**SPI with Single Port RAM Verification Project**

|  |  |
| --- | --- |
| Group number |  |
| 3 | كريم أيمن محمود يحيى |
| 2 | كريم حسن عاطف |
| 2 | Ahmed Mohamed Abdelmoneam |

***Spi Slave part :***

1. **Spi slave UVM architecture**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Comment :** first top set the virtual interface in configuration data base ,then test get this virtual interface and set configuration data base with all interface then agent get this configuration object and use it to connect between driver and monitor with the interface of DUT , Driver ask sequencer if there is data from sequence item , if there is data take data and give it to interface then monitor get data from interface and give it to scoreboard to compare it with refrence model by using fifo to get sequence item and give it to scoreboard , the same way for collector get sequence item and cover if sequence done , then after the simulation done test raise object and end simulation .

1. **Design file**

module SLAVE (spi\_slave\_if.DUT ss\_if);

localparam IDLE      = 3'b000;

localparam WRITE     = 3'b001;

localparam CHK\_CMD   = 3'b010;

localparam READ\_ADD  = 3'b011;

localparam READ\_DATA = 3'b100;

reg [3:0] counter;

reg       received\_address;

reg [2:0] cs, ns;

always @(posedge ss\_if.clk) begin

    if (~ss\_if.rst\_n) begin

        cs <= IDLE;

    end

    else begin

        cs <= ns;

    end

end

always @(\*) begin

    case (cs)

        IDLE : begin

            if (ss\_if.SS\_n)

                ns = IDLE;

            else

                ns = CHK\_CMD;

        end

        CHK\_CMD : begin

            if (ss\_if.SS\_n)

                ns = IDLE;

            else begin

                if (~ss\_if.MOSI)

                    ns = WRITE;

                else begin

                      if (received\_address)

                        ns = READ\_DATA;

                    else

                        ns = READ\_ADD;

                end

            end

        end

        WRITE : begin

            if (ss\_if.SS\_n)

                ns = IDLE;

            else

                ns = WRITE;

        end

        READ\_ADD : begin

            if (ss\_if.SS\_n)

                ns = IDLE;

            else

                ns = READ\_ADD;

        end

        READ\_DATA : begin

            if (ss\_if.SS\_n)

                ns = IDLE;

            else

                ns = READ\_DATA;

        end

    endcase

end

always @(posedge ss\_if.clk) begin

    if (~ss\_if.rst\_n) begin

        ss\_if.rx\_data <= 0;

        ss\_if.rx\_valid <= 0;

        received\_address <= 0;

        ss\_if.MISO <= 0;

        counter<=0; // reset

    end

    else begin

        case (cs)

            IDLE : begin

                ss\_if.rx\_valid <= 0;

            end

            CHK\_CMD : begin

                counter <= 10;

            end

            WRITE : begin

                if (counter > 0) begin

                    ss\_if.rx\_data[counter-1] <= ss\_if.MOSI;

                    counter <= counter - 1;

                end

                else begin

                    ss\_if.rx\_valid <= 1;

                end

            end

            READ\_ADD : begin

                if (counter > 0) begin

                    ss\_if.rx\_data[counter-1] <= ss\_if.MOSI;

                    counter <= counter - 1;

                end

                else begin

                    ss\_if.rx\_valid <= 1;

                    received\_address <= 1;

                end

            end

            READ\_DATA : begin

                if (ss\_if.tx\_valid) begin

                    if (counter > 0) begin

                        ss\_if.MISO <= ss\_if.tx\_data[counter-1];

                        counter <= counter - 1;

                    end

                    else begin

                        received\_address <= 0;

                         ss\_if.rx\_valid <= 0;

                    end

                end

                else begin

                    if (counter > 0) begin // write data before going to read data

                        ss\_if.rx\_data[counter-1] <= ss\_if.MOSI;

                        counter <= counter - 1;

                    end

                    else begin

                        ss\_if.rx\_valid <= 1;

                        counter <= 8;

                    end

                end

            end

        endcase

    end

end

// READ\_ADD and READ\_DATA is replaced

// counter not equal 0 at reset

 `ifdef SIM

*sequence write\_add\_seq;*

*(ss\_if.SS\_n==1) ##1 (ss\_if.SS\_n==0) ##1 (ss\_if.MOSI == 0)[\*3];*

*endsequence*

*sequence write\_data\_seq;*

*(ss\_if.SS\_n==1) ##1 (ss\_if.SS\_n==0) ##1 (ss\_if.MOSI == 0)[\*2] ##1(ss\_if.MOSI==1);*

*endsequence*

*sequence read\_add\_seq;*

*(ss\_if.SS\_n==1) ##1 (ss\_if.SS\_n==0) ##1 (ss\_if.MOSI == 1)[\*2] ##1(ss\_if.MOSI==0);*

*endsequence*

*sequence read\_data\_seq;*

*(ss\_if.SS\_n==1) ##1 (ss\_if.SS\_n==0) ##1 (ss\_if.MOSI == 1)[\*3];*

*endsequence*

*property chck\_rx\_valid;*

*@(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n)*

*(write\_add\_seq or write\_data\_seq or read\_add\_seq or read\_data\_seq) |=> ##9 ($rose(ss\_if.rx\_valid) && $rose(ss\_if.SS\_n)[->1]);*

*endproperty*

*property chck\_reset ;*

*@(posedge ss\_if.clk)   (~ss\_if.rst\_n) |=>(ss\_if.MISO ==0 && ss\_if.rx\_valid==0 && ss\_if.rx\_data==0);*

*endproperty*

*property chck\_state\_idle;*

*@(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n) (cs==IDLE && !ss\_if.SS\_n) |=>(cs==CHK\_CMD);*

*endproperty*

*property chck\_state\_write;*

*@(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n) (cs==CHK\_CMD && !ss\_if.SS\_n && !ss\_if.MOSI) |=>  (cs==WRITE);*

*endproperty*

*property chck\_state\_read\_add;*

*@(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n)  (cs==CHK\_CMD && !ss\_if.SS\_n && ss\_if.MOSI && !received\_address) |=>(cs==READ\_ADD);*

*endproperty*

*property chck\_state\_read\_data;*

*@(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n)  (cs==CHK\_CMD && !ss\_if.SS\_n && ss\_if.MOSI && received\_address) |=>(cs==READ\_DATA);*

*endproperty*

*property chck\_state\_write\_to\_idle;*

*@(posedge ss\_if.clk)  (cs==WRITE &&(~ss\_if.rst\_n)) |=> (cs==IDLE);*

*endproperty*

*property chck\_state\_read\_add\_to\_idle;*

*@(posedge ss\_if.clk)  (cs==READ\_ADD && (~ss\_if.rst\_n)) |=> (cs==IDLE);*

*endproperty*

*property chck\_state\_read\_datato\_idle;*

*@(posedge ss\_if.clk)  (cs==READ\_DATA && (~ss\_if.rst\_n)) |=> (cs==IDLE);*

*endproperty*

*assert property (chck\_reset);*

*assert property (chck\_rx\_valid);*

*assert property (chck\_state\_idle);*

*assert property (chck\_state\_write);*

*assert property (chck\_state\_read\_add);*

*assert property (chck\_state\_read\_data);*

*assert property (chck\_state\_write\_to\_idle);*

*assert property (chck\_state\_read\_add\_to\_idle);*

*assert property (chck\_state\_read\_datato\_idle);*

*cover property (chck\_reset);*

*cover property (chck\_rx\_valid);*

*cover property (chck\_state\_idle);*

*cover property (chck\_state\_write);*

*cover property (chck\_state\_read\_add);*

*cover property (chck\_state\_read\_data);*

*cover property (chck\_state\_write\_to\_idle);*

*cover property (chck\_state\_read\_add\_to\_idle);*

*cover property (chck\_state\_read\_datato\_idle);*

`endif

Endmodule

1. **Design Bugs**

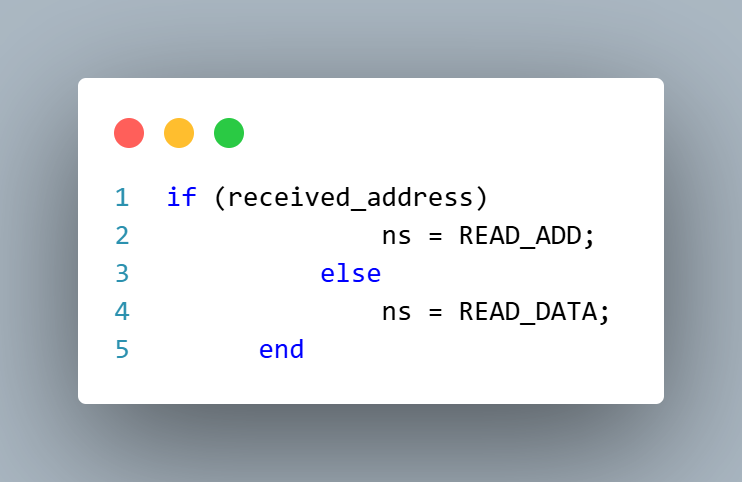
****

Figure bug 1

A white background with black text

AI-generated content may be incorrect.

Figure bug1 fixed

***A computer code with green and blue text

AI-generated content may be incorrect.***

Figure bug 2

***A computer code with text

AI-generated content may be incorrect.***

Figure bug2 fixed

***A computer code with text

AI-generated content may be incorrect.***

Figure bug 3

***A computer code with text

AI-generated content may be incorrect.***

Figure bug 3 fixed

***A white background with black text

AI-generated content may be incorrect.***

Figure bug 4

***A computer code on a white background

AI-generated content may be incorrect.***

Figure bug4 fixed

1. **Refrence model file**

module SPI\_Slave(MOSI,SS\_n,clk,rst\_n,rx\_data,tx\_valid,tx\_data,MISO,rx\_valid);

input MOSI,SS\_n,clk,rst\_n,tx\_valid;

input [7:0] tx\_data;

output reg MISO,rx\_valid;

output  reg [9:0] rx\_data;

parameter IDLE=3'b000;

parameter CHK\_CMD=3'b001;

parameter WRITE=3'b010;

parameter READ\_ADD=3'b011;

parameter  READ\_DATA=3'b100;

reg ADDRESS\_read; // this is signal to increment when  reading an address

reg[3:0] counter;

reg [2:0] cs ,ns;

always@(posedge clk) begin

    if(~rst\_n)

     cs<=IDLE;

     else

     cs<=ns;

end

always@(\*) begin

    case(cs)

    IDLE: begin

        if(SS\_n)

        ns=IDLE;

        else

        ns=CHK\_CMD;

    end

   CHK\_CMD : begin

    if (SS\_n)

    ns = IDLE;

    else begin

     if( ~MOSI) begin

        ns=WRITE;

    end

    else begin

        casex(ADDRESS\_read)

        1'b0 : ns=READ\_ADD;

        1'b1: ns=READ\_DATA;

        1'bx: ns=READ\_ADD;

        endcase

    end

   end

   end

   WRITE: begin

     if(SS\_n==0 )

     ns=WRITE;

     else begin

        ns=IDLE;

     end

   end

   READ\_ADD :  begin

    if(SS\_n==0) begin

    ns=READ\_ADD;

    end

    else begin

        ns=IDLE;

    end

   end

   READ\_DATA : begin

    if(SS\_n==0)

    ns=READ\_DATA;

    else begin

        ns=IDLE;

    end

   end

    endcase

end

always @(posedge clk) begin

      if (~rst\_n) begin

        rx\_data <= 0;

        rx\_valid <= 0;

        ADDRESS\_read<= 0;

        MISO <= 0;

    end

    else begin

   case(cs)

    IDLE: rx\_valid<=0;

    CHK\_CMD : begin

         counter<=10;

    end

    WRITE  : begin

        if(counter>0) begin

        rx\_data[counter-1]<=MOSI;

        counter<=counter-1;

        end

        else begin

            rx\_valid<=1;

        end

    end

    READ\_ADD : begin

         if(counter>0) begin

        rx\_data[counter-1]<=MOSI;

        counter<=counter-1;

        end

        else begin

            rx\_valid<=1;

