Asymmetric Width FIFO Core User Guide

UG-CORE-ASYM-FIFO-v1.0
July 2021
www.efinixinc.com

Contents

Introduction

The Asymmetric Width FIFO core is a customizable first-in first-out memory queue that uses block RAM in the FPGA for storage. The core has parameters you use to create a custom instance. For example, you can set the FIFO depth, the data bus width, whether the read and write domains are synchronous or asynchronous, etc.

Features

- Depths up to 131,072 words
- Data widths from 1 to 1024 bits
- Symmetric or non-symmetric aspect ratios (read-to-write port ratios ranging from 1:16 to 16:1)
- Synchronous or asynchronous clock domains supports standard
- Programmable full and empty status flags, set by user–defined parameters
- Almost full and almost empty flags indicate one word left
- Configurable handshake signals
- FIFO datacount to indicate how many words available in FIFO
- Option to exclude optional flags
- Verilog RTL and simulation testbench

FPGA Support

The Asymmetric Width FIFO core supports all Trion® and Titanium FPGAs.

Titanium Resource Utilization and Performance

Note: This is an early access version of the IP core. The resources utilization and performance of this version is not fully optimized.

Table 1: Synchronous Clock FIFO

FPGA	Asymmetric Width	XLR	Memory Blocks	DSP48 Blocks	f_{MAX} (MHz) - clk_i	Efinity Version
Ti60 F225 ES	0	160	2	0	422	2021.1
	1	152	2	0	440	
	2	150	2	0	377	
	3	136	2	0	433	
	4	60	2	0	474	
	5	56	4	0	497	
	6	55	10	0	501	
	7	52	20	0	539	
	8	50	32	0	501	

Table 2: Asynchronous Clock FIFO

FPGA	Asymmetric Width	XLR	Memory Blocks	DSP48 Blocks	f_{MAX} (MHz)		Efinity Version
					wr_clk_i	rd_clk_i	
Ti60 F225 ES	0	160	2	0	324	312	2021.1
	1	152	2	0	341	350	
	2	150	2	0	340	326	
	3	136	2	0	323	346	
	4	60	2	0	426	378	
	5	56	4	0	383	391	
	6	55	10	0	395	408	
	7	52	20	0	416	430	
	8	50	32	0	412	431	

Trion Resource Utilization and Performance

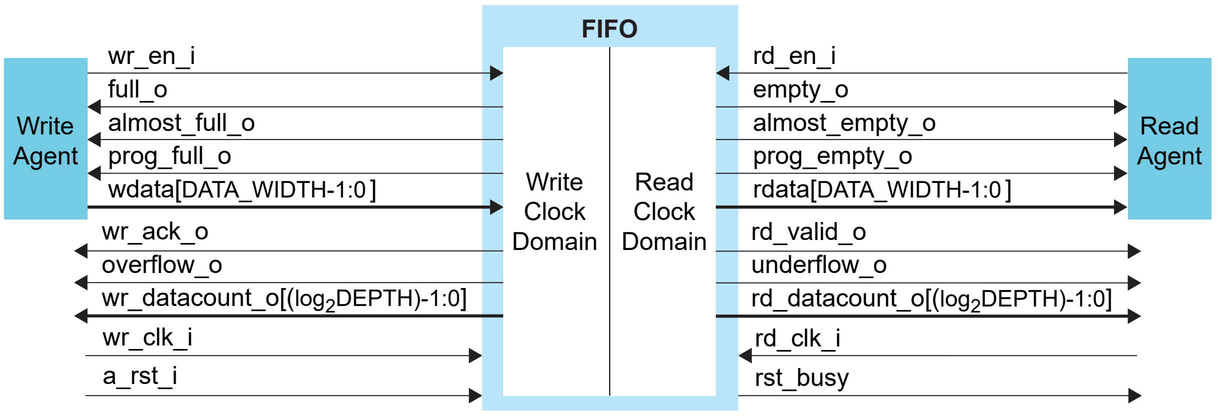
FPGA	Asymmetric Width	Logic Utilization (LUTs)	Memory Blocks	Multipliers	f_{MAX} (MHz) - clk_i	Efinity Version
T20 BGA256	0	161	4	0	134	2021.1
	1	161	4	0	134	
	2	157	4	0	140	
	3	162	4	0	144	
	4	70	4	0	180	
	5	57	4	0	172	
	6	54	8	0	176	
	7	52	20	0	168	
	8	47	32	0	165	

