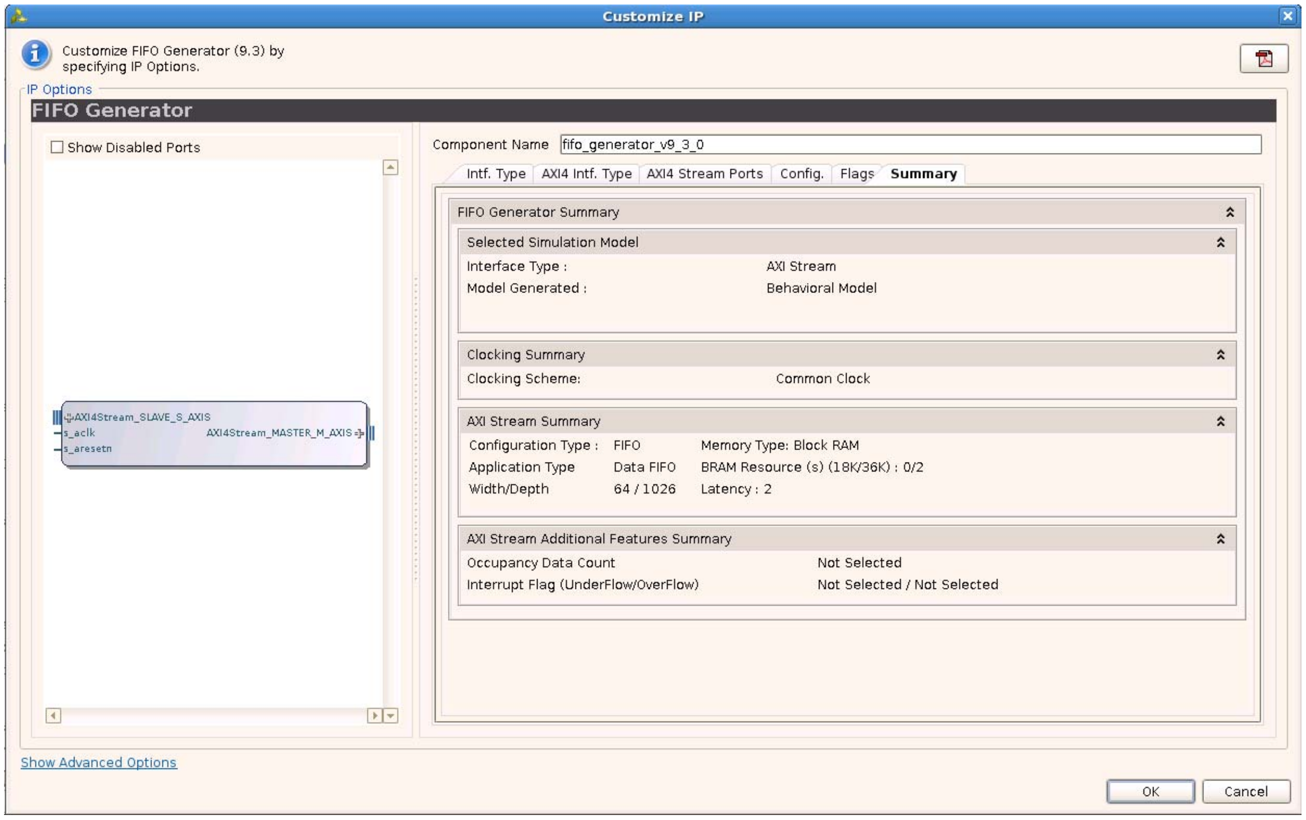
For more details on Overflow and Underflow Flags, see Underflow in Chapter 3 and Overflow in Chapter 3.

Summary

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

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

AXI4-Stream Summary



Component Name: fifo_generator_v9_3_0

FIFO Generator Summary

Selected Simulation Model

Interface Type :	AXI Stream
Model Generated :	Behavioral Model

Clocking Summary

Clocking Scheme:	Common Clock

AXI Stream Summary

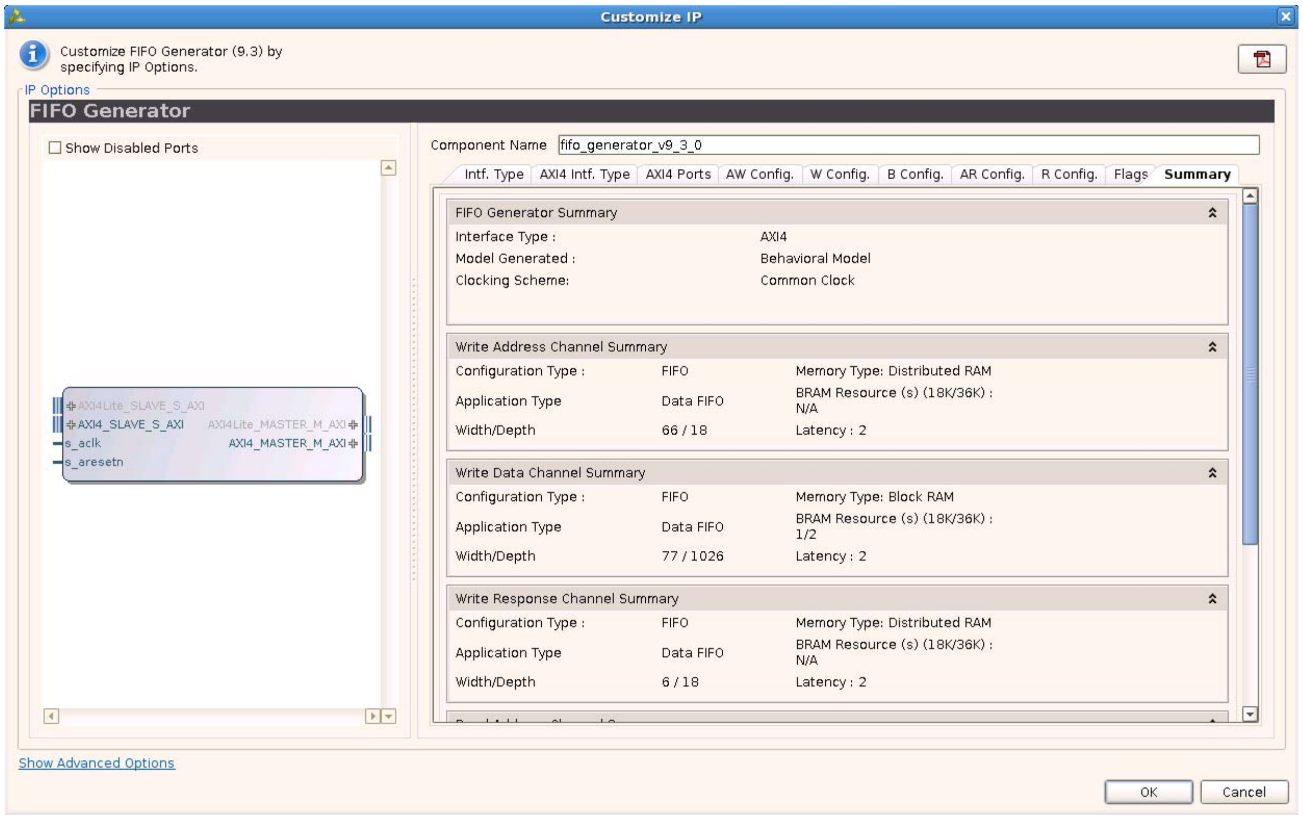
Configuration Type :	FIFO	Memory Type: Block RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) : 0/2
Width/Depth	64 / 1026	Latency : 2

AXI Stream Additional Features Summary

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

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

AXI4 and AXI4-Lite Summary



Component Name: fifo_generator_v9_3_0

FIFO Generator Summary

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

Write Address Channel Summary

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

Write Data Channel Summary

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

Write Response Channel Summary

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

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

Output Generation

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

Constraining the Core

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

Required Constraints

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

Device, Package, and Speed Grade Selections

See IP Facts for details about supported devices.

Clock Frequencies

There are no clock frequency constraints.

Clock Management

There are no additional clock management constraints for this core.

Clock Placement

There are no additional clock placement constraints for this core.

Detailed Example Design

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

Directory and File Contents

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

Example Design

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

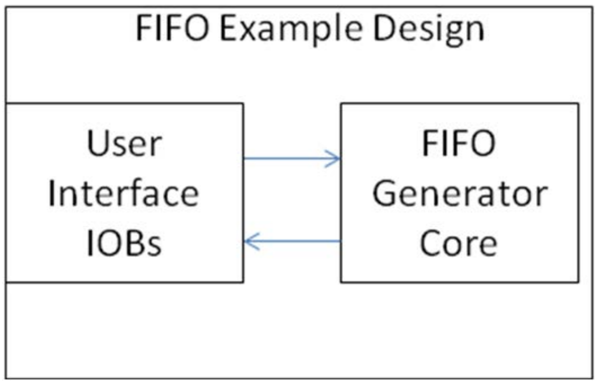


Figure 8-1: **Example Design Configuration**

The example design contains the following:

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

Demonstration Test Bench

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

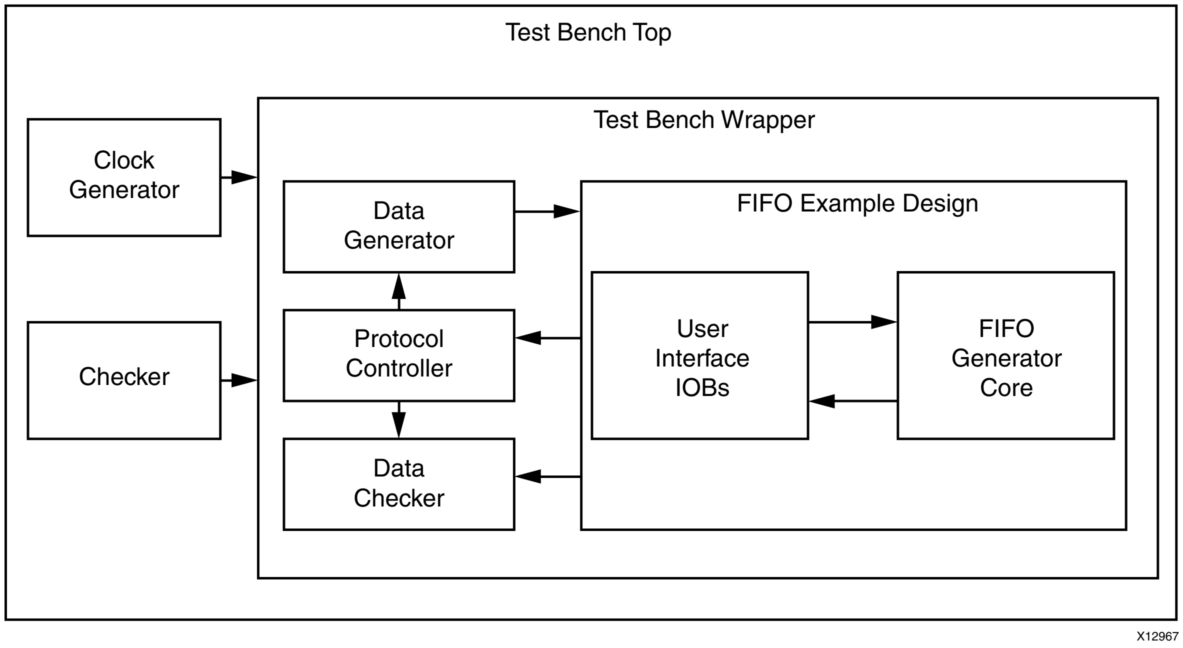


Figure 8-2: **Demonstration Test Bench**

Test Bench Functionality

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

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

Core with Native Interface

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

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

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

Core with AXI4 Interface

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

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

Customizing the Demonstration Test Bench

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

Changing the Data/Stimulus

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

Changing the Test Bench Run Time

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

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

Implementation

The implementation script is either a shell script (.sh) or batch file (.bat) that processes the example design through the Xilinx tool flow. It is located at:

Linux

```
<project_dir>/<component_name>/implement/implement.sh
```

Windows

```
<project_dir>/<component_name>/implement/implement.bat
```

The implement script performs these steps:

- Synthesizes the HDL example design files using XST
- Runs NGDBuild to consolidate the core netlist and the example design netlist into the NGD file containing the entire design
- Maps the design to the target technology
- Place-and-routes the design on the target device
- Performs static timing analysis on the routed design using Timing Analyzer (TRCE)
- Generates a bitstream
- Enables Netgen to run on the routed design to generate a VHDL or Verilog netlist (as appropriate for the Design Entry project setting) and timing information in the form of SDF files

The Xilinx tool flow generates several output and report files. These are saved in the following directory which is created by the implement script:

```
<project_dir>/<component_name>/implement/results
```

Simulation

This section contains details about the test scripts included in the example design.

Functional Simulation

The test scripts are ModelSim macros that automate the simulation of the test bench. They are available from the following location:

```
<project_dir>/<component_name>/simulation/functional/
```

The test script performs these tasks:

- Compiles the behavioral model/structural UNISIM simulation model.
- Compiles HDL Example Design source code.
- Compiles the demonstration test bench.
- Starts a simulation of the test bench.
- Opens a Wave window and adds signals of interest (`wave_mti.do`).
- Runs the simulation to completion.

Timing Simulation

The test scripts are ModelSim macros that automate the simulation of the test bench. They are located in:

```
<project_dir>/<component_name>/simulation/timing/
```

The test script performs these tasks:

- Compiles the SIMPRIM based gate level netlist simulation model.
- Compiles the demonstration test bench.
- Starts a simulation of the test bench.
- Opens a Wave window and adds signals of interest (`wave_mti.do`).
- Runs the simulation to completion.