            ADDRESS\_read<=1;

        end

    end

   READ\_DATA : begin

    if (tx\_valid) begin

        if(counter>0) begin

            MISO<=tx\_data[counter-1];

             counter<=counter-1;

        end

         else begin

                ADDRESS\_read <= 0;

                rx\_valid<=0;

         end

    end

    else begin

       if(counter>0) begin

        rx\_data[counter-1]<=MOSI;

        counter<=counter-1;

       end

       else begin

        rx\_valid<=1;

        counter<=9;

       end

    end

   end

   endcase

    end

end

endmodule

1. **Interface file**

interface spi\_slave\_if (clk);

  input  bit clk;

 logic           MOSI, rst\_n, SS\_n, tx\_valid;

 logic      [7:0] tx\_data;

logic [9:0] rx\_data,rx\_data\_ref;

logic       rx\_valid, MISO,MISO\_ref , rx\_valid\_ref;

modport DUT(input clk,MOSI,rst\_n,SS\_n,tx\_data,tx\_valid, output MISO,rx\_data,rx\_valid);

endinterface : spi\_slave\_if

1. **Top file**

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

import spi\_slave\_test\_pkg::\*;

import spi\_slave\_seq\_item\_package::\*;

import shared\_pkg::\*;

module top();

    bit clk;

    initial begin

      forever #2 clk=~clk;

    end

    spi\_slave\_if ss\_if(clk);

    SLAVE DUT(ss\_if);

    SPI\_Slave dut(ss\_if.MOSI,ss\_if.SS\_n,ss\_if.clk,ss\_if.rst\_n,ss\_if.rx\_data\_ref,ss\_if.tx\_valid,ss\_if.tx\_data,ss\_if.MISO\_ref,ss\_if.rx\_valid\_ref);

    initial begin

      uvm\_config\_db #(virtual spi\_slave\_if) :: set(null,"uvm\_test\_top","SPI\_IF",ss\_if);

    run\_test ("spi\_slave\_test");

    end

endmodule

1. **Sequence item file**

package spi\_slave\_seq\_item\_package;

import uvm\_pkg::\*;

 `include "uvm\_macros.svh"

import shared\_pkg::\*;

class spi\_slave\_seq\_item extends uvm\_sequence\_item;

  `uvm\_object\_utils(spi\_slave\_seq\_item);

 rand logic [7:0] tx\_data;

 logic [9:0] rx\_data ,rx\_data\_ref;

 logic rx\_valid, MISO,MISO\_ref ,rx\_valid\_ref;

  logic MOSI;

 rand logic SS\_n;

 rand logic rst\_n;

 rand logic tx\_valid;

 rand bit[0:10] array\_rand;

  function new(string name= "spi\_slave\_seq\_item");

     super.new(name);

 endfunction

   //reset

    constraint reset

    {

      rst\_n  dist { 1:/98 , 0:/2};

    }

    //SS\_n for all cases

    constraint serial\_comm\_all\_cases

    {

      if(  array\_rand[0:2] inside {3'b000, 3'b001, 3'b110} && counter\_allcases%14 !=0) {

         SS\_n==0;

      }

       else

        {

          SS\_n==1;

        }

    }

    //SS\_n for read data

    constraint serial\_comm\_read\_data{

     if(  array\_rand[0:2] inside {3'b111} && counter\_read%24 !=0) {

         SS\_n==0;

      }

       else

        {

          SS\_n==1;

        }

    }

    //tx\_valid for read data

    constraint trans\_ram

    {

            if (array\_rand[0:2] == 3'b111  && counter\_read==23)

              {

                tx\_valid==1;

              }

              if(array\_rand[0:2] != 3'b111)

                {

                  tx\_valid==0;

                }

    }

    // array\_rand for all cases

    constraint mosi\_in

    {

      if (SS\_n\_prev && !SS\_n)

      array\_rand[0:2] inside {3'b000, 3'b001, 3'b110 , 3'b111};

    }

    function void post\_randomize;

      SS\_n\_prev=SS\_n;

       if(counter\_allcases<11 )

        MOSI=array\_rand[counter\_allcases];

    endfunction

    function update\_counter\_allcases;

     counter\_allcases++;

    if (counter\_allcases==14 )

    counter\_allcases=0;

    endfunction

    function update\_counter\_read;

     counter\_read++;

    if (counter\_read==24 )

    counter\_read=0;

    endfunction

 function string convert2string();

 return $sformatf("%s reset = 0b%0b , mosi=%b , miso=%b , ss\_n = %b ",super.convert2string(),rst\_n,MOSI,MISO,SS\_n);

 endfunction

 function string convert2string\_stimulus();

 return $sformatf("reset = 0b%0b , mosi=%b , ss\_n = %b ",rst\_n,MOSI,SS\_n);

 endfunction

endclass

endpackage

1. **Main sequence file**

package spi\_slave\_main\_sequence\_package;

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

import spi\_slave\_seq\_item\_package::\*;

import shared\_pkg::\*;

class spi\_slave\_main\_sequence extends  uvm\_sequence #(spi\_slave\_seq\_item);

 `uvm\_object\_utils(spi\_slave\_main\_sequence);

 spi\_slave\_seq\_item seq\_item\_main;

 function new(string name= "spi\_slave\_main\_sequence");

     super.new(name);

 endfunction

 task body ;

 seq\_item\_main= spi\_slave\_seq\_item :: type\_id :: create("seq\_item\_main");

 repeat(1000) begin

 start\_item(seq\_item\_main);

 seq\_item\_main.reset.constraint\_mode(1);

 seq\_item\_main.serial\_comm\_all\_cases.constraint\_mode(1);

 seq\_item\_main.mosi\_in.constraint\_mode(1);

 seq\_item\_main.serial\_comm\_read\_data.constraint\_mode(0);

 seq\_item\_main.trans\_ram.constraint\_mode(0);

 if(counter\_allcases==0) begin

  seq\_item\_main.array\_rand.rand\_mode(1);

    $display("array\_rand=%b , time =%t ,conter=%d",seq\_item\_main.array\_rand,$time,counter\_allcases);

 end

 else begin

  seq\_item\_main.array\_rand.rand\_mode(0);

 end

 assert (seq\_item\_main.randomize() with {seq\_item\_main.array\_rand[0:2] inside {3'b000,3'b001,3'b110};} );

 seq\_item\_main.update\_counter\_allcases;

 finish\_item(seq\_item\_main);

 end

 repeat(1000) begin

 start\_item(seq\_item\_main);

 seq\_item\_main.reset.constraint\_mode(1);

 seq\_item\_main.serial\_comm\_all\_cases.constraint\_mode(0);

 seq\_item\_main.mosi\_in.constraint\_mode(0);

 seq\_item\_main.serial\_comm\_read\_data.constraint\_mode(1);

 seq\_item\_main.trans\_ram.constraint\_mode(1);

 if(counter\_read==0) begin

  seq\_item\_main.array\_rand.rand\_mode(1);

    $display("array\_rand=%b , time =%t ,conter=%d",seq\_item\_main.array\_rand,$time,counter\_read);

 end

 else begin

  seq\_item\_main.array\_rand.rand\_mode(0);

 end

 assert (seq\_item\_main.randomize() with {seq\_item\_main.array\_rand[0:2] inside {3'b111};} );

 seq\_item\_main.update\_counter\_read;

 finish\_item(seq\_item\_main);

 end

 endtask

 endclass

endpackage

1. **Reset sequence file**

package spi\_slave\_reset\_sequence\_package;

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

import spi\_slave\_seq\_item\_package::\*;

class spi\_slave\_reset\_sequence extends  uvm\_sequence #(spi\_slave\_seq\_item);

 `uvm\_object\_utils(spi\_slave\_reset\_sequence);

 spi\_slave\_seq\_item seq\_item;

 function new(string name= "spi\_slave\_reset\_sequence");

     super.new(name);

 endfunction

 task body ;

 seq\_item= spi\_slave\_seq\_item :: type\_id :: create("seq\_item");

 start\_item(seq\_item);

        seq\_item.rst\_n=1;

        seq\_item.rx\_valid=0;

        seq\_item.tx\_valid=0;

        seq\_item.SS\_n=1;

 finish\_item(seq\_item);

 endtask

endclass

endpackage

1. **Sequencer file**

package spi\_slave\_sequencer\_package ;

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

import spi\_slave\_seq\_item\_package::\*;

class spi\_slave\_sequencer extends uvm\_sequencer #(spi\_slave\_seq\_item);

  `uvm\_component\_utils(spi\_slave\_sequencer)

  function new(string name= "spi\_slave\_sequencer",uvm\_component parent=null);

    super.new(name,parent);

  endfunction

endclass

endpackage

1. **Configuration object file**

package spi\_slave\_config\_pkg;

    import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

class spi\_slave\_config extends uvm\_object;

    `uvm\_object\_utils(spi\_slave\_config) // factory store name of class

   virtual spi\_slave\_if ss\_vif; //  create interface inside configration class

 function new(string name= "spi\_slave\_config");

     super.new(name);

 endfunction

endclass

endpackage

1. **Shared pkg**

package shared\_pkg;

int counter\_allcases =0  ;

int counter\_read=0;

 logic SS\_n\_prev=1;

endpackage

1. **Driver**

 package spi\_slave\_driver\_pkg;

 import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    import spi\_slave\_seq\_item\_package::\*;

    import shared\_pkg ::\*;

class spi\_slave\_driver extends uvm\_driver #(spi\_slave\_seq\_item);

 `uvm\_component\_utils(spi\_slave\_driver);

 virtual spi\_slave\_if ss\_vif;

 spi\_slave\_seq\_item seq\_item;

 function new(string name ="spi\_slave\_driver",uvm\_component parent =null);

    super.new(name,parent);

    endfunction

    task run\_phase(uvm\_phase phase);

    super.run\_phase(phase);

    forever begin

      seq\_item =  spi\_slave\_seq\_item :: type\_id :: create ( "seq\_item");

      seq\_item\_port.get\_next\_item(seq\_item);

      ss\_vif.rst\_n=seq\_item.rst\_n;

      ss\_vif.SS\_n=seq\_item.SS\_n;

      ss\_vif.MOSI=seq\_item.MOSI;

      ss\_vif.tx\_valid=seq\_item.tx\_valid;

      ss\_vif.tx\_data=seq\_item.tx\_data;

      @(negedge ss\_vif.clk);

      seq\_item\_port.item\_done();

      `uvm\_info ("run\_phase",seq\_item.convert2string\_stimulus(),UVM\_HIGH)

    end

    endtask

endclass

 endpackage

1. **Monitor**

package spi\_slave\_monitor\_pkg;

 import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    import  spi\_slave\_seq\_item\_package::\*;

    import shared\_pkg ::\*;

class spi\_slave\_monitor extends uvm\_monitor;

 `uvm\_component\_utils(spi\_slave\_monitor);

 virtual spi\_slave\_if ss\_vif;

spi\_slave\_seq\_item seq\_item;

 uvm\_analysis\_port #(spi\_slave\_seq\_item) mon\_ap;

  function new(string name ="spi\_slave\_monitor",uvm\_component parent =null);

    super.new(name,parent);

    endfunction

            function void build\_phase( uvm\_phase phase);

            super.build\_phase(phase);

            mon\_ap = new("mon\_ap", this);

        endfunction

    task run\_phase (uvm\_phase phase);

    super.run\_phase(phase);

    forever begin

        seq\_item=spi\_slave\_seq\_item::type\_id::create("seq\_item");

        @(negedge ss\_vif.clk);

        seq\_item.rst\_n=ss\_vif.rst\_n;

        seq\_item.MOSI=ss\_vif.MOSI;

        seq\_item.SS\_n=ss\_vif.SS\_n;

        seq\_item.MISO=ss\_vif.MISO;

        seq\_item.MISO\_ref=ss\_vif.MISO\_ref;

        seq\_item.tx\_valid=ss\_vif.tx\_valid;

        seq\_item.tx\_data=ss\_vif.tx\_data;

        seq\_item.rx\_valid=ss\_vif.rx\_valid;

        seq\_item.rx\_data=ss\_vif.rx\_data;

        seq\_item.rx\_valid\_ref=ss\_vif.rx\_valid\_ref;

         seq\_item.rx\_data\_ref=ss\_vif.rx\_data\_ref;

        mon\_ap.write(seq\_item);

        `uvm\_info ("run\_phase",seq\_item.convert2string\_stimulus(),UVM\_HIGH)

    end

    endtask

endclass

endpackage

1. **Agent**

package spi\_slave\_agent\_pkg;

  import uvm\_pkg::\*;

  `include "uvm\_macros.svh"

  import  spi\_slave\_seq\_item\_package::\*;

  import spi\_slave\_config\_pkg::\*;

  import  spi\_slave\_driver\_pkg::\*;

  import spi\_slave\_monitor\_pkg::\*;

  import spi\_slave\_sequencer\_package::\*;

  class spi\_slave\_agent extends uvm\_agent;

