LogiCORE IP FIFO Generator v8.2

DS317 June 22, 2011 **Product Specification**

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 Features, page 2.

LogiCORE IP Facts	
Core Specifics	
Supported FPGA Device Families[1]	Zynq-7000, 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	AXI4-Stream, AXI4, AXI4-Lite
Performance and Resources Used	See Table 20 through Table 26
Provided with Core	
Documentation	Product Specification User Guide Release Notes Migration Guide[2]
Design File Formats	NGC
Instantiation Template	VHDL, Verilog
Design Tool Requirements	
Implementation	Xilinx ISE v13.2
Simulation[3]	Mentor Graphics ModelSim Cadence Incisive Enterprise Simulator
Supported Simulation Models	Verilog Behavioral[4] VHDL Behavioral[4] Verilog Structural VHDL Structural
Support	
Provided by Xilinx, Inc.	

1. For the complete list of supported devices, see Table 2, page 6, Table 6, page 17 and the release notes for this core.
2. The Migration Guide provides instructions for converting legacy Asynchronous FIFO and Synchronous FIFO LogiCORE IP cores to FIFO Generator cores.
3. For the supported versions of the tools, see the ISE Design Suite 13: Release Notes Guide.
4. Behavioral models do not model synchronization delay. See the "Simulating Your Design" section of UG175, *FIFO Generator User Guide* for details.

Features

Common Features

- Supports Native, AXI4-Stream, AXI4 and AXI4-Lite interfaces
- FIFO depths up to 4,194,304 words
- FIFO data widths from 1 to 1024 bits
- Independent or common clock domains
- Fully configurable using the Xilinx CORE Generator

Native FIFO Specific Features

- 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

- 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 or low latency FIFO)
- Operates in First-Word Fall-Through mode (FWFT)
- Configurable Ready and Valid handshake signals mappable to Native FIFO Full and Empty flags, to Almost Full and Almost Empty flags for one-word-left, as well as to Programmable Full and Empty levels
- 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

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.

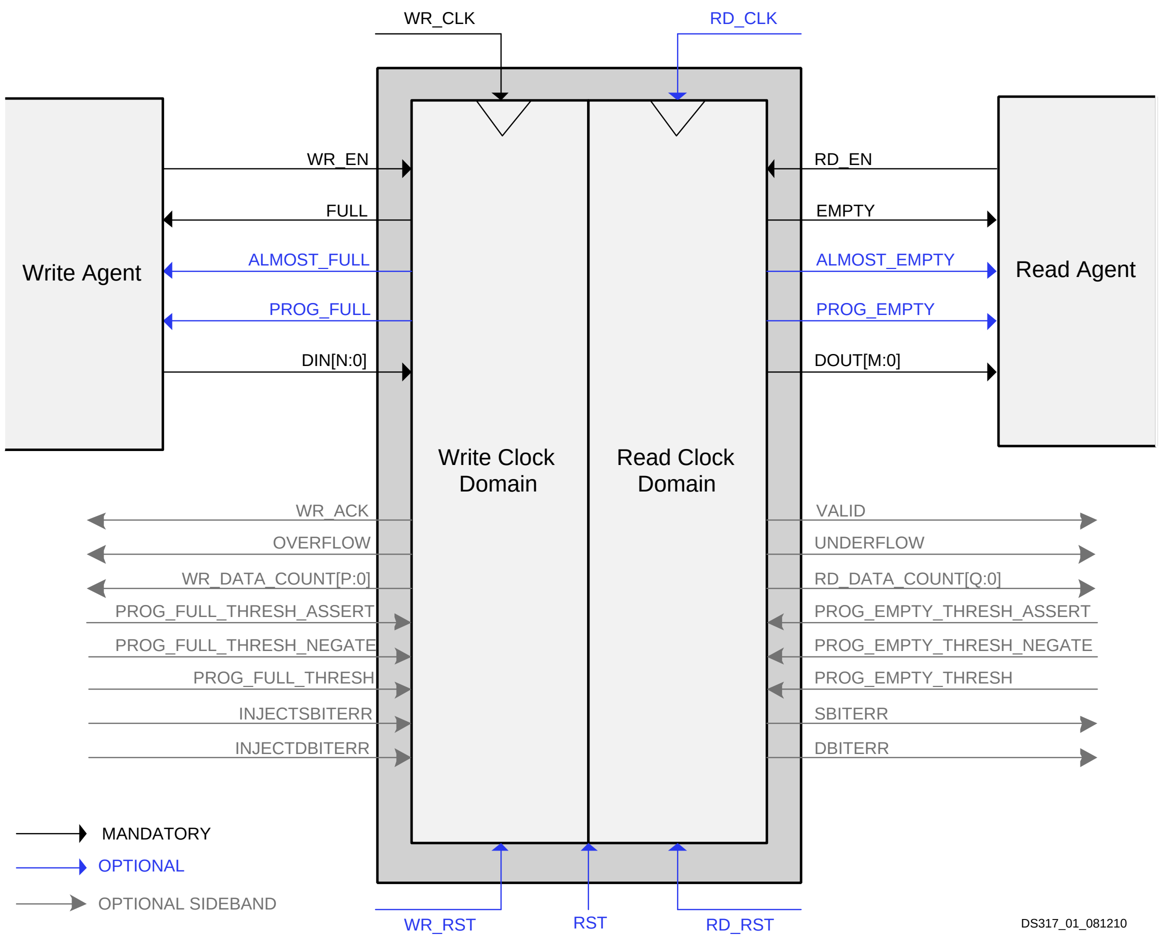

Figure 1: **Native FIFOs Signals**

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 2 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, the user is 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.

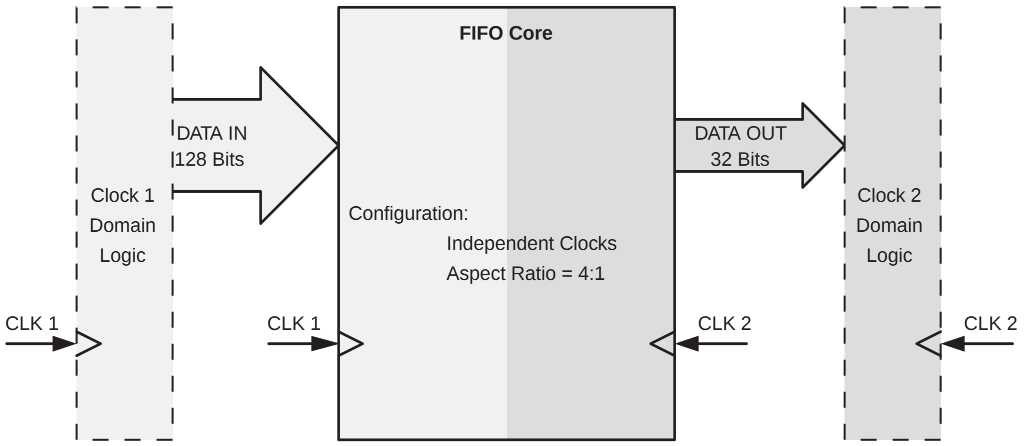

Figure 2: **FIFO Generator Application Example**

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 the user to implement unique clock domains on the write and read ports. The FIFO Generator handles the synchronization between clock domains, placing no requirements on phase and frequency. When data buffering in a single clock domain is required, the FIFO Generator can be used to generate a core optimized for that 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 UG070, *Virtex-4 FPGA User Guide*.

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 or Virtex-5 devices.

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. The following table 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*.

Table 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 in UG175, *FIFO Generator User Guide* 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 2 shows the families and sub-families supported by the Native FIFO Generator. For more details about device support, see the Release Notes.

