VERIFICATION TEST PLAN

For

Asynchronous FIFO

Fundamentals of Pre-Silicon Validation  
Winter -2024

By

**Abram Fouts & Yunus Syed**

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

**February 1, 2024**

[**Introduction: 4**](#_gjdgxs)

[1.1 Objective 4](#_30j0zll)

[1.2 Top Level block diagram 4](#)

[1.3 Specifications for the design 4](#_prc6ghfmls6f)

[1.3.1 Producer Specification 4](#_3znysh7)

[**2 Verification Requirements 4**](#_2et92p0)

[2.1 Verification Levels 4](#_tyjcwt)

[2.1.1 Hierarchy 4](#_3dy6vkm)

[2.1.2 Control and Observation 5](#_1t3h5sf)

[2.1.3 Comparison Tagging 5](#_4d34og8)

[**3 Required Tools 5**](#_2s8eyo1)

[3.1 Simulation Software 5](#_17dp8vu)

[3.2 Directory structure 5](#_3rdcrjn)

[**4 Risks and Dependencies 7**](#_26in1rg)

[**5 Functions to be Verified. 7**](#_35nkun2)

[5.1 Write and Read Operation 7](#_if6piysvpc58)

[5.2 Empty and Full Status Handling 7](#_fhz69akoax9v)

[5.3 Idle Cycle Handling 8](#_rv3zq83ntz7)

[5.4 Burst Mode 8](#_u88tdhrmrlsi)

[5.5 Duty Cycle Handling 8](#_8xtv75xyay7p)

[**6 Tests and Methods 8**](#_3j2qqm3)

[6.1 Method 8](#_6wntpmg7gbne)

[White box testing is being used because it is known how the internal operation of the FIFO is done. The FIFO is not very complex so it is very easy to perform predictions using two simple input and output mailboxes. 8](#_1y810tw)

[6.2 Advantages and Disadvantages 8](#_4i7ojhp)

[6.3 Architecture 9](#_2xcytpi)

[6.4 Strategy 9](#_1ci93xb)

[6.5 What is your driving methodology? 9](#_3whwml4)

[6.5.1.1 List the test generation methods (Directed test, constrained random) 9](#)

[6.6 What will be your checking methodology? 9](#_2bn6wsx)

[6.6.1.1 From specification, from implementation, from context, from architecture etc 9](#)

[6.7 Detailed Test Descriptions 9](#_qsh70q)

[**7 Coverage Requirements 9**](#_3as4poj)

[7.1.1.1 Describe Code and Functional Coverage goals for the DUV 9](#)

[7.1.1.2 Formulate conditions of how you will achieve the goals. Explain the Covergroups and Coverpoints and your selection of bins. 10](#)

[7.1.2 Assertions 10](#_1pxezwc)

[7.1.2.1 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. 10](#)

[**8 Resources requirements 10**](#_49x2ik5)

[8.1 10](#_2p2csry)

[**9 Schedule 10**](#_147n2zr)

[9.1 Create a table with a plan of completion. You can use milestones as a guide to fill this. 10](#_3o7alnk)

[**10 References Uses / Citations/Acknowledgements 10**](#_23ckvvd)

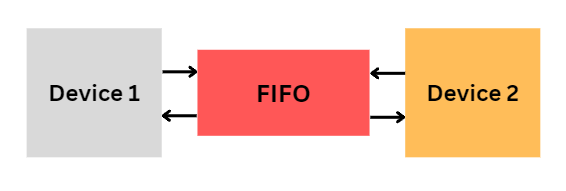
# 

# Introduction:

## **Objective**

The following sections will describe how the asynchronous FIFO will be verified using strategies learned from ECE-593. The goal of this section is to describe what type of testbench style will be used, what verification strategies will be implemented, and what type of stimulus will be used.

## **Top Level block diagram**



## **Specifications for the design**

## Producer Specification

* Frequency: 500MHz
* Idle Cycles: 2 clock cycles
* Burst Length: 1024\
* Duty cycle 50:50
  + 1. **Consumer Specifications**
* Frequency: 225MHz
* Idle Cycles: 1 clock cycles
* Duty cycle 50:50

# Verification Requirements

## **Verification Levels**

### **Hierarchy**

The testing will be performed at a module / block level environment that instantiates the FIFO as a DUT, and where a model of the external interfaces that handle all of the writing and reading from the FIFO. This allows there to be no limitations from any chip level testing. This will allow many corner cases or unique tests to be run against the block.

