High Level Design Specification (HLDS)

for

Asynchronous FIFO

Version 1.1

ECE-593: Fundamentals of Pre-Silicon Validation – Venkatesh Patil

03 February 2024

Prepared by:

Snigdha Reddy Baddam

Venkata Ramana Molabanti

Sri Hari Anne

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Sri Hari Anne	15 Jan 24	Initial Version	1.0
Team 7	2 Feb 24	Changes for Project Milestone 1	1.1

1. Introduction

1.1 Purpose

This document is the High-Level Design Specifications (HLDS) for the Asynchronous FIFO. This document summarizes the high-level design specifications and the logic design of different submodules and features, and Verification plan including testbench style and test case scenarios.

1.2 Document Conventions

The following conventions are used in this document:

- `Courier New Font` - Shows the contents of code files or any transcript output, and used when we refer to any keywords that appear in the code.
- **Bold** - Indicates all Heading of sections and subsections in the document.
- *Italic* - Used to indicate file names, directory names, and command-line code, if any.
- Hyperlinks - Used to indicate hyperlinks to any URLs or references.

1.3 Intended Audience and Reading Suggestions

This HLDS document is primarily targeted for Logic design and Verification engineers who are using the Asynchronous FIFO in their designs. An understanding of how multiple clock domains interact with each other is a prerequisite to reading this document.

1.4 Product Scope

Asynchronous FIFOs are used to transfer data from one clock domain to another clock domain. i.e., data items are written to FIFO from a producer module running at one frequency, and data items are read from FIFO by the consumer module running at a different clock frequency. Since the two frequencies are different, we call this FIFO as an Asynchronous FIFO.

Applications include rate-matching video interfaces, communicating to off-chip components (typically running at different clock speeds such as network cards, PCIE peripherals etc.), bulk data transfer/DMA across a chip.

1.5 References

[1] Clifford E. Cummings and Peter Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, March 2002, Section TB2, 3rd paper. Also available at www.sunburst-design.com/papers

[2] Clifford E. Cummings, "Simulation and Synthesis Techniques for Asynchronous FIFO Design," *SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers*, March 2002, Section TB2, 2nd paper. Also available at www.sunburst-design.com/papers

[3] Putta Satish, "CALCULATION OF FIFO DEPTH MADE EASY"
Available at https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdf

[4] FIFO Architecture, Functions, and Applications by Texas Instruments. Available at https://www.ti.com/lit/an/scaa042a/scaa042a.pdf

[5] W. J. Dally EE 108a Lecture 13: Metastability And Synchronization Failure (Or When Good Flip-Flops Go Bad), *Stanford University Lectures.* Also available at *http://cva.stanford.edu/people/davidbbs/classes/ee108a/winter0607%20labs/lect.9.Metastability-blackschaffer.ppt*

2. Overall Description

2.1 Product Perspective

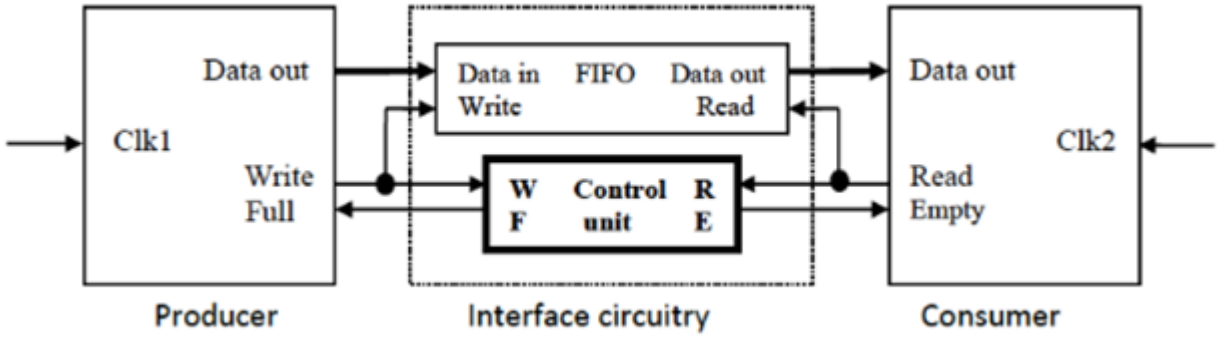
A FIFO primarily consists of a set of read and write pointers, storage elements to store data items, and control logic to ensure the functionality is as per requirement. This also includes signals such as empty and full to stop writing or reading of data items when the FIFO is full/empty.

