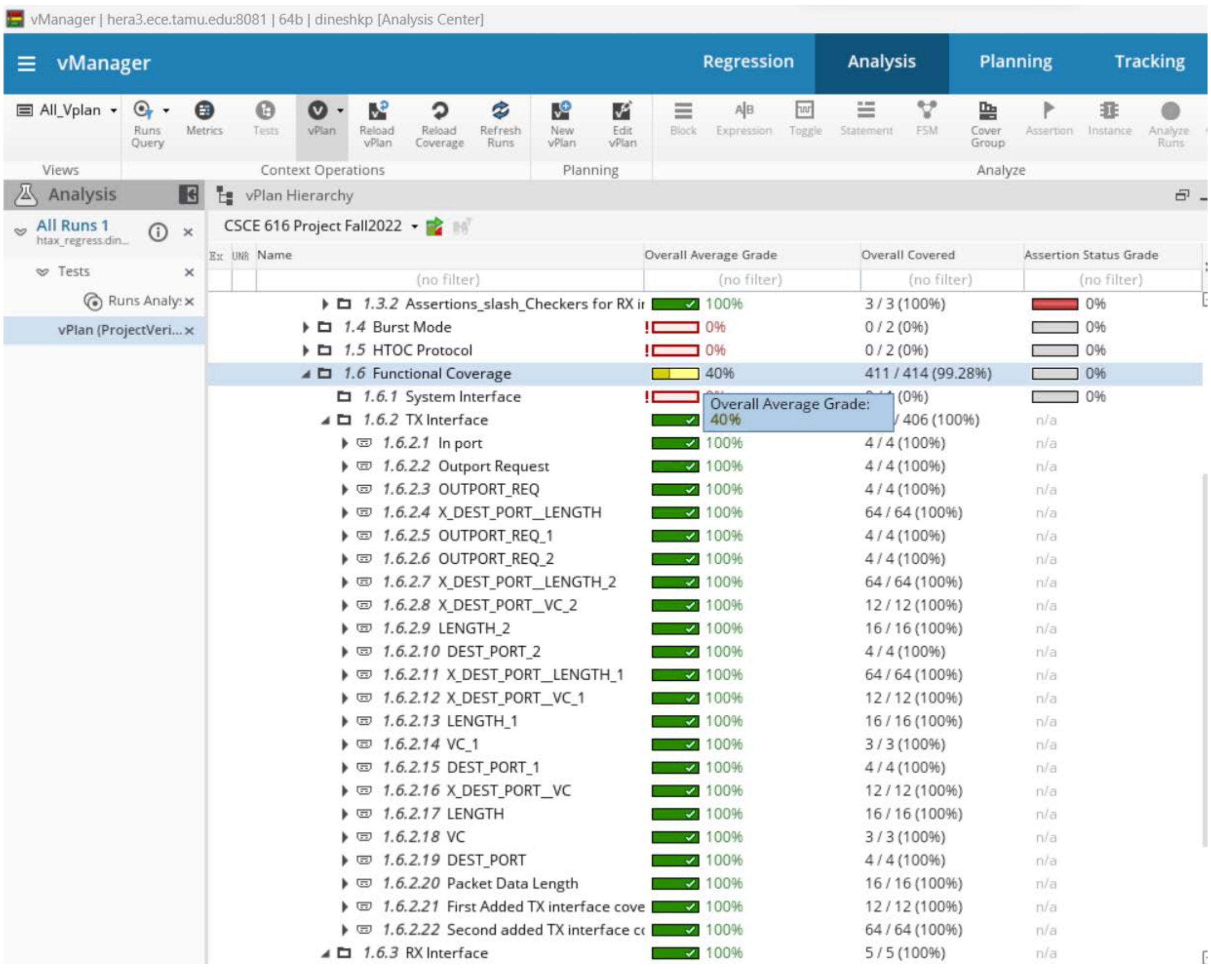
Lab 10 report

Coverage closure report:

Functional coverage is 100% for both TX and RX interface

Initially 100% coverage was not achieved, in order to hit 100% coverage additional testcases were written. It can be seen that all possible values for the fields of packet like VC, tx_outport_req, length and dest_port are being covered.



vManager | hera3.ece.tamu.edu:8081 | 64b | dineshkp [Analysis Center]

vManager — Regression | Analysis | Planning | Tracking

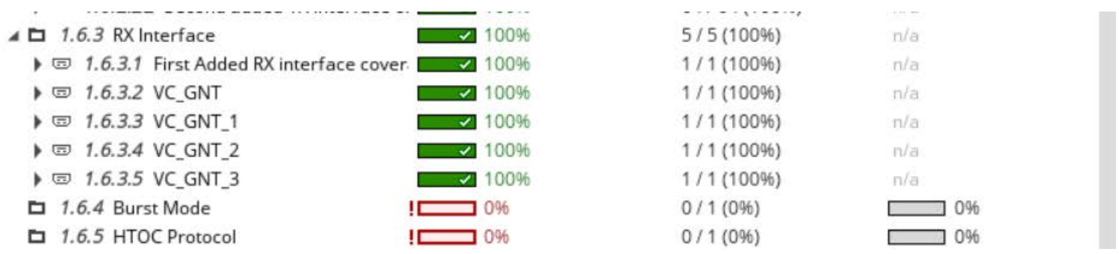
Views: All_Vplan · Context Operations: Runs Query, Metrics, Tests, vPlan, Reload vPlan, Reload Coverage, Refresh Runs · Planning: New vPlan, Edit vPlan · Analyze: Block, Expression, Toggle, Statement, FSM, Cover Group, Assertion, Instance, Analyze Runs