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

FPGA Family	Sub-Family
Spartan-3	
Spartan-3E	
Spartan-3A	
Spartan-3AN	
Spartan-3A DSP	
Spartan-6	LX/LXT
Virtex-4	LX/FX/SX
Virtex-5	LXT/FXT/SXT/TXT
Virtex-6	CXT/HXT/LXT/SXT
Virtex-7	
Kintex-7	
Artix-7	
Zynq-7000	

Native FIFO Configuration and Implementation

Table 3 defines the supported memory and clock configurations.

Table 3: **FIFO Configurations**

Clock Domain	Memory Type	Non-symmetric Aspect Ratios	First-word Fall-Through	ECC Support	Embedded Register Support
Common	Block RAM		✓	✓	✓ (1)
Common	DistributedRAM		✓		
Common	Shift Register				
Common	Built-in FIFO(2)		✓ (3)	✓	✓ (1)
Independent	Block RAM	✓	✓	✓	✓ (1)
Independent	Distributed RAM		✓		
Independent	Built-in FIFO(2), (4)		✓ (3)	✓	

1. 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.
2. The built-in FIFO primitive is only available in the Vortex-6, Virtex-5 and Virtex-4 architectures.
3. FWFT is supported for Built-in FIFOs in Zynq-7000, 7 series, Virtex-6 and Virtex-5 devices only.
4. 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 the user to select block RAM, distributed RAM, or shift register and supports a common clock for write and read data accesses. The feature set supported for this configuration includes status flags (full, almost full, empty, and almost empty) and programmable empty and full flags generated with user-defined thresholds.

In addition, optional handshaking and error flags are supported (write acknowledge, overflow, valid, and underflow), and an optional data count provides the number of words in the FIFO. In addition, for the block RAM and distributed RAM implementations, the user has the option to select a synchronous or asynchronous reset for the core. For 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 the user 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 the user to select block RAM or distributed RAM and supports independent clock domains for write and read data accesses. Operations in the read domain are synchronous to the read clock and operations in the write domain are synchronous to the write clock.

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

Table 4 summarizes the supported FIFO Generator features for each clock configuration and memory type. For detailed information, see UG175, *FIFO Generator User Guide*.

Table 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(1)	✓					
Symmetric Aspect Ratios	✓	✓	✓	✓	✓	✓
Almost Full	✓	✓		✓	✓	
Almost Empty	✓	✓		✓	✓	
Handshaking	✓	✓	✓	✓	✓	✓
Data Count	✓	✓		✓	✓	
Programmable Empty/Full Thresholds	✓	✓	✓(2)	✓	✓	✓(2)
First-Word Fall-Through(3)	✓	✓	✓	✓	✓	✓
Synchronous Reset				✓	✓	
Asynchronous Reset	✓(4)	✓(4)	✓	✓(4)	✓(4)	✓
DOUT Reset Value	✓	✓		✓	✓	✓(5)
ECC	✓(6)		✓(6)	✓(6)		✓(6)
Embedded Register	✓(7)			✓(7)		✓(7)

1. For applications with a single clock that require non-symmetric ports, use the independent clock configuration and connect the write and read clocks to the same source. A dedicated solution for common clocks will be available in a future release. Contact your Xilinx representative for more details.
2. For built-in FIFOs, the range of Programmable Empty/Full threshold is limited to take advantage of the logic internal to the macro.
3. First-Word-Fall-Through is not supported for the shift RAM FIFOs and Virtex-4 built-in FIFOs.
4. Asynchronous reset is optional for all FIFOs built using distributed and block RAM.
5. DOUT Reset Value is supported only in Zynq-7000, 7 series, and Virtex-6 FPGA common clock built-in FIFOs.
6. ECC is only supported for the Zynq-7000, 7 series, Virtex-6 and Virtex-5 FPGAs and block RAM and built-in FIFOs.
7. 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.

Native FIFO Port Summary

Table 5 describes all the FIFO Generator ports. For detailed information about any of the ports, see Chapter 3, Core Architecture, in the *FIFO Generator User Guide*.

Table 5: **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

Table 5: **FIFO Generator Ports *(Cont'd)***

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

AXI4 Interface FIFOs

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

Use the AXI4 FIFOs in the same applications supported by the Native Interface FIFO when you need to connect to other AXI functions. 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.

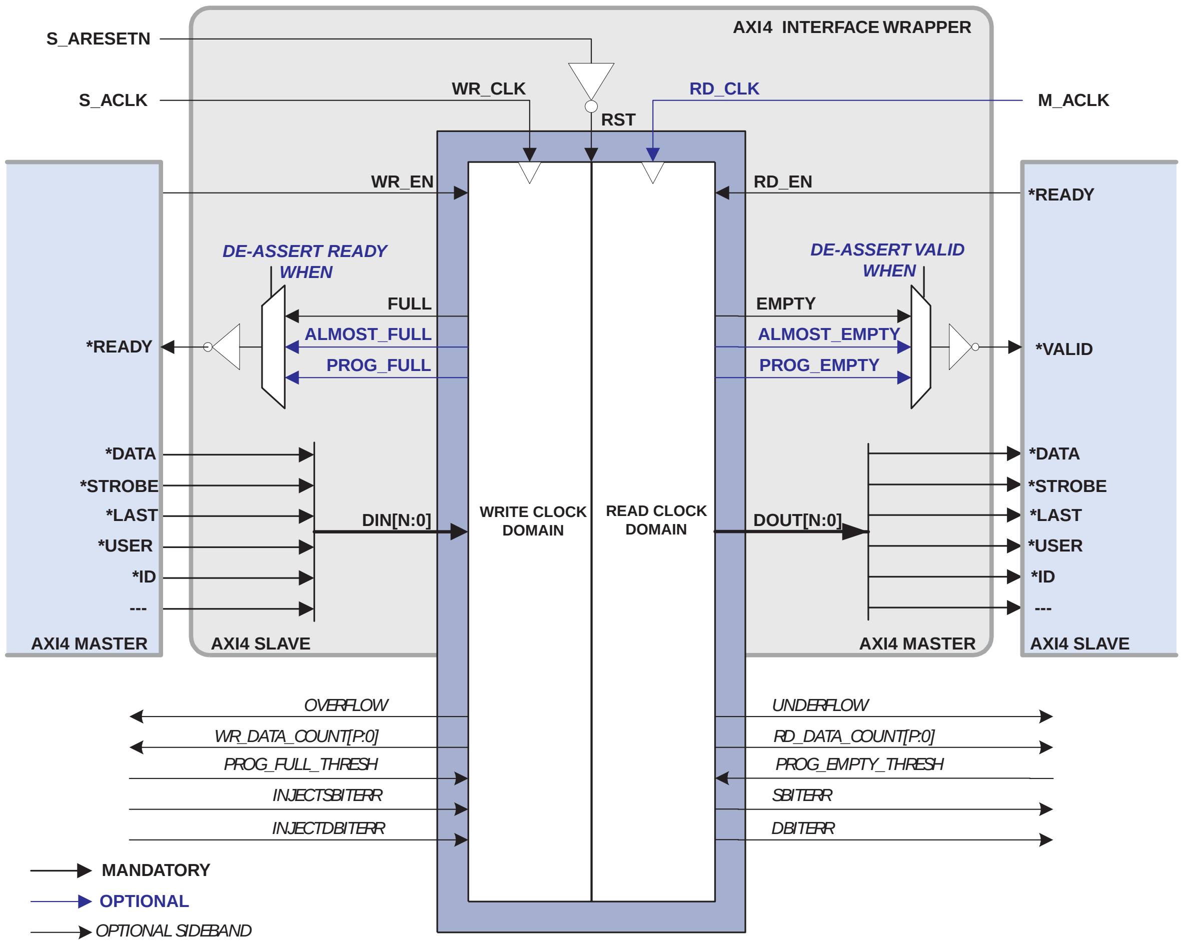

Figure 3: **AXI4 FIFO Derivation**

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 4 shows an example timing diagram for write and read operations to the AXI4 FIFO.