FPGA	Asymmetric Width	LUTs	Memory Blocks	Multipliers	f_{MAX} (MHz)		Efinity Version
					wr_clk_i	rd_clk_i	
T20 BGA256	0	255	4	0	110	110	2021.1
	1	249	4	0	110	110	
	2	250	4	0	118	118	
	3	250	4	0	110	110	
	4	153	4	0	132	132	
	5	141	4	0	134	123	
	6	128	8	0	138	134	
	7	116	20	0	133	146	
	8	104	32	0	131	131	

Functional Description

The Asymmetric Width FIFO core is a first-in first-out memory queue for any application requiring an ordered storage buffer and retrieval. The core provides an optimized solution using the block RAM in Trion® and Titanium FPGAs. The core supports synchronous (read and write use the same clock) and asynchronous (read and write use different clocks) clocking.

Figure 1: FIFO System Block Diagram



Ports

Table 3: Asymmetric Width FIFO Core Clock, Reset and Datacount Ports

Port	Synchronous	Asynchronous	Direction	Description
a_rst_i	✓	✓	Input	Reset. Asynchronous reset signal that initializes all internal pointers and output flags.
rst_busy		✓	Output	When asserted, this signal indicate the core is being reset.
wr_clk_i		✓	Input	Write clock. All signals in the write domain are synchronous to this clock.
rd_clk_i		✓	Input	Read clock. All signals in the read domain are synchronous to this clock.
clk_i	✓		Input	Clock. All signals on the write and read domains are synchronous to this clock.
wr_datacount_o [*n*-1:0]		✓	Output	Asynchronous FIFO write domain data count. n=log$_2$[DEPTH].
rd_datacount_o [*n*-1:0]		✓	Output	Asynchronous FIFO read domain data count. n=log$_2$[DEPTH].
datacount_o [*n*-1:0]	✓		Output	Synchronous FIFO data count. n=log$_2$[DEPTH].

Table 4: Asymmetric Width FIFO Core Write Ports

For both synchronous and asynchronous clocks.

Port	Direction	Description
wdata [*m*-1:0]	Input	Write data. The input data bus used when writing to the FIFO buffer. *m*=DATA_WIDTH.
wr_en_i	Input	Write enable. If the FIFO buffer is not full, asserting this signal causes data (on wdata) to be written to the FIFO.
full_o	Output	Full flag. When asserted, this signal indicates that the FIFO buffer is full. Write requests are ignored when the FIFO is full. Initiating a write while full is not destructive to the FIFO.
almost_full_o	Output	Optional, almost full. When asserted, this signal indicates that only one more write can be performed before the FIFO is full.
prog_full_o	Output	Optional, programmable full. This signal is asserted when the number of words in the FIFO is greater than or equal to the assert threshold. It is deasserted when the number of words in the FIFO is less than the negate threshold.
wr_ack_o	Output	Optional, write acknowledge. This signal indicates that a write request (wr_en_i) during the prior clock cycle succeeded.
overflow_o	Output	Optional, overflow. This signal indicates that a write request (wr_en_i) during the prior clock cycle was rejected because the FIFO buffer is full. Overflowing the FIFO is not destructive to the contents of the FIFO.

Table 5: Asymmetric Width FIFO Core Read Ports

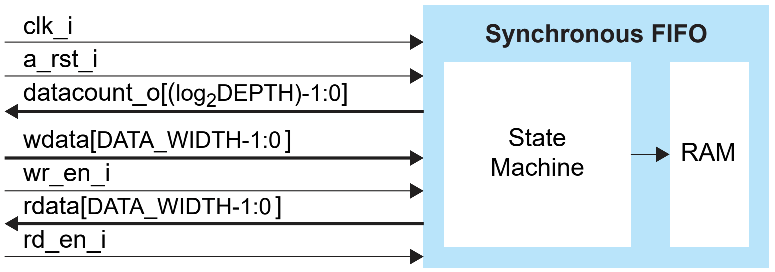
For both synchronous and asynchronous clocks.

Port	Direction	Description
rdata [*m*-1:0]	Output	Read data. The output data bus driven when reading the FIFO buffer. *m*=DATA_WIDTH.
rd_en_i	Input	Read enable. If the FIFO buffer is not empty, asserting this signal causes data to be read from the FIFO (output on rdata).
empty_o	Output	Empty flag. When asserted, this signal indicates that the FIFO buffer is empty. When empty, Read requests are ignored. Initiating a read while empty is not destructive to the FIFO.
almost_empty_o	Output	Optional, almost empty flag. When asserted, this signal indicates that only one word remains in the FIFO buffer before it is empty.
prog_empty_o	Output	Optional, programmable empty. This signal is asserted when the number of words in the FIFO buffer is less than or equal to the assert threshold. It is de-asserted when the number of words in the FIFO exceeds the negate threshold.
rd_valid_o	Output	Optional, read valid. This signal indicates that valid data is available on the output bus (rdata).
underflow_o	Output	Optional, underflow. Indicates that the read request (rd_en_i) during the previous clock cycle was rejected because the FIFO buffer is empty. Underflowing the FIFO is not destructive to the FIFO.