The first known FIFO was implemented by Peter Alfke in 1969. He is also the co-author of one of the references to this document [1].

There are two types of FIFOs that are primarily used. They are:
1. Synchronous FIFO
2. Asynchronous FIFO

A synchronous FIFO is a FIFO where the same clock is used for both reading and writing. An asynchronous FIFO uses different clocks for reading and writing and they can introduce metastability issues.

The top-level block diagram of an Asynchronous FIFO is given below:



2.2 Product Functions

Following are the major functions of the system:

- **Data transfer between producer and consumer:** The FIFO is used to safely transfer data between asynchronous clock domains.
- **Full and Empty signals:** Inclusion of two signals to detect FIFO full and empty states.
- **Almost full and almost empty signals:** Inclusion of two signals to detect FIFO almost full and almost empty states.
- **Gray Code Counters to manage FIFO read and write pointers:** Usage of Gray code counters for read and write pointers to avoid metastable conditions. We can also use binary to gray and gray to binary converters and replace the gray code counters with binary counters. A decision on this will be made during the design stage.
- **Scalability and reusability:** Ensuring that the FIFO architecture can be scaled to different capacities.

2.3 User Classes and Characteristics

Various users may find this useful for integration into their designs, They include:

- **ASIC/FPGA/SoC Design Engineers:**

Design Engineers working on systems with multiple clock domains benefit from HLDS as it facilitates safe data transfer across these domains. This design ensures that data is properly synchronized between different clock regions, preventing metastability issues. This design can also be integrated as an IP block within larger SoCs, providing a solution for handling asynchronous data transfers.

- **Verification Engineers:**

Verification engineers may find the verification strategy and test case scenarios useful to reuse in some of their own systems of similar applications.

2.4 Tools and Software

We use QuestaSim from Mentor Graphics as the primary tool for running simulation.

2.5 Design and Implementation Constraints

It is mentioned that the FIFO should be able operate under following conditions:

- Producer clk1 frequency = 500 MHz
- Consumer clk2 frequency = 500MHz
- With duty-cycle of 50%
- Max write Burst size = 450
- No. of ideal cycle between successive writes = 0
- No. of ideal cycles between successive reads = 4
- The no. of idle cycles between two successive writes is 0 clock cycles. It means that, after writing one data item, the producer will initiate writing in the next cycle without any delay. So, it can be understood that for every 1 clock cycle one data is written.
- The no. of idle cycles between two successive reads is 4 clock cycles. It means that, after reading one data item, the consumer waits for 4 clock cycles before initiating the next read. So, it can be understood that for every five clock cycles, one data is read.
- Time required to write one data item = $1 * 1/500\text{MHz} = 2$ nSec.
- Time required to write all the data in the burst = 450 * 2 nSec. = 900 nSec.
- Time required to read one data item = $5 * 1/500\text{MHz} = 10$ nSec.
- So, for every 10 nSec, the consumer is going to read one data item in the burst.
- So, in a period of 900 nSec, 450 no. of data items can be written.
- The no. of data items can be read in a period of 900 nSec = 900 nSec/10 nSec = 90
- The remaining no. of bytes to be stored in the FIFO = 450-90 = 360.
- So, the FIFO must be capable of storing 360 data items.

So, the minimum depth of the FIFO should be 360.

2.6 Assumptions and Dependencies

The following were the input and output clock values, idle cycles, and burst size that were used to calculate the depth of the FIFO.