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
```

SECTION III: ISE DESIGN SUITE

Chapter 9

Customizing and Generating the Native Core

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

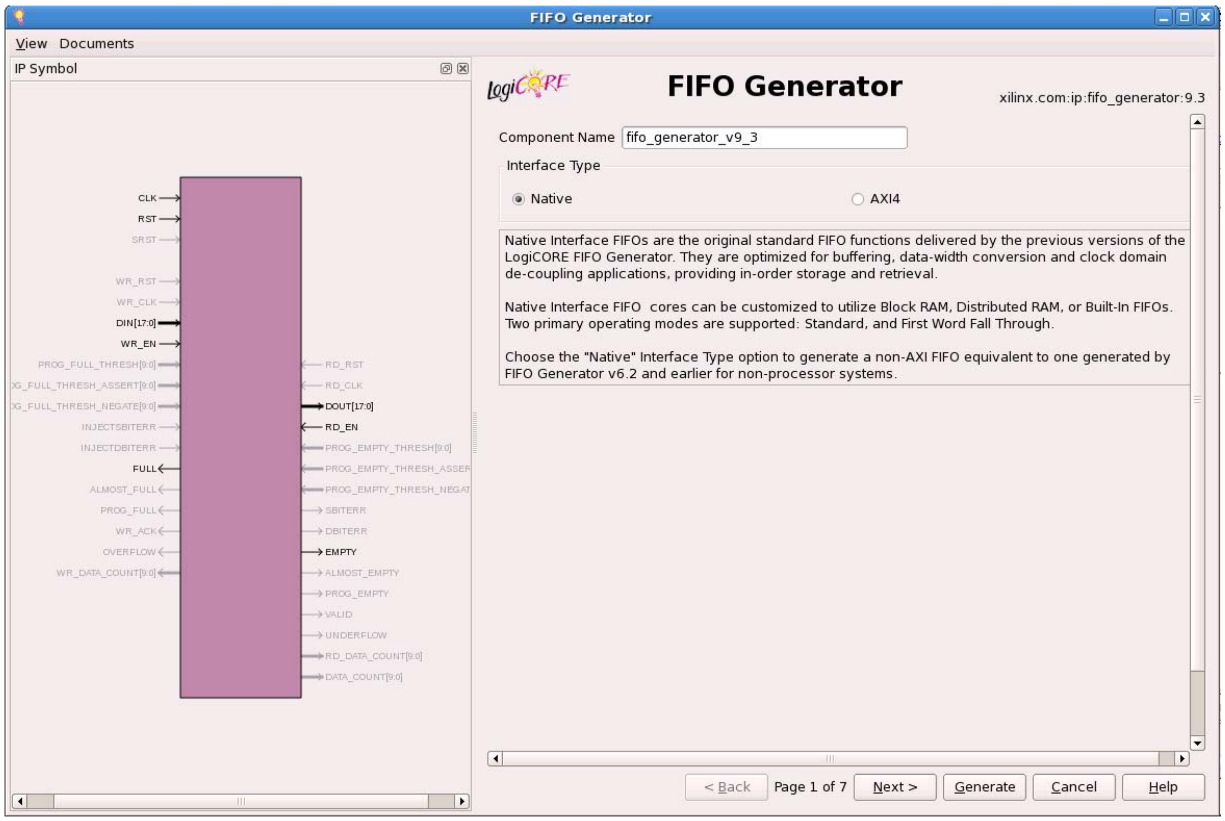
GUI

The Native FIFO Interface GUI includes seven configuration screens.

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

Interface Type

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



Component Name fifo_generator_v9_3

Interface Type

(● Native) ○ AXI4

Native Interface FIFOs are the original standard FIFO functions delivered by the previous versions of the LogiCORE FIFO Generator. They are optimized for buffering, data-width conversion and clock domain de-coupling applications, providing in-order storage and retrieval.

Native Interface FIFO cores can be customized to utilize Block RAM, Distributed RAM, or Built-In FIFOs. Two primary operating modes are supported: Standard, and First Word Fall Through.

Choose the "Native" Interface Type option to generate a non-AXI FIFO equivalent to one generated by FIFO Generator v6.2 and earlier for non-processor systems.

Figure 9-1: **Main FIFO Generator Screen**

- **Component Name**

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

- **Interface Type**
  - **Native**

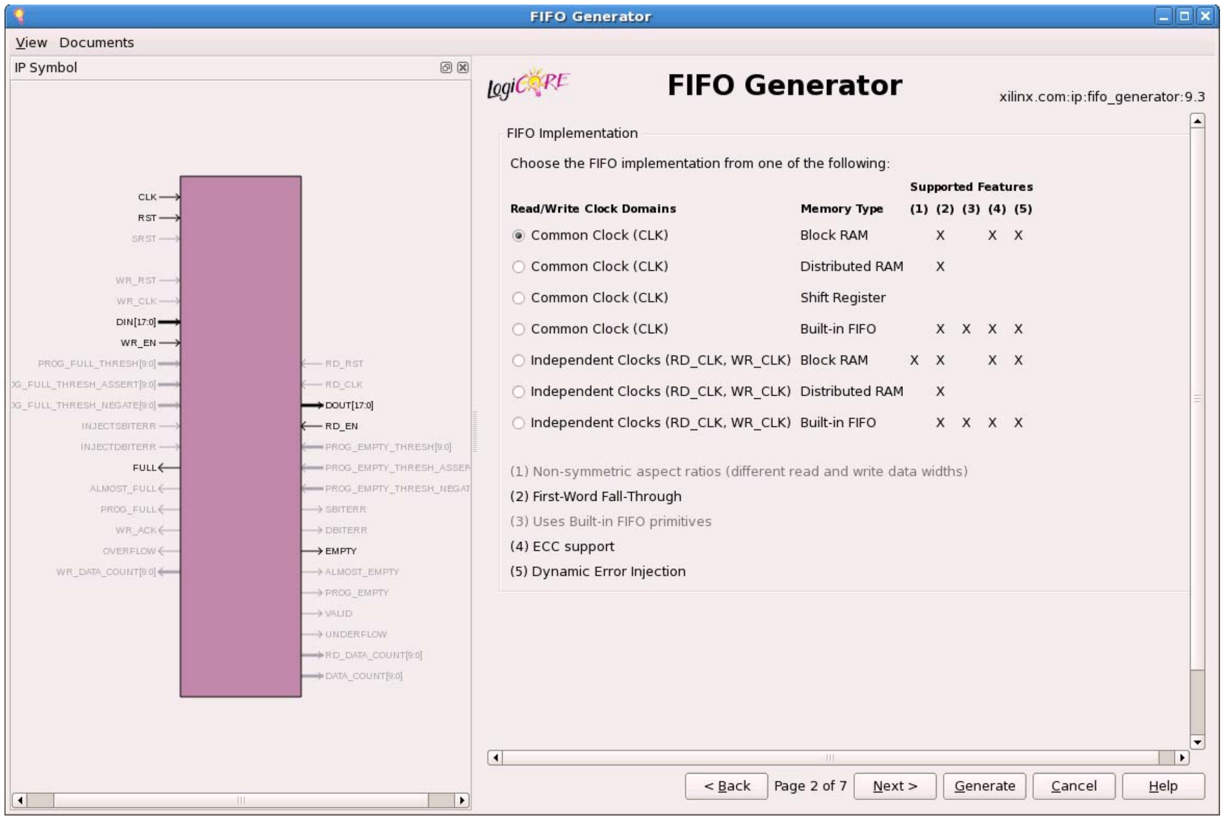
    Implements a Native FIFO.

  - **AXI4**

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

FIFO Implementation

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



FIFO Implementation

Choose the FIFO implementation from one of the following:

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

(Columns (1)–(5): Supported Features)

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

Figure 9-2: **FIFO Implementation Screen**

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

- **Common Clock (CLK), Block RAM**

  For details, see Common Clock FIFO: Block RAM and Distributed RAM, page 123. This implementation optionally supports first-word-fall-through (selectable in the second GUI screen, shown in Figure 9-3).

- **Common Clock (CLK), Distributed RAM**

  For details, see Common Clock FIFO: Block RAM and Distributed RAM, page 123. This implementation optionally supports first-word-fall-through (selectable in the second GUI screen, shown in Figure 9-3).

- **Common Clock (CLK), Shift Register**

For details, see Common Clock FIFO: Shift Registers, page 124. This implementation is only available in Virtex-4 FPGA and newer architectures.

- **Common Clock (CLK), Built-in FIFO**

  For details, see Common Clock: Built-in FIFO, page 123. This implementation is only available when using the Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA architectures. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, shown in Figure 9-3).

- **Independent Clocks (RD_CLK, WR_CLK), Block RAM**

  For details, see Independent Clocks: Block RAM and Distributed RAM, page 120. This implementation optionally supports asymmetric read/write ports and first-word fall-through (selectable in the second GUI screen, shown in Figure 9-3).

- **Independent Clocks (RD_CLK, WR_CLK), Distributed RAM**

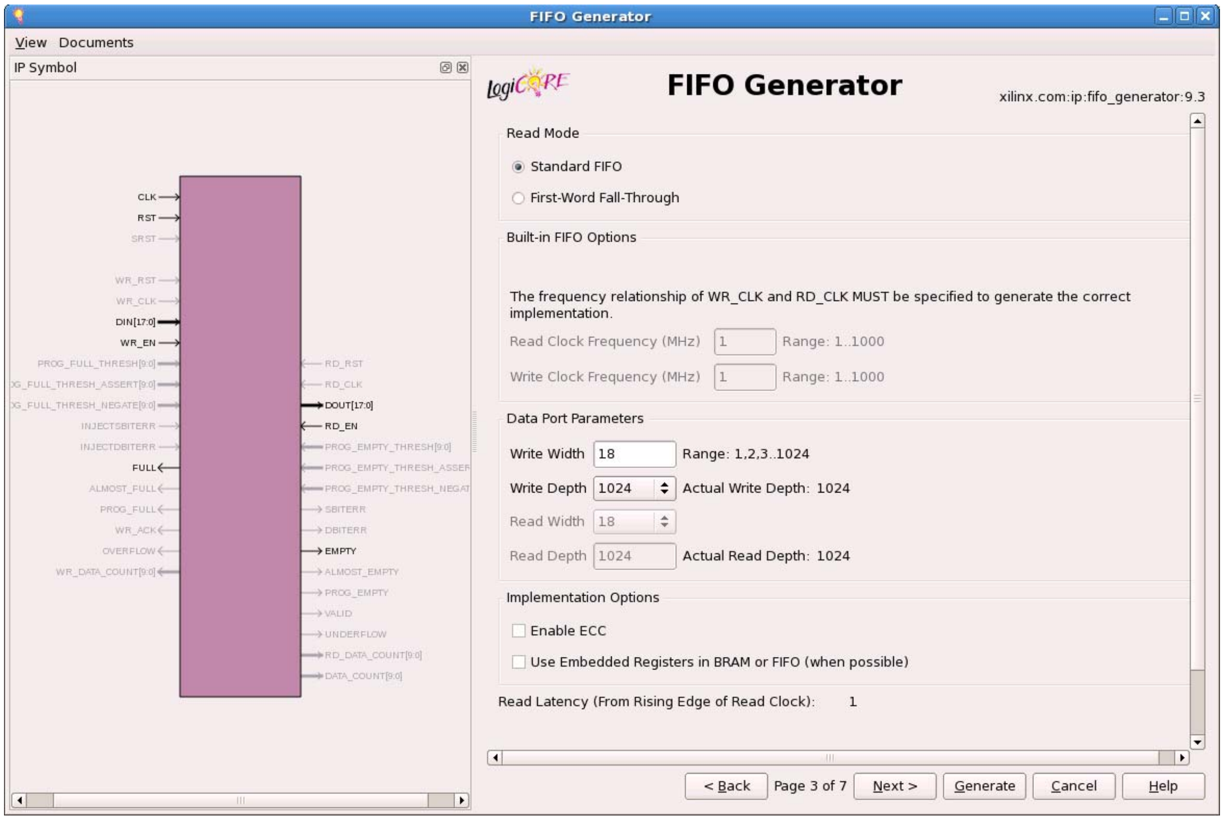
  For more information, see Independent Clocks: Block RAM and Distributed RAM, page 120. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, shown in Figure 9-3).

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

  For more information, see Independent Clocks: Built-in FIFO, page 122. This implementation is only available when using Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA architectures. This implementation optionally supports first-word fall-through (selectable in the second GUI screen, shown in Figure 9-3).

Performance Options and Data Port Parameters

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



FIFO Generator xilinx.com:ip:fifo_generator:9.3

Read Mode

(●) Standard FIFO

( ) First-Word Fall-Through

Built-in FIFO Options

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

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

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

Data Port Parameters

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

Write Depth 1024 Actual Write Depth: 1024

Read Width 18

Read Depth 1024 Actual Read Depth: 1024

Implementation Options

[ ] Enable ECC

[ ] Use Embedded Registers in BRAM or FIFO (when possible)

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

Figure 9-3: **Performance Options and Data Port Parameters Screen**

- **Read Mode**

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

  - **Standard FIFO**

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

  - **First-Word Fall-Through FIFO**

    Implements a FIFO with registered outputs. For more information about FWFT functionality, see First-Word Fall-Through FIFO Read Operation, page 98.

- **Built-in FIFO Options**

  - **Read/Write Clock Frequencies**

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

IMPORTANT: *It is critical that Read Clock and Write Clock frequency data is entered and accurate. If this information is not provided, it can result in a sub-optimal solution with incorrect core behavior.*

- **Data Port Parameters**
  - **Write Width**

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

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

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

    Available only if independent clocks configuration with block RAM is selected. Valid range must comply with asymmetric port rules. See Non-symmetric Aspect Ratios, page 113.
  - **Read Depth**

    Automatically calculated based on Write Width, Write Depth, and Read Width.
- **Implementation Options**
  - **Error Correction Checking in Block RAM or Built-in FIFO**

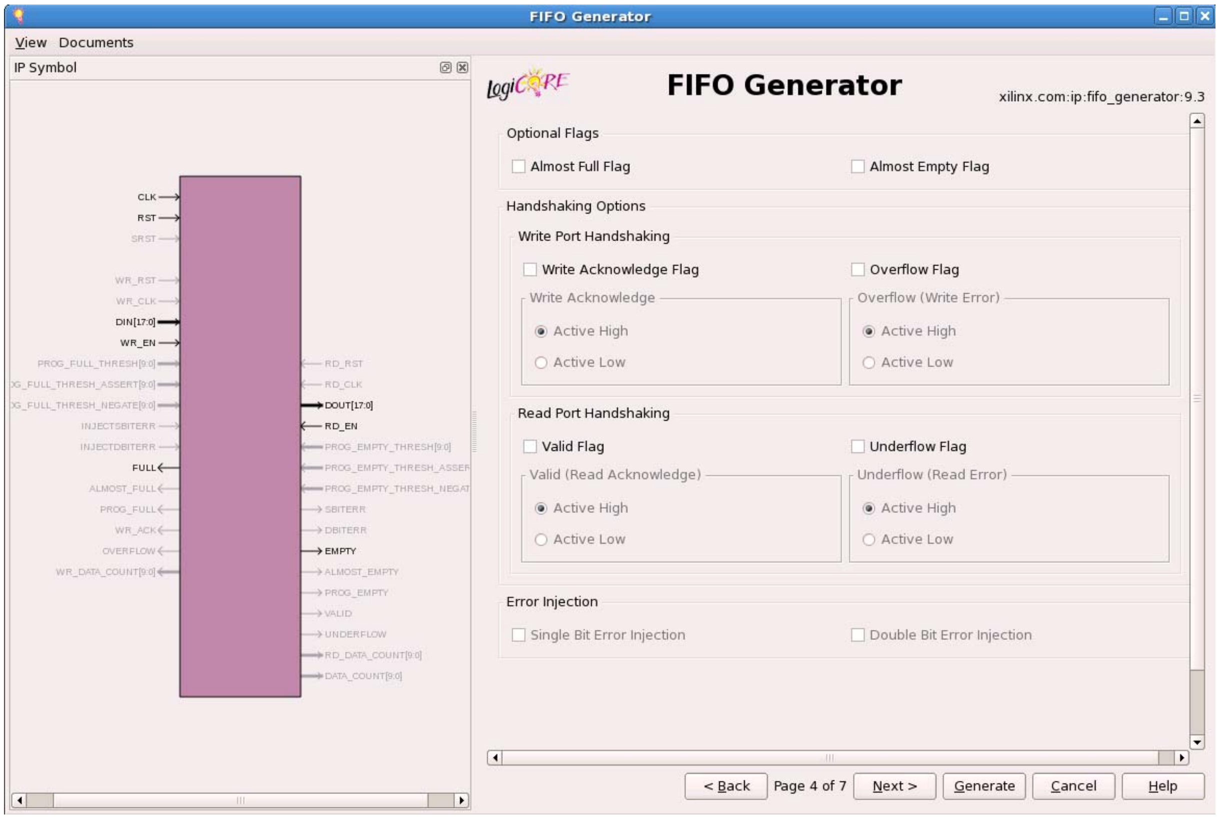
    The Error Correction Checking (ECC) feature enables built-in error correction in the Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA block RAM and built-in FIFO macros. When this feature is enabled, the block RAM or built-in FIFO is set to the full ECC mode, where both the encoder and decoder are enabled.

- **Use Embedded Registers in Block RAM or FIFO**

  The block RAM macros available in Kintex-7, Virtex-7, Virtex-6, Virtex-5 and Virtex-4 FPGA, as well as built-in FIFO macros available in Kintex-7, Virtex-7, Virtex-6 and Virtex-5 FPGA, have built-in embedded registers that can be used to pipeline data and improve macro timing. This option enables users to add one pipeline stage to the output of the FIFO and take advantage of the available embedded registers; however, the ability to reset the data output of the Virtex-5 FPGA built-in FIFO is disabled when this feature is used. For built-in FIFOs, this feature is only supported for synchronous FIFO configurations that have only 1 FIFO macro in depth. See Embedded Registers in Block RAM and FIFO Macros (Zynq-7000, 7 Series, Virtex-6, Virtex-5 and Virtex-4 FPGAs), page 116.

Optional Flags, Handshaking, and Initialization

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



Optional Flags

☐ Almost Full Flag ☐ Almost Empty Flag

Handshaking Options

Write Port Handshaking

☐ Write Acknowledge Flag ☐ Overflow Flag

Write Acknowledge: [✓ Active High] Active Low

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

Read Port Handshaking

☐ Valid Flag ☐ Underflow Flag

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

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

Error Injection

☐ Single Bit Error Injection ☐ Double Bit Error Injection

Figure 9-4: **Optional Flags, Handshaking, and Error Injection Options Screen**

- **Optional Flags**

  Refer to 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 Kintex-7, Virtex-7, Virtex-6, Virtex-5 or Virtex-4 FPGA built-in FIFOs. Generates an output port that indicates the FIFO is almost full (only one more word can be written).

- **Almost Empty Flag**

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

- **Handshaking Options**

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

    - **Underflow (Read Error)**

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

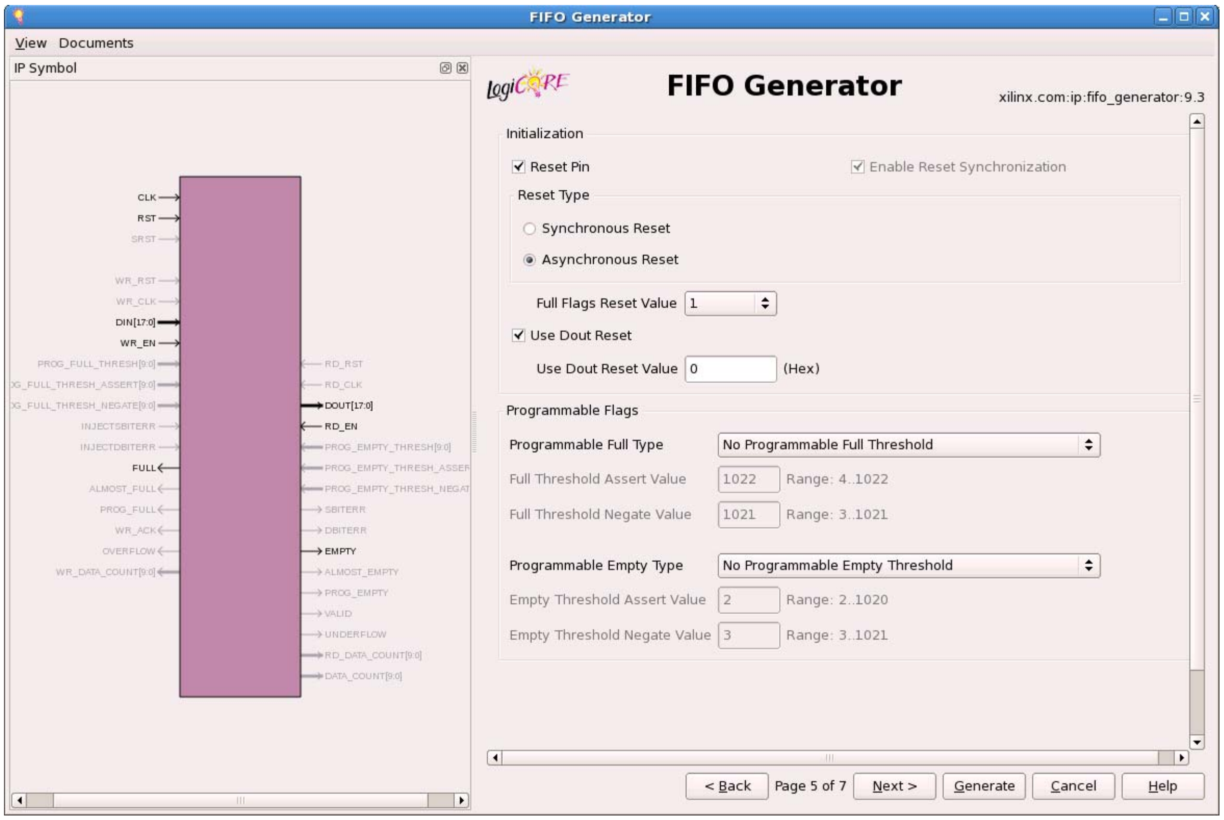
- **Error Injection**
  - **Single Bit Error Injection**

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

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

Initialization and Programmable Flags

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



FIFO Generator

xilinx.com:ip:fifo_generator:9.3

Initialization

[✓] Reset Pin [✓] Enable Reset Synchronization

Reset Type

○ Synchronous Reset

● Asynchronous Reset

Full Flags Reset Value: 1

[✓] Use Dout Reset

Use Dout Reset Value: 0 (Hex)

Programmable Flags

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

Figure 9-5: **Programmable Flags and Reset Screen**

- **Initialization**
  - **Reset Pin**

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

    - **Reset Type**

      - **Enable Reset Synchronization**

        Optional selection only available for independent clock block RAM or distributed RAM FIFOs. When unchecked, WR_RST/RD_RST is available. See Resets in Chapter 3 for details.

        - **Asynchronous Reset**

          Optional selection for a common-clock FIFO implemented using distributed or block RAM.

        - **Synchronous Reset**

          Optional selection for a a common-clock FIFO implemented using distributed or block RAM.

      - **Full Flags Reset Value**

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

  - **Use Dout Reset**

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

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

      - **Use Dout Reset Value**

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

- **Programmable Flags**

  Refer to Latency in Chapter 3 for the latency of the programmable flags due to write/ read operation.

  - **Programmable Full Type**

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

- **Full Threshold Assert Value**

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

- **Full Threshold Negate Value**

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

◦ **Programmable Empty Type**

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

- **Empty Threshold Assert Value**

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

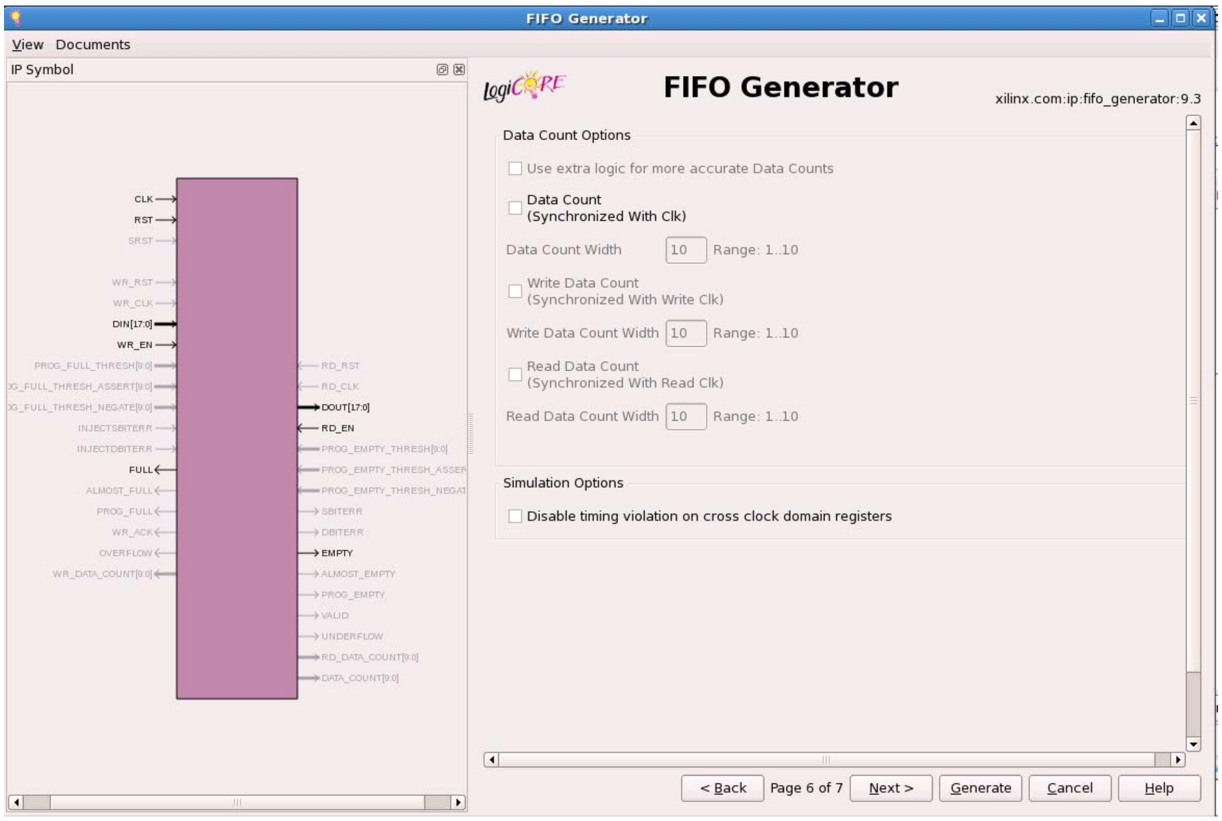
- **Empty Threshold Negate Value**

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

Data Count

Use this screen to set data count options.

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



FIFO Generator

xilinx.com:ip:fifo_generator:9.3

Data Count Options

☐ Use extra logic for more accurate Data Counts

☐ Data Count (Synchronized With Clk)

Data Count Width 10 Range: 1..10

☐ Write Data Count (Synchronized With Write Clk)

Write Data Count Width 10 Range: 1..10

☐ Read Data Count (Synchronized With Read Clk)

Read Data Count Width 10 Range: 1..10

Simulation Options

☐ Disable timing violation on cross clock domain registers

Figure 9-6: **Data Count Screen**

- **Data Count Options**

  Refer to Latency in Chapter 3 for the latency of the data counts due to write/read operation.

  - **Use Extra Logic For More Accurate Data Counts**

    Only available for independent clocks FIFO with block RAM or distributed RAM, and when using first-word fall-through. This option uses additional external logic to generate a more accurate data count. This feature is always enabled for common clock FIFOs with block RAM or distributed RAM and when using first-word-fall-through. See First-Word Fall-Through Data Count, page 110 for details.

- **Data Count (Synchronized With Clk)**

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

  - **Data Count Width**

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

- **Write Data Count (Synchronized with Write Clk)**

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

  - **Write Data Count Width**

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

- **Read Data Count (Synchronized with Read Clk)**

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

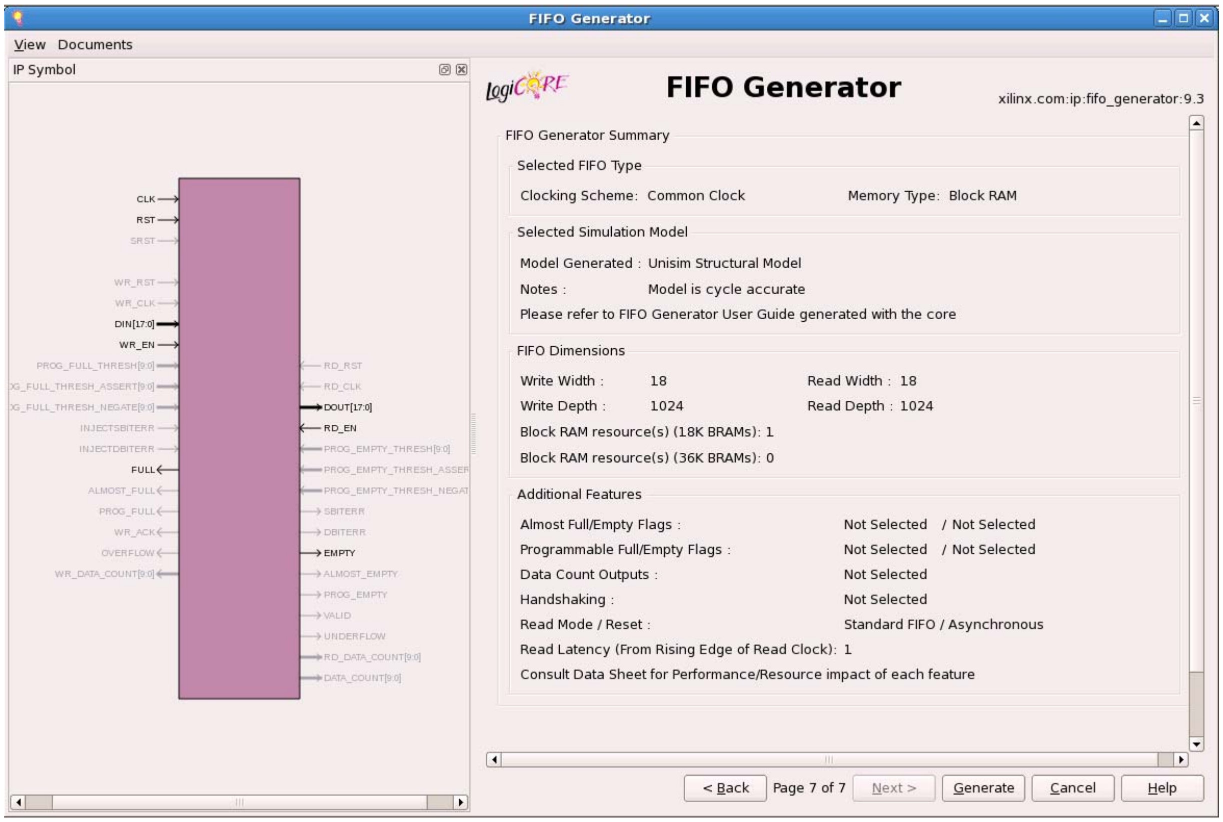
  - **Read Data Count Width**

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

Summary

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

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



FIFO Generator

xilinx.com:ip:fifo_generator:9.3

FIFO Generator Summary

Selected FIFO Type

Clocking Scheme: Common Clock   Memory Type: Block RAM

Selected Simulation Model

Model Generated : Unisim Structural Model
Notes : Model is cycle accurate
Please refer to FIFO Generator User Guide generated with the core

FIFO Dimensions

Write Width : 18   Read Width : 18
Write Depth : 1024   Read Depth : 1024
Block RAM resource(s) (18K BRAMs): 1
Block RAM resource(s) (36K BRAMs): 0

Additional Features

Almost Full/Empty Flags :	Not Selected / Not Selected
Programmable Full/Empty Flags :	Not Selected / Not Selected
Data Count Outputs :	Not Selected
Handshaking :	Not Selected
Read Mode / Reset :	Standard FIFO / Asynchronous

Read Latency (From Rising Edge of Read Clock): 1
Consult Data Sheet for Performance/Resource impact of each feature

Figure 9-7: **Summary Screen**

Parameter Values in the XCO File

Table 9-1 describes the Native FIFO core parameters, including the XCO file value and the default settings.

Table 9-1: **Native Interface FIFO XCO Parameter Table**

Native FIFO XCO Parameter Name	XCO File Values	Default GUI Settings
interface_type	Native, AXI4	Native
almost_empty_flag	True, False	False

Table 9-1: **Native Interface FIFO XCO Parameter Table *(Cont'd)***

Native FIFO XCO Parameter Name	XCO File Values	Default GUI Settings
almost_full_flag	True, False	False
component_name	instance_name ASCII text starting with a letter and using the following character set: a-z, 0-9, and _	fifo_generator_v9_3
data_count	True, False	False
data_count_width	1 – $\log_2$(output_depth)	10
disable_timing_violations	True, False	False
dout_reset_value	Hex value in range of 0 to output data width - 1	0
empty_threshold_assert_value	For STD: 2 - 4194300 For FWFT: 4 - 4194302	2
empty_threshold_negate_value	For STD: 3 - 4194301 For FWFT: 5 - 4194303	3
enable_ecc	True, False	False
enable_reset_synchronization	True, False	True
fifo_implementation	Common_Clock_Block_RAM Common_Clock_Distributed_RAM Common_Clock_Shift_Register Common_Clock_Builtin_FIFO Independent_Clocks_Block_RAM Independent_Clocks_Distributed_RAM Independent_Clocks_Builtin_FIFO	Common_Clock_Block_RAM
full_flags_reset_value	0, 1	1
full_threshold_assert_value	For STD: 4 - 4194302 For FWFT: 6 - 4194303	1022
full_threshold_negate_value	For STD: 3 - 4194301 For FWFT: 5 - 4194302	1021
inject_dbit_error	True, False	False
inject_sbit_error	True, False	False
input_data_width	1 - 1024	18
input_depth	2^4 - 2^{22}	1024
output_data_width	1 - 1024	18
output_depth	2^4 - 2^{22}	1024
overflow_flag	True, False	False
overflow_sense	Active_High, Active_Low	Active_High
performance_options	Standard_FIFO (STD), First_Word_Fall_Through (FWFT)	Standard_FIFO

Table 9-1: **Native Interface FIFO XCO Parameter Table *(Cont'd)***

Native FIFO XCO Parameter Name	XCO File Values	Default GUI Settings
programmable_empty_type	No_Programmable_Empty_ Threshold Single_Programmable_Empty_ Threshold_Constant Multiple_Programmable_Empty_ Threshold_Constants Single_Programmable_Empty_ Threshold_Input_Port Multiple_Programmable_Empty_ Threshold_Input_Ports	No_Programmable_ Empty_Threshold
programmable_full_type	No_Programmable_Full_Threshold Single_Programmable_Full_ Threshold_Constant Multiple_Programmable_Full_ Threshold_Constants Single_Programmable_Full_ Threshold_Input_Port Multiple_Programmable_Full_ Threshold_Input_Ports	No_Programmable_ Full_Threshold
read_clock_frequency	1 - 1000	1
read_data_count	True, False	False
read_data_count_width	1 – $\log_2$(output_depth)	10
reset_pin	True, False	True
reset_type	Synchronous_Reset, Asynchronous_Reset	Asynchronous_Reset
underflow_flag	True, False	False
underflow_sense	Active_High, Active_Low	Active_High
use_dout_reset	True, False	False
use_embedded_registers	True, False	False
use_extra_logic	True, False	False
valid_flag	True, False	False
valid_sense	Active_High, Active_Low	Active_High
write_acknowledge_flag	True, False	False
write_acknowledge_sense	Active_High, Active_Low	Active_High
write_clock_frequency	1 – 1000	1
write_data_count	True, False	False
write_data_count_width	1 – $\log_2$(input_depth)	10

Output Generation

This section provides a detailed description of files and the directory structure generated by the Xilinx® CORE Generator™ software.

- **<project_directory>**
  Top-level project directory; name is user-defined
  - <project_directory>/<component_name>
    Contains the FIFO Generator release notes text file
    - <component_name>/example design
      Verilog and VHDL design files
    - <component_name>/implement
      Implementation script files
    - <component_name>/implement/results
      Created after implementation scripts are run and contains implement script results
    - <component_name>/simulation
      Contains the test bench and other supporting source files used to create the simulation model
      - simulation/functional
        Functional simulation scripts
      - simulation/timing
        Timing simulation scripts

<project_directory>

The `project` directory contains all the CORE Generator tool project files.

Table 9-2: **Project Directory**

Name	Description
`<project_directory>`	
`<component_name>.ngc`	Top-level netlist.
`<component_name>.v[hd]`	Verilog or VHDL simulation model.
`<component_name>.xco`	CORE Generator software project-specific option file; can be used as an input to the CORE Generator software.
`<component_name>_flist.txt`	List of files delivered with the core.
`<component_name>.{veo|vho}`	VHDL or Verilog instantiation template.

Back to Top

<project_directory>/<component_name>

The `component name` directory contains the release notes in the readme file provided with the core, which can include tool requirements, updates, and issue resolution.

Table 9-3: **Component Name Directory**

Name	Description
`<project_directory>/<component_name>`	
`fifo_generator_v9_3_readme.txt`	Core name release notes file.

Back to Top

<component_name>/example design

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

Table 9-4: **Example Design Directory**

Name	Description
`<project_directory>/<component_name>/example_design`	
`<component_name>_top.ucf`	Provides example constraints necessary for processing the FIFO Generator core using the Xilinx implementation tools.
`<component_name>_top.vhd`	The VHDL top-level file for the example design; it instantiates the FIFO Generator core. This file contains entity with the IO's required for the core configuration.
`<component_name>_top_wrapper.v[hd]`	The VHDL wrapper file for the example design <component_name>_top.vhd file. This file contains entity with all ports of FIFO Generator core.

Back to Top

<component_name>/implement

The `implement` directory contains the core implementation script files.

Table 9-5: **Implement Directory**

Name	Description
`<project_directory>/<component_name>/implement`	
`implement.{bat|sh}`	A Windows (.bat) or Linux script that processes the example design.
`xst.prj`	The XST project file for the example design that lists all of the source files to be synthesized. Only available when the CORE Generator software project option is set to ISE or Other.

Table 9-5: **Implement Directory *(Cont'd)***

Name	Description
`xst.scr`	The XST script file for the example design used to synthesize the core. Only available when the CORE Generator software Vendor project option is set to ISE or Other.

Back to Top

<component_name>/implement/results

The `results` directory is created by the implement script. The implement script results are placed in the `results` directory.

Table 9-6: **Results Directory**

Name	Description
`<project_directory>/<component_name>/results`	
Implement script result files.	

Back to Top

<component_name>/simulation

The `simulation` directory contains the demo test bench files provided with the core.

Table 9-7: **Simulation Directory**

Name	Description
`<project_directory>/<component_name>/simulation`	
`fg_tb_pkg.vhd`	VHDL File provided with demonstration test bench. It contains common functions required by the test bench.
`fg_tb_rng.vhd`	VHDL File provided with demonstration test bench. It contains logic for pseudo random number generation.
`fg_tb_dgen.vhd`	VHDL File provided with demonstration test bench. It contains logic for random data generation.
`fg_tb_dverif.vhd`	VHDL File provided with demonstration test bench. It contains logic for verifying the correctness of the FIFO Generator core data output.
`fg_tb_pctrl.vhd`	VHDL File provided with demonstration test bench. It contains the test bench control logic and some checks.
`fg_tb_synth.vhd`	VHDL File provided with demonstration test bench. This file has the instances and connections for the core and test bench modules.
`fg_tb_top.vhd`	VHDL File provided with demonstration test bench.This is the top file for the test bench which generates the clock and reset signals. It also checks the test bench status.

Back to Top

simulation/functional

The `functional` directory contains functional simulation scripts provided with the core.

