```
//      APB-Based Single-Port RAM with FSM-Controlled Read/Write Access
//
//      EDAPLAYGROUND LINK: https://www.edaplayground.com/x/j96D

//      This project presents the design and implementation of a single-port
//      RAM module interfaced with the AMBA Advanced Peripheral Bus (APB) protocol
.
//      The APB RAM acts as a memory-mapped slave that can be accessed by an APB
//      master for read and write operations. The design supports 1024 memory loca
tions
//      (1K x 32-bit) with a simple state machine controlling bus transactions.

//      The RAM is implemented as a 1K x 32-bit memory array using SystemVerilog.
//      It is indexed using the lower 10 bits of the address bus (PADDR[9:0]).

//      Features:-
//       - Memory Depth: 1024 words (1K)
//       - Data Width: 32 bits
//       - Bus Protocol: AMBA APB (v2)
//       - Synchronous operation with respect to PCLK
//       - Support for both read and write operations
//       - Zero wait-state (always ready) with PREADY=1
//       - No error condition (PSLVERR=0)

//      A 4-state FSM controls the APB protocol:
//       • IDLE   - Waiting for transaction
//       • SETUP  - Address setup phase
//       • ACCESS - Data transfer phase (read/write)
//       • WAIT   - Optional wait cycles (not used, since PREADY=1)

================================================================================

=====------------------------------DESIGN  DUT---------------------------------=====
//design.sv

module apb_single_port_ram(
    // APB Interface signals
    input logic        PCLK,     // Clock input
    input logic        PRESETn,  // Active low reset
    input logic        PSEL,     // Peripheral select
    input logic        PENABLE,  // Enable signal
    input logic        PWRITE,   // Write control (1=Write, 0=Read)
    input logic [31:0] PADDR,    // Address bus
    input logic [31:0] PWDATA,   // Write data bus
    output logic       PREADY,   // Slave ready signal
    output logic [31:0] PRDATA,  // Read data bus
    output logic       PSLVERR   // Error response
);
    // Memory array: 1024 words, each 32-bits wide
    logic [31:0] mem [0:1023];  // 1K words of memory
    logic [31:0] tmp;           // Temporary storage for debug purposes

    // APB state machine definition
    typedef enum logic [1:0] {
        IDLE   = 2'b00,  // Idle state - waiting for transaction
        SETUP  = 2'b01,  // Setup state - address phase
        ACCESS = 2'b10,  // Access state - data phase
        WAIT   = 2'b11   // Wait state - for multi-cycle transactions
    } state_t;
```

```
    state_t present_state, next_state;  // State machine variables

    // State Transition Logic - Sequential part of state machine
    always_ff @(posedge PCLK or negedge PRESETn) begin
        if (!PRESETn) begin
            // Reset to IDLE state on active-low reset
            present_state <= IDLE;
        end else begin
            // Move to next state on clock edge
            present_state <= next_state;
        end
    end

    // Next State Logic - Combinational part of state machine
    always_comb begin
        // Default case to prevent latches
        next_state = IDLE;

        case (present_state)
            IDLE: begin
                // Move to SETUP when peripheral is selected
                if (PSEL) begin
                    next_state = SETUP;
                end else begin
                    next_state = IDLE;
                end
            end

            SETUP: begin
                // Move to ACCESS when PENABLE is asserted
                if (PENABLE) begin
                    next_state = ACCESS;
                end else begin
                    next_state = SETUP;
                end
            end

            ACCESS: begin
                if (PREADY) begin
                    // Transaction completed
                    if (!PSEL) begin
                        // If PSEL deasserted, go to IDLE
                        next_state = IDLE;
                    end else begin
                        // If PSEL still asserted, start new transaction
                        next_state = SETUP;
                    end
                end else begin
                    // If not ready, wait
                    next_state = WAIT;
                end
            end

            WAIT: begin
                // Stay in WAIT state until PREADY is asserted
                if (PREADY) begin
                    next_state = ACCESS;
                end else begin
```

```
                    next_state = WAIT;
                end
            end

            default: next_state = IDLE;
        endcase
    end

    // Memory Write Logic - Handles write operations to the memory
    always_ff @(posedge PCLK or negedge PRESETn) begin
        if (!PRESETn) begin
            // Reset temporary register on reset
            tmp <= 32'h0;
        end else if ((present_state == ACCESS) && PSEL && PENABLE && PWRITE) begi
n
            // Write operation: When in ACCESS state with PSEL, PENABLE active an
d PWRITE=1
            // Only use lower 10 bits of PADDR to address the 1024-entry memory
          mem[PADDR[9:0]] <= PWDATA;
            tmp <= PWDATA;  // Store in temp register for debug purposes

            // Debug messages for write operation
            //$display("\n[%0t]:WRITE:\n memory[%0h] = %0h\n PWDATA = %0h\n PWRIT
E = %0h",
            //        $time, PADDR[9:0], PWDATA, PWDATA, PWRITE);

            //$display("[%0t] Write: memory[%0h] = %0h", $time, PADDR[9:0], PWDAT
A);
        end
    end

    // Memory Read Logic - Handles read operations from the memory