### **Control and Observation**

The control the user has is within the top level test. Using OOP the test has been simplified to a single start task. This task access protected tasks that read and write from the FIFO, so the controllability within the will be restricted to the scope of the number of data writes and the data vector supplied. Modifications can be made if necessary, but the single action task will perform all the needed transactions.

There will be multithreaded tasks that check all the necessary requirements provided to make sure that no requirement is being violated. This will report a pass or fail monitor/display into the transcript that can be extracted and manually verified to prove that no requirement is being violated.

### **Comparison Tagging**

Any comparison made to verify the asynchronous FIFO is working correctly must have an associated test tag with it and it must be displayed as a pass or failure inside of the transcript. This will be achievable by logging the stimulus imputed into the DUT and by comparing the outputs of the DUT to the inputs since it’s a FIFO the data should not change.

# **Required Tools**

## **Simulation Software**

We are using Mentor Questa as of now. In the future, we are also planning to use formal verification(FPV) so we might end up using the VCS.

## **Directory structure**

The top level directory for this project is structured as follows:

1. **README.md:** Lists the directories and directory descriptions along with how to run the simulation.
2. **docs/:** Any supporting documentation will be stored here for the project e.g. HLDS.
3. **sim/:** Contains simulation files.
   1. work/ directory for simulation files.
   2. run.do file.
   3. wave.do file containing the tests signals for the waveform.
   4. makefile to run the simulation
   5. manifest.f, a RTL and simulation files used in the project.
4. **src/:** Contains the RTL files.
5. **verify/:** Contains the testbench and device model systemverilog code.

A manifest.f file will contain all of the systemverilog files.

| *# File list for RTL and Simulation*    *# RTL*  +incdir+../src/  ../src/async\_fifo.sv    *#../src/design.sv*  *#../src/FIFO\_mem.sv*  *#../src/R\_PTR\_handler.sv*  *#../src/synchronizer.sv*  *#../src/W\_PTR\_handler.sv*    *# Sim*  +incdir+../verify/  ../verify/ccd\_if.sv  ../verify/sim\_ccd.sv  ../verify/sim\_async\_fifo\_tb.sv |
| --- |

A wave.do file exists to autopopulate the waveform signal names.

A makefile is used to compile the systemverilog and run the simulation.

| *# Define Environment Variable*    PROJ\_NAME = 'ASYNC\_FIFO'  PROJ\_HOME = ../    make:  rm -rf work  vlib work  vlog -sv -f manifest.f  vsim -gui -do run.do -classdebug -voptargs="+acc" -uvmcontrol=all -msgmode both -displaymsgmode both work.top -wlf $(PRO    #Clean is ran automatically performed on 'make' to limit simulator issues  clean:  rm -rf work |
| --- |

For the run we are using run.do file

| set StdArithNoWarnings 1 set NumericStdNoWarnings 1 do wave.do run 0 log -r /\* run -all |
| --- |

To run the simulation all the user needs to do is run *make* inside of the *sim/* directory.



# Risks and Dependencies

As Design RTL and Verification is progressing we might get a condition called metastability and rollover issue. It's an issue on hardware, not a simulation issue. As for now we haven't received any issues since the rollover is handled by a rollover or wrapped flag bit that is asserted when the write pointer rolls over or wraps memory, but is deasserted when the read pointer is rolling over or wrapping.

But we can handle using dependencies like Strategies such as effective clock domain crossover procedures, the use of gray-code counters, and the implementation of strong synchronizers can all help to reduce the risks associated with metastability and roll-over difficulties in hardware designs.

# Functions to be Verified.

## Write and Read Operation

The write operation will be verified through the use of an input and output mailbox that the test adds input write data to, and captures the output read data in the other mailbox. A test will be performed when both mailboxes contain an item. This item will be the input and the output of the FIFO, which should contain the same item. A comparison will be made and a test tag will be generated documenting the successful or failure comparison.

## Empty and Full Status Handling

The empty and full status flags are determined by the values of the read and write pointers. On a write, or a read the pointers are incremented, but the result of the increment is modulus to the size of memory to prevent overflow. The conditions to write are that the memory must not be full. This can be determined when the wrapped / rollover bit is asserted and the read and write pointers are at the same location. The empty signal is asserted when the wrapped / rollover bit is deasserted and the write and read pointers are at the same location. The wrapped bit is asserted when the write pointer reaches the end of memory and wraps to the beginning, and is deasserted when the read pointer reaches the end of memory and wraps to the beginning.

