Asynchronous FIFO V1.0.3

December 17, 1999 Product Specification

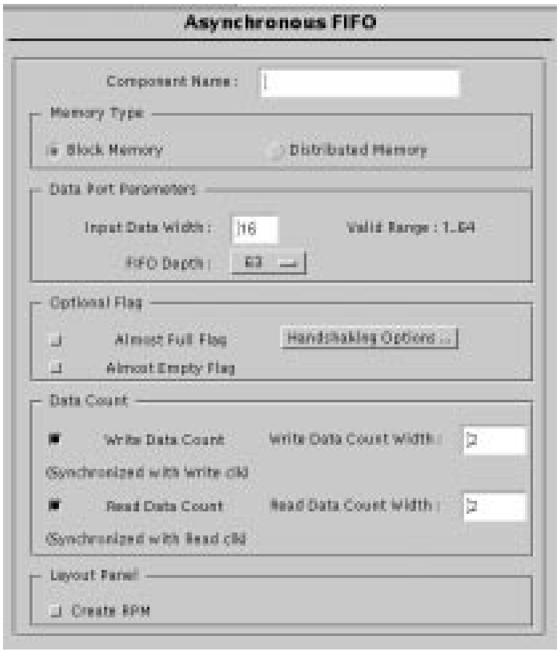
Xilinx Inc.
2100 Logic Drive
San Jose, CA 95124
Phone: +1 408-559-7778
Fax: +1 408-559-7114
E-mail: coregen@xilinx.com
URL: www.xilinx.com/support/techsup/appinfo
www.xilinx.com/ipcenter

Features

- Drop-in module for Virtex, Virtex™-E, and Spartan™-II FPGAs
- Supports data widths up to 64 bits
- Supports memory depths of up to 4095 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) provides visibility into number of data words currently in the FIFO, synchronized to either clock domain
- Incorporates Xilinx Smart-IP technology for maximum performance
- To be used with version 2.1i or later of the Xilinx CORE Generator System

Functional Description

The Asynchronous FIFO is a First In First Out memory queue. Its control logic performs all the necessary read and write pointer management, generates status flags, and optional handshake signals for interfacing to user logic. The individual read and write ports are fully synchronous (all operations qualified by a rising clock edge), but this FIFO



Asynchronous FIFO

Component Name:

Memory Type
[✓ Block Memory] Distributed Memory

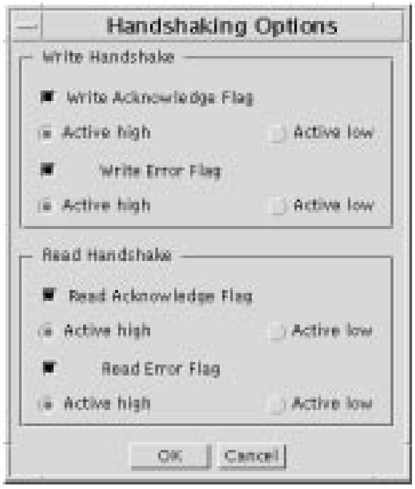
Data Port Parameters
Input Data Width: 16 Valid Range: 1..64
FIFO Depth: 63

Optional Flag
☐ Almost Full Flag
☐ Almost Empty Flag

Data Count
[✓] Write Data Count Write Data Count Width: 2
(Synchronized with Write clk)
[✓] Read Data Count Read Data Count Width: 2
(Synchronized with Read clk)

Layout Panel
☐ Create RPM

Figure 1: Asynchronous FIFO Parameterization Screen



Handshaking Options

Write Handshake
[✓] Write Acknowledge Flag
[✓ Active high] Active low
[✓] Write Error Flag
[✓ Active high] Active low

Read Handshake
[✓] Read Acknowledge Flag
[✓ Active high] Active low
[✓] Read Error Flag
[✓ Active high] Active low

