




# DynamicFifo Product User Guide

rocksavagetech.chiselWare.DynamicFifo

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

*Mr. Example*  Digitally signed by your common  
name here  
DN: your distinguished name here  
Reason: your signing reason here  
Location: your signing location here  
Date: 2010.03.17 17:42:22 -0700'

Warren Savage

November 25, 2023

## Contents

<b>1</b>	<b>Errata and Known Issues</b>	<b>3</b>
1.1	Errata . . . . .	3
1.2	Known Issues . . . . .	3
<b>2</b>	<b>Port Descriptions</b>	<b>4</b>
<b>3</b>	<b>Parameter Descriptions</b>	<b>5</b>
<b>4</b>	<b>Theory of Operations</b>	<b>6</b>
4.1	Introduction . . . . .	6
4.2	Interface Timing . . . . .	7
<b>5</b>	<b>Simulation</b>	<b>8</b>
<b>6</b>	<b>Synthesis</b>	<b>9</b>

# 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 four empty and full flags.

## 1.2 Known Issues

None.

## 2 Port Descriptions

The ports for **DynamicFifo** are shown below in Table1. 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	<i>dataWidth</i>	Input	Data to be pushed into the FIFO
dataOut	<i>dataWidth</i>	Output	Data popped from the FIFO
empty	1	Output	Indicates the FIFO is empty
full	1	Output	Indicates the FIFO is full
almostEmptyLevel	$\log_2\text{Ceil}(\text{fifoDepth})$	Input	Sets the threshold for the almostEmpty port. almostEmpty will be active when the FIFO is at or below this level.
almostFullLevel	$\log_2\text{Ceil}(\text{fifoDepth})$	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	$\log_2\text{Ceil}(\text{fifoDepth})$	Output	Write address to the external FIFO RAM
ramDataIn	<i>dataWidth</i>	Output	Data to the external FIFO RAM
ramReadEnable	1	Input	Read enable to the external FIFO RAM
ramReadAddress	$\log_2\text{Ceil}(\text{fifoDepth})$	Output	Read address to the external FIFO RAM
ramDataOut	<i>dataWidth</i>	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	$\geq 1$	The data width of the FIFO
fifoDepth	Int	2	$\geq 2$	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.

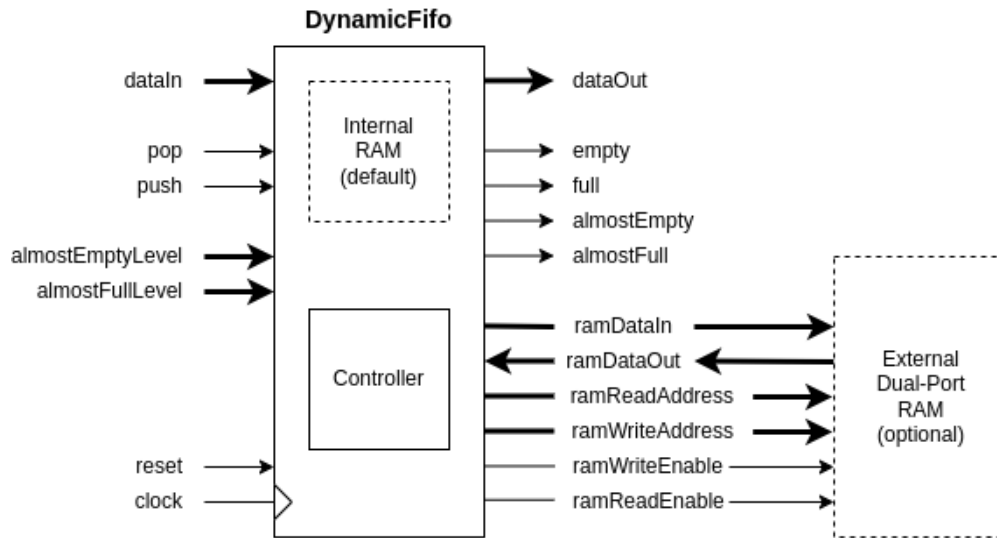


Figure 1: Block Diagram

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 enqueued on the next rising edge of *clock*. When *pop* is asserted, the top of the FIFO is dequeued on the next rising edge of *clock*. Pop and Push operations can be simultaneous.

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 enqueued.

