

SILICONOVA

**SYNCHRONOUS FIFO**

Contents

[CHAPTER 01 1](#_Toc168699803)

[Introduction 1](#_Toc168699804)

[1.1 Design Goals and Requirements 1](#_Toc168699805)

[1.2 Synchronous FIFO 1](#_Toc168699806)

[1.3 Operations in the Synchronous FIFO 1](#_Toc168699807)

[1.4 Pointers to Control Operations 1](#_Toc168699808)

[1.4 Flags in FIFO 2](#_Toc168699809)

[CHAPTER 02 3](#_Toc168699810)

[Module Description 3](#_Toc168699811)

[2.1 Synchronous FIFO Signals 3](#_Toc168699812)

[2.2 Parameters in Design 4](#_Toc168699813)

[2.3 FIFO Module Diagram 4](#_Toc168699814)

[2.4 Description of the Diagram 5](#_Toc168699815)

[2.5 Signal Connections: 5](#_Toc168699816)

# CHAPTER 01

## Introduction

### Design Goals and Requirements

* FIFO is able to write (push) and read (pop) into a given dual port memory
* Depth of the FIFO is configurable
* FIFO has empty and full flags to indicate empty FIFO and full FIFO respectively
* When written to a full FIFO , or read from empty FIFO, another flag the error flag will be high

### Synchronous FIFO

A Synchronous FIFO is a First-In-First-Out queue in which there is a single clock pulse for both data write and data read.[In Synchronous FIFO the read and write operations are performed at the same rate.](https://www.amazon.in/shop/semiconductorclub) The number of rows is called depth or number of words of FIFO and number of bits in each row is called as width or word length of FIFO. This kind of FIFO is termed as Synchronous because the rate of read and write operations are same. Basically Synchronous FIFO are used for High speed systems because of their high operating speed. Synchronous FIFO are easier to handle at high speed because they use free running clocks whereas in case of Asynchronous FIFO they uses two different clocks for read and write. Synchronous FIFO is more complex then the Asynchronous FIFO.

### Operations in the Synchronous FIFO

**Push Operation**: The operation involves in pushing or storing the data in to the FIFO memory till it rises any flag conditions for not to write anymore.

**Pop Operation**: Pop operation performed when we want to get data out from the FIFO memory until it informs there is no more data to be read from the memory, the condition called empty condition. Empty conditions are generated using empty flags.

### Pointers to Control Operations

**Write Pointer**: This pointer controls the push operation of the FIFO. It used to point to the FIFO memory location where the data will be written.

**Read Pointer**: The read operation is controlled by the read pointer. It will be pointing the location from where next data is to be read.

### Flags in FIFO

Synchronous FIFO provides us with few flags, [to determine the status or to interrupt the operation of FIFO.](https://www.amazon.in/shop/semiconductorclub)

**EMPTY flag**: This flags is useful to avoid the case of the invalid request of read operation when the FIFO is already empty.

**FULL flag**: This flags is useful to avoid the case of the invalid request of write operation when the FIFO is already full.

**ERROR flag**: This flag is high when write operation is done on a full FIFO. Reading data when the flag is high will give erroneous data.

