VERIFICATION TEST PLAN

Fundamentals of Pre-Silicon Validation || Winter -2024

Project Name:

Asynchronous FIFO

Project Team:

Asritha Manju Immadishetty - asritha@pdx.edu

Vijaya Manikanta Kotagiri – kotagiri@pdx.edu

Bhavani Gunda - gunda@pdx.edu

Date: 3/6/2024

Acknowledgement:

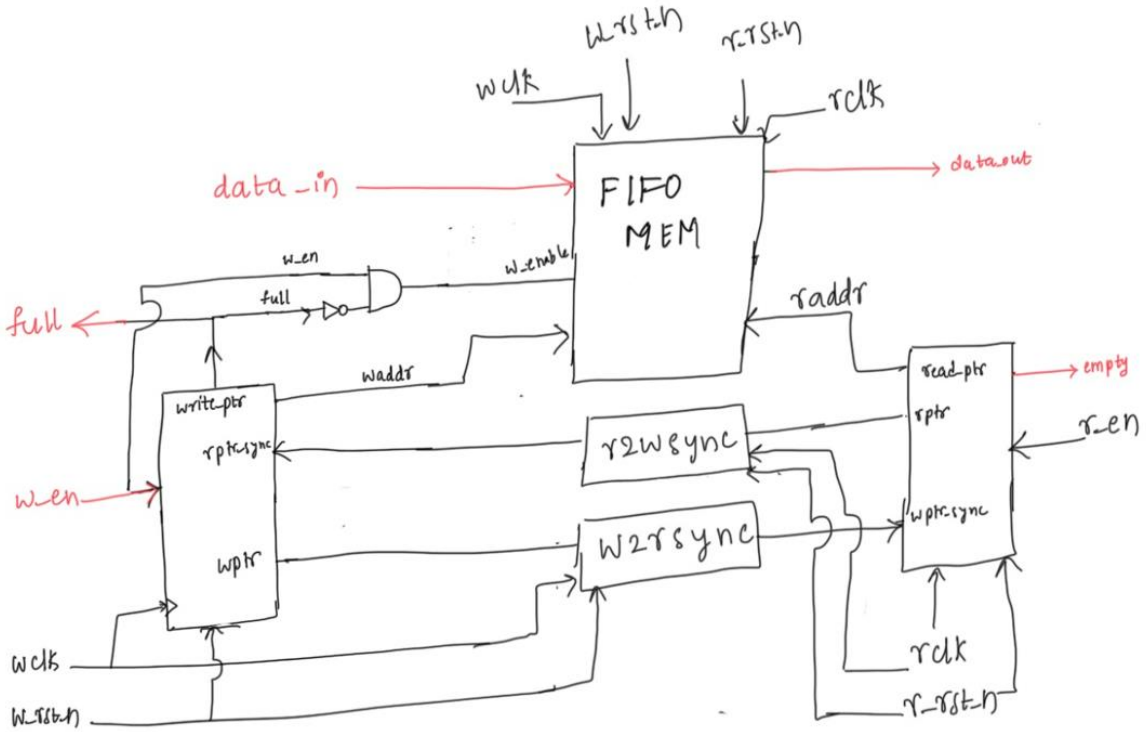
INDEX

2 Introduction:

2.1 Objective of the verification plan

The main objective of this verification plan is to thoroughly verify the design functionality and correctness through various verification methods and implementing error correction mechanisms and checking for errors.

2.2 Top Level block diagram



2.3 Specifications for the design

The given specifications of the Design:

- Writing to FIFO frequency (*Producer Frequency*) – 250 Mhz
- Reading from FIFO frequency (*Consumer Frequency*) - 100Mhz
- **Write Idle cycles** - 0
- **Read Idle cycles** - 2 *(Not included in the design functionality)*
- **Brust length** -150
- The calculated **FIFO DEPTH** - 130 (*submitted in a separate folder*)

3 Verification Requirements

3.1 Verification Levels

3.1.1 What hierarchy level are you verifying and why?

We are verifying the Top level of design hierarchy using System Verilog and UVM verification environment because it is reusable and easy to use and modify. This involves verifying the interactions between all subsystems, interfaces with external components, and ensuring overall system functionality and performance meet the design requirements.