## Idle Cycle Handling

The idle cycle is handled in the external device models and it uses a parameter to determine how many loops must be done of *@(posedge clk)* must be done for the associated clock. This allows the idles to occur before performing a read or a write. A test tag needs to be generated confirming the idle cycles exist and this can be done by having a multithreaded task fork join a time tracker for the associated number of clock cycles before asserting a write enable or read enable.

## Burst Mode

Burst writes of 1024 will be tested as it is a maximum burst write requirement. This will also be tested through robustness and many different sizes will be tested. Every single size can be tested as it is a very quick simulation, but in the future if the tests become longer, less tests in the middle of 1 to 1024 range will be used less. This function will have corner cases that include exceeding the max, and performing below the minimum transactions and observing the behavior. A test tag can be generated for the number of expected writes/reads vs the number of actual writes/reads to the dut.

## Duty Cycle Handling

Duty cycle can be determined by kicking off a multithreaded fork join that expects the associated CLK to be asserted for half a period (time), and low for the exact same amount of time. This will be done by using the $time function at the rising edge and falling edge to verify the cycle.

# Tests and Methods

## Method

White box testing is being used because it is known how the internal operation of the FIFO is done. The FIFO is not very complex so it is very easy to perform predictions using two simple input and output mailboxes.

## Tests

### FIFO Full Test

The fifo will be filled and the FIFO\_FULL signal will confirm that the FIFO is full. More data will be attempted to be written, but it should wait until data is read from the FIFO. The FIFO full boundary will be tested.

### FIFO Empty Test

The fifo will initialize in an empty state, and will be filled to make the FIFO in a non-empty state, and then the FIFO will read until it no longer has contents. The FIFO will not read when the FIFO is empty. At the max burst of 1024, the FIFO will eventually become fully written to, and the FIFO will read the remaining data until it is empty.

### Reset Test

The reset test will reset the FIFO in multiple boundaries.

1. Reset After a burst write, and perform another burst write.
2. Reset during a burst write then perform another burst write to show the model properly recovers from a reset state.

### Back To Back Operations

The FIFO will be buffered with data to continuously perform write and read operations. The ideal operation is for the FIFO to wait for the FIFO to be emptied before writing another burst up to 1024 after receiving a burst of 1024. The FIFO will receive back to back 1024 burst writes, and multiple others such as 512 back to back writes. As long as there is a valid consumer and data in the FIFO, it will be read. During a full burst write the producer will finish writing to the FIFO, then the reader will continuously read until the FIFO is empty.

## Advantages and Disadvantages

We already know the internals of the design, therefore we use the white Box method.

Provides detailed testing of internal logic, code paths, and branching, resulting in excellent test coverage. enables early detection and correction of issues during the development process. Provides insights on optimizing code and boosting overall software efficiency.

## Architecture

Object oriented programming is used by making a model of the external interfaces, an interface is used to connect the model to the instantiated asynchronous FIFO (DUT). Methods inside of the model class will trigger single writes or burst writes depending on the stimulus provided.

## Strategy

Dynamic simulation for now to verify at block level and Sub system level. We also plan on using formal verification in the future to test the system

## What is your driving methodology?

For now it’s basic methodology Exhaustive Testing, we will be using UVM in the future

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

Both methods will be implemented

## What will be your checking methodology?

We plan Functional coverage to check and also assertions and scoreboard

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

## Detailed Test Descriptions

| **Test Name** | **Test Tag** | **Description** |
| --- | --- | --- |
| Test | T0001 | This is a test description |
|  | T0004 |  |

# Coverage Requirements

#### Describe Code and Functional Coverage goals for the DUV

Yet to do all the below Steps!

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

### Assertions

#### 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.

# Resources requirements

## 

# Schedule

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

| Milestone | Completed in Verification | Progress | Date of Completion |
| --- | --- | --- | --- |
| 1 | conventional test bench to check RTL, run.do files | Done | Feb 3, 2024 |
| 2 | introduced - test using OOP | Done | Feb 1, 2024 |
| 2 | Introduced - Generators, transactions, Drivers | Done | Feb 6, 2024 |
| 2 | Updated verification plan and got all transcripts matching burts | Done | Feb 11, 2024 |
| 3 | scoreboards and coverage | In Progress |  |
| 3 | revised verification plan & coverage report, do file for simulations |  |  |
| Increment week by week |  |  |  |
| Final Project Submission |  |  | Mar 7, 2024 |

# References Uses / Citations/Acknowledgements