Table 9-8: **Functional Directory**

Name	Description
`<project_directory>/<component_name>/simulation/functional`	
`simulate_mti.do`	A ModelSim macro file that compiles the HDL sources and runs the simulation.
`wave_mti.do`	A ModelSim macro file that opens a wave window and adds key signals to the wave viewer. This file is called by the simulate_mti.do file and is displayed after the simulation is loaded.
`simulate_isim.bat`	ISim macro file for Windows that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.
`simulate_isim.sh`	ISim macro file for Linux machines that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.
`wave_isim.tcl`	ISim macro file that opens a Wave window with top-level signals.
`simulate_ncsim.sh`	Linux shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using the Cadence IES simulator.
`wave_ncsim.sv`	The Cadence IES simulator macro file that opens a wave window and adds interesting signals to it. This macro is called by the simulate_ncsim.sh script.

Back to Top

simulation/timing

The `timing` directory contains functional simulation scripts provided with the core.

Table 9-9: **Timing Directory**

Name	Description
`<project_directory>/<component_name>/simulation/timing`	
`simulate_mti.do`	A ModelSim macro file that compiles the HDL sources and runs the simulation.
`wave_mti.do`	A ModelSim macro file that opens a wave window and adds key signals to the wave viewer. This file is called by the simulate_mti.do file and is displayed after the simulation is loaded.

Table 9-9: **Timing Directory *(Cont'd)***

Name	Description
`simulate_isim.bat`	ISim macro file for Windows that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.
`simulate_isim.sh`	ISim macro file for Linux that compiles the example design sources and the structural simulation model. The demonstration test bench then runs the functional simulation to completion.
`wave_isim.tcl`	ISim macro file that opens a Wave window with top-level signals.
`simulate_ncsim.sh`	Linux shell script that compiles the example design sources and the structural simulation model then runs the functional simulation to completion using the Cadence IES simulator.
`wave_ncsim.sv`	The Cadence IES simulator macro file that opens a wave window and adds interesting signals to it. This macro is called by the simulate_ncsim.sh script.
Back to Top	

Customizing and Generating the AXI4 Core

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

GUI

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

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

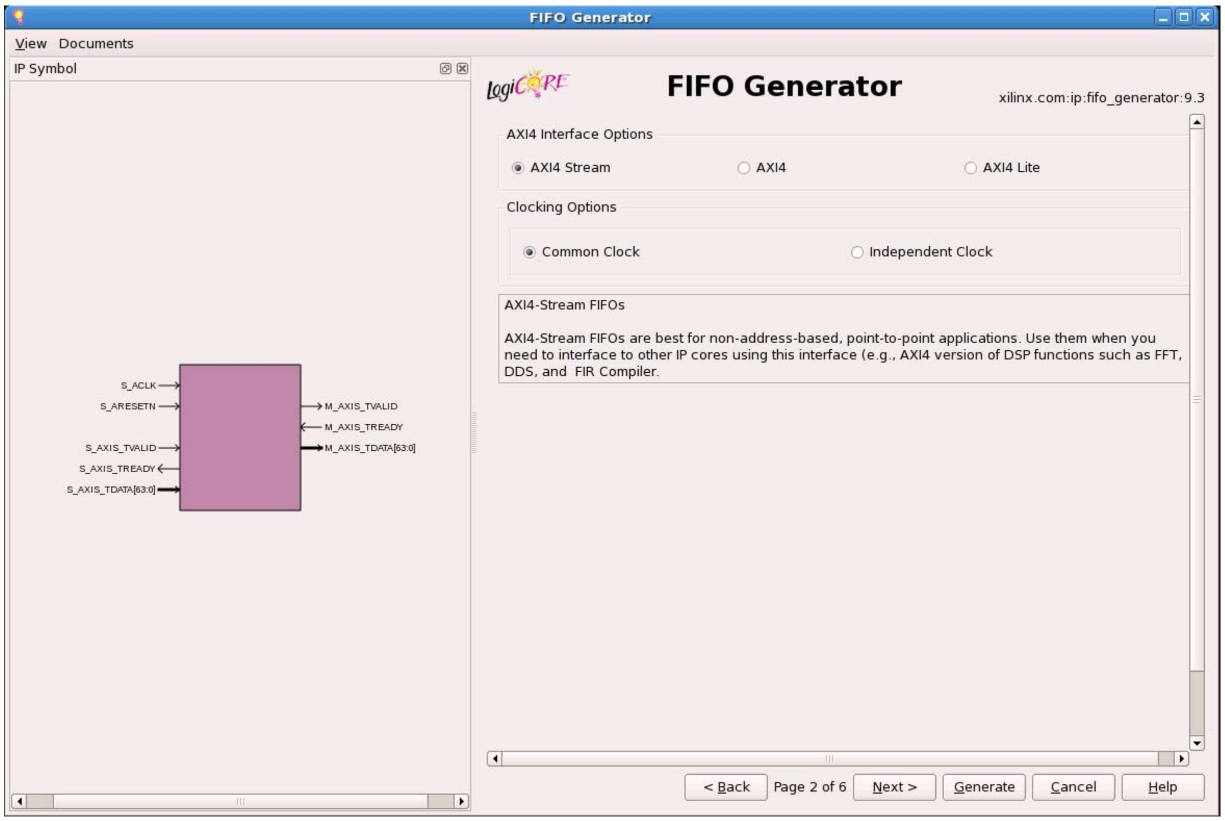
  For AXI4 and AXI4-Lite interfaces, FIFO Generator provides a separate page to configure each FIFO channel. For more details, see Easy Integration of Independent FIFOs for Read and Write Channels in Chapter 1.
- Summary

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

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

AXI4 Interface Selection

Figure 10-1 shows the AXI4 interface selection screen.



FIFO Generator

xilinx.com:ip:fifo_generator:9.3

AXI4 Interface Options

[✓ AXI4 Stream] AXI4 AXI4 Lite

Clocking Options

[✓ Common Clock] Independent Clock

AXI4-Stream FIFOs

AXI4-Stream FIFOs are best for non-address-based, point-to-point applications. Use them when you need to interface to other IP cores using this interface (e.g., AXI4 version of DSP functions such as FFT, DDS, and FIR Compiler.

Figure 10-1: **AXI4 Interface Selection Screen**

- **AXI4 Interface Options**

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

- **Clocking Options**

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

  The Independent Clock configuration enables the user to implement unique clock domains on the Write and Read ports. The FIFO Generator handles the synchronization between clock domains, placing no requirements on phase and frequency. When data buffering in a single clock domain is required, the FIFO Generator can be used to generate a core optimized for a single clock by selecting the Common Clocks option.

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

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

Performing Writes with Slave Clock Enable

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

Performing Reads with Master Clock Enable

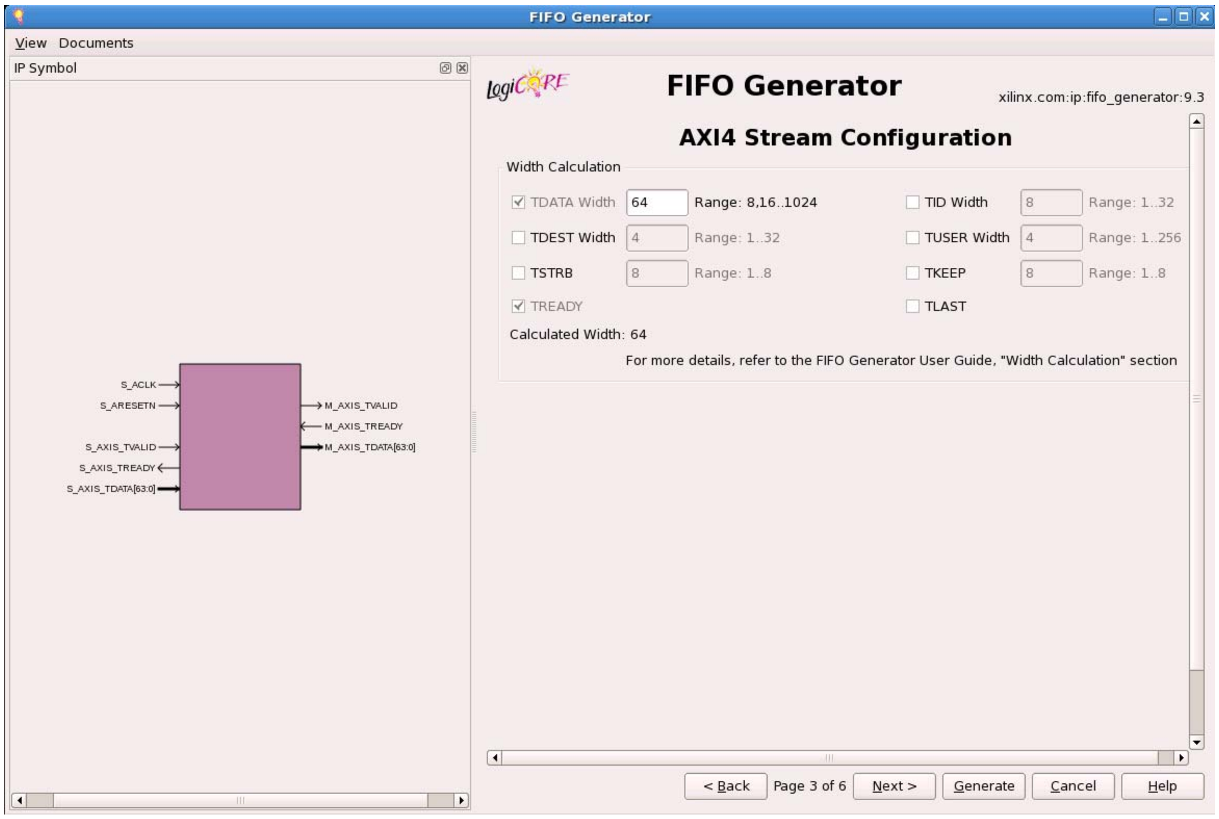
The Master Interface Clock Enable allows AXI4 Slave to operate at fractional rates of AXI4 Master Interface (or Read side) of the FIFO. The above timing diagram shows the AXI4 Slave operating at half the frequency of the FIFO AXI4 Master Interface. The Clock Enable in this case is Single Clock Wide,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

Width Calculation

The AXI4 FIFO Width is determined by aggregating all of the channel information signals in a channel. The channel information signals for AXI4-Stream, AXI4 and AXI4-Lite interfaces are listed in Table 10-1 and Table 10-2. GUI screens are available for configuring:

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

AXI4-Stream Width Calculation



Width Calculation

✓ TDATA Width	64	Range: 8,16..1024	TID Width	8	Range: 1..32
TDEST Width	4	Range: 1..32	TUSER Width	4	Range: 1..256
TSTRB	8	Range: 1..8	TKEEP	8	Range: 1..8
✓ TREADY			TLAST		

Calculated Width: 64

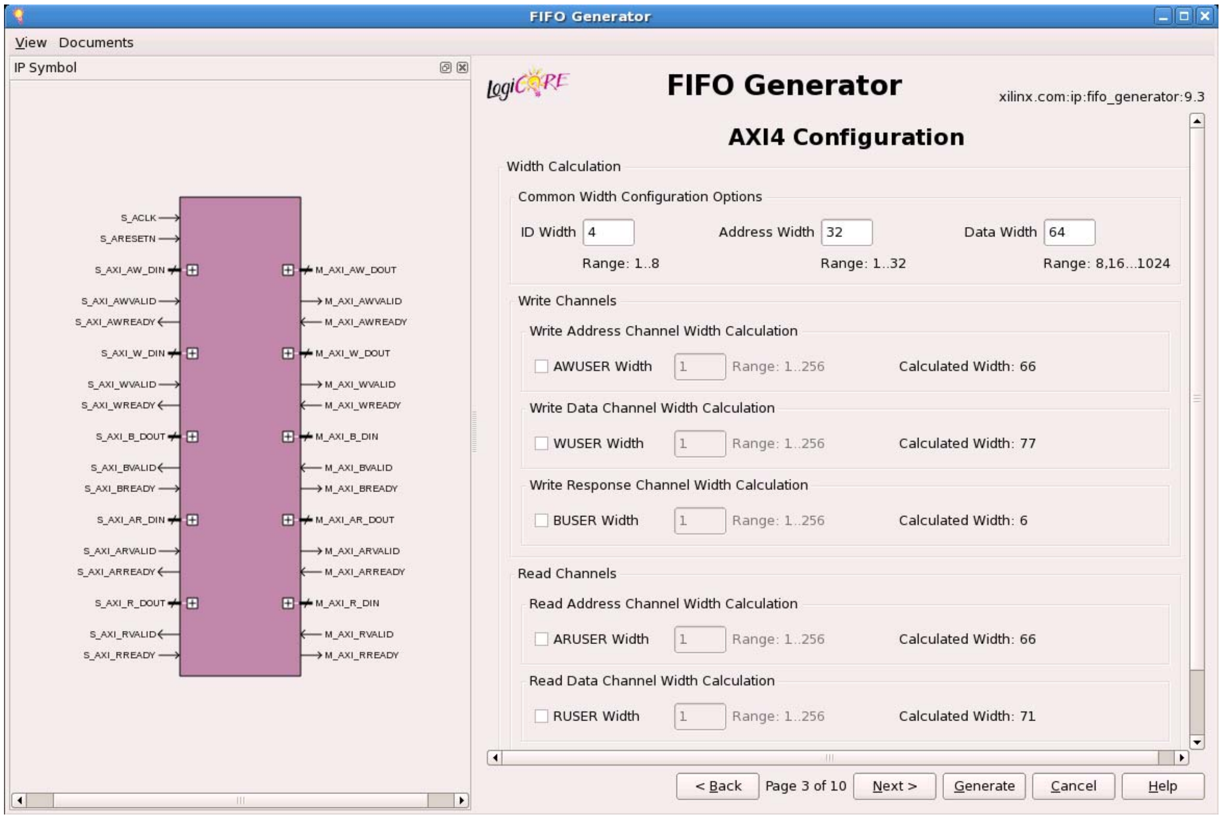
For more details, refer to the FIFO Generator User Guide, "Width Calculation" section

Figure 10-2: **AXI4-Stream Width Calculation Screen**

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

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

AXI4 Width Calculation



FIFO Generator

xilinx.com:ip:fifo_generator:9.3

AXI4 Configuration

Width Calculation

Common Width Configuration Options

ID Width 4 Range: 1..8

Address Width 32 Range: 1..32

Data Width 64 Range: 8,16...1024

Write Channels

Write Address Channel Width Calculation

AWUSER Width 1 Range: 1..256 Calculated Width: 66

Write Data Channel Width Calculation

WUSER Width 1 Range: 1..256 Calculated Width: 77

Write Response Channel Width Calculation

BUSER Width 1 Range: 1..256 Calculated Width: 6

Read Channels

Read Address Channel Width Calculation

ARUSER Width 1 Range: 1..256 Calculated Width: 66

Read Data Channel Width Calculation

RUSER Width 1 Range: 1..256 Calculated Width: 71

Figure 10-3: **AXI4 Width Calculation Screen**

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

For all the selected signals, the AXI4 FIFO width for the respective channel is determined by summing up the widths of signals in the particular channel, as shown in Table 10-1.

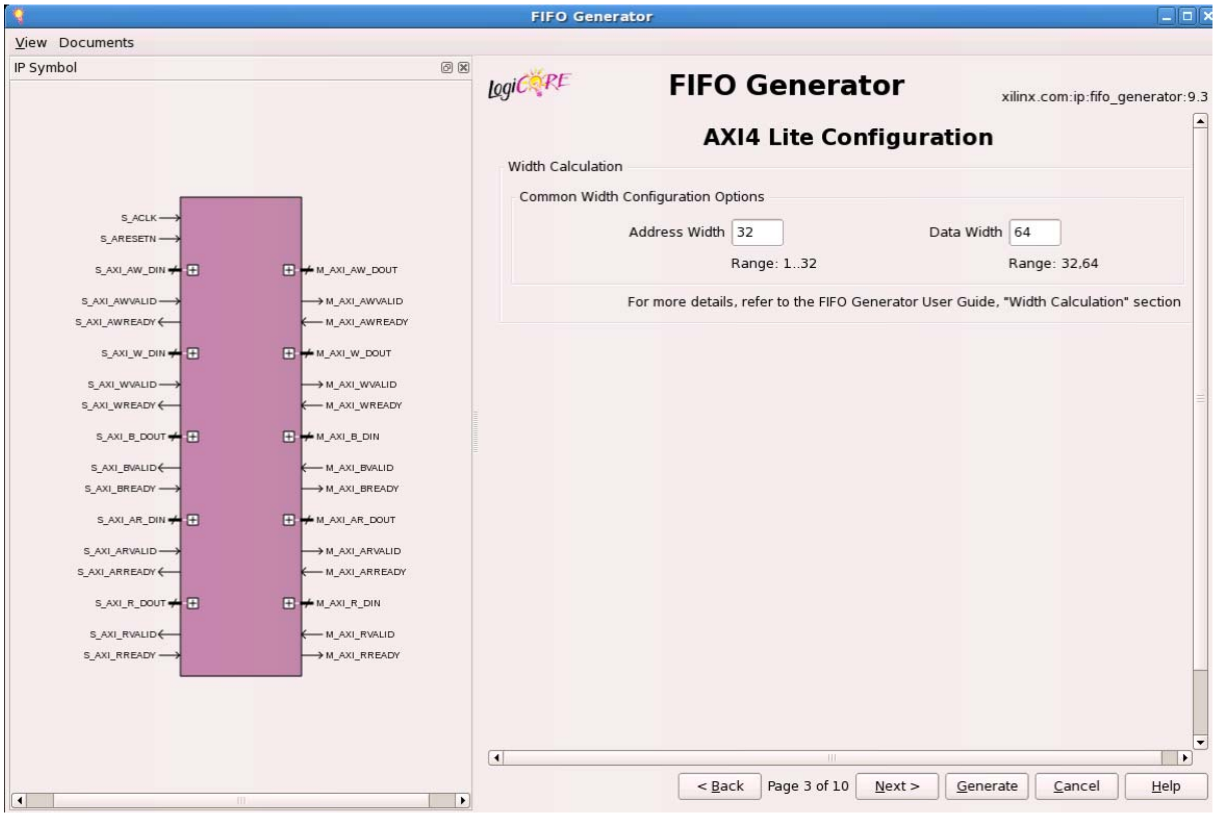
Table 10-1: **AXI4 Signals used in AXI FIFO Width Calculation**

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

Table 10-1: **AXI4 Signals used in AXI FIFO Width Calculation *(Cont'd)***

Write Address Channel	Read Address Channel	Write Data Channel	Read Data Channel	Write Resp Channel
AWCACHE[4:0]	ARCACHE[4:0]			
AWPROT[3:0]	ARPROT[3:0]			
AWREGION[3:0]	ARREGION[3:0]			
AWQOS[3:0]	ARQOS[3:0]			
AWUSER[m:0]	ARUSER[m:0]			

AXI4-Lite Width Calculation



FIFO Generator

xilinx.com:ip:fifo_generator:9.3

AXI4 Lite Configuration

Width Calculation

Common Width Configuration Options

Address Width 32 Range: 1..32

Data Width 64 Range: 32,64

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

Figure 10-4: **AXI4-Lite Width Calculation Screen**

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

AXI4-Lite FIFO width for the respective channel is determined by summing up the widths of all the signals in the particular channel, as shown in Table 10-2.

Table 10-2: **AXI4-Lite Width Calculation**

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

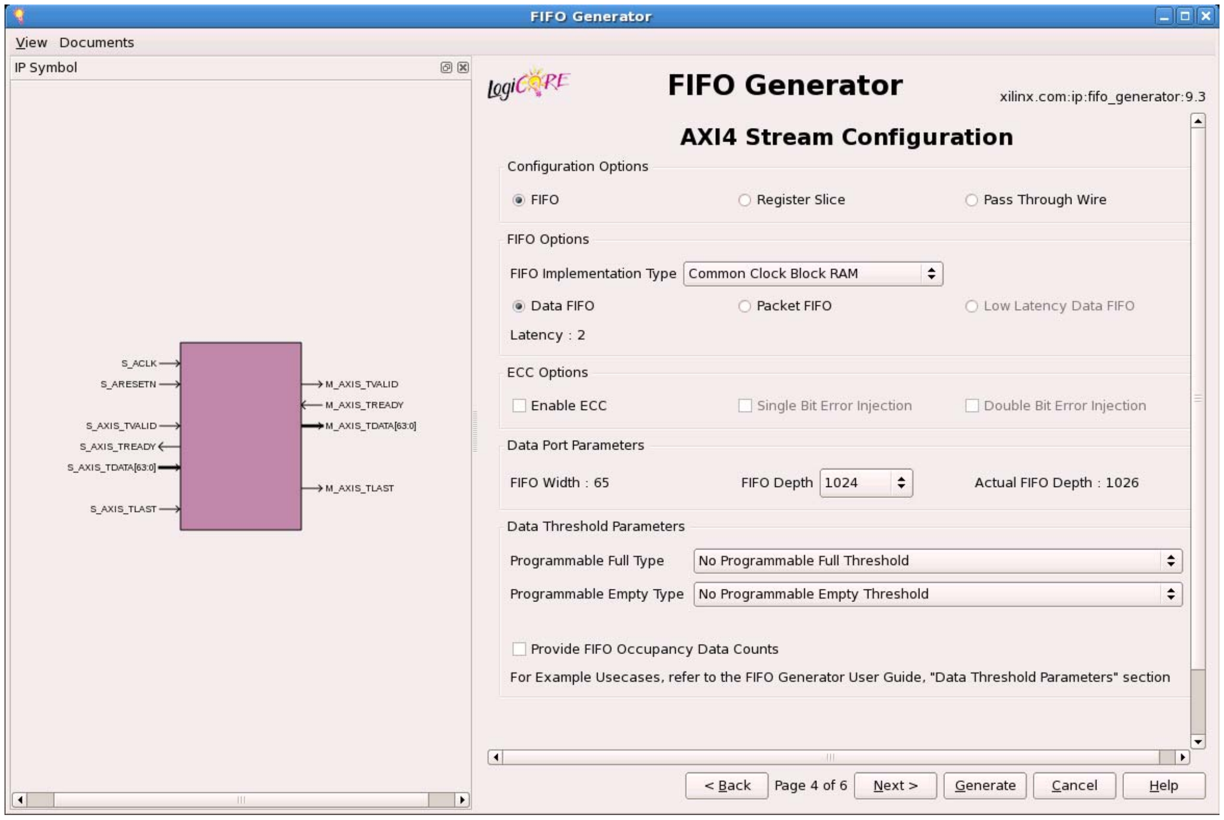
Default Settings

Table 10-3 shows the default settings for each AXI4 interface type.

Table 10-3: **AXI4 FIFO Default Settings**

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

FIFO Configurations



FIFO Generator xilinx.com:ip:fifo_generator:9.3

AXI4 Stream Configuration

Configuration Options

[✓ FIFO] Register Slice Pass Through Wire

FIFO Options

FIFO Implementation Type: Common Clock Block RAM

[✓ Data FIFO] Packet FIFO Low Latency Data FIFO

Latency : 2

ECC Options

Enable ECC Single Bit Error Injection Double Bit Error Injection

Data Port Parameters

FIFO Width : 65 FIFO Depth: 1024 Actual FIFO Depth : 1026

Data Threshold Parameters

Programmable Full Type: No Programmable Full Threshold

Programmable Empty Type: No Programmable Empty Threshold

Provide FIFO Occupancy Data Counts

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

Figure 10-5: **AXI4 FIFO Configurations Screen**

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

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

Memory Types

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

Error Injection and Correction (ECC)

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

For more details on Error Injection and Correction, see Built-in Error Correction Checking in Chapter 3.

FIFO Width

AXI4 FIFOs support symmetric Write and Read widths. The width of the AXI4 FIFO is determined based on the selected Interface Type (AXI4-Stream, AXI4 or AXI4-Lite), and the selected signals and configured signal widths within the given interface. The calculation of the FIFO Write Width is defined in Width Calculation, page 216.

FIFO Depth

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

Programmable Flags

This section includes details about the available programmable flags.

Programmable Full Type

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

Full Threshold Assert Value

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

Programmable Empty Type

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

Empty Threshold Assert Value

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

Data Threshold Parameters

This section includes details about data threshold parameters.

Occupancy Data Counts

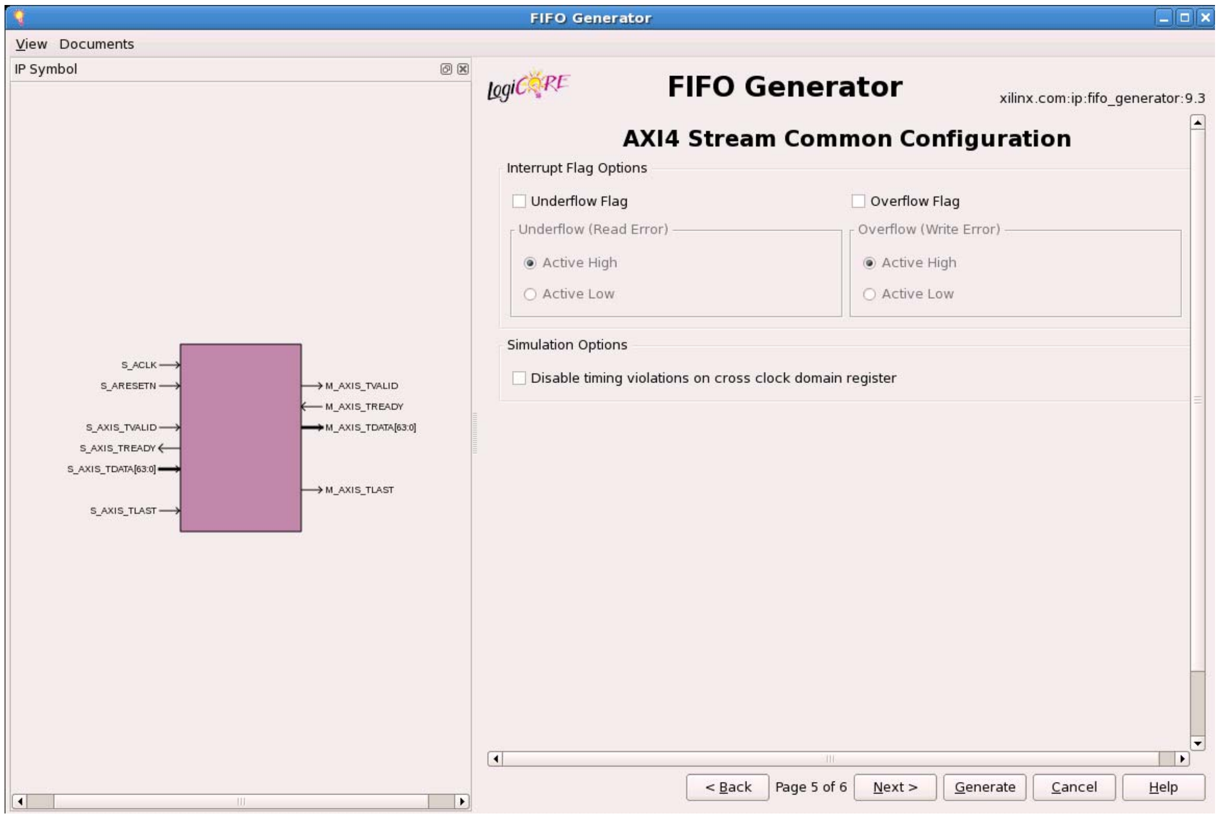
DATA_COUNT tracks the number of words in the FIFO. The width of the data count bus will be always be set to $\log_2$(FIFO depth)+1. In common clock mode, the AXI4 FIFO provides a single "Data Count" output. In independent clock mode, it provides Read Data Count and Write Data Count outputs.

For more details on Occupancy Data Counts, see First-Word Fall-Through Data Count in Chapter 3 and More Accurate Data Count (Use Extra Logic) in Chapter 3.

Examples for Data Threshold Parameters

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

Common Configurations



FIFO Generator xilinx.com:ip:fifo_generator:9.3

AXI4 Stream Common Configuration

Interrupt Flag Options

☐ Underflow Flag

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

☐ Overflow Flag

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

Simulation Options

☐ Disable timing violations on cross clock domain register

Figure 10-6: **AXI4 FIFO Common Configurations Screen**

Interrupt Flags

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

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

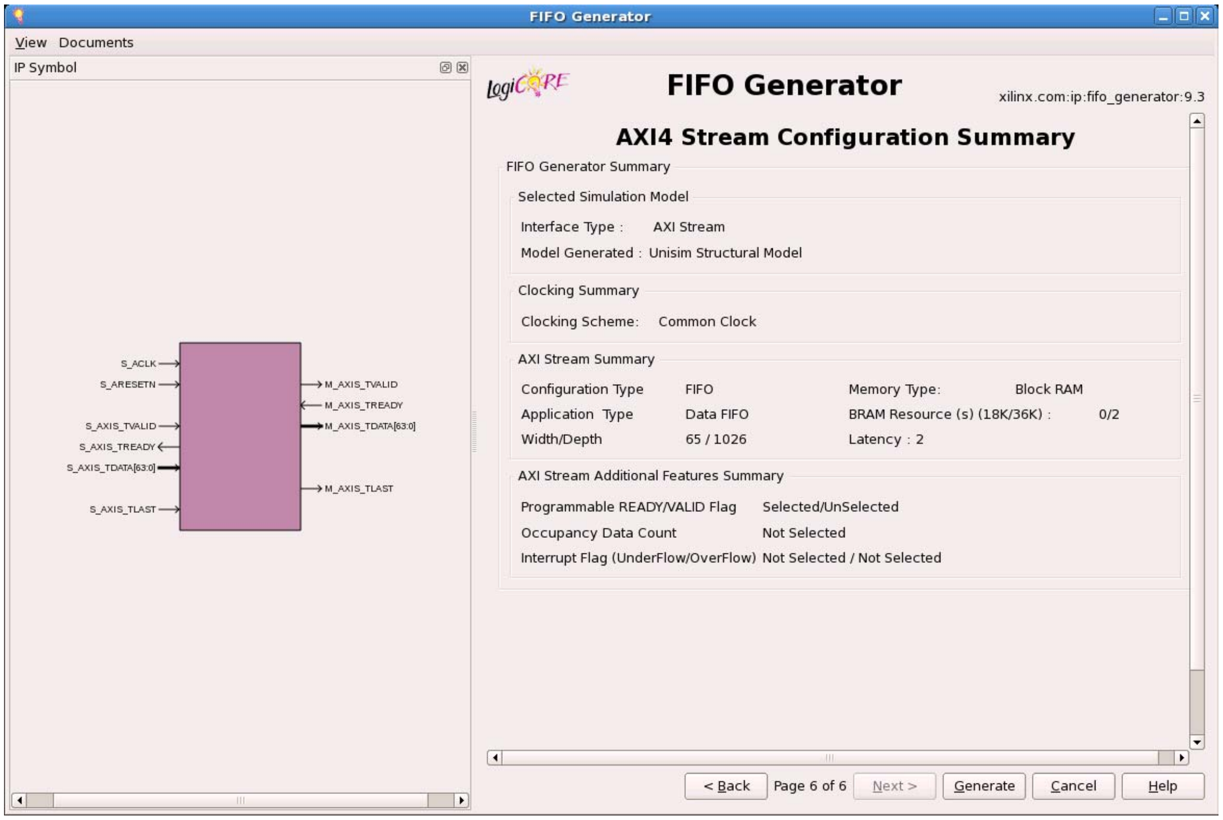
For more details on Overflow and Underflow Flags, see Underflow in Chapter 3 and Overflow in Chapter 3.

Summary

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

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

AXI4-Stream Summary



AXI4 Stream Configuration Summary

FIFO Generator Summary

Selected Simulation Model

Interface Type : AXI Stream
Model Generated : Unisim Structural Model

Clocking Summary

Clocking Scheme: Common Clock

AXI Stream Summary

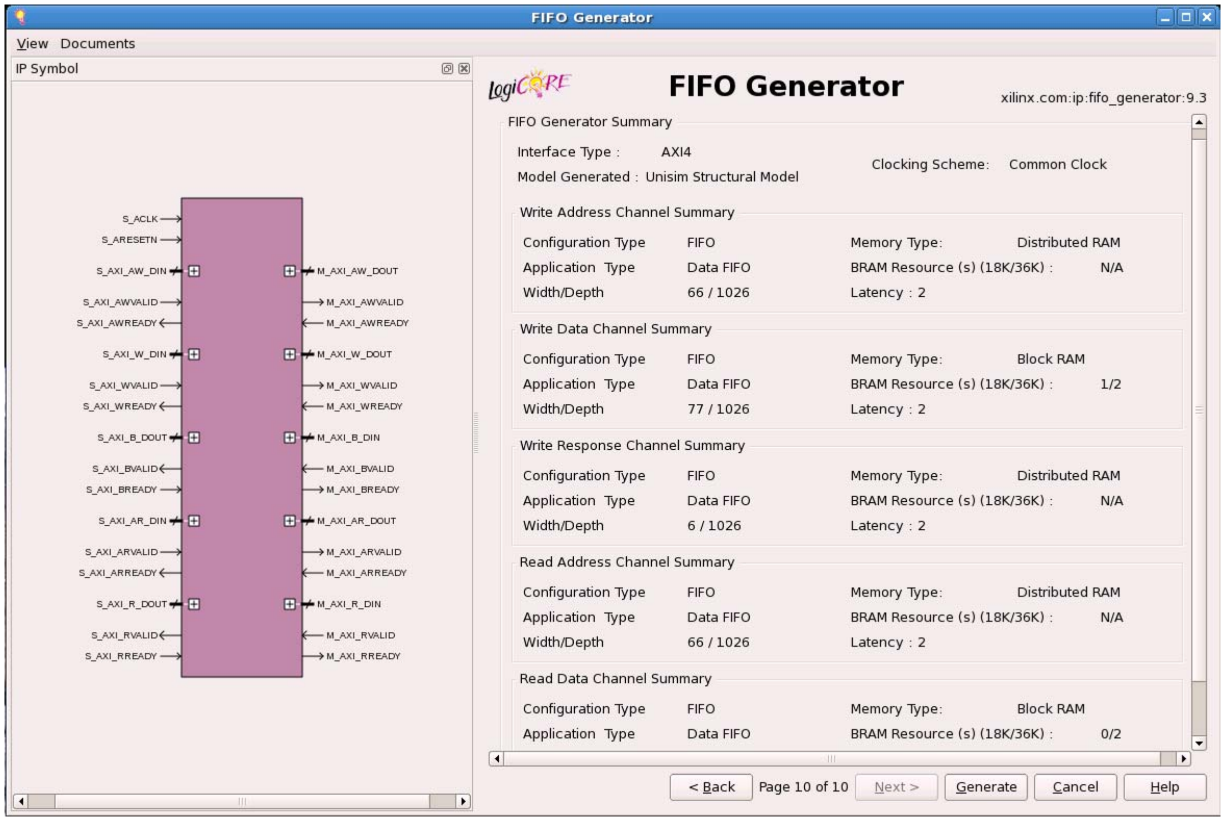
Configuration Type	FIFO	Memory Type:	Block RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	0/2
Width/Depth	65 / 1026	Latency : 2	

AXI Stream Additional Features Summary

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

Figure 10-7: **AXI4-Stream Summary Screen**

AXI4 and AXI4-Lite Summary



FIFO Generator

xilinx.com:ip:fifo_generator:9.3

FIFO Generator Summary

Interface Type : AXI4
Model Generated : Unisim Structural Model
Clocking Scheme: Common Clock

Write Address Channel Summary

Configuration Type	FIFO	Memory Type:	Distributed RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	N/A
Width/Depth	66 / 1026	Latency : 2	

Write Data Channel Summary

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

Write Response Channel Summary

Configuration Type	FIFO	Memory Type:	Distributed RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	N/A
Width/Depth	6 / 1026	Latency : 2	

Read Address Channel Summary

Configuration Type	FIFO	Memory Type:	Distributed RAM
Application Type	Data FIFO	BRAM Resource (s) (18K/36K) :	N/A
Width/Depth	66 / 1026	Latency : 2	

Read Data Channel Summary

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

Figure 10-8: **AXI4 / AXI4-Lite Summary Screen**

Parameter Values in the XCO File

Table 10-4 describes the AXI4 FIFO core parameters, including the XCO file value and the default settings.

Table 10-4: **AXI4 FIFO XCO Parameter Table**

Parameter Name	XCO File Values	Default GUI Settings
component_name	instance_name ASCII text starting with a letter and using the following character set: a-z, 0-9, and _	fifo_generator_v9_3
interface_type	Native AXI4	Native
axi_type	AXI4_Stream AXI4_Full, AXI4_Lite	AXI4_Stream

Table 10-4: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
enable_write_channel	True False	True
enable_read_channel	True False	True
clock_type_axi	Common_Clock Independent_Clock	Common_Clock
use_clock_enable[a]	True False	False
clock_enable_type[a]	Slave_Interface_Clock_Enable Master_Interface_Clock_Enable	Slave_Interface_Clock_Enable
id_width	1 - 8	8
axi_address_width	1 – 32	32
axi_data_width	2^3 - 2^9	64
enable_awuser	True False	False
enable_wuser	True False	False
enable_buser	True False	False
enable_aruser	True False	False
enable_ruser	True False	False
enable_tuser	True False	False
awuser_width	1 - 256	1
wuser_width	1 - 256	1
buser_width	1 - 256	1
aruser_width	1 - 256	1
ruser_width	1 - 256	1
tuser_width	1 - 256	4
enable_tdata	True False	True
enable_tdest	True False	False
enable_tid	True False	False

Table 10-4: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
enable_tkeep	True False	False
enable_tlast	True False	False
enable_tready	True False	True
enable_tstrobe	True False	False
tdata_width	2^3 - 2^9	64
tdest_width	1 - 4	4
tid_width	1 - 8	8
tkeep_width	tdata_width/8	8
tstrb_width	tdata_width/8	8
axis_type	FIFO	FIFO
wach_type	FIFO	FIFO
wdch_type	FIFO	FIFO
wrch_type	FIFO	FIFO
rach_type	FIFO	FIFO
rdch_type	FIFO	FIFO
fifo_implementation_type_axis	Common_Clock_Block_RAM Common_Clock_Distributed_RAM Independent_Clock_Block_RAM Independent_Clock_Distributed_RA M	Common_Clock_Block_RAM
fifo_implementation_type_rach		Common_Clock_Distributed_RAM
fifo_implementation_type_rdch		Common_Clock_Block_RAM
fifo_implementation_type_wach		Common_Clock_Distributed_RAM
fifo_implementation_type_wdch		Common_Clock_Block_RAM
fifo_implementation_type_wrch		Common_Clock_Distributed_RAM
fifo_application_type_axis	Data_FIFO	Data_FIFO
fifo_application_type_rach	Data_FIFO	Data_FIFO
fifo_application_type_rdch	Data_FIFO	Data_FIFO
fifo_application_type_wach	Data_FIFO	Data_FIFO
fifo_application_type_wdch	Data_FIFO	Data_FIFO
fifo_application_type_wrch	Data_FIFO	Data_FIFO
enable_ecc_axis	True False	False

Table 10-4: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
enable_ecc_rach	True False	False
enable_ecc_rdch	True False	False
enable_ecc_wach	True False	False
enable_ecc_wdch	True False	False
enable_ecc_wrch	True False	False
inject_sbit_error_axis	True False	False
inject_sbit_error_rach	True False	False
inject_sbit_error_rdch	True False	False
inject_sbit_error_wach	True False	False
inject_sbit_error_wdch	True False	False
inject_sbit_error_wrch	True False	False
inject_dbit_error_axis	True False	False
inject_dbit_error_rach	True False	False
inject_dbit_error_rdch	True False	False
inject_dbit_error_wach	True False	False
inject_dbit_error_wdch	True False	False
inject_dbit_error_wrch	True False	False
input_depth_axis	2^4 - 2^{16}	1024
input_depth_rach	2^4 - 2^{16}	16
input_depth_rdch	2^4 - 2^{16}	1024
input_depth_wach	2^4 - 2^{16}	16
input_depth_wdch	2^4 - 2^{16}	1024

Table 10-4: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
input_depth_wrch	2^4 - 2^{16}	16
enable_data_counts_axis	True False	False
enable_data_counts_rach	True False	False
enable_data_counts_rdch	True False	False
enable_data_counts_wach	True False	False
enable_data_counts_wdch	True False	False
enable_data_counts_wrch	True False	False
enable_handshake_flag_options_axis	True False	False
programmable_full_type_axis	No_Programmable_Full_Threshold Single_Programmable_Full_ Threshold_Constant Single_Programmable_Full_ Threshold_Input_Port	No_Programmable_Full_Threshol d
programmable_full_type_rach		No_Programmable_Full_Threshol d
programmable_full_type_rdch		No_Programmable_Full_Threshol d
programmable_full_type_wach		No_Programmable_Full_Threshol d
programmable_full_type_wdch		No_Programmable_Full_Threshol d
programmable_full_type_wrch		No_Programmable_Full_Threshol d
full_threshold_assert_value_axis	5 - 65535	1023
full_threshold_assert_value_rach	5 - 65535	1023
full_threshold_assert_value_rdch	5 - 65535	1023
full_threshold_assert_value_wach	5 - 65535	1023
full_threshold_assert_value_wdch	5 - 65535	1023
full_threshold_assert_value_wrch	full_threshold_assert_value_wrch	1023

Table 10-4: **AXI4 FIFO XCO Parameter Table *(Cont'd)***

Parameter Name	XCO File Values	Default GUI Settings
programmable_empty_type_axis	No_Programmable_Full_Threshold Single_Programmable_ Empty _Threshold_Constant Single_Programmable_ Empty _Threshold_Input_Port	No_Programmable_Full_Threshol d
programmable_empty_type_rach		No_Programmable_Full_Threshol d
programmable_empty_type_rdch		No_Programmable_Full_Threshol d
programmable_empty_type_wach		No_Programmable_Full_Threshol d
programmable_empty_type_wdch		No_Programmable_Full_Threshol d
programmable_empty_type_wrch		No_Programmable_Full_Threshol d
empty_threshold_assert_value_axis	4 - 65534	1022
empty_threshold_assert_value_rach	4 - 65534	1022
empty_threshold_assert_value_rdch	4 - 65534	1022
empty_threshold_assert_value_wach	4 - 65534	1022
empty_threshold_assert_value_wdch	4 - 65534	1022
empty_threshold_assert_value_wrch	4 - 65534	1022
underflow_flag_axi	True False	False
underflow_sense_axi	Active_High Active_Low	Active_High
overflow_flag_axi	True False	False
overflow_sense_axi	Active_High Active_Low	Active_High
enable_common_overflow	True False	False
enable_common_underflow[a]	True False	False
disable_timing_violations_axi	True False	False
add_ngc_constraint_axi[a]	True False	False

a. Feature presently not supported

Output Generation

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

Constraining the Core

This chapter contains details about constraints for the FIFO Generator core when implemented with the ISE Design Suite. See Setup and Hold Time Violations in Chapter 4 for additional constraint details.

Required Constraints

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

Device, Package, and Speed Grade Selections

See IP Facts for supported devices.

Clock Frequencies

There are no clock frequency constraints.

Clock Management

There are no additional clock management constraints for this core.

Detailed Example Design

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

Directory and File Contents

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

Example Design

Figure 12-1 shows the configuration of the example design.

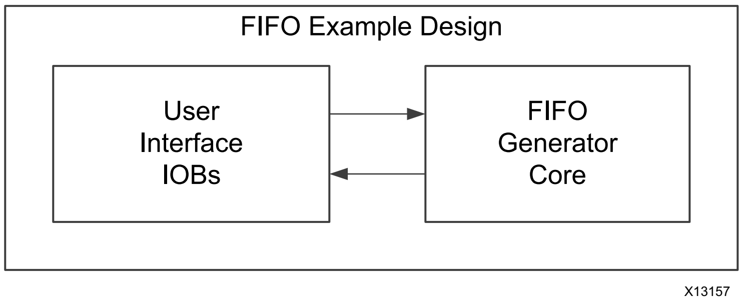


Figure 12-1: **Example Design Configuration**

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 with the CORE Generator software netlist model during implementation for timing simulation or XST netlist/behavioral model for the functional simulation.
- Global clock buffers for top-level port clock signals.

Demonstration Test Bench

Figure 12-2 shows a block diagram of the demonstration test bench.

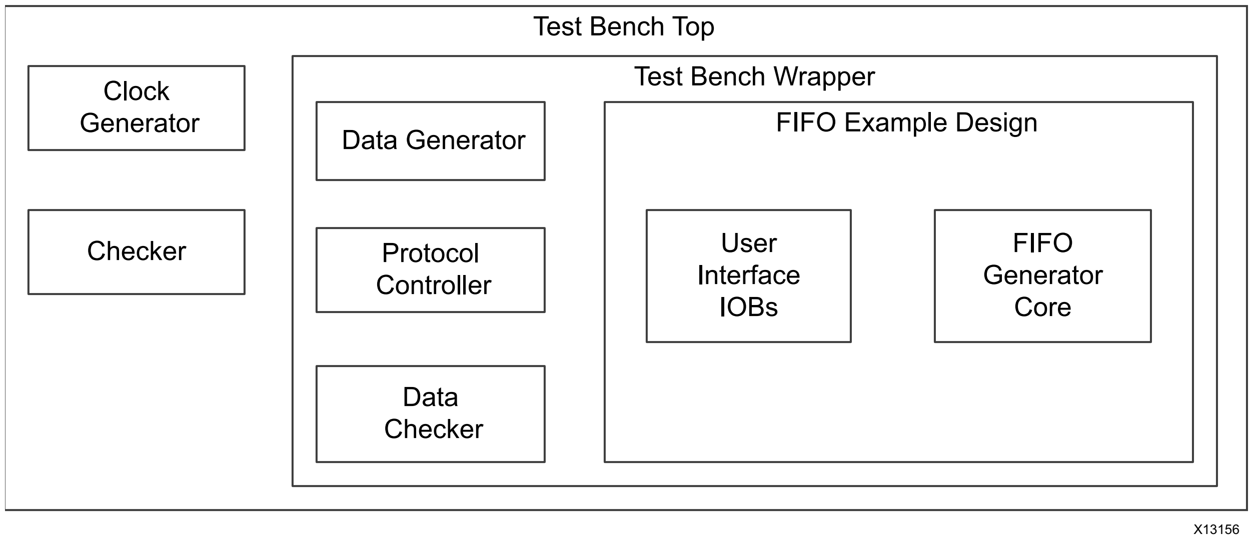


Figure 12-2: **Demonstration Test Bench**

Test Bench Functionality

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

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

Core with Native Interface

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

- Input clock signals are generated.
- A reset is applied to the example design.
- Pseudo random data is generated and given as input to FIFO data input port.
- Data on DOUT port of the FIFO generator core is cross checked using another pseudo random generator with same seed as data input generator.
- Core is exercised for two full and empty conditions.

- Full/almost_full and empty/almost_empty flags are checked.

Core with AXI4 Interface

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

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

Customizing the Demonstration Test Bench

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

Changing the Data/Stimulus

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

Changing the Test Bench Run Time

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

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

Implementation

The implementation script is either a shell script (.sh) or batch file (.bat) that processes the example design through the Xilinx tool flow. It is located at:

Linux

