Asynchronous FIFO v6.1

Introduction

The Asynchronous FIFO is a First-In-First-Out memory queue with control logic that performs management of the read and write pointers, generation of status flags, and optional handshake signals for interfacing with the user logic.

About This Revision

Version 6.1 is the final release of the Asynchronous 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 65,535 locations
- Memory may be implemented in either SelectRAM+ or Distributed RAM
- Fully synchronous and independent clock domains for the read and write ports
- Supports FULL and EMPTY status flags
- Optional ALMOST_FULL and ALMOST_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(s) provide visibility into number of data words currently in the FIFO, synchronized to either clock domain
- 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 v6.3i or later of the Xilinx CORE Generator™ system

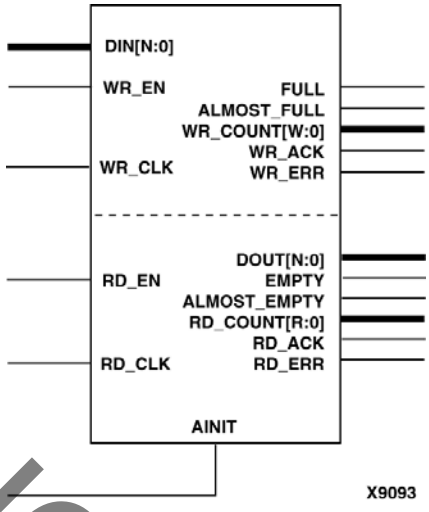

Figure 1: **Core Schematic Symbol**

Functional Description

The Asynchronous FIFO is a First-In-First-Out memory queue with control logic that performs management of the read and write pointers, generation of status flags, and optional handshake signals for interfacing with the user logic. The individual read and write ports are fully synchronous (all operations qualified by a rising clock edge), but this FIFO does not require the read and write clocks to be synchronized to each other.

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, requesting 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, ALMOST_EMPTY, FULL, and ALMOST_FULL flags, a count vector can be enabled to provide a more granular measure of the FIFO state. For the write domain the vector is WR_COUNT[W:0], and for the read domain it is RD_COUNT[R:0]. The width of these vectors are 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. In keeping with the fully synchronous interface, the count vector can be synchronized to either the read or the write clock domain, or two independent counts can be enabled, one for each clock domain.

Synchronization and Timing Issues

As previously stated, the read and write ports can be operated on independent asynchronous clock domains. However, the user interface logic still must address synchronization issues. The core schematic symbol, shown in Figure 1, divides the signals according to their appropriate clock domains—write on the top half, read on the bottom. All signals, either input or output, are synchronous to one of the two clocks, with the exception of AINIT, which performs an asynchronous reset of the entire FIFO. On the write side, the control (WR_EN) and data input (DIN) are sampled by the rising edge of WR_CLK and should be synchronous to the WR_CLK. For the read side the read control (RD_EN) should be synchronous to the RD_CLK and the output data (DOUT) is valid after the subsequent rising edge of RD_CLK. All status outputs are synchronous to their respective clock domains and should be sampled only by logic operating on a synchronous clock. FIFO performance can be effectively constrained and analyzed by placing the desired clock PERIOD constraints on both the WR_CLK and RD_CLK source signals.

WR_CLK and RD_CLK are always rising edge active for the FIFO core. They can be made falling edge active (relative to the clock source) by inserting an inverter between the clock source and the FIFO's clock inputs.

Behavior of Status Signals

The activation of the AINIT, asynchronous initialization (reset), will force all four FIFO flags to the active (high) state. On the first WR_CLK after the release of AINIT the FULL and ALMOST_FULL flags will become inactive, indicating that the FIFO is now ready to accept write operations. EMPTY and ALMOST_EMPTY are deactivated on a rising edge of the RD_CLK following the first and second writes respectively. The ALMOST_EMPTY flag is active when the FIFO has one data word or is EMPTY. The ALMOST_FULL flag is active when the FIFO has only one available memory location or is FULL.