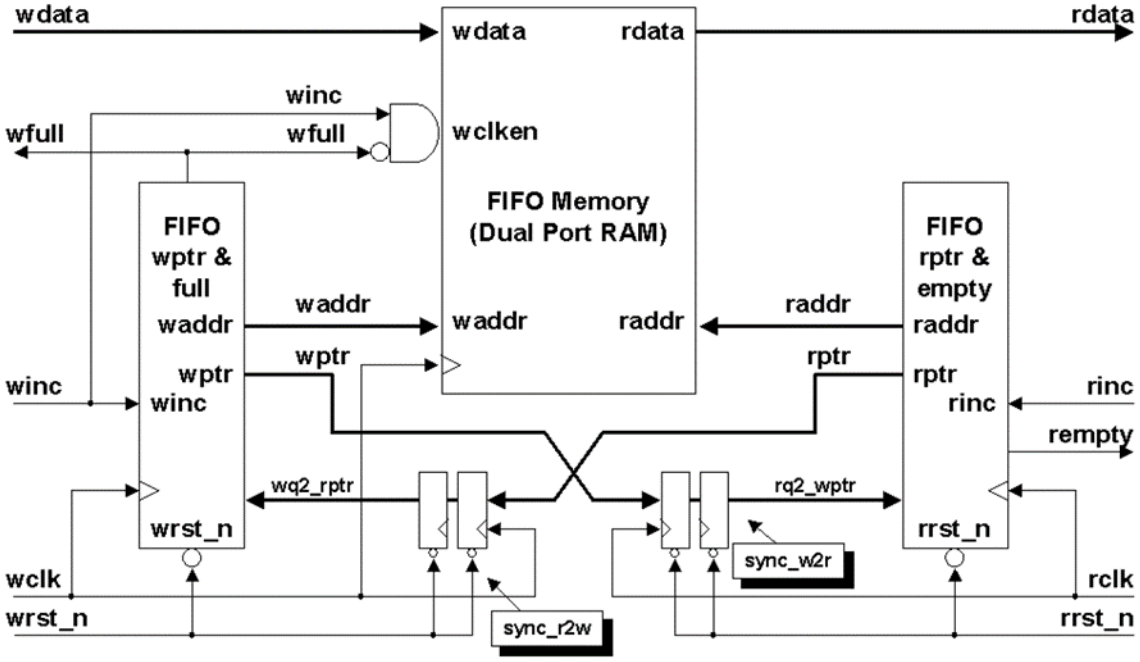
- Producer clk1 frequency = 500 MHz
- Consumer clk2 frequency = 500MHz
- With duty-cycle of 50%
- Max write Burst size = 450
- No. of ideal cycle between successive writes = 0
- No. of ideal cycles between successive reads = 4

It is worth mentioning that the FIFO depth will depend on all of the above values.
Assumptions: Data width = 8. Depth = 512.

3. External Interface Requirements

3.1 Hardware Interfaces

The detailed block diagram of the FIFO is given below. (refer [2])



rdata : read data

waddr : binary write pointer

wptr : gray write pointer

raddr : binary read pointer

rptr : gray read pointer

wq2_rptr : read pointer synchronized

wq2_wptr : write pointer synchronized

The following are the major modules in the block diagram:

FIFO Memory Module: This module is responsible for storing the data items.

Control Unit: This module contains the 2-FF synchronizers used for read and write pointers, FIFO wptr & full logic, FIFO rptr & empty logic, and responsible for generating control signals such as empty, full etc.

Gray counter: Gray counters are used to eliminate the possibility of error during transition from one counter value to the next counter value. This is because only one bit changes at a time.

4. Product Features

4.1 FIFO memory

The FIFO memory module has read and write enable signals, read and write pointers, data in and data out lines and the actual memory element connected to both the pointers. As the name implies, the data item that has been written first will be the first one to be read out.
Note: Timing diagrams will be added later during the design stage.