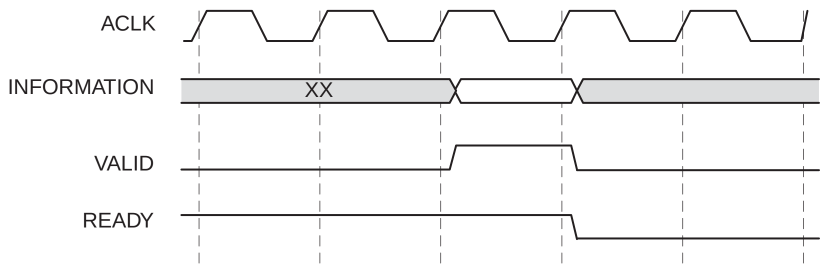

Figure 4: **AXI4-FIFO Timing Diagram**

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

Because 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.

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).

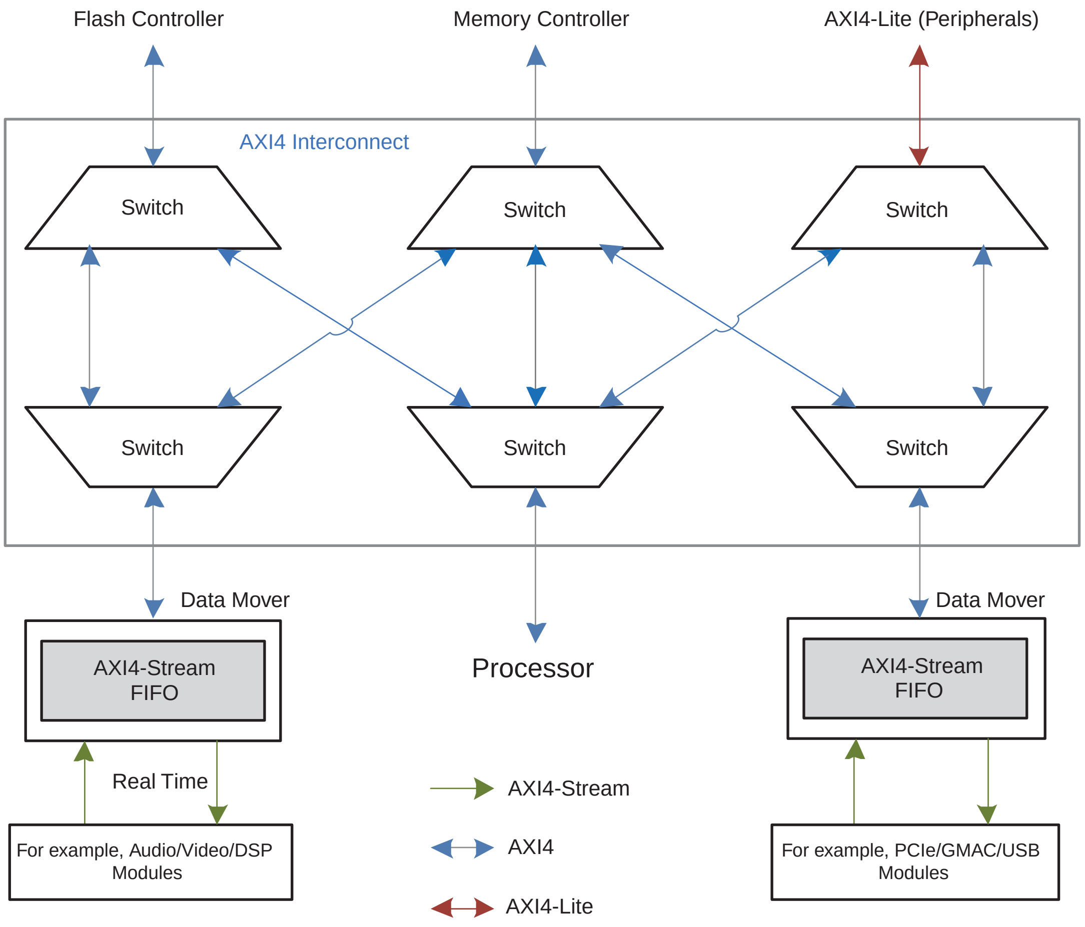

Figure 5: **AXI4-Stream Application Diagram**

Figure 5 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 5 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.

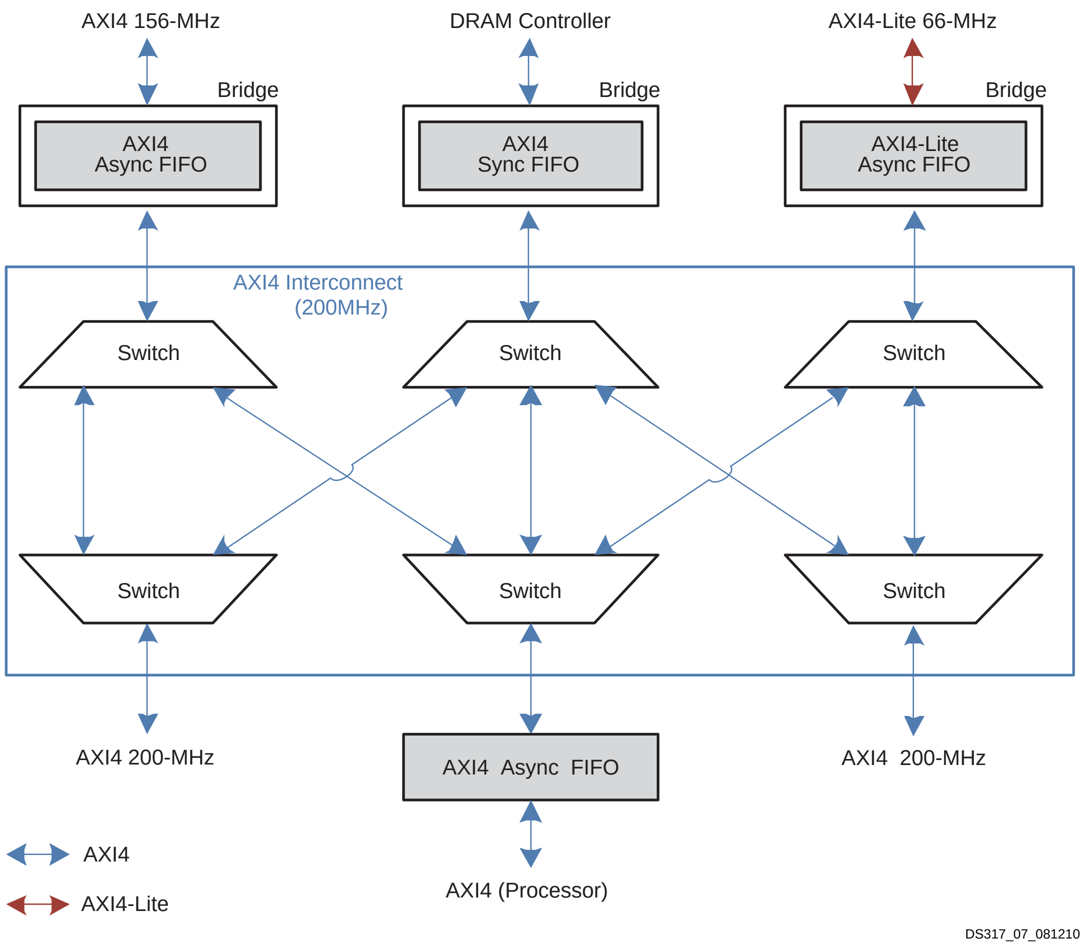

Figure 6: **AXI4 Application Diagram**

Figure 6 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 6 is an Embedded IP core which is available in the Xilinx Embedded Development Kit (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 7 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 7 is also available as an Embedded IP core in the Xilinx Embedded Development Kit (EDK).

