Verification Test Plan

BY: OMAR MAHMOUD ELSHERIF

Table of Contents

Table of Figures

Introduction

This document defines the detailed verification test plan for a memory block, which has a data width of 32-bits and 4-bits address bus. The memory is synchronized using a single clock input and an Asynchronous reset signal. The memory also consists of two operational modes, writing and reading modes, that are controlled by an enable signal. It also has an output valid signal that when it is asserted high, indicates a reading operation taking place, otherwise it is asserted low.

Memory Block Description:

I. Block Interface:

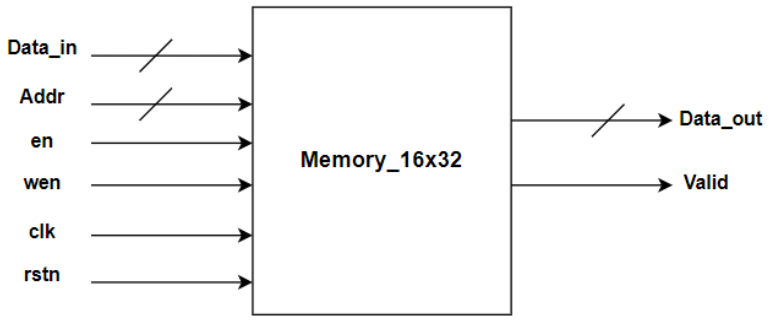

Figure 1 Block Diagram of Memory

II. Parameters:

Parameter Name	Default Value (bits)	Description
ADDR_WIDTH	4	Address bus width of memory
DATA_WIDTH	32	Data bus width of memory

Table 1 Parameters Table

III. IO Ports:

Port Name	Direction	Width	Description
clk	IN	1	Clock Signal.
rstn	IN	1	Asynchronous reset signal.
en	IN	1	Enable signal that enables the memory operation.
wen	IN	1	Write Enable signal that identifies the operational mode. when asserted high it identifies writing operation. when asserted low it identifies a reading operation.
Addr	IN	4	Address bus of memory.
Data_in	IN	32	Data bus that is written into the memory in case of a writing operation.
Data_out	OUT	32	Data bus that is read from the memory in case of a reading operation.
Valid	OUT	1	Valid signal that is asserted high during a reading operation, otherwise asserted low.

Table 2 IO Ports Table

IV. RTL Code:

```
////////////////////////////////////////////////////////////////////////////
//////////////// Module ports list, declaration, and data type ////////////////
////////////////////////////////////////////////////////////////////////////
module memory_16x32 #(
    parameter   ADDR_WIDTH  =   4,
    parameter   DATA_WIDTH  =   32
) (
//////////////////// Inputs ////////////////////
    input   wire    logic                       clk,
    input   wire    logic                       rstn,
    input   wire    logic                       en,
    input   wire    logic                       wen,
    input   wire    logic   [ADDR_WIDTH-1:0]    Addr,
    input   wire    logic   [DATA_WIDTH-1:0]    Data_in,

//////////////////// Outputs ////////////////////
    output  wire    logic   [DATA_WIDTH-1:0]    Data_out,
    output  wire    logic                       Valid
);

////////////////////////////////////////////////////////////////
//////////////////////  local Parameters  //////////////////////
////////////////////////////////////////////////////////////////

localparam MEM_DEPTH = 2**ADDR_WIDTH;

////////////////////////////////////////////////////////////////
//////////////////  Internal  Elements  //////////////////////
////////////////////////////////////////////////////////////////

// memory 16x32
reg [DATA_WIDTH-1:0]    Memory  [MEM_DEPTH-1:0];
// iterator
integer i;

////////////////////////////////////////////////////////////////
//////////////////////  Write Operation   //////////////////////
////////////////////////////////////////////////////////////////

always_ff @( posedge clk, negedge rstn ) begin
    // Asynchronous reset
    if(!rstn) begin
        for(i=0; i<MEM_DEPTH; i=i+1) begin
            Memory[i]   <=    'b0;
        end
    end
    // memory enabled and write operation
    else if (en && wen) begin
            Memory[Addr]    <=  Data_in;
    end

end
////////////////////////////////////////////////////////////////
//////////////////////  Read Operation   //////////////////////
////////////////////////////////////////////////////////////////

assign Data_out = ( en & !wen ) ? Memory[Addr] : 'b0;
assign Valid    = ( en & !wen ) ? 1'b1 : 1'b0;

endmodule
```

Verification Plan/Environment

The verification environment used in this document is **Block Level Verification**, which is a verification process that focuses on validating the functionality of individual blocks or modules within a chip.

