Project: FIFO

Verification Plan & Results Document

Dor Agababa 208133116

Matan Shemesh 318449501

Matan Levy 318262383

Eden Anto 318459518

Version 1.0

February , 2024

1.2 Revision History

Revision	Author	Date	Comment
1.0	Matan Shemesh & Dor Agababa & Eden Anto & Matan Levy	16/03/2024	File creation

Terms and Abbreviations

Term	Meaning
RST	Reset
CLK	Clock
Rd_en	Read enable
We_en	Write enable
Data_in	Input data to be written into FIFO
Data_out	Output data read from FIFO
Empty	Flag to indicate that no data to read
Full	Flag to indicate that there is no option to write data

TABLE OF CONTENTS

2 Device Under Test

1.1Functionality

The DUT is FIFO memory that acts as a buffer for data transfer between 2 synchronous and independent blocks. When the block generating data has data to send, it puts the data on data_in bus and initiates write operation by asserting wr_en control signal. When block consuming the data is ready to read, it asserts rd_en control signal and samples the data from data_out bus. Data are read from the buffer in the order in which they are written.

2.2 Functional Block Diagram

The FIFO module architecture is depicted below:

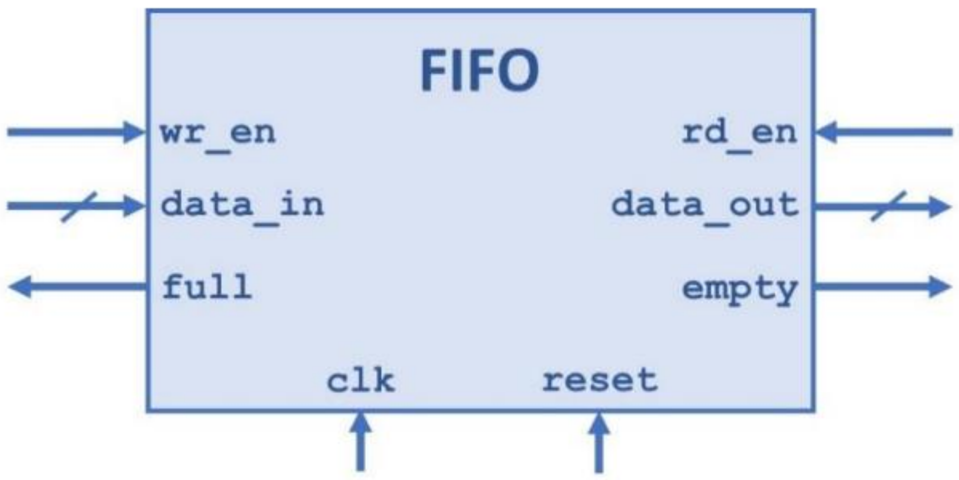


Figure 2-1: FIFO functional block diagram

2.3 Timing Diagram

Timing Diagram for the **FIFO**

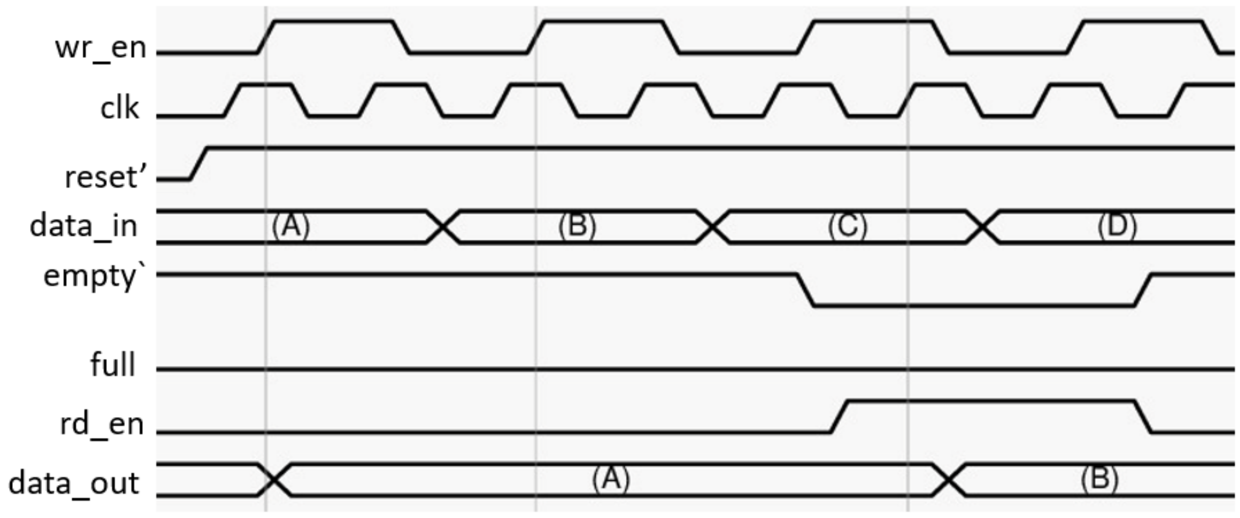


Figure 2-2: FIFO timing diagram

2.4 Features to verify

The following features of the **FIFO** will be tested and covered*:*

- Reset – **FIFO** goes to initial state and its outputs go low at first clock's rising edge
- When the block generating data has data to send, it puts the data on data_in bus and initiates write operation by asserting wr_en control.
- Assertion for all output signals - checks that signal doesn't get 'X value when reset isn't active.
- Assertion for all the signals (excepting clock) – checks there are no glitches (pulses shorter than clock period) on the signal.
- Run all the tests in regression and measure coverage.
- Verify that the empty signal not stuck on low when new data arrives.
- Do the same as above for the full signal.

3 Verification Environment

3.2 Architecture

The **FIFO** verification environment consists of the following components, as shown in figure 2 below:

- **DRIVER**
- **INTERFACE**
- **MONITOR**
- **SEQUENCE**
- **SEQUENCER**
- **SCOREBOARD**

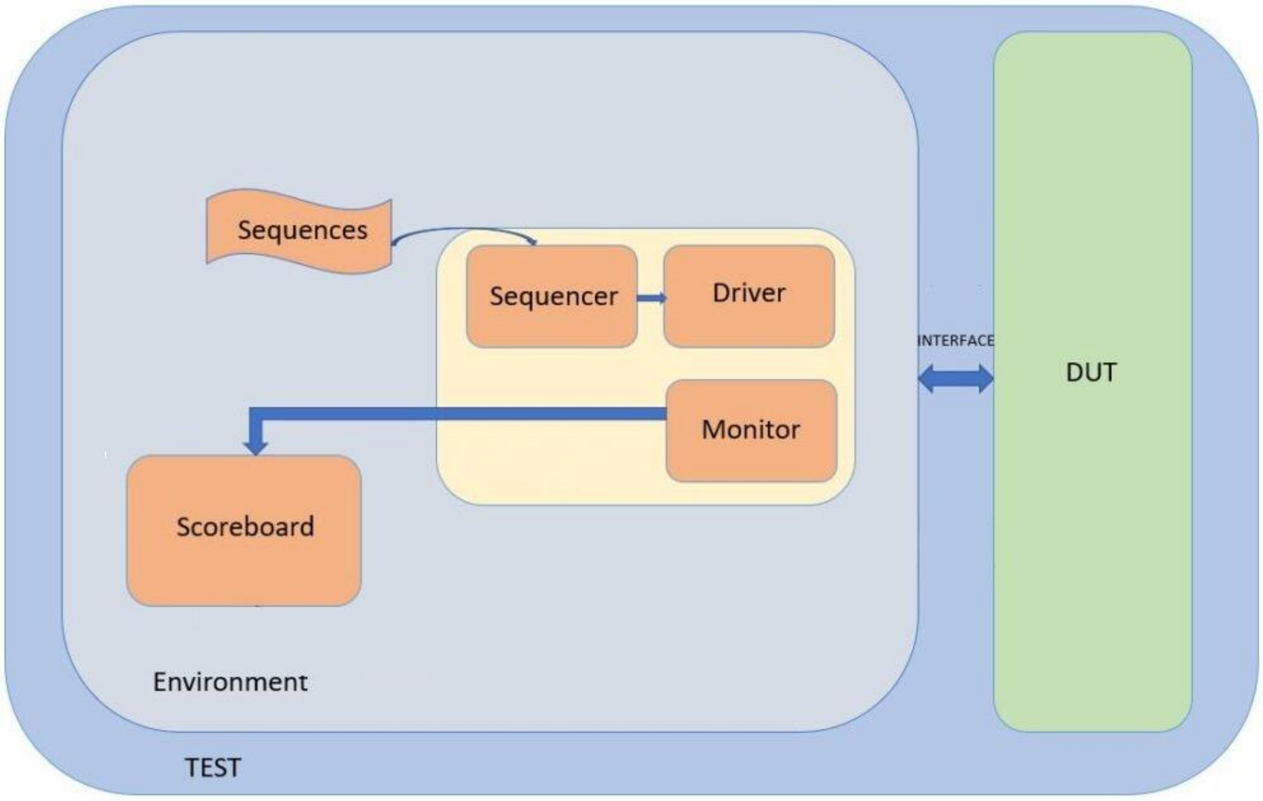


Figure 3-1: FIFO Verification Environment diagram

3.2.1 SEQUENCE

The **SEQUENCE** is used to generate inputs for the **FIFO**. Signals that need to be driven are encapsulated in Transaction (sequence item class) which are sent to **SEQUENCER** which connects **SEQUENCE** and **DRIVER**. Transaction contains all the signals that exist in communication between verification environment and **FIFO**: wr_en , rd_en, data_in,full,empty,data_out. sequence randomizes Transaction and sends it to the **DRIVER** via **SEQUENCER**.

3.2.2 SEQUENCER

The **SEQUENCER** manages the flow of transactions between the **SEQUENCE** and the **DRIVER**. Receiving requests for transaction generation(sequence item) from the **SEQUENCE**, and schedule when to send it to the **DRIVER .**

3.2.3 SCOREBOARD

The **SCOREBOARD** receives transaction data(sequence item) from the **MONITOR** and generates expected results based on the input transactions. It compares these expected results with the actual results received from the DUT (via the **MONITOR**), identifying any discrepancies or errors, and it is the one who determines the test result.

3.2.4 DRIVER

The **DRIVER** is used to drive inputs to the **FIFO** through the **INTERFACE**. It awaits Transaction to be sent by the **SEQUNCER**. When the **DRIVER** receives Transaction it drives its signals to **FIFO** via **INTERFACE**.

3.2.5 INTERFACE

The **INTERFACE** groups functionally related signals used for driving rd_en / wr_en and Reset signals to **FIFO** and sampling data_in received from it, that transfer as output as data_out.

