XILINX® LogiCORE

Synchronous FIFO 5.0

DS256 May 21, 2004 **Product Specification**

Introduction

The Synchronous FIFO is a First-In-First-Out memory queue with control logic that manages the read and write pointers, generates status flags, and provides optional handshake signals for interfacing with the user logic.

About This Revision

Version 5.0 is the final release of the Synchronous FIFO core. For new designs, Xilinx suggests you use the FIFO Generator Logicore, which includes expanded support for applications requiring independent (asynchronous) or common (synchronous) read/write clock domains. See **FIFO Generator** for detailed information.

Features

- Drop-in module for Virtex™, Virtex-E, Virtex-II™, Virtex-II Pro™, Virtex-4™, Spartan™-II, Spartan-IIE, and Spartan-3 FPGAs
- Supports data widths up to 256 bits
- Supports memory depths of up to 65,536 locations
- Memory is implemented in either Distributed RAM or SelectRAM+
- Supports full and empty status flags
- Invalid read or write requests are rejected without affecting the FIFO state
- Four optional handshake signals (WR_ACK, WR_ERR, RD_ACK, RD_ERR) provide feedback (acknowledgment or rejection) in response to write and read requests in the prior clock cycle
- Optional count vector provides visibility into the number of data words currently in the FIFO
- Uses relationally placed macro (RPM) mapping and placement technology for maximum and predictable performance
- Incorporates Xilinx Smart-IP™ technology for utmost parameterization and optimum implementation
- To be used with version 6.2i or later of the Xilinx CORE Generator™ System

Functional Description

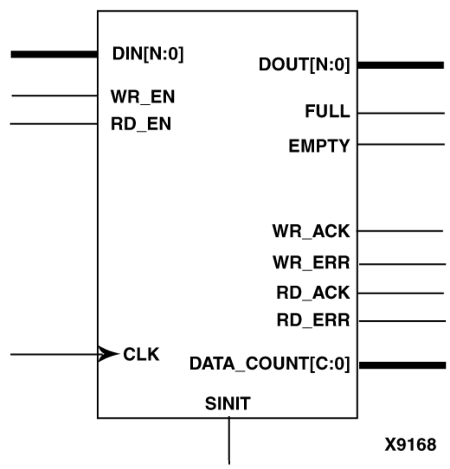

Figure 1: **Core Schematic Symbol**

The Synchronous FIFO has a single clock port for both data-read and data-write operations. Data presented at the module's data-input port (DIN) is written into the next available empty memory location on a rising clock edge when the write-enable input (WR_EN) is High. The memory full status output (FULL) indicates that no more empty locations remain in the module's internal memory.

Data can be read out of the FIFO via the module's data-output port (DOUT) in the order in which it was written by asserting read-enable (RD_EN) prior to a rising clock edge. The memory-empty status output (EMPTY) indicates that no more data resides in the module's internal memory.

The FIFO status cannot be corrupted by invalid requests. Requesting a read operation while the EMPTY flag is active will not cause any change in the current sate of the FIFO. Similarly, a write operation while the FULL flag is active will not cause any change in the current state of the FIFO. If enabled, the RD_ERR and WR_ERR handshake signals will indicate the rejection of these invalid requests.

In addition to the EMPTY and FULL flags, you can enable a count vector (DATA_COUNT) to provide a more granular measure of the FIFO state. The width of this vector is user programmable to provide easy generation of additional flags. For instance, a vector width of one creates a half-full flag; a width of two creates binary-encoded quadrant flags, and so on.

The Synchronous FIFO clock (CLK) is rising edge active for the FIFO core. However, it can be made falling edge active (relative to the clock source) by inserting an inverter between the clock source and the FIFO's clock input.

Behavior of Status Signals

The activation of the synchronous initialization input (SINIT) will reset the internal pointers and initialize the EMPTY output to 1 and FULL output to 0. This effectively empties the FIFO, discarding any data that may have been stored in the module but which had not been read-out.

Optional handshaking signals are provided to simplify the user control logic designed to interact with the FIFO. The WR_ACK and WR_ERR signals indicate acknowledgement or rejection of requested write operations (WR_EN active) respectively. Similarly, RD_ACK and RD_ERR signals indicate the acknowledgement or rejection of read operations (RD_EN active). Each of these control signals, shown in Figure 1, can be made Active High or Low from the GUI. Because an acknowledgement or error response depends on an active request (WR_EN or RD_EN), the acknowledge and error signals are not always the inverse of each other. If no operation is requested then both the acknowledge and the error signals will be inactive during the subsequent clock period. For an example of expected signal sequencing, refer to the timing diagram shown in Figure 2.