Synchronous FIFO Operation

The FIFO core signals are synchronized on the rising edge clock of the respective clock domain. If you want to synchronize to the falling clock edge, use an inverter before sending the signal to the clock input.

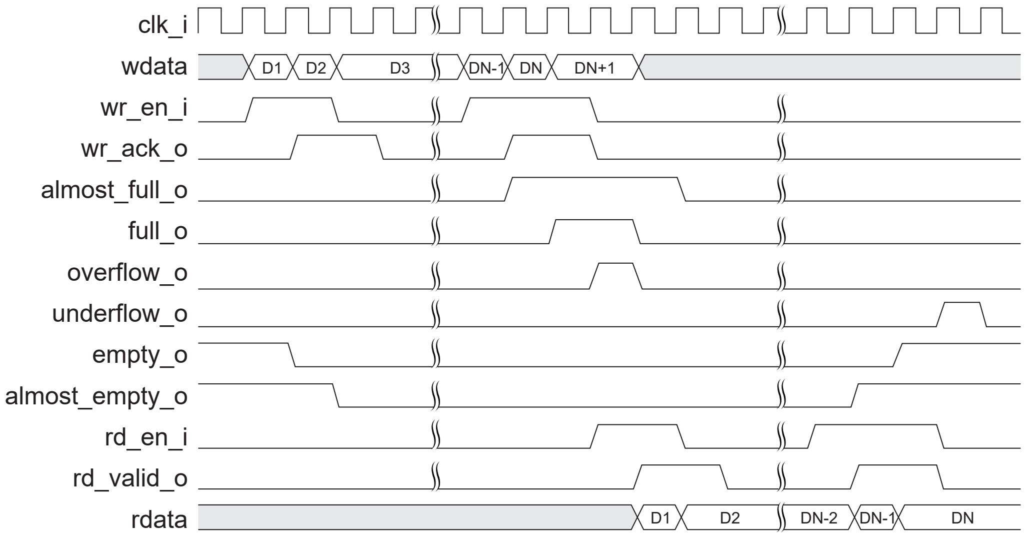
Figure 2: Synchronous FIFO Block Diagram



Standard Mode

The following waveform shows the FIFO behavior in standard mode when it is written until full and then read until empty. D1 and DN are the first and last data, respectively.

Figure 3: Synchronous FIFO Standard Mode Waveform

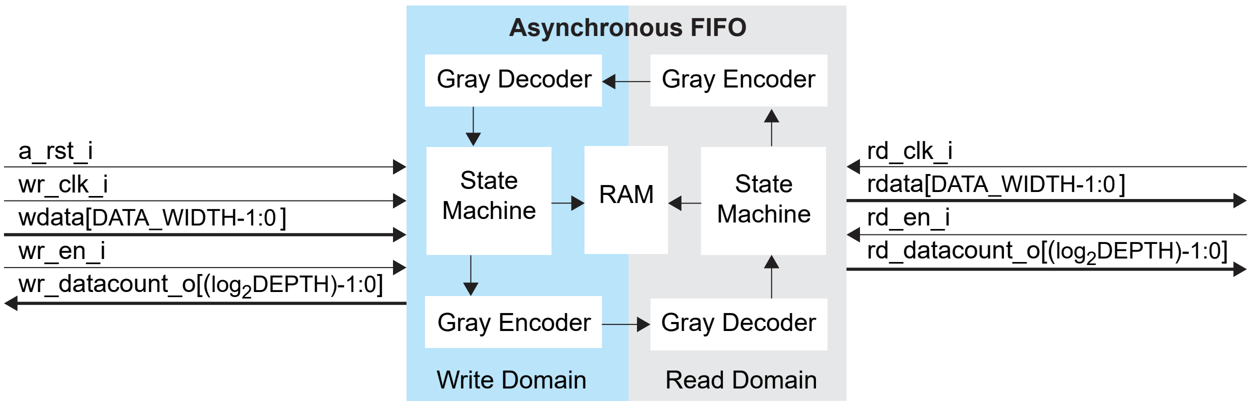


If the system tries to write data DN+1 when `full_o` is asserted, the core ignores DN+1 and asserts `overflow_o`. `full_o` deasserts during a read request, signaling that the FIFO is ready for more write requests. When the last data is read from the FIFO, the core asserts `empty_o`, indicating there is no more data. Further read requests when there is no more data triggers an assertion on `underflow_o`.