    always_ff @(posedge PCLK or negedge PRESETn) begin
        if (!PRESETn) begin
            // Reset read data on reset
            PRDATA <= 32'h0;
        end else if ((present_state == ACCESS) && PSEL && PENABLE && !PWRITE) beg
in
            // Read operation: When in ACCESS state with PSEL, PENABLE active and
 PWRITE=0
            // Only use lower 10 bits of PADDR to address the 1024-entry memory
          PRDATA <= mem[PADDR[9:0]];

            // Debug messages for read operation
            //$display("\n[%0t]:READ:\n memory[%0h] = %0h\n PRDATA = %0h\n PWRITE
 = %0h",
            //        $time, PADDR[9:0], mem[PADDR[9:0]], mem[PADDR[9:0]], PWRITE
);
            //$display("tmp %0h, PRDATA %0h", tmp, mem[PADDR[9:0]]);
            //$display("[%0t] Read: PRDATA = %0h", $time, mem[PADDR[9:0]]);
        end
    end

    // PREADY and PSLVERR logic
    assign PSLVERR = 1'b0;  // Always indicate no error condition
    assign PREADY = 1'b1;   // Slave is always ready in this implementation
endmodule
================================================================================
```

```
=====------------------------------TB_TOP-------------------------------------=====
//testbench.sv

`include "interface.sv"
`include "uvm_macros.svh"//
`include "test_1.sv"

module ahb_ram_tb;

  bit clk;
  bit rst;

  always #5 clk = ~clk;

 initial begin
    rst = 0;
    #5 rst =1;
  end

  intf vif(clk, rst);

  apb_single_port_ram DUT(.PCLK(vif.clk),
                          .PRESETn(vif.rst),
                          .PSEL(vif.psel),
                          .PADDR(vif.paddr),
                          .PENABLE(vif.pen),
                          .PWRITE(vif.wr_en),
                          .PWDATA(vif.pwdata),
                          .PRDATA(vif.prdata),
                          .PREADY(vif.pready),
                          .PSLVERR(vif.pselverr)

                         );

 initial begin
    // set interface in config_db
    uvm_config_db#(virtual intf)::set(uvm_root::get(), "*", "vif", vif);
    // Dump waves
    $dumpfile("dump.vcd");
    $dumpvars(0);
  end

  initial begin
    run_test("test_1");
    #500;
    $finish();
  end

endmodule


================================================================================
```

```
=====------------------------------INTERFACE----------------------------------=====
// Interface.sv

interface intf(input logic clk, input logic rst);

  logic [31:0] paddr;
  logic wr_en;
  logic psel;
  logic pen;
  logic [31:0] pwdata;
  logic [31:0] prdata;
  logic pready;
  logic pselverr;

  clocking cb_drv @(posedge clk);
    default input #1 output #1;

     output paddr;
     output wr_en;
     output psel;
     output pen;
     output pwdata;

     input prdata;
     input pready;
     input pselverr;

  endclocking

    clocking cb_mon @(posedge clk);
    default input #1 output #1;

     input paddr;
     input wr_en;
     input psel;
     input pen;
     input pwdata;

     input prdata;
     input pready;
     input pselverr;

  endclocking

  modport DUT(clocking cb_drv, input clk, input rst);

  modport MON(clocking cb_mon, input clk, input rst);

endinterface

================================================================================
```

```
=====------------------------------SEQ_ITEM----------------------------------=====
//seq_item.sv

import uvm_pkg::*;
`include "uvm_macros.svh"

class seq_item extends uvm_sequence_item;

  randc bit psel;
  rand bit [31:0] paddr;
  rand bit pen;
  randc bit wr_en;
  rand bit [31:0] pwdata;
  bit [31:0] prdata;
  bit pready;
  bit pselverr;


  function new(string name="seq_item");
    super.new(name);
  endfunction

  `uvm_object_utils_begin(seq_item)

    `uvm_field_int(psel, UVM_ALL_ON)
    `uvm_field_int(paddr, UVM_ALL_ON)
    `uvm_field_int(pen, UVM_ALL_ON)
    `uvm_field_int(wr_en, UVM_ALL_ON)
    `uvm_field_int(pwdata, UVM_ALL_ON)
    `uvm_field_int(prdata, UVM_ALL_ON)
    `uvm_field_int(pready, UVM_ALL_ON)
  `uvm_field_int(pselverr, UVM_ALL_ON)

  `uvm_object_utils_end

  constraint A1 { //paddr[12:0] inside {[13'h0000:13'h1FFF]};
    paddr[12:0] inside {'h00ef,'h102f};

                // pwdata inside {[1111:'hFFFF]};
                              }

  constraint wr_en_c { soft wr_en == 1;
                       soft psel == 1;

                     }
endclass


================================================================================
```

```
=====------------------------------UVM_SEQUENCE_1--------------------------=====
//uvm_sequence.sv

`include "uvm_macros.svh"
import uvm_pkg::*;

class ram_sequence extends uvm_sequence#(seq_item);

  `uvm_object_utils(ram_sequence)

  seq_item txn;

  function new(string name ="ram_sequence");
    super.new(name);
  endfunction

  task body();
    txn = seq_item::type_id::create("txn");

    repeat(2) begin
      `uvm_do(txn);                                  // this one line is enough for wo
rk from start_item(req) to
                        //`uvm_info(get_type_name(),"Data send to driver",UVM_NON
E)     //`uvm_info("GENERATOR","Data send to driver",UVM_NONE)
                        // txn.print();
      $display("Created object of type :: SEQ 1.1 %s", txn.get_type_name());
     end

    repeat(2)  begin
      start_item(txn);

      if(!txn.randomize()) begin
        `uvm_error(get_type_name(), "!! Randomization failed !!");
      end
      else begin
        // $display( "WRITE_SEQ::  paddr=%0d, pwdata=%0d, wr_en=%0d, psel=%0d, pe
n=%0d, prdata=%0d\n",txn.paddr, txn.pwdata, txn.wr_en, txn.psel, txn.pen, txn.prd
ata);
        `uvm_info(get_type_name(), "!! Randomization success !!", UVM_HIGH);