Pinout Description

Signal names are described in Table 1.

Table 1: **Core Signal Pinout**

Signal	Direction	Description
DIN[N:0]	Input	Data Input
WR_EN	Input	Write Enable (request)
RD_EN	Input	Read Enable (request)
CLK	Input	Clock for read and write operations (rising edge)
SINIT	Input	Synchronous initialization of all FIFO functions, flags, and pointers
FULL	Output	Full: No additional writes can be performed
DATA_ COUNT[C:0]	Output	Data Count: Vector (unsigned binary) of number of data words currently in FIFO
WR_ACK	Output	Write Acknowledge: Handshake signal indicates that data was written to the FIFO on the previous CLK edge while WR_EN was active

Table 1: **Core Signal Pinout *(Continued)***

Signal	Direction	Description
WR_ERR	Output	Write Error: Handshake signal indicates that no data word was written to the FIFO on the previous CLK edge while WR_EN was active
DOUT[N:0]	Output	Data Output: Synchronous to CLK
EMPTY	Output	Empty: No additional reads can be performed
RD_ACK	Output	Read Acknowledge: Handshake signal indicates that data was read from the FIFO and placed on the DOUT output pins on the previous CLK edge while RD_EN was active
RD_ERR	Output	Read Error: Handshake signal indicates that no data word was read from the FIFO on the previous CLK edge while RD_EN was active and subsequently data on DOUT output pins was not updated

CORE Generator Parameters

The main CORE Generator parameterization default values can be found in Table 2, and the parameter descriptions are as follows:

- **Component Name**: The component name is used as the base name of the output files generated for this module. Names must begin with a letter and must be composed from the following characters: a to z, 0 to 9 and "_".
- Memory Type: Distributed Memory radio button will implement the FIFO's memory using LUT-based SRL16 elements. Select the appropriate radio button for the type of memory desired. Block Memory implements the FIFO's memory using SelectRAM+.
- **Data Width**: The width of the input data bus (also the width of the output data bus). The valid range is 1-256.
- **FIFO Depth**: Select the available depth from the pull-down list. Depths are (2^N). *N* can be any integer from 4 to 16, with additional restrictions based on Memory Type and Data Width.
- **Data Count**: When selected, the corresponding data count width dialog box becomes active.
- **Data Count Width:** Valid count widths are any integer from 1 to N+1 (where 2^N = FIFO Depth). If an integer greater than N+1 is entered, the core generation will be inhibited until it is corrected.

  For example, for a FIFO Depth of 16, the internal counter will have a width of 5-bits, or INT_COUNT[4:0]. There are two cases to be considered:

  - Case 1: The selected width of DATA_COUNT is equal to the internal counter (C=5). In this case the output vector generated is equal to the internal counter: DATA_COUNT[4:0] = INT_COUNT[4:0].
  - Case 2: The selected width of the DATA_COUNT is less than the internal counter (C<5). In this case, the output vector generated is: DATA_COUNT[(C-1):0] = INT_COUNT[(C-2):($\log_2$(FIFO Depth)-C)
    - For C=4, DATA_COUNT[3:0]=INT_COUNT[3:0]
    - For C=3, DATA_COUNT[2:0]=INT_COUNT[3:1]
    - For C=2, DATA_COUNT[1:0]=INT_COUNT[3:2]

· For C=1, DATA_COUNT[0]=INT_COUNT[3]

The optional handshaking control signals (acknowledge and/or error) can be enabled via the Handshaking Options button.

- **Read Acknowledge Flag**: Asserted active on the clock cycle after a successful read has occurred. This signal can be made active high or low through the GUI.
- **Read Error Flag**: Asserted active on the clock cycle after a read from the FIFO was attempted, but not successful. This signal can be made active high or low through the GUI.
- **Write Acknowledge Flag**: Asserted active on the clock cycle after a successful write has occurred. This signal can be made active high or low through the GUI.
- **Write Error Flag**: Asserted active on the clock cycle after a write to the FIFO was attempted, but not successful. This signal can be made active high or low through the GUI.

Parameter Values in XCO File

Names of XCO file parameters and their parameter values are identical to the names and values shown in the GUI, except that underscore characters (_) are used instead of spaces. The text in an XCO file is case insensitive.