Optional handshake signals are provided to simplify user control logic interacting with the FIFO. The WR_ACK and WR_ERR signals indicate acknowledgment or rejection of requested write operations. Similarly, RD_ACK and RD_ERR signals indicate the acknowledgment or rejection of read operations. Each of these control signals can be made active high or low from the GUI. Note that all of these handshake signals are synchronous to their respective clock domains and indicate the acknowledgment or rejection of requests during the prior rising clock edge. Because an acknowledgment or error response depends on an active request (WR_EN or RD_EN), the ACK and ERR signals are not always the inverse of each other. If no operation is requested, both the acknowledgment and the error signal will be inactive during the subsequent clock period. For an example of expected signal sequencing, refer to the timing diagram shown in Figure 2.

The optional data count outputs (WR_COUNT and RD_COUNT) support the generation of user programmable flags. In the simplest case, selecting a width of one for a data count produces a half-full flag. Like all other FIFO outputs, the counts are synchronized to their respective clock domains and should be sampled only by logic operating on the same (or a synchronous) clock. The data count vectors have clock latency and should not be used as substitutes for the FULL, ALMOST_FULL, EMPTY, or ALMOST _EMPTY flags. The clock latency of the counts in their respective clock domains is one cycle. For example, the WR_COUNT does not reflect the impact of a write operation performed as a result of a request (WR_EN active) during the prior clock cycle. WR_COUNT and RD_COUNT values are not guaranteed to produce a precise representation of the FIFO contents at a particular point in time. These values should be used as a gauge to determine the FIFO status (*see answer record 14518 for more information*). The latency for operations in the opposing clock domain can be up to three clock cycles. For example, in the case of the WR_COUNT, read operations that may have been performed during the immediate three prior RD_CLK periods will not be reflected in the data count vector. This latency results from a design trade-off between clock frequency and count accuracy and is not as limiting as it may at first appear.

Consider the following scenario of a FIFO configured depth of 63 and a write count of two bits (WR_COUNT[1:0]).

Note that for this example:

Write_COUNT[1:0]=00: Indicates that the FIFO is less than 1/4 full and corresponds to the occupancy range of (0:16). The upper bound is 16 and not 15 due to the write latency of 1 clock cycle.

Write_COUNT[1:0]=01: Indicates that the FIFO is between 1/4 full and 1/2 full and corresponds to the occupancy range of (13:32). The lower bound is 13 and not 16 due to the read latency of 3 clock cycles.

Write_COUNT[1:0]=10: Indicates that the FIFO is between 1/2 full and 3/4 full and corresponds to the occupancy range of (29-48).

Write_COUNT[1:0]=11: Indicates that the FIFO is between 3/4 full and full and corresponds to the occupancy range of (45-63).

If the control logic needs to throttle back write operations based on the FIFO occupancy, it can use the write count vector in the following way. As shown above, WR_COUNT[1:0] equal to 11 corresponds to an occupancy greater than 45. As long as the user's WR_COUNT is not 11, no more than 48 data words (47 plus one for the write operation clock latency) are present in the FIFO. The user's control logic is assured that at least 15 (63-48) additional memory locations are available in the queue. There could be up to three more locations because of recent read operations, but this only increases the available memory locations. In this scenario, at least 14 additional writes can be performed without causing the FULL flag to transition to true.

Alternatively the control logic might want to wait for a fixed FIFO occupancy prior to performing a burst read operation. In this case, read operations are suspended before the appropriate count is reached. So for the same FIFO configuration, when the RD_COUNT transitions to 11, there are at least 47 data words in the FIFO. The write operation latency means that there can be as many as 51 words in the FIFO, but the user's read logic is guaranteed that at least 47 words are present. Read operations can be initiated with the assurance that at least 47 assured reads can continue as long as the EMPTY flag is inactive, indicating that data is available.

Pinout

Core signal names are shown in Figure 1 and described in Table 1.

Table 1: **Core Signal Pinout**

Name	Direction	Description
DIN[N:0]	Input	Data Input
WR_EN	Input	Write Enable (request)
WR_CLK	Input	Clock for write domain operations (rising edge)
RD_EN	Input	Read Enable (request)
RD_CLK	Input	Clock for read domain operations (rising edge)
AINIT	Input	Asynchronous reset of all FIFO functions, flags, and pointers
FULL	Output	Full: no additional writes can be performed, synchronous to WR_CLK
ALMOST_ FULL	Output	Almost Full: Only one additional write can be performed before FIFO is FULL, synchronous to WR_CLK
WR_ COUNT[W:0]	Output	Write Count: Count vector (unsigned binary) representing the number of data words currently in FIFO, synchronized to WR_CLK. If 2^(W+1) < [FIFO depth +1], the least significant bits of count are truncated. (W=0 produces a half-full flag)
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
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. This is an indication that a write operation was attempted, but the FIFO was Full.
DOUT[N:0]	Output	Data Output: Synchronous to RD_CLK
EMPTY	Output	Empty: No additional reads can be performed, synchronous to RD_CLK
ALMOST_ EMPTY	Output	Almost Empty: Only one additional read can be performed before FIFO is EMPTY, synchronous to RD_CLK.

