# Memory Controller Design & Verification: Using SystemVerilog, OOP, and UVM Principles

---

**Tool: EDA Playground ( www.edaplayground.com)**
**Simulator: Aldec Riviera Pro**

---

## Program Code & Testbench

---

```
// design.sv
//------------------------------------------------------------
// A simple, parameterized single-port synchronous RAM module.
//------------------------------------------------------------
module simple_ram
#(
  parameter DATA_WIDTH = 8,
  parameter ADDR_WIDTH = 8
)
(
  input  logic                          clk,
  input  logic                          rst_n,
  // Port signals
  input  logic                          cs,       // Chip Select
  input  logic                          we,       // Write Enable (1=Write,
0=Read)
  input  logic          addr,
  input  logic          wdata,
  output logic          rdata
);

  // Calculate the depth of the memory based on address width
  localparam DEPTH = 2**ADDR_WIDTH;

  // Declare the internal memory array
  logic mem;

  // Synchronous read/write logic
  always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      // Asynchronous reset: clear the memory contents
      for (int i = 0; i < DEPTH; i++) begin
        mem[i] <= '0;
      end
      rdata <= '0;
    end
    else begin
```

```
      if (cs) begin
        if (we) begin
          // Write operation
          mem[addr] <= wdata;
        end
        // Read operation: Data is available on the next clock cycle
        // This models a typical block RAM behavior where the output is
registered.
        rdata <= mem[addr];
      end
    end
  end

endmodule
//------------------------------------------------------------
// Transaction Class
//------------------------------------------------------------
class mem_transaction;
  rand bit [7:0] addr;
  rand bit [7:0] wdata;
  rand bit       we;
  bit [7:0] rdata;

  constraint wr_rd_dist_c { we dist {1 := 6, 0 := 4}; }

  function void print(string tag = "TRANS");
    $display("[%s] @ %0t: Addr=0x%0h, WE=%b, WData=0x%0h, RData=0x%0h",
             tag, $time, addr, we, wdata, rdata);
  endfunction
endclass
//------------------------------------------------------------
// Generator Class
//------------------------------------------------------------
class generator;
  mailbox #(mem_transaction) gen2drv_mbx;
  int repeat_count;

  function new(mailbox #(mem_transaction) gen2drv_mbx, int repeat_count = 50);
    this.gen2drv_mbx  = gen2drv_mbx;
    this.repeat_count = repeat_count;
  endfunction

  task run();
    mem_transaction tx;
    $display("[GEN] Starting to generate %0d transactions...", repeat_count);
    repeat (repeat_count) begin
      tx = new();
      if (!tx.randomize()) $fatal(1, "[GEN] Transaction randomization
failed!");
```

```
      tx.print("GEN SEND");
      gen2drv_mbx.put(tx);
    end
    $display("[GEN] Finished generating transactions.");
  endtask
endclass
//------------------------------------------------------------
// Driver Class
//------------------------------------------------------------
class driver;
  mailbox #(mem_transaction) gen2drv_mbx;
  virtual mem_if vif;

  function new(mailbox #(mem_transaction) gen2drv_mbx);
    this.gen2drv_mbx = gen2drv_mbx;
  endfunction

  task reset_signals();
    vif.cs   <= 0;
    vif.we   <= 0;
    vif.addr <= '0;
    vif.wdata<= '0;
    repeat(2) @(posedge vif.clk);
  endtask

  task run();
    $display(" Driver is running...");
    forever begin
      mem_transaction tx;
      gen2drv_mbx.get(tx);
      tx.print("DRV RECV");

      @(posedge vif.clk);
      vif.cs <= 1;
      vif.we <= tx.we;
      vif.addr <= tx.addr;
      if (tx.we) vif.wdata <= tx.wdata;

      @(posedge vif.clk);
      vif.cs <= 0;
      vif.wdata <= '0;
    end
  endtask
endclass

//------------------------------------------------------------
// Monitor Class
//------------------------------------------------------------
class monitor;
```

```
  mailbox #(mem_transaction) mon2scb_mbx;
  virtual mem_if vif;