```
<project_dir>/<component_name>/implement/implement.sh
```

Windows

```
<project_dir>/<component_name>/implement/implement.bat
```

The implement script performs these steps:

- Synthesizes the HDL example design files using XST.
- Runs NGDBuild to consolidate the core netlist and the example design netlist into the NGD file containing the entire design.
- Maps the design to the target technology.
- Place-and-routes the design on the target device.
- Performs static timing analysis on the routed design using Timing Analyzer (TRCE).
- Generates a bitstream.
- Enables Netgen to run on the routed design to generate a VHDL or Verilog netlist (as appropriate for the Design Entry project setting) and timing information in the form of SDF files.

The Xilinx tool flow generates several output and report files. These are saved in the following directory which is created by the implement script:

```
<project_dir>/<component_name>/implement/results
```

Simulation

This section contains details about the test scripts included in the example design.

Functional Simulation

The test scripts are ModelSim macros that automate the simulation of the test bench. They are available from the following location:

```
<project_dir>/<component_name>/simulation/functional/
```

The test script performs these tasks:

- Compiles the behavioral model/structural UNISIM simulation model.
- Compiles HDL Example Design source code.
- Compiles the demonstration test bench.
- Starts a simulation of the test bench.
- Opens a Wave window and adds signals of interest (`wave_mti.do`).
- Runs the simulation to completion.

Timing Simulation

The test scripts are ModelSim macros that automate the simulation of the test bench. They are located in:

```
<project_dir>/<component_name>/simulation/timing/
```

The test script performs these tasks:

- Compiles the SIMPRIM based gate level netlist simulation model.
- Compiles the demonstration test bench.
- Starts a simulation of the test bench.
- Opens a Wave window and adds signals of interest (`wave_mti.do`).
- Runs the simulation to completion.

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
```

SECTION IV: APPENDICES

Verification, Compliance, and Interoperability

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

Simulation

The FIFO Generator has been tested with Xilinx ISE® software v14.4, Xilinx Vivado software v2012.4, Xilinx ISIM/XSIM, Cadence Incisive Enterprise Simulator (IES), Synopsys VCS and VCS MX and Mentor Graphics ModelSim simulator.

Migrating

This appendix provides step-by-step instructions for migrating existing designs containing instances of legacy FIFO cores (Synchronous FIFO v5.x and Asynchronous FIFO v6.x) to the latest version of the FIFO Generator.

For information on migration from the ISE Design Suite to the Vivado Design Suite, see UG911, *Vivado Design Suite Migration Methodology Guide*.

Note: For all new designs, Xilinx recommends that you use the most recent version of the FIFO Generator core for your FIFO function requirements.

Migration Overview

The FIFO Generator Migration Kit, uses a perl script to automate the FIFO core migration process. The Migration Kit can be obtained from:

https://secure.xilinx.com/webreg/clickthrough.do?cid=151908&license=RefDesLicense

Differences from Legacy Cores

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

The FIFO Generator core can generate synchronous and asynchronous FIFOs, and can also leverage the built-in FIFOs in the Artix™-7. Kintex™-7, Virtex®-7, Virtex-6, Virtex-5 and Virtex-4 FPGA device families. In addition, the FIFO Generator core benefits from the use of the Block Memory Generator, which optimizes block memory resource utilization and enhances the overall performance of the FIFO Generator.

Core Compatibility

The FIFO Generator core is not backward-compatible with the Synchronous FIFO and Asynchronous FIFO cores in the following ways:

- FIFO Generator uses different port names. For example, the names of the reset and handshaking flags are different.
- The XCO files for the previous generation memory cores are NOT compatible with the FIFO Generator.
- Behavioral differences between Synchronous FIFO cores generated by FIFO Generator and those generated by Synchronous FIFO include the following:
    - The maximum data count width for FIFO Generator is one bit less than that for Synchronous FIFO and the behavior of DATA_COUNT is different. For example, for the Synchronous FIFO, when the FIFO depth is 256, the maximum data count width is 9 bits. However, for FIFO Generator, the maximum data count width is only 8 bits. When the FIFO is full:
      `DATA_COUNT = 1 0000 0000` in Synchronous FIFO
      `DATA_COUNT = 0000 0000` in FIFO Generator
    - In this case, when using the maximum data count width, you can connect the full flag as the most significant bit of DATA_COUNT to make the FIFO Generator backward- compatible.
    - If both data count widths are equal, the behavior of DATA COUNT is different. When the FIFO is full, `DATA_COUNT` is `11111111` in Synchronous FIFO and `DATA_COUNT` is `00000000` in FIFO Generator. In this case, the FIFO Generator is not backward compatible.
- Behavioral differences between Asynchronous FIFO cores generated by FIFO Generator and those generated by Asynchronous FIFO include the following:
    - The falling edge of the `EMPTY` and `ALMOST_EMPTY` flags may occur one clock cycle later in the FIFO Generator in response to a write operation.
    - The falling edge of the `FULL` and `ALMOST_FULL` flags may occur one clock cycle later in the FIFO Generator in response to a read operation.
    - `WR_COUNT` takes longer to respond to a read, and `RD_COUNT` takes longer to respond to a write in FIFO Generator.
    - Reset requirements differ between the two cores.
    - The default FULL reset value differs between the two cores—set the FULL reset value to '1' for backward-compatibility.