Analysis — All Runs 1 (htax_regress.din...) — Tests — Runs Analy: — vPlan (ProjectVeri...

vPlan Hierarchy — CSCE 616 Project Fall2022

Name	Overall Average Grade	Overall Covered	Assertion Status Grade
(no filter)	(no filter)	(no filter)	(no filter)
1.3.2 Assertions_slash_Checkers for RX ir	100%	3 / 3 (100%)	0%
1.4 Burst Mode	0%	0 / 2 (0%)	0%
1.5 HTOC Protocol	0%	0 / 2 (0%)	0%
1.6 Functional Coverage	40%	411 / 414 (99.28%)	0%
1.6.1 System Interface	[illegible]	[illegible] (0%)	0%
1.6.2 TX Interface	[illegible]	[illegible] / 406 (100%)	n/a
1.6.2.1 In port	100%	4 / 4 (100%)	n/a
1.6.2.2 Outport Request	100%	4 / 4 (100%)	n/a
1.6.2.3 OUTPORT_REQ	100%	4 / 4 (100%)	n/a
1.6.2.4 X_DEST_PORT__LENGTH	100%	64 / 64 (100%)	n/a
1.6.2.5 OUTPORT_REQ_1	100%	4 / 4 (100%)	n/a
1.6.2.6 OUTPORT_REQ_2	100%	4 / 4 (100%)	n/a
1.6.2.7 X_DEST_PORT__LENGTH_2	100%	64 / 64 (100%)	n/a
1.6.2.8 X_DEST_PORT__VC_2	100%	12 / 12 (100%)	n/a
1.6.2.9 LENGTH_2	100%	16 / 16 (100%)	n/a
1.6.2.10 DEST_PORT_2	100%	4 / 4 (100%)	n/a
1.6.2.11 X_DEST_PORT__LENGTH_1	100%	64 / 64 (100%)	n/a
1.6.2.12 X_DEST_PORT__VC_1	100%	12 / 12 (100%)	n/a
1.6.2.13 LENGTH_1	100%	16 / 16 (100%)	n/a
1.6.2.14 VC_1	100%	3 / 3 (100%)	n/a
1.6.2.15 DEST_PORT_1	100%	4 / 4 (100%)	n/a
1.6.2.16 X_DEST_PORT__VC	100%	12 / 12 (100%)	n/a
1.6.2.17 LENGTH	100%	16 / 16 (100%)	n/a
1.6.2.18 VC	100%	3 / 3 (100%)	n/a
1.6.2.19 DEST_PORT	100%	4 / 4 (100%)	n/a
1.6.2.20 Packet Data Length	100%	16 / 16 (100%)	n/a
1.6.2.21 First Added TX interface cove	100%	12 / 12 (100%)	n/a
1.6.2.22 Second added TX interface c	100%	64 / 64 (100%)	n/a
1.6.3 RX Interface	100%	5 / 5 (100%)	n/a

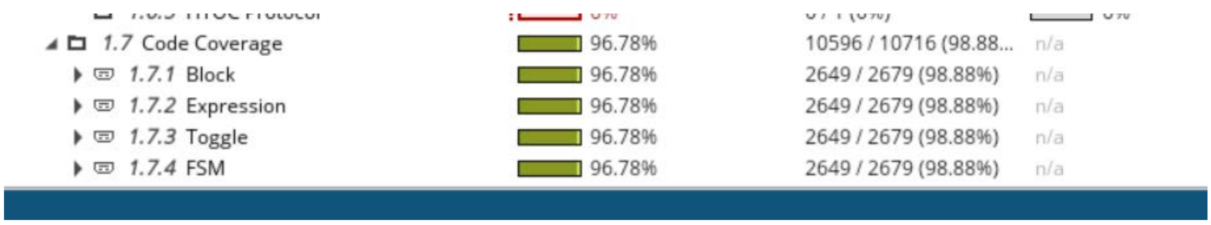
Overall Average Grade: 40%



Name	Overall Average Grade	Overall Covered	Assertion Status Grade
1.6.3 RX Interface	100%	5 / 5 (100%)	n/a
1.6.3.1 First Added RX interface cover	100%	1 / 1 (100%)	n/a
1.6.3.2 VC_GNT	100%	1 / 1 (100%)	n/a
1.6.3.3 VC_GNT_1	100%	1 / 1 (100%)	n/a
1.6.3.4 VC_GNT_2	100%	1 / 1 (100%)	n/a
1.6.3.5 VC_GNT_3	100%	1 / 1 (100%)	n/a
1.6.4 Burst Mode	0%	0 / 1 (0%)	0%
1.6.5 HTOC Protocol	0%	0 / 1 (0%)	0%

Code coverage:

From the below screenshot, it can be seen that the code coverage is above 95% for block, toggle, expression, and FSM.