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

Name	Direction	Description
RD_ COUNT [R:0]	Output	Read Count: Count vector (unsigned binary) representing the number of data word currently in FIFO, synchronized to RD_CLK. If (2^R+1)<(FIFO depth+1), the least significant bits of count are truncated (R=0, produces a half-full flag)
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. This is an indication that a read operation was attempted, but the FIFO was Empty.

CORE Generator Parameters

The main Core Generator parameterization 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**: Select the appropriate radio button for the type of memory desired. Block Memory implements the FIFO's memory using SelectRAM+. Selecting the Distributed Memory radio button will implement the FIFO memory using LUT-based dual-port memory.
- **Input Data Width**: Enter 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. As one memory location has been sacrificed in the interest of optimizing FIFO performance available depths are (2^N –1). N can be any integer from 4 to 16, with additional restrictions based on the Data Width.
- **Data Count**: Two Data Counts, one for each clock domain, can be enabled by selecting the appropriate radio button. Once selected, the corresponding count width dialog box becomes active. Valid count widths are any integer from 1 to N (where 2^N = (FIFO Depth + 1). If an integer greater than N is entered, it will turn red and the core generation will be inhibited until this error is corrected.
- **Create RPM**: When this box is checked, the Asynchronous FIFO will be generated using Relationally Placed Macros (RPMs). This means that the module will be generated with relative location attributes attached. The FIFO will be produced with two (or three, if distributed memory was selected) individual RPMs. A single RPM is not produced to allow the FIFO to support varying footprints.
- **Almost Full Flag**: Generates an Almost Full signal, indicating that one additional write can be performed before the FIFO is full.
- **Almost Empty Flag**: Generates an Almost Empty signal, indicating that one additional read can be performed before the FIFO is empty.

The optional handshaking control signals (acknowledge and/or error) can be enabled via the Handshaking Options button. When selected, a popup dialog box will appear.

- **Read Acknowledge Flag**: Asserted active on the clock cycle after a successful read has occurred. This signal, when selected, 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, when selected, 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, when selected, 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, when selected, 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 <parameter> = <desired_option>

For example:

CSET component_name = my_fifo_name

Figure 2 shows the waveform output of the VHDL behavioral model for a FIFO with depth of 15. Initially, the FULL and ALMOST_FULL output flags are high, indicating that the FIFO is in a reset state and the user should not write to the FIFO. When WR_EN is set to 1, the first write operation fails and returns a WR_ERR because on that rising clock edge the FIFO is reporting FULL and can not be written to.

The WR_COUNT and RD_COUNT outputs in Figures 2 and 3 report an estimated value of the number of words in the FIFO relative to their respective clock domains. These outputs are expressed as the fraction of the FIFO that is full, and can be used to generate user-threshold flags. Due to delays in the core, these outputs can never be relied upon as an exact measure of the number of words in the FIFO.

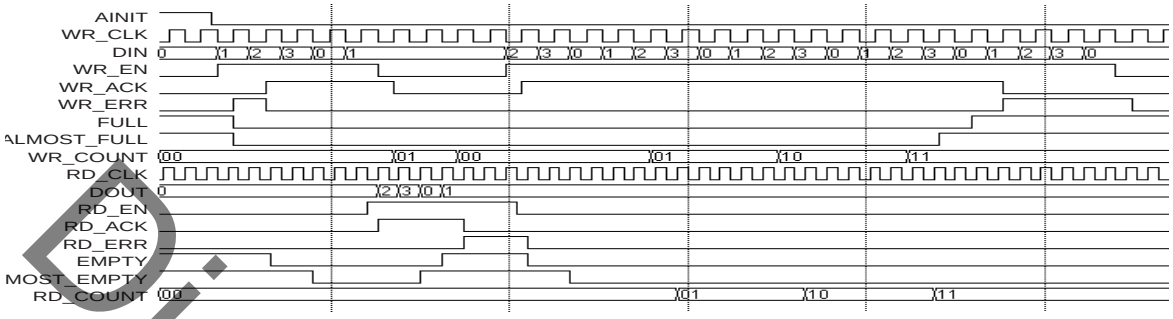

Figure 2: **Timing Diagram of Read and Write Operations for FIFO VHDL Behavioral Model**