Differences in port names and XCO parameters for legacy cores (Synchronous and Asynchronous FIFOs) are defined in Converting the XCO File, page 246 and Modifying the Instantiations of the Old Core, page 261.

Obsolete Features for Legacy Cores

Create RPM

The FIFO Generator core does not support Relationally Placed Macros (RPMs), which were supported in the Asynchronous FIFO core.

Migrating a Design

The Migration Kit provides a Perl script to help automate the process of converting existing Synchronous or Asynchronous FIFO cores to the latest FIFO Generator core version.

The migration script automates all the manual steps, from converting the XCO file to modifying the instantiation of the old core. In addition, this script can be used to only automate specific steps, making it also useful when following the Manual Migration Process, page 245.

Migration Script

Note: ISE® software v12.2 and later require a CGP file when CORE Generator is run in command line mode. This version of the Migration Script comes with a sample CGP file (`coregen.cgp`) which the user can modify according to their requirements. The modified CGP file should be kept in the user directory where the XCO file and instantiation files are located and must be named `coregen.cgp` to work with the migration script.

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

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

About the Migration Script

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

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

- Old XCO core configuration files (created by the GUI when the FIFO core was generated)

- HDL source file(s) containing the core instantiations (VHDL or Verilog)

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

- Old FIFO XCO files to FIFO Generator v9.3 XCO files (use -x option).
- Generate the new netlists and convert the instantiations of the Old FIFO cores in your HDL source code to latest FIFO Generator core instances by running the CORE Generator software (-x and -m options).

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

Output Products

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

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

Using the Migration Script

To start the Migration Script, type the commands specific to your environment at the command prompt.

Linux

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

Windows

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

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

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

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

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

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

Usage Examples

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

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

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

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

Manual Migration Process

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

1. Convert the XCO file.

   The XCO file is used by the CORE Generator to determine a core's configuration. The format for the FIFO Generator core differs from the Synchronous and Asynchronous FIFO cores.

   Note: If you plan to generate a new FIFO Generator core via the CORE Generator GUI, skip this step.
2. Generate the FIFO Generator core.
3. Modify the instantiations of the old core. As the final step in the migration, you must update all instantiations of the old cores in your HDL source code to reference the new core. This includes changing the port names, as explained in Modifying the

Instantiations of the Old Core, page 261. Differences from Legacy Cores, page 241 discusses whether design modifications are needed to compensate for obsolete features.

Converting the XCO File

Table B-1 and Table B-2 define the mapping between the XCO file parameters for the Synchronous and Asynchronous FIFO cores and the XCO for the latest FIFO Generator core. You may need to change your design to compensate for obsolete features. See Core Compatibility, page 242 for information about difference between the old and new cores.

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

For example, for the legacy Synchronous FIFO core, change the line:

```
SELECT Synchronous_FIFO family Xilinx,_Inc. 5.0
```

to:

```
SELECT FIFO_Generator family Xilinx,_Inc. 9.3
```

For the legacy Asynchronous FIFO core, change the line:

```
SELECT Asynchronous_FIFO family Xilinx,_Inc. 6.1
```

to:

```
SELECT FIFO_Generator family Xilinx,_Inc. 9.3
```

Table B-1: **XCO Parameter Mapping: Synchronous FIFO Cores**

Synchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Synchronous FIFO)
component_name	component_name	No change required

memory_type	fifo_implementation	Common_Clock_Block_RAM = block_memory
		Common_Clock_Distributed_RAM = distributed_memory

data_width	input_data_width
	output_data_width
		input_data_width = data_width
		output_data_width = data_width

fifo_depth	input_depth
	output_depth
		input_depth = fifo_depth
		output_depth = fifo_depth

write_acknowledge_flag	write_acknowledge_flag	No change required
write_acknowledge_sense	write_acknowledge_sense	No change required
write_error_flag	overflow_flag	Direct replacement
write_error_sense	overflow_sense	Direct replacement
read_acknowledge_flag	valid_flag	Direct replacement

Table B-1: **XCO Parameter Mapping: Synchronous FIFO Cores *(Cont'd)***

Synchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Synchronous FIFO)
read_acknowledge_sense	valid_sense	Direct replacement
read_error_flag	underflow_flag	Direct replacement
read_error_sense	underflow_sense	Direct replacement
data_count	data_count	No change required
data_count_width	data_count_width	No change required
-	interface_type	Add this parameter and set the value to Native
-	reset_pin	Add this parameter and set the value to True
-	reset_type	Add this parameter and set the value to Synchronous_Reset
-	use_dout_reset	True (for backward compatibility)
-	almost_empty_flag	Add this parameter and set the value to False
-	almost_full_flag	Add this parameter and set the value to False
-	disable_timing_violations	Add this parameter and set the value to False
-	dout_reset_value	Add this parameter and set the value to 0
-	empty_threshold_assert_value	Add this parameter and set the value to 2
-	empty_threshold_negate_value	Add this parameter and set the value to 3
-	enable_ecc	Add this parameter and set the value to False
-	enable_reset_synchronization	Add this parameter and set the value to False
-	full_flags_reset_value	1 (for backward compatibility)
-	full_threshold_assert_value	Add this parameter and set the value to 4
-	full_threshold_negate_value	Add this parameter and set the value to 3
-	inject_sbit_error	Add this parameter and set the value to False
-	inject_dbit_error	Add this parameter and set the value to False
-	performance_options	Add this parameter and set the value to Standard_FIFO
-	programmable_empty_type	Add this parameter and set the value to No_Programmable_Empty_Threshold
-	programmable_full_type	Add this parameter and set the value to No_Programmable_Full_Threshold
-	read_clock_frequency	Add this parameter and set the value to 1
-	read_data_count	Add this parameter and set the value to False
-	read_data_count_width	Add this parameter and set the value to 1
-	use_embedded_registers	Add this parameter and set the value to False
-	use_extra_logic	Add this parameter and set the value to False
-	write_clock_frequency	Add this parameter and set the value to 1

Table B-1: **XCO Parameter Mapping: Synchronous FIFO Cores *(Cont'd)***

Synchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Synchronous FIFO)
-	write_data_count	Add this parameter and set the value to False
-	write_data_count_width	Add this parameter and set the value to 1
-	axi_type	Add this parameter and set the value to AXI4_Stream
-	enable_write_channel	Add this parameter and set the value to False
-	enable_read_channel	Add this parameter and set the value to False
-	clock_type_axi	Add this parameter and set the value to Common_Clock
-	use_clock_enable	Add this parameter and set the value to False
-	id_width	Add this parameter and set the value to 1
-	axi_address_width	Add this parameter and set the value to 1
-	axi_data_width	Add this parameter and set the value to 8
-	enable_awuser	Add this parameter and set the value to False
-	enable_wuser	Add this parameter and set the value to False
-	enable_buser	Add this parameter and set the value to False
-	enable_aruser	Add this parameter and set the value to False
-	enable_ruser	Add this parameter and set the value to False
-	enable_tuser	Add this parameter and set the value to False
-	awuser_width	Add this parameter and set the value to 1
-	wuser_width	Add this parameter and set the value to 1
-	buser_width	Add this parameter and set the value to 1
-	aruser_width	Add this parameter and set the value to 1
-	ruser_width	Add this parameter and set the value to 1
-	tuser_width	Add this parameter and set the value to 1
-	enable_tdata	Add this parameter and set the value to False
-	enable_tdest	Add this parameter and set the value to False
-	enable_tid	Add this parameter and set the value to False
-	enable_tkeep	Add this parameter and set the value to False
-	enable_tlast	Add this parameter and set the value to False
-	enable_tready	Add this parameter and set the value to False
-	enable_tstrobe	Add this parameter and set the value to False
-	tdata_width	Add this parameter and set the value to 8
-	tdest_width	Add this parameter and set the value to 1
-	tid_width	Add this parameter and set the value to 1
-	tkeep_width	Add this parameter and set the value to 1

Table B-1: **XCO Parameter Mapping: Synchronous FIFO Cores *(Cont'd)***

Synchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Synchronous FIFO)
-	tstrb_width	Add this parameter and set the value to 1
-	axis_type	Add this parameter and set the value to FIFO
-	wach_type	Add this parameter and set the value to FIFO
-	wdch_type	Add this parameter and set the value to FIFO
-	wrch_type	Add this parameter and set the value to FIFO
-	rach_type	Add this parameter and set the value to FIFO
-	rdch_type	Add this parameter and set the value to FIFO
-	fifo_implementation_type_axis	Add this parameter and set the value to Common_Clock_Block_RAM
-	fifo_implementation_type_rach	Add this parameter and set the value to Common_Clock_Distributed_RAM
-	fifo_implementation_type_rdch	Add this parameter and set the value to Common_Clock_Block_RAM
-	fifo_implementation_type_wach	Add this parameter and set the value to Common_Clock_Distributed_RAM
-	fifo_implementation_type_wdch	Add this parameter and set the value to Common_Clock_Block_RAM
-	fifo_implementation_type_wrch	Add this parameter and set the value to Common_Clock_Distributed_RAM
-	fifo_application_type_axis	Add this parameter and set the value to Data_FIFO
-	fifo_application_type_rach	Add this parameter and set the value to Data_FIFO
-	fifo_application_type_rdch	Add this parameter and set the value to Data_FIFO
-	fifo_application_type_wach	Add this parameter and set the value to Data_FIFO
-	fifo_application_type_wdch	Add this parameter and set the value to Data_FIFO
-	fifo_application_type_wrch	Add this parameter and set the value to Data_FIFO
-	enable_ecc_axis	Add this parameter and set the value to False
-	enable_ecc_rach	Add this parameter and set the value to False
-	enable_ecc_rdch	Add this parameter and set the value to False
-	enable_ecc_wach	Add this parameter and set the value to False
-	enable_ecc_wdch	Add this parameter and set the value to False
-	enable_ecc_wrch	Add this parameter and set the value to False
-	inject_sbit_error_axis	Add this parameter and set the value to False
-	inject_sbit_error_rach	Add this parameter and set the value to False
-	inject_sbit_error_rdch	Add this parameter and set the value to False
-	inject_sbit_error_wach	Add this parameter and set the value to False
-	inject_sbit_error_wdch	Add this parameter and set the value to False

Table B-1: **XCO Parameter Mapping: Synchronous FIFO Cores *(Cont'd)***

Synchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Synchronous FIFO)
-	inject_sbit_error_wrch	Add this parameter and set the value to False
-	inject_dbit_error_axis	Add this parameter and set the value to False
-	inject_dbit_error_rach	Add this parameter and set the value to False
-	inject_dbit_error_rdch	Add this parameter and set the value to False
-	inject_dbit_error_wach	Add this parameter and set the value to False
-	inject_dbit_error_wdch	Add this parameter and set the value to False
-	inject_dbit_error_wrch	Add this parameter and set the value to False
-	input_depth_axis	Add this parameter and set the value to 1024
-	input_depth_rach	Add this parameter and set the value to 16
-	input_depth_rdch	Add this parameter and set the value to 1024
-	input_depth_wach	Add this parameter and set the value to 16
-	input_depth_wdch	Add this parameter and set the value to 1024
-	input_depth_wrch	Add this parameter and set the value to 16
-	enable_data_counts_axis	Add this parameter and set the value to False
-	enable_data_counts_rach	Add this parameter and set the value to False
-	enable_data_counts_rdch	Add this parameter and set the value to False
-	enable_data_counts_wach	Add this parameter and set the value to False
-	enable_data_counts_wdch	Add this parameter and set the value to False
-	enable_data_counts_wrch	Add this parameter and set the value to False
-	enable_prog_flags_axis	Add this parameter and set the value to False
-	enable_prog_flags_rach	Add this parameter and set the value to False
-	enable_prog_flags_rdch	Add this parameter and set the value to False
-	enable_prog_flags_wach	Add this parameter and set the value to False
-	enable_prog_flags_wdch	Add this parameter and set the value to False
-	enable_prog_flags_wrch	Add this parameter and set the value to False
-	programmable_full_type_axis	Add this parameter and set the value to Full
-	programmable_full_type_rach	Add this parameter and set the value to Full
-	programmable_full_type_rdch	Add this parameter and set the value to Full
-	programmable_full_type_wach	Add this parameter and set the value to Full
-	programmable_full_type_wdch	Add this parameter and set the value to Full
-	programmable_full_type_wrch	Add this parameter and set the value to Full
-	full_threshold_assert_value_axis	Add this parameter and set the value to 5
-	full_threshold_assert_value_rach	Add this parameter and set the value to 5

Table B-1: **XCO Parameter Mapping: Synchronous FIFO Cores *(Cont'd)***

Synchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Synchronous FIFO)
-	full_threshold_assert_value_rdch	Add this parameter and set the value to 5
-	full_threshold_assert_value_wach	Add this parameter and set the value to 5
-	full_threshold_assert_value_wdch	Add this parameter and set the value to 5
-	full_threshold_assert_value_wrch	Add this parameter and set the value to 5
-	programmable_empty_type_axis	Add this parameter and set the value to Empty
-	programmable_empty_type_rach	Add this parameter and set the value to Empty
-	programmable_empty_type_rdch	Add this parameter and set the value to Empty
-	programmable_empty_type_wach	Add this parameter and set the value to Empty
-	programmable_empty_type_wdch	Add this parameter and set the value to Empty
-	programmable_empty_type_wrch	Add this parameter and set the value to Empty
-	empty_threshold_assert_value_axis	Add this parameter and set the value to 5
-	empty_threshold_assert_value_rach	Add this parameter and set the value to 4
-	empty_threshold_assert_value_rdch	Add this parameter and set the value to 4
-	empty_threshold_assert_value_wach	Add this parameter and set the value to 4
-	empty_threshold_assert_value_wdch	Add this parameter and set the value to 4
-	empty_threshold_assert_value_wrch	Add this parameter and set the value to 4
-	underflow_flag_axi	Add this parameter and set the value to False
-	underflow_sense_axi	Add this parameter and set the value to Active_High
-	overflow_flag_axi	Add this parameter and set the value to False
-	overflow_sense_axi	Add this parameter and set the value to Active_High
-	disable_timing_violations_axi	Add this parameter and set the value to False

Table B-2: **XCO Parameter Mapping: Asynchronous FIFO Cores**

Asynchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Asynchronous FIFO)
component_name	component_name	No change required

memory_type	fifo_implementation	Independent_Clocks_Block_RAM = block
		Independent_Clocks_Distributed_RAM = distributed

data_width	input_data_width
	output_data_width
		input_data_width = input_data_width
		output_data_width = input_data_width

fifo_depth	input_depth
	output_depth
		input_depth = fifo_depth + 1
		output_depth = fifo_depth + 1

almost_empty_flag	almost_empty_flag	No change required
almost_full_flag	almost_full_flag	No change required
write_acknowledge	write_acknowledge_flag	Direct replacement
write_acknowledge_sense	write_acknowledge_sense	No change required
write_error	overflow_flag	Direct replacement
write_error_sense	overflow_sense	Direct replacement
read_acknowledge	valid_flag	Direct replacement
read_acknowledge_sense	valid_sense	Direct replacement
read_error	underflow_flag	Direct replacement
read_error_sense	underflow_sense	Direct replacement
write_count	write_data_count	Direct replacement
write_count_width	write_data_count_width	Direct replacement
read_count	read_data_count	Direct replacement
read_count_width	read_data_count_width	Direct replacement
create_rpm	-	Not supported
-	interface_type	Add this parameter and set the value to Native
-	reset_pin	Add this parameter and set the value to True
-	reset_type	Add this parameter and set the value to Asynchronous_Reset
-	use_dout_reset	True (for backward compatibility)
-	data_count	Add this parameter and set the value to False
-	data_count_width	Add this parameter and set the value to 1
-	almost_empty_flag	Add this parameter and set the value to False
-	almost_full_flag	Add this parameter and set the value to False
-	disable_timing_violations	Add this parameter and set the value to False
-	dout_reset_value	Add this parameter and set the value to 0
-	empty_threshold_assert_value	Add this parameter and set the value to 2

Table B-2: **XCO Parameter Mapping: Asynchronous FIFO Cores *(Cont'd)***

Asynchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Asynchronous FIFO)
-	empty_threshold_negate_value	Add this parameter and set the value to 3
-	enable_ecc	Add this parameter and set the value to False
-	enable_reset_synchronization	Add this parameter and set the value to False
-	full_flags_reset_value	1 (for backward compatibility)
-	full_threshold_assert_value	Add this parameter and set the value to 4
-	full_threshold_negate_value	Add this parameter and set the value to 3
-	inject_sbit_error	Add this parameter and set the value to False
-	inject_dbit_error	Add this parameter and set the value to False
-	performance_options	Add this parameter and set the value to Standard_FIFO
-	programmable_empty_type	Add this parameter and set the value to No_Programmable_Empty_Threshold
-	programmable_full_type	Add this parameter and set the value to No_Programmable_Full_Threshold
-	read_clock_frequency	Add this parameter and set the value to 1
-	use_embedded_registers	Add this parameter and set the value to False
-	use_extra_logic	Add this parameter and set the value to False
-	write_clock_frequency	Add this parameter and set the value to 1
-	axi_type	Add this parameter and set the value to AXI4_Stream
-	enable_write_channel	Add this parameter and set the value to False
-	enable_read_channel	Add this parameter and set the value to False
-	clock_type_axi	Add this parameter and set the value to Common_Clock
-	use_clock_enable	Add this parameter and set the value to False
-	id_width	Add this parameter and set the value to 1
-	axi_address_width	Add this parameter and set the value to 1
-	axi_data_width	Add this parameter and set the value to 8
-	enable_awuser	Add this parameter and set the value to False
-	enable_wuser	Add this parameter and set the value to False
-	enable_buser	Add this parameter and set the value to False
-	enable_aruser	Add this parameter and set the value to False
-	enable_ruser	Add this parameter and set the value to False
-	enable_tuser	Add this parameter and set the value to False
-	awuser_width	Add this parameter and set the value to 1
-	wuser_width	Add this parameter and set the value to 1

Table B-2: **XCO Parameter Mapping: Asynchronous FIFO Cores *(Cont'd)***

Asynchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Asynchronous FIFO)
-	buser_width	Add this parameter and set the value to 1
-	aruser_width	Add this parameter and set the value to 1
-	ruser_width	Add this parameter and set the value to 1
-	tuser_width	Add this parameter and set the value to 1
-	enable_tdata	Add this parameter and set the value to False
-	enable_tdest	Add this parameter and set the value to False
-	enable_tid	Add this parameter and set the value to False
-	enable_tkeep	Add this parameter and set the value to False
-	enable_tlast	Add this parameter and set the value to False
-	enable_tready	Add this parameter and set the value to False
-	enable_tstrobe	Add this parameter and set the value to False
-	tdata_width	Add this parameter and set the value to 8
-	tdest_width	Add this parameter and set the value to 1
-	tid_width	Add this parameter and set the value to 1
-	tkeep_width	Add this parameter and set the value to 1
-	tstrb_width	Add this parameter and set the value to 1
-	axis_type	Add this parameter and set the value to FIFO
-	wach_type	Add this parameter and set the value to FIFO
-	wdch_type	Add this parameter and set the value to FIFO
-	wrch_type	Add this parameter and set the value to FIFO
-	rach_type	Add this parameter and set the value to FIFO
-	rdch_type	Add this parameter and set the value to FIFO
-	fifo_implementation_type_axis	Add this parameter and set the value to Common_Clock_Block_RAM
-	fifo_implementation_type_rach	Add this parameter and set the value to Common_Clock_Distributed_RAM
-	fifo_implementation_type_rdch	Add this parameter and set the value to Common_Clock_Block_RAM
-	fifo_implementation_type_wach	Add this parameter and set the value to Common_Clock_Distributed_RAM
-	fifo_implementation_type_wdch	Add this parameter and set the value to Common_Clock_Block_RAM
-	fifo_implementation_type_wrch	Add this parameter and set the value to Common_Clock_Distributed_RAM
-	fifo_application_type_axis	Add this parameter and set the value to Data_FIFO

Table B-2: **XCO Parameter Mapping: Asynchronous FIFO Cores *(Cont'd)***

Asynchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Asynchronous FIFO)
-	fifo_application_type_rach	Add this parameter and set the value to Data_FIFO
-	fifo_application_type_rdch	Add this parameter and set the value to Data_FIFO
-	fifo_application_type_wach	Add this parameter and set the value to Data_FIFO
-	fifo_application_type_wdch	Add this parameter and set the value to Data_FIFO
-	fifo_application_type_wrch	Add this parameter and set the value to Data_FIFO
-	enable_ecc_axis	Add this parameter and set the value to False
-	enable_ecc_rach	Add this parameter and set the value to False
-	enable_ecc_rdch	Add this parameter and set the value to False
-	enable_ecc_wach	Add this parameter and set the value to False
-	enable_ecc_wdch	Add this parameter and set the value to False
-	enable_ecc_wrch	Add this parameter and set the value to False
-	inject_sbit_error_axis	Add this parameter and set the value to False
-	inject_sbit_error_rach	Add this parameter and set the value to False
-	inject_sbit_error_rdch	Add this parameter and set the value to False
-	inject_sbit_error_wach	Add this parameter and set the value to False
-	inject_sbit_error_wdch	Add this parameter and set the value to False
-	inject_sbit_error_wrch	Add this parameter and set the value to False
-	inject_dbit_error_axis	Add this parameter and set the value to False
-	inject_dbit_error_rach	Add this parameter and set the value to False
-	inject_dbit_error_rdch	Add this parameter and set the value to False
-	inject_dbit_error_wach	Add this parameter and set the value to False
-	inject_dbit_error_wdch	Add this parameter and set the value to False
-	inject_dbit_error_wrch	Add this parameter and set the value to False
-	input_depth_axis	Add this parameter and set the value to 1024
-	input_depth_rach	Add this parameter and set the value to 16
-	input_depth_rdch	Add this parameter and set the value to 1024
-	input_depth_wach	Add this parameter and set the value to 16
-	input_depth_wdch	Add this parameter and set the value to 1024
-	input_depth_wrch	Add this parameter and set the value to 16
-	enable_data_counts_axis	Add this parameter and set the value to False
-	enable_data_counts_rach	Add this parameter and set the value to False

Table B-2: **XCO Parameter Mapping: Asynchronous FIFO Cores *(Cont'd)***

Asynchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Asynchronous FIFO)
-	enable_data_counts_rdch	Add this parameter and set the value to False
-	enable_data_counts_wach	Add this parameter and set the value to False
-	enable_data_counts_wdch	Add this parameter and set the value to False
-	enable_data_counts_wrch	Add this parameter and set the value to False
-	enable_prog_flags_axis	Add this parameter and set the value to False
-	enable_prog_flags_rach	Add this parameter and set the value to False
-	enable_prog_flags_rdch	Add this parameter and set the value to False
-	enable_prog_flags_wach	Add this parameter and set the value to False
-	enable_prog_flags_wdch	Add this parameter and set the value to False
-	enable_prog_flags_wrch	Add this parameter and set the value to False
-	programmable_full_type_axis	Add this parameter and set the value to Full
-	programmable_full_type_rach	Add this parameter and set the value to Full
-	programmable_full_type_rdch	Add this parameter and set the value to Full
-	programmable_full_type_wach	Add this parameter and set the value to Full
-	programmable_full_type_wdch	Add this parameter and set the value to Full
-	programmable_full_type_wrch	Add this parameter and set the value to Full
-	full_threshold_assert_value_axis	Add this parameter and set the value to 5
-	full_threshold_assert_value_rach	Add this parameter and set the value to 5
-	full_threshold_assert_value_rdch	Add this parameter and set the value to 5
-	full_threshold_assert_value_wach	Add this parameter and set the value to 5
-	full_threshold_assert_value_wdch	Add this parameter and set the value to 5
-	full_threshold_assert_value_wrch	Add this parameter and set the value to 5
-	programmable_empty_type_axis	Add this parameter and set the value to Empty
-	programmable_empty_type_rach	Add this parameter and set the value to Empty
-	programmable_empty_type_rdch	Add this parameter and set the value to Empty
-	programmable_empty_type_wach	Add this parameter and set the value to Empty
-	programmable_empty_type_wdch	Add this parameter and set the value to Empty
-	programmable_empty_type_wrch	Add this parameter and set the value to Empty
-	empty_threshold_assert_value_axis	Add this parameter and set the value to 5
-	empty_threshold_assert_value_rach	Add this parameter and set the value to 4

Table B-2: **XCO Parameter Mapping: Asynchronous FIFO Cores *(Cont'd)***

Asynchronous FIFO XCO Parameter	FIFO Generator XCO Parameter	Description of Conversion (FIFO Generator = Asynchronous FIFO)
-	empty_threshold_assert_value_rdch	Add this parameter and set the value to 4
-	empty_threshold_assert_value_wach	Add this parameter and set the value to 4
-	empty_threshold_assert_value_wdch	Add this parameter and set the value to 4
-	empty_threshold_assert_value_wrch	Add this parameter and set the value to 4
-	underflow_flag_axi	Add this parameter and set the value to False
-	underflow_sense_axi	Add this parameter and set the value to Active_High
-	overflow_flag_axi	Add this parameter and set the value to False
-	overflow_sense_axi	Add this parameter and set the value to Active_High
-	disable_timing_violations_axi	Add this parameter and set the value to False

Generate the FIFO Generator Core

Generate a new FIFO Generator netlist. Note that you must have ISE 14.4 installed on your system. The newly generated netlist (NGC) file replaces the old netlist file.

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

Parameterizing the FIFO Generator GUI

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

To open the FIFO Generator core, do the following:

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

Table B-3 compares the GUI parameters between the Synchronous and Asynchronous FIFO cores and the FIFO Generator cores. This table helps you choose the appropriate options when creating a new core using the FIFO Generator GUI.