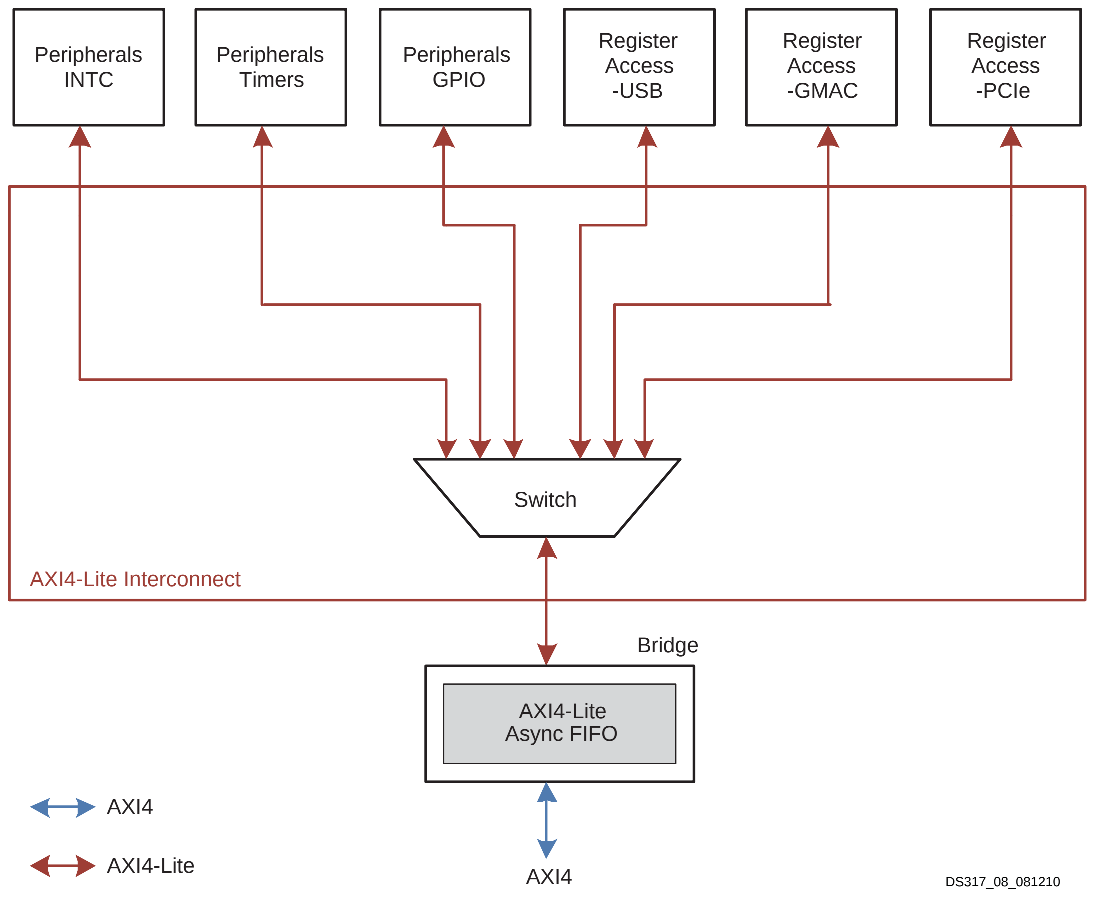

Figure 7: **AXI4-Lite Application Diagram**

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 8. 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.

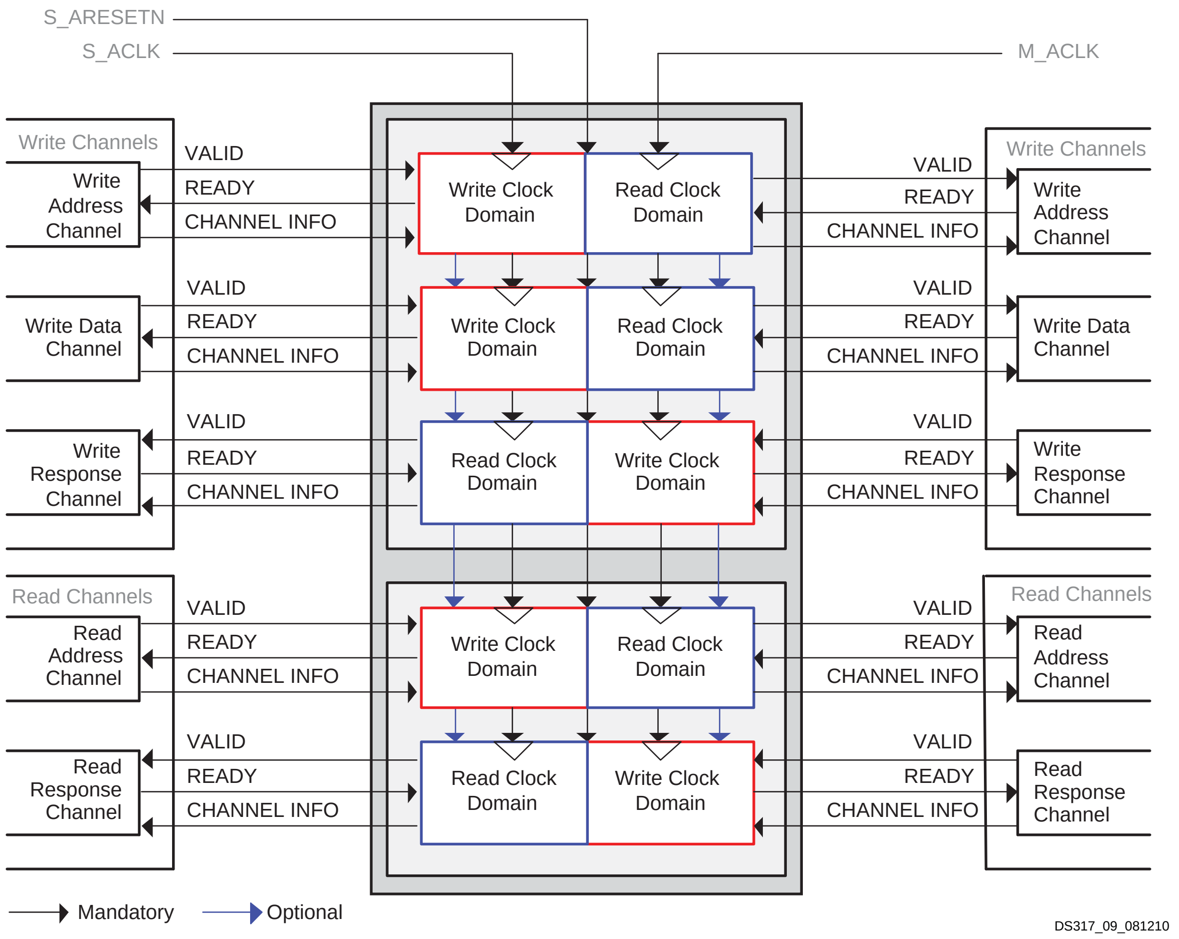

Figure 8: **AXI4 Block Diagram**

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 8). In addition, all instantiated FIFOs can support either independent clock or common clock operation.

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

Automatic FIFO Width Calculation

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

For more details on width calculation, refer to UG175, *FIFO Generator User Guide*.

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 or low latency FIFO), the core combines memory primitives in an optimal configuration based on the calculated width and selected depth of the FIFO.

Error Injection and Correction (ECC) Support

The block RAM macros are equipped with built-in Error Injection and Correction Checking in the 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 UG175, *FIFO Generator User Guide*.

AXI4 Slave Interface for Performing Writes

AXI4 FIFOs provide an AXI4 Slave interface for performing Writes. In Figure 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 either Full, Almost Full or when the Programmable Full threshold is reached. The de-assertion of READY can be controlled by setting “Deassert READY When” option.

