Verification Document for Memory Module

- Author: Islam Ashraf

- Revision Number: Rev 1.0
- Date: 19/1/2025

Contents

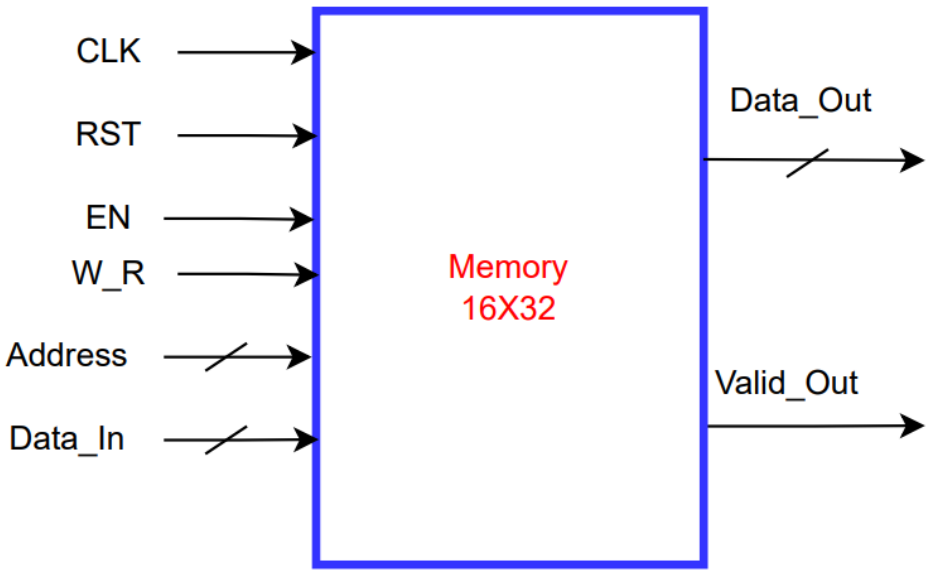
Introduction

This document illustrates the verification plan and results for the Memory module.

The module is designed for reading and writing 32-bit data to a 16-word memory array.

The memory latency for writing and reading operations is one clock cycle obviously if the active signals are enabled. We also can say that the throughput of this memory is one output per one clock cycle.