Table B-3: **GUI Parameter Comparison**

Synchronous/ Asynchronous FIFO	FIFO Generator	Functionality of GUI Parameter (FIFO Generator= Synchronous/ Asynchronous FIFO)
Block Memory Distributed Memory	• Common Clock (CLK), Block RAM • Common Clock (CLK), Distributed RAM • Common Clock (CLK), Shift Register • Common Clock (CLK), Built-in FIFO • Independent Clocks (RD_CLK, WR_CLK), Block RAM • Independent Clocks (RD_CLK, WR_CLK), Distributed RAM • Independent Clocks (RD_CLK, WR_CLK), Built-in FIFO	Synchronous • Common Clock (CLK), Block RAM=Block Memory • Common Clock (CLK), Distributed RAM=Distributed Memory Asynchronous • Independent Clocks (RD_CLK, WR_CLK), Block RAM=Block Memory • Independent Clocks (RD_CLK, WR_CLK), Distributed RAM=Distributed Memory
-	Read Mode • Standard FIFO • First-Word Fall-Through	Synchronous and Asynchronous FIFO Generators only implement Standard FIFO. First-word fall-through ensures that there is always a valid data in the output port when fifo is not empty. The first-word fall-through (FWFT) provides the ability to look ahead to the next word available from the FIFO without issuing a read operation.
-	Built-in FIFO Options • Read Clock Frequency • Write Clock Frequency	Set Read, Write Clock frequencies. This option is only available for built-in FIFOs.
Data Port Parameters • Input Data Width • FIFO Depth	Data Port Parameters • Write Width • Write Depth • Read Width • Read Depth	These parameters define the FIFO size. For a Synchronous FIFO generated by FIFO Generator, Read Width and Read Depth are not available. Write Width=Input Data Width. Write Depth=FIFO Depth. For an Asynchronous FIFO generated by FIFO Generator, Read Depth is automatically calculated from the other three parameters. Write Depth=FIFO Depth+1, Read Depth=FIFO Depth+1. For FIFO Generator, actual FIFO Write Depth=FIFO Depth=Write Depth-1; actual FIFO Read Depth=FIFO Depth=Read Depth-1.
-	Implementation Options • Enable ECC • Use Embedded Registers in BRAM or FIFO (when possible)	Enable Virtex-6, Virtex-5 and Virtex-4 FPGA-specific features. These options are only available for block RAM and built-in based FIFOs.

Table B-3: **GUI Parameter Comparison *(Cont'd)***

Synchronous/ Asynchronous FIFO	FIFO Generator	Functionality of GUI Parameter (FIFO Generator= Synchronous/ Asynchronous FIFO)
Asynchronous • Almost Full Flag • Almost Empty Flag	Optional Flags • Almost Full Flag • Almost Empty Flag	Available for all FIFOs except built-in FIFOs. These parameters control generation of ALMOST FULL and ALMOST EMPTY flags. These flags indicate that FIFO is almost full or almost empty.
Handshaking Options • Write Acknowledge Flag • Write Error Flag • Read Acknowledge Flag • Read Error Flag (Active High or Active Low)	Handshaking Options • Write Acknowledge Flag • Overflow Flag • Valid Flag (Read Acknowledge) • Underflow Flag (Read Error)	These parameters control generation of FIFO status flags.
-	Initialization • Reset Pin • Asynchronous Reset • Synchronous Reset ◦ Full Flags Reset Value • Use Dout Reset • Enable Reset Synchronization	When Reset Pin is selected, reset is generated in input ports. Otherwise, reset is always tied to zero. Asynchronous Reset is supported for both Asynchronous and Synchronous FIFOs while Synchronous is only available in Synchronous FIFO. Synchronous Reset behavior is equal to SINIT in Synchronous FIFO Generator. Full flags reset value determine the value of full flags (FULL, ALMOST_FULL, PROG_FULL) during asynchronous reset. Set the value to '1' for backward-compatibility. Use dout reset to determine if the DOUT output resets to a specified value when the reset signal is asserted. Set to true for backward compatibility. Enable Reset Synchronization determines the use of internal reset synchronization logic. Set to true for backward compatibility.

Table B-3: **GUI Parameter Comparison *(Cont'd)***

Synchronous/ Asynchronous FIFO	FIFO Generator	Functionality of GUI Parameter (FIFO Generator= Synchronous/ Asynchronous FIFO)
-	Programmable Flags • Programmable Full Type	Programmable Full Type provides the following options: • No_Programmable_Full_ Threshold • Single_Programmable_Full_ Threshold_Constant • Multiple_Programmable_Full_ Threshold_Constants • Single_Programmable_Full_ Threshold_Input_Port • Multiple_Programmable_Full_ Threshold_Input_Ports
	- full_threshold_assert_value	- Full threshold assert value is used to set the upper threshold value for the programmable full flag.
	- full_threshold_negate_value	- Full Threshold Negate is used to set the lower threshold value for the programmable full flag.
	• Programmable Empty Type	Programmable Empty Type provides the following options: • No_Programmable_Empty_ Threshold • Single_Programmable_Empty_ Threshold_Constant • Multiple_Programmable_Empty_ Threshold_Constants • Single_Programmable_Empty_ Threshold_Input_Port • Multiple_Programmable_Empty_ Threshold_Input_Ports
	- empty_threshold_assert_value	- Empty Threshold Assert is used to set the lower threshold value for the programmable empty flag.
	- empty_threshold_negate_value	- Empty Threshold Negate is used to set the upper threshold value for the programmable empty flag.
Synchronous • Data Count • Data Count Width Asynchronous • Write Count • Write Count Width • Read Count • Read Count Width	Common Clock • Data Count ◦ Data Count Width Independent Clock • Write Data Count ◦ Write Data Count Width • Read Data Count ◦ Read Data Count Width	These parameters control generation of DATA COUNT.

Table B-3: **GUI Parameter Comparison *(Cont'd)***

Synchronous/ Asynchronous FIFO	FIFO Generator	Functionality of GUI Parameter (FIFO Generator= Synchronous/ Asynchronous FIFO)
Asynchronous • Layout (Create RPM)	-	Enable to locate the FIFO according to design attributes. When this option is selected, the Asynchronous FIFO is generated using Relationally Placed Macros (RPMs).
-	Error Injection • Single Bit Error Injection • Double Bit Error Injection	Enable Virtex-6 FPGA-specific features. These options are only available for block RAM and built-in based FIFOs.

Modifying the Instantiations of the Old Core

For each memory core instantiation, do the following:

1. Change the name of the module. (Only necessary if component name of the core has changed.)
2. Change the port names. For port name conversions, see Tables B-4 and B-5.
3. Modify your design, if necessary, to compensate for obsolete features. See Core Compatibility, page 242 for information about difference between old and new cores.

Table B-4: **Port Name Mapping: Asynchronous FIFO Cores**

Legacy Asynchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
DIN[N:0]	Available	DIN[N:0]	Available	Same	Available
WR_EN	Available	WR_EN	Available	Same	Available
WR_CLK	Available	WR_CLK	Available	Same	Available
RD_EN	Available	RD_EN	Available	Same	Available
RD_CLK	Available	RD_CLK	Available	Same	Available
AINIT	Available	RST	Optional	Asynchronous reset	Available
FULL	Available	FULL	Available	Same	Available
ALMOST_FULL	Optional	ALMOST_FULL	Available	Same	Available
-	-	PROG_FULL	Optional	-	Optional
-	-	PROG_FULL_THRESH	Optional	-	Optional
-	-	PROG_FULL_THRESH_ASSERT	Optional	-	Optional
-	-	PROG_FULL_THRESH_NEGATE	Optional	-	Optional

Table B-4: **Port Name Mapping: Asynchronous FIFO Cores** ***(Cont'd)***

Legacy Asynchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
WR_COUNT[W:0]	Optional	WR_DATA_COUNT[D:0]	Optional	Direct replacement	Optional
WR_ACK	Optional	WR_ACK	Optional	Same	Optional
WR_ERR	Optional	OVERFLOW	Optional	Direct replacement	Optional
DOUT[N:0]	Available	DOUT[M:0]	Available	Same	Optional
EMPTY	Available	EMPTY	Available	Same	Optional
ALMOST_EMPTY	Optional	ALMOST_EMPTY	Optional	Same	Optional
-	-	PROG_EMPTY	Optional	-	Optional
-	-	PROG_EMPTY_THRESH	Optional	-	Optional
-	-	PROG_EMPTY_THRESH_ASSERT	Optional	-	Optional
-	-	PROG_EMPTY_THRESH_NEGATE	Optional	-	Optional
RD_COUNT[R:0]	Optional	RD_DATA_COUNT[C:0]	Optional	Direct replacement	Optional
RD_ACK	Optional	VALID	Optional	Direct replacement	Optional
RD_ERR	Optional	UNDERFLOW	Optional	Direct replacement	Optional
-	-	SBITERR	Optional	-	Optional
-	-	DBITERR	Optional	-	Optional
-	-	INJECTSBITERR	Optional	-	Optional
-	-	INJECTDBITERR	Optional	-	Optional
-	-	M_ACLK	Optional	-	Optional
-	-	S_ACLK	Optional	-	Optional
-	-	S_ARESETN	Optional	-	Optional
-	-	M_AXIS_TVALID	Optional	-	Optional
-	-	M_AXIS_TREADY	Optional	-	Optional
-	-	M_AXIS_TDATA[m-1:0]	Optional	-	Optional
-	-	M_AXIS_TSTRB[m/8-1:0]	Optional	-	Optional
-	-	M_AXIS_TKEEP[m/8-1:0]	Optional	-	Optional
-	-	M_AXIS_TLAST	Optional	-	Optional
-	-	M_AXIS_TID[m:0]	Optional	-	Optional
-	-	M_AXIS_TDEST[m:0]	Optional	-	Optional
-	-	M_AXIS_TUSER[m:0]	Optional	-	Optional

Table B-4: **Port Name Mapping: Asynchronous FIFO Cores *(Cont'd)***

Legacy Asynchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	S_AXIS_TVALID	Optional	-	Optional
-	-	S_AXIS_TREADY	Optional	-	Optional
-	-	S_AXIS_TDATA[m-1:0]	Optional	-	Optional
-	-	S_AXIS_TSTRB[m/8-1:0]	Optional	-	Optional
-	-	S_AXIS_TKEEP[m/8-1:0]	Optional	-	Optional
-	-	S_AXIS_TLAST	Optional	-	Optional
-	-	S_AXIS_TID[m:0]	Optional	-	Optional
-	-	S_AXIS_TDEST[m:0]	Optional	-	Optional
-	-	S_AXIS_TUSER[m:0]	Optional	-	Optional
-	-	AXIS_PROG_FULL_THRESH[m:0]	Optional	-	Optional
-	-	AXIS_PROG_EMPTY_THRESH[m:0]	Optional	-	Optional
-	-	AXIS_INJECTSBITERR	Optional	-	Optional
-	-	AXIS_INJECTDBITERR	Optional	-	Optional
-	-	AXIS_SBITERR	Optional	-	Optional
-	-	AXIS_DBITERR	Optional	-	Optional
-	-	AXIS_OVERFLOW	Optional	-	Optional
-	-	AXIS_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXIS_UNDERFLOW	Optional	-	Optional
-	-	AXIS_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXIS_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_AWID[m:0]	Optional	-	Optional
-	-	S_AXI_AWADDR[m:0]	Optional	-	Optional
-	-	S_AXI_AWLEN[7:0]	Optional	-	Optional
-	-	S_AXI_AWSIZE[2:0]	Optional	-	Optional
-	-	S_AXI_AWBURST[1:0]	Optional	-	Optional
-	-	S_AXI_AWLOCK[2:0]	Optional	-	Optional
-	-	S_AXI_AWCACHE[4:0]	Optional	-	Optional
-	-	S_AXI_AWPROT[3:0]	Optional	-	Optional
-	-	S_AXI_AWQOS[3:0]	Optional	-	Optional
-	-	S_AXI_AWREGION[3:0]	Optional	-	Optional
-	-	S_AXI_AWUSER[m:0]	Optional	-	Optional
-	-	S_AXI_AWVALID	Optional	-	Optional
-	-	S_AXI_AWREADY	Optional	-	Optional

Table B-4: **Port Name Mapping: Asynchronous FIFO Cores *(Cont'd)***

Legacy Asynchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	M_AXI_AWID[m:0]	Optional	-	Optional
-	-	M_AXI_AWADDR[m:0]	Optional	-	Optional
-	-	M_AXI_AWLEN[7:0]	Optional	-	Optional
-	-	M_AXI_AWSIZE[2:0]	Optional	-	Optional
-	-	M_AXI_AWBURST[1:0]	Optional	-	Optional
-	-	M_AXI_AWLOCK[2:0]	Optional	-	Optional
-	-	M_AXI_AWCACHE[4:0]	Optional	-	Optional
-	-	M_AXI_AWPROT[3:0]	Optional	-	Optional
-	-	M_AXI_AWQOS[3:0]	Optional	-	Optional
-	-	M_AXI_AWREGION[3:0]	Optional	-	Optional
-	-	M_AXI_AWUSER[m:0]	Optional	-	Optional
-	-	M_AXI_AWVALID	Optional	-	Optional
-	-	M_AXI_AWREADY	Optional	-	Optional
-	-	AXI_AW_PROG_FULL_THRESH [m:0]	Optional	-	Optional
-	-	AXI_AW_PROG_EMPTY_ THRESH[m:0]	Optional	-	Optional
-	-	AXI_AW_INJECTSBITERR	Optional	-	Optional
-	-	AXI_AW_INJECTDBITERR	Optional	-	Optional
-	-	AXI_AW_SBITERR	Optional	-	Optional
-	-	AXI_AW_DBITERR	Optional	-	Optional
-	-	AXI_AW_OVERFLOW	Optional	-	Optional
-	-	AXI_AW_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_AW_UNDERFLOW	Optional	-	Optional
-	-	AXI_AW_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_AW_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_WID[m:0]	Optional	-	Optional
-	-	S_AXI_WDATA[m-1:0]	Optional	-	Optional
-	-	S_AXI_WSTRB[m/8-1:0]	Optional	-	Optional
-	-	S_AXI_WLAST	Optional	-	Optional
-	-	S_AXI_WUSER[m:0]	Optional	-	Optional
-	-	S_AXI_WVALID	Optional	-	Optional
-	-	S_AXI_WREADY	Optional	-	Optional
-	-	M_AXI_WID[m:0]	Optional	-	Optional

Table B-4: **Port Name Mapping: Asynchronous FIFO Cores *(Cont'd)***

Legacy Asynchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	M_AXI_WDATA[m-1:0]	Optional	-	Optional
-	-	M_AXI_WSTRB[m/8-1:0]	Optional	-	Optional
-	-	M_AXI_WLAST	Optional	-	Optional
-	-	M_AXI_WUSER[m:0]	Optional	-	Optional
-	-	M_AXI_WVALID	Optional	-	Optional
-	-	M_AXI_WREADY	Optional	-	Optional
-	-	AXI_W_PROG_FULL_THRESH[m:0]	Optional	-	Optional
-	-	AXI_W_PROG_EMPTY_THRESH [m:0]	Optional	-	Optional
-	-	AXI_W_INJECTSBITERR	Optional	-	Optional
-	-	AXI_W_INJECTDBITERR	Optional	-	Optional
-	-	AXI_W_SBITERR	Optional	-	Optional
-	-	AXI_W_DBITERR	Optional	-	Optional
-	-	AXI_W_OVERFLOW	Optional	-	Optional
-	-	AXI_W_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_W_UNDERFLOW	Optional	-	Optional
-	-	AXI_W_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_W_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_BID[m:0]	Optional	-	Optional
-	-	S_AXI_BRESP[1:0]	Optional	-	Optional
-	-	S_AXI_BUSER[m:0]	Optional	-	Optional
-	-	S_AXI_BVALID	Optional	-	Optional
-	-	S_AXI_BREADY	Optional	-	Optional
-	-	M_AXI_BID[m:0]	Optional	-	Optional
-	-	M_AXI_BRESP[1:0]	Optional	-	Optional
-	-	M_AXI_BUSER[m:0]	Optional	-	Optional
-	-	M_AXI_BVALID	Optional	-	Optional
-	-	M_AXI_BREADY	Optional	-	Optional
-	-	AXI_B_PROG_FULL_THRESH[m:0]	Optional	-	Optional
-	-	AXI_B_PROG_EMPTY_THRESH [m:0]	Optional	-	Optional
-	-	AXI_B_INJECTSBITERR	Optional	-	Optional
-	-	AXI_B_INJECTDBITERR	Optional	-	Optional
-	-	AXI_B_SBITERR	Optional	-	Optional

Table B-4: **Port Name Mapping: Asynchronous FIFO Cores *(Cont'd)***

Legacy Asynchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	AXI_B_DBITERR	Optional	-	Optional
-	-	AXI_B_OVERFLOW	Optional	-	Optional
-	-	AXI_B_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_B_UNDERFLOW	Optional	-	Optional
-	-	AXI_B_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_B_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_ARID[m:0]	Optional	-	Optional
-	-	S_AXI_ARADDR[m:0]	Optional	-	Optional
-	-	S_AXI_ARLEN[7:0]	Optional	-	Optional
-	-	S_AXI_ARSIZE[2:0]	Optional	-	Optional
-	-	S_AXI_ARBURST[1:0]	Optional	-	Optional
-	-	S_AXI_ARLOCK[2:0]	Optional	-	Optional
-	-	S_AXI_ARCACHE[4:0]	Optional	-	Optional
-	-	S_AXI_ARPROT[3:0]	Optional	-	Optional
-	-	S_AXI_ARQOS[3:0]	Optional	-	Optional
-	-	S_AXI_ARREGION[3:0]	Optional	-	Optional
-	-	S_AXI_ARUSER[m:0]	Optional	-	Optional
-	-	S_AXI_ARVALID	Optional	-	Optional
-	-	S_AXI_ARREADY	Optional	-	Optional
-	-	M_AXI_ARID[m:0]	Optional	-	Optional
-	-	M_AXI_ARADDR[m:0]	Optional	-	Optional
-	-	M_AXI_ARLEN[7:0]	Optional	-	Optional
-	-	M_AXI_ARSIZE[2:0]	Optional	-	Optional
-	-	M_AXI_ARBURST[1:0]	Optional	-	Optional
-	-	M_AXI_ARLOCK[2:0]	Optional	-	Optional
-	-	M_AXI_ARCACHE[4:0]	Optional	-	Optional
-	-	M_AXI_ARPROT[3:0]	Optional	-	Optional
-	-	M_AXI_ARQOS[3:0]	Optional	-	Optional
-	-	M_AXI_ARREGION[3:0]	Optional	-	Optional
-	-	M_AXI_ARUSER[m:0]	Optional	-	Optional
-	-	M_AXI_ARVALID	Optional	-	Optional
-	-	M_AXI_ARREADY	Optional	-	Optional

Table B-4: **Port Name Mapping: Asynchronous FIFO Cores *(Cont'd)***

Legacy Asynchronous FIFO Core		**New FIFO Generator Core**		**Conversion Description**	**Functionality**
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	AXI_AR_PROG_FULL_THRESH [m:0]	Optional	-	Optional
-	-	AXI_AR_PROG_EMPTY_THRESH [m:0]	Optional	-	Optional
-	-	AXI_AR_INJECTSBITERR	Optional	-	Optional
-	-	AXI_AR_INJECTDBITERR	Optional	-	Optional
-	-	AXI_AR_SBITERR	Optional	-	Optional
-	-	AXI_AR_DBITERR	Optional	-	Optional
-	-	AXI_AR_OVERFLOW	Optional	-	Optional
-	-	AXI_AR_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_AR_UNDERFLOW	Optional	-	Optional
-	-	AXI_AR_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_AR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_RID[m:0]	Optional	-	Optional
-	-	S_AXI_RDATA[m-1:0]	Optional	-	Optional
-	-	S_AXI_RRESP[1:0]	Optional	-	Optional
-	-	S_AXI_RLAST	Optional	-	Optional
-	-	S_AXI_RUSER[m:0]	Optional	-	Optional
-	-	S_AXI_RVALID	Optional	-	Optional
-	-	S_AXI_RREADY	Optional	-	Optional
-	-	M_AXI_RID[m:0]	Optional	-	Optional
-	-	M_AXI_RDATA[m-1:0]	Optional	-	Optional
-	-	M_AXI_ RRESP[1:0]	Optional	-	Optional
-	-	M_AXI_RLAST	Optional	-	Optional
-	-	M_AXI_RUSER[m:0]	Optional	-	Optional
-	-	M_AXI_RVALID	Optional	-	Optional
-	-	M_AXI_RREADY	Optional	-	Optional
-	-	AXI_R_PROG_FULL_THRESH[m:0]	Optional	-	Optional
-	-	AXI_R_PROG_EMPTY_THRESH [m:0]	Optional	-	Optional
-	-	AXI_R_INJECTSBITERR	Optional	-	Optional
-	-	AXI_R_INJECTDBITERR	Optional	-	Optional
-	-	AXI_R_SBITERR	Optional	-	Optional
-	-	AXI_R_DBITERR	Optional	-	Optional

Table B-4: **Port Name Mapping: Asynchronous FIFO Cores *(Cont'd)***

Legacy Asynchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	AXI_R_OVERFLOW	Optional	-	Optional
-	-	AXI_R_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_R_UNDERFLOW	Optional	-	Optional
-	-	AXI_R_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_R_DATA_COUNT[m:0]	Optional	-	Optional

Table B-5: **Port Name Mapping: Synchronous FIFO Cores**

Legacy Synchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
DIN[N:0]	Available	DIN[N:0]	Available	Same	Available
WR_EN	Available	WR_EN	Available	Same	Available
CLK	Available	CLK	Available	Same	Available
RD_EN	Available	RD_EN	Available	Same	Available
SINIT	Available	SRST	Optional	Direct replacement	Available
FULL	Available	FULL	Available	Same	Available
-	-	ALMOST_FULL	Available	-	Available
-	-	PROG_FULL	Optional	-	Optional
-	-	PROG_FULL_THRESH	Optional	-	Optional
-	-	PROG_FULL_THRESH_ASSERT	Optional	-	Optional
-	-	PROG_FULL_THRESH_NEGATE	Optional	-	Optional

Table B-5: **Port Name Mapping: Synchronous FIFO Cores *(Cont'd)***

Legacy Synchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
DATA_COUNT[C:0]	Optional	DATA_COUNT[D:0]	Optional	FIFO Generator only supports Data Count Width from 1 to LOG2(FIFO depth). Additional logic: If Data Count Width is equal to LOG2(FIFO Depth)+1, need to link FULL port to the most significant bit of the DATA_COUNT port. (DATA_COUNT[D:0] = {FULL, DATA_COUNT [C:0]})	Optional
-	-	WR_DATA_COUNT[D:0]	Optional	-	Optional
WR_ACK	Optional	WR_ACK	Optional	Same	Optional
WR_ERR	Optional	OVERFLOW	Optional	Direct replacement	Optional
DOUT[N:0]	Available	DOUT[M:0]	Available	Same	Optional
EMPTY	Available	EMPTY	Available	Same	Optional
-	-	ALMOST_EMPTY	Optional	-	Optional
-	-	PROG_EMPTY	Optional	-	Optional
-	-	PROG_EMPTY_THRESH	Optional	-	Optional
-	-	PROG_EMPTY_THRESH_ASSERT	Optional	-	Optional
-	-	PROG_EMPTY_THRESH_NEGATE	Optional	-	Optional
-	-	RD_DATA_COUNT[C:0]	Optional	-	Optional
RD_ACK	Optional	VALID	Optional	Direct replacement	Optional

Table B-5: **Port Name Mapping: Synchronous FIFO Cores *(Cont'd)***

Legacy Synchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
RD_ERR	Optional	UNDERFLOW	Optional	Direct replacement	Optional
-	-	SBITERR	Optional	-	Optional
-	-	DBITERR	Optional	-	Optional
-	-	INJECTSBITERR	Optional	-	Optional
-	-	INJECTDBITERR	Optional	-	Optional
-	-	M_ACLK	Optional	-	Optional
-	-	S_ACLK	Optional	-	Optional
-	-	S_ARESETN	Optional	-	Optional
-	-	M_AXIS_TVALID	Optional	-	Optional
-	-	M_AXIS_TREADY	Optional	-	Optional
-	-	M_AXIS_TDATA[m-1:0]	Optional	-	Optional
-	-	M_AXIS_TSTRB[m/8-1:0]	Optional	-	Optional
-	-	M_AXIS_TKEEP[m/8-1:0]	Optional	-	Optional
-	-	M_AXIS_TLAST	Optional	-	Optional
-	-	M_AXIS_TID[m:0]	Optional	-	Optional
-	-	M_AXIS_TDEST[m:0]	Optional	-	Optional
-	-	M_AXIS_TUSER[m:0]	Optional	-	Optional
-	-	S_AXIS_TVALID	Optional	-	Optional
-	-	S_AXIS_TREADY	Optional	-	Optional
-	-	S_AXIS_TDATA[m-1:0]	Optional	-	Optional
-	-	S_AXIS_TSTRB[m/8-1:0]	Optional	-	Optional
-	-	S_AXIS_TKEEP[m/8-1:0]	Optional	-	Optional
-	-	S_AXIS_TLAST	Optional	-	Optional
-	-	S_AXIS_TID[m:0]	Optional	-	Optional
-	-	S_AXIS_TDEST[m:0]	Optional	-	Optional
-	-	S_AXIS_TUSER[m:0]	Optional	-	Optional
-	-	AXIS_PROG_FULL_THRESH[m:0]	Optional	-	Optional
-	-	AXIS_PROG_EMPTY_THRESH[m:0]	Optional	-	Optional
-	-	AXIS_INJECTSBITERR	Optional	-	Optional
-	-	AXIS_INJECTDBITERR	Optional	-	Optional
-	-	AXIS_SBITERR	Optional	-	Optional
-	-	AXIS_DBITERR	Optional	-	Optional

Table B-5: **Port Name Mapping: Synchronous FIFO Cores *(Cont'd)***

Legacy Synchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	Availability	Port	Availability	Port	Availability
-	-	AXIS_OVERFLOW	Optional	-	Optional
-	-	AXIS_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXIS_UNDERFLOW	Optional	-	Optional
-	-	AXIS_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXIS_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_AWID[m:0]	Optional	-	Optional
-	-	S_AXI_AWADDR[m:0]	Optional	-	Optional
-	-	S_AXI_AWLEN[7:0]	Optional	-	Optional
-	-	S_AXI_AWSIZE[2:0]	Optional	-	Optional
-	-	S_AXI_AWBURST[1:0]	Optional	-	Optional
-	-	S_AXI_AWLOCK[2:0]	Optional	-	Optional
-	-	S_AXI_AWCACHE[4:0]	Optional	-	Optional
-	-	S_AXI_AWPROT[3:0]	Optional	-	Optional
-	-	S_AXI_AWQOS[3:0]	Optional	-	Optional
-	-	S_AXI_AWREGION[3:0]	Optional	-	Optional
-	-	S_AXI_AWUSER[m:0]	Optional	-	Optional
-	-	S_AXI_AWVALID	Optional	-	Optional
-	-	S_AXI_AWREADY	Optional	-	Optional
-	-	M_AXI_AWID[m:0]	Optional	-	Optional
-	-	M_AXI_AWADDR[m:0]	Optional	-	Optional
-	-	M_AXI_AWLEN[7:0]	Optional	-	Optional
-	-	M_AXI_AWSIZE[2:0]	Optional	-	Optional
-	-	M_AXI_AWBURST[1:0]	Optional	-	Optional
-	-	M_AXI_AWLOCK[2:0]	Optional	-	Optional
-	-	M_AXI_AWCACHE[4:0]	Optional	-	Optional
-	-	M_AXI_AWPROT[3:0]	Optional	-	Optional
-	-	M_AXI_AWQOS[3:0]	Optional	-	Optional
-	-	M_AXI_AWREGION[3:0]	Optional	-	Optional
-	-	M_AXI_AWUSER[m:0]	Optional	-	Optional
-	-	M_AXI_AWVALID	Optional	-	Optional
-	-	M_AXI_AWREADY	Optional	-	Optional
-	-	AXI_AW_PROG_FULL_THRESH[m:0]	Optional	-	Optional