Figure 2: Handshaking Options Dialog Box

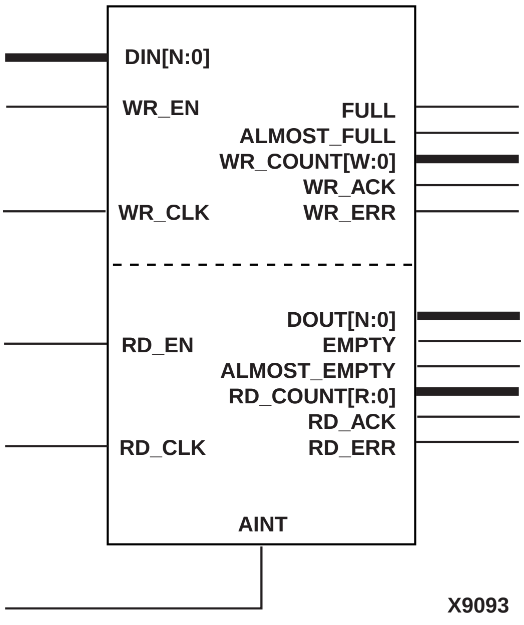


Figure 3: Core Schematic Symbol

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 or conversely 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, you may enable a count vector 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. In keeping with the fully synchronous interface, you can synchronize the count to either clock domain or two independent counts may be enabled, one for each clock domain.

Synchronization and Timing Issues

As previously stated, the read and write ports may be operated on independent asynchronous clock domains. However, the user interface logic still needs to be concerned with synchronization issues. The Core Schematic Symbol, see Figure 3, divides the signals between 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 domain and should only be sampled by logic operating on a synchronous clock. FIFO performance can be effectively constrained and analyzed by placing the desired **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 mode 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. Stated another way, the almost flags are active during the almost condition and their respective empty or full conditions.

Optional handshake signals are provided to simplify the user control logic designed to interact with the FIFO. The wr_ack and wr_err signals indicate acknowledgment or rejection of requested write operations (wr_en active) respectively. Similarly, rd_ack and rd_err signals indicate the acknowledgment or rejection of read operations (rd_en active). Each of these control signals maybe 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, (wr_en or rd_en) if active, 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 then both the acknowledgment and the error signal will be inactive during the subsequent clock period.

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 only be sampled 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 domain is one cycle, e.g. 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. The latency for operations in the opposing clock domain can be up to three clock cycles, e.g. in the case of the wr_count reads operations that may have been performed during the immediately three prior rd_clk periods will not be reflected in the data count vector. This latency was a clock frequency versus count accuracy trade-off and is not as extreme as it may appear at first glance.

Consider the following scenario, FIFO depth of 63, wr_count[1:0] is feedback to the users write logic to throttle back write operations. As writes are performed the first wr_count of 11, corresponds to 110000 (=48). As long as the users 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 users control logic is assured that at least 15 (63-48) additional memory locations are available in the queue. There could be a few more due to recent read operations occurring on the read side, but this only increases the available memory locations. In this scenario at least 14 additional writes may be performed after the write that causes the wr_count to transition from 10 to 11. In another scenario, the users read control logic may wish to wait for a fixed number of data words to be present in the FIFO in preparation for performing a burst operation. In this case since read operations are suspended before the appropriate count is reached the read latency. It is not an issue. (The user logic isn’t requesting reads). So for the same FIFO when the rd_count transitions to 11 (110000) there are at least 48 data words in the FIFO. The write operation latency means that there maybe as many as 51 words in the FIFO, but the users read logic is guaranteed that at least 48 words are present. So it can start reading the FIFO assured that at least 48 words are available or it can read and until the empty flag indicates that the FIFO is empty.

Pinout

Signal names are shown in Figure 3 and described in Table 1.

CORE Generator Parameters

The main Core Generator parameterization screen for this module is shown in Figure 1. The parameters are as follows:

- **Component Name**: The component names 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 “_”.

Table 1: Core Signal Pinout

Signal	Signal Direction	Description
DIN[N:0]	Input	Data_INput: N is any integer 1 to 64
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) of 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	WR_ACKnowledge: handshake signal WR_EN active on prior WR_CLK edge has written a data word into FIFO
WR_ERR	Output	WRite_ERRor: handshake signal indicates WR_EN active on prior WR_CLK edge was ignored and no data word was written into the FIFO.
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

Signal	Signal Direction	Description
RD_ COUNT [R:0]	Output	ReaD_COUNT: count vector (unsigned binary) of 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 RD_EN active on prior RD_CLK edge has placed next data word on Q output pins
RD_ERR	Output	ReadD_ERRor: handshake signal RD_EN active on prior RD_CLK edge was ignored and subsequently data on Q output pins was not updated