Asynchronous FIFO Operation

With an asynchronous FIFO, the two protocols can work in their respective clock domains and still transfer reliable data to each other. When there is a write or read request affecting its own respective domain's flags, the asynchronous FIFO has 0 delays. Whereas when affecting the other domain's flags, it has a 1 clock cycle delay from its respective domain plus 2 clock cycles of the other domain. For example, a write request only reflects on the read domain after 1 write clock cycle plus 2 read clock cycles and vice versa. Enabling the PIPELINE_REGadds 1 more additional clock cycle of the other domain on top of it. Refer to the latency table for asynchronous FIFO in Latency on page 13 for more info.

Figure 4: Asynchronous FIFO Block Diagram



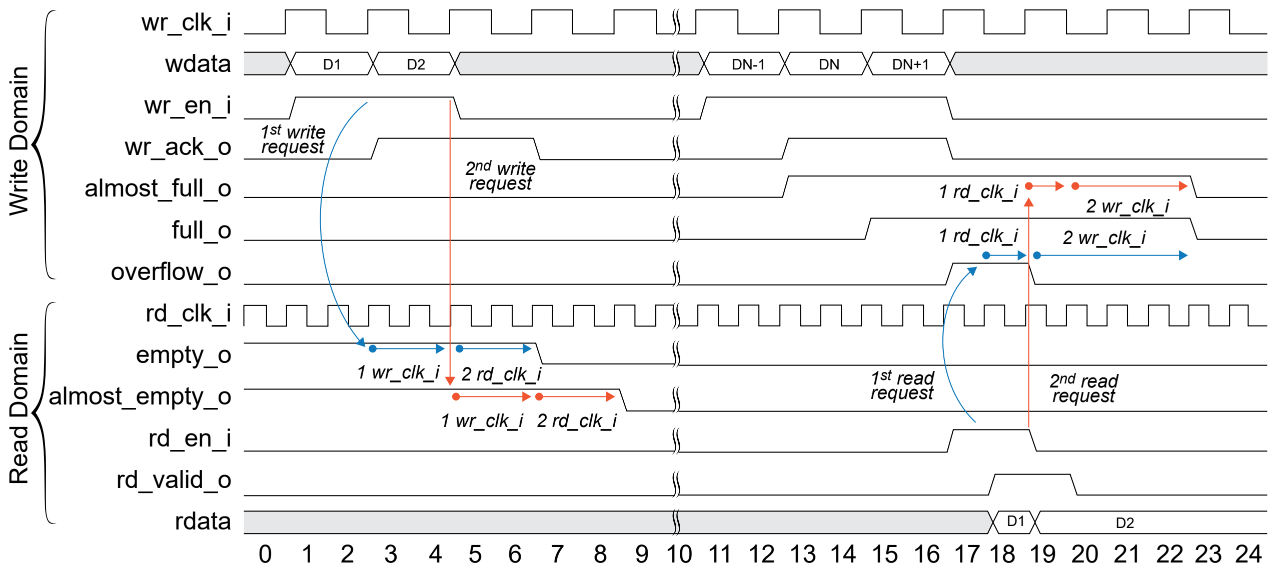
For asynchronous FIFO, a write operation affecting the write domain flags and a read operation affecting the read domain flags have the same behavior as the synchronous FIFO except when they are affecting crossed domain flags. The following examples emphasize the cross-clock domain flags update latency.

Standard Mode

The following figures show examples of asynchronous FIFO standard mode with a faster read clock and write clock, respectively. The waveforms show the FIFO written until full and a few read requests afterwards.