Table B-5: **Port Name Mapping: Synchronous FIFO Cores *(Cont'd)***

Legacy Synchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	AXI_AW_PROG_EMPTY_THRESH[m:0]	Optional	-	Optional
-	-	AXI_AW_INJECTSBITERR	Optional	-	Optional
-	-	AXI_AW_INJECTDBITERR	Optional	-	Optional
-	-	AXI_AW_SBITERR	Optional	-	Optional
-	-	AXI_AW_DBITERR	Optional	-	Optional
-	-	AXI_AW_OVERFLOW	Optional	-	Optional
-	-	AXI_AW_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_AW_UNDERFLOW	Optional	-	Optional
-	-	AXI_AW_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_AW_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_WID[m:0]	Optional	-	Optional
-	-	S_AXI_WDATA[m-1:0]	Optional	-	Optional
-	-	S_AXI_WSTRB[m/8-1:0]	Optional	-	Optional
-	-	S_AXI_WLAST	Optional	-	Optional
-	-	S_AXI_WUSER[m:0]	Optional	-	Optional
-	-	S_AXI_WVALID	Optional	-	Optional
-	-	S_AXI_WREADY	Optional	-	Optional
-	-	M_AXI_WID[m:0]	Optional	-	Optional
-	-	M_AXI_WDATA[m-1:0]	Optional	-	Optional
-	-	M_AXI_WSTRB[m/8-1:0]	Optional	-	Optional
-	-	M_AXI_WLAST	Optional	-	Optional
-	-	M_AXI_WUSER[m:0]	Optional	-	Optional
-	-	M_AXI_WVALID	Optional	-	Optional
-	-	M_AXI_WREADY	Optional	-	Optional
-	-	AXI_W_PROG_FULL_THRESH[m:0]	Optional	-	Optional
-	-	AXI_W_PROG_EMPTY_THRESH[m:0]	Optional	-	Optional
-	-	AXI_W_INJECTSBITERR	Optional	-	Optional
-	-	AXI_W_INJECTDBITERR	Optional	-	Optional
-	-	AXI_W_SBITERR	Optional	-	Optional
-	-	AXI_W_DBITERR	Optional	-	Optional
-	-	AXI_W_OVERFLOW	Optional	-	Optional
-	-	AXI_W_WR_DATA_COUNT[m:0]	Optional	-	Optional

Table B-5: **Port Name Mapping: Synchronous FIFO Cores *(Cont'd)***

Legacy Synchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	AXI_W_UNDERFLOW	Optional	-	Optional
-	-	AXI_W_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_W_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_BID[m:0]	Optional	-	Optional
-	-	S_AXI_BRESP[1:0]	Optional	-	Optional
-	-	S_AXI_BUSER[m:0]	Optional	-	Optional
-	-	S_AXI_BVALID	Optional	-	Optional
-	-	S_AXI_BREADY	Optional	-	Optional
-	-	M_AXI_BID[m:0]	Optional	-	Optional
-	-	M_AXI_BRESP[1:0]	Optional	-	Optional
-	-	M_AXI_BUSER[m:0]	Optional	-	Optional
-	-	M_AXI_BVALID	Optional	-	Optional
-	-	M_AXI_BREADY	Optional	-	Optional
-	-	AXI_B_PROG_FULL_THRESH[m:0]	Optional	-	Optional
-	-	AXI_B_PROG_EMPTY_THRESH[m:0]	Optional	-	Optional
-	-	AXI_B_INJECTSBITERR	Optional	-	Optional
-	-	AXI_B_INJECTDBITERR	Optional	-	Optional
-	-	AXI_B_SBITERR	Optional	-	Optional
-	-	AXI_B_DBITERR	Optional	-	Optional
-	-	AXI_B_OVERFLOW	Optional	-	Optional
-	-	AXI_B_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_B_UNDERFLOW	Optional	-	Optional
-	-	AXI_B_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_B_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_ARID[m:0]	Optional	-	Optional
-	-	S_AXI_ARADDR[m:0]	Optional	-	Optional
-	-	S_AXI_ARLEN[7:0]	Optional	-	Optional
-	-	S_AXI_ARSIZE[2:0]	Optional	-	Optional
-	-	S_AXI_ARBURST[1:0]	Optional	-	Optional
-	-	S_AXI_ARLOCK[2:0]	Optional	-	Optional
-	-	S_AXI_ARCACHE[4:0]	Optional	-	Optional
-	-	S_AXI_ARPROT[3:0]	Optional	-	Optional

Table B-5: **Port Name Mapping: Synchronous FIFO Cores *(Cont'd)***

Legacy Synchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	S_AXI_ARQOS[3:0]	Optional	-	Optional
-	-	S_AXI_ARREGION[3:0]	Optional	-	Optional
-	-	S_AXI_ARUSER[m:0]	Optional	-	Optional
-	-	S_AXI_ARVALID	Optional	-	Optional
-	-	S_AXI_ARREADY	Optional	-	Optional
-	-	M_AXI_ARID[m:0]	Optional	-	Optional
-	-	M_AXI_ARADDR[m:0]	Optional	-	Optional
-	-	M_AXI_ARLEN[7:0]	Optional	-	Optional
-	-	M_AXI_ARSIZE[2:0]	Optional	-	Optional
-	-	M_AXI_ARBURST[1:0]	Optional	-	Optional
-	-	M_AXI_ARLOCK[2:0]	Optional	-	Optional
-	-	M_AXI_ARCACHE[4:0]	Optional	-	Optional
-	-	M_AXI_ARPROT[3:0]	Optional	-	Optional
-	-	M_AXI_ARQOS[3:0]	Optional	-	Optional
-	-	M_AXI_ARREGION[3:0]	Optional	-	Optional
-	-	M_AXI_ARUSER[m:0]	Optional	-	Optional
-	-	M_AXI_ARVALID	Optional	-	Optional
-	-	M_AXI_ARREADY	Optional	-	Optional
-	-	AXI_AR_PROG_FULL_THRESH[m:0]	Optional	-	Optional
-	-	AXI_AR_PROG_EMPTY_THRESH[m:0]	Optional	-	Optional
-	-	AXI_AR_INJECTSBITERR	Optional	-	Optional
-	-	AXI_AR_INJECTDBITERR	Optional	-	Optional
-	-	AXI_AR_SBITERR	Optional	-	Optional
-	-	AXI_AR_DBITERR	Optional	-	Optional
-	-	AXI_AR_OVERFLOW	Optional	-	Optional
-	-	AXI_AR_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_AR_UNDERFLOW	Optional	-	Optional
-	-	AXI_AR_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_AR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	S_AXI_RID[m:0]	Optional	-	Optional
-	-	S_AXI_RDATA[m-1:0]	Optional	-	Optional
-	-	S_AXI_RRESP[1:0]	Optional	-	Optional

Table B-5: **Port Name Mapping: Synchronous FIFO Cores *(Cont'd)***

Legacy Synchronous FIFO Core		New FIFO Generator Core		Conversion Description	Functionality
Port	**Availability**	**Port**	**Availability**	**Port**	**Availability**
-	-	S_AXI_RLAST	Optional	-	Optional
-	-	S_AXI_RUSER[m:0]	Optional	-	Optional
-	-	S_AXI_RVALID	Optional	-	Optional
-	-	S_AXI_RREADY	Optional	-	Optional
-	-	M_AXI_RID[m:0]	Optional	-	Optional
-	-	M_AXI_RDATA[m-1:0]	Optional	-	Optional
-	-	M_AXI_ RRESP[1:0]	Optional	-	Optional
-	-	M_AXI_RLAST	Optional	-	Optional
-	-	M_AXI_RUSER[m:0]	Optional	-	Optional
-	-	M_AXI_RVALID	Optional	-	Optional
-	-	M_AXI_RREADY	Optional	-	Optional
-	-	AXI_R_PROG_FULL_THRESH[m:0]	Optional	-	Optional
-	-	AXI_R_PROG_EMPTY_THRESH[m:0]	Optional	-	Optional
-	-	AXI_R_INJECTSBITERR	Optional	-	Optional
-	-	AXI_R_INJECTDBITERR	Optional	-	Optional
-	-	AXI_R_SBITERR	Optional	-	Optional
-	-	AXI_R_DBITERR	Optional	-	Optional
-	-	AXI_R_OVERFLOW	Optional	-	Optional
-	-	AXI_R_WR_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_R_UNDERFLOW	Optional	-	Optional
-	-	AXI_R_RD_DATA_COUNT[m:0]	Optional	-	Optional
-	-	AXI_R_DATA_COUNT[m:0]	Optional	-	Optional

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. It also contains a sample flow diagram and other design samples to guide you through the debug process.

The following topics are included in this appendix:

Finding Help on Xilinx.com

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

Documentation

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

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

Release Notes

Known issues for all cores, including the FIFO Generator are described in the IP Release Notes Guide (XTP025).

Known Issues

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

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

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

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

Answer Records for the FIFO Generator

- AR50917, FIFO Generator Release Notes

Contacting Technical Support

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

To contact Xilinx Technical Support:

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

When opening a WebCase, include:

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

Debug Tools

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

Example Design

The FIFO Generator is delivered with an example design that can be synthesized, complete with functional test benches. Information about the example design can be found inChapter 8, Detailed Example Design for the Vivado Design Suite and Chapter 12, Detailed Example Design for the ISE Design Suite.

ChipScope Pro Tool

The ChipScope™ Pro tool inserts logic analyzer, bus analyzer, and virtual I/O cores directly into your design. The ChipScope Pro tool allows you to set trigger conditions to capture application and integrated block port signals in hardware. Captured signals can then be analyzed through the ChipScope Pro Logic Analyzer tool. For detailed information for using the ChipScope Pro tool, see www.xilinx.com/tools/cspro.htm.

Reference Boards

Various Xilinx development boards can be used to prototype the FIFO Generator and establish that the core can communicate with the system.

- 7 series evaluation boards
  - KC705
  - KC724

License Checkers

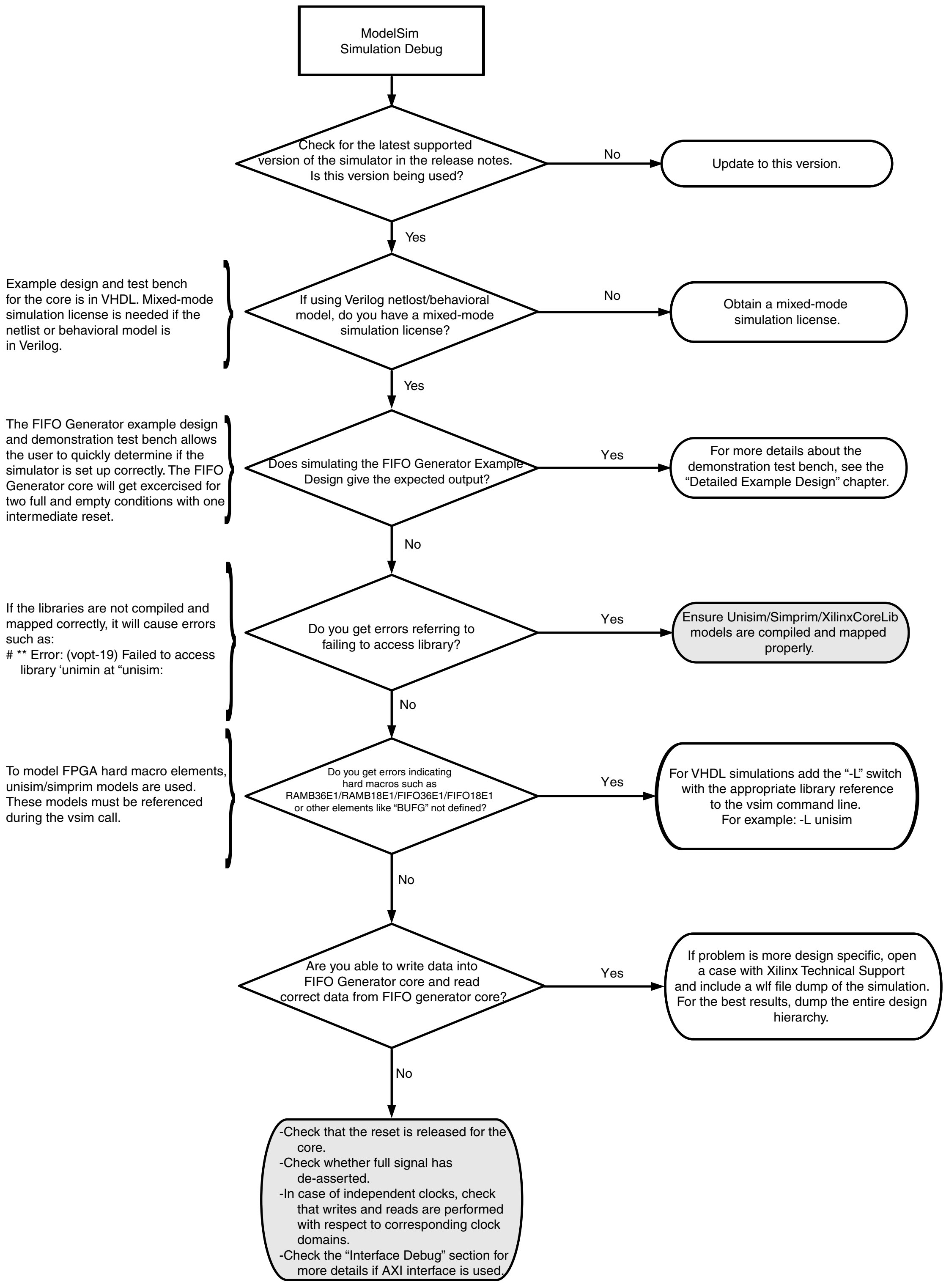
If the IP requires a license key, the key must be verified. The ISE and Vivado tool flows have a number of license check points for gating licensed IP through the flow. If the license check succeeds, the IP may continue generation; otherwise generation halts with an error. License checkpoints are enforced by the following tools:

- ISE flow: XST, NgdBuild, Bitgen
- Vivado flow: Vivado Synthesis, Vivado Implementation, write_bitstream (Tcl command)

IMPORTANT: *IP license level is ignored at checkpoints. The test confirms a valid license exists. It does not check IP license level.*

Simulation Debug

The simulation debug flow for ModelSim is illustrated below. A similar approach can be used with other simulators.


ModelSim Simulation Debug
Check for the latest supported version of the simulator in the release notes. Is this version being used?
No
Update to this version.
Yes
Example design and test bench for the core is in VHDL. Mixed-mode simulation license is needed if the netlist or behavioral model is in Verilog.
If using Verilog netlost/behavioral model, do you have a mixed-mode simulation license?
No
Obtain a mixed-mode simulation license.
Yes
The FIFO Generator example design and demonstration test bench allows the user to quickly determine if the simulator is set up correctly. The FIFO Generator core will get excercised for two full and empty conditions with one intermediate reset.
Does simulating the FIFO Generator Example Design give the expected output?
Yes
For more details about the demonstration test bench, see the "Detailed Example Design" chapter.
No
If the libraries are not compiled and mapped correctly, it will cause errors such as:
# ** Error: (vopt-19) Failed to access library 'unimin at "unisim:
Do you get errors referring to failing to access library?
Yes
Ensure Unisim/Simprim/XilinxCoreLib models are compiled and mapped properly.
No
To model FPGA hard macro elements, unisim/simprim models are used. These models must be referenced during the vsim call.
Do you get errors indicating hard macros such as RAMB36E1/RAMB18E1/FIFO36E1/FIFO18E1 or other elements like "BUFG" not defined?
Yes
For VHDL simulations add the "-L" switch with the appropriate library reference to the vsim command line. For example: -L unisim
No
Are you able to write data into FIFO Generator core and read correct data from FIFO generator core?
Yes
If problem is more design specific, open a case with Xilinx Technical Support and include a wlf file dump of the simulation. For the best results, dump the entire design hierarchy.
No
-Check that the reset is released for the core.
-Check whether full signal has de-asserted.
-In case of independent clocks, check that writes and reads are performed with respect to corresponding clock domains.
-Check the "Interface Debug" section for more details if AXI interface is used.

Hardware Debug

Hardware issues can range from system start to problems seen after hours of testing. This section provides debug steps for common issues. The ChipScope tool is a valuable resource to use in hardware debug. The signal names mentioned in the following individual sections can be probed using the ChipScope tool for debugging the specific problems.

General Checks

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

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

Interface Debug

AXI4/AXI4-Lite Interfaces

See Figure 1-3 for an AXI4/AXI4-Lite timing diagram.

On the slave side, output `S_AXI_AWREADY/S_AXI_WREADY/S_AXI_ARREADY` asserts when the FIFO is ready to accept the address/data, and output `S_AXI_BVALID/S_AXI_RVALID` asserts when the read data/write response is valid.

On the master side, output `M_AXI_AWVALID/M_AXI_WVALID/M_AXI_ARVALID` asserts when the write address/data is valid, and output `M_AXI_BREADY/M_AXI_RREADY` asserts when the FIFO is ready to accept read data/write response.

If the interface is unresponsive, ensure that the following conditions are met:

- The S_ACLK/M_ACLK inputs are connected and toggling.
- The interface is not being held in reset, and S_ARESETN is an active-Low reset.
- The interface is enabled, and S_ACLK_EN/M_ACLK_EN is active-High (if used).
- The main core clocks are toggling and that the enables are also asserted.
- If the simulation has been run, verify in simulation and/or a ChipScope tool capture that the waveform is correct for accessing the AXI4/AXI4-Lite interface.

AXI4-Stream Interfaces

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

- If the slave side S_AXIS_TREADY is stuck low following the S_AXIS_TVALID input being asserted, the core cannot accept data.
- If the master M_AXIS_TVALID is stuck low, the core is not reading data.
- Check that the S_ACLK/M_ACLK inputs are connected and toggling.
- Check that the AXI4-Stream waveforms are being followed Figure 1-3.
- Check core configuration.
- Add appropriate core specific checks.

Native Interface

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

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

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

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

Quick Start Example Design

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

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

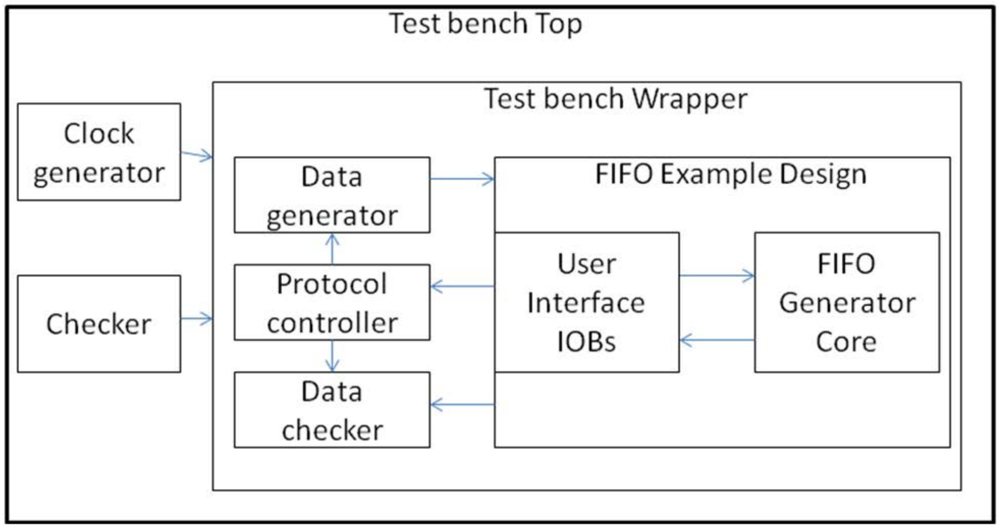


Figure D-1: **Example Design and Demonstration Test Bench**

The FIFO generator example design consists of the following:

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

The FIFO generator example design has been tested with Xilinx® ISE® software v14.4, Xilinx ISim, Cadence Incisive Enterprise Simulator (IES) and Mentor Graphics ModelSim simulator .

Implementing the Example Design

After generating a core the netlist and example design can be processed by the Xilinx implementation tools. The generated output files include scripts to assist you in running the Xilinx software.

To implement the FIFO Generator example design, open a command prompt or terminal window and type these commands:

For Windows:

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

For Linux:

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

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

Simulating the Example Design

The FIFO Generator core provides a quick way to simulate and observe the behavior of the core by using the provided example design. There are two different simulation types: functional and timing. The simulation models provided are either in VHDL or Verilog, depending on the CORE Generator software Design Entry project option.

Setting up for Simulation

The Xilinx UNISIM and SIMPRIM libraries must be mapped into the simulator. If the UNISIM or SIMPRIM libraries are not set for your environment, go to the Synthesis and Simulation Guide in the Xilinx Software Manuals for assistance compiling Xilinx simulation models.

Simulation scripts are provided for ISIM, IES and ModelSim simulation tools.

Functional Simulation

This section provides instructions for running a functional simulation of the FIFO Generator core using either VHDL or Verilog. Functional simulation models are provided when the core is generated. Implementing the core before simulating the functional models is not required.

To run a VHDL or Verilog functional simulation of the example design:

1. Set the current directory to:

   `<component_name>/simulation/functional/`

2. Launch the simulation script.

   For Linux:

   - ISIM:./simulate_isim.sh
   - IES:./simulate_ncsim.sh
   - MTI: vsim -do simulate_mti.do

   For Windows:

   - ISIM:./simulate_isim.bat
   - MTI: vsim -do simulate_mti.do

The simulation script compiles the functional simulation models and demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

Timing Simulation

This section contains instructions for running a timing simulation of the FIFO Generator core using either VHDL or Verilog. A timing simulation model is generated when the core is run through the Xilinx tools using the implement script. It is a requirement that the core is implemented before attempting to run timing simulation.

To run a VHDL or Verilog functional simulation of the example design:

1. Set the current directory to:

   `<component_name>/simulation/timing/`

2. Launch the simulation script:

   For Linux:

   - **ISIM**: `./simulate_isim.sh`
   - **IES**: `./simulate_ncsim.sh`
   - **MTI**: `vsim -do simulate_mti.do`

   For Windows:

   - **ISIM**: `./simulate_isim.bat`
   - **MTI**: `vsim -do simulate_mti.do`

The simulation script compiles the timing simulation model and the demonstration test bench, adds relevant signals to the wave window, and runs the simulation. To observe the operation of the core, inspect the simulation transcript and the waveform.

Simulating Your Design

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

Simulation Models

The FIFO Generator supports two types of simulation models based on the Xilinx CORE Generator system project options. The models are available in both VHDL and Verilog®. Both types of models are described in detail in this chapter.

To choose a model:

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

Behavioral Models

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

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

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

The following considerations apply to the behavioral models.

- Write operations always occur relative to the write clock (`WR_CLK`) or common clock (`CLK`) domain, as do the corresponding handshaking signals.

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

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

Structural Models

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

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

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

Comparison of Native and AXI4 FIFO XCO Parameters

Table F-1 describes the comparison of Native FIFO and AXI4 FIFO XCO parameters, including the possible values.

