VERIFICATION TEST PLAN

Fundamentals of Pre-Silicon Validation
Winter -2024

Project Name: Asynchronous FIFO

Members: Daniyal Ahmad, Fardeen Wasey, Adeel Ahmed

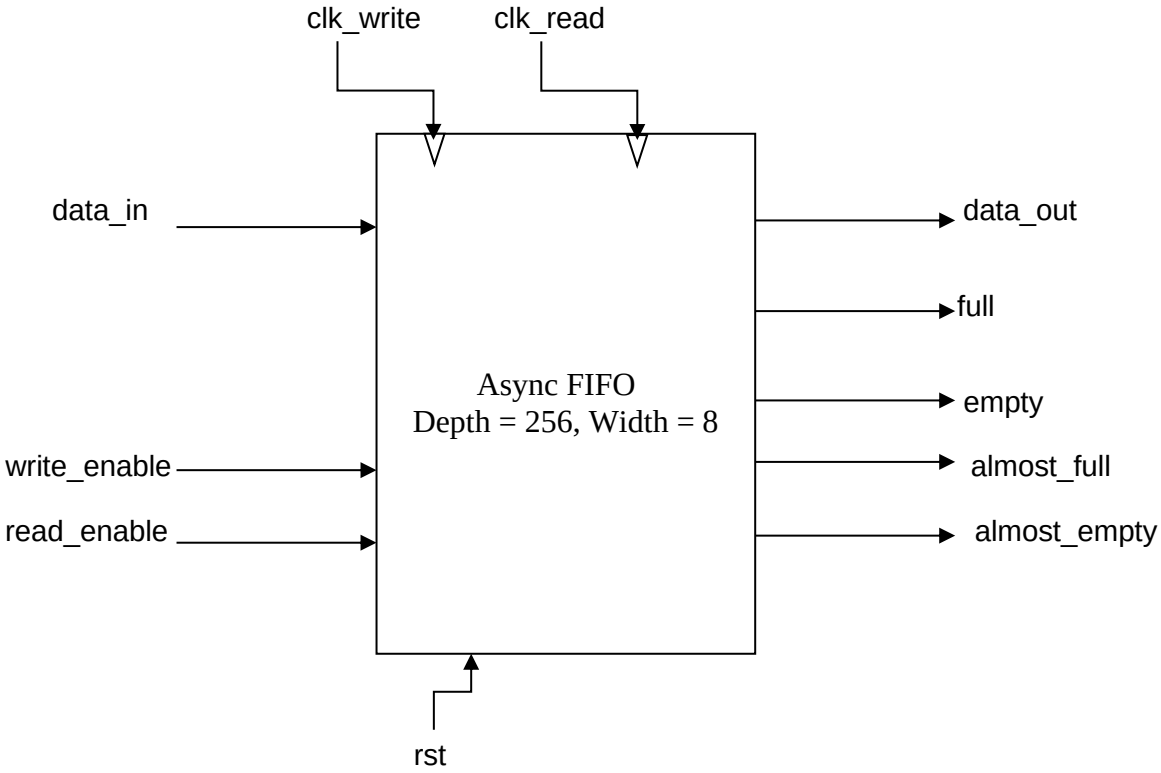
Date: 02/03/2024

Acknowledgement:

1 Table of Contents

2 Introduction:

2.1 This document describes hardware architecture and Implementation off the FIFO block. It basically to understand and validate the internal hardware implementation of the block, along with its configuration, integration and use within FIFO. It also provides guidance and information on implementation specific issues for functional verification as appropriate.



2.2

2.3 A FIFO (First In First Out) serves as a buffer for data transfer between two clock domains operating asynchronously or within the same clock domain but with varying throughputs. FIFOs are versatile solutions suitable for various applications. Different types of FIFOs exist, with synchronous and asynchronous FIFOs being particularly noteworthy. The choice between them hinges on the specific application. Synchronous FIFOs are generally preferred because they provide fully
synchronized flags, unlike asynchronous FIFOs. Various architectures can be employed to design FIFOs, but the most widely accepted and reliable one is the

static memory-based FIFO. In contrast, fall-through FIFOs lack benefits and have notable drawbacks compared to FIFOs with static memory.

3 Verification Requirements

3.1 There are basically different levels of verification.

3.1.1 The primary verification is Unit Level Verification. In which the basic functionality of the design is tested.

3.1.2 Interfacing signals:

For this version, we are directly providing stimulus into the module. We will be using interface while implanting UVM based testbench.

```
module AsyncFIFO (
    input logic clk_write,
    input logic clk_read,
    input logic rst,
    input logic [7:0] data_in,
    input logic write_enable,
    input logic read_enable,
    output logic [7:0] data_out,
    output logic full,
    output logic empty,
    output logic almost_full,
    output logic almost_empty
)
```

4 Required Tools

4.1 For this Verification process we are using Questasim to check the functionality of the design.

4.2 We are using VSCode which is mapped to our git repository, from that we can push and pull changes made by the team members.