4.2 Gray Counter

Gray Code Counters are used to manage FIFO read and write pointers. This is done to avoid metastable conditions. In the Gray code number system, two consecutive values differ only by one bit. This single-bit change feature is critical in reducing errors, especially when many bits are updated at the same time. Gray code counters are used to track the read and write pointers in FIFO because they considerably reduce the possibility of erroneous data read/write operations caused by the asynchronous update of multiple bits.

Alternatively, we can use binary counters along with binary to gray converters which will make the calculation of full, almost full, empty and almost empty easier. A decision on this will be made during the design stage.

4.3 Synchroniser

In the case of an asynchronous FIFO, the write pointer is aligned to the write clock domain and the read pointer is aligned to the read clock domain. Hence, it requires domain crossing to calculate FIFO full and empty conditions. This in turn, causes metastability. In order to resolve this metastability issue, 2 flip flop synchronizers can be used to pass write and read pointers to calculate full and empty conditions.

We must note that a single "2 FF synchronizer" can resolve metastability for only one bit. Hence, depending on write and read pointers multiple 2FF synchronizers are required. (one 2 FF synchronizer for every bit of read and write pointers)

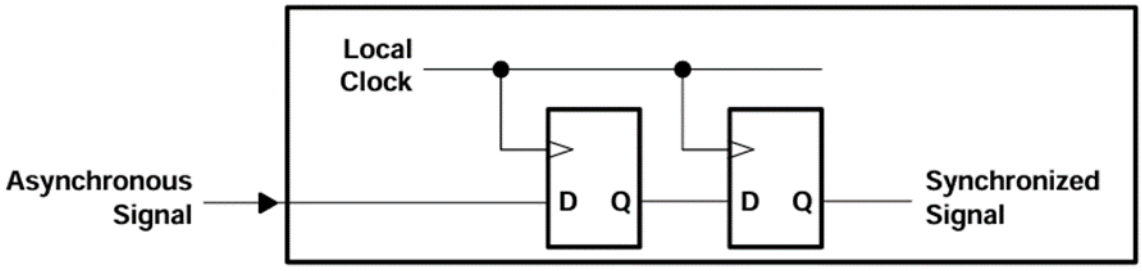


Fig.3 Block diagram of a 2 FF synchroniser

4.4 Status signals in Control block

Full
The Full (*wfull*) signal is an output signal from FIFO. It is 1-bit wide, active high signal. When the FIFO reaches the maximum depth of the buffer (defined by a parameter) then the *wfull* signal shall be active. A read will deassert this signal.

Almost Full
The Almost Full (*walmostfull*) signal is an output signal from FIFO. It is 1-bit wide, active high signal. When the number of entries in the FIFO reaches **or** is greater than the predefined value of ¾ or more of the maximum depth of the buffer, (defined by a parameter), then the *walmostfull* signal shall be active.

Empty
The Empty (*rempty*) signal is an output signal from FIFO. It is 1-bit wide, active high signal. When all the data has been read, then the *rempty* flag shall be active. A reset will also cause the *rempty* signal to be active. A write after a reset will deassert this signal.

Almost Empty
The Almost (*ralmostempty*) signal is an output signal from FIFO. It is 1-bit wide, active high signal. When the number of entries in the FIFO reaches or is less than the predefined value of ¼ or less of the maximum depth of the buffer, (defined by a parameter), then the *ralmostempty* flag shall be active.

4.5 Reset signals and functionality

The rrst_n and wrst_n are two reset signals, both are input signals to FIFO, both 1 bit wide and active low. The reset usually clears the pointers and the status flags. The reset_n is asynchronous to the system clock clk.

The empty flag is set when reset is asserted. The position of pointers are brought to initial values and the memory contents are also cleared.

4.6 Clock signals and functionality

The rclk and wclk are two clocks, both are input signals to FIFO, 1 bit wide and the rising edge is the active edge. The rclk is the clock for the read transactions, and wclk is the clock for write transactions, both active on the positive edge of the clock. The clock is at 50% duty cycle.