  function new(mailbox #(mem_transaction) mon2scb_mbx);
    this.mon2scb_mbx = mon2scb_mbx;
  endfunction

  task run();
    mem_transaction tx;
    $display("[MON] Monitor is running...");
    forever begin
      @(posedge vif.clk);
      if (vif.cs) begin
        tx = new();
        tx.addr  = vif.addr;
        tx.we    = vif.we;
        if (tx.we) tx.wdata = vif.wdata;
        else begin
          @(posedge vif.clk);
          tx.rdata = vif.rdata;
        end
        tx.print("MON CAPTURE");
        mon2scb_mbx.put(tx);
      end
    end
  endtask
endclass

//------------------------------------------------------------------
// Scoreboard Class
//------------------------------------------------------------------
class scoreboard;
  mailbox #(mem_transaction) mon2scb_mbx;
  bit [7:0] ref_mem [bit [7:0]];
  int pass_count = 0;
  int fail_count = 0;

  function new(mailbox #(mem_transaction) mon2scb_mbx);
    this.mon2scb_mbx = mon2scb_mbx;
  endfunction

  task run();
    mem_transaction tx;
    $display(" Scoreboard is running...");
    forever begin
      mon2scb_mbx.get(tx);
      tx.print("SCB RECV");

      if (tx.we) begin
```

```
        ref_mem[tx.addr] = tx.wdata;
        $display(" Write to Addr 0x%0h with Data 0x%0h recorded.", tx.addr,
tx.wdata);
      end
      else begin
        if (ref_mem.exists(tx.addr)) begin
          if (tx.rdata == ref_mem[tx.addr]) begin
            $display(" PASS: Addr 0x%0h, RData 0x%0h matches expected 0x%0h.",
tx.addr, tx.rdata, ref_mem[tx.addr]);
            pass_count++;
          end else begin
            $error(" FAIL: Addr 0x%0h, RData 0x%0h!= expected 0x%0h.", tx.addr,
tx.rdata, ref_mem[tx.addr]);
            fail_count++;
          end
        end
        else begin
          if (tx.rdata == 8'h00) begin
            $display(" PASS: Addr 0x%0h, RData 0x%0h matches expected default
0x00.", tx.addr, tx.rdata);
            pass_count++;
          end else begin
            $error(" FAIL: Addr 0x%0h, RData 0x%0h!= expected default 0x00.",
tx.addr, tx.rdata);
            fail_count++;
          end
        end
      end
    end
  endtask
endclass

//-------------------------------------------------------------
// Environment Class
//-------------------------------------------------------------
class environment;
  generator  gen;
  driver     drv;
  monitor    mon;
  scoreboard scb;
  mailbox #(mem_transaction) gen2drv_mbx;
  mailbox #(mem_transaction) mon2scb_mbx;
  virtual mem_if vif;
  event gen_done;

  function new(virtual mem_if vif);
    this.vif = vif;
    gen2drv_mbx = new();
    mon2scb_mbx = new();
```

```
    gen = new(gen2drv_mbx, 50);
    drv = new(gen2drv_mbx);
    mon = new(mon2scb_mbx);
    scb = new(mon2scb_mbx);
  endfunction

  task run();
    $display("[ENV] Environment run phase starting...");
    drv.vif = this.vif;
    mon.vif = this.vif;
    fork
      begin
        gen.run();
        ->gen_done; // Trigger event when generator is done
      end
      drv.run();
      mon.run();
      scb.run();
    join_none
  endtask

  task wait_for_completion();
    @gen_done;
    #200; // Allow final transactions to propagate and be checked
    $display("------------------------------------------------------------");
    $display("[ENV] Test finished. Final Score: %0d Passed, %0d Failed.",
scb.pass_count, scb.fail_count);
    $display("------------------------------------------------------------");
  endtask
endclass

//---------------------------------------------------------------
// Interface
//---------------------------------------------------------------
interface mem_if(input logic clk, input logic rst_n);
  // FIX: Added parameters to define widths, which can be overridden at
instantiation.
  parameter DATA_WIDTH = 8;
  parameter ADDR_WIDTH = 8;

  // FIX: Declared signals as vectors with the correct widths.
  logic cs;
  logic we;
  logic [ADDR_WIDTH-1:0] addr;
  logic [DATA_WIDTH-1:0] wdata;
  logic [DATA_WIDTH-1:0] rdata;
endinterface

//---------------------------------------------------------------
```