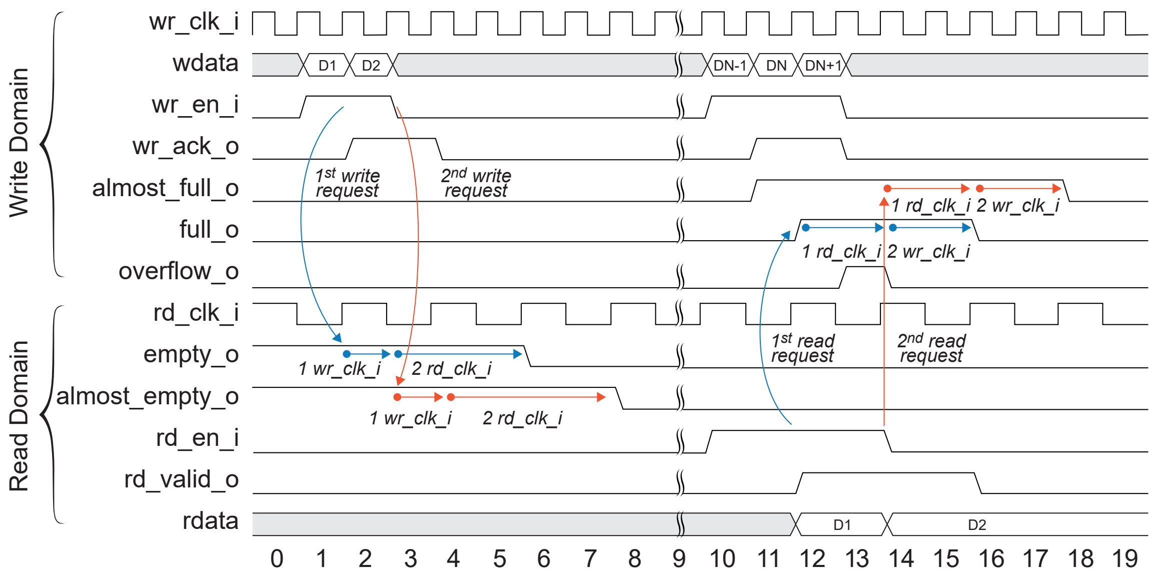
In the read example shown in Figure 5: Asynchronous FIFO Standard Mode Faster Read Clock with PIPELINE_REG=0 on page 9, the read clock frequency is double that of the write clock with the same phase. When there is a write request at node 2, empty_o does not deassert immediately; instead, it deasserts 1 write clock plus 2 clock read clocks later at node 6. Similarly, almost_empty_o deasserts at node 8, which is 1 write clock plus 2 read clocks later after the second write request at node 4. almost_full_o and full_o deassert at the same time at node 22 because there are 2 read requests detected before the write domain is synchronized at node 20.

Figure 5: Asynchronous FIFO Standard Mode Faster Read Clock with PIPELINE_REG=0



In the write example shown in Figure 6: Asynchronous FIFO Standard Mode Faster Write Clock with PIPELINE_REG=0 on page 9, the write clock frequency is double that of the read clock with the same phase. The `empty_o` deasserts at node 5 and `almost_empty_o` deasserts at node 7. Each of these signals are affected by write requests on node 1 and node 2 respectively. Read requests at node 11 and 13 reflect on the write domain at node 15 and 17, respectively.

Figure 6: Asynchronous FIFO Standard Mode Faster Write Clock with PIPELINE_REG=0



Asymmetric Width Operation

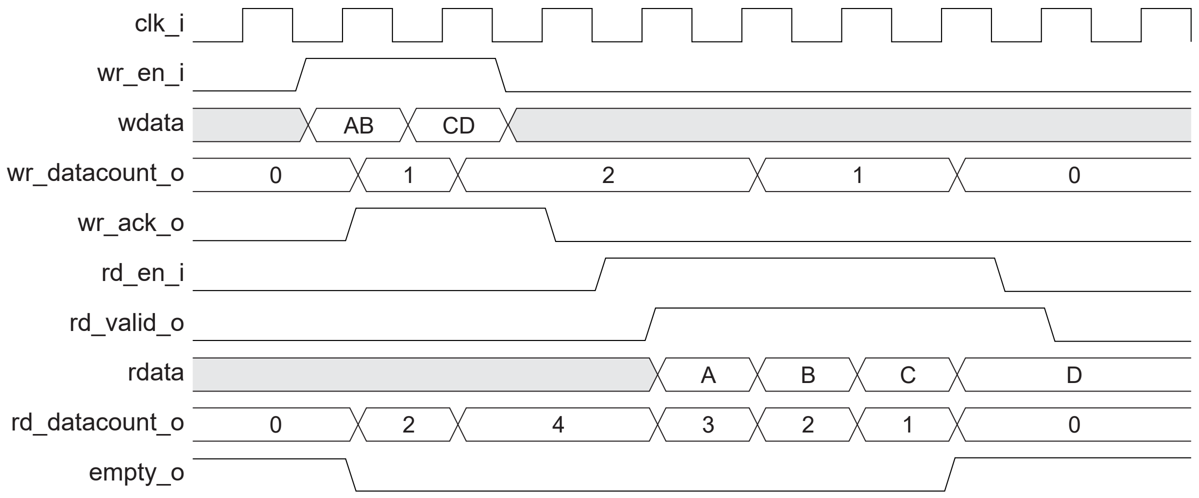
Asymmetric aspect ratios allow the input and output of the FIFO width and depth to be configured in differently. You only need to configure the write width and depth, while the read width and read depth are be computed automatically by the Asymmetric Width FIFO based on your parameter settings. The following table lists the supported asymmetric width ratio.