        $display("Created object of type :: SEQ 1.2 %s", txn.get_type_name());
      end

      finish_item(txn);
      //#10;

    end
  endtask

endclass


================================================================================
```

```
=====------------------------------UVM_SEQUENCE_2-----------------------------=====
// sequence_2.sv
`include "uvm_macros.svh"
import uvm_pkg::*;

class write_sequence extends uvm_sequence#(seq_item);

  `uvm_object_utils(write_sequence)

  seq_item txn;

  function new(string name ="write_sequence");
    super.new(name);
  endfunction

  task body();
    txn = seq_item::type_id::create("txn");

    begin
      start_item(txn);
      txn.paddr = 32'hF12345;
      txn.pwdata = 32'hFAFAFA;
      txn.wr_en = 1'b1;
      txn.psel =1'b1;
       begin
         $display("Created object of type :: SEQ 2.1 %s", txn.get_type_name());
         $display( "WRITE_SEQ 2::  paddr=%0h, pwdata=%0h, wr_en=%0d, psel=%0d, pe
n=%0d, prdata=%0h\n",txn.paddr, txn.pwdata, txn.wr_en, txn.psel, txn.pen, txn.prd
ata);
         `uvm_info(get_type_name(), "!! Sequence 2 is selected !!", UVM_HIGH);
      end

      finish_item(txn);
    end
          begin
      `uvm_do(txn);
              //txn.print();
            $display("Created object of type :: SEQ 2.2 %s", txn.get_type_name()
);

            $display( "WRITE_SEQ 2.2 ::  paddr=%0h, pwdata=%0h, wr_en=%0d, psel=
%0d, pen=%0d, prdata=%0h\n",txn.paddr, txn.pwdata, txn.wr_en, txn.psel, txn.pen,
txn.prdata);
          end
  endtask

endclass


================================================================================
```

```
=====------------------------------UVM_SEQUENCE_3-----------------------------=====
// sequence_3.sv

`include "uvm_macros.svh"
import uvm_pkg::*;

class read_sequence extends uvm_sequence#(seq_item);

  `uvm_object_utils(read_sequence)
  seq_item txn;

  function new(string name ="read_sequence");
    super.new(name);
  endfunction

  task body();
    txn = seq_item::type_id::create("txn");
  fork
    begin
      start_item(txn);
      txn.paddr = 32'h101010;
      txn.pwdata = 32'h77777;
     // txn.wr_en = 1'b1;
     // txn.psel =1'b1;
       begin
         $display("Created object of type :: SEQ 3.1 %s", txn.get_type_name());
         $display( "WRITE_SEQ 3.1 ::  paddr=%0h, pwdata=%0h, wr_en=%0d, psel=%0d,
 pen=%0d, prdata=%0h\n",txn.paddr, txn.pwdata, txn.wr_en, txn.psel, txn.pen, txn.
prdata);
         `uvm_info(get_type_name(), "!! Sequence 2 is selected !!", UVM_HIGH);
       end

      finish_item(txn);

    end

       begin
      start_item(txn);
      txn.paddr = 32'hA1B1C2D3;
      txn.pwdata = 32'hE7D6D7D;

       begin
         $display("Created object of type :: SEQ 3.2 %s", txn.get_type_name());
         $display( "WRITE_SEQ 3.2 ::  paddr=%0h, pwdata=%0h, wr_en=%0d, psel=%0d,
 pen=%0d, prdata=%0h\n",txn.paddr, txn.pwdata, txn.wr_en, txn.psel, txn.pen, txn.
prdata);
         `uvm_info(get_type_name(), "!! Sequence 2 is selected !!", UVM_HIGH);
       end

      finish_item(txn);

    end
  join
 endtask
endclass

================================================================================
```

```
=====------------------------------VIRTUAL_SEQUENCE-------------------------=====
// v_sequence.sv

`include "uvm_macros.svh"
import uvm_pkg::*;

// Forward declarations - these classes should be defined elsewhere in your testb
ench
typedef class virtual_sequencer;
typedef class seq_item;
typedef class ram_sequence;
typedef class write_sequence;
typedef class read_sequence;

class virtual_sequence extends uvm_sequence#(seq_item);

  `uvm_object_utils(virtual_sequence)
  `uvm_declare_p_sequencer(virtual_sequencer)

  ram_sequence   seq1;
  write_sequence seq2;
  read_sequence  seq3;
event seq1_done, seq2_done;
  function new(string name = "virtual_sequence");
    super.new(name);
  endfunction

  virtual task body();
    seq1 = ram_sequence::type_id::create("seq1");
    seq2 = write_sequence::type_id::create("seq2");
    seq3 = read_sequence::type_id::create("seq3");

  fork
  begin : SEQ1_THREAD
    seq1.start(p_sequencer.vir_seqr);
    `uvm_info(get_type_name(), "Seq1 done", UVM_HIGH)
    -> seq1_done;   // trigger event when seq1 completes
  end

  begin : SEQ2_THREAD
    @seq1_done;     // wait until seq1 completes
    seq2.start(p_sequencer.vir_seqr);
    `uvm_info(get_type_name(), "Seq2 done", UVM_HIGH)
    -> seq2_done;   // trigger event when seq2 completes
  end

  begin : SEQ3_THREAD
    @seq2_done;     // wait until seq2 completes
    seq3.start(p_sequencer.vir_seqr);
    `uvm_info(get_type_name(), "Seq3 done", UVM_LOW)
  end
join_none
    wait fork;
    `uvm_info(get_type_name(), "All 3 sequences completed!", UVM_HIGH)
  endtask
endclass

================================================================================
```

```
=====------------------------------VIRTUAL_SEQUENCER-------------------------=====
// v_sequencer.sv

`include "uvm_macros.svh"
import uvm_pkg::*;

class virtual_sequencer extends uvm_sequencer#(seq_item);  // parmaterize with uv
m_sequence_item
                                                                                /
/ bcz it's a commen for multiple sequencer(multiple agents will have multiple seq
uencers) if there depends upon the tb infrasrtucture
  uvm_sequencer #(seq_item) vir_seqr;                       // handle of physical
sequencers


        `uvm_component_utils(virtual_sequencer)             // FR macrow

         function new(input string name = "virtual_sequencer", uvm_component pare
nt = null);// default constructor component class in
                super.new(name,parent);
         endfunction: new