# CHAPTER 02

## Module Description

### 2.1 Synchronous FIFO Signals

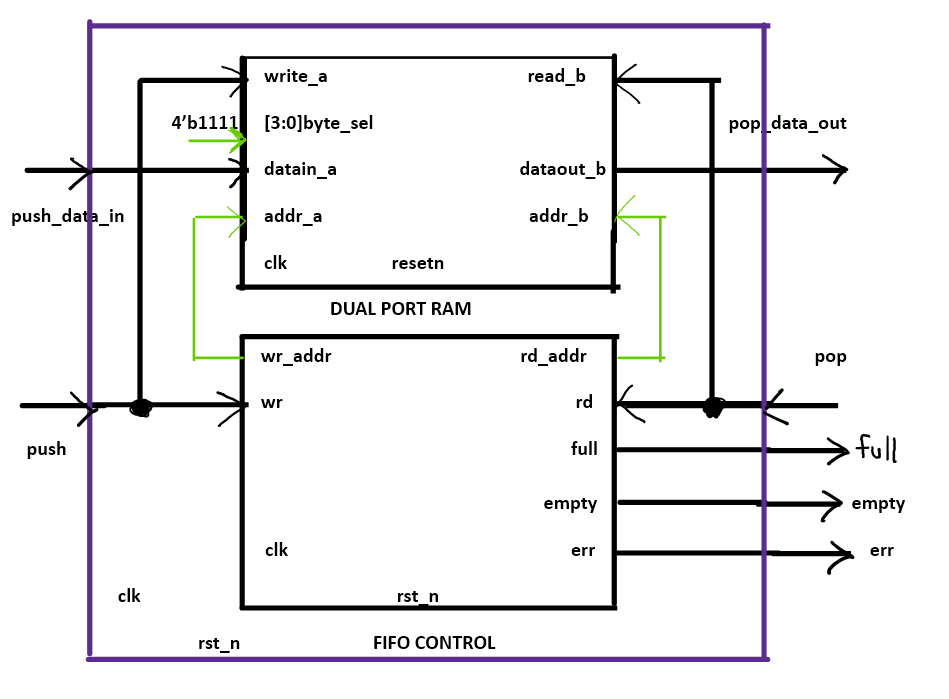
|  |  |  |  |
| --- | --- | --- | --- |
| **Signal** | **Source** | **Width** | **Description** |
|  |  |  |  |
| **clk** | Clock | 1 | clock signal  **clk** is clock signal.All FIFO signals are timed against the rising edge of **clk** |
| **rst\_n** | System reset | 1 | reset signal  **rst\_n** is active LOW reset  Directly connected to the system bus reset |
| **push** | Write request | 1 | write control signal  **push** indicates write access when HIGH |
| **push\_data\_in** | Write data | dataWidth | write data  **push\_data\_in** write data bus is driven when **push** is HIGH and **pop** is LOW  **push\_data\_in** can be 8, 16, 32, 64 bits wide |
| **pop** | Read request | 1 | read control signal  **rd** indicates read access when HIGH |
| **pop\_data\_out** | Read data | dataWidth | read data  **pop\_data\_out** read data bus is driven when **pop** is HIGH and **push** is LOW  **pop\_data\_out** can be 8, 16, 32, 64 bits wide |
| **empty** | Empty flag | 1 | **empty** flag is HIGH when FIFO is empty |
| **full** | Full flag | 1 | **full** flag is HIGH when FIFO is full |
| **err** | Error flag | 1 | **err** flag is HIGH when write cycle is performed on FIFO when **full** flag is HIGH |

### 2.2 Parameters in Design

|  |  |
| --- | --- |
| **Parameters** | **Description** |
| DEPTH | Depth of the FIFO |
| ADDR\_WIDTH | Width of the address bus, calculated using $clog2(DEPTH). |
| DATA\_WIDTH | Width of the data bus |

### 2.3 FIFO Module Diagram





**Figure: 2.1 Block diagram of FIFO module**

### 2.4 Description of the Diagram

1. **FIFO Module**:
   * Contains the overall FIFO functionality.
   * Instantiates the dual\_port\_ram and fifo\_cntrl submodules.
   * Connects global signals (clk, rst\_n), control signals (push, pop), data signals (push\_data\_in, pop\_data\_out), and status flags (empty, full, err).
2. **FIFO Control Logic Submodule**:
   * Manages the read and write pointers.
   * Generates status flags (empty, full, err).
   * Provides read and write addresses (r\_addr, w\_addr) for the dual\_port\_ram.
3. **Dual-Port RAM Submodule**:
   * Provides storage for the FIFO.
   * Allows concurrent read and write operations.
   * Receives write enable (write\_a) and read enable (read\_b) signals.
   * Uses byte select (byte\_sel) to enable specific bytes.
   * Connects to fifo\_cntrl for read and write addresses and data input/output.

### 2.5 Signal Connections:

1. **Global Signals**:
   * clk and rst\_n are connected to clock and reset of both dual\_port\_ram and fifo\_cntrl.
2. **Control Signals**:
   * push and pop are connected to fifo\_cntrl.
3. **Data Signals**:
   * push\_data\_in is connected to dual\_port\_ram for writing data.
   * pop\_data\_out is connected to dual\_port\_ram for reading data.
4. **Status Flags**:
   * empty, full, and err are outputs from fifo\_cntrl.
5. **Internal Signals**:
   * wr\_w (write enable) is generated based on push and full\_w (internal full flag).
   * rd\_w (read enable) is generated based on pop and empty.
   * full is directly assigned from full\_w.

This block diagram and description provides a comprehensive overview of the design and functionality of the fifo module and its submodules.