Table 6: Supported Asymmetric Width FIFO Ratio

Ratio	Write Width	Read Width	Write Depth	Read Depth
16:1	N	N/16	2^M	2^M x 16
8:1	N	N/8	2^M	2^M x 8
4:1	N	N/4	2^M	2^M x 4
2:1	N	N/2	2^M	2^M x 2
1:1	N	N	2^M	2^M
1:2	N	N*2	2^M	2^M / 2
1:4	N	N*4	2^M	2^M / 4
1:8	N	N*8	2^M	2^M / 8
1:16	N	N*16	2^M	2^M / 16

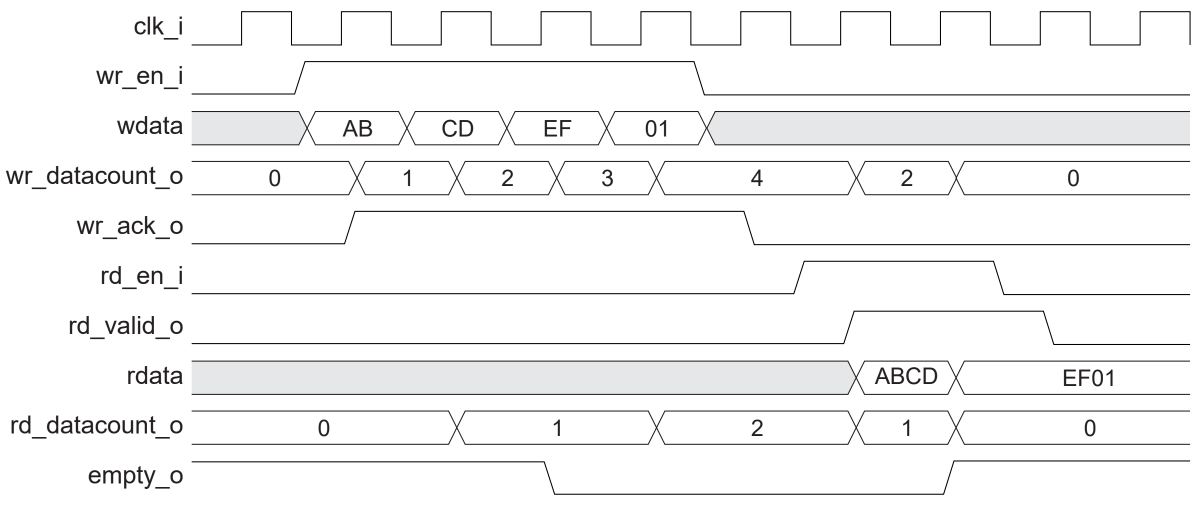
In operations with 2:1 aspect ratio, the write width is two times the read width. In the example below, each write request has 8-bit data which requires 2 read requests (4-bit width per clock cycle) to free-up the entry.

Figure 7: 2:1 Aspect Ratio Example Waveform



In operations with 1:2 aspect ratio, the read width is two times the write width. In the example below, each write request has 8-bit data where two write requests are required to contribute to a single read word (16-bit width).

Figure 8: 1:2 Aspect Ratio Example Waveform



Programmable Full and Empty Signals

The FIFO core supports user-defined full and empty signals with customized depths (`prog_full_o` and `prog_empty_o`). To enable these signals, set the `PROGRAMMABLE_FULL` or `PROGRAMMABLE_EMPTY` parameters as `STATIC_SINGLE` or `STATIC_DUAL`. Refer to Parameters for more info on the available values.

Important: For the asynchronous FIFO, these signals are synchronized to their respective clock domain's available words.