endclass: virtual_sequencer


================================================================================




=====------------------------------UVM_DRIVER--------------------------------=====
// uvm_driver.sv

class ram_driver extends uvm_driver#(seq_item);

  `uvm_component_utils(ram_driver)

  virtual intf vif;
  seq_item txn;
  bit ok;

  function new(string name="ram_driver", uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info(get_type_name(), "RAM:: DRIVER", UVM_HIGH);
    //$display("Inside build phase= %s", get_type_name());

    ok = uvm_config_db#(virtual intf)::get(uvm_root::get(),"get_name()","vif", vi
f);

    if(!ok | vif == null) begin
      `uvm_fatal(get_type_name(), "DRV Not set at top level !! failed to get inte
rface");
```

```
    end else begin
      `uvm_info(get_type_name(), "Interface is found", UVM_HIGH);
    end

  endfunction

  task run_phase(uvm_phase phase);
     super.run_phase(phase);
    forever begin
      seq_item_port.get_next_item(txn);
        // `uvm_info(get_type_name, $sformatf("A = %0d, B = %0d", req.a, req.b),
UVM_LOW);
        reset();
        drive_write();
        drive_read();
      seq_item_port.item_done();


    end
  endtask

  task reset();
     @(posedge vif.DUT.clk) begin
      vif.DUT.rst <= 0;
         vif.DUT.cb_drv.psel <=0 ;
         vif.DUT.cb_drv.paddr <=0 ;
         vif.DUT.cb_drv.pen <=0 ;
         vif.DUT.cb_drv.wr_en <=0 ;
         vif.DUT.cb_drv.pwdata <=0 ;
         vif.DUT.cb_drv.prdata <=0 ;
         vif.DUT.cb_drv.pready <=0 ;
         vif.DUT.cb_drv.pselverr <=0 ;

       vif.DUT.rst <= 1;
     end

  endtask

  task drive_write();

    @(posedge vif.DUT.clk) begin

      //SETUP phase
      vif.DUT.cb_drv.psel <= 1;
      vif.DUT.cb_drv.pen <= 1;
    end

     @(posedge vif.DUT.clk) begin

      vif.DUT.cb_drv.wr_en <= 1;

      vif.DUT.cb_drv.paddr <= txn.paddr;
      vif.DUT.cb_drv.pwdata <= txn.pwdata;

    end

  endtask

    task drive_read();
```

```
    @(posedge vif.DUT.clk) begin

      //SETUP phase
      vif.DUT.cb_drv.psel <= 1;
      vif.DUT.cb_drv.pen <= 1;
    end

    @(posedge vif.DUT.clk) begin

       vif.DUT.cb_drv.wr_en <= 0;

       vif.DUT.cb_drv.paddr <= txn.paddr;
       txn.prdata = vif.DUT.cb_drv.prdata ;
      //$display("txn.prdata =%0h vif.DUT.cb_drv.prdata=%0h ",txn.prdata , vif.DU
T.cb_drv.prdata );

    end

  endtask

endclass
```

==========

```
=====------------------------------UVM_MONITOR-------------------------------=====
// uvm_monitor.sv

class ram_monitor extends uvm_monitor;

  `uvm_component_utils(ram_monitor)

  uvm_analysis_port #(seq_item) mon_port;

  virtual intf vif;

  seq_item mon_txn;

  bit ok;

  function new(string name="ram_monitor", uvm_component parent);
    super.new(name, parent);
    mon_port = new("mon_port", this);
    mon_txn = new();
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info(get_type_name(), "RAM:: MONITOR Inside build phase", UVM_HIGH);
    //$display("Inside build phase= %s", get_type_name());

    ok = uvm_config_db#(virtual intf)::get(uvm_root::get(),"get_name()","vif", vi
f);
```

```
    if(!ok || vif == null) begin
      `uvm_fatal(get_type_name(), "MON:: Not set at top level !! failed to get in
terface");
    end else begin
      `uvm_info(get_type_name(), "Interface is found", UVM_HIGH);
    end
  endfunction

  task run_phase(uvm_phase phase);
    forever begin
      @(posedge vif.MON.clk);

      // Valid APB access: only when transfer is completed
      if (vif.MON.cb_mon.psel && vif.MON.cb_mon.pen) begin
        mon_txn = seq_item::type_id::create("mon_txn", this);
        //$display("Created object of type MON:: %s", mon_txn.get_type_name());

        mon_txn.paddr     = vif.MON.cb_mon.paddr;
        mon_txn.pwdata    = vif.MON.cb_mon.pwdata;
        mon_txn.wr_en     = vif.MON.cb_mon.wr_en;
        mon_txn.psel      = vif.MON.cb_mon.psel;
        mon_txn.pen       = vif.MON.cb_mon.pen;
        mon_txn.pready    = vif.MON.cb_mon.pready;
        mon_txn.pselverr  = vif.MON.cb_mon.pselverr;

        // Wait 1 clk for valid PRDATA (especially for reads)
        if (!vif.MON.cb_mon.wr_en) begin
          @(posedge vif.MON.clk);
        //end

        mon_txn.prdata    = vif.MON.cb_mon.prdata;

        //$display("MON:: paddr=%0h, pwdata=%0h, prdata=%0h, wr_en=%0h, psel=%0h
pen=%0h, pready=%0h",
          //      mon_txn.paddr, mon_txn.pwdata, mon_txn.prdata, mon_txn.wr_en,
mon_txn.psel, mon_txn.pen, mon_txn.pready);
        end

        mon_port.write(mon_txn);
      end
    end
  endtask

endclass


================================================================================
```

```
=====------------------------------UVM_AGENT----------------------------------=====
// uvm_agent.sv

class ram_agent extends uvm_agent;

  `uvm_component_utils(ram_agent)

  uvm_sequencer#(seq_item) seqr;
  ram_driver drv;
  ram_monitor mon;

  function new(string name="ram_agent", uvm_component parent);
    super.new(name, parent);
  endfunction

  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info(get_type_name(), "RAM:: AGENT", UVM_HIGH);
    $display("Inside build phase= %s", get_type_name());

    mon = ram_monitor::type_id::create("mon", this);
    $display("Created object of type AGT :: %s", mon.get_type_name());

    if(get_is_active == UVM_ACTIVE) begin
      seqr = uvm_sequencer#(seq_item)::type_id::create("seqr", this);
      $display("Created object of type AGT :: %s", seqr.get_type_name());
      drv = ram_driver::type_id::create("drv", this);
      $display("Created object of type AGT:: %s", drv.get_type_name());

    end
  endfunction