Table F-1: **Native FIFO and AXI4 FIFO XCO Parameter Comparison**

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
1	almost_empty_flag	almost_empty_flag	True, False
2	almost_full_flag	almost_full_flag	True, False
3	data_count	data_count	True, False
4	data_count_width	data_count_width	1 – $\log_2$(output_depth)
5	disable_timing_violations	disable_timing_violations	True, False
6	dout_reset_value	dout_reset_value	Any hexadecimal value of width 1 - 1024
7	empty_threshold_assert_value	empty_threshold_assert_value	For STD: 2 - 4194300 For FWFT: 4 - 4194302
8	empty_threshold_negate_valu e	empty_threshold_negate_value	For STD: 3 - 4194301 For FWFT: 5 - 4194303
9	enable_ecc	enable_ecc	True, False
10	enable_reset_synchronization	enable_reset_synchronization	True, False
11	fifo_implementation	fifo_implementation	Common_Clock_Block_RAM Common_Clock_Distributed_RAM Common_Clock_Shift_Register Common_Clock_Builtin_FIFO Independent_Clocks_Block_RAM Independent_Clocks_Distributed_R AM Independent_Clocks_Builtin_FIFO
12	full_flags_reset_value	full_flags_reset_value	0, 1
13	full_threshold_assert_value	full_threshold_assert_value	For STD: 4 - 4194302 For FWFT: 6 - 4194303
14	full_threshold_negate_value	full_threshold_negate_value	For STD: 3 - 4194301 For FWFT: 5 - 4194302

Table F-1: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
15	inject_dbit_error	inject_dbit_error	True, False
16	inject_sbit_error	inject_sbit_error	True, False
17	input_data_width	input_data_width	1 - 1024
18	input_depth	input_depth	2^4 - 2^{22}
19	output_data_width	output_data_width	1 - 1024
20	output_depth	output_depth	2^4 - 2^{22}
21	overflow_flag	overflow_flag	True, False
22	overflow_sense	overflow_sense	Active_High, Active_Low
23	performance_options	performance_options	Standard_FIFO (STD), First_Word_Fall_Through (FWFT)
24	programmable_empty_type	programmable_empty_type	No_Programmable_Empty_Threshold Single_Programmable_Empty_Threshold_Constant Multiple_Programmable_Empty_Threshold_Constants Single_Programmable_Empty_Threshold_Input_Port Multiple_Programmable_Empty_Threshold_Input_Ports
25	programmable_full_type	programmable_full_type	No_Programmable_Full_Threshold Single_Programmable_Full_Threshold_Constant Multiple_Programmable_Full_Threshold_Constants Single_Programmable_Full_Threshold_Input_Port Multiple_Programmable_Full_Threshold_Input_Ports
26	read_clock_frequency	read_clock_frequency	1 - 1000
27	read_data_count	read_data_count	True, False
28	read_data_count_width	read_data_count_width	1 – $\log_2$(output_depth)
29	reset_pin	reset_pin	True, False
30	reset_type	reset_type	Synchronous_Reset, Asynchronous_Reset
31	underflow_flag	underflow_flag	True, False
32	underflow_sense	underflow_sense	Active_High, Active_Low
33	use_dout_reset	use_dout_reset	True, False
34	use_embedded_registers	use_embedded_registers	True, False
35	use_extra_logic	use_extra_logic	True, False

Table F-1: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
36	valid_flag	valid_flag	True, False
37	valid_sense	valid_sense	Active_High, Active_Low
38	write_acknowledge_flag	write_acknowledge_flag	True, False
39	write_acknowledge_sense	write_acknowledge_sense	Active_High, Active_Low
40	write_clock_frequency	write_clock_frequency	1 – 1000
41	write_data_count	write_data_count	True, False
42	write_data_count_width	write_data_count_width	1 – $\log_2$(input_depth)
43	N/A	interface_type	Native, AXI4
44	N/A	axi_type	AXI4_Stream, AXI4_Full, AXI4_Lite
45	N/A	image_type	Application_Diagram, Block_Diagram
46	N/A	enable_write_channel	True, False
47	N/A	enable_read_channel	True, False
48	N/A	clock_type_axi	Common_Clock, Independent_Clock
49	N/A	use_clock_enable	True, False
50	N/A	clock_enable_type	Slave_Interface_Clock_Enable, Master_Interface_Clock_Enable
51	N/A	id_width	1 - 8
52	N/A	axi_address_width	1 – 32
53	N/A	axi_data_width	2^3 - 2^9
54	N/A	enable_awuser	True, False
55	N/A	enable_wuser	True, False
56	N/A	enable_buser	True, False
57	N/A	enable_aruser	True, False
58	N/A	enable_ruser	True, False
59	N/A	enable_tuser	True, False
60	N/A	awuser_width	1 - 256
61	N/A	wuser_width	1 - 256
62	N/A	buser_width	1 - 256
63	N/A	aruser_width	1 - 256
64	N/A	ruser_width	1 - 256
65	N/A	tuser_width	1 - 256
66	N/A	enable_tdata	True, False

Table F-1: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
67	N/A	enable_tdest	True, False
68	N/A	enable_tid	True, False
69	N/A	enable_tkeep	True, False
70	N/A	enable_tlast	True, False
71	N/A	enable_tready	True, False
72	N/A	enable_tstrobe	True, False
73	N/A	tdata_width	2^3 - 2^9
74	N/A	tdest_width	1 - 4
75	N/A	tid_width	1 - 8
76	N/A	tkeep_width	tdata_width/8
77	N/A	tstrb_width	tdata_width/8
78	N/A	axis_type	FIFO
79	N/A	wach_type	FIFO
80	N/A	wdch_type	FIFO
81	N/A	wrch_type	FIFO
82	N/A	rach_type	FIFO
83	N/A	rdch_type	FIFO
84	N/A	fifo_implementation_type_axis	Common_Clock_Block_RAM Common_Clock_Distributed_RAM Independent_Clock_Block_RAM Independent_Clock_Distributed_RAM
85	N/A	fifo_implementation_type_rach	
86	N/A	fifo_implementation_type_rdch	
87	N/A	fifo_implementation_type_wach	
88	N/A	fifo_implementation_type_wdch	
89	N/A	fifo_implementation_type_wrch	
90	N/A	fifo_application_type_axis	Data_FIFO
91	N/A	fifo_application_type_rach	Data_FIFO
92	N/A	fifo_application_type_rdch	Data_FIFO
93	N/A	fifo_application_type_wach	Data_FIFO
94	N/A	fifo_application_type_wdch	Data_FIFO
95	N/A	fifo_application_type_wrch	Data_FIFO
96	N/A	enable_ecc_axis	True, False

Table F-1: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
97	N/A	enable_ecc_rach	True, False
98	N/A	enable_ecc_rdch	True, False
99	N/A	enable_ecc_wach	True, False
100	N/A	enable_ecc_wdch	True, False
101	N/A	enable_ecc_wrch	True, False
102	N/A	inject_sbit_error_axis	True, False
103	N/A	inject_sbit_error_rach	True, False
104	N/A	inject_sbit_error_rdch	True, False
105	N/A	inject_sbit_error_wach	True, False
106	N/A	inject_sbit_error_wdch	True, False
107	N/A	inject_sbit_error_wrch	True, False
108	N/A	inject_dbit_error_axis	True, False
109	N/A	inject_dbit_error_rach	True, False
110	N/A	inject_dbit_error_rdch	True, False
111	N/A	inject_dbit_error_wach	True, False
112	N/A	inject_dbit_error_wdch	True, False
113	N/A	inject_dbit_error_wrch	True, False
114	N/A	input_depth_axis	2^4 - 2^{16}
115	N/A	input_depth_rach	2^4 - 2^{16}
116	N/A	input_depth_rdch	2^4 - 2^{16}
117	N/A	input_depth_wach	2^4 - 2^{16}
118	N/A	input_depth_wdch	2^4 - 2^{16}
119	N/A	input_depth_wrch	2^4 - 2^{16}
120	N/A	enable_data_counts_axis	True, False
121	N/A	enable_data_counts_rach	True, False
122	N/A	enable_data_counts_rdch	True, False
123	N/A	enable_data_counts_wach	True, False
124	N/A	enable_data_counts_wdch	True, False
125	N/A	enable_data_counts_wrch	True, False
126	N/A	programmable_full_type_axis	No_Programmable_Full_Threshold, Single_Programmable_Full_Threshold_Constant, Single_Programmable_Full_Threshold_Input_Port
127	N/A	programmable_full_type_rach	

Table F-1: **Native FIFO and AXI4 FIFO XCO Parameter Comparison *(Cont'd)***

	FIFO Generator XCO Parameter Prior to v7.2	FIFO Generator XCO Parameter from v7.2 and Later	Possible Values
128	N/A	programmable_full_type_rdch	
129	N/A	programmable_full_type_wach	
130	N/A	programmable_full_type_wdch	
131	N/A	programmable_full_type_wrch	
132	N/A	full_threshold_assert_value_axis	5 - 65535
133	N/A	full_threshold_assert_value_rach	5 - 65535
134	N/A	full_threshold_assert_value_rdch	5 - 65535
135	N/A	full_threshold_assert_value_wach	5 - 65535
136	N/A	full_threshold_assert_value_wdch	5 - 65535
137	N/A	full_threshold_assert_value_wrch	5 - 65535
138	N/A	programmable_empty_type_axis	No_Programmable_Full_Threshold, Single_Programmable_Empty _Threshold_Constant, Single_Programmable_Empty _Threshold_Input_Port
139	N/A	programmable_empty_type_rach	
140	N/A	programmable_empty_type_rdch	
141	N/A	programmable_empty_type_wach	
142	N/A	programmable_empty_type_wdch	
143	N/A	programmable_empty_type_wrch	
144	N/A	empty_threshold_assert_value_axis	4 - 65534
145	N/A	empty_threshold_assert_value_rach	4 - 65534
146	N/A	empty_threshold_assert_value_rdch	4 - 65534
147	N/A	empty_threshold_assert_value_wach	4 - 65534
148	N/A	empty_threshold_assert_value_wdch	4 - 65534
149	N/A	empty_threshold_assert_value_wrch	4 - 65534
150	N/A	underflow_flag_axi	True, False
151	N/A	underflow_sense_axi	Active_High, Active_Low
152	N/A	overflow_flag_axi	True, False
153	N/A	overflow_sense_axi	Active_High, Active_Low
154	N/A	enable_common_overflow	True, False
155	N/A	enable_common_underflow	True, False
156	N/A	disable_timing_violations_axi	True, False
157	N/A	add_ngc_constraint_axi	True, False

DOUT Reset Value Timing

Figure G-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).

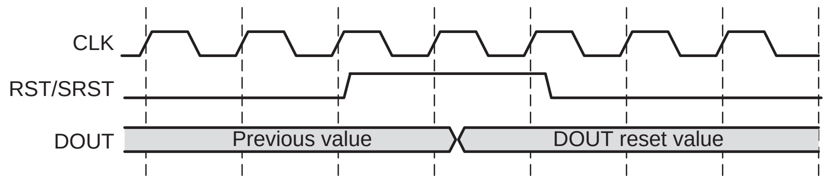


Figure G-1: **DOUT Reset Value for Synchronous Reset (SRST) and for Asynchronous Reset (RST) for Common Clock Block RAM Based FIFO**

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

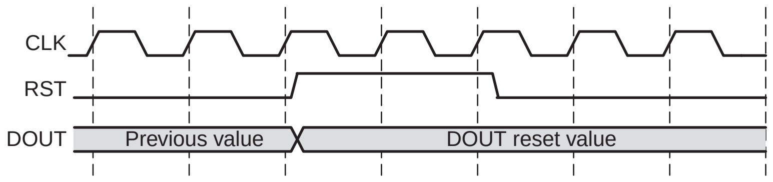


Figure G-2: **DOUT Reset Value for Asynchronous Reset (RST) for Common Clock Distributed/ Shift RAM Based FIFO**

Figure G-3 shows the DOUT reset value for Kintex-7, Virtex-7, and Virtex-6 FPGA common clock Built-in FIFOs with Embedded register for asynchronous reset (RST).

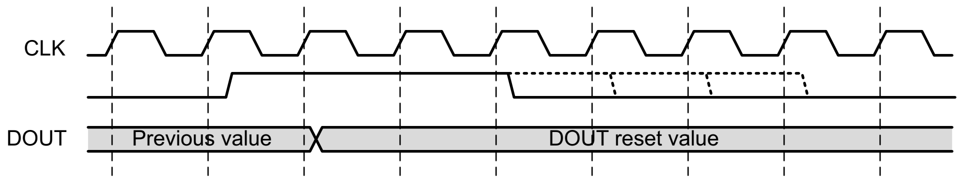


Figure G-3: **DOUT Reset Value for Common Clock Built-in FIFO**

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

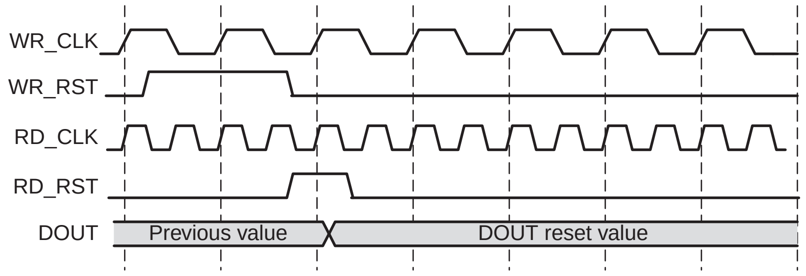


Figure G-4: **DOUT Reset Value for Independent Clock Block RAM Based FIFO**

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

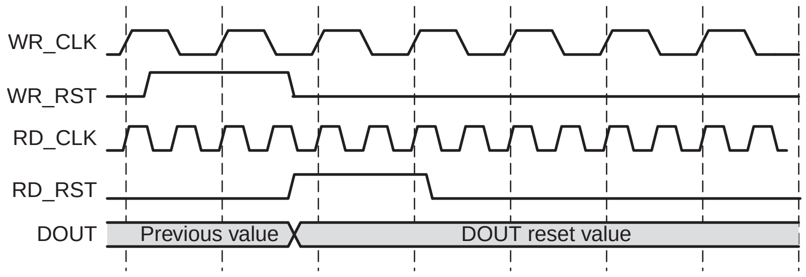


Figure G-5: **DOUT Reset Value for Independent Clock Distributed RAM Based FIFO**

Supplemental Information

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

Auto-Upgrade Feature

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

There are two types of upgrades that you can perform: chose what version to upgrade to, or automatically upgrade the core to the latest version:

- Select Upgrade Version, and Regenerate (Under Current Project Settings): This upgrades an older FIFO Generator core version (4.4, 5.1, 5.2, 5.3, 6.1, 6.2, 7.2, 8.1, 8.2, 8.3. 8.4 or 9.1) to the intermediate version you select -- v5.1, 5.2, 5.3, 6.1, 6.2, 7.2, 8.1, 8.2, 8.3, 8.4, 9.1 or 9.2.
- Upgrade to Latest Version, and Regenerate (Under Current Project Settings): This automatically upgrades an older FIFO Generator core to the latest version. Use this option to upgrade any earlier version of FIFO Generator (4.4, 5.1, 5.2, 5.3, 6.1, 6.2, 7.2, 8.1, 8.2, 8.3, 8.4, 9.1 and 9.2) to v9.3.

Native FIFO SIM Parameters

Table H-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 Generator dynamically. This parameter list does not apply to a core generated using the CORE Generator GUI.

Table H-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 – 23)

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

	SIM Parameter	Type	Description
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
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-5 or Virtex-6)
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

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

	SIM Parameter	Type	Description
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 • 3: Virtex-4 Built-in FIFO • 4: Virtex-5 Built-in FIFO • 5: Virtex-6 Built-in FIFO
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	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[a]
29	C_PROG_EMPTY_THRESH_NEGATE_VAL	Integer	PROG_EMPTY negate threshold[(a)]

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

	SIM Parameter	Type	Description
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 [(a)]
32	C_PROG_FULL_THRESH_NEGATE_VAL	Integer	PROG_FULL negate threshold [(a)]
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 - 23)
35	C_RD_DEPTH	Integer	Depth of read interface (16 – 4194305)
36	C_RD_FREQ	Integer	Read clock frequency (1 MHz - 1000 MHz)
37	C_RD_PNTR_WIDTH	Integer	log2(C_RD_DEPTH)
38	C_UNDERFLOW_LOW	Integer	• 0: UNDERFLOW active high • 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 ECC feature
41	C_USE_EMBEDDED_REG	Integer	• 0: Does not use BRAM embedded output register • 1: Uses BRAM embedded output register
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
43	C_VALID_LOW	Integer	• 0: VALID active high • 1: VALID active low
44	C_WR_ACK_LOW	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 – 23)
46	C_WR_DEPTH	Integer	Depth of write interface (16 – 4194305)
47	C_WR_FREQ	Integer	Write clock frequency (1 MHz - 1000 MHz)
48	C_WR_PNTR_WIDTH	Integer	log2(C_WR_DEPTH)

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

AXI4 FIFO SIM Parameters

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

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

	SIM Parameter	Type	Description
1	C_INTERFACE_TYPE	Integer	• 0: Native FIFO • 1: AXI4 FIFO
2	C_AXI_TYPE	Integer	• 0: AXI4-Stream • 1: AXI4 • 2: AXI4-Lite
3	C_HAS_AXI_WR_CHANNEL	Integer	• 0: Core does not have Write Channel[a] • 1: Core has Write Channel[a]
4	C_HAS_AXI_RD_CHANNEL	Integer	• 0: Core does not have Read Channel[b] • 1: Core has Read Channel[b]
5	C_HAS_SLAVE_CE[c]	Integer	• 0: Core does not have Slave Interface Clock Enable • 1: Core has Slave Interface Clock Enable
6	C_HAS_MASTER_CE[c]	Integer	• 0: Core does not have Master Interface Clock Enable • 1: Core has Master Interface Clock Enable
7	C_ADD_NGC_CONSTRAINT[c]	Integer	• 0: Core does not add NGC constraint • 1: Core adds NGC constraint
8	C_USE_COMMON_UNDERFLOW[c]	Integer	• 0: Core does not have common UNDERFLOW flag • 1: Core has common UNDERFLOW flag
9	C_USE_COMMON_OVERFLOW[c]	Integer	• 0: Core does not have common OVERFLOW flag • 1: Core has common OVERFLOW flag
10	C_USE_DEFAULT_SETTINGS[c]	Integer	• 0: Core does not use default settings • 1: Core uses default settings
11	C_AXI_ID_WIDTH	Integer	ID Width
12	C_AXI_ADDR_WIDTH	Integer	Address Width
13	C_AXI_DATA_WIDTH	Integer	Data Width
14	C_HAS_AXI_AWUSER	Integer	• 0: Core does not have AWUSER • 1: Core has AWUSER
15	C_HAS_AXI_WUSER	Integer	• 0: Core does not have WUSER • 1: Core has WUSER

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

	SIM Parameter	Type	Description
16	C_HAS_AXI_BUSER	Integer	• 0: Core does not have BUSER • 1: Core has BUSER
17	C_HAS_AXI_ARUSER	Integer	• 0: Core does not have ARUSER • 1: Core has ARUSER
18	C_HAS_AXI_RUSER	Integer	• 0: Core does not have RUSER • 1: Core has RUSER
19	C_AXI_AWUSER_WIDTH	Integer	AWUSER Width
20	C_AXI_WUSER_WIDTH	Integer	WUSER Width
21	C_AXI_BUSER_WIDTH	Integer	BUSER Width
22	C_AXI_ARUSER_WIDTH	Integer	ARUSER Width
23	C_AXI_RUSER_WIDTH	Integer	RUSER Width
24	C_HAS_AXIS_TDATA	Integer	• 0: AXI4 Stream does not have TDATA • 1: AXI4 Stream has TDATA
25	C_HAS_AXIS_TID	Integer	• 0: AXI4 Stream does not have TID • 1: AXI4 Stream has TID
26	C_HAS_AXIS_TDEST	Integer	• 0: AXI4 Stream does not have TDEST • 1: AXI4 Stream has TDEST
27	C_HAS_AXIS_TUSER	Integer	• 0: AXI4 Stream does not have TUSER • 1: AXI4 Stream has TUSER
28	C_HAS_AXIS_TREADY	Integer	• 0: AXI4 Stream does not have TREADY • 1: AXI4 Stream has TREADY
29	C_HAS_AXIS_TLAST	Integer	• 0: AXI4 Stream does not have TLAST • 1: AXI4 Stream has TLAST
30	C_HAS_AXIS_TSTRB	Integer	• 0: AXI4 Stream does not have TSTRB • 1: AXI4 Stream has TSTRB
31	C_HAS_AXIS_TKEEP	Integer	• 0: AXI4 Stream does not have TKEEP • 1: AXI4 Stream has TKEEP
32	C_AXIS_TDATA_WIDTH	Integer	AXI4 Stream TDATA Width
33	C_AXIS_TID_WIDTH	Integer	AXI4 Stream TID Width
34	C_AXIS_TDEST_WIDTH	Integer	AXI4 Stream TDEST Width
35	C_AXIS_TUSER_WIDTH	Integer	AXI4 Stream TUSER Width
36	C_AXIS_TSTRB_WIDTH	Integer	AXI4 Stream TSTRB Width
37	C_AXIS_TKEEP_WIDTH	Integer	AXI4 Stream TKEEP Width
38	C_WACH_TYPE	Integer	Write Address Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic

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

	SIM Parameter	Type	Description
39	C_WDCH_TYPE	Integer	Write Data Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
40	C_WRCH_TYPE	Integer	Write Response Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
41	C_RACH_TYPE	Integer	Read Address Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
42	C_RDCH_TYPE	Integer	Read Data Channel type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
43	C_AXIS_TYPE	Integer	AXI4 Stream type • 0: FIFO • 1: Register Slice • 2: Pass Through Logic
44	C_REG_SLICE_MODE_WACH	Integer	Write Address Channel configuration type • 0: Fully Registered • 1: Light Weight
45	C_REG_SLICE_MODE_WDCH	Integer	Write Data Channel configuration type • 0: Fully Registered • 1: Light Weight
46	C_REG_SLICE_MODE_WRCH	Integer	Write Response Channel configuration type • 0: Fully Registered • 1: Light Weight
47	C_REG_SLICE_MODE_RACH	Integer	Read Address Channel configuration type • 0: Fully Registered • 1: Light Weight
48	C_REG_SLICE_MODE_RDCH	Integer	Read Data Channel configuration type • 0: Fully Registered • 1: Light Weight
49	C_REG_SLICE_MODE_AXIS	Integer	AXI4 Stream configuration type • 0: Fully Registered • 1: Light Weight

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

	SIM Parameter	Type	Description
50	C_IMPLEMENTATION_TYPE_WACH	Integer	Write Address Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO
51	C_IMPLEMENTATION_TYPE_WDCH	Integer	Write Data Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO
52	C_IMPLEMENTATION_TYPE_WRCH	Integer	Write Response Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO
53	C_IMPLEMENTATION_TYPE_RACH	Integer	Read Address Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO
54	C_IMPLEMENTATION_TYPE_RDCH	Integer	Read Data Channel Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO
55	C_IMPLEMENTATION_TYPE_AXIS	Integer	AXI4 Stream Implementation type • 1: Common Clock Block RAM FIFO • 2: Common Clock Distributed RAM FIFO • 11: Independent Clock Block RAM FIFO • 12: Independent Clock Distributed RAM FIFO
56	C_APPLICATION_TYPE_WACH	Integer	Write Address Channel Application type • 0: Data FIFO • 1: Packet FIFO[(c)] • 2: Low Latency Data FIFO
57	C_APPLICATION_TYPE_WDCH	Integer	Write Data Channel Application type • 0: Data FIFO • 1: Packet FIFO[(c)] • 2: Low Latency Data FIFO

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

	SIM Parameter	Type	Description
58	C_APPLICATION_TYPE_WRCH	Integer	Write Response Channel Application type • 0: Data FIFO • 1: Packet FIFO(c) • 2: Low Latency Data FIFO
59	C_APPLICATION_TYPE_RACH	Integer	Read Address Channel Application type • 0: Data FIFO • 1: Packet FIFO(c) • 2: Low Latency Data FIFO
60	C_APPLICATION_TYPE_RDCH	Integer	Read Data Channel Application type • 0: Data FIFO • 1: Packet FIFO(c) • 2: Low Latency Data FIFO
61	C_APPLICATION_TYPE_AXIS	Integer	AXI4 Stream Application type • 0: Data FIFO • 1: Packet FIFO(c) • 2: Low Latency Data FIFO
62	C_USE_ECC_WACH	Integer	• 0: ECC option not used for Write Address Channel • 1: ECC option used for Write Address Channel
63	C_USE_ECC_WDCH	Integer	• 0: ECC option not used for Write Data Channel • 1: ECC option used for Write Data Channel
64	C_USE_ECC_WRCH	Integer	• 0: ECC option not used for Write Response Channel • 1: ECC option used for Write Response Channel
65	C_USE_ECC_RACH	Integer	• 0: ECC option not used for Read Address Channel • 1: ECC option used for Read Address Channel
66	C_USE_ECC_RDCH	Integer	• 0: ECC option not used for Read Data Channel • 1: ECC option used for Read Data Channel
67	C_USE_ECC_AXIS	Integer	• 0: ECC option not used for AXI4 Stream • 1: ECC option used for AXI4 Stream
68	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

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

	SIM Parameter	Type	Description
69	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
70	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
71	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
72	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
73	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
74	C_DIN_WIDTH_WACH	Integer	DIN Width of Write Address Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID).
75	C_DIN_WIDTH_WDCH	Integer	DIN Width of Write Data Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID).
76	C_DIN_WIDTH_WRCH	Integer	DIN Width of Write Response Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID).
77	C_DIN_WIDTH_RACH	Integer	DIN Width of Read Address Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID).

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

	SIM Parameter	Type	Description
78	C_DIN_WIDTH_RDCH	Integer	DIN Width of Read Data Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID).
79	C_DIN_WIDTH_AXIS	Integer	DIN Width of AXI4 Stream bus (1 - 1024) Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID).
80	C_WR_DEPTH_WACH	Integer	FIFO Depth of Write Address Channel
81	C_WR_DEPTH_WDCH	Integer	FIFO Depth of Write Data Channel
82	C_WR_DEPTH_WRCH	Integer	FIFO Depth of Write Response Channel
83	C_WR_DEPTH_RACH	Integer	FIFO Depth of Read Address Channel
84	C_WR_DEPTH_RDCH	Integer	FIFO Depth of Read Data Channel
85	C_WR_DEPTH_AXIS	Integer	FIFO Depth of AXI4 Stream
86	C_WR_PNTR_WIDTH_WACH	Integer	Log_2(C_WR_DEPTH_WACH)
87	C_WR_PNTR_WIDTH_WDCH	Integer	Log_2(C_WR_DEPTH_WDCH)
88	C_WR_PNTR_WIDTH_WRCH	Integer	Log_2(C_WR_DEPTH_WRCH)
89	C_WR_PNTR_WIDTH_RACH	Integer	Log_2(C_WR_DEPTH_RACH)
90	C_WR_PNTR_WIDTH_RDCH	Integer	Log_2(C_WR_DEPTH_RDCH)
91	C_WR_PNTR_WIDTH_AXIS	Integer	Log_2(C_WR_DEPTH_AXIS)
92	C_HAS_DATA_COUNTS_WACH	Integer	Write Address Channel • 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
93	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
94	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

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

	SIM Parameter	Type	Description
95	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
96	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
97	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
98	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
99	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
100	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
101	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

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

	SIM Parameter	Type	Description
102	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
103	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
104	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
105	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
106	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
107	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
108	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
109	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
110	C_PROG_FULL_THRESH_ASSERT_VAL_WACH	Integer	PROG_FULL assert threshold[d] for Write Address Channel
111	C_PROG_FULL_THRESH_ASSERT_VAL_WDCH	Integer	PROG_FULL assert threshold[(d)] for Write Data Channel

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

	SIM Parameter	Type	Description
112	C_PROG_FULL_THRESH_ASSERT_VAL_WRCH	Integer	PROG_FULL assert threshold (d) for Write Response Channel
113	C_PROG_FULL_THRESH_ASSERT_VAL_RACH	Integer	PROG_FULL assert threshold (d) for Read Address Channel

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

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

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

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

Additional Resources

Xilinx Resources

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

www.xilinx.com/support.

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

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

References

These documents provide supplemental material useful with this product guide:

1. *AMBA® AXI4-Stream Protocol Specification*
2. *AXI4 AMBA® AXI Protocol Version: 2.0 Specification*
3. UG911, *Vivado Design Suite Migration Methodology Guide*
4. UG070, *Virtex-4 FPGA User Guide*
5. Vivado™ Design Suite user documentation (www.xilinx.com/cgi-bin/docs/rdoc?v=2012.3;t=vivado+docs)

Technical Support

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

See the IP Release Notes Guide (XTP025) for more information on this core. For each core, there is a master Answer Record that contains the Release Notes and Known Issues list for the core being used. The following information is listed for each version of the core:

- New Features
- Resolved Issues
- Known Issues

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
07/25/12	1.0	Initial release of this document as a product guide. This document replaces DS317, *FIFO Generator Data Sheet*, UG175, *FIFO Generator User Guide*, and XAPP992, *FIFO Generator Migration Guide*.
10/16/12	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.
12/18/12	3.0	Updated for core v9.3, Vivado Design Suite v2012.3, and ISE Design Suite v14.3. Added Appendix C, Debugging.

Notice of Disclaimer