Figure 3 shows the waveform output of the Verilog behavioral model for a FIFO with a depth of 15. Unlike the VHDL model, it is a purely functional model. The actual waveform of the Verilog model in Figure 3 looks quite different from the VHDL waveform in Figure 2, but they are functionally equivalent from the point of view of either the read or write interface of the FIFO. As there is no delay in the Verilog model, the effects of full, empty, read, or write can occur instantaneously.

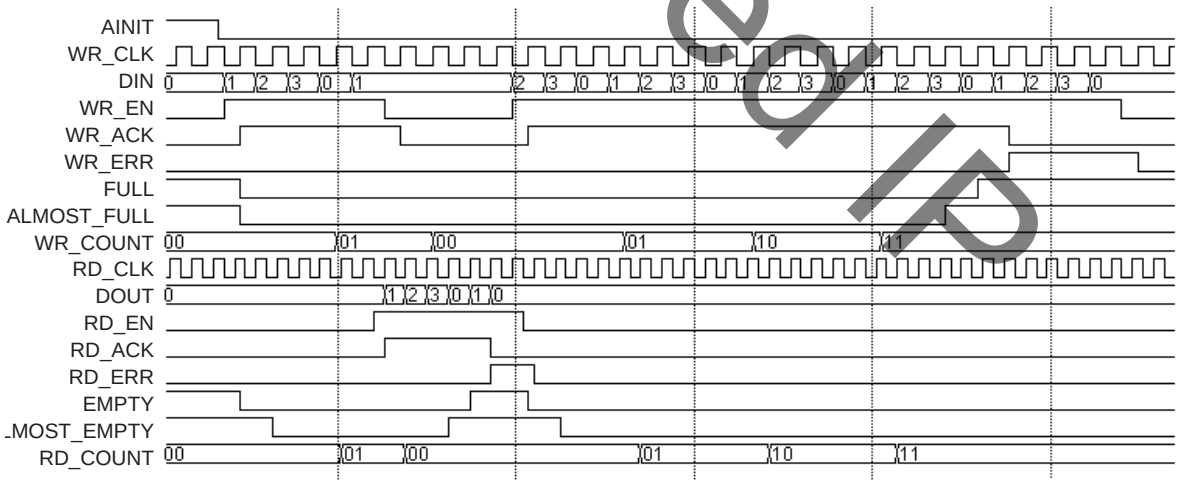

Figure 3: **Timing Diagram of Read and Write Operations for FIFO Verilog Behavioral Model**

Table 2 provides the XCO file parameters and values and summarizes the GUI defaults.

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

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	Keyword block, anything else generates LUT RAM	block
input_data_width	Integer in the range 1 to 256	16
fifo_depth	Integer in the range 15 to 65,535. Must be equal to (2^N-1;, N = 4 to 16)	63
almost_full_flag	One of the following keywords: true, false	false
almost_empty_flag	One of the following keywords: true, false	false
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
write_count	One of the following keywords: true, false	false
write_count_width	Integer in the range 1 to N, where N is determined by the fifo_depth	2
read_count	One of the following keywords: true, false	false
read_count_width	Integer in the range 1 to N, where N is determined by the fifo_depth	2
create_rpm	One of the following keywords: true, false	false

Core Resource Utilization

The resource requirements of the asynchronous FIFO are highly dependent on the memory size, memory type, and the presence of optional ports. Resource utilization can be estimated by addition of the requirements for the FIFO's memory and control logic.

Table 3 lists the number of SelectRAM+ blocks required for the Virtex family to implement various width and depth combinations when using the block memory implementation. Similarly, Table 4 lists the number of SelectRAM+ blocks required for the Virtex-II family.