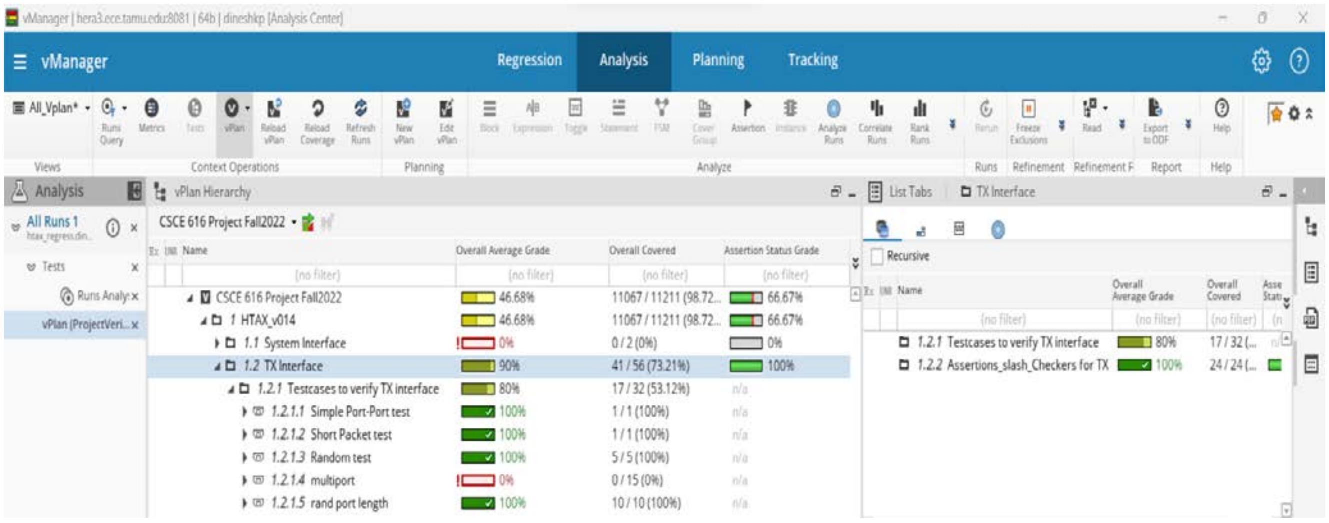


1.7 Code Coverage	96.78%	10596 / 10716 (98.88...	n/a
1.7.1 Block	96.78%	2649 / 2679 (98.88%)	n/a
1.7.2 Expression	96.78%	2649 / 2679 (98.88%)	n/a
1.7.3 Toggle	96.78%	2649 / 2679 (98.88%)	n/a
1.7.4 FSM	96.78%	2649 / 2679 (98.88%)	n/a

Bug report:

Additional tests were included to identify the bug. The htax_random_test file contains the testcases that were used to identify the bug in the design. In this, we generate packets to all the 4 ports parallel using fork-join. Additional constraints were written such that all dest_ports are also covered. Also, packet length and delay has to be kept constant for all 4 packets to hit this bug.

Testcases for TX and RX interface in Vmanager:

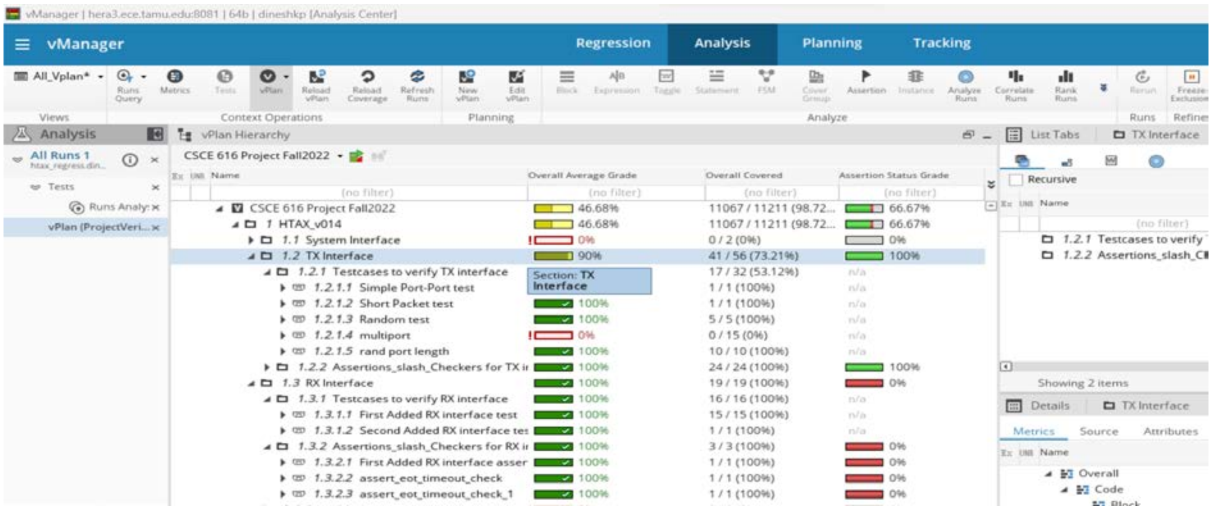


CSCE 616 Project Fall2022

Name	Overall Average Grade	Overall Covered	Assertion Status Grade
CSCE 616 Project Fall2022	46.68%	11067 / 11211 (98.72...	66.67%
1 HTAX_v014	46.68%	11067 / 11211 (98.72...	66.67%
1.1 System Interface	0%	0 / 2 (0%)	0%
1.2 TX Interface	90%	41 / 56 (73.21%)	100%
1.2.1 Testcases to verify TX interface	80%	17 / 32 (53.12%)	n/a
1.2.1.1 Simple Port-Port test	100%	1 / 1 (100%)	n/a
1.2.1.2 Short Packet test	100%	1 / 1 (100%)	n/a
1.2.1.3 Random test	100%	5 / 5 (100%)	n/a
1.2.1.4 multiport	0%	0 / 15 (0%)	n/a
1.2.1.5 rand port length	100%	10 / 10 (100%)	n/a

TX Interface

Name	Overall Average Grade	Overall Covered	Asse Stat
1.2.1 Testcases to verify TX interface	80%	17 / 32 (...	n/
1.2.2 Assertions_slash_Checkers for TX	100%	24 / 24 (...	🟩

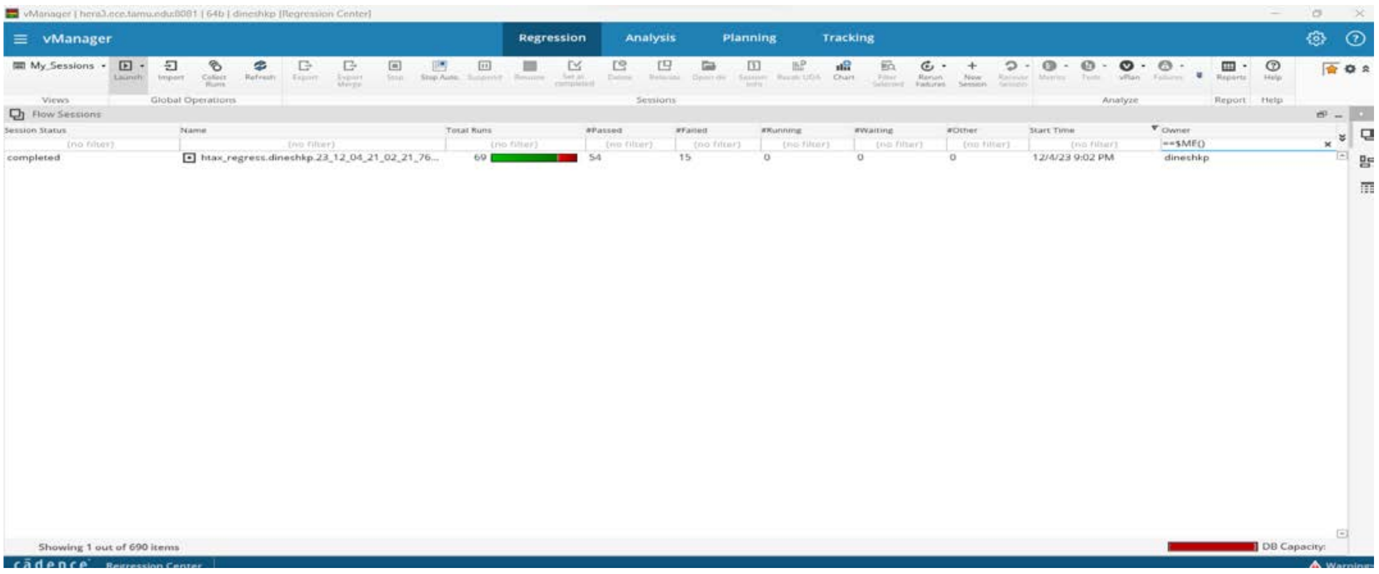


vManager | hera3.ece.tamu.edu:8081 | 64b | dineshkp [Analysis Center]

CSCE 616 Project Fall2022

Name	Overall Average Grade	Overall Covered	Assertion Status Grade
CSCE 616 Project Fall2022	46.68%	11067 / 11211 (98.72...	66.67%
1 HTAX_v014	46.68%	11067 / 11211 (98.72...	66.67%
1.1 System Interface	0%	0 / 2 (0%)	0%
1.2 TX Interface	90%	41 / 56 (73.21%)	100%
1.2.1 Testcases to verify TX interface	Section: TX Interface	17 / 32 (53.12%)	n/a
1.2.1.1 Simple Port-Port test		1 / 1 (100%)	n/a
1.2.1.2 Short Packet test	100%	1 / 1 (100%)	n/a
1.2.1.3 Random test	100%	5 / 5 (100%)	n/a
1.2.1.4 multiport	0%	0 / 15 (0%)	n/a
1.2.1.5 rand port length	100%	10 / 10 (100%)	n/a
1.2.2 Assertions_slash_Checkers for TX ir	100%	24 / 24 (100%)	100%
1.3 RX Interface	100%	19 / 19 (100%)	0%
1.3.1 Testcases to verify RX interface	100%	16 / 16 (100%)	n/a
1.3.1.1 First Added RX interface test	100%	15 / 15 (100%)	n/a
1.3.1.2 Second Added RX interface tes	100%	1 / 1 (100%)	n/a
1.3.2 Assertions_slash_Checkers for RX ir	100%	3 / 3 (100%)	0%
1.3.2.1 First Added RX interface asser	100%	1 / 1 (100%)	0%
1.3.2.2 assert_eot_timeout_check	100%	1 / 1 (100%)	0%
1.3.2.3 assert_eot_timeout_check_1	100%	1 / 1 (100%)	0%

In the above figure, only the testcase that contains the bug tests fails.