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)
```

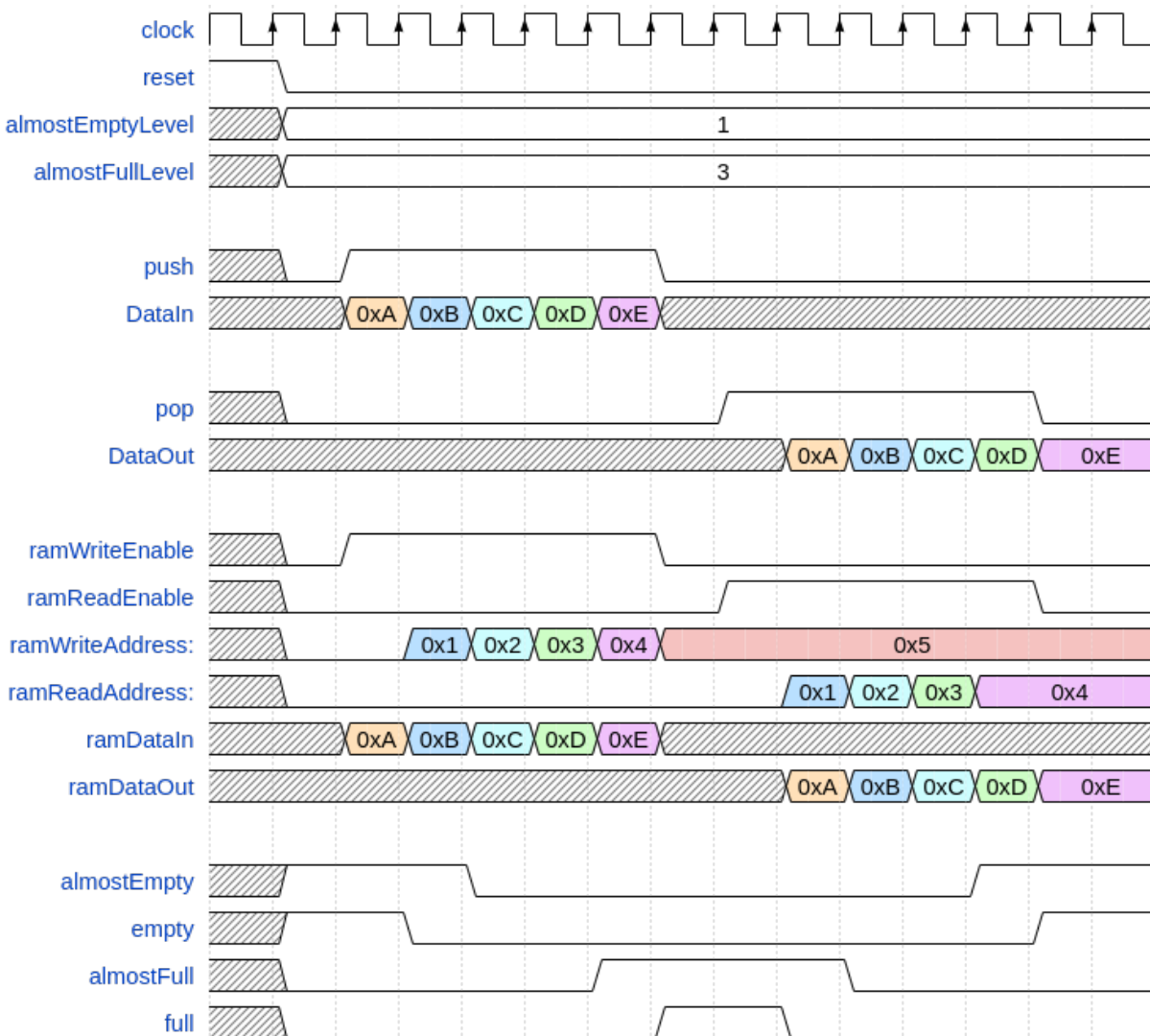


Figure 2: Timing Diagram

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 held high for 5 clock cycles. The status flags show the FIFO going from full to empty again.

## 5 Simulation

This section is under construction and will be completed as the code is completed. But will include the following subsections:

- How to configure test operations
- How to run tests
- List of tests with description of each test
- Code coverage results



## 6 Synthesis

This section is under construction and will be completed as the code is completed. But will include the following subsections:

- Explanation of the .sdc file
- List of false or multicycle paths
- Expected synthesis results