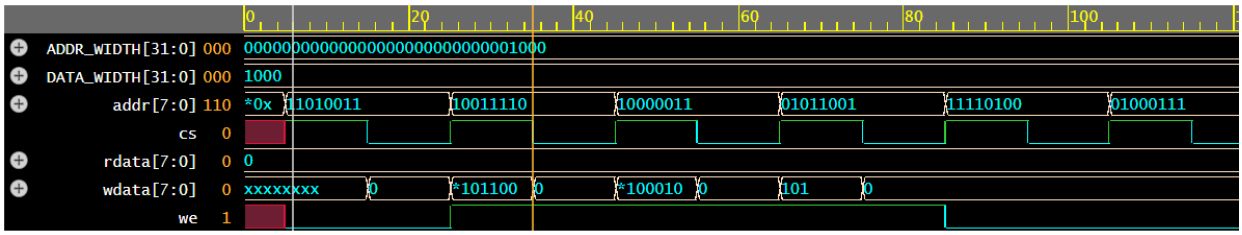
```
// DUT: simple_ram
//------------------------------------------------------------
module simple_ram
#(parameter DATA_WIDTH = 8, parameter ADDR_WIDTH = 8)
(
  input  logic clk, rst_n, cs, we,
  // FIX: Declared ports as vectors with the correct widths.
  input  logic [ADDR_WIDTH-1:0] addr,
  input  logic [DATA_WIDTH-1:0] wdata,
  output logic [DATA_WIDTH-1:0] rdata
);
  localparam DEPTH = 2**ADDR_WIDTH;

  // FIX: Declared 'mem' as a 2D array: [width] for data and [depth] for
addresses.
  logic [DATA_WIDTH-1:0] mem [DEPTH];

  always_ff @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
      for (int i = 0; i < DEPTH; i++) mem[i] <= '0;
      rdata <= '0;
    end
    else begin
      if (cs) begin
        if (we) mem[addr] <= wdata;
        // For a more realistic RAM model, read should present data on the next
cycle.
        // This combinatorial read is okay for this simple model but can cause
issues.
        rdata <= mem[addr];
      end
    end
  end
endmodule
//------------------------------------------------------------
// Top-Level Testbench Module
//------------------------------------------------------------
module test_top;
  logic clk;
  logic rst_n;

  always #5 clk = ~clk;

  initial begin
    clk <= 0;
    rst_n <= 0;
    #20;
    rst_n <= 1;
  end
```

```
  mem_if vif(clk, rst_n);
  simple_ram dut
(.clk(vif.clk),.rst_n(vif.rst_n),.cs(vif.cs),.we(vif.we),.addr(vif.addr),.wdata
(vif.wdata),.rdata(vif.rdata));

  initial begin
    environment env;
    env = new(vif);
    env.run();
    env.wait_for_completion();
    $finish;
  end

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, test_top);
  end
endmodule
```

