ROCKSAVAGE
TECHNOLOGY

DynamicFifo
Product User Guide

tech.rocksavage.chiselWare.dynamicfifo

chiselWare certified to level: **0** of 5

Warren Savage

July 6, 2025

Contents

1 Errata and Known Issues

1.1 Errata

- Care should be taken in creating instances of **DynamicFifo** with internal very large memory (hundreds or thousands of memory cells) as this can generate very large designs. There is currently no checks or constraints on users from doing this.
- Care should be taken regarding dynamically changing the values on the *almostEmptyLevel* and *almostFullLevel* ports when the FIFO is not empty as that may result in unpredictable behaviors on the *almostEmpty* and *almostFull* flags.

1.2 Known Issues

None.

2 Port Descriptions

The ports for **DynamicFifo** are shown below in Table 1. The width of several ports is controlled by the following input parameters:

- *dataWidth* is the width the dataIn and dataOut ports in bits
- *fifoDepth* controls the width of the external RAM addresses
- *externalRam* controls whether the ports (below, in gray) are generated for an external dual-port SRAM

Port Name	**Width**	**Direction**	**Description**
clock	1	Input	Positive edge clock
reset	1	Input	Active high reset
push	1	Input	Push a word into the FIFO
pop	1	Input	Pop a word from the FIFO
dataIn	*dataWidth*	Input	Data to be pushed into the FIFO
dataOut	*dataWidth*	Output	Data popped from the FIFO
empty	1	Output	Indicates the FIFO is empty
full	1	Output	Indicates the FIFO is full
almostEmptyLevel	log2Ceil(*fifoDepth*)	Input	Sets the threshold for the almostEmpty port. almostEmpty will be active when the FIFO is at or below this level.
almostFullLevel	log2Ceil(*fifoDepth*)	Input	Sets the threshold for the almostFull port. almostFull will be active when the FIFO is at or above this level.
ramWriteEnable	1	Output	Write enable to the external FIFO RAM
ramWriteAddress	log2Ceil(*fifoDepth*)	Output	Write address to the external FIFO RAM
ramDataIn	*dataWidth*	Output	Data to the external FIFO RAM
ramReadEnable	1	Input	Read enable to the external FIFO RAM
ramReadAddress	log2Ceil(*fifoDepth*)	Output	Read address to the external FIFO RAM
ramDataOut	*dataWidth*	Input	Data from the external FIFO RAM

Table 1: Port Descriptions

3 Parameter Descriptions

The parameters for **DynamicFifo** are shown below in Table 2.

Name	Type	Min	Max	Description
externalRam	Boolean	false	true	Determines whether to build FIFO memory with flip-flops or provide and an external interface to a SRAM
dataWidth	Int	1	≥ 1	The data width of the FIFO
fifoDepth	Int	2	≥ 2	The depth of the FIRO

Table 2: Parameter Descriptions

The DynamicFifo is instantiated into a design as follows:

```
// Instantiate small FIFO using internal flip-flops
val mySmallFifo = new DynamicFifo(
  externalRAM = false,
  dataWidth = 8,
  fifoDepth = 16)

// Instantiate large FIFO using external SRAM
val myLargeFifo = new DynamicFifo(
  externalRAM = true,
  dataWidth = 32,
  fifoDepth = 512)
```

4 Theory of Operations

4.1 Introduction

The **DynamicFifo** is a highly parameterized FIFO and FIFO controller. It is configurable as a full self-contained FIFO with internal memory being constructed from flip-flops, or a FIFO controller that uses an external SRAM for memory.