The verification environment includes the following:

- **Testbench**: a dedicated self-checking testbench is created to provide stimulus and check the behavior of the block under different conditions. This testbench mimics the inputs that the block might encounter in the full system.

- **Checkers**: Checkers are used to validate output signals from the block under test, ensuring they meet the specifications.

There are 3 types of famous Block level verification methodologies as:

- Module Based
- Class Based
- UVM Based

The used methodology in this verification test plan is the **Class Based block level verification**.

Where a Class Based block level verification is a methodology used in verifying individual blocks or modules in an integrated circuit. It is particularly useful for complex designs, where creating a modular, reusable, and scalable verification environment is essential.

This approach is based on object-oriented programming (OOP) concepts implemented using System Verilog language.

Test items

The test items for the memory block are as following:

- **rstn**:
  - Set to 0 → All of memory's internal storage data is reset to 0.
  - Set to 1 → No effect on memory's internal storage data.

- **en**:
  - Set to 0 → Disables the memory's operation.
  - Set to 1 → Enables the memory's operation.

- **wen**:
  - Set to 0 → Indicates a memory's reading operation.
  - Set to 1 → Indicates a memory's writing operation.

- **Addr**:
  - Valid address → Indicates a valid memory's address location.
  - Invalid address → Indicates an invalid memory's address location.

- **Data_in**:
  - Valid data → Indicates a valid memory's input data.
  - Invalid data → Indicates an invalid memory's input data.

- **Data_out**:
  - Valid data → Indicates a valid memory's output data.
  - Invalid data → Indicates an invalid memory's output data.

- **Valid**:
  - Asserted to 1 → Indicates a memory's reading operation.
  - Asserted to 0 → Indicates a memory's writing operation.

Test Case Table

The definition of the test cases is listed in the following table, with each test case permutations that invokes it.

Test Name	rstn	en	wen	Addr	Data_in	Data_out	Valid	Description
Memory Reset	Low	-	-	-	-	-	Low	Ensure Proper Asynchronous reset functionality
Memory disabled	Low	Low	-	-	-	-	Low	Check that no read/write operation occurs to memory
Read Operation	High	high	Low	Valid address	-	Valid data	High	Ensure Proper Reading operation
Write Operation	High	high	High	Valid address	Valid data	-	Low	Ensure Proper Writing operation
Concurrent Read/Write Operation	High	high	Low/ High	Valid address	Valid data	Valid data	Low/ High	Check consequent read, write operations
Invalid input Data	High	high	Low/ High	Valid address	Invalid data	-	Low	Check data validity on memory
Out of Scope address	High	high	Low/ High	Invalid address	Valid data	-	Low/ High	Check address boundaries

Coverage Measurement

Test coverage metrics are quantitative measures that indicate how thoroughly your test cases cover the features, functions, and specifications of your IC design.

We have many types of Coverage measurement in digital IC verification:

- **Code Coverage:**
  - **Purpose**: To measure whether each executable statement in the code has been executed.
  - **Types**:
    - Statement Coverage: Measures whether each executable statement in the code has been executed.
    - Branch Coverage: Measures whether each possible branch (if-else) has been tested.
    - Condition Coverage: Tracks the individual conditions within complex logic, ensuring all Boolean conditions have been evaluated as both true and false.
    - FSM (Finite State Machine) Coverage: Verifies that all states and transitions in a state machine have been visited during simulation.

- **Functional Coverage:**
  - **Purpose**: To ensure that the design's intended functionality has been fully exercised and verified.
  - **Types**:
    - Cover Points: Specific functional points in the design that should be hit during testing (e.g., certain outputs or data paths).
    - Cross Coverage: A combination of two or more cover points to verify their interaction (e.g., checking different inputs and outputs together).
    - Assertion Coverage: Ensures that specific assertions or properties about the design's behavior are met during simulation, ensuring correct functional behavior.

- **Test Plan Coverage:**
  - **Purpose**: Ensures that all test cases in the verification plan have been executed and evaluated.
  - **Types**:
    - Test Case Coverage: Tracks which tests from the verification plan have been run and completed.
    - Scenario Coverage: Ensures that all predefined design scenarios (corner cases, typical operations, etc.) are covered.

Exit Criteria

An exit scenario in digital IC verification refers to the specific conditions or criteria under which the verification process for a particular design module or the entire IC design can be considered complete. It defines the endpoint of the verification process, ensuring that all key aspects of the design have been thoroughly tested, and no critical issues remain unresolved.

For the memory block design, the exit scenario include:

1. Functional coverage, code coverage, and Test Plan coverage have met > 95%.
2. All test cases (including corner cases) in the verification plan have been executed.
3. All critical and high-priority bugs have been fixed and verified.