---

## Output : EP Wave and Logs Genereted

---




```
[2025-10-17 05:47:14 UTC] vlib work && vlog '-timescale' '1ns/1ns' design.sv testbench.sv  &&
vsim -c -do "vsim +access+r; run -all; exit"
VSIMSA: Configuration file changed: `/home/runner/library.cfg'
ALIB: Library "work" attached.
work = /home/runner/work/work.lib
MESSAGE "Unit top modules: test_top."
SUCCESS "Compile success 0 Errors 0 Warnings  Analysis time: 0[s]."
done
# Aldec, Inc. Riviera-PRO version 2023.04.112.8911 built for Linux64 on May 12, 2023.
# HDL, SystemC, and Assertions simulator, debugger, and design environment.
# (c) 1999-2023 Aldec, Inc. All rights reserved.
# ELBREAD: Elaboration process.
# ELBREAD: Elaboration time 0.0 [s].
# KERNEL: Main thread initiated.
# KERNEL: Kernel process initialization phase.
# ELAB2: Elaboration final pass...
# KERNEL: PLI/VHPI kernel's engine initialization done.
# PLI: Loading library '/usr/share/Riviera-PRO/bin/libsystf.so'
# ELAB2: Create instances ...
# KERNEL: Time resolution set to 1ns.
# ELAB2: Create instances complete.
# SLP: Started
# SLP: Elaboration phase ...
# SLP: Elaboration phase ... done : 0.0 [s]
# SLP: Generation phase ...
```

```
# SLP: Generation phase ... done : 0.0 [s]
# SLP: Finished : 0.1 [s]
# SLP: 0 primitives and 4 (80.00%) other processes in SLP
# SLP: 6 (1.61%) signals in SLP and 19 (5.11%) interface signals
# ELAB2: Elaboration final pass complete - time: 0.1 [s].
# KERNEL: SLP loading done - time: 0.0 [s].
# KERNEL: Warning: You are using the Riviera-PRO EDU Edition. The performance of simulation is
reduced.
# KERNEL: Warning: Contact Aldec for available upgrade options - sales@aldec.com.
# KERNEL: SLP simulation initialization done - time: 0.0 [s].
# KERNEL: Kernel process initialization done.
# Allocation: Simulator allocated 5596 kB (elbread=459 elab2=4969 kernel=166 sdf=0)
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: [ENV] Environment run phase starting...
# KERNEL: [GEN] Starting to generate 50 transactions...
# KERNEL: [GEN SEND] @ 0: Addr=0xd3, WE=0, WData=0xad, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x9e, WE=1, WData=0xec, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x83, WE=1, WData=0xa2, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x59, WE=1, WData=0x5, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xf4, WE=0, WData=0x6e, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x47, WE=0, WData=0xb3, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xec, WE=0, WData=0xac, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xef, WE=1, WData=0x7a, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x46, WE=1, WData=0xf9, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x7d, WE=1, WData=0xcd, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xa7, WE=1, WData=0xee, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x85, WE=1, WData=0xfe, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xa2, WE=0, WData=0x6a, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x3d, WE=1, WData=0xc3, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xda, WE=0, WData=0xe5, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xd6, WE=0, WData=0xbb, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x17, WE=0, WData=0xf8, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xe7, WE=0, WData=0x80, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xa9, WE=1, WData=0xf6, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x45, WE=1, WData=0x59, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xd6, WE=1, WData=0x3b, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x47, WE=1, WData=0x98, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x20, WE=1, WData=0x1b, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x45, WE=0, WData=0xde, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x7c, WE=1, WData=0x9a, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x4a, WE=1, WData=0x75, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xe8, WE=1, WData=0x75, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x9d, WE=1, WData=0x26, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x8b, WE=1, WData=0xe5, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xfe, WE=0, WData=0x25, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x67, WE=1, WData=0x1a, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x77, WE=0, WData=0xec, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x12, WE=0, WData=0xbd, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x31, WE=1, WData=0xc7, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x1a, WE=0, WData=0xd2, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x22, WE=1, WData=0x99, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x22, WE=0, WData=0x77, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xc, WE=0, WData=0x6d, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xd6, WE=0, WData=0x87, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xa2, WE=0, WData=0x90, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x20, WE=1, WData=0x2f, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xc5, WE=1, WData=0xdf, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x1d, WE=1, WData=0xae, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xa7, WE=1, WData=0x5e, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x98, WE=0, WData=0x59, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xaf, WE=1, WData=0x28, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xd6, WE=0, WData=0xbf, RData=0x0
```

```
# KERNEL: [GEN SEND] @ 0: Addr=0x6f, WE=1, WData=0xdc, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0xfe, WE=0, WData=0xf5, RData=0x0
# KERNEL: [GEN SEND] @ 0: Addr=0x7c, WE=1, WData=0xe8, RData=0x0
# KERNEL: [GEN] Finished generating transactions.
# KERNEL:  Driver is running...
# KERNEL: [DRV RECV] @ 0: Addr=0xd3, WE=0, WData=0xad, RData=0x0
# KERNEL: [MON] Monitor is running...
# KERNEL:  Scoreboard is running...
# KERNEL: [DRV RECV] @ 15: Addr=0x9e, WE=1, WData=0xec, RData=0x0
# KERNEL: [MON CAPTURE] @ 25: Addr=0xd3, WE=0, WData=0x0, RData=0x0
# KERNEL: [SCB RECV] @ 25: Addr=0xd3, WE=0, WData=0x0, RData=0x0
# KERNEL:  PASS: Addr 0xd3, RData 0x0 matches expected default 0x00.
# KERNEL: [DRV RECV] @ 35: Addr=0x83, WE=1, WData=0xa2, RData=0x0
# KERNEL: [MON CAPTURE] @ 35: Addr=0x9e, WE=1, WData=0xec, RData=0x0
# KERNEL: [SCB RECV] @ 35: Addr=0x9e, WE=1, WData=0xec, RData=0x0
# KERNEL:  Write to Addr 0x9e with Data 0xec recorded.
# KERNEL: [DRV RECV] @ 55: Addr=0x59, WE=1, WData=0x5, RData=0x0
# KERNEL: [MON CAPTURE] @ 55: Addr=0x83, WE=1, WData=0xa2, RData=0x0
# KERNEL: [SCB RECV] @ 55: Addr=0x83, WE=1, WData=0xa2, RData=0x0
# KERNEL:  Write to Addr 0x83 with Data 0xa2 recorded.
# KERNEL: [DRV RECV] @ 75: Addr=0xf4, WE=0, WData=0x6e, RData=0x0
# KERNEL: [MON CAPTURE] @ 75: Addr=0x59, WE=1, WData=0x5, RData=0x0
# KERNEL: [SCB RECV] @ 75: Addr=0x59, WE=1, WData=0x5, RData=0x0
# KERNEL:  Write to Addr 0x59 with Data 0x5 recorded.
# KERNEL: [DRV RECV] @ 95: Addr=0x47, WE=0, WData=0xb3, RData=0x0
# KERNEL: [MON CAPTURE] @ 105: Addr=0xf4, WE=0, WData=0x0, RData=0x0
# KERNEL: [SCB RECV] @ 105: Addr=0xf4, WE=0, WData=0x0, RData=0x0
# KERNEL:  PASS: Addr 0xf4, RData 0x0 matches expected default 0x00.
# KERNEL: [DRV RECV] @ 115: Addr=0xec, WE=0, WData=0xac, RData=0x0
# KERNEL: [MON CAPTURE] @ 125: Addr=0x47, WE=0, WData=0x0, RData=0x0
# KERNEL: [SCB RECV] @ 125: Addr=0x47, WE=0, WData=0x0, RData=0x0
# KERNEL:  PASS: Addr 0x47, RData 0x0 matches expected default 0x00.
# KERNEL: [DRV RECV] @ 135: Addr=0xef, WE=1, WData=0x7a, RData=0x0
# KERNEL: [MON CAPTURE] @ 145: Addr=0xec, WE=0, WData=0x0, RData=0x0
# KERNEL: [SCB RECV] @ 145: Addr=0xec, WE=0, WData=0x0, RData=0x0
# KERNEL:  PASS: Addr 0xec, RData 0x0 matches expected default 0x00.
# KERNEL: [DRV RECV] @ 155: Addr=0x46, WE=1, WData=0xf9, RData=0x0
# KERNEL: [MON CAPTURE] @ 155: Addr=0xef, WE=1, WData=0x7a, RData=0x0
# KERNEL: [SCB RECV] @ 155: Addr=0xef, WE=1, WData=0x7a, RData=0x0
# KERNEL:  Write to Addr 0xef with Data 0x7a recorded.
# KERNEL: [DRV RECV] @ 175: Addr=0x7d, WE=1, WData=0xcd, RData=0x0
# KERNEL: [MON CAPTURE] @ 175: Addr=0x46, WE=1, WData=0xf9, RData=0x0
# KERNEL: [SCB RECV] @ 175: Addr=0x46, WE=1, WData=0xf9, RData=0x0
# KERNEL:  Write to Addr 0x46 with Data 0xf9 recorded.
# KERNEL: [DRV RECV] @ 195: Addr=0xa7, WE=1, WData=0xee, RData=0x0
# KERNEL: [MON CAPTURE] @ 195: Addr=0x7d, WE=1, WData=0xcd, RData=0x0
# KERNEL: [SCB RECV] @ 195: Addr=0x7d, WE=1, WData=0xcd, RData=0x0
# KERNEL:  Write to Addr 0x7d with Data 0xcd recorded.
# KERNEL: ------------------------------------------------------------
# KERNEL: [ENV] Test finished. Final Score: 4 Passed, 0 Failed.
# KERNEL: ------------------------------------------------------------
# RUNTIME: Info: RUNTIME_0068 testbench.sv (284): $finish called.
# KERNEL: Time: 200 ns,  Iteration: 0,  Instance: /test_top,  Process: @INITIAL#279_2@.
# KERNEL: stopped at time: 200 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./dump.vcd
[2025-10-17 05:47:17 UTC] Opening EPWave...
Done
```