Table 5 shows the approximate number of slices per bit for a distributed ram-based FIFO for the Virtex family. Multiply this number by the data width to determine the approximate slice count for the memory. Note that resource utilization for distributed ram-based FIFO and control logic for Virtex-II will be similar to that show for Virtex.

Control logic resource utilization is a function of the required addressing width N (N = log2(fifo_depth+1) and the optional features enabled. The slice count calculation varies slightly, depending on N being odd or even. For example, for the Virtex family:

For N even, slice count is:

- (N * 3.5) + 6 (Base)
- +(N * 0.5) + 2 (per almost flag)
- +(N * 2.0) + 1 (per data count)
- +(1) (for write handshaking)
- +(1) (for read handshaking))

For N odd, slice count is:

- (N * 3.5) + 7.5 (Base)
- +(N * 0.5) + 1.5 (per almost flag)
- +(N * 2.0) + 2.0 (per data count)
- +(1) (for write handshaking)
- +(1) (for read handshaking))

Example: a 1023x8 SelectRAM+ based FIFO with all of the features enabled requires 2 blockRAMs (see Table 3) and an additional 99 slices (N=10) for the control logic.

41+7+7+21+21+1+1 = 99 slices (N=10)

Table 3: **Virtex and Virtex-E Select Ram+ Usage**

Data Width	FIFO Depth												
	15	31	63	127	255	511	1023	2047	4095	8191	16383	32767	65535
1	1	1	1	1	1	1	1	1	1	2	4	8	16
2	1	1	1	1	1	1	1	1	2	4	8	16	32
3	1	1	1	1	1	1	1	2	3	6	12	24	48
4	1	1	1	1	1	1	1	2	4	8	16	32	64
5	1	1	1	1	1	1	2	3	5	10	20	40	80
6	1	1	1	1	1	1	2	3	6	12	24	48	96
7	1	1	1	1	1	1	2	4	7	14	28	56	112
8	1	1	1	1	1	1	2	4	8	16	32	64	128
9-12	1	1	1	1	1	2	3	5/6	9/12	18/24	36/48	72/96	144/192
13-16	1	1	1	1	1	2	4	7/8	13/16	26/32	52/64	104/128	208/256
17-32	2	2	2	2	2	3/4	5/8	9/16	17/32	34/64	68/128	136/256	N/S

Table 3: **Virtex and Virtex-E Select Ram+ Usage *(Continued)***

Data Width	FIFO Depth												
	15	31	63	127	255	511	1023	2047	4095	8191	16383	32767	65535
33-40	3	3	3	3	3	5	9/10	17/20	33/40	66/80	132/160	N/S	N/S
41-48	3	3	3	3	3	6	11/12	21/24	41/48	82/96	164/192	N/S	N/S
49-64	4	4	4	4	4	7/8	13/16	25/32	49/64	98/128	196/256	N/S	N/S
65-128	5/8	5/8	5/8	5/8	5/8	9/16	17/32	33/64	65/128	130/256	N/S	N/S	N/S
129-192	9/12	9/12	9/12	9/12	9/12	17/24	33/48	65/96	129/192	N/S	N/S	N/S	N/S
193-256	13/16	13/16	13/16	13/16	13/16	25/32	49/64	97/128	193/256	N/S	N/S	N/S	N/S

Table 4: **Virtex-II Select Ram+ Usage**

Data Width	FIFO Depth												
	15	31	63	127	255	511	1023	2047	4095	8191	16383	32767	65535
1	1	1	1	1	1	1	1	1	1	1	1	2	4
2	1	1	1	1	1	1	1	1	1	1	2	4	8
3	1	1	1	1	1	1	1	1	1	2	3	6	11
4	1	1	1	1	1	1	1	1	1	2	4	8	15
5	1	1	1	1	1	1	1	1	2	3	5	9	18
6	1	1	1	1	1	1	1	1	2	3	6	11	22
7	1	1	1	1	1	1	1	1	2	4	7	13	25
8	1	1	1	1	1	1	1	1	2	4	8	15	29
9-12	1	1	1	1	1	1	1	1/2	3	5/6	9/12	16/22	32/43
13-16	1	1	1	1	1	1	1	2	4	7/8	13/16	24/29	47/57
17-32	1	1	1	1	1	1	1/2	2/4	5/8	9/16	17/32	31/57	N/S
33-40	1/2	1/2	1/2	1/2	1/2	1/2	2/3	4/5	9/10	17/20	33/40	N/S	N/S
41-48	2	2	2	2	2	2	3	5/6	11/12	21/24	41/48	N/S	N/S
49-64	2	2	2	2	2	2	3/4	6/8	13/16	25/32	49/64	N/S	N/S
65-128	2/4	2/4	2/4	2/4	2/4	2/4	4/8	8/15	17/32	33/64	65/128	N/S	N/S
129-192	4/6	4/6	4/6	4/6	4/6	4/6	8/11	15/22	33/48	65/96	129/192	N/S	N/S
193-256	6/8	6/8	6/8	6/8	6/8	6/8	11/15	22/29	49/64	97/128	193/256	N/S	N/S