    `uvm\_component\_utils(spi\_slave\_agent)

    spi\_slave\_seq\_item seq\_item;

    spi\_slave\_config           s\_cfg;

    spi\_slave\_driver       drv;

   spi\_slave\_sequencer sqr;

   spi\_slave\_monitor       mon;

    uvm\_analysis\_port #(spi\_slave\_seq\_item) agt\_ap;

    function new(string name = "spi\_slave\_agent", uvm\_component parent = null);

      super.new(name, parent);

    endfunction

    function void build\_phase(uvm\_phase phase);

      super.build\_phase(phase);

      if (!uvm\_config\_db#(spi\_slave\_config)::get(this, "", "CFG", s\_cfg)) begin

        `uvm\_fatal("build\_phase", "unable to get configuration object")

      end

      sqr = spi\_slave\_sequencer::type\_id::create("sqr", this);

      drv = spi\_slave\_driver::type\_id::create("drv", this);

      mon = spi\_slave\_monitor::type\_id::create("mon", this);

      agt\_ap = new("agt\_tb",this);

    endfunction

    function void connect\_phase(uvm\_phase phase);

      super.connect\_phase(phase);

      drv.ss\_vif = s\_cfg.ss\_vif;

      mon.ss\_vif = s\_cfg.ss\_vif;

      drv.seq\_item\_port.connect(sqr.seq\_item\_export);

     mon.mon\_ap.connect(agt\_ap);

    endfunction

  endclass

endpackage

**16. Scoreboard**

package spi\_slave\_scoreboard\_pkg;

 import uvm\_pkg::\*;

    `include "uvm\_macros.svh"

    import  spi\_slave\_seq\_item\_package::\*;

    class spi\_slave\_scoreboard extends uvm\_scoreboard ;

     `uvm\_component\_utils(spi\_slave\_scoreboard);

     uvm\_analysis\_export #(spi\_slave\_seq\_item) sb\_export;

     uvm\_tlm\_analysis\_fifo  #(spi\_slave\_seq\_item) sb\_fifo;

    spi\_slave\_seq\_item sq\_item\_sb;

     int err\_count=0;

     int correct\_count=0;

     function new(string name ="spi\_slave\_scoreboard",uvm\_component parent =null);

    super.new(name,parent);

    endfunction

    function void build\_phase (uvm\_phase phase);

    super.build\_phase(phase);

    sb\_export = new("sb\_export",this);

    sb\_fifo = new("sb\_fifo" , this);

    endfunction

    function void connect\_phase (uvm\_phase phase);

    super.connect\_phase(phase);

    sb\_export.connect(sb\_fifo.analysis\_export);

    endfunction

    task run\_phase (uvm\_phase phase);

  super.run\_phase(phase);

  forever begin

  sb\_fifo.get(sq\_item\_sb);

  if (sq\_item\_sb.MISO != sq\_item\_sb.MISO\_ref || sq\_item\_sb.rx\_valid !=sq\_item\_sb.rx\_valid\_ref || sq\_item\_sb.rx\_data!=sq\_item\_sb.rx\_data\_ref) begin

    `uvm\_error("run\_phase",$sformatf("Comparison failed, Transaction received by the DUT: %s While the reference out:0b%0b",

sq\_item\_sb.convert2string(),sq\_item\_sb.MISO\_ref))

    err\_count++;

  end

  else begin

    `uvm\_info("run\_phase",$sformatf("Correct ALU out: %s", sq\_item\_sb.convert2string()), UVM\_HIGH)

    correct\_count++;

  end

  end

endtask

function void report\_phase(uvm\_phase phase) ;

super. report\_phase (phase) ;

 `uvm\_info("report\_phase", $sformatf("Total successful transactions: %0d", correct\_count), UVM\_MEDIUM);

`uvm\_info("report\_phase", $sformatf("Total failed transactions: %0d", err\_count), UVM\_MEDIUM);

endfunction

    endclass

 endpackage

**17. collector**

package  spi\_slave\_collector\_package ;

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

 import  spi\_slave\_seq\_item\_package::\*;

 import shared\_pkg::\*;

class spi\_slave\_coverage extends uvm\_component;

  `uvm\_component\_utils(spi\_slave\_coverage)

  uvm\_analysis\_export #(spi\_slave\_seq\_item) cov\_export;

  uvm\_tlm\_analysis\_fifo #(spi\_slave\_seq\_item) cov\_fifo;

  spi\_slave\_seq\_item seq\_item\_cov;

  covergroup covcode ;

  reciver\_data: coverpoint seq\_item\_cov.rx\_data[9:8];

  ss\_n\_allcases : coverpoint seq\_item\_cov.SS\_n

  {

    bins trans\_all\_cases= (1 => 0[\*13] =>1) ;

  }

  ss\_n\_read\_data : coverpoint seq\_item\_cov.SS\_n

  {

     bins trans\_read= (1 => 0[\*23] =>1) ;

  }

  ss\_n : coverpoint seq\_item\_cov.SS\_n

  {

    bins ss\_n\_val={0};

  }

  mosi: coverpoint seq\_item\_cov.MOSI

  {

    bins write\_add = (0=>0=>0);

    bins write\_data = (0=>0=>1);

    bins read\_add = (1=>1=>0);

    bins read\_data =(1=>1=>1);

    bins mosi\_val\_low={0};

     bins mosi\_val\_high={1};

  }

  cross\_mosi\_ss\_n : cross mosi,ss\_n

  {

     option.cross\_auto\_bin\_max=0;

     bins ss\_n\_low\_mosi\_low = binsof(ss\_n.ss\_n\_val) && binsof(mosi.mosi\_val\_low);

     bins ss\_n\_low\_mosi\_high = binsof(ss\_n.ss\_n\_val)  && binsof(mosi.mosi\_val\_high);

  }

  endgroup

  function new(string name = "spi\_slave\_coverage", uvm\_component parent = null);

    super.new(name, parent);

    covcode=new();

  endfunction

  function void build\_phase(uvm\_phase phase);

    super.build\_phase(phase);

    cov\_export = new("cov\_export", this);

    cov\_fifo   = new("cov\_fifo", this);

  endfunction

  function void connect\_phase(uvm\_phase phase) ;

    super.connect\_phase(phase);

    cov\_export.connect(cov\_fifo.analysis\_export);

  endfunction

  task run\_phase(uvm\_phase phase);

  super.run\_phase(phase);

  forever begin

    cov\_fifo.get(seq\_item\_cov);

    covcode.sample();

  end

  endtask

endclass

endpackage

**18. Environment**

package spi\_slave\_env\_pkg;

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

import spi\_slave\_agent\_pkg::\*;

import spi\_slave\_scoreboard\_pkg::\*;

import spi\_slave\_collector\_package::\*;

class spi\_slave\_env extends uvm\_env;

      `uvm\_component\_utils(spi\_slave\_env);

      spi\_slave\_scoreboard score;

       spi\_slave\_coverage cover\_grp;

     spi\_slave\_agent agt;

      function new(string name ="spi\_slave\_env",uvm\_component parent =null);

    super.new(name,parent);

    endfunction

   function void build\_phase (uvm\_phase phase);

   super.build\_phase(phase);

   agt = spi\_slave\_agent :: type\_id ::create("agt",this);

   score =   spi\_slave\_scoreboard :: type\_id :: create("score",this);

   cover\_grp = spi\_slave\_coverage  ::type\_id ::create("cover\_grp",this);

   endfunction

   function void connect\_phase (uvm\_phase phase);

   agt.agt\_ap.connect(score.sb\_export);

   agt.agt\_ap.connect(cover\_grp.cov\_export);

   endfunction

endclass

endpackage

**19. Test**

package spi\_slave\_test\_pkg;

import uvm\_pkg::\*;

`include "uvm\_macros.svh"

import spi\_slave\_env\_pkg::\*;

 import spi\_slave\_reset\_sequence\_package::\*;

 import spi\_slave\_main\_sequence\_package::\*;

import spi\_slave\_config\_pkg ::\*;

class spi\_slave\_test extends uvm\_test;

    `uvm\_component\_utils(spi\_slave\_test);

   spi\_slave\_env  env;

   spi\_slave\_config   s\_cfg;

    spi\_slave\_reset\_sequence rst\_seq;

   spi\_slave\_main\_sequence main\_seq;

    function new(string name ="spi\_slave\_test",uvm\_component parent =null);

    super.new(name,parent);

    endfunction

    function void build\_phase (uvm\_phase phase);

    super.build\_phase(phase);

    env =spi\_slave\_env :: type\_id :: create("env",this);

    s\_cfg = spi\_slave\_config   :: type\_id :: create ("s\_cfg");

    rst\_seq= spi\_slave\_reset\_sequence :: type\_id :: create ("rst\_seq");

    main\_seq=spi\_slave\_main\_sequence :: type\_id :: create("main\_seq");

    if(!uvm\_config\_db #(virtual spi\_slave\_if) :: get(this,"","SPI\_IF",s\_cfg.ss\_vif))

    `uvm\_fatal("build\_phase","Test - unable to get the virtual interface");

    uvm\_config\_db #(spi\_slave\_config) :: set(this,"\*","CFG",s\_cfg);

    endfunction

    task run\_phase(uvm\_phase phase);

    super.run\_phase(phase);

    phase.raise\_objection(this);

     `uvm\_info("run\_phase","reset\_asserted",UVM\_LOW)

     rst\_seq.start(env.agt.sqr);

     `uvm\_info("run\_phase","reset\_deasserted",UVM\_LOW)

     `uvm\_info ("run\_phase","stimulus generation begin",UVM\_LOW)

      main\_seq.start(env.agt.sqr);

       `uvm\_info ("run\_phase","stimulus generation done",UVM\_LOW)

    phase.drop\_objection(this);

    endtask

endclass: spi\_slave\_test

endpackage

**20. Src\_file**

**A screenshot of a computer

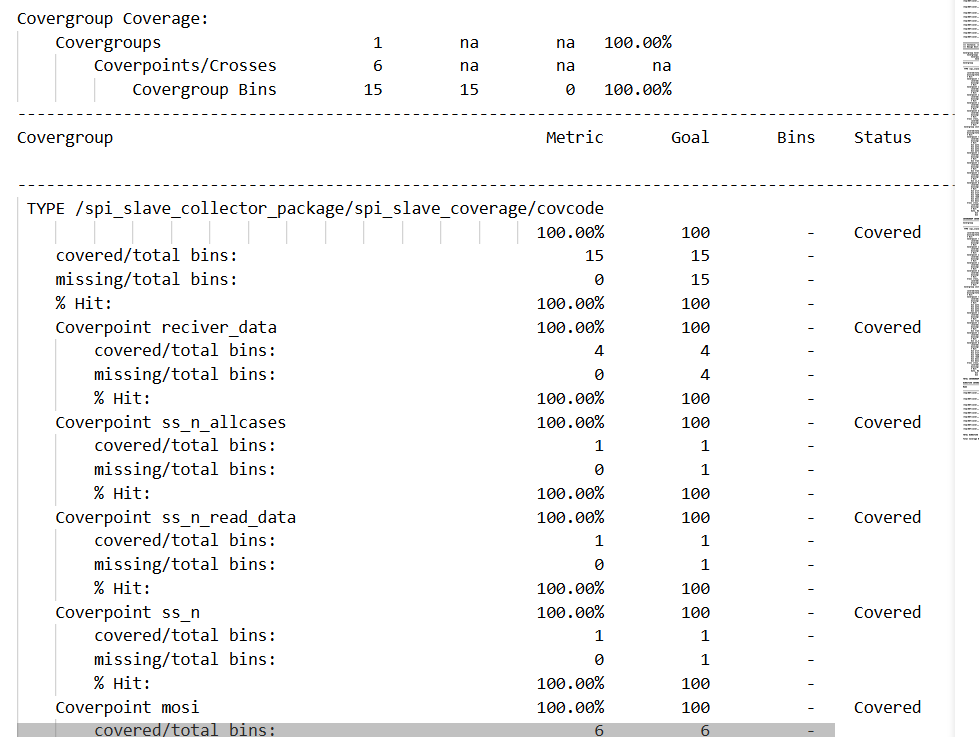
AI-generated content may be incorrect.**

**21. Do file**

**A screen shot of a computer

AI-generated content may be incorrect.**

**22. Functional coverage**

****

**23. Code coverage**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

**24. Assertion coverage**

**A screenshot of a computer

AI-generated content may be incorrect.**

**25. Questasim snippets**

***A screenshot of a computer

AI-generated content may be incorrect.***

Figure when ss\_n is deasserted mosi read\_add sequence (110)

***A computer screen shot of a computer program

AI-generated content may be incorrect.***

Figure when ss\_n is deasserted mosi read\_data sequence (111)

***A computer screen shot of a computer program

AI-generated content may be incorrect.***

Figure when ss\_n is deasserted mosi write\_add sequence (000)

A screenshot of a computer

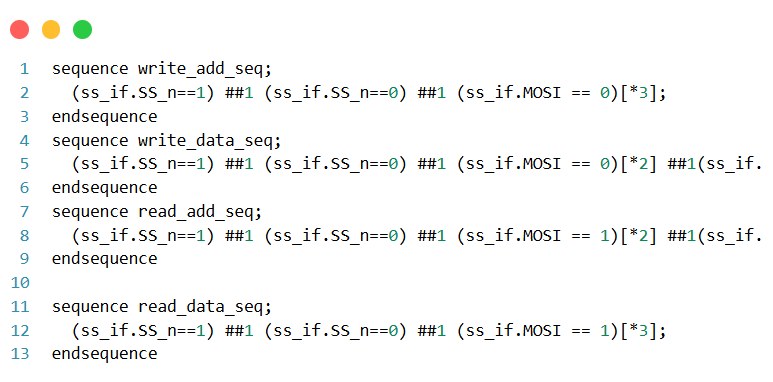
AI-generated content may be incorrect.