3.1.2 How is the controllability and observability at the level you are verifying?

The controllability and observability decrease as we go higher in the design hierarchy, and we found some difficulties in finding the bugs.

3.1.3 Are the interfaces and specifications clearly defined at the level you are verifying. List them.

The interfaces and the specifications are well articulated in the design and exhibit the correct working functionality and are clearly defined in the level we are verifying. Implemented clocking blocks for monitor and driver and modports for coverage but actually removed because of some issues.

4 Required Tools

4.1 List of required software and hardware toolsets needed.

Required software- Siemens Questasim,

Notepad++ (we have used it, but not required)

No hardware tools are used.

4.2 Directory structure of your runs, what computer resources you will be using.

I am using the run.do file for both compilation and simulation and computer resources we are using is the remote lab in PSU.

5 Risks and Dependencies

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

Critical threats that result in hanging Questasim in remote desktop suddenly.

6 Functions to be Verified.

6.1 Functions from specification and implementation

6.1.1 List of functions that will be verified. Description of each function

Reset Behavior, only write, Only read, read and write simultaneously.

Back-to-back reads and writes, checking for FIFO full and empty conditions.

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

FIFO full and empty conditions in metastable state. By using synchronizers and gray code converters we can avoid putting full and empty in metastable state more likely.

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

Checking for all zero's and all one's and alternative zeros and one's as input data-critical functions.

Checking for all immediate values up to the FIFO depth when reset is asserted.

7 Tests and Methods

7.1.1 **Testing methods to be used: Black/White/Gray Box.**

We have chosen our testing method as white box testing.

7.1.2 State the PROs and CONs for each and why you selected the method for this DUV.

Every testing method has pros and cons and the reason for choosing the white box testing for the Design under verification is because it ensures all the parts of design are tested but it is time consuming process and we know the inputs of the design.

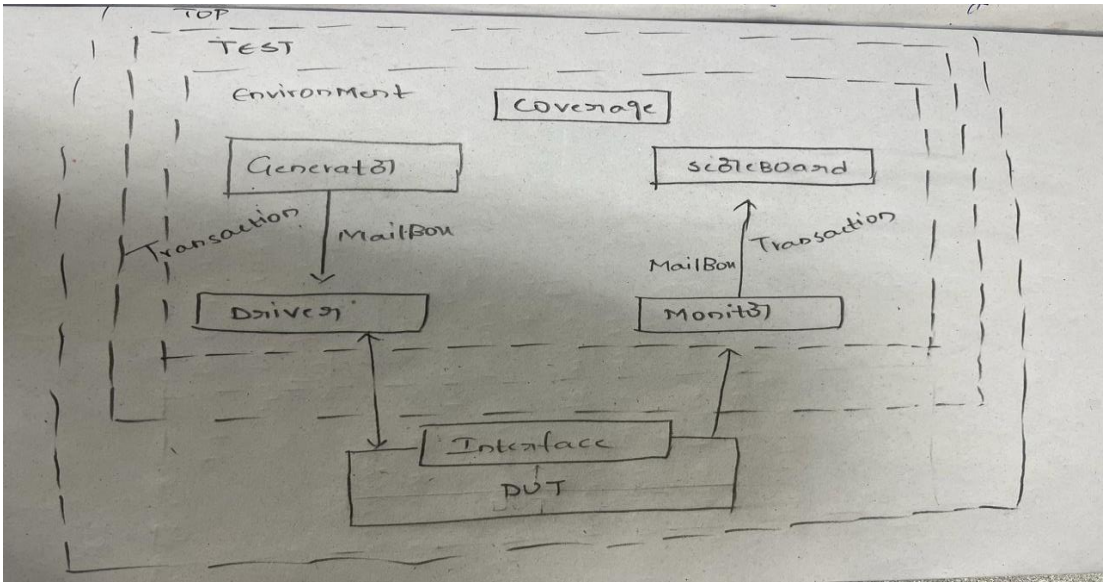
Black Box Testing: It enables faster testing and better simulation of real world scenarios but it lacks the technical insights.