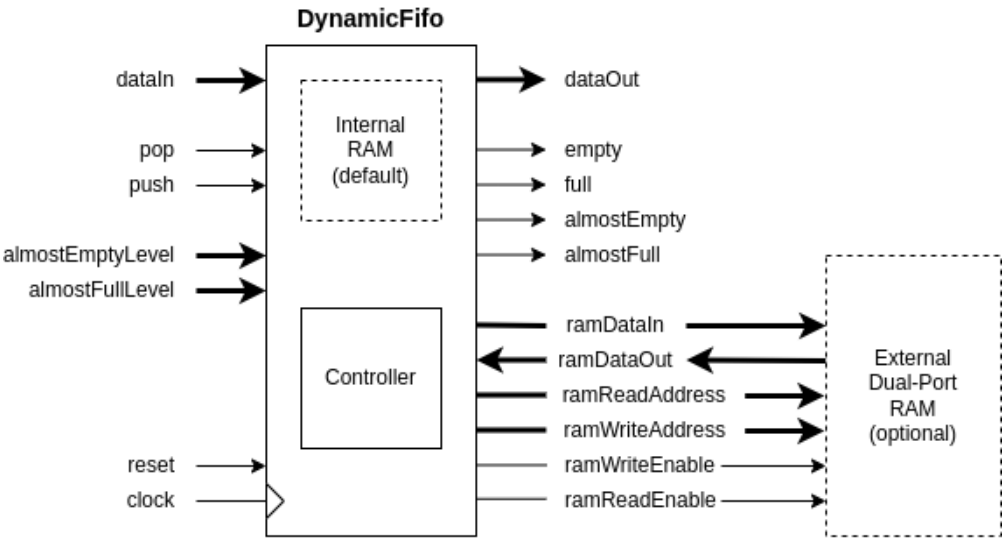

Figure 1: Block Diagram

It features the following status flags which are described in Table 1.

- empty
- full
- almostEmpty
- almostFull

When *push* is asserted, the data on the *dataIn* port is enqued on the next rising edge of *clock*. When *pop* is asserted, the top of the FIFO is dequed and immediately available on the *dataOut* port. Pop and Push operations can be simulataneous.

There are two error conditions which produce the following effects:

- When *pop* is asserted and the FIFO is empty (*empty* is active), *dataOut* will contain the last valid data held in the FIFO.
- When *push* is asserted and the FIFO is full (*full* is active), *dataIn* will be ignored and not enqued.

The *almostEmpty* and *almostFull* flags allow for additional feedback to the system that is useful for optimizing data flow control. The levels of these flags can be programmed dynamically through the *almostEmptyLevel* and *almostFullLevel* ports.

4.2 Interface Timing

DynamicFifo has a simple, synchronous interface. The timing diagram shown below in Figure 2 represents an instantiation with the following parameters.

```
val myFifo = new DynamicFifo(
  externalRAM = true,
  dataWidth = 16,
  fifoDepth = 5)
```