  function void connect_phase(uvm_phase phase);


    if(get_is_active == UVM_ACTIVE) begin
      drv.seq_item_port.connect(seqr.seq_item_export) ;

    end

  endfunction

 // function run_phase(uvm_phase phase);
   // super.run_phase(phase);
 // endfunction

endclass


================================================================================
```

```
=====------------------------------UVM_ENVIRONMENT--------------------------=====
// uvm_env.sv

class ram_env extends uvm_env;

  ram_agent ragt;
  ram_scb rscb;
  virtual_sequencer v_seqr;
  //ram_sequencer ram_seqr;

  `uvm_component_utils(ram_env)

  //constructor
  function new(string name ="ram_env", uvm_component parent);
    super.new(name, parent);
    `uvm_info("ENV_const", "Inside Constructor!", UVM_HIGH)
  endfunction

  //build phase
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    `uvm_info(get_type_name(), "Inside Build_phase!", UVM_HIGH)
    //ram_seqr = ram_sequencer::type_id::create("ram_seqr", this);
    v_seqr = virtual_sequencer::type_id::create("v_seqr", this);
    $display("Created object of type:: ENV %s", v_seqr.get_type_name());
    // Create the agents and other components here
    ragt = ram_agent::type_id::create("ragt", this);
    $display("Created object of type:: ENV %s", ragt.get_type_name());
    rscb = ram_scb::type_id::create("rscb", this);
    $display("Created object of type:: ENV %s", rscb.get_type_name());

    // Ensure the objects are allocated before accessing them
    assert(ragt != null) else `uvm_fatal("ENV", "Failed to create ragt");
    assert(rscb != null) else `uvm_fatal("ENV", "Failed to create rscb");
  endfunction

  //connect phase
  function void connect_phase(uvm_phase phase);
    assert(ragt != null) else `uvm_fatal("ENV", "ragt not created during build_ph
ase");
    assert(rscb != null) else `uvm_fatal("ENV", "rscb not created during build_ph
ase");

    ragt.mon.mon_port.connect(rscb.scb_export);
    v_seqr.vir_seqr = ragt.seqr;
  endfunction

  function void end_of_elaboration_phase(uvm_phase phase);
  uvm_factory factory = uvm_factory::get();
  `uvm_info(get_type_name(), "Information printed from top_env::end_of_elaboratio
n_phase method", UVM_MEDIUM)
  `uvm_info(get_type_name(), $sformatf("Verbosity threshold is %d", get_report_ve
rbosity_level()), UVM_MEDIUM)
  uvm_top.print_topology();
  factory.print();
endfunction : end_of_elaboration_phase

endclass
================================================================================
```

```
=====----------------------------SCOREBOARD-----------------------------------=====
// uvm_scb.sv
//you can use uvm_analysis_imp instead of uvm_analysis_export + fifo if you want
to directly receive transactions into the scoreboard without using a FIFO.

class ram_scb extends uvm_scoreboard;

  `uvm_component_utils(ram_scb)

  // Direct analysis implementation port (monitor connects to this)
  uvm_analysis_imp #(seq_item, ram_scb) scb_export;

  // Reference model: sparse RAM of 32-bit data
  bit [31:0] mem_model [*];

  int check_count=1;

  //----------------------
  // Constructor
  //----------------------
  function new(string name = "ram_scb", uvm_component parent = null);
    super.new(name, parent);
  endfunction

  //----------------------
  // Build phase
  //----------------------
  function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    scb_export = new("scb_imp", this);
  endfunction

  //----------------------
  // write(): Called when monitor sends a txn
  //----------------------
  function void write(seq_item txn);
    if (txn.psel && txn.pen && txn.pready) begin
      if (txn.wr_en) begin
        // Write operation
        mem_model[txn.paddr] = txn.pwdata;
        //`uvm_info(get_type_name(),
         // $sformatf("T=%0t WRITE: Addr=0x%0h Data=0x%0h", $time, txn.paddr, txn
.pwdata), UVM_LOW);
      end
      else begin
        // Read operation
        if (!mem_model.exists(txn.paddr)) begin
          //`uvm_warning(get_type_name(),
                //       $sformatf("READ BEFORE WRITE @ Addr=0x%0h: Actual=0x%0h
mem_model[txn.paddr]=%0h  mem_model.exists(txn.paddr)=%0d", txn.paddr, txn.prdata
,mem_model[txn.paddr], mem_model.exists(txn.paddr)));
        end
        else if (txn.prdata !== mem_model[txn.paddr]) begin
          `uvm_error(get_type_name(),
                     $sformatf("READ MISMATCH @ Addr=0x%0h: Expected=0x%0h, Got=0
x%0h\n",
```

```
                txn.paddr, mem_model[txn.paddr], txn.prdata));
          end
          else begin
            `uvm_info(get_type_name(),
                      $sformatf("T=%0t <<< READ MATCH >>> Addr=0x%0h: Expected=0x%0
h, Got=0x%0h\n",
                $time, txn.paddr, mem_model[txn.paddr], txn.prdata), UVM_LOW);
            check_count++;
          end

        end
      end
    endfunction

    //----------------------
    // report_phase(): Print summary
    //----------------------
    function void report_phase(uvm_phase phase);
      super.report_phase(phase);
      `uvm_info(get_type_name(),
        $sformatf("RAM SCB: Total Read Checks = %0d", check_count), UVM_NONE);
    endfunction

endclass

================================================================================

=====------------------------------PACKAGE-----------------------------------=====
//package.sv

package ram_env_pkg;

`include "seq_item.sv"
`include "uvm_sequence.sv"
`include "sequence_2.sv"
`include "sequence_3.sv"
`include "v_sequence.sv"
`include "v_sequencer.sv"
`include "uvm_driver.sv"
`include "uvm_monitor.sv"
`include "uvm_scb.sv"
`include "uvm_agent.sv"
`include "uvm_env.sv"

endpackage

================================================================================