Gray Box Testing: It provides early detection of inputs and Improved accuracy but it is time consuming and has some security concerns.

7.1.3 Testbench Architecture:

Component used (list and describe Drivers, Monitors, scoreboards, checkers etc.)

Include general testbench architecture diagram and how it relates to your design



7.1.4 Verification Strategy:
(Dynamic Simulation, Formal Simulation, Emulation etc.)

7.1.4.1 Describe why you chose the strategy.

Our verification strategy involves Direct Simulation, class-based verification, constraint random verification, UVM based verification and writing assertions and coverage driven verification is included as well in the design.

7.1.5 What is your driving methodology?

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

Starts with Directed test and generating random values, constraint random cases, checking the functional and code coverage using coverages. Verifying using generator, driver, monitors and scoreboard through system Verilog and uvm based verification(slightly different from system Verilog verification environment).

7.1.6 What will be your checking methodology?

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

My checking methodology will be mostly from the specification and the DUT.

7.1.7 Testcase Scenarios (Matrix)

7.1.7.1 Basic Tests

Test Name / Number	Test Description/ Features
1.1.1	Check basic read operation
1.1.2	Check basic write operation

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	Check errors in data_out compared with data_in

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	Tests that should never pass

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

8.1.1.1 **Describe Code and Functional Coverage goals for the DUV.**

Tested the coverage graph verifying the functionality of the design using covergroups, cross cover, bins, coverpoints and verified the coverage for different conditions like read and write with idle cycles and checking the functional behavior of full and empty conditions.

Checking the code and functional coverage of the entire asynchronous FIFO with all the write and read conditions for all the bursts and etc..

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

My selection of covergroups is two one for read and other for write and have different covergroups for different conditions for ex: raed_after writes, continues two reads, continues two_writes, read_after empty, write after full and etc.

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 Team members and who is doing what and expertise.

- **Bhavani Gunda** - Implemented write functionality with 0 idle cycles as per the specifications and designed FIFO memory and thoroughly tested them and written the verification plan document and written the generator and monitor and tested the correct working functionality. Implemented monitor, Write Sequence, Sequencer,Sequence Item successfully.
- **Asritha Manju Immadishetty** - Implemented the read functionality with 0 idle cycles as of now and thoroughly verified the design with some test cases and written the HLDS Document and written the driver and scoreboard functionality and tested it as well without any errors. Implemented scoreboard,driver, read sequence , agent successfully.
- **Vijaya Manikanta Kotagiri** - Implemented the design through interfaces and tested the very basic functionality only because of improper connection of signals with top module, calculated and written the depth of the FIFO and written the scoreboard and verified the percentage thoroughly . Implemented Environment, Test, TestBench Top successfully.

10 Schedule

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

Schedule		
3rd February	**Milestone 1**	- Create HLDS, Create Design Code, Initial Verification Plan Draft - Create a conventional testbench to check basic functioning of RTL. - Create a run.do file
11th February	**Milestone 2**	- Create and run Interface. - Create SV verification environment (Transaction, Generator, Driver) - Burst should be created and displayed its length (minimum 30 randomized bursts in your transcript)
17th February	**Milestone 3**	- Along with the above modified changes update the verification plan. - Create a Coverage report (Functional & Code Coverage) - 'do' or 'make' file for simulations. - Description of the Hierarchy
25th February	**Milestone 4**	- Create UVM verification plan. - Create a UVM testbench implementation document. - Test cases in the UVM testbench. - Create and pass burst from Sequence->Sequencer->Driver
3rd March	**Milestone 5**	- Complete the UVM Environment - Environment contains Agent, scoreboard, test, env. - Update the Verification Plan.

11 References Uses / Citations/Acknowledgements

https://vlsiverify.com/verilog/verilog-codes/asynchronous-fifo/

I acknowledged the source code in this reference link and implemented using System Verilog interfaces and throughly tested the design with and without interfaces as well but considered using interfaces as it is essential for system Verilog and uvm based verification methodology.

https://hardwaregeeksblog.files.wordpress.com/2016/12/fifodepthcalculationmadeeasy2.pdf

http://www.sunburst-design.com/papers/CummingsSNUG2002SJ_FIFO1.pdf