3.2.6 MONITOR

The **MONITOR** is used to keep track of **INTERFACE** signals. Every change of Reset , rd_en / wr_en, full , empty signals will be observed in monitor on upcoming rising edge of clock (using clocking block). **MONITOR** convert the upcoming data from the **INTERFACE** into sequence item and send it to the **SCOREBOARD.**

3.3 Tests list

3.3.1 Reset Test

1. Activate Reset input signal.
2. Check if empty on high, full on low.

3.3.2 Full Flag Test

1. Write data till Full is high.
2. Check that it's impossible to write data when this flag on high.
3. Try to repeat the test.

3.3.3 Empty Flag Test

1. Read data till Empty flag is high.
2. Try to read data and see that is not returning nothing.
3. Verify that the empty signal not stuck on low when new data arrives.

3.3.4 Parallel Test

1. Ensure the FIFO is in a state where it's neither full or empty to perform parallel operations.
2. Start writing data into the FIFO and simultaneously begin reading from it.
3. Check that the specific data that wrote equal to the data that has been reading.
4. Repeatedly execute this test to ensure consistent behavior across multiple iterations.

3.3.5 Random test

1. Randomly choice of write/read/parallel read & write regardless of the fifo state.
2. Check that the action done and the result is as we expect.
3. Repeatedly execute this test to ensure consistent behavior across 100 iterations.

3.4 Tests result

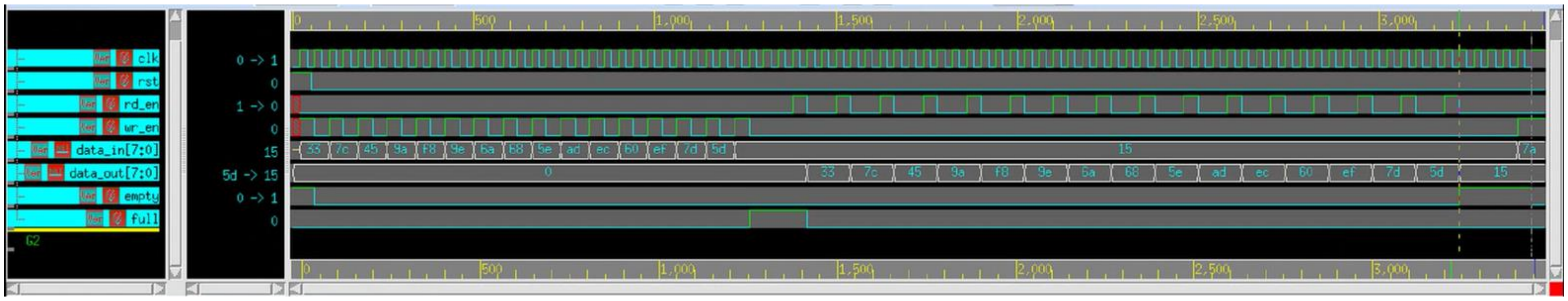
All files can be found in this path : </users/dor.agababa/lab/lab_fifo>