Figure when ss\_n is deasserted mosi write\_data sequence (001)

**26. Table of assertions with features**

|  |  |
| --- | --- |
| **Feature** | **Assertion** |
| Whenever the rst is asserted, MISO , rx\_valid , rx\_data is low | @(posedge ss\_if.clk)   (~ss\_if.rst\_n) |=>(ss\_if.MISO ==0 && ss\_if.rx\_valid==0 && ss\_if.rx\_data==0); |
| Whenever the current state is idle and SS\_n is low so current state will be CHCK\_CMD in next cycle and reset is high | @(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n) (cs==IDLE && !ss\_if.SS\_n) |=>(cs==CHK\_CMD); |
| Whenever the current state CHCK\_CMD and SS\_n is low and mosi is low so current state will be WRITE in next cycle and reset is high | @(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n) (cs==CHK\_CMD && !ss\_if.SS\_n && !ss\_if.MOSI) |=>  (cs==WRITE); |
| Whenever the current state CHCK\_CMD and SS\_n is low and mosi is high and received\_add is low so current state will be WRITE in next cycle and reset is high | @(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n)  (cs==CHK\_CMD && !ss\_if.SS\_n && ss\_if.MOSI && !received\_address) |=>(cs==READ\_ADD); |
| Whenever the current state CHCK\_CMD and SS\_n is low and mosi is high and received\_add is high so current state will be WRITE in next cycle and reset is high | @(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n)  (cs==CHK\_CMD && !ss\_if.SS\_n && ss\_if.MOSI && received\_address) |=>(cs==READ\_DATA); |
| Whenever the rst is asserted and current state is WRITE , the current state will be IDEAL in next clock cycle | @(posedge ss\_if.clk)  (cs==WRITE &&(~ss\_if.rst\_n)) |=> (cs==IDLE); |
| Whenever the rst is asserted and current state is READ\_ADD , the current state will be IDEAL in next clock cycle | @(posedge ss\_if.clk)  (cs==READ\_ADD && (~ss\_if.rst\_n)) |=> (cs==IDLE); |
| Whenever the rst is asserted and current state is READ\_DATA , the current state will be IDEAL in next clock cycle | @(posedge ss\_if.clk)  (cs==READ\_DATA && (~ss\_if.rst\_n)) |=> (cs==IDLE); |
| Whenever SS\_n is deasserted and wite\_add or write\_data or read\_add or read\_data sequence done then rx\_valid will assert and SS\_n will assert to end communication | @(posedge ss\_if.clk) disable iff(~ss\_if.rst\_n) (write\_add\_seq or write\_data\_seq or read\_add\_seq or read\_data\_seq) |=> ##9 ($rose(ss\_if.rx\_valid) && $rose (ss\_if.SS\_n)[->1]); |

***Sequence which I mentioned in last row :***

******

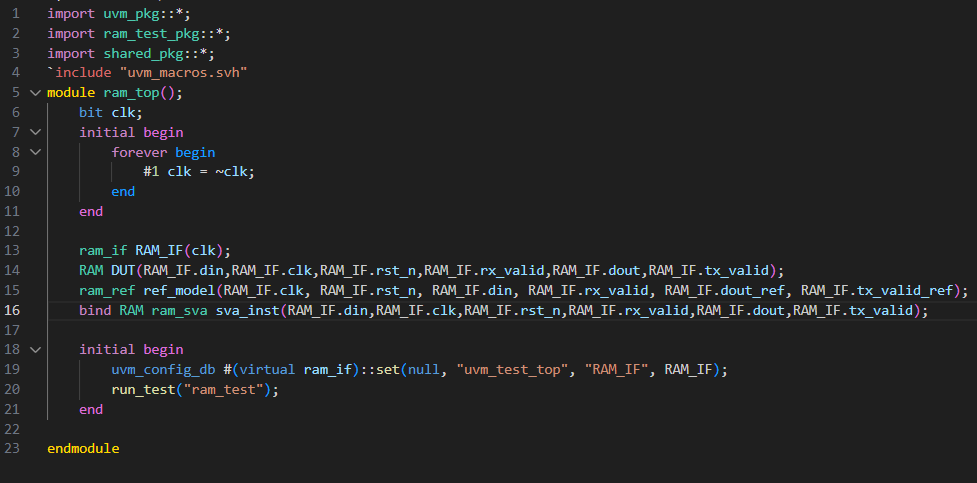
***RAM Part:***

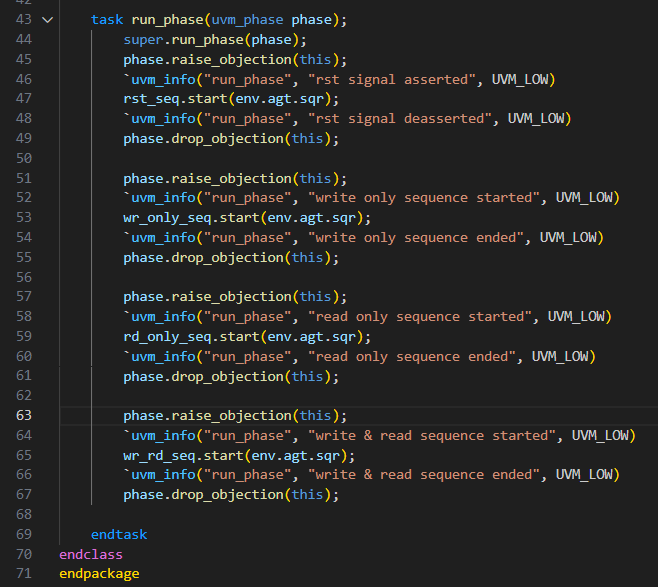
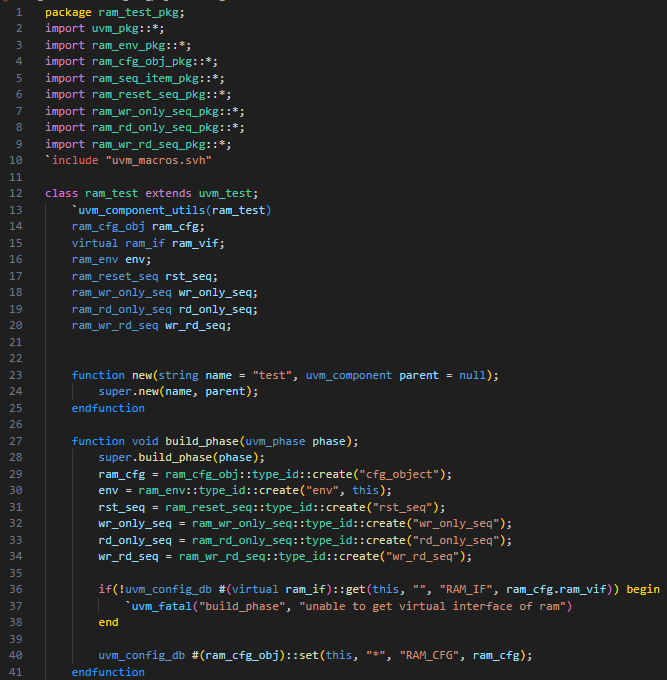
|  |  |
| --- | --- |
| Feature | Assertion |
| When rst\_n is asserted, dout & tx\_valid are low | (@(posedge clk) (!rst\_n) |=> (dout == 0) && (tx\_valid == 0)) |
| When din[9:8] = 11 and rx\_valid is asserted, tx\_valid should be high and should eventually fall. | @(posedge clk) disable iff(!rst\_n)  ((din[8] && din[9]) && rx\_valid) |=> (tx\_valid == 1) |=> $fell(tx\_valid)[->1]) |
| When din[9:8] = 10, tx\_valid should be low. | (@(posedge clk) disable iff(!rst\_n) (!(din[8] && din[9])) |=> (tx\_valid == 0)) |
| When din[9:8] = 00 which is a write address instruction, there should eventually be a write data instruction din[9:8] = 01 | (@(posedge clk) disable iff(!rst\_n) ((!din[8] && !din[9])) |=> (din[8] && !din[9])[->1]); |
| When din[9:8] = 10 which is a read address instruction, there should eventually be a read data instruction din[9:8] = 11 | (@(posedge clk) disable iff(!rst\_n) ((!din[8] && din[9])) |=> (din[8] && din[9])[->1]); |

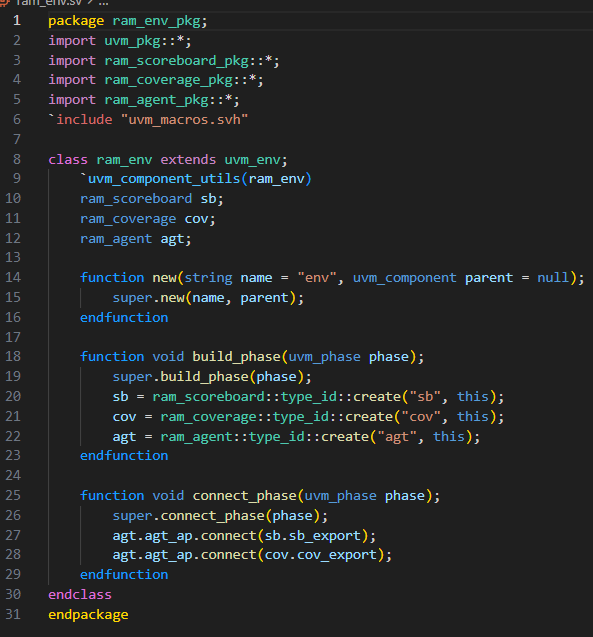
Ram Uvm architecture :

A screenshot of a computer

AI-generated content may be incorrect.