Table 7: prog_full_o Assert and Deassert Conditions

Value	Type	Condition
STATIC_SINGLE	Assert	*number of words in FIFO* ≥ PROG_FULL_ASSERT
	Deassert	*number of words in FIFO* < PROG_FULL_ASSERT
STATIC_DUAL	Assert	*number of words in FIFO* ≥ PROG_FULL_ASSERT
	Deassert	*number of words in FIFO* < PROG_FULL_NEGATE

Table 8: prog_empty_o Assert and Deassert Conditions

Value	Type	Condition
STATIC_SINGLE	Assert	*number of words in FIFO* ≤ PROG_EMPTY_ASSERT
	Deassert	*number of words in FIFO* > PROG_EMPTY_ASSERT
STATIC_DUAL	Assert	*number of words in FIFO* ≤ PROG_EMPTY_ASSERT
	Deassert	*number of words in FIFO* > PROG_EMPTY_NEGATE

To avoid erratic behavior, follow these rules for `STATIC_DUAL` modes:

- `PROG_FULL_ASSERT` ≥ `PROG_FULL_NEGATE`
- `PROG_EMPTY_ASSERT` ≤ `PROG_EMPTY_NEGATE`

Reset

The Asymmetric Width FIFO core can be reset through the `a_rst_i` reset signal, which is active high. In synchronous mode, the FIFO operation can be started as soon as 3 cycles after the reset signal go low. For asynchronous mode, the FIFO operation only can be started after the `reset_busy` signal go low. Ensure that the reset pulse is equivalent or more than 2 clock cycles of the slowest clock

Datacount

The FIFO core includes datacount signal as output. Synchronous FIFO enables `datacount_o` while asynchronous FIFO enables both `wr_datacount_o` and `rd_datacount_o`.

The datacount is zero when the FIFO is in empty and full state. You must ensure that the width of datacount is $\log_2$(`DEPTH`) to get the correct value.

Note: Always refer to the `empty_o` and `full_o` signals when reading or writing FIFO.

Latency

This section defines the latency of the output signals in the FIFO core. The output signals latency are updated in response to the read or write requests. Latency is described in the following waveform. A `0` latency means the signal is asserted or deasserted at the same rising edge of the clock at which the write or read request is sampled. A latency of 1 means the signal is asserted or deasserted at the next rising edge of the clock.

Synchronous FIFO

Table 9: Synchronous FIFO Write Flags Update Latency (clk_i) Due to `wr_en_i` and `rd_en_i` Signals

Port	wr_en_i	rd_en_i
wr_ack_o	0	–
full_o	0	0
almost_full_o	0	0
prog_full_o	0	0
overflow_o	0	–

Table 10: Synchronous FIFO Read Flags Update Latency Due to `wr_en_i` and `rd_en_i` Signals

Port	wr_en_i	rd_en_i
rd_valid_o	–	0[1]
empty_o	0	0
almost_empty_o	0	0
prog_empty_o	0	0
underflow_o	–	0
datacount_o	0	0

Asynchronous FIFO

Table 11: Asynchronous FIFO Write Flags Update Latency Due to wr_en_i

Port	Latency (PIPELINE_REG=0)	Latency (PIPELINE_REG=1)
wr_ack_o	0	0
full_o	0	0
almost_full_o	0	0
prog_full_o	0	0
overflow_o	0	0
wr_datacount_o	0	0

[1] OUTPUT_REG adds one latency to these signal.

Table 12: Asynchronous FIFO Read Flags Update Latency Due to wr_en_i

Port	Latency (PIPELINE_REG=0)	Latency (PIPELINE_REG=1)
rd_valid_o	–	–
empty_o	1 wr_clk_i + 2 rd_clk_i	1 wr_clk_i + 3 rd_clk_i
almost_empty_o	1 wr_clk_i + 2 rd_clk_i	1 wr_clk_i + 3 rd_clk_i
prog_empty_o	1 wr_clk_i + 2 rd_clk_i	1 wr_clk_i + 3 rd_clk_i
underflow_o	–	–
rd_datacount_o	1 wr_clk_i + 2 rd_clk_i	1 wr_clk_i + 3 rd_clk_i

Table 13: Asynchronous FIFO Write Flags Update Latency Due to rd_en_i

Port	Latency (PIPELINE_REG=0)	Latency (PIPELINE_REG=1)
wr_ack_o	–	–
full_o	1 rd_clk_i + 2 wr_clk_i	1 rd_clk_i + 3 wr_clk_i
almost_full_o	1 rd_clk_i + 2 wr_clk_i	1 rd_clk_i + 3 wr_clk_i
prog_full_o	1 rd_clk_i + 2 wr_clk_i	1 rd_clk_i + 3 wr_clk_i
overflow_o	–	–
wr_datacount_o	1 rd_clk_i + 2 wr_clk_i	1 rd_clk_i + 3 wr_clk_i

Table 14: Asynchronous FIFO Read Flags Update Latency Due to rd_en_i

Port	Latency (PIPELINE_REG=0)	Latency (PIPELINE_REG=1)
rd_valid_o	0(2)	0(3)
empty_o	0	0
almost_empty_o	0	0
prog_empty_o	0	0
underflow_o	0	0
rd_datacount_o	0	0

(2) OUTPUT_REG adds one latency to these signal.
(3) OUTPUT_REG adds one latency to these signal.