All tests found in comment in the test file : </users/dor.agababa/lab/lab_fifo/test/fifo_test.sv>

All the logs result can be found under this path:

</users/dor.agababa/lab/build_debug_fifo/final_tests_results>

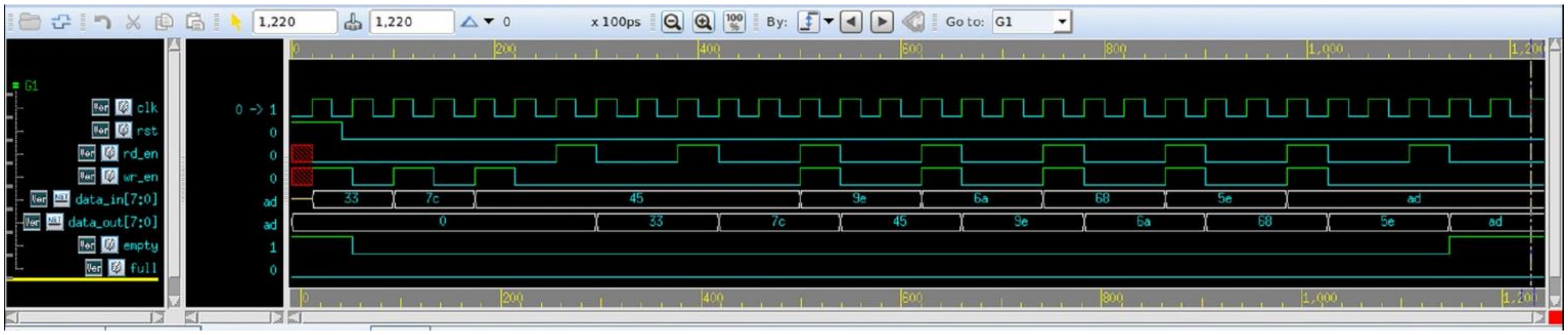
Full flag test + empty flag test



This test start as only writing, until we can see that the full flag is high, we still tried to write inside but it's not worked – as it should be.

In a similar way , we tried to performed the test of read until empty. We can see that the values are as in FIFO – first in first out (for example the data 33 write first and 33 read first).

Parallel test:



In this test, we started from write, and then reading. We can see that the first element the we read is the first one that we write to the FIFO as expected.

Before we started to do the parallel actions, we insert to the FIFO '45' value element and then only we started to generate read and write in parallel.

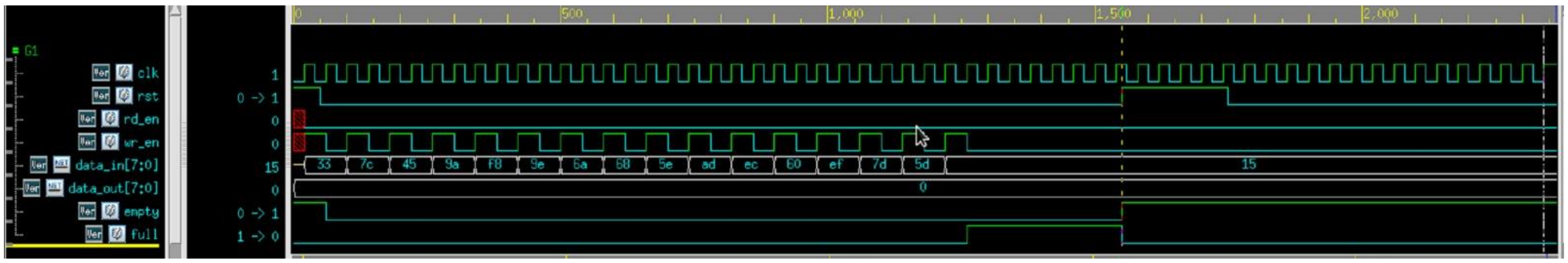
We can see that the first value that we read is really the 45 we insert.

As expected, we read the value from the last read write action, for example if we in time 0 the value x, we will be able to read it only at time x+1 (because we have only one element inside the FIFO).