AXI4 Master Interface for Performing Reads

The AXI4 FIFO provides an AXI4 Master interface for performing Reads. In Figure 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 either Empty, Almost Empty or when the FIFO Occupancy is less than the Programmable Empty threshold. The de-assertion of VALID can be controlled by setting the “Deassert VALID When” option.

AXI4 FIFO Supported Devices

Table 6 shows the families and sub-families supported by the FIFO Generator. For more details about device support, see the Release Notes.

Table 6: **Supported FPGA Families and Sub-Families**

FPGA Family	Sub-Family
Spartan-6	LX/LXT
Virtex-6	CXT/HXT/LXT/SXT
Virtex-7	
Kintex-7	
Artix-7	
Zynq-7000	

AXI4 FIFO Feature Summary

Table 7 summarizes the supported FIFO Generator features for each clock configuration and memory type. For detailed information, see UG175, *FIFO Generator User Guide*.

Table 7: **AXI4 FIFO Configuration Summary**

FIFO Options	Common Clock		Independent Clock	
	Block RAM	Distributed Memory	Block RAM	Distributed Memory
Full[(1)]	✓	✓	✓	✓
Almost Full[(1)]	✓	✓	✓	✓
Programable Full[(1)]	✓	✓	✓	✓
Empty[(2)]	✓	✓	✓	✓
Almost Empty[(2)]	✓	✓	✓	✓
Programable Empty[(2)]	✓	✓	✓	✓
Data Counts	✓	✓	✓	✓
ECC	✓		✓	
Interrupt Flags	✓	✓	✓	✓

1. Mapped to S_AXIS_TREADY/S_AXI_AWREADY/S_AXI_WREADY/M_AXI_BREADY/S_AXI_ARREADY/M_AXI_RREADY depending on the Handshake Flag Options in the GUI.
2. 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 Port Summary

AXI4 Global Interface Ports

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

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

AXI4-Stream FIFO Interface Ports

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

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

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

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

AXI4 FIFO Interface Ports

Write Channels

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

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

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

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
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	No	Yes
AXI_AW_UNDERFLOW	Output	Yes	Yes	Yes
AXI_AW_RD_DATA_COUNT[m:0]	Output	Yes	No	Yes
AXI_AW_DATA_COUNT[m:0]	Output	Yes	Yes	No

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

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

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

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI4 Write Data Channel FIFO: Optional Sideband Signals				
AXI_W_PROG_FULL_THRESH[m:0]	Input	Yes	Yes	Yes
AXI_W_PROG_EMPTY_THRESH[m:0]	Input	Yes	Yes	Yes
AXI_W_INJECTSBITERR	Input	Yes	Yes	Yes
AXI_W_INJECTDBITERR	Input	Yes	Yes	Yes
AXI_W_SBITERR	Output	Yes	Yes	Yes
AXI_W_DBITERR	Output	Yes	Yes	Yes
AXI_W_OVERFLOW	Output	Yes	Yes	Yes
AXI_W_WR_DATA_COUNT[m:0]	Output	Yes	No	Yes
AXI_W_UNDERFLOW	Output	Yes	Yes	Yes
AXI_W_RD_DATA_COUNT[m:0]	Output	Yes	No	Yes
AXI_W_DATA_COUNT[m:0]	Output	Yes	Yes	No

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

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

Table 12: **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_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	No	Yes
AXI_B_UNDERFLOW	Output	Yes	Yes	Yes
AXI_B_RD_DATA_COUNT[m:0]	Output	Yes	No	Yes
AXI_B_DATA_COUNT[m:0]	Output	Yes	Yes	No

Read Channels

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

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI4 Interface Read Address Channel: Information Signals Mapped to FIFO Data Input (DIN) Bus				
S_AXI_ARID[m:0]	Input	No	Yes	Yes
S_AXI_ARADDR[m:0]	Input	No	Yes	Yes
S_AXI_ARLEN[7:0]	Input	No	Yes	Yes
S_AXI_ARSIZE[2:0]	Input	No	Yes	Yes
S_AXI_ARBURST[1:0]	Input	No	Yes	Yes
S_AXI_ARLOCK[2:0]	Input	No	Yes	Yes
S_AXI_ARCACHE[4:0]	Input	No	Yes	Yes
S_AXI_ARPROT[3:0]	Input	No	Yes	Yes
S_AXI_ARQOS[3:0]	Input	No	Yes	Yes
S_AXI_ARREGION[3:0]	Input	No	Yes	Yes
S_AXI_ARUSER[m:0]	Input	Yes	Yes	Yes
AXI4 Interface Read Address Channel: Handshake Signals for FIFO Write Interface				
S_AXI_ARVALID	Input	No	Yes	Yes
S_AXI_ARREADY	Output	No	Yes	Yes
AXI4 Interface, Read Address Channel: Information Signals Derived from FIFO Data Output (DOUT) Bus				
M_AXI_ARID[m:0]	Output	No	Yes	Yes
M_AXI_ARADDR[m:0]	Output	No	Yes	Yes
M_AXI_ARLEN[7:0]	Output	No	Yes	Yes
M_AXI_ARSIZE[2:0]	Output	No	Yes	Yes
M_AXI_ARBURST[1:0]	Output	No	Yes	Yes
M_AXI_ARLOCK[2:0]	Output	No	Yes	Yes
M_AXI_ARCACHE[4:0]	Output	No	Yes	Yes
M_AXI_ARPROT[3:0]	Output	No	Yes	Yes
M_AXI_ARQOS[3:0]	Output	No	Yes	Yes
M_AXI_ARREGION[3:0]	Output	No	Yes	Yes
M_AXI_ARUSER[m:0]	Output	Yes	Yes	Yes
AXI4 Interface Read Address Channel: Handshake Signals for FIFO Read Interface				
M_AXI_ARVALID	Output	No	Yes	Yes
M_AXI_ARREADY	Input	No	Yes	Yes
AXI4 Read Address Channel FIFO: Optional Sideband Signals				
AXI_AR_PROG_FULL_THRESH[m:0]	Input	Yes	Yes	Yes

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

Port Name	Input or Output	Optional Port	Port Available	
			Independent Clocks	Common Clock
AXI_AR_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	No	Yes
AXI_AR_UNDERFLOW	Output	Yes	Yes	Yes
AXI_AR_RD_DATA_COUNT[m:0]	Output	Yes	No	Yes
AXI_AR_DATA_COUNT[m:0]	Output	Yes	Yes	No

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

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

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

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

AXI4-Lite FIFO Interface Ports

Write Channels

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

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

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

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

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

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

Read Channels

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

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

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

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

Resource Utilization and Performance

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. The following tables 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.

Note: 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 20 identifies the results for a FIFO configured without optional features. Benchmarks were performed using Virtex-4 (XC4VLX15-FF668-10), Virtex-5 (XC5VLX50-FF324-1), Virtex-6 (XC6VLX760-FF1760-1) and Spartan®-6 (XC6SLX150T-FGG484-2) FPGAs.

