Assignment 1: Verification of Parameterized RAM Module

Objective

The goal of this assignment is to verify the functionality of a parameterized RAM module by creating directed and random test cases using $random. You are required to develop a testbench in SystemVerilog to test the module under different conditions and ensure correctness.

Design Specifications

1. **Parameterized Configuration:**
    - **MEM_WIDTH:** Defines the bit-width of the memory (default: 8 bits).
    - **MEM_DEPTH:** Specifies the number of memory locations (default: 256 locations).
    - **ADDR_SIZE:** Computed using $clog2(MEM_DEPTH), representing the address width.
2. **Ports:**
    - **clk:** Clock signal controlling memory operations.
    - **rst_n:** Asynchronous active-low reset, which resets the dout when asserted.
    - **wr_en:** Write enable signal (when high, stores din at addr_wr).
    - **rd_en:** Read enable signal (when high, outputs data from addr_rd to dout).
    - **addr_wr:** Address for write operation.
    - **addr_rd:** Address for read operation.
    - **din:** Data input for writing.
    - **dout:** Data output for reading.

3. **Functionality:**
    - Data is written into memory when **wr_en** is high on the rising edge of clk.
    - Data is read from memory when **rd_en** is high on the rising edge of clk.
    - Reset (**rst_n**) clears the output (**dout**).

Verification Requirements

Your task is to develop a SystemVerilog testbench that verifies the RAM module using directed and random test cases. The verification should include:

1. **Directed Test Cases:**
    - **Basic Read/Write Operations:** Write a value to a specific address and read it back.
    - **Reset Behavior:** Ensure dout resets correctly when rst_n is asserted.
    - **Edge Cases:** Test the first and last memory locations.
2. **Random Test Cases:**
    - Use $random to generate random addresses, data, and control signals.
    - Verify that random sequences of read and write operations behave as expected.
    - Test corner cases such as consecutive reads/writes and invalid accesses.
3. **Coverage Goals:**
    - Ensure all key functionalities are exercised.
    - Generate a basic report summarizing the test results.

Note: You **must** use tasks in your testbench to improve modularity and code readability. Tasks should be implemented for common operations like writing to and reading from memory, ensuring better structure and reusability in the verification process.

Submission Requirements

One PDF file having the following:

1. Verification plan
2. Design file
   If the design has bugs, then fix them otherwise upload the given design file.
3. Testbench file
4. Do file
5. Coverage report text file
6. Clear and neat QuestaSim waveform snippets showing the functionality of the design with each test plan item
7. Branch, statement and toggle coverage report snippets with justification if you could not reach 100% coverage for the designs.
8. Below is a basic SystemVerilog testbench to help you get started:

```
module ram_tb;
  parameter MEM_WIDTH = 8;  // Bit-width of each memory location
  parameter MEM_DEPTH = 256; // Total number of memory locations
  parameter ADDR_SIZE = $clog2(MEM_DEPTH); // Address width calculated from MEM_DEPTH

  // Define signals
  logic clk, rst_n, wr_en, rd_en;
  logic [ADDR_SIZE-1:0] addr_wr, addr_rd;
  logic [MEM_WIDTH-1:0] din;
  logic [MEM_WIDTH-1:0] dout;
  // Instantiate RAM module
  RAM #(.MEM_WIDTH(MEM_WIDTH), .MEM_DEPTH(MEM_DEPTH)) dut (
    .clk(clk), .rst_n(rst_n), .wr_en(wr_en), .rd_en(rd_en),
    .addr_wr(addr_wr), .addr_rd(addr_rd), .din(din), .dout(dout)
  ); // you can use dut(.*) if names of signal in testbench is similar to design

  // Clock generation
  always #5 clk = ~clk;
  // Task for writing data to memory
  task write(input logic [ADDR_SIZE-1:0] addr, input logic [MEM_WIDTH-1:0] data);
    begin
      wr_en = 1;
      addr_wr = addr;
      din = data;
      #10;
      wr_en = 0;
    end
  endtask
  // Test sequence
  initial begin
    clk = 0; rst_n = 0; wr_en = 0; rd_en = 0;
    addr_wr = 0; addr_rd = 0; din = 0;

    #10 rst_n = 1; // Release reset
    // Directed test case using the write task
    write(5, 8'hA5); // Write 0xA5 to address 5

    // Single randomized test case
    write($random % MEM_DEPTH, $random);
    #50 $stop; // Stop simulation
  end
endmodule
```

   You should modify and extend this testbench to include more corner cases, improve coverage, and ensure correctness.

Good luck!