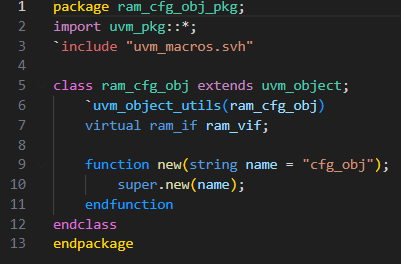
Top:

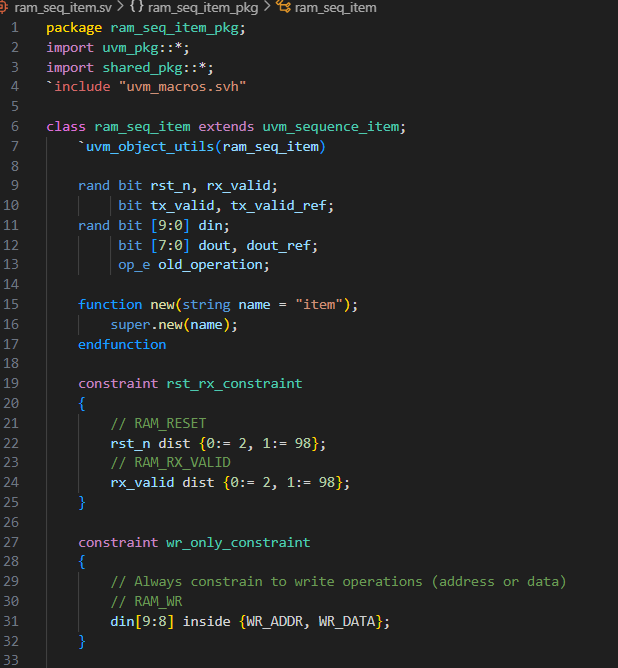
Test:

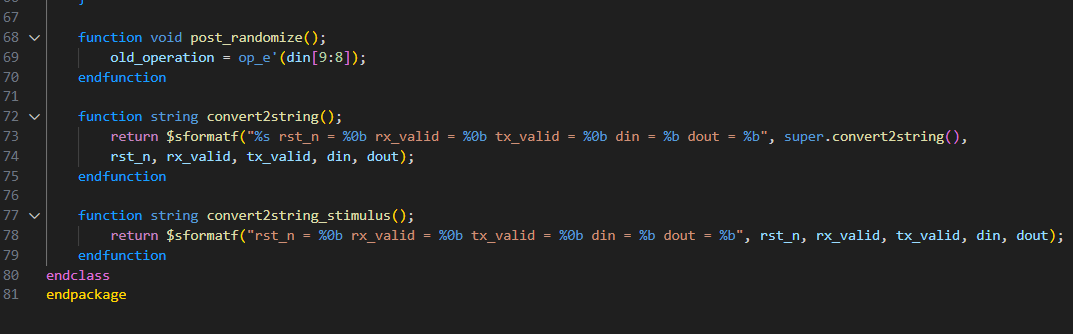
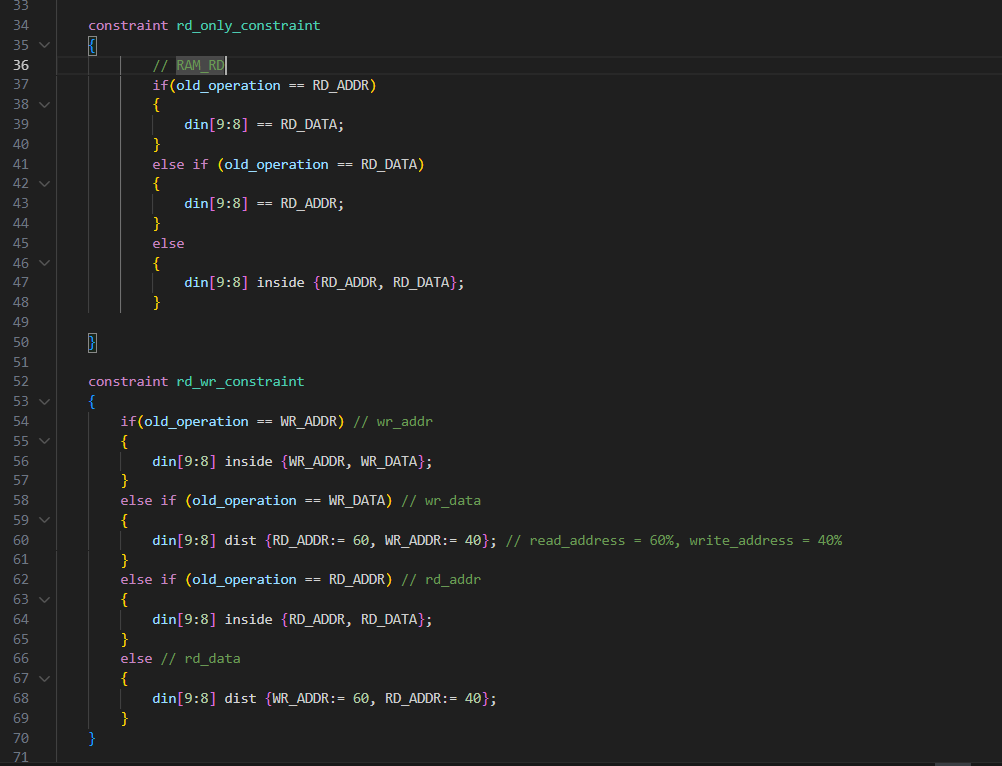
Env:

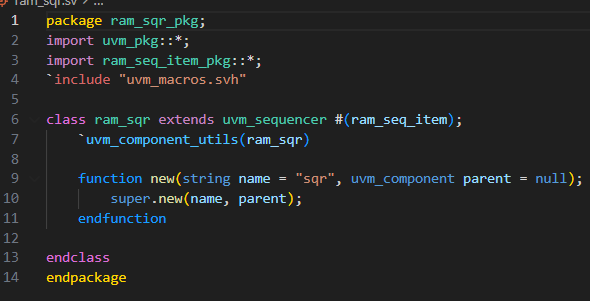
Agent:

Config Object, Sequence Item and Sequencer:

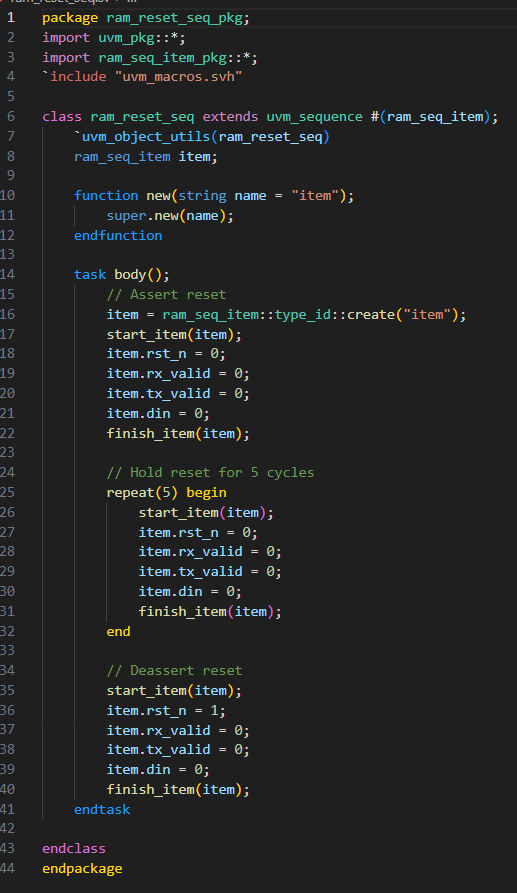


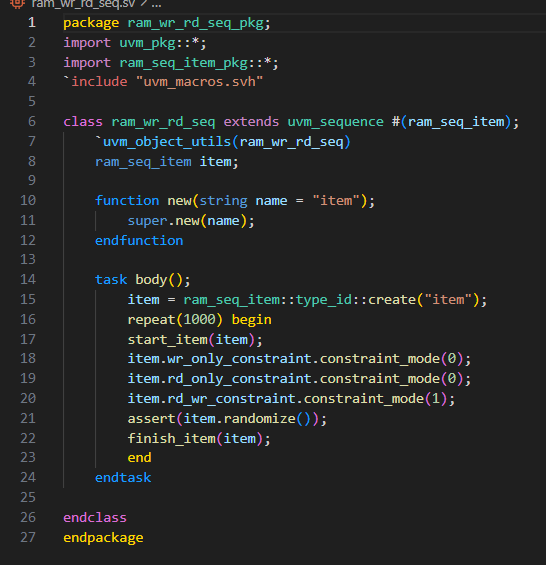


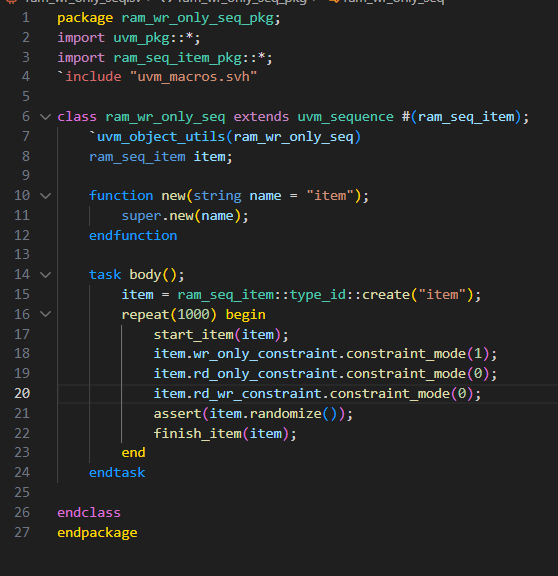




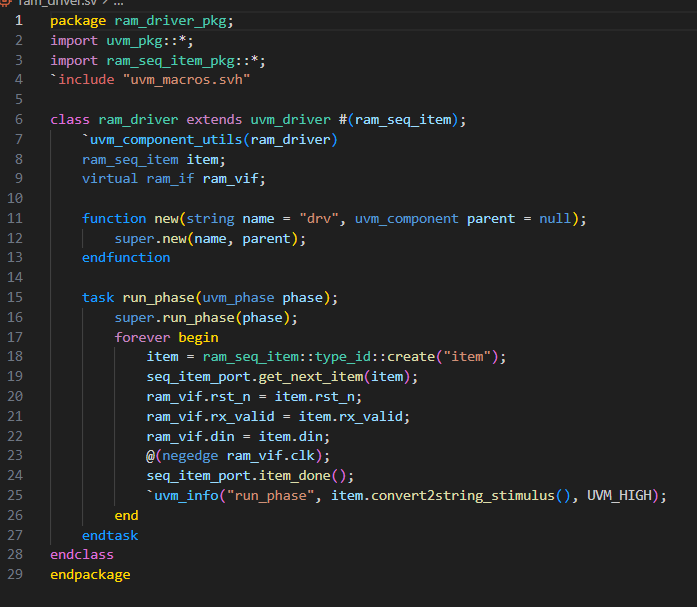
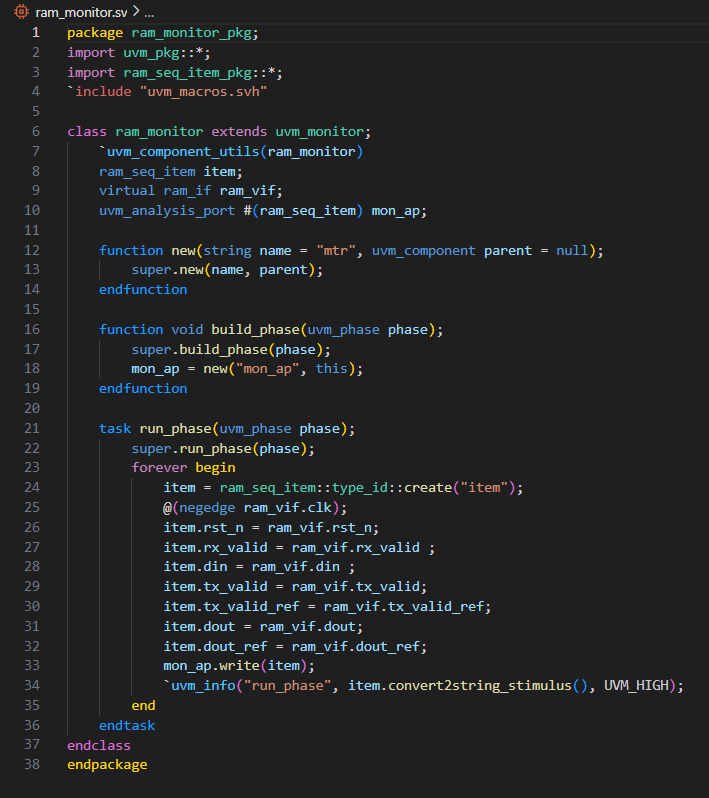
Reset, Write only sequence, Read only sequence and W&R sequence:

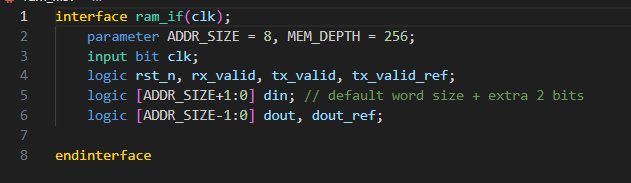


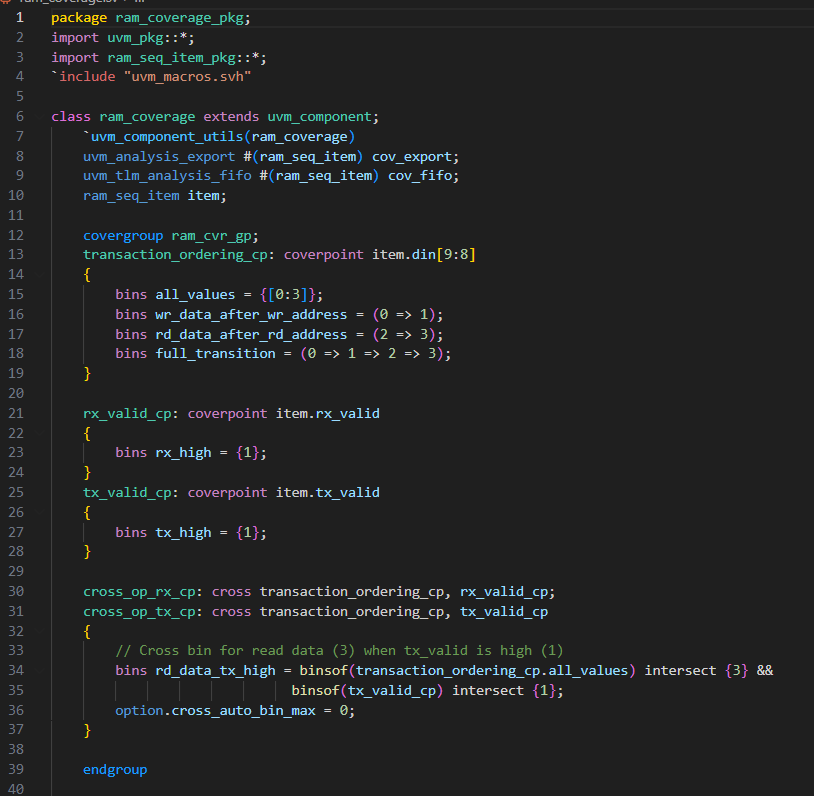


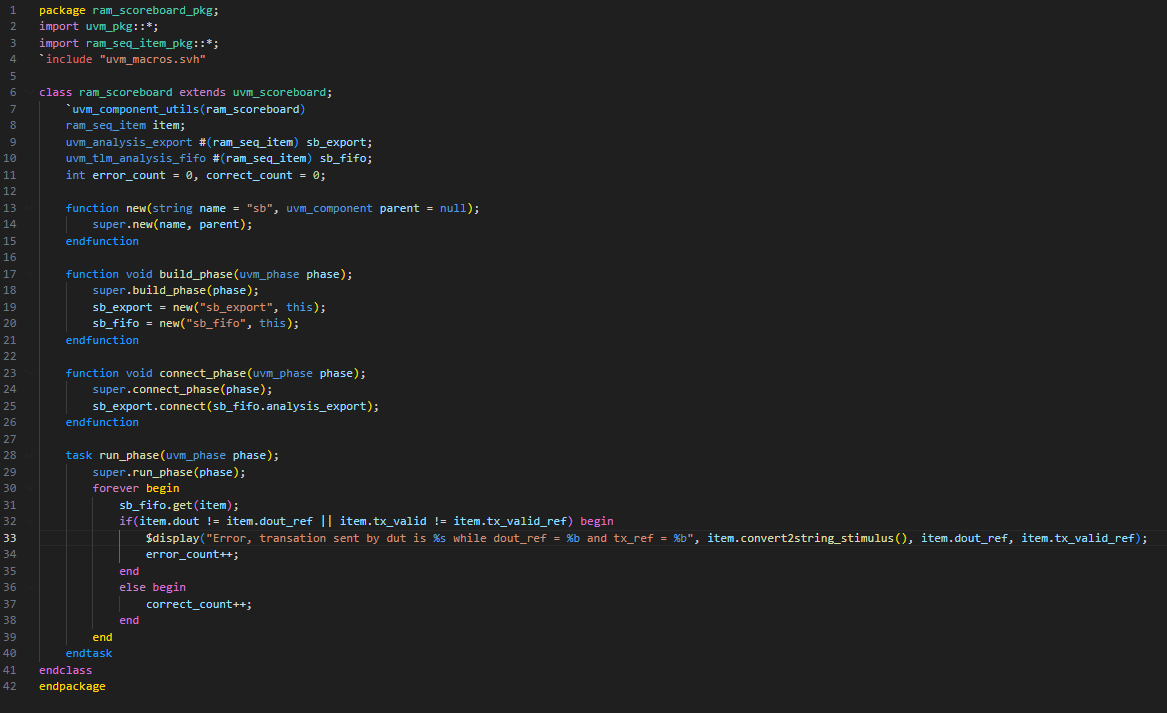


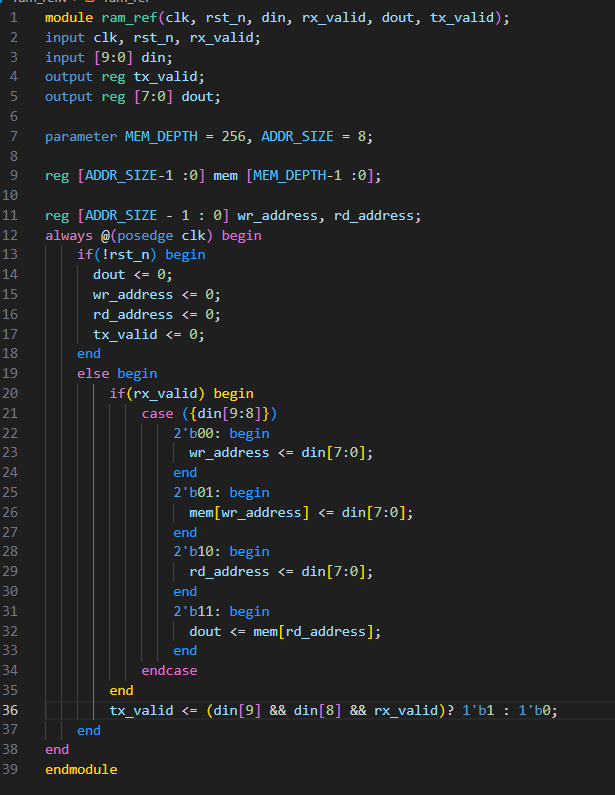


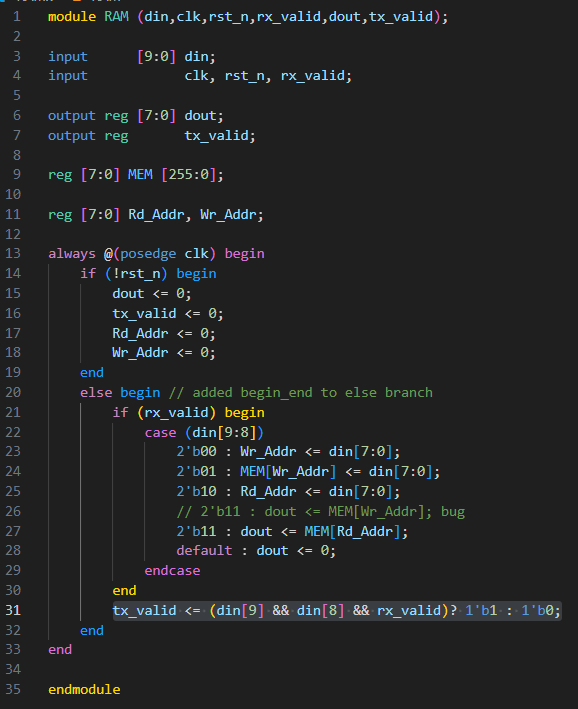
Monitor and Driver:

Interface:

Coverage:

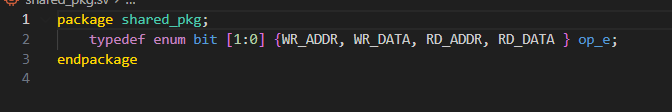
Scoreboard:

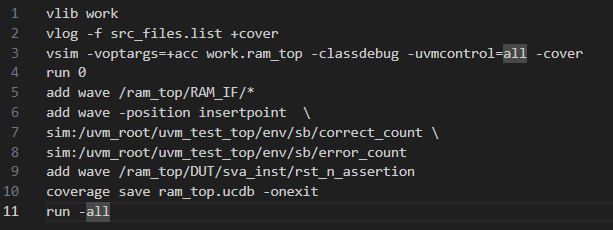
Reference Model:

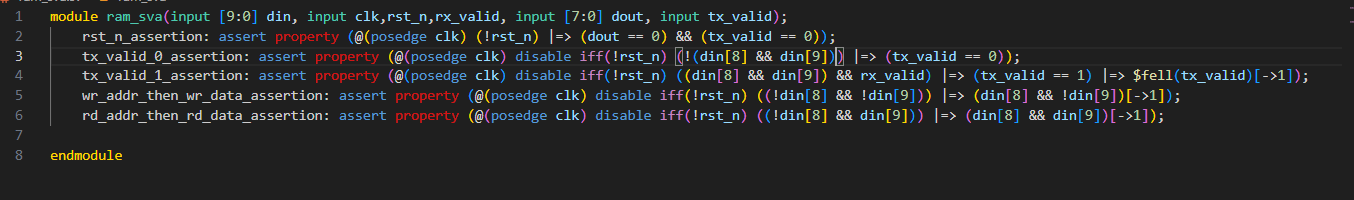
Design:

Bugs Found:

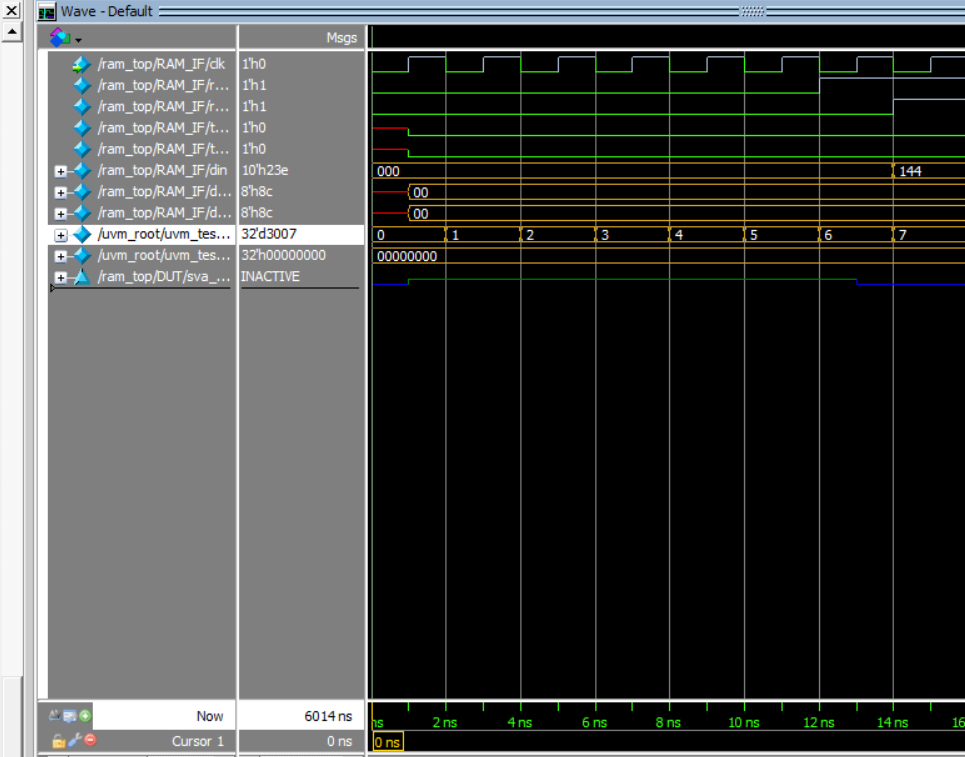
1. In READ\_DATA case, output would read from Wr\_addr.
2. Missing begin end in else branch