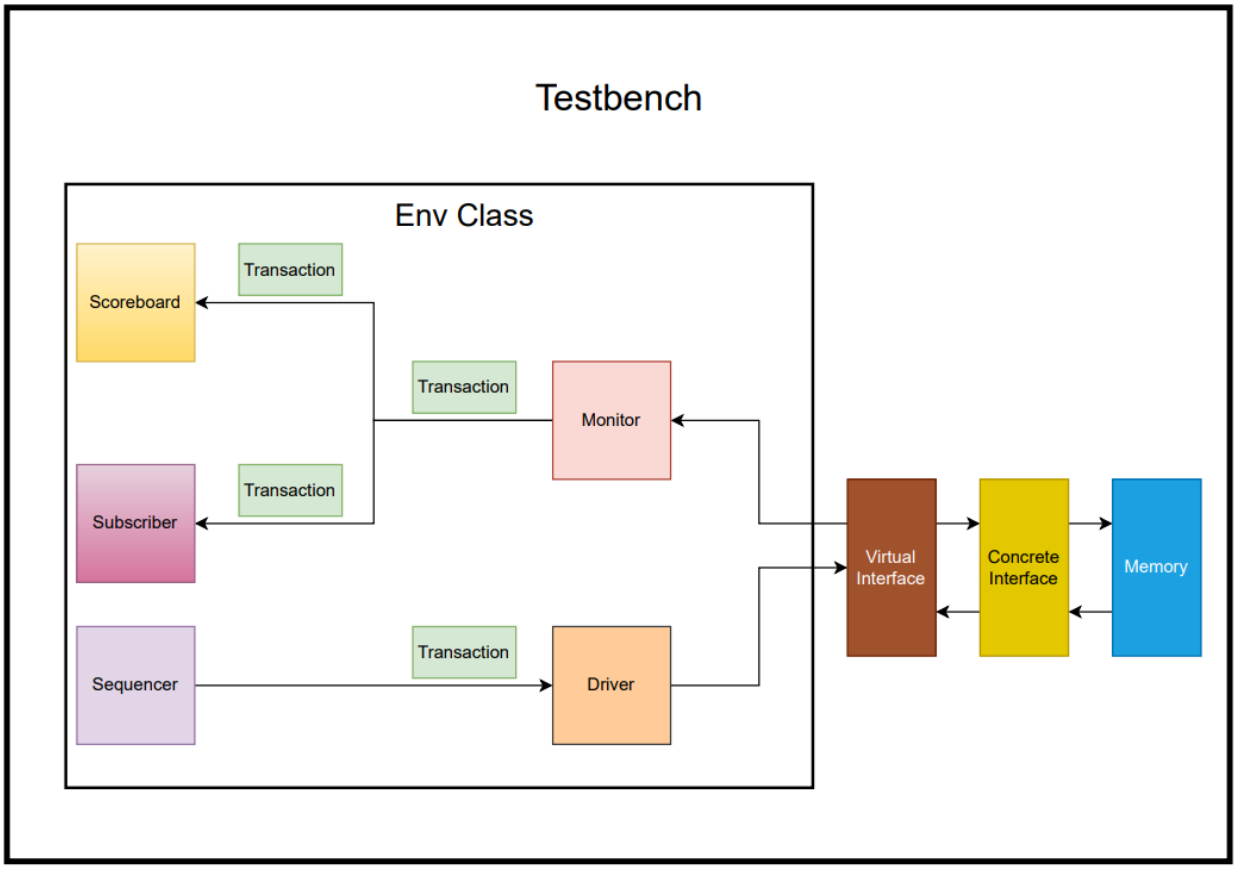
IP Design details



Port	Direction	Width	Description	Connected to
CLK	INPUT	1 bit	Clock Signal	TOP Input Port
RST	INPUT	1 bit	Active Low Reset	TOP Input Port
EN	INPUT	1 bit	Enable Signal	TOP Input Port
Address	INPUT	4 bits	Specifies the memory location for the read or write operation.	TOP Input Port
Data_In	INPUT	32 bits	The data to be written into the memory location	TOP Input Port
W_R	INPUT	1 bit	Write/Read control W_R = 1 >> Write W_R = 0 >> Read	TOP Input Port
Data_Out	OUTPUT	32 bits	Data read from the memory location	TOP Output Port
Valid_Out	OUTPUT	1 bit	Only valid when a read operation is performed	TOP Output Port

Verification Approach

In this project, we used a class-based approach for the verification environment. The design under test (DUT) was connected to various components to simulate and check its behavior effectively.



1. **Transaction-Based Communication:**
    - We created a transaction class to define the inputs and outputs of the DUT, like address, data, and control signals. These transactions acted as the basic communication unit across the environment.

2. **Sequencer:**
    - The sequencer generated transactions and sent them to the driver. It ensured random and directed stimulus to cover all possible scenarios.

3. **Driver:**
    - The driver received transactions from the sequencer and converted them into DUT-specific signal-level interactions through the virtual interface.

4. **Mailbox:**
    - We used a bounded mailbox for synchronization between components. Transactions were passed from the sequencer to the driver through this mailbox.

5. **Monitor:**
    - The monitor observed the DUT's output and captured it into transactions. and forward these transactions for further analysis, such as checking in the scoreboard or collecting coverage in the subscriber.

6. **Scoreboard:**
    - The scoreboard validates the DUT's functionality by comparing its actual outputs to expected values. The expected outputs were generated by a golden model, implemented as a dynamic 2D array.
7. **Subscriber:**
    - The subscriber collected coverage data for analysis. Coverage was added to check if all scenarios were tested.