```

```
=====------------------------------TEST 1-------------------------------------=====
//test_1.sv

`include "uvm_macros.svh"
import uvm_pkg::*;
`include "package.sv"
import ram_env_pkg::*;

//==============================================================================
// BASE TEST CLASS - Common functionality for all tests
//==============================================================================
class test_1 extends uvm_test;
  `uvm_component_utils(test_1)

  // Environment handle
  ram_env env;

  // Virtual sequence handle
  virtual_sequence v_seq;

  // Test configuration parameters
  protected int num_iterations = 1;

  function new(string name = "test_1", uvm_component parent);
    super.new(name, parent);
  endfunction

  // Build phase - Create environment
  virtual function void build_phase(uvm_phase phase);
    super.build_phase(phase);

    // Create environment
    env = ram_env::type_id::create("ram_env", this);
    `uvm_info("BASE_TEST", $sformatf("Created environment of type: %s", env.get_t
ype_name()), UVM_LOW)
  endfunction

  // Connect phase - Make connections
  virtual function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    `uvm_info("BASE_TEST", "Connect Phase completed!", UVM_LOW)
  endfunction

  // End of elaboration phase - Print topology
  virtual function void end_of_elaboration_phase(uvm_phase phase);
    super.end_of_elaboration_phase(phase);
    `uvm_info("BASE_TEST", "Printing testbench topology:", UVM_LOW)
    uvm_top.print_topology();
  endfunction

  // Run phase - Base implementation
  virtual task run_phase(uvm_phase phase);
    super.run_phase(phase);

    // Raise objection
    phase.raise_objection(this);

    `uvm_info("BASE_TEST", "Starting base test run phase", UVM_MEDIUM)
```

```
    // Create virtual sequence
    v_seq = virtual_sequence::type_id::create("v_seq");

    // Execute test-specific sequence
   `uvm_info("BASE_TEST", "Executing default sequence", UVM_MEDIUM)
    repeat(num_iterations) begin
      v_seq.start(env.v_seqr);
    end

    // Add some settling time
    #100;

    // Drop objection
    phase.drop_objection(this);
    `uvm_info(get_type_name(), " > > > > > Test completed successfully < < < < < 
", UVM_LOW)
  endtask

endclass: test_1

================================================================================




=====------------------------------RESULT-------------------------------------=====


[2025-09-05 11:51:58 UTC] vlib work && vlog '-timescale' '1ns/1ns' +incdir+$RIVIE
RA_HOME/vlib/uvm-1.2/src -l uvm_1_2 -err VCP2947 W9 -err VCP2974 W9 -err VCP3003
W9 -err VCP5417 W9 -err VCP6120 W9 -err VCP7862 W9 -err VCP2129 W9 design.sv test
bench.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_SP VCP2124 "Package uvm_pkg found in library uvm_1_2."
MESSAGE "Unit top modules: ahb_ram_tb."
SUCCESS "Compile success 0 Errors 0 Warnings  Analysis time: 6[s]."
done
# Aldec, Inc. Riviera-PRO version 2023.04.112.8911 built for Linux64 on May 12, 2
023.
# HDL, SystemC, and Assertions simulator, debugger, and design environment.
# (c) 1999-2023 Aldec, Inc. All rights reserved.
# ELBREAD: Elaboration process.
# ELBREAD: Warning: ELBREAD_0049 Package 'uvm_pkg' does not have a `timescale dir
ective, but previous modules do.
# ELBREAD: Elaboration time 0.7 [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: Info: Loading library:  /usr/share/Riviera-PRO/bin/uvm_1_2_dpi
# 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 : 0.1 [s]
# SLP: Finished : 0.1 [s]
# SLP: 0 primitives and 5 (33.33%) other processes in SLP
# SLP: 25 (0.08%) signals in SLP and 105 (0.34%) interface signals
# ELAB2: Elaboration final pass complete - time: 2.8 [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 30149 kB (elbread=2110 elab2=23049 kernel=4989
sdf=0)
# KERNEL: UVM_INFO /home/build/vlib1/vlib/uvm-1.2/src/base/uvm_root.svh(392) @ 0:
 reporter [UVM/RELNOTES]
# KERNEL: ----------------------------------------------------------------
# KERNEL: UVM-1.2
# KERNEL: (C) 2007-2014 Mentor Graphics Corporation
# KERNEL: (C) 2007-2014 Cadence Design Systems, Inc.
# KERNEL: (C) 2006-2014 Synopsys, Inc.
# KERNEL: (C) 2011-2013 Cypress Semiconductor Corp.
# KERNEL: (C) 2013-2014 NVIDIA Corporation
# KERNEL: ----------------------------------------------------------------
# KERNEL:
# KERNEL:   ***********       IMPORTANT RELEASE NOTES         ************
# KERNEL:
# KERNEL:   You are using a version of the UVM library that has been compiled
# KERNEL:   with `UVM_NO_DEPRECATED undefined.
# KERNEL:   See http://www.eda.org/svdb/view.php?id=3313 for more details.
# KERNEL:
# KERNEL:   You are using a version of the UVM library that has been compiled
# KERNEL:   with `UVM_OBJECT_DO_NOT_NEED_CONSTRUCTOR undefined.
# KERNEL:   See http://www.eda.org/svdb/view.php?id=3770 for more details.
# KERNEL:
# KERNEL:       (Specify +UVM_NO_RELNOTES to turn off this notice)
# KERNEL:
# KERNEL: ASDB file was created in location /home/runner/dataset.asdb
# KERNEL: UVM_INFO @ 0: reporter [RNTST] Running test test_1...
# KERNEL: UVM_INFO /home/runner/test_1.sv(31) @ 0: uvm_test_top [BASE_TEST] Creat
ed environment of type: ram_env
# KERNEL: Created object of type:: ENV virtual_sequencer
# KERNEL: Created object of type:: ENV ram_agent
# KERNEL: Created object of type:: ENV ram_scb
# KERNEL: Inside build phase= ram_agent
# KERNEL: Created object of type AGT :: ram_monitor
# KERNEL: Created object of type AGT :: uvm_sequencer
# KERNEL: Created object of type AGT:: ram_driver
# KERNEL: UVM_INFO /home/runner/test_1.sv(37) @ 0: uvm_test_top [BASE_TEST] Conne
ct Phase completed!