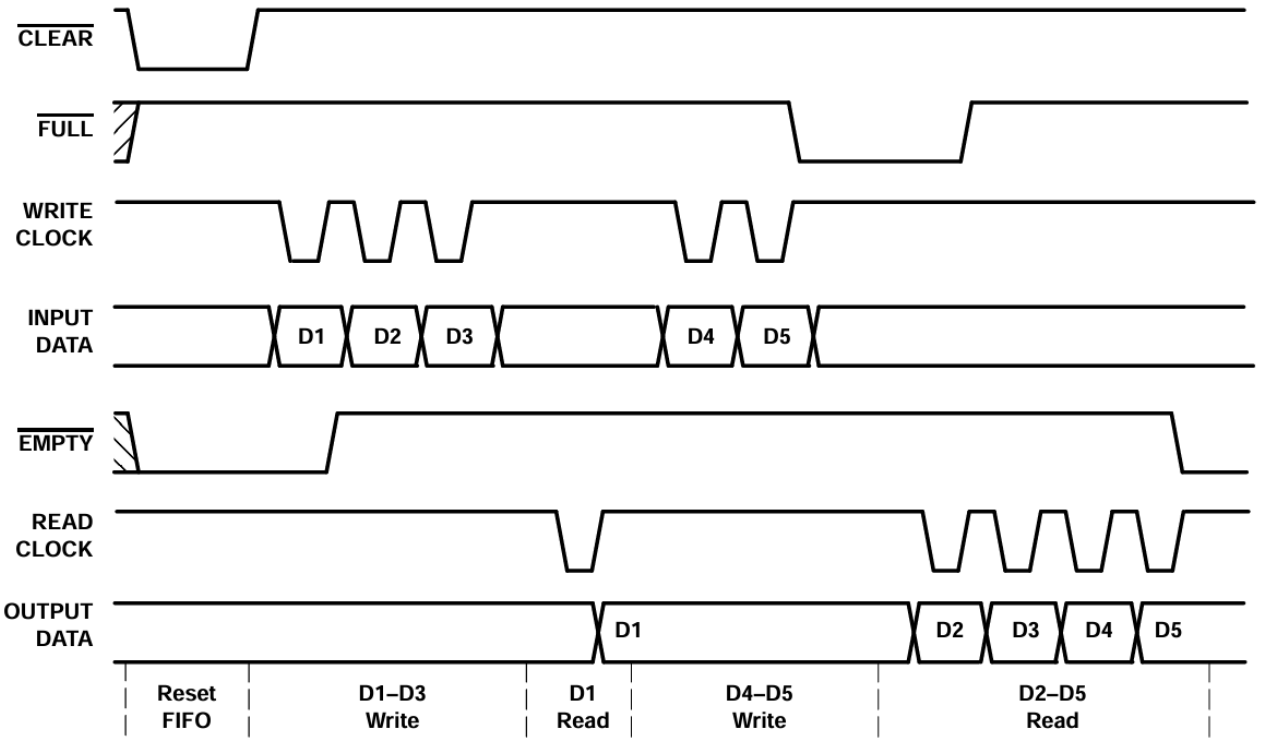


Fig.4 Timing diagram of an Asynchronous FIFO

5. Logic Design

5.1 Directory Structure

For Milestone 1, all the RTL files (including the design and the testbench and run.do) are in one directory. The file names are as following:

- *top.sv*
    - *fifo_mem.sv*
    - *sync_r2w.sv*
    - *sync_w2r.sv*
    - *full.sv*
    - *empty.sv*
- *tb.sv*
- *run.do*

5.2 Design modules

The modules are connected via logic signals for this milestone. THe use of interfaces is planed for the next milestones.