5 Risks and Dependencies // will be covered in coming milestones.

5.1 List all the critical threats or any known risks. List contingency and mitigation plans.

6 Functions to be Verified. // will be covered in coming milestones.

6.1 Functions from specification and implementation

6.1.1

6.1.2 List of functions that will not be verified. Description of each function and why it will not be verified.

6.1.3 List of critical functions and non-critical functions for tapeout

7 Tests and Methods

7.1.1 Testing methods to be used: White Box

Black Box
Pros: Function-focused, straightforward, and unaffected by internal details.
Cons: May not detect some design faults; limited coverage of internal pathways and issues.

White Box:
Pros: Optimal test cases, early identification of internal problems, and comprehensive code coverage.
Cons: Time-consuming and requiring in-depth knowledge of internal implementation.

Gray Box:
Pros: more thorough than black-box, balanced approach, partly internal knowledge used.
Cons: It might not obtain complete code coverage and depends on the correctness and availability of incomplete information.

We simply chose white box because we have in-depth knowledge of the internal implementation and wish to guarantee complete code coverage.

7.1.2 To test the basic RTL functionality, a simple conventional directed testbench was developed with few test scenarios like:-

1. Read enable is high, write enable low and checking status of full, empty and dataout signals.
2. Read enable is high, write enable is high.
3. Read enable is high, write enable is low.
4. Reset condition.
5. Full/empty condition.
6. Almost Full/empty condition.

Portion of Output:

```
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=0,dataIn=00000000,dataOut=xxxxxxxx,EMPTY=x,FULL=x,almost_full=x,almost_empty=x
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,almost_full=0,almost_empty=0
```

```
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=0,RD=0,WR=0,wr_clk=0,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=1,RD=0,WR=0,wr_clk=1,Rst=1,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=0,RD=0,WR=1,wr_clk=0,Rst=0,dataIn=00000000,dataOut=00000000,EMPTY=1,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=1,RD=0,WR=1,wr_clk=1,Rst=0,dataIn=00000000,dataOut=00000000,EMPTY=0,FULL=0,al
most_full=0,almost_empty=1
#
rd_clk=0,RD=0,WR=1,wr_clk=0,Rst=0,dataIn=00000000,dataOut=00000000,EMPTY=0,FULL=0,al
most_full=0,almost_empty=1
#
rd_clk=1,RD=0,WR=1,wr_clk=1,Rst=0,dataIn=00000000,dataOut=00000000,EMPTY=0,FULL=0,al
most_full=0,almost_empty=0
#
rd_clk=0,RD=0,WR=1,wr_clk=0,Rst=0,dataIn=00000000,dataOut=00000000,EMPTY=0,FULL=0,al
most_full=0,almost_empty=0
```

#
rd_clk=1,RD=0,WR=1,wr_clk=1,Rst=0,dataIn=00000000,dataOut=00000000,EMPTY=0,FULL=0,almost_full=0,almost_empty=0

7.1.4 Verification Strategy: (Dynamic Simulation, Formal Simulation, Emulation etc.) Describe why you chose the strategy.

7.1.5 What is your driving methodology?

7.1.5.1 List the test generation methods (Directed test, constrained random)

7.1.6 What will be your checking methodology?

7.1.6.1 From specification, from implementation, from context, from architecture etc

7.1.7 Testcase Scenarios (Matrix)

7.1.7.1 Basic Tests

Test Name / Number	Test Description/ Features
1.1.1Read and write	Check basic read and write operation
1.1.2Reset	Checks for reset

7.1.7.2 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	

7.1.7.3 Regression Tests (Must pass every time)

Test Name / Number	Test Description/Features
1.3.1	Tests that should always pass
1.3.2	

7.1.7.4 Any special or 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

8 Coverage Requirements // will be covered in coming milestones.

8.1.1.1 Describe Code and Functional Coverage goals for the DUV

8.1.1.2 Formulate conditions of how you will achieve the goals. Explain the Covergroups and Coverpoints and your selection of bins.

8.1.2 Assertions

8.1.2.1 Describe the assertions that you are planning to use and how it will help you improve the overall coverage and functional aspects of the design.

9 Resources requirements

9.1 Daniyal Ahmad:- Currently Working on the design. and keeping the track of the verification flow.

Fardeen Wasey:- Working on the conventional testbench to check the basic functionality of the RTL.

Adeel Ahmed:- Working on the conventional testbench. Making sure that there are no compilation errors.

10 Schedule

10.1 Create a table with a plan of completion. You can use milestones as a guide to fill this.

11 References Uses / Citations/Acknowledgements

1. http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdfLinks to an external site.
2. http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO2.pdfLinks to an external site.
3. https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdfLinks to an external site.
4. https://github.com/raysalemi/uvmprimerLinks to an external site.
5. https://www.chipverify.com/verification/constraint-random-verification

NOTE: This document is in its basic version and will be updated accordingly at each milestone and will be made as a proper report.