8. **Environment:**
    - All components were integrated within the environment class, which acted as the top-level module for the testbench.

Clock Blocking Technique

In this project, we used a clock blocking technique to align all operations with the clock cycle, avoiding race conditions and ensuring stable data flow.

Test Items

- **EN**
  - Set to 1 → Memory accepting data
  - Set to 0 → Memory not accepting data
- **W_R**
  - Set to 1 → Write data to the memory
  - Set to 0 → Read data from the memory
- **Address**
  - Specifies the memory location for the read or write operation.
- **Data_In**
  - Check if the data valid or invalid

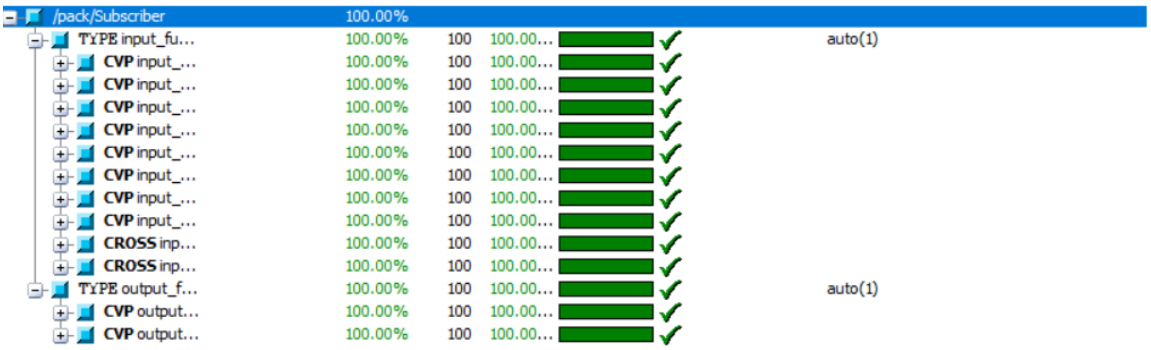
Test case table

Test Case ID	Test Name	Description	Expected Outcome
TC1	Reset Test	Reset all memory locations to 0	Memory cleared Valid_Out = 0
TC2	Enable Test	Disable En and perform operations	No change in memory values or output
TC3	Write Test	Write data to address X	Data stored at address X
TC4	Read Test	Read data from address Y	Data read matches address Y
TC5	Data Test	Try different data valid and invalid	See if the system crashes
TC6	Address Test	Try different address valid and invalid	See if the system crashes

Coverage

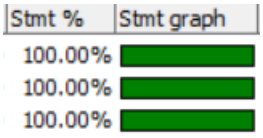
Functional Coverage

Functional coverage is a measure of what functionalities/features of the design have been exercised by the tests.



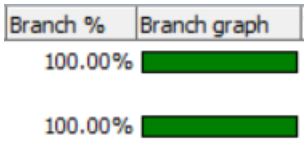
Code Coverage

This measures the percentage of code statements that have been executed during the simulation.



Branch Coverage

This measures the percentage of decision points in the code that have been exercised by the testbench.



Traceability table

Test ID	Reset		Enable		Write		Read		Data		Address	
	Enabled	Disabled	Enabled	Disabled	Enabled	Disabled	Enabled	Disabled	Valid	Invalid	Valid	Invalid
TC1	☐	☑	☑	☐	☑	☐	☐	☑	☑	☐	☑	☐
TC2	☐	☑	☑	☐	☐	☑	☑	☐	☑	☐	☑	☐
TC3	☑	☐	☑	☐	☑	☐	☐	☑	☑	☐	☑	☐
TC4	☐	☑	☐	☐	☐	☐	☐	☐	☐	☐	☐	☐
TC5	☐	☑	☐	☑	☑	☐	☐	☑	☑	☐	☑	☐
TC6	☐	☑	☑	☐	☐	☑	☑	☐	☑	☐	☐	☑

List of issues

No issues found

Assessment

Good to go 😊