Table 20: **Benchmarks: FIFO Configured without Optional Features**

FIFO Type	Depth x Width	FPGA Family	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Synchronous FIFO (Block RAM)	512 x 16	Kintex-7 (1)						
		Virtex-7 (1)						
		Virtex-6	325	48	50	1	0	0
		Virtex-5	300	44	50	1	0	0
		Virtex-4	250	29	50	1	0	0
		Spartan-6	175	46	50	1	0	0
	4096 x 16	Kintex-7 (1)						
		Virtex-7 (1)						
		Virtex-6	325	56	62	2	0	0
		Virtex-5	325	54	62	2	0	0
		Virtex-4	275	33	62	4	0	0
		Spartan-6	200	57	62	4	0	0

Table 20: **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
Synchronous FIFO (Distributed RAM)	64 x 16	Kintex-7[(1)]						
		Virtex-7[(1)]						
		Virtex-6	400	25	54	0	0	22
		Virtex-5	400	27	54	0	0	22
		Virtex-4	350	70	64	0	0	128
		Spartan-6	175	27	54	0	0	22
	512 x 16	Kintex-7[(1)]						
		Virtex-7[(1)]						
		Virtex-6	325	109	66	0	0	176
		Virtex-5	300	83	66	0	0	256
		Virtex-4	300	326	202	0	0	1024
		Spartan-6	150	104	66	0	0	176
Independent Clocks FIFO (Block RAM)	512 x 16	Kintex-7[(1)]						
		Virtex-7[(1)]						
		Virtex-6	325	79	132	1	0	0
		Virtex-5	325	75	132	1	0	0
		Virtex-4	300	63	132	1	0	0
		Spartan-6	175	70	132	1	0	0
	4096 x 16	Kintex-7[(1)]						
		Virtex-7[(1)]						
		Virtex-6	325	109	171	2	0	0
		Virtex-5	300	106	171	2	0	0
		Virtex-4	275	91	171	4	0	0
		Spartan-6	200	98	171	4	0	0
Independent Clocks FIFO (Distributed RAM	64 x 16	Kintex-7[(1)]						
		Virtex-7[(1)]						
		Virtex-6	425	121	183	0	0	22
		Virtex-5	400	46	109	0	0	22
		Virtex-4	350	92	112	0	0	128
		Spartan-6	200	42	109	0	0	22
	512 x 16	Kintex-7[(1)]						
		Virtex-7[(1)]						
		Virtex-6	350	135	148	0	0	176
		Virtex-5	300	114	148	0	0	256
		Virtex-4	200	352	280	0	0	1024
		Spartan-6	150	129	148	0	0	176

Table 20: **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
Shift Register FIFO[2]	64 x 16	Kintex-7[1]						
		Virtex-7[1]						
		Virtex-5	425	56	44	0	32	0
		Virtex-4	300	66	44	0	64	0
		Spartan-6	175	55	44	0	48	0
	512 x 16	Kintex-7[1]						
		Virtex-7[1]						
		Virtex-5	275	134	53	0	256	0
		Virtex-4	225	324	57	0	512	0
		Spartan-6	75	339	56	0	496	0

1. Benchmarking data for Virtex-7 and Kintex-7 will be available in future release.
2. Virtex-6 benchmarking data for the Shift Register FIFO will be available in a future release.

Table 21 provides results for FIFOs configured with multiple programmable thresholds. Benchmarks were performed using Virtex-4 (XC4VLX15-FF668-10), Virtex-5 (XC5VLX50-FF324-1), Virtex-6 (XC6VLX760-FF1760-1) and Spartan-6 (XC6SLX150T-FGG484-2) FPGAs.

Table 21: **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
Synchronous FIFO (Block RAM)	512 x 16	Kintex-7[1]						
		Virtex-7[1]						
		Virtex-6	325	80	74	1	0	0
		Virtex-5	325	75	74	1	0	0
		Virtex-4	250	66	74	1	0	0
		Spartan-6	200	78	74	1	0	0
	4096 x 16	Kintex-7[1]						
		Virtex-7[1]						
		Virtex-6	325	94	92	2	0	0
		Virtex-5	300	94	92	2	0	0
		Virtex-4	250	77	92	4	0	0
		Spartan-6	175	93	92	4	0	0

Table 21: **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
Synchronous FIFO (Distributed RAM)	64 x 16	Kintex-7(1)						
		Virtex-7(1)						
		Virtex-6	425	47	72	0	0	22
		Virtex-5	400	48	72	0	0	22
		Virtex-4	300	97	73	0	0	128
		Spartan-6	175	45	72	0	0	22
	512 x 16	Kintex-7(1)						
		Virtex-7(1)						
		Virtex-6	300	139	90	0	0	176
		Virtex-5	300	114	90	0	0	176
		Virtex-4	225	358	226	0	0	1024
		Spartan-6	150	132	90	0	0	176
Independent Clocks FIFO (Block RAM)	512 x 16	Kintex-7(1)						
		Virtex-7(1)						
		Virtex-6	325	110	153	1	0	0
		Virtex-5	300	103	153	1	0	0
		Virtex-4	300	97	153	1	0	0
		Spartan-6	200	101	153	1	0	0
	4096 x 16	Kintex-7(1)						
		Virtex-7(1)						
		Virtex-6	325	149	198	2	0	0
		Virtex-5	325	144	198	2	0	0
		Virtex-4	275	125	198	4	0	0
		Spartan-6	200	141	198	4	0	0
Independent Clocks FIFO (Distributed RAM	64 x 16	Kintex-7(1)						
		Virtex-7(1)						
		Virtex-6	450	65	124	0	0	22
		Virtex-5	400	66	124	0	0	22
		Virtex-4	325	116	128	0	0	128
		Spartan-6	200	63	124	0	0	22
	512 x 16	Kintex-7(1)						
		Virtex-7(1)						
		Virtex-6	350	148	169	0	0	176
		Virtex-5	275	142	169	0	0	176
		Virtex-4	225	388	301	0	0	1024
		Spartan-6	150	153	169	0	0	176

Table 21: **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
Shift Register FIFO[2]	64 x 16	Kintex-7[1]						
		Virtex-7[1]						
		Virtex-5	400	78	60	0	32	0
		Virtex-4	300	121	60	0	64	0
		Spartan-6	175	77	60	0	48	0
	512 x 16	Kintex-7[1]						
		Virtex-7[1]						
		Virtex-5	275	167	75	0	256	0
		Virtex-4	225	374	79	0	512	0
		Spartan-6	75	370	78	0	496	0

1. Benchmarking data for Virtex-7 and Kintex-7 will be available in future release.
2. Virtex-6 benchmarking data for the Shift Register FIFO will be available in a future release.

Table 22 provides results for FIFOs configured to use the Virtex-5 FPGA built-in FIFO. The benchmarks were performed using a Virtex-5 (XC5VLX50-FF324-1) and Virtex-6 (XC6VLX760-FF1760-1) FPGA.

Table 22: **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
Synchronous FIFO36 (basic)	512 x 72	Virtex-5	Standard	300	0	2	1
			FWFT	300	2	4	1
		Virtex-6	Standard	325	1	2	1
			FWFT	325	4	5	1
		Kintex-7[1]					
		Virtex-7[1]					
	16k x 8[2]	Virtex-5	Standard	275	10	6	4
			FWFT	275	13	10	4
		Kintex-7[1]					
		Virtex-7[1]					
Synchronous FIFO36 (with handshaking)	512 x 72	Virtex-5	Standard	300	2	6	1
			FWFT	300	5	6	1
		Virtex-6	Standard	325	4	6	1
			FWFT	325	8	8	1
		Kintex-7[1]					
		Virtex-7[1]					
	16k x 8[2]	Virtex-5	Standard	300	12	12	4
			FWFT	300	16	13	4
		Kintex-7[1]					
		Virtex-7[1]					

Table 22: **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 Clocks FIFO36 (basic)	512 x 72	Virtex-5	Standard	440	0	2	1
			FWFT	440	0	2	1
		Virtex-6	Standard	450	1	3	1
			FWFT	450	1	3	1
		Kintex-7(1)					
		Virtex-7(1)					
	16k x 8	Virtex-5	Standard	300	6	2	4
			FWFT	300	6	2	4
		Virtex-6	Standard	350	6	4	4
			FWFT	350	6	7	4
		Kintex-7(1)					
		Virtex-7(1)					
Independent Clocks FIFO36 (with handshaking)	512 x 72	Virtex-5	Standard	440	2	6	1
			FWFT	440	2	3	1
		Virtex-6	Standard	450	4	7	1
			FWFT	450	3	4	1
		Kintex-7(1)					
		Virtex-7(1)					
	16k x 8	Virtex-5	Standard	280	8	8	4
			FWFT	320	9	5	4
		Virtex-6	Standard	325	10	10	4
			FWFT	325	9	7	4
		Kintex-7(1)					
		Virtex-7(1)					

1. Benchmarking data for Virtex-7 and Kintex-7 will be available in future release.
2. Virtex-6 benchmarking data will be available in a future release.