- **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 FIFO memory using LUT based dual port memory. The defaults is Block 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 - 64. The default value is 16.
- **FIFO Depth: Select** the available depth from the pull down list. Note: the available depths are dependent on the selected memory type. Since one memory location has been sacrificed in the interest of optimizing FIFO performance available, depths are (2^N –1). When using SelectRAM+, N may be any integer from 1 to 12, with additional restrictions based on the Data Width. Distributed RAM FIFOs have a maximum depth of 255 (N = 8) and Block Memory FIFO have a maximum depth of 4095.
- **Optional Flags**: Generate Almost Full and Almost Empty status flags by selecting the appropriate check boxes. The default value is unchecked.
- **Optional Handshake Signals**: Handshaking control signals (acknowledge and/or error) can be enabled via the Handshaking Options button. A pop up dialog box will appear, as shown in Figure 2. Each of the four handshake signals (write acknowledge, write error, read acknowledge, and read error) can be enabled by selecting the appropriate check box. Selecting any handshaking signal will enable its associated Active High, Active Low radio buttons (default is Active High). To make any of these flags Active Low, check the corresponding Active Low check box. The default state for all four handshaking signals is disabled.
- **Data Count**: Two Data Counts, one for each clock domain may 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 in will turn red and core generation will be inhibited until this error is corrected. The default value is 2 (encoded quadrant flags).
- **Create RPM**: When this box is checked the module will be generated will relative location attributes attached. The FIFO with be produced with two (or three if distributed memory was selected) individual RPMs. A single RPM is not produced to allow an RPM'd FIFO to support varying footprints.

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.

Table 3 shows the XCO file parameters and values, as well as summarizing the GUI defaults. The following is an example of the CSET parameters in an XCO file:

```
CSET component_name = my_fifo_name
CSET fifo_depth = 255
CSET input_data_width = 8
CSET memory_type = block
CSET almost_full_flag = TRUE
CSET almost_empty_flag = TRUE
CSET write_count = TRUE
CSET write_count_width = 1
CSET read_count = TRUE
CSET read_count_width = 8
CSET write_acknowledge = TRUE
CSET write_acknowledge_sense = active_high
CSET write_error = TRUE
CSET write_error_sense = active_low
CSET read_acknowledge = TRUE
CSET read_acknowledge_sense = active_high
CSET read_error = TRUE
CSET read_error_sense = active_low
CSET create_rpm = TRUE
```

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 FIFOs memory and control logic. Table 3 lists the number of SelectRAM+ blocks required to implement various width and depth combinations when using SelectRAM+ blocks for the FIFO's memory.