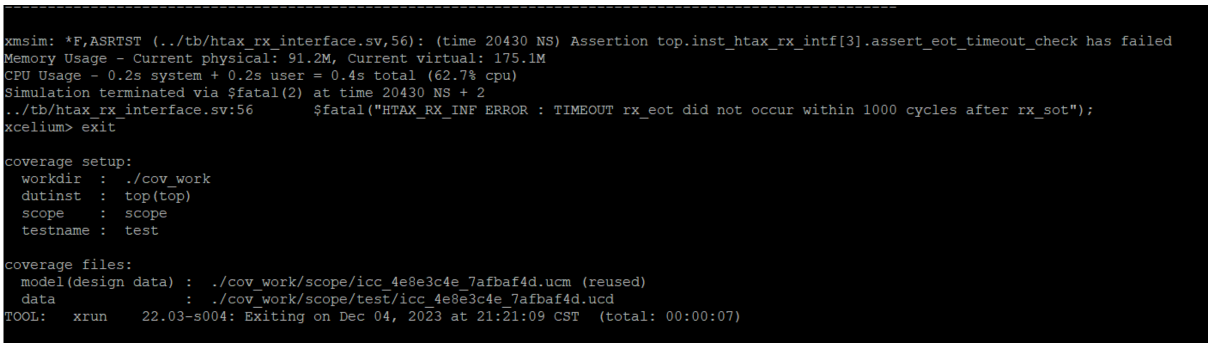
vManager | hera3.ece.tamu.edu:8081 | 64b | dineshkp [Regression Center]

Session Status	Name	Total Runs	#Passed	#Failed	#Running	#Waiting	#Other	Start Time	Owner
completed	htax_regress.dineshkp.23_12_04_21_02_21_76...	69	54	15	0	0	0	12/4/23 9:02 PM	dineshkp

Showing 1 out of 690 items

The test that includes the testcases for finding the bug has been called 15 times. So these tests 15 fails and other tests pass.

The above test is executed using the irun command and the assertion fails.



```
xmsim: *F,ASRTST (../tb/htax_rx_interface.sv,56): (time 20430 NS) Assertion top.inst_htax_rx_intf[3].assert_eot_timeout_check has failed
Memory Usage - Current physical: 91.2M, Current virtual: 175.1M
CPU Usage - 0.2s system + 0.2s user = 0.4s total (62.7% cpu)
Simulation terminated via $fatal(2) at time 20430 NS + 2
../tb/htax_rx_interface.sv:56       $fatal("HTAX_RX_INF ERROR : TIMEOUT rx_eot did not occur within 1000 cycles after rx_sot");
xcelium> exit

coverage setup:
  workdir  :  ./cov_work
  dutinst  :  top(top)
  scope    :  scope
  testname :  test

coverage files:
  model(design data) :  ./cov_work/scope/icc_4e8e3c4e_7afbaf4d.ucm (reused)
  data               :  ./cov_work/scope/test/icc_4e8e3c4e_7afbaf4d.ucd
TOOL:    xrun    22.03-s004: Exiting on Dec 04, 2023 at 21:21:09 CST  (total: 00:00:07)
```

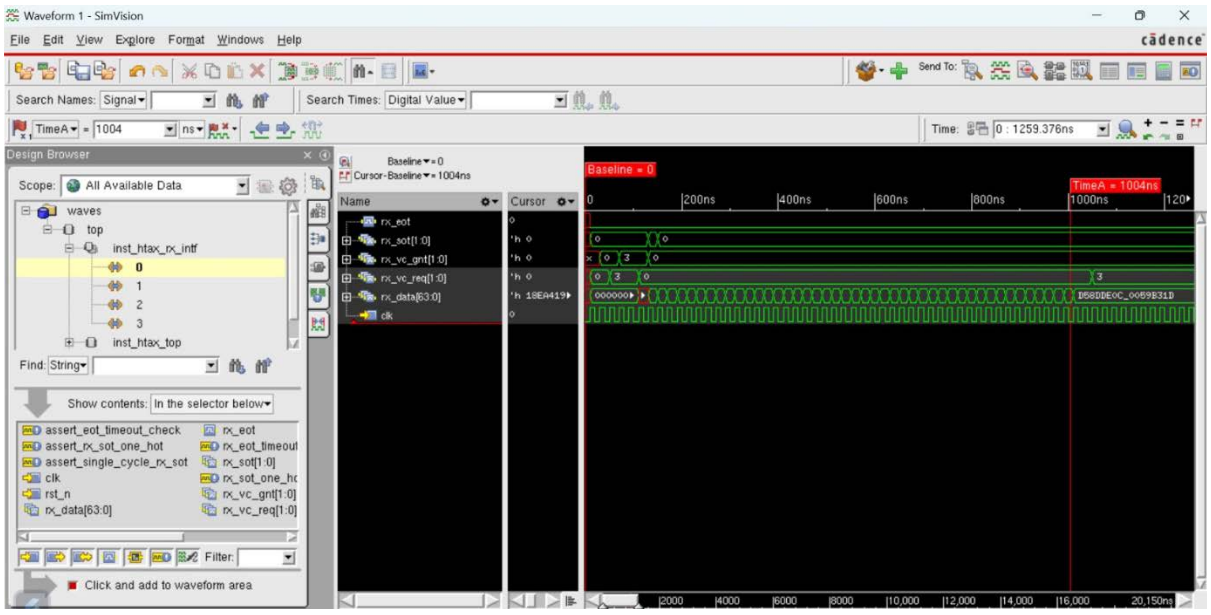
Bug in the code:

When packets are generated in all ports the eot_in become 1. The complement of it produces a 0 and thus the AND with it makes eot always 0.

```
    always @( * )
    begin
            (* full_case *) (* parallel_case *)
            casex (inport_sel)
                    4'b1xxx: selected_sot = sot_in[((4*VC)-1):(3*VC)];
                    4'bx1xx: selected_sot = sot_in[((3*VC)-1):(2*VC)];
                    4'bxx1x: selected_sot = sot_in[((2*VC)-1):(1*VC)];
                    4'bxxx1: selected_sot = sot_in[((1*VC)-1):(0*VC)];
            endcase
    end

    assign selected_eot = |(eot_in & inport_sel_reg) & ~(&(eot_in));
```

Waveform with eot not asserted:



From the above waveform, it is clear that the rx_eot signal is not asserted for any packet that is transmitted. But, it is supposed to be asserted when the final packet is transmitted for each transaction. Tracing this bug in the design, it was found that a NOT of bitwise AND was performed on eot_in.

 Due to this, whenever all the ports are accessed, all the bits become high and the bitwise AND becomes 1. Hence, a complement of this will give a 0. So the EOT signal is 0 at all times.

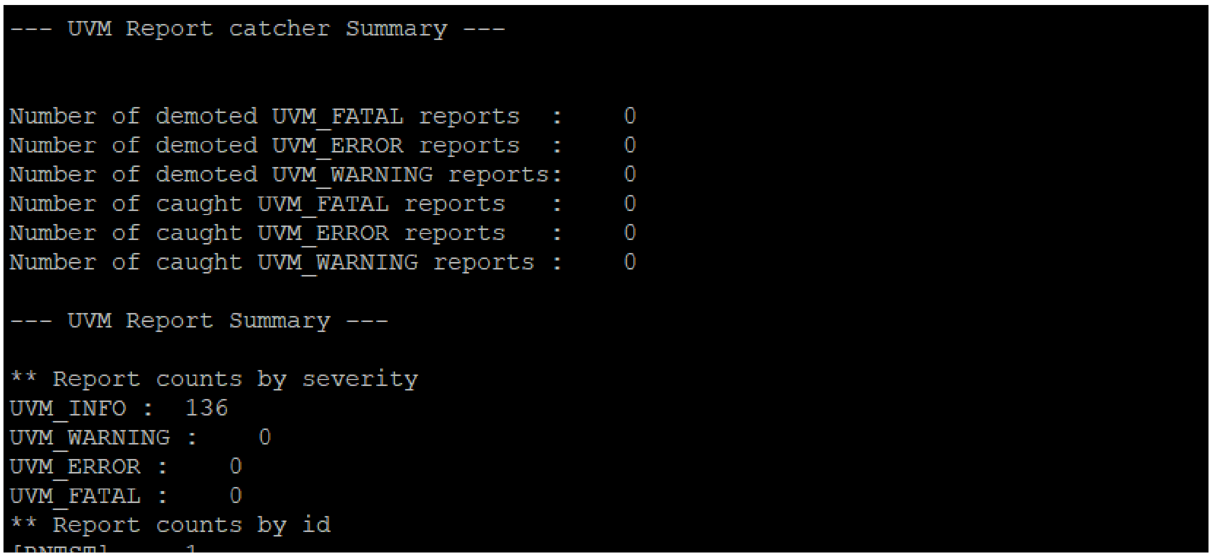
To fix this, we can remove the bitwise AND part and this fixes the bug.

Fixed code:

```
always @( * )
begin
        (* full_case *) (* parallel_case *)
        casex (inport_sel)
                4'b1xxx: selected_sot = sot_in[((4*VC)-1):(3*VC)];
                4'bx1xx: selected_sot = sot_in[((3*VC)-1):(2*VC)];
                4'bxx1x: selected_sot = sot_in[((2*VC)-1):(1*VC)];
                4'bxxx1: selected_sot = sot_in[((1*VC)-1):(0*VC)];
        endcase
end

assign selected_eot = |(eot_in & inport_sel_reg);
```

After fixing the code the UVM fatal does not occur:



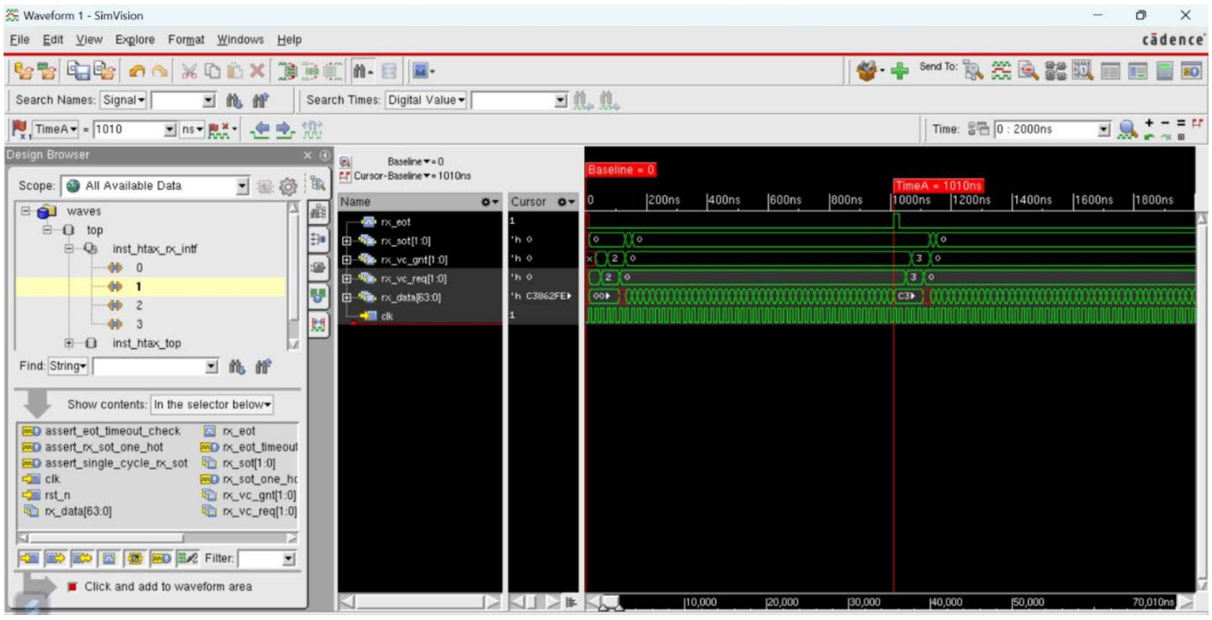
```
--- UVM Report catcher Summary ---


Number of demoted UVM_FATAL reports  :    0
Number of demoted UVM_ERROR reports  :    0
Number of demoted UVM_WARNING reports:    0
Number of caught UVM_FATAL reports   :    0
Number of caught UVM_ERROR reports   :    0
Number of caught UVM_WARNING reports :    0

--- UVM Report Summary ---

** Report counts by severity
UVM_INFO :  136
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]      1
```

EOT gets asserted in the waveform after fixing the bug:



Testcases added:

```
task body();


      repeat(5) begin
      //port = $urandom_range(0,3);
      fork
    `uvm_do_on_with(pkt0, p_sequencer.htax_seqr[0], { pkt0.length ==40; pkt0.dest_port == 0; pkt0.delay==15;} )
                      //USE `uvm_do_on_with to add constraints on req
    `uvm_do_on_with(pkt1,  p_sequencer.htax_seqr[1], { pkt1.length ==40; pkt1.dest_port == 1;pkt1.delay==15;} )
    `uvm_do_on_with(pkt2, p_sequencer.htax_seqr[2], { pkt2.length ==40 ; pkt2.dest_port == 2;pkt2.delay==15;} )
                      //USE `uvm_do_on_with to add constraints on req
    `uvm_do_on_with(pkt3,  p_sequencer.htax_seqr[3], { pkt3.length ==40 ; pkt3.dest_port == 3;pkt3.delay==15;} )

      join

   end
```

```
task body();
              // Exectuing 10 TXNs on ports {0,1,2,3} randomly

      repeat(2) begin
      port = $urandom_range(0,3);

      for(int i=0;i<4;i++)
              for(int j=3;j<64;j++)

    `uvm_do_on_with(req, p_sequencer.htax_seqr[port], { req.dest_port == i; req.length == j;} )

                      //USE `uvm_do_on_with to add constraints on req
  end
```

```
endfunction : new

task body();


      repeat(5) begin
      delay = $urandom_range(1,20);

    `uvm_do_on_with(req, p_sequencer.htax_seqr[0], { req.delay==delay;} )
                        //USE `uvm_do_on_with to add constraints on req


  end
```

```
endfunction : new

task body();
              // Exectuing 10 TXNs on ports {0,1,2,3} randomly

      repeat(5) begin

      `uvm_do_on_with(req, p_sequencer.htax_seqr[0], { req.dest_port == 0;} )
      `uvm_do_on_with(req, p_sequencer.htax_seqr[0], { req.dest_port == 1;} )
      `uvm_do_on_with(req, p_sequencer.htax_seqr[0], { req.dest_port == 2;} )
      `uvm_do_on_with(req, p_sequencer.htax_seqr[0], { req.dest_port == 3;} )

                        //USE `uvm_do_on_with to add constraints on req
  end



endtask : body
```