# KERNEL: UVM_INFO /home/runner/uvm_env.sv(46) @ 0: uvm_test_top.ram_env [ram_env
] Information printed from top_env::end_of_elaboration_phase method
# KERNEL: UVM_INFO /home/runner/uvm_env.sv(47) @ 0: uvm_test_top.ram_env [ram_env
] Verbosity threshold is         200
# KERNEL: UVM_INFO /home/build/vlib1/vlib/uvm-1.2/src/base/uvm_root.svh(583) @ 0:
 reporter [UVMTOP] UVM testbench topology:
# KERNEL: ----------------------------------------------------------------
# KERNEL: Name                       Type                     Size  Value
```

```
# KERNEL: ----------------------------------------------------------------
# KERNEL: uvm_test_top                test_1                    -      @340
# KERNEL:   ram_env                   ram_env                   -      @353
# KERNEL:     ragt                    ram_agent                 -      @501
# KERNEL:       drv                   ram_driver                -      @679
# KERNEL:         rsp_port            uvm_analysis_port         -      @698
# KERNEL:         seq_item_port       uvm_seq_item_pull_port    -      @688
# KERNEL:       mon                   ram_monitor               -      @520
# KERNEL:         mon_port            uvm_analysis_port         -      @529
# KERNEL:       seqr                  uvm_sequencer             -      @542
# KERNEL:         rsp_export          uvm_analysis_export       -      @551
# KERNEL:         seq_item_export     uvm_seq_item_pull_imp     -      @669
# KERNEL:         arbitration_queue   array                     0      -
# KERNEL:         lock_queue          array                     0      -
# KERNEL:         num_last_reqs       integral                  32     'd1
# KERNEL:         num_last_rsps       integral                  32     'd1
# KERNEL:     rscb                    ram_scb                   -      @510
# KERNEL:       scb_imp               uvm_analysis_imp          -      @715
# KERNEL:     v_seqr                  virtual_sequencer         -      @364
# KERNEL:       rsp_export            uvm_analysis_export       -      @373
# KERNEL:       seq_item_export       uvm_seq_item_pull_imp     -      @491
# KERNEL:       arbitration_queue     array                     0      -
# KERNEL:       lock_queue            array                     0      -
# KERNEL:       num_last_reqs         integral                  32     'd1
# KERNEL:       num_last_rsps         integral                  32     'd1
# KERNEL: ----------------------------------------------------------------
# KERNEL:
# KERNEL: UVM_INFO /home/build/vlib1/vlib/uvm-1.2/src/base/uvm_factory.svh(1645)
@ 0: reporter [UVM/FACTORY/PRINT]
# KERNEL: #### Factory Configuration (*)
# KERNEL:
# KERNEL:   No instance or type overrides are registered with this factory
# KERNEL:
# KERNEL: All types registered with the factory: 66 total
# KERNEL:   Type Name
# KERNEL:   ---------
# KERNEL:   ram_agent
# KERNEL:   ram_driver
# KERNEL:   ram_env
# KERNEL:   ram_monitor
# KERNEL:   ram_scb
# KERNEL:   ram_sequence
# KERNEL:   read_sequence
# KERNEL:   seq_item
# KERNEL:   test_1
# KERNEL:   virtual_sequence
# KERNEL:   virtual_sequencer
# KERNEL:   write_sequence
# KERNEL: (*) Types with no associated type name will be printed as <unknown>
# KERNEL:
# KERNEL: ####
# KERNEL:
# KERNEL:
# KERNEL: UVM_INFO /home/runner/test_1.sv(43) @ 0: uvm_test_top [BASE_TEST] Print
ing testbench topology:
# KERNEL: UVM_INFO /home/build/vlib1/vlib/uvm-1.2/src/base/uvm_root.svh(583) @ 0:
 reporter [UVMTOP] UVM testbench topology:
# KERNEL: ----------------------------------------------------------------
# KERNEL: Name                        Type                      Size  Value
```

```
# KERNEL: ----------------------------------------------------------------
# KERNEL: uvm_test_top                test_1                    -     @340
# KERNEL:   ram_env                   ram_env                   -     @353
# KERNEL:     ragt                    ram_agent                 -     @501
# KERNEL:       drv                   ram_driver                -     @679
# KERNEL:         rsp_port            uvm_analysis_port         -     @698
# KERNEL:         seq_item_port       uvm_seq_item_pull_port    -     @688
# KERNEL:       mon                   ram_monitor               -     @520
# KERNEL:         mon_port            uvm_analysis_port         -     @529
# KERNEL:       seqr                  uvm_sequencer             -     @542
# KERNEL:         rsp_export          uvm_analysis_export       -     @551
# KERNEL:         seq_item_export     uvm_seq_item_pull_imp     -     @669
# KERNEL:         arbitration_queue   array                     0     -
# KERNEL:         lock_queue          array                     0     -
# KERNEL:         num_last_reqs       integral                  32    'd1
# KERNEL:         num_last_rsps       integral                  32    'd1
# KERNEL:     rscb                    ram_scb                   -     @510
# KERNEL:       scb_imp               uvm_analysis_imp          -     @715
# KERNEL:     v_seqr                  virtual_sequencer         -     @364
# KERNEL:       rsp_export            uvm_analysis_export       -     @373
# KERNEL:       seq_item_export       uvm_seq_item_pull_imp     -     @491
# KERNEL:       arbitration_queue     array                     0     -
# KERNEL:       lock_queue            array                     0     -
# KERNEL:       num_last_reqs         integral                  32    'd1
# KERNEL:       num_last_rsps         integral                  32    'd1
# KERNEL: ----------------------------------------------------------------
# KERNEL:
# KERNEL: UVM_INFO /home/runner/test_1.sv(54) @ 0: uvm_test_top [BASE_TEST] Start
ing base test run phase