Customizing the Asymmetric Width FIFO

The core has parameters so you can customize its function. You set the parameters in the **test_param.vh** file.

Table 15: Asymmetric Width FIFO Core Parameter

Parameter	Options	Description
SYNC_CLK	Asynchronous, Synchronous	Defines whether the FIFO read and write domain is synchronous or asynchronous. Default: Synchronous
DEPTH	16 – 131072	Defines the FIFO depth, which determines the maximum number of words the FIFO can store before it is full. The depth is multiples of 2 from 16 – 2^{17}. Default: 512
DATA_WIDTH	1 – 256	Defines the FIFO's read and write data bus widths. Default: 32
MODE	STANDARD	Defines the FIFO's read mode. Default: STANDARD
OUTPUT_REG	Enable, Disable	Adds one pipeline stage to rdata and rd_valid_o to improve timing delay out from efx_ram. Default: Enable
PROG_FULL_ASSERT	1 – DEPTH	Threshold value when prog_full_o is enabled. When Enable Programmable Full Option is: STATIC_SINGLE: Single threshold value for assertion and deassertion of prog_full_o. STATIC_DUAL: Upper threshold value for assertion of prog_full_o. Default: 512
PROGRAMMABLE_FULL	NONE, STATIC_SINGLE, STATIC_DUAL	Controls the prog_full_o signal: NONE: Disabled. STATIC_SINGLE: Enabled, asserts and deasserts at a single threshold value. (default) STATIC_DUAL: Enabled, asserts or deasserts at different threshold values.
PROG_FULL_NEGATE	1 – Programmable Full Assert Value	Use when PROGRAMMABLE_FULL is set to STATIC_DUAL. Sets the lower threshold value for prog_full_o during deassertion. Default: 512
PROG_EMPTY_ASSERT	0 – (FIFO Depth-1)	Threshold value when prog_empty_o is enabled. When Enable Programmable Full Option is: STATIC_SINGLE: Single threshold value for assertion and deassertion of prog_empty_o. STATIC_DUAL: Lower threshold value for assertion of prog_empty_o. Default: 0

Parameter	Options	Description
PROG_EMPTY_NEGATE	Programmable Empty Assert Value - (DEPTH-1)	Use when PROGRAMMABLE_EMPTY is set to STATIC_DUAL. Sets the upper threshold value for prog_empty_o during deassertion. Default: 0
PROGRAMMABLE_EMPTY	NONE, STATIC_SINGLE, STATIC_DUAL	Controls the prog_empty_o signal: NONE: Disabled. STATIC_SINGLE: Enabled, asserts and deasserts at a single threshold value. (default) STATIC_DUAL: Enabled, asserts or deasserts at different threshold values.
OPTIONAL_FLAGS	Enable, Disable	Enables the optional signals: wr_ack_o, almost_full_o, overflow_o, rd_valid_o, almost_empty_o and underflow_o. You can disable this feature to improve macro timing. Default: Enable
PIPELINE_REG	Enable, Disable	Applicable to Asynchronous FIFO mode only. Adds one latency of the opposing clock domain to all applicable output signals when wr_en_i or rd_en_i signal is asserted. Enable this feature to improve the macro timing. You can disable this feature if a project does not require fast speed. Default: Enable
SYNC_STAGE	1 - 4	Number of synchronization stages in asynchronous mode. Default: 2
ASYM_WIDTH_RATIO	0 - 8	Selects asymmetrical width ratios: 0: 16:1 ratio 1: 8:1 ratio 2: 4:1 ratio 3: 2:1 ratio 4: 1:1 ratio (default) 5: 1:2 ratio 6: 1:4 ratio 7: 1:8 ratio 8: 1:16 ratio

Asymmetric Width FIFO Testbench

The core includes a simulation testbench, **tb.sv**, which performs asymmetric width FIFO write and FIFO read operation according to the configured asymmetric width ratio. The FIFO read data is verified during the simulation.

Refer to the **readme.txt** file in the testbench folder for detailed steps to run the simulation testbench.

The simulation run time depends on the `DEPTH` parameter set. The default value is 512.

Revision History

Table 16: Revision History

Date	Version	Description
July 2020	1.0	Initial release.