Adding testcases in vsif file:

```
    -------------------------------------------------------------
    -- Add your tests here ------------------------------------
    -------------------------------------------------------------
 test test7 {
          run_script: "cd $ENV(PWD) ; irun -f run_vm.f +UVM_TESTNAME=htax_random_test -define TEST7" ;
          scan_script: "vm_scan.pl ius.flt shell.flt" ;
          count : 15;
     };

test test8 {
          run_script: "cd $ENV(PWD) ; irun -f run_vm.f +UVM_TESTNAME=htax_random_test2 -define TEST8" ;
          scan_script: "vm_scan.pl ius.flt shell.flt" ;
          count : 10;
     };

test test9 {
          run_script: "cd $ENV(PWD) ; irun -f run_vm.f +UVM_TESTNAME=htax_random_test3 -define TEST9" ;
          scan_script: "vm_scan.pl ius.flt shell.flt" ;
          count : 10;
     };

test test10 {
          run_script: "cd $ENV(PWD) ; irun -f run_vm.f +UVM_TESTNAME=htax_random_test4 -define TEST10" ;
          scan_script: "vm_scan.pl ius.flt shell.flt" ;
          count : 10;
     };

};
```

Assertions:

```
//ASSERTIONS

   // --------------------------
   // tx_outport_req is one-hot
   // --------------------------
   property tx_outport_req_one_hot;
      @(posedge clk) disable iff(!rst_n)
      (|tx_outport_req) |-> $onehot(tx_outport_req);
   endproperty

   assert_tx_outport_req_one_hot : assert property(tx_outport_req_one_hot)
   else
      $error("HTAX_TX_INF ERROR : tx_outport request is not one hot encoded");

   // ----------------------------------
   // no tx_outport_req without tx_vc_req
   // ----------------------------------
   property tx_outport_req_vc_req;
     @(posedge clk) disable iff(!rst_n)
     (~(|tx_outport_req) ##1 (|tx_outport_req)) |-> ( (|tx_vc_req) && ~($past(tx_vc_req)));
   endproperty

   assert_tx_outport_req_vc_req : assert property(tx_outport_req_vc_req)
   else
      $error("HTAX_TX_INF ERROR : tx_outport_req high without tx_vc_req");

   // ----------------------------------
   // no tx_vc_req without tx_outport_req
   // ----------------------------------
   property tx_vc_req_outport_req;
      @(posedge clk) disable iff(!rst_n)
   //     $rose(tx_vc_req) |-> $rose(tx_outport_req);
      (~(|tx_vc_req) ##1 (|tx_vc_req)) |-> ( (|tx_outport_req) && ~($past(tx_outport_req)) && $onehot(tx_outport_req));
   endproperty

   assert_tx_vc_req_outport_req : assert property(tx_vc_req_outport_req)
   else
      $error("HTAX_TX_INF ERROR : tx_vc_req high without tx_outport_req");
```

```
   assert_tx_vc_req_outport_req : assert property(tx_vc_req_outport_req)
   else
      $error("HTAX_TX_INF ERROR : tx_vc_req high without tx_outport_req");
```

```
   // ------------------------------------
   // no tx_sot without previous tx_vc_gnt
   // ------------------------------------
   property tx_vc_sot_vc_gnt(int i);
      @(posedge clk) disable iff(!rst_n)
      $rose(tx_sot[i]) |-> $past(tx_vc_gnt[i]);
   endproperty

   assert_tx_vc_sot_vc_gnt_0 : assert property(tx_vc_sot_vc_gnt(0))
   else
      $error("HTAX_TX_INF ERROR : tx_sot[0] raised without previous vc_gnt[0]");

   assert_tx_vc_sot_vc_gnt_1 : assert property(tx_vc_sot_vc_gnt(1))
   else
      $error("HTAX_TX_INF ERROR : tx_sot[1] raised without previous vc_gnt[1]");

   // ---------------------------------------------
   // tx_eot is asserted for a single clock cycle
   // ---------------------------------------------
   property tx_eot_single_cycle;
      @(posedge clk) disable iff(!rst_n)
      $rose(tx_eot) |=> $fell(tx_eot);
   endproperty

   assert_tx_eot_single_cycle : assert property(tx_eot_single_cycle)
   else
      $error("HTAX_TX_INF ERROR : tx_eot is not high for exactly one clock cycle");

   // ------------------------------------------------------------
   // tx_release_gnt one clock cycle before or same cycle as tx_eot
   // ------------------------------------------------------------
   property tx_rel_gnt_tx_eot;
      @(posedge clk) disable iff(!rst_n)
      $rose(tx_release_gnt) |-> ##[0:1]  $rose(tx_eot);
   endproperty

   assert_tx_rel_gnt_tx_eot : assert property(tx_rel_gnt_tx_eot)
```

Creating new packets:

```
    htax_packet_c pkt0, pkt1, pkt2, pkt3;

    `uvm_object_utils(htax_base_vseq)
    `uvm_declare_p_sequencer(htax_virtual_sequencer_c)

    function new (string name = "htax_base_vseq");
            super.new(name);
            pkt0 = new();
            pkt1 = new();
            pkt2 = new();
            pkt3 = new();
    endfunction : new
```