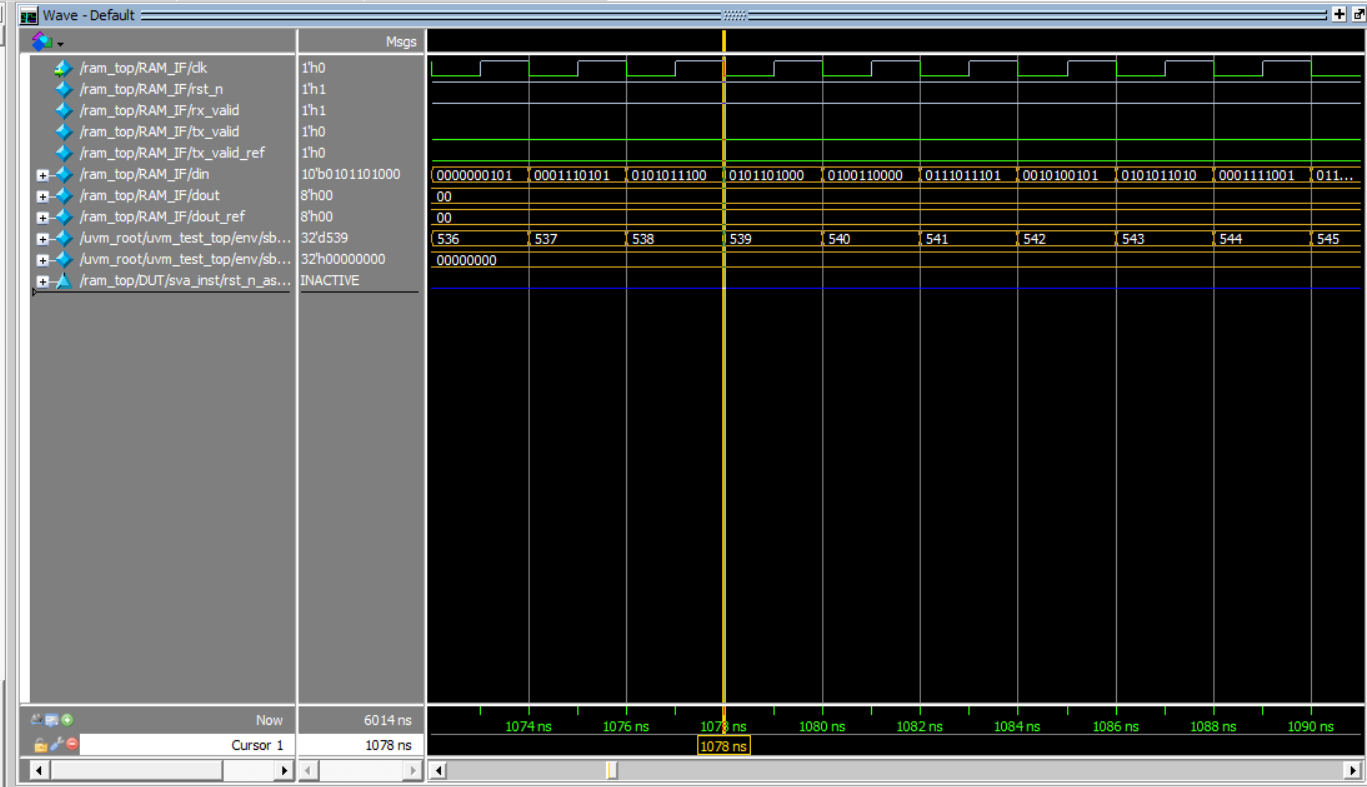
Shared\_Pkg:

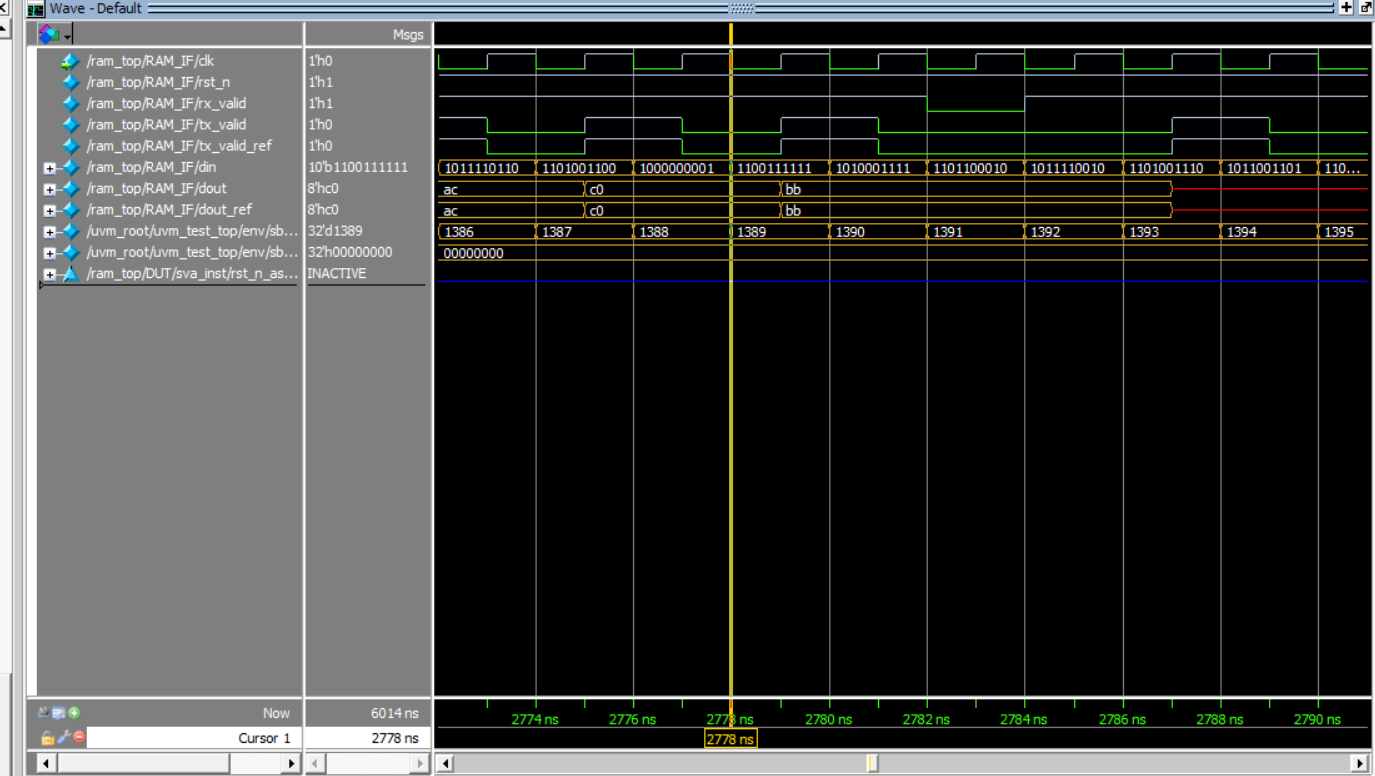
Do File:

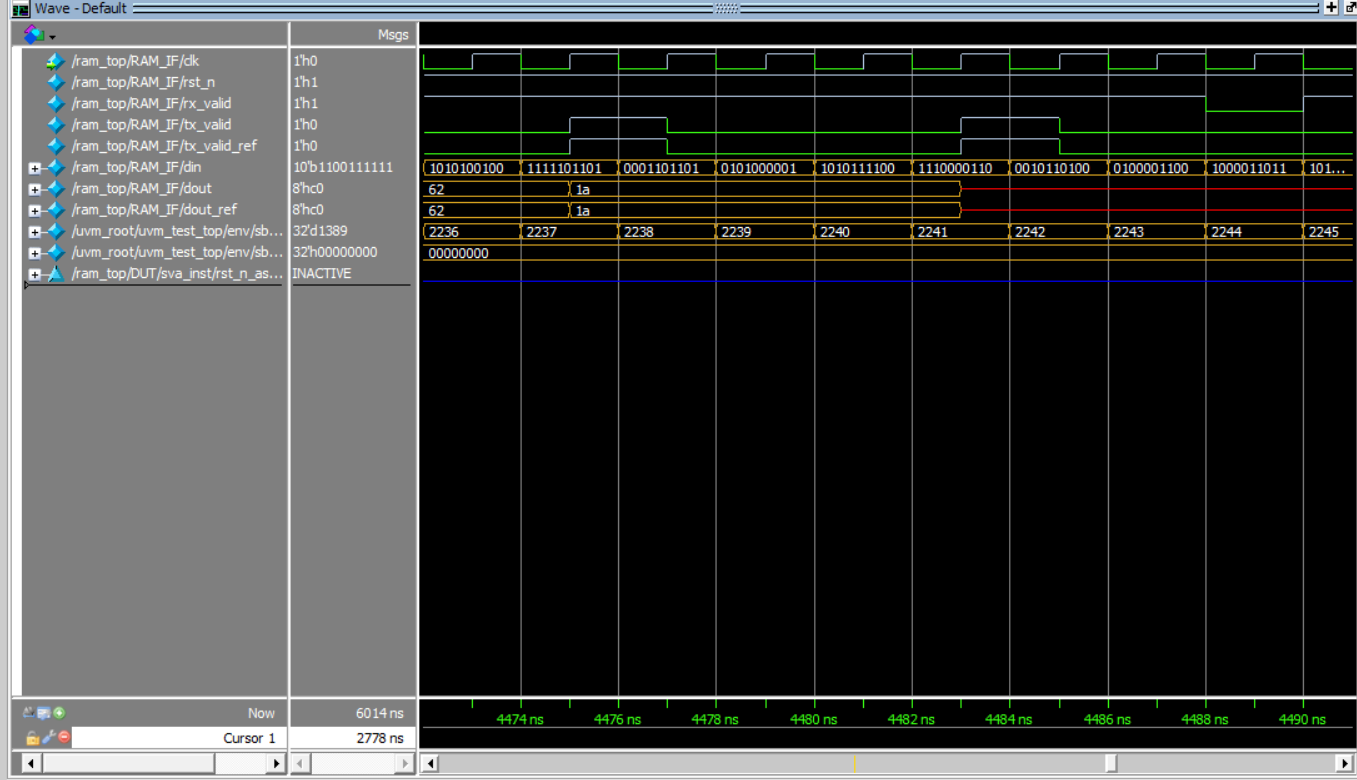
Assertions:

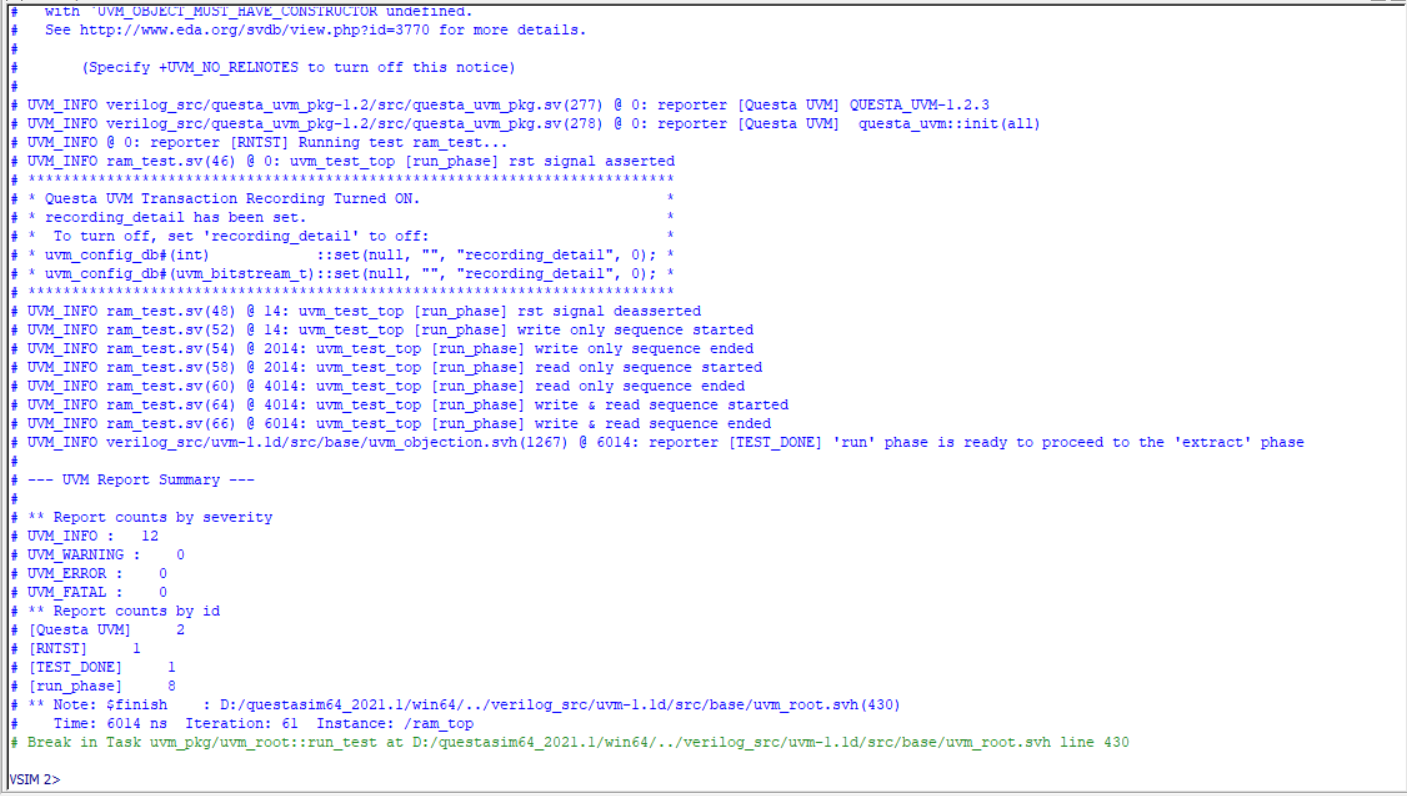
Waveforms:

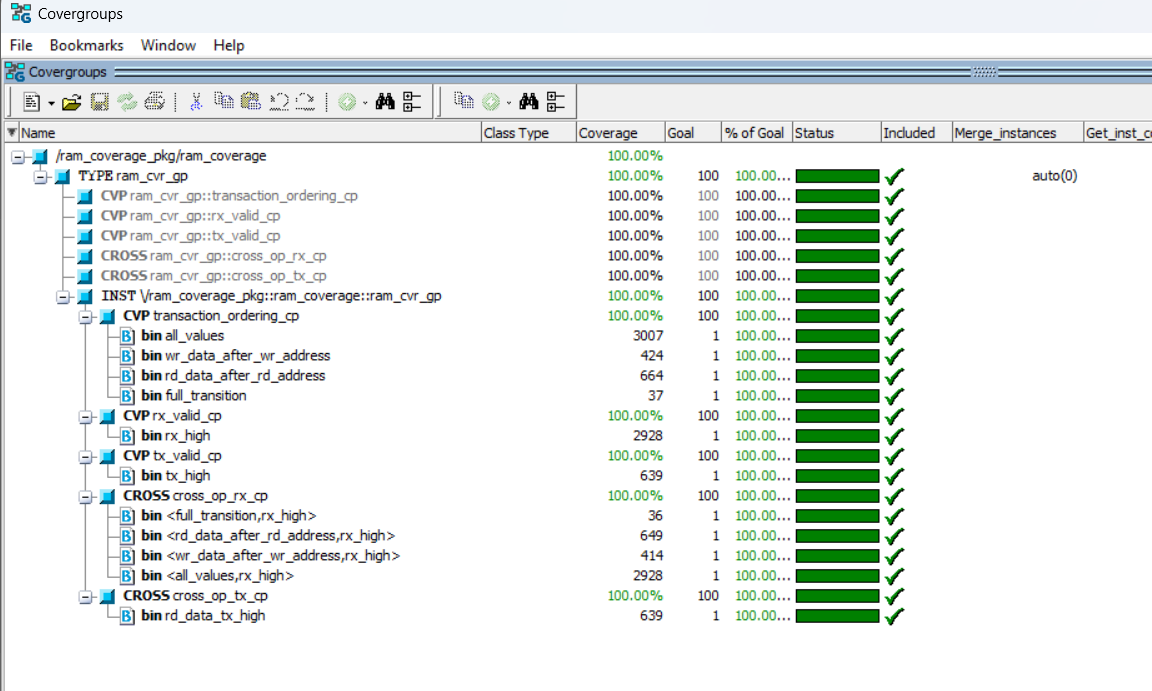
Reset Sequence:

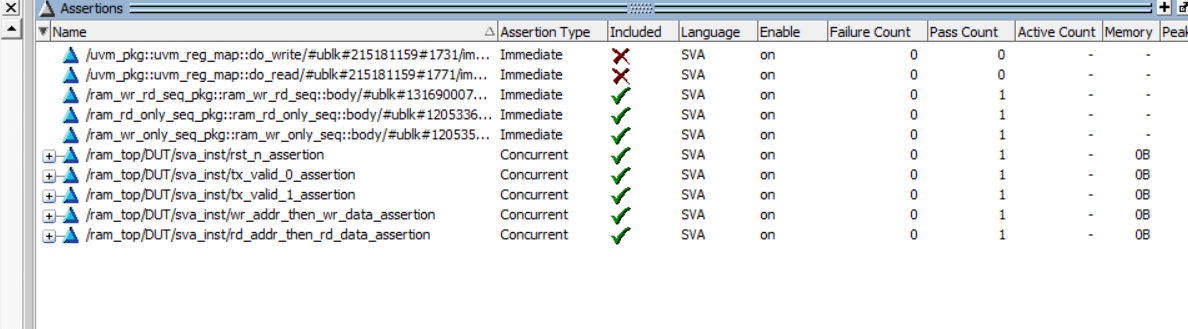
Write-Only Sequence:

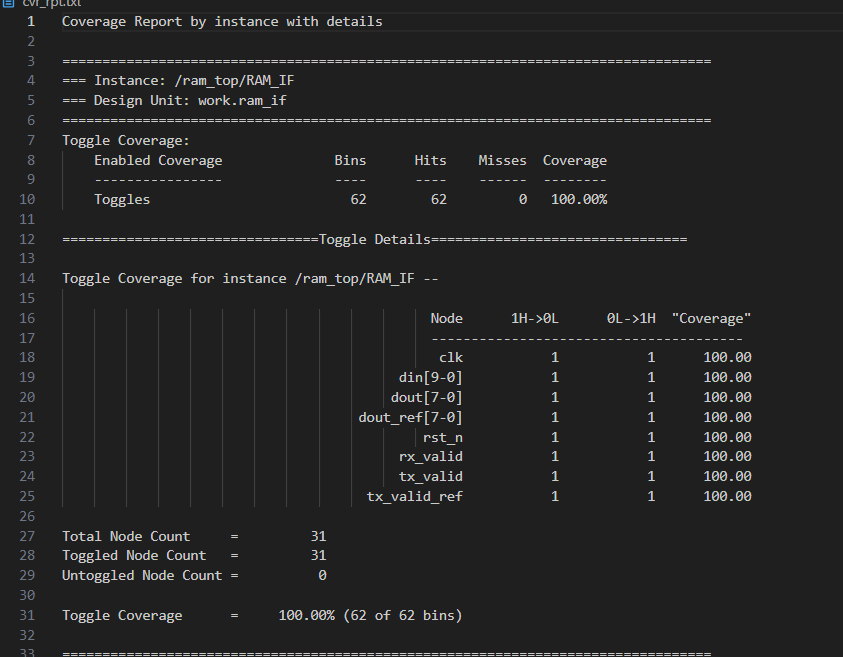
Read-Only Sequence:

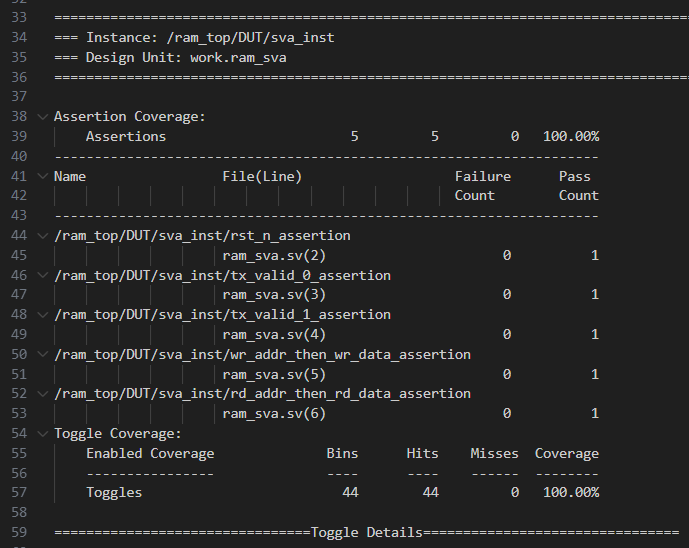
Write & Read Sequence:

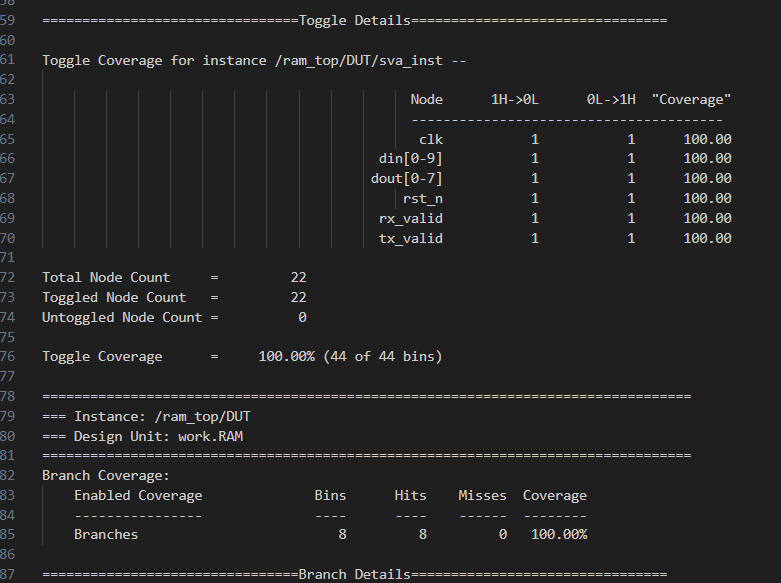
Transcript:

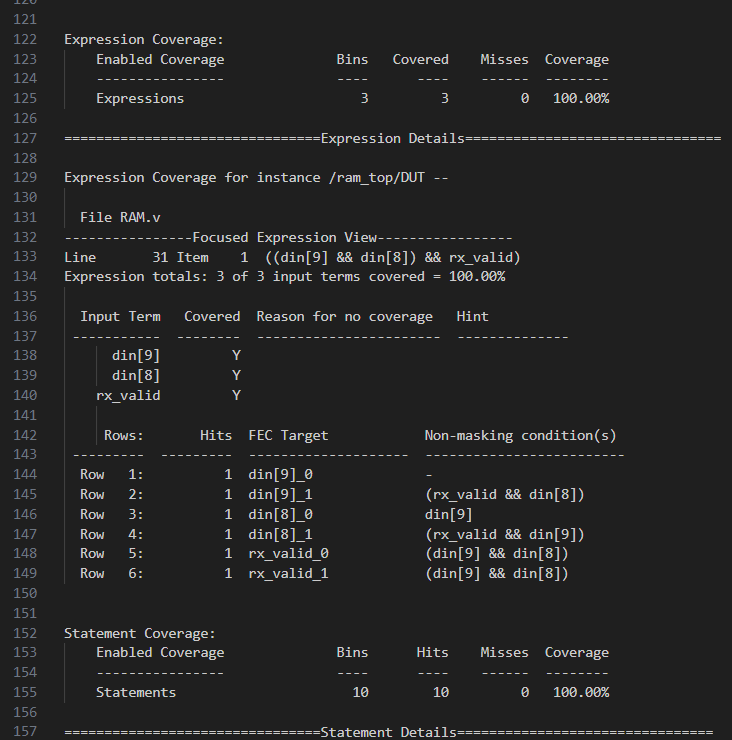
Functional Coverage:

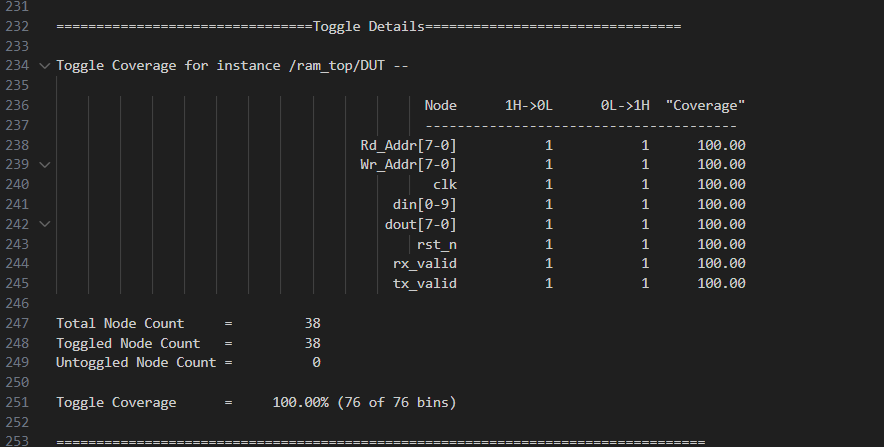
Assertions Coverage:

Code Coverage:









Default branch excluded as all cases are covered.