# KERNEL: UVM_INFO /home/runner/test_1.sv(60) @ 0: uvm_test_top [BASE_TEST] Execu
ting default sequence
# KERNEL: Created object of type :: SEQ 1.1 seq_item
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 75: uvm_test_top.ram_env.rscb [r
am_scb] T=75 <<< READ MATCH >>> Addr=0xf90460ef: Expected=0xbd257066, Got=0xbd257
066
# KERNEL:
# KERNEL: Created object of type :: SEQ 1.1 seq_item
# KERNEL: Created object of type :: SEQ 1.2 seq_item
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 125: uvm_test_top.ram_env.rscb [
ram_scb] T=125 <<< READ MATCH >>> Addr=0xfc45302f: Expected=0xd195b843, Got=0xd19
5b843
# KERNEL:
# KERNEL: Created object of type :: SEQ 1.2 seq_item
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 175: uvm_test_top.ram_env.rscb [
ram_scb] T=175 <<< READ MATCH >>> Addr=0xfedf60ef: Expected=0xaebc5827, Got=0xaeb
c5827
# KERNEL:
# KERNEL: Created object of type :: SEQ 2.1 seq_item
# KERNEL: WRITE_SEQ 2::  paddr=f12345, pwdata=fafafa, wr_en=1, psel=1, pen=0, prd
ata=0
# KERNEL:
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 225: uvm_test_top.ram_env.rscb [
ram_scb] T=225 <<< READ MATCH >>> Addr=0x220502f: Expected=0xc32ca004, Got=0xc32c
a004
# KERNEL:
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 275: uvm_test_top.ram_env.rscb [
ram_scb] T=275 <<< READ MATCH >>> Addr=0xf12345: Expected=0xfafafa, Got=0xfafafa
# KERNEL:
# KERNEL: Created object of type :: SEQ 2.2 seq_item
```

```
# KERNEL: WRITE_SEQ 2.2 ::  paddr=6ee1b02f, pwdata=ca6ffb69, wr_en=1, psel=1, pen
=0, prdata=fafafa
# KERNEL:
# KERNEL: Created object of type :: SEQ 3.1 seq_item
# KERNEL: WRITE_SEQ 3.1 ::  paddr=101010, pwdata=77777, wr_en=0, psel=0, pen=0, p
rdata=0
# KERNEL:
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 325: uvm_test_top.ram_env.rscb [
ram_scb] T=325 <<< READ MATCH >>> Addr=0x6ee1b02f: Expected=0xca6ffb69, Got=0xca6
ffb69
# KERNEL:
# KERNEL: Created object of type :: SEQ 3.2 seq_item
# KERNEL: WRITE_SEQ 3.2 ::  paddr=a1b1c2d3, pwdata=e7d6d7d, wr_en=0, psel=0, pen=
0, prdata=ca6ffb69
# KERNEL:
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 375: uvm_test_top.ram_env.rscb [
ram_scb] T=375 <<< READ MATCH >>> Addr=0x101010: Expected=0x77777, Got=0x77777
# KERNEL:
# KERNEL: UVM_INFO /home/runner/v_sequence.sv(54) @ 395: uvm_test_top.ram_env.v_s
eqr@@v_seq [virtual_sequence] Seq3 done
# KERNEL: UVM_ERROR /home/runner/uvm_scb.sv(51) @ 425: uvm_test_top.ram_env.rscb
[ram_scb] READ MISMATCH @ Addr=0xa1b1c2d3: Expected=0xe7d6d7d, Got=0x77777
# KERNEL:
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 445: uvm_test_top.ram_env.rscb [
ram_scb] T=445 <<< READ MATCH >>> Addr=0xa1b1c2d3: Expected=0xe7d6d7d, Got=0xe7d6
d7d
# KERNEL:
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 465: uvm_test_top.ram_env.rscb [
ram_scb] T=465 <<< READ MATCH >>> Addr=0xa1b1c2d3: Expected=0xe7d6d7d, Got=0xe7d6
d7d
# KERNEL:
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(56) @ 485: uvm_test_top.ram_env.rscb [
ram_scb] T=485 <<< READ MATCH >>> Addr=0xa1b1c2d3: Expected=0xe7d6d7d, Got=0xe7d6
d7d
# KERNEL:
# KERNEL: UVM_INFO /home/runner/test_1.sv(70) @ 495: uvm_test_top [test_1]  > > >
 > > Test completed successfully < < < < <
# KERNEL: UVM_INFO /home/build/vlib1/vlib/uvm-1.2/src/base/uvm_objection.svh(1271
) @ 495: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' ph
ase
# KERNEL: UVM_INFO /home/runner/uvm_scb.sv(70) @ 495: uvm_test_top.ram_env.rscb [
ram_scb] RAM SCB: Total Read Checks = 11
# KERNEL: UVM_INFO /home/build/vlib1/vlib/uvm-1.2/src/base/uvm_report_server.svh(
869) @ 495: reporter [UVM/REPORT/SERVER]
# KERNEL: --- UVM Report Summary ---
# KERNEL:
# KERNEL: ** Report counts by severity
# KERNEL: UVM_INFO :   26
# KERNEL: UVM_WARNING :    0
# KERNEL: UVM_ERROR :    1
# KERNEL: UVM_FATAL :    0
# KERNEL: ** Report counts by id
# KERNEL: [BASE_TEST]     5
# KERNEL: [RNTST]     1
# KERNEL: [TEST_DONE]     1
# KERNEL: [UVM/FACTORY/PRINT]     1
# KERNEL: [UVM/RELNOTES]     1
# KERNEL: [UVMTOP]     2
# KERNEL: [ram_env]     2
```

```
# KERNEL: [ram_scb]     12
# KERNEL: [test_1]      1
# KERNEL: [virtual_sequence]      1
# KERNEL:
# RUNTIME: Info: RUNTIME_0068 uvm_root.svh (521): $finish called.
# KERNEL: Time: 495 ns,  Iteration: 57,  Instance: /ahb_ram_tb,  Process: @INITIA
L#44_3@.
# KERNEL: stopped at time: 495 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# VSIM: Simulation has finished.
Finding VCD file...
./dump.vcd
[2025-09-05 11:52:12 UTC] Opening EPWave...
Done
```