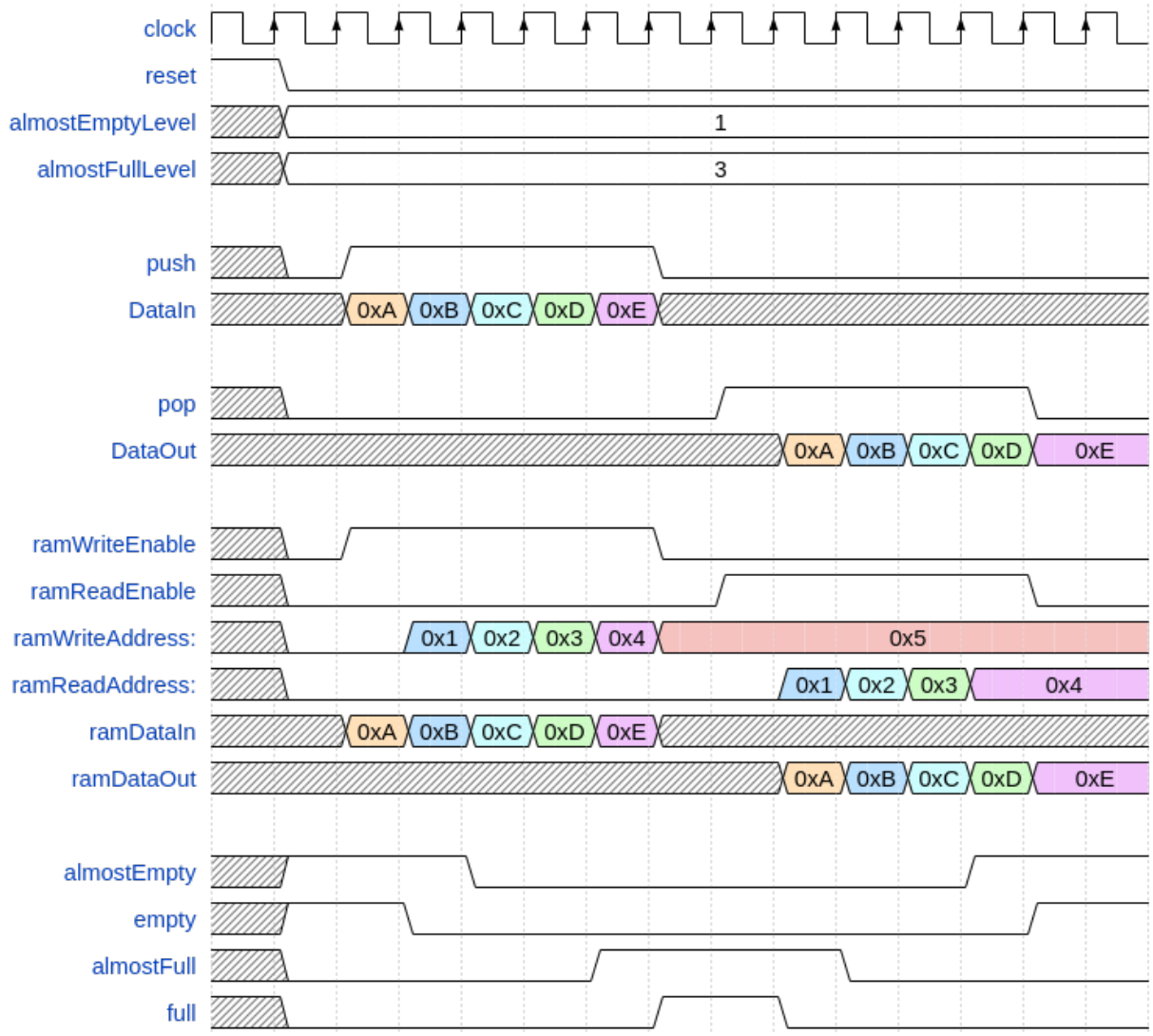

Figure 2: Timing Diagram

The *almostEmptyLevel* port is driven by external logic to a static value of 1 after reset and the *almostFull* port is driven to 3.

Beginning in the third clock cycle, 5 words of data are pushed into the FIFO. The status flags show the FIFO going from empty to full.

The FIFO is then fully emptied when the *pop* port is help high for 5 clock cycles. The status flags show the FIFO going from full to empty again.

5 Simulation

5.1 Tests

The test bench generates a number (default is 50) configurations of the DynamicFifo that are highly randomized. There are two flavors of tests:

- Directed tests that fill the FIFO with random data and then read back the results to verify that the read data matches the writted data.
- Lengthy random tests that are used to check odd combinations of configurations and to compile code coverage data.

5.2 Code coverage

All inputs and outputs are checked to insure each toggle at least once. An error will be thrown in case any port fails to toggle.

The only exception are the *almostEmptyLevel* and *almostFullLevel* which are intended to be static during each simulation. These signals are excluded from coverage checks.

5.3 Running simulation

Simulations can be run directly from the command prompt as follows:

```
$ sbt "test"
```

or from make as follows:

```
$ make test
```

6 Synthesis

6.1 Area

The DynamicFifo has been tested in a number of configurations and the following results should be representative of what a user should see in their own technology.

Config Name	externalRAM	dataWidth	fifoDepth	Gates
small_false_8_8	false	8	8	769
medium_false_32_64	false	32	64	19,283
large_false_64_256	false	64	256	152,808
small_true_64_256	true	64	256	355
medium_true_128_128	true	128	128	477
large_true_256_2048	true	256	2048	502

Table 3: Synthesis results

6.2 SDC File

An `.sdc` file is generated to provide synthesis and static timing analysis tools guidance for synthesis.

The `DynamicFifo.sdc` file is emitted and found in the `./syn` directory.

6.3 Timing

The following timing was extracted using the generated .sdc files using the Nangate 45nm free library.

Config Name	Period	Duty Cycle	Input Delay	Output Delay	Slack
small_false_8_8	5ns	50%	20%	20%	2.93 (MET)
medium_false_32_64	5ns	50%	20%	20%	2.69 (MET)
large_false_64_256	5ns	50%	20%	20%	2.80 (MET)
small_true_64_256	5ns	50%	20%	20%	2.80 (MET)
medium_true_128_128	5ns	50%	20%	20%	2.70 (MET)
large_true_256_2048	5ns	50%	20%	20%	2.77 (MET)

Table 4: Static Timing Analysis results

6.4 Multicycle Paths

None.