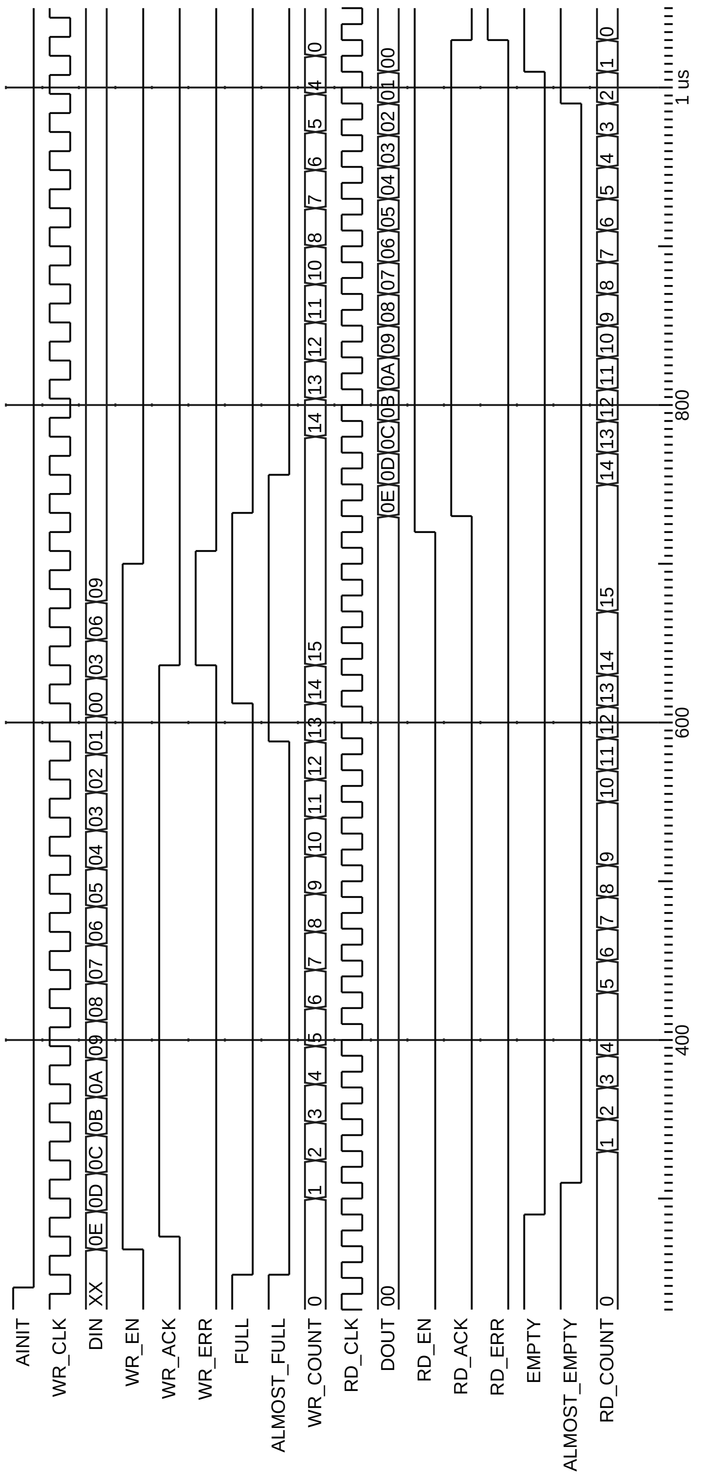


Figure 4: Write/Read Waveform for 15deep by 16 FIFO

Table 4 shows the approximate number of slices per bit for a distributed ram based FIFO. Multiply this number by the data width to determine the total slice count for the memory. 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 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)

Table 3: 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_wiidth	Integer in the range 1 to 64	16
fifo_depth	Integer in the range 15 to 4095. Must be equal to (2^N-1;, N = 4 to 12)	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

Table 2: Select RAM+ Usage

Data Width	FIFO Depth								
	15	31	63	127	255	511	1023	2047	4095
1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	1	1	1	1	2
3	1	1	1	1	1	1	1	1	3
4	1	1	1	1	1	1	1	2	4
5 to 6	1	1	1	1	1	1	2	3	NS[1]
7 to 8	1	1	1	1	1	1	2	4	NS
9 to 12	1	1	1	1	1	2	3	NS	NS
13 or 16	1	1	1	1	1	2	4	NS	NS
17 to 24	2	2	2	2	2	3	NS	NS	NS
25 to 32	2	2	2	2	2	4	NS	NS	NS
33 to 48	3	3	3	3	3	NS	NS	NS	NS
49 to 64	4	4	4	4	4	NS	NS	NS	NS

Note:
1. FIFOs requiring more than 4 BlockRAM blocks are not currently supported.

- +(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 2, and an additional 99 slices (N=10) for the control logic.

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

Table 4: Resource Utilization (LUT/MUXF5/MUXF6/FD) for Distributed RAM FIFO Memory Only (per bit, multiply by data width)

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

Ordering Information

This core is downloadable free of charge from the Xilinx IP Center (www.xilinx.com/ipcenter), for use with the Xilinx Core Generator System version 2.1i and later. The Core Generator System 2.1i tool is bundled with the Alliance 2.1i and Foundation 2.1i implementation tools.

To order Xilinx software contact your local Xilinx sales representative at www.xilinx.com/company/sales.htm.

Table 5: Virtex ASYNC_FIFO_V1_0 Performance Benchmarking (SelectRAM+ implementation)

PART	FIFO Implementation		
V50PQ240	255x16 no options (BASE)	255x16 all options (MID)	1023X8 all options BIG)
-4	110 MHz – (9.0 nS	106 MHz – (9.6 nS)	96 MHz – (10.4 nS)
-5	125 MHz – (7.9 nS)	120 MHz – (8.3 nS	105 MHz - (9.5 nS)
-6	150 MHz – (6.6 nS)	136 MHz – (7.3 nS)	120 MHz - (8.3 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 your own floorplanning.
2. Over constraining the FIFO (applying overly aggressive timing constraints) will degrade the achievable performance. For example, applying a 6.0nS constraint to the BASE implementation (-6) will result in a placed and routed implementation that is considerably slower than the 6.6nS shown in the table.