Table 23 provides results for FIFOs configured to use the Virtex-4 built-in FIFO with patch. The benchmarks were performed using a Virtex-4 (XC4VLX15-FF668-10) FPGA.

Table 23: **Benchmarks: FIFO Configured with Virtex-4 FIFO16 Patch**

FIFO Type	Depth x Width	Clock Ratios	Performance (MHz)	LUTs	FFs	Distributed RAMs	FIFO16s
Built-in FIFO (basic)	512x36	WR_CLK ≥ RD_CLK	250	115	264	72	1
		RD_CLK > WR_CLK	225	118	269	72	1
Built-in FIFO (Handshaking)	512x36	WR_CLK ≥ RD_CLK	250	117	277	72	1
		RD_CLK > WR_CLK	225	121	282	72	1

AXI4 FIFO Resource Utilization and Performance

Table 24 provides the default configuration settings for the benchmarks data. Table 25 shows benchmark information for AXI4 and AXI4-Lite configurations. The benchmarks were obtained using Virtex-6 (XC6VLX760-FF1760-1) and Spartan-6 (XC6SLX150-FGG900-2) FPGAs.

Table 24: **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 25: **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	Virtex-6	300	164	601	5		92
		Virtex-7(1)						
		Kintex-7(1)						
		Spartan-6	175	172	541	6		92
	Independent Clock	Virtex-6	325	238	890	5		92
		Virtex-7(1)						
		Kintex-7(1)						
		Spartan-6	200	502	822	6		92
AXI4-Lite	Common Clock	Virtex-6	325	164	389	2		52
		Virtex-7(1)						
		Kintex-7(1)						
		Spartan-6	200	168	395	4		52
	Independent Clock	Virtex-6	300	239	680	2		52
		Virtex-7(1)						
		Kintex-7(1)						
		Spartan-6	200	317	680	4		52

1. Benchmarking data for Virtex-7 and Kintex-7 will be available in future release.

Table 26 provides benchmarking results for AXI4-Stream FIFO configurations. The benchmarks were obtained using Virtex-6 (XC6VLX760-FF1760-1) and Spartan-6 (XC6SLX150-FGG900-2) FPGAs.

Table 26: **AXI4-Stream Resource Utilization**

FIFO Type	FPGA Family	Depth x Width	Performance (MHz)	Resources				
				LUTs	FFs	Block RAM	Shift Register	Distributed RAM
Synchronous FIFO (Block RAM)	Virtex-6	512 x 16	325	46	67	1		
		4096 x 16	325	58	79	2		
	Virtex-7(1)							
	Kintex-7(1)							
	Spartan-6	512 x 16	200	48	67	1		
		4096 x 16	200	63	79	4		
Synchronous FIFO (Distributed RAM)	Virtex-6	512 x 16	375	82	87			176
		64 x 16	425	28	71			22
	Virtex-7(1)							
	Kintex-7(1)							
	Spartan-6	512 x 16	175	83	90			176
		64 x 16	200	28	71			22
Asynchronous FIFO (Block RAM)	Virtex-6	512 x 16	325	71	151	1		
		4096 x 16	325	102	190	2		
	Virtex-7(1)							
	Kintex-7(1)							
	Spartan-6	512 x 16	200	72	151	1		
		4096 x 16	200	105	190	4		
Asynchronous FIFO (Distributed RAM)	Virtex-6	512 x 16	375	111	167			176
		64 x 16	475	65	128			22
	Virtex-7(1)							
	Kintex-7(1)							
	Spartan-6	512 x 16	175	154	184			176
		64 x 16	250	42	128			22

1. Benchmarking data for Virtex-7 and Kintex-7 will be available in future release.

Supplemental Information

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

Compatibility with Older FIFO Cores

The FIFO Generator Migration Kit can be used to migrate from legacy FIFO cores (Asynchronous FIFO v6.x and Synchronous FIFO v5.x cores) and older versions of the FIFO Generator core to the latest version of the FIFO Generator core.

Use the `fifo_migrate.pl` script shipped with the FIFO Migration Kit zip file (`xapp992.zip`), and XAPP992, *FIFO Generator Migration Guide*, to migrate older FIFO cores to the most recent version. In addition, UG175, *LogiCORE IP FIFO Generator User Guide*, contains migration information with details about migrating to an AXI4 Interface FIFO Generator.

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. One allows you to specify what version you wish to upgrade to, and the other automatically upgrades the core to the latest version:

- Select Upgrade Version, and Regenerate (Under Current Project Settings): This upgrades an older FIFO Generator core version (v4.4, 5.1, 5.2, 5.3, 6.1, 6.2, or 7.2) to the intermediate version you select -- v5.1, 5.2, 5.3, 6.1, 6.2, 7.2 or 8.1.
- 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 and 8.1) to v8.2.

Native FIFO SIM Parameters

Table 27 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 27: **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)
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)

Table 27: **Native FIFO SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
10	C_HAS_ALMOST_EMPTY	Integer	• 0: Core does not have ALMOST_EMPTY flag • 1: Core has ALMOST_EMPTY flag
11	C_HAS_ALMOST_FULL	Integer	• 0: Core does not have ALMOST_FULL flag • 1: Core has ALMOST_ FULL flag
12	C_HAS_DATA_COUNT	Integer	• 0: Core does not have DATA_COUNT bus • 1: Core has DATA_COUNT bus
13	C_HAS_OVERFLOW	Integer	• 0: Core does not have OVERFLOW flag • 1: Core has OVERFLOW flag
14	C_HAS_RD_DATA_COUNT	Integer	• 0: Core does not have RD_DATA_COUNT bus • 1: Core has RD_DATA_COUNT bus
15	C_HAS_RST	Integer	• 0: Core does not have asynchronous reset (RST) • 1: Core has asynchronous reset (RST)
16	C_HAS_SRST	Integer	• 0: Core does not have synchronous reset (SRST) • 1: Core has synchronous reset (SRST)
17	C_HAS_UNDERFLOW	Integer	• 0: Core does not have UNDERFLOW flag • 1: Core has UNDERFLOW flag
18	C_HAS_VALID	Integer	• 0: Core does not have VALID flag • 1: Core has VALID flag
19	C_HAS_WR_ACK	Integer	• 0: Core does not have WR_ACK flag • 1: Core has WR_ACK flag
20	C_HAS_WR_DATA_COUNT	Integer	• 0: Core does not have WR_DATA_COUNT bus • 1: Core has WR_DATA_COUNT bus
21	C_IMPLEMENTATION_TYPE	Integer	• 0: Common-Clock Block RAM/Distributed RAM FIFO • 1: Common-Clock Shift RAM FIFO • 2: Independent Clocks Block RAM/Distributed RAM FIFO • 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

Table 27: **Native FIFO SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
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(1)
29	C_PROG_EMPTY_THRESH_NEGATE_VAL	Integer	PROG_EMPTY negate threshold(1)
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(1)
32	C_PROG_FULL_THRESH_NEGATE_VAL	Integer	PROG_FULL negate threshold(1)
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)