The format for the XCO file should be as follows:

CSET <parameters> = <desired_options>

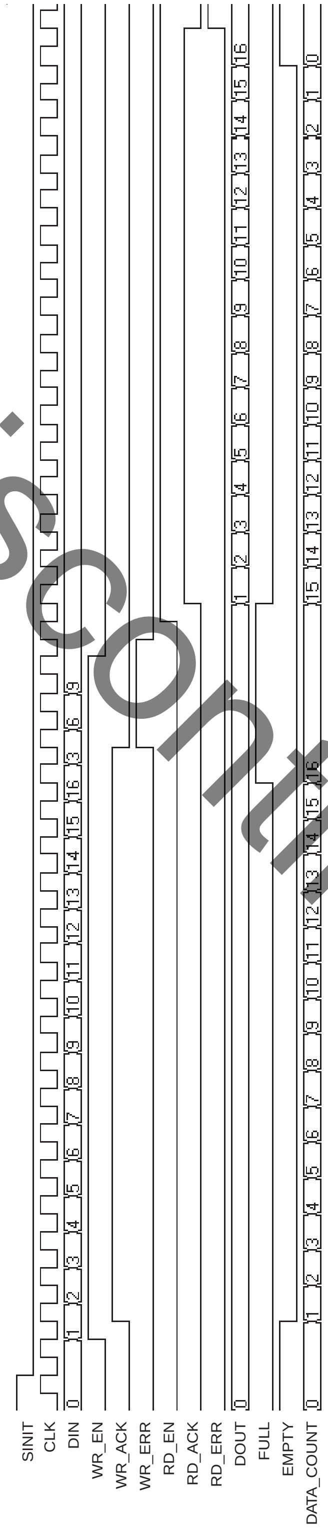

Figure 2: **Write/Read Waveform for FIFO Depth of 16**

For example:

CSET component_name = my_fifo_name

Table 2 shows the XCO file parameters and values, as well as summarizes the GUI defaults.

Table 2: **XCO File Values and Default GUI Settings**

Parameter	XCO File Values	Default GUI Setting
component_name	ASCII text starting with a letter and based upon the following character set: a..z, 0..9, and _	blank
memory_type	One of the following keywords: Distributed Memory, Block Memory	Distributed Memory
data_width	Integer in the range 1 to 256	1
fifo_depth	Integer in the range 15 to 65536. Must be equal to (2^N; N = 4 to 16)	16
write_acknowledge_flag	One of the following keywords: true, false	false
write_acknowledge_sense	One of the following keywords: active_high, active_low	active_high
write_error_flag	One of the following keywords: true, false	false
write_error_sense	One of the following keywords: active_high, active_low	active_high
read_acknowledge_flag	One of the following keywords: true, false	false
read_acknowledge_sense	One of the following keywords: active_high, active_low	active_high
read_error_flag	One of the following keywords: true, false	false
read_error_sense	One of the following keywords: active_high, active_low	active_high
data_count	One of the following keywords: true, false	false
data_count_width	Integer in the range 1 to N+1, where N is determined by the fifo_depth	1

Core Resource Utilization

The resource requirements of the synchronous FIFO are highly dependent on the memory size and the presence of optional ports. For some memory depths extra logic is required to decode the address and multiplex the output from various primitives.

Table 3 shows the maximum number of slices for various depth and width combinations for the Virtex family. Similarly, Table 4 and Table 5 show maximum resource utilizations for the Virtex-II and Virtex-II Pro families.

Table 3: **Virtex Distributed RAM Resource Utilizations**

Depth	Min Width	Max Width	Max Size (bits)	Max Number of Slices
16	1	256	4096	270
32	1	256	8192	655

Table 3: **Virtex Distributed RAM Resource Utilizations**

Depth	Min Width	Max Width	Max Size (bits)	Max Number of Slices
64	1	256	16384	1680
128	1	144	18432	1961
256	1	72	18432	2000

Table 4: **Virtex-II Distributed RAM Resource Utilizations**

Depth	Min Width	Max Width	Max Size (bits)	Max Number of Slices
16	1	256	4096	271
32	1	256	8192	400
64	1	256	16384	913
128	1	256	32768	1683
256	1	256	65536	3221

Table 5: **Virtex-II Pro Distributed RAM Resource Utilizations**

Depth	Min Width	Max Width	Max Size (bits)	Max Number of Slices
16	1	256	4096	396
32	1	256	8192	397
64	1	256	16384	783
128	1	256	32768	1552
256	1	256	65536	3218

Performance Benchmarking

To properly constrain the Synchronous FIFO, place an appropriate period constraint on the FIFO clock (CLK). The Synchronous FIFO benchmark results are shown in Tables 6 and 7 for Distributed Memory implementation, and Tables 9 and 10 SelectRam+ implementation.