Performance Benchmarking

To properly constrain the Asynchronous FIFO, place appropriate period constraints on the read (RD_CLK) and write (WR_CLK) clocks. The Asynchronous FIFO benchmark results are shown in Table 6 for Virtex, Table 7 for Virtex-E, and Table 8 for Virtex-II.

Table 5: **Virtex Distributed RAM Resource Utilization**

FIFO_depth	Resources Used	Slice Estimate
15	2/0/0/1	2
31	6/0/0/1	3
63	12/2/0/1	6
127	24/4/2/1	11
255	49/8/4/1	22

Note: Resource Utilization (LUT / MUXF5 / MUXF6 / FD) for Distributed RAM FIFO Memory Only (per bit, multiply by data width)

Table 6: **Virtex Asynchronous FIFO Performance Benchmarking (SelectRAM+ Implementation)**

PART	FIFO Implementation		
V50PQ240	255x16 no options	255x16 all options	1023X8 all options
-4	114 MHz – (8.8 ns)	113 MHz – (8.8 ns)	113 MHz – (8.8 ns)
-5	141 MHz – (7.1 ns)	125 MHz – (8.0 ns)	133 MHz - (7.5 ns)
-6	156 MHz – (6.4 ns)	151 MHz – (6.6 ns)	147 MHz - (6.8 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 user conditions. Highest frequencies will be obtained by using the create RPM option or through custom floor planning.
2. Over constraining the FIFO (applying overly aggressive timing constraints) will degrade the achievable performance. For example, applying a 6.0ns constraint to the 255x16 no options implementation (-6) will result in a placed and routed implementation that is considerably slower than the 6.4ns shown in the table.

Table 7: **Virtex-E Asynchronous FIFO Performance Benchmarking (SelectRAM+ Implementation)**

PART	FIFO Implementation		
V50EPQ240	255x16 no options	255x16 all options	1023X8 all options
-6	178 MHz – (5.6 ns)	174 MHz – (5.7 ns)	172 MHz – (5.8 ns)
-7	192 MHz – (5.2 ns)	188 MHz – (5.3 ns)	188 MHz - (5.3 ns)
-8	196 MHz – (5.1 ns)	192 MHz – (5.2 ns)	196 MHz - (5.1 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 user conditions. Highest frequencies will be obtained by using the create RPM option or through custom floor planning.
2. Over constraining the FIFO (applying overly aggressive timing constraints) will degrade the achievable performance.

Table 8: **Virtex-II Asynchronous FIFO Performance Benchmarking (SelectRAM+ Implementation)**

PART	FIFO Implementation		
2V250FG256	255x16 no options	255x16 all options	1023X8 all options
-5	233 MHz – (4.3 ns)	217 MHz – (4.3 ns)	213 MHz - (4.7 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 user conditions. Highest frequencies will be obtained by using the create RPM option or through custom floor planning.
2. Over constraining the FIFO (applying overly aggressive timing constraints) will degrade the achievable performance.
3. Speed files used are preview.

Ordering Information

This core may be downloaded from the Xilinx IP Center for use with the Xilinx CORE Generator System v6.3i and later. The Xilinx CORE Generator system 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 about additional Xilinx LogiCORE modules is available on the Xilinx IP Center.

Revision History

Date	Version	Revision
03/28/03	1.0	Revision History added to document.
5/21/04	1.1	Added support for Virtex-4 and v6.2i of the Xilinx CORE Generator system.
11/11/04	1.2	Updated document to indicate support for v6.3i of the Xilinx CORE Generator system.