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

AXI4 FIFO SIM Parameters

Table 28 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 28: **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(1) • 1: Core has Write Channel(1)
4	C_HAS_AXI_RD_CHANNEL	Integer	• 0: Core does not have Read Channel(2) • 1: Core has Read Channel(2)
5	C_HAS_SLAVE_CE(3)	Integer	• 0: Core does not have Slave Interface Clock Enable • 1: Core has Slave Interface Clock Enable
6	C_HAS_MASTER_CE(3)	Integer	• 0: Core does not have Master Interface Clock Enable • 1: Core has Master Interface Clock Enable
7	C_ADD_NGC_CONSTRAINT(3)	Integer	• 0: Core does not add NGC constraint • 1: Core adds NGC constraint
8	C_USE_COMMON_UNDERFLOW(3)	Integer	• 0: Core does not have common UNDERFLOW flag • 1: Core has common UNDERFLOW flag
9	C_USE_COMMON_OVERFLOW(3)	Integer	• 0: Core does not have common OVERFLOW flag • 1: Core has common OVERFLOW flag
10	C_USE_DEFAULT_SETTINGS(3)	Integer	• 0: Core does not use default settings • 1: Core uses default settings
11	C_AXI_ID_WIDTH	Integer	ID Width
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
16	C_HAS_AXI_BUSER	Integer	• 0: Core does not have BUSER • 1: Core has BUSER

Table 28: **AXI4 SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
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
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

Table 28: **AXI4 SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
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 28: **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

Table 28: **AXI4 SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
56	C_APPLICATION_TYPE_WACH	Integer	Write Address Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
57	C_APPLICATION_TYPE_WDCH	Integer	Write Data Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
58	C_APPLICATION_TYPE_WRCH	Integer	Write Response Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
59	C_APPLICATION_TYPE_RACH	Integer	Read Address Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
60	C_APPLICATION_TYPE_RDCH	Integer	Read Data Channel Application type • 0: Data FIFO • 1: Packet FIFO(3) • 2: Low Latency Data FIFO
61	C_APPLICATION_TYPE_AXIS	Integer	AXI4 Stream Application type • 0: Data FIFO • 1: Packet FIFO(3) • 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

Table 28: **AXI4 SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
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
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).

Table 28: **AXI4 SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
76	C_DIN_WIDTH_WRCH	Integer	DIN Width of Write Response Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID).
77	C_DIN_WIDTH_RACH	Integer	DIN Width of Read Address Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID).
78	C_DIN_WIDTH_RDCH	Integer	DIN Width of Read Data Channel bus (1 - 1024). Width is the accumulation of all signal's width of this channel (except AWREADY and AWVALID).
79	C_DIN_WIDTH_AXIS	Integer	DIN Width of AXI4 Stream bus (1 - 1024) 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 28: **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 28: **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

Table 28: **AXI4 SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
110	C_PROG_FULL_THRESH_ASSERT_VAL_WACH	Integer	PROG_FULL assert threshold(4) for Write Address Channel
111	C_PROG_FULL_THRESH_ASSERT_VAL_WDCH	Integer	PROG_FULL assert threshold(4) for Write Data Channel
112	C_PROG_FULL_THRESH_ASSERT_VAL_WRCH	Integer	PROG_FULL assert threshold(4) for Write Response Channel
113	C_PROG_FULL_THRESH_ASSERT_VAL_RACH	Integer	PROG_FULL assert threshold(4) for Read Address Channel
114	C_PROG_FULL_THRESH_ASSERT_VAL_RDCH	Integer	PROG_FULL assert threshold(4) for Read Data Channel
115	C_PROG_FULL_THRESH_ASSERT_VAL_AXIS	Integer	PROG_FULL assert threshold(4) 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(4).

Table 28: **AXI4 SIM Parameters *(Cont'd)***

	SIM Parameter	Type	Description
123	C_PROG_EMPTY_THRESH_ASSERT_VAL_WDCH	Integer	PROG_EMPTY assert threshold for Write Data Channel(4).
124	C_PROG_EMPTY_THRESH_ASSERT_VAL_WRCH	Integer	PROG_EMPTY assert threshold for Write Response Channel(4).
125	C_PROG_EMPTY_THRESH_ASSERT_VAL_RACH	Integer	PROG_EMPTY assert threshold for Read Address Channel(4).
126	C_PROG_EMPTY_THRESH_ASSERT_VAL_RDCH	Integer	PROG_EMPTY assert threshold for Read Data Channel(4).
127	C_PROG_EMPTY_THRESH_ASSERT_VAL_AXIS	Integer	PROG_EMPTY assert threshold for AXI4 Stream(4).

1. Includes Write Address Channel, Write Data Channel and Write Response Channel.
2. Includes Read Address Channel, Read Data Channel.
3. Presently this feature is not supported.
4. See the FIFO Generator GUI for the allowable range of values.

Verification

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.

References

1. UG175, *FIFO Generator User Guide*
2. AXI4 AMBA® AXI Protocol Version: 2.0 Specification
3. AMBA® 4 AXI4-Stream Protocol Version: 1.0 Specification

Support

Xilinx provides technical support for this LogiCORE 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*.

Ordering Information

This Xilinx LogiCORE IP module is included at no additional charge with the Xilinx ISE® Design Suite software and is provided under the terms of the Xilinx End User License Agreement. The core is generated using the Xilinx ISE CORE Generator™ software, which is a standard component of the Xilinx ISE software.

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

Information about additional LogiCORE IP modules can be found on the Xilinx.com Intellectual Property page. Contact your local Xilinx sales representative for pricing and availability.

Revision History

The following table shows the revision history for this document:

Date	Version	Description of Revisions
4/23/04	1.0	Initial Xilinx release.
5/21/04	1.1	Support for Virtex-4 built-in FIFO implementation.
11/11/04	2.0	Updated for Xilinx software v6.3i.
04/28/05	2.1	The original product specification has been divided into two documents - a data sheet and a user guide. The document has also been updated to indicate core support of first-word fall-through feature and Xilinx software v7.1i.
8/31/05	2.2	Updated Xilinx v7.1i to SP3, removed references to first-word fall-through as new in this release. Updated basic FIFO benchmark value to reflect increased performance.
1/18/06	2.3	Minor updates for release v2.3, advanced ISE support to 8.1i.
7/13/06	3.0	Added support for Virtex-5, ISE to v8.2i, core version to 3.1
9/21/06	4.0	Core version updated to 3.2, support for Spartan-3A added to Facts table.
2/15/07	5.0	Updates to Xilinx tools 9.1i, core version 3.3.
4/02/07	5.5	Added support for Spartan-3A DSP devices, upgraded Cadence IUS version to 5.7
8/8/07	5.6	Updated to Xilinx tools v9.2i; Cadence IUS v5.8.
10/10/07	6.0	Updated for IP2 Jade Minor release.
3/24/08	7.0	Updated core to version 4.3; Xilinx tools to 10.1.
9/19/08	8.0	Updated core to version 4.4; Xilinx tools 10.1, SP3.
12/17/08	8.0.1	Early access documentation.
4/24/09	9.0	Updated core to version 5.1 and Xilinx tools to version 11.1.
6/24/09	10.0	Updated core to version 5.2 and Xilinx tools to version 11.2.
6/24/09	10.1	Updated Resource Utilization and Performance, page 32.
9/16/09	11.0	Updated core to version 5.3 and Xilinx tools to version 11.3.
4/19/10	12.0	Updated core to version 6.1 and Xilinx tools to version 12.1.
7/23/10	13.0	Updated core to version 6.2 and Xilinx tools to version 12.2.
9/21/10	14.0	Updated core to version 7.2 and Xilinx tools to version 12.3. Added AXI4 Interface FIFOs.
3/1/11	15.0	Updated core to version 8.1 and Xilinx tools to version 13.1. Added support for Kintex-7 and Virtex-7 FPGAs.
6/22/11	16.0	Updated core to v8.2 and Xilinx ISE Design Suite to v13.2. Added support for Zynq-7000 and Artix-7 FPGAs.

Notice of Disclaimer