- *top.sv* - top module containing instantiations of all sub-modules.
    - *fifo_mem.sv* - module for fifo memory
    - *sync_r2w.sv* - module for synchronising read pointer to write clock domain.
    - *sync_w2r.sv* - module for synchronising write pointer to read clock domain
    - *full.sv* - full detection logic module
    - *empty.sv* - empty detection logic module
- *tb.sv* - testbench for milestone 1

5.3 SystemVerilog abstraction Features used.

The fixed values such as FIFO depth, Address width and FIFO data width have been parameterized. This enables scaling of design. All the parameters can be changed and the design and testbench will work for any depth and width.

Some SystemVerilog abstractions such as multidimensional arrays and tasks were also used.

For milestone 2, all the parameters will be added to a global package file.

5.4 Simulation, Tools, Directory Structure

For simulation, QuestaSim was used. The transcript will show if any assertion has failed. The waveform will show the testing for full, empty, reads and writes.

6. Verification

For Milestone 1, the verification is done on a basic level. The basic functions of an asynchronous FIFO, i.e., reads, writes, burst length, empty and full conditions are tested.

Assertions are used to indicate failures and some functions are verified from the waveforms.

6.1 Testbench Style

A simple sanity testbench is included for milestone 1. It verifies the functionality of FIFO.

6.2 Testing Strategies

- Directed testcases with assertions for verifying reset functionality.
- Randomized testcases for verifying concurrent reads and writes.
- Assertions for verifying empty and full status signals.

6.3 Test case scenarios

Basic Tests

Test Name / Number	Test Description/ Features
1.1.1 Read	Check basic read operation
1.1.2 Read and write	Verifying data write and read in the expected order with the same data

Complex Tests

Test Name / Number	Test Description/ Features
1.2.1	Concurrent events (R+W) Conditions: fifo_full/fifo_empty/always_full/always empty etc.
1.2.2 reset	Verifying reset conditions
1.2.3 extreme levels	Verifying fifo overflow and underflow levels
12.4 concurrent reset	Verifying concurrent reset conditions

Regression Tests (Must pass every time)

Test Name / Number	Test Description/Features
1.3.1	Verifying read and write operations without errors
1.3.2	Verifying fifo full and fifo empty conditions after read and write operations
1.3.3	Verifying that same data should be read whatever the data write is happened

Corner cases testcases

Test Name / Number	Test Description
1.4.1	Special Case testing tests and conditions
1.4.2	Bug injection and testing scenario
1.4.3	To detect stuck at 0 or stuck at 1 faults, write all 0's and 1's to all locations
1.4.4	setting almost empty and almost full at extreme levels

NOTE: Some test cases were not implemented for this milestone but will be part of class based and UVM testbenches.

6.4 Others

<N/A>

Summary

Appendix A: Glossary

<Nomenclature used in your document.>

Appendix B: FIFO depth calculation

It is mentioned that the FIFO should be able operate under following conditions:

- Producer clk1 frequency = 500 MHz
- Consumer clk2 frequency = 500MHz
- With duty-cycle of 50%
- Max write Burst size = 450
- No. of ideal cycle between successive writes = 0
- No. of ideal cycles between successive reads = 4

- The no. of idle cycles between two successive writes is 0 clock cycles. It means that, after writing one data item, the producer doesn't wait before initiating the next write. So, it can be understood that for every **1** clock cycle, one data is written.
- The no. of idle cycles between two successive reads is 4 clock cycles. It means that, after reading one data item, the consumer waits for 4 clock cycles before initiating the next read. So, it can be understood that for every **five** clock cycles, one data is read.
- Time required to write one data item = $1 * 1/500\text{MHz}$ = 2 nSec.
- Time required to write all the data in the burst = 450 * 2 nSec. = 900 nSec.
- Time required to read one data item = $5 * 1/500\text{MHz}$ = 10 nSec.
- So, for every 10 nSec, the consumer is going to read one data item in the burst.
- So, in a period of 900 nSec, 450 no. of data items can be written.
- The no. of data items can be read in a period of 900 nSec = 900 nSec/10 nSec = 90
- The remaining no. of bytes to be stored in the FIFO = 450-90 = 360.
- So, the FIFO must be capable of storing 360 data items.

So, the minimum depth of the FIFO should be 360.