Table 6: **Virtex-E Synchronous FIFO Performance Benchmarking (Distributed Memory implementation)**

Data Width	8	16	32	64	128
Depth=64	159 MHz (6.3 ns)	137 MHz (7.3 ns)	130 MHz (7.7 ns)	125 MHz (8.0 ns)	110 MHz (9.0 ns)

Note: These benchmark designs contain only one FIFO without any additional logic, so benchmark numbers approach the performance ceiling rather than representing performance under typical conditions.

Table 7: **Virtex-II Synchronous FIFO Performance Benchmarking (Distributed Memory implementation)**

Data Width	8	16	32	64	128
Depth=64	200 MHz (5.0 ns)	185 MHz (5.4 ns)	185 MHz (5.4 ns)	169 MHz (5.9 ns)	156 MHz (6.4 ns)
Depth=128	181 MHz (5.5 ns)	166 MHz (6.0 ns)	166 MHz (6.0 ns)	156 MHz (6.4 ns)	145 MHz (6.9 ns)

Notes:

1. These benchmark designs contain only one FIFO without any additional logic, so benchmark numbers approach the performance ceiling rather than representing performance under typical conditions.
2. These results were obtained using preview speed files.

Table 8: **Virtex-E Synchronous FIFO Performance Benchmarking (Distributed Memory Implementation)**

Data Width	8	16	32	64	128
Depth = 64	339 MHz (2.9ns)	339 MHz (2.9ns)	318 MHz (3.1 ns)	204 MHz (4.9 ns)	167 MHz (6.0 ns)
Depth = 128	290 MHz (3.5ns)	245 MHz (4.1ns)	234 MHz (4.3 ns)	220 MHz (4.6 ns)	152 MHz (6.6 ns)

Notes:

1. These benchmark designs contain only one FIFO without any additional logic, so benchmark numbers approach the performance ceiling rather than representing performance under typical conditions.
2. These results were obtained using preview speed files.

Table 9: **Virtex-E Synchronous FIFO Performance Benchmarking (SelectRAM+ implementation)**

Part	FIFO Implementation		
V50E	256 x 16	512 x 8	1024 x 8
-7	192 MHz - (5.2 ns)	188 MHz (5.3 ns)	182 MHz (5.5 ns)
-8	204 MHz (4.9 ns)	200 MHz (5 ns)	200 MHz (5 ns)

Note: These benchmark designs contain only one FIFO without any additional logic, so benchmark numbers approach the performance ceiling rather than representing performance under typical conditions.

Table 10: **Virtex-II Synchronous FIFO Performance Benchmarking (SelectRAM+ implementation)**

Part	FIFO Implementation		
2V250	256 x 16	512 x 8	1024 x 8
-5	250 MHz - (4 ns)	250 MHz (4 ns)	250 MHz (4 ns)

Note: These benchmark designs contain only one FIFO without any additional logic, so benchmark numbers approach the performance ceiling rather than representing performance under typical conditions.

Table 11: **Virtex-II Pro Synchronous FIFO Performance Benchmarking (SelectRAM+ implementation)**

Part	FIFO Implementation		
2VP20	256 x 16	512 x 8	1024 x 8
-7	360 MHz (2.8ns)	330 MHz (3.0ns)	348 MHz (2.9ns)

Note: These benchmark designs contain only one FIFO without any additional logic, so benchmark numbers approach the performance ceiling rather than representing performance under typical conditions.

Ordering Information

This core may be downloaded from the Xilinx IP Center for use with the Xilinx CORE Generator System v6.2i and later. The Xilinx CORE Generator System tool is bundled with all Alliance Series Software packages, at no additional charge.

To order Xilinx software, please visit the Xilinx Silicon Xpresso Cafe or contact your local Xilinx sales representative.

Information on additional Xilinx LogiCORE modules is available on the Xilinx IP Center.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
03/28/03	1.0	Revision History added to document.
05/21/04	1.1	Updated for Virtex-4 and Xilinx Core Generator v6.2i.