Reset test:

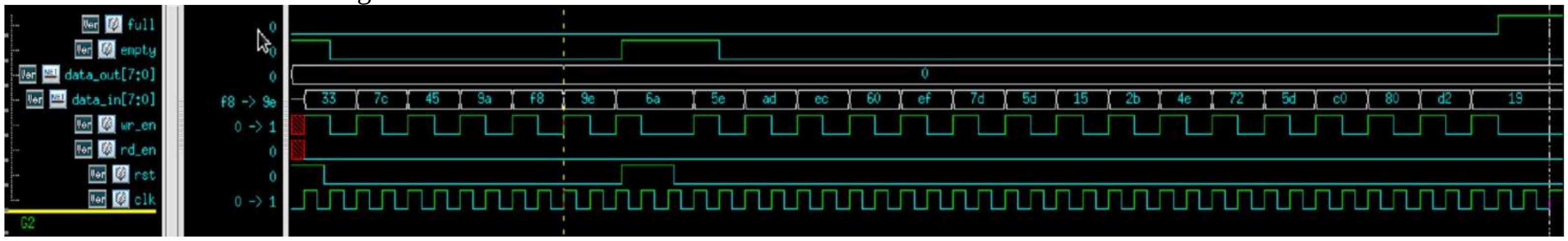
In the reset test, we checked 2 types of reset that can occur:

- Reset in the end of the run:



In this test, we only performed writing (when wr_en on high) until the FIFO become full and at the end of the run we set the rst signal on high. As expected, the full signal become low and the empty signal become high. We added extra time at the end of the test to explicitly see the reset operation.

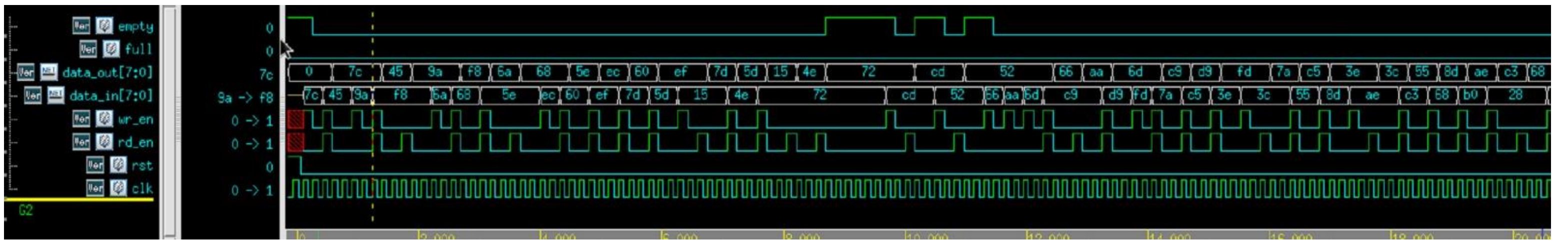
- Reset during a run:

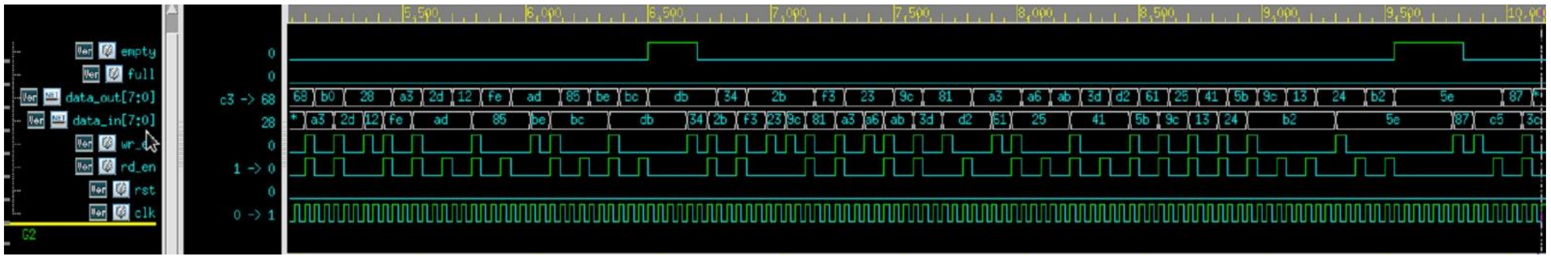


In this test, we performed rest during the run. We start by writing and after few clk cycles we set the rst on high. We can see that in the same moment the empty signal become high, the wr_en stayed high but continue to write only after the rst become low. We keep writing until the full signal become high.

Random test:

In the random test, we performed 100 random actions – read / write / parallel read & write.





We can see in the results that every few clock cycles a different operation is performed, sometimes reading, sometimes writing, and accordingly the corresponding signals changed. We can see that all the signals become high or low in the right time.