CSCE 616: Introduction to Hardware Design Verification

Lab-10: HTAX Regression

Name: Sai Surendra Reddy Yaraballi

UIN: 333002926

The added assertions in the htax_tx_interface

```
//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)                 //assert the property
    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_without_tx_vc_req;
        @(posedge clk) disable iff(!rst_n)
        $rose(|tx_vc_req) |-> $rose(|tx_outport_req);
    endproperty

    assert_tx_outport_req_without_tx_vc_req : assert property(tx_outport_req_without_tx_vc_req)        //assert the property
    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_without_tx_outport_req;
        @(posedge clk) disable iff(!rst_n)
        $rose(|tx_outport_req) |-> $rose(|tx_vc_req);
    endproperty
```

```
    assert_tx_vc_req_without_tx_outport_req : assert property(tx_vc_req_without_tx_outport_req)          //assert the property
    else
        $error("HTAX_TX_INF ERROR : tx_vc_req high without tx_vc_req tx_outport_req");


    //-----------------------------------
    // tx_vc_gnt is subset of vc_request
    //-----------------------------------
    property tx_vc_gnt_subset_vc_request;
        @(posedge clk) disable iff(!rst_n)
        $rose(|tx_vc_req) |-> (tx_vc_gnt == (tx_vc_gnt & tx_vc_req));
    endproperty

    assert_tx_vc_gnt_subset_vc_request : assert property(tx_vc_gnt_subset_vc_request)                   //assert the property
    else
        $error("HTAX_TX_INF ERROR : tx_vc_gnt didn't rise or fall within the tx_vc_req");


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


    assert_tx_sot_without_tx_vc_gnt : assert property(tx_sot_without_tx_vc_gnt(0))                      //assert the property
    else
        $error("HTAX_TX_INF ERROR : tx_sot high when tx_vc_gnt");
```

```
    // ------------------------------------
    // no tx_eot without previous tx_vc_gnt
    // ------------------------------------
    property tx_eot_without_previous_tx_vc_gnt;
        @(posedge clk) disable iff(!rst_n)
        $rose(tx_eot) |-> (tx_eot | $past(tx_vc_gnt));
        //@(posedge (|tx_vc_gnt)) $rose(tx_eot) |-> 1;
        //$rose(tx_eot) -> tran_on;
    endproperty

    assert_tx_eot_without_previous_tx_vc_gnt : assert property(tx_eot_without_previous_tx_vc_gnt)          //assert the property
    else
        $error("HTAX_TX_INF ERROR : tx_eot high when tx_vc_gnt");


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

    assert_tx_eot_single_clk_cycle : assert property(tx_eot_single_clk_cycle)                             //assert the property
    else
        $error("HTAX_TX_INF ERROR : tx_eot not high for a single clk cycle");


    // ---------------------------------------------------------------
    // tx_release_gnt for pkt(t) one clock cycle or same clock cycle with tx_eot of pkt(t)
    // ---------------------------------------------------------------
    property tx_release_gnt_pkt_tx_eot;
        @(posedge clk) disable iff(!rst_n)
        $rose(tx_release_gnt) |-> ##[0:1] $rose(tx_eot);
    endproperty
```

```
    assert_tx_release_gnt_pkt_tx_eot : assert property(tx_release_gnt_pkt_tx_eot)                         //assert the property
    else
        $error("HTAX_TX_INF ERROR : tx_release_gnt not high for one cycle for the pkt");


    // ---------------------------------------------------------
    // No tx_sot of p(t+1) without tx_eot for p(t)
    // ---------------------------------------------------------
    property tx_sot_without_tx_eot(int i);
        @(posedge clk) disable iff(!rst_n)
        $rose(|tx_sot[i]) |-> (|tx_sot[i] | $past(tx_eot));
    endproperty

    assert_tx_sot_without_tx_eot : assert property(tx_sot_without_tx_eot(1))                              //assert the property
    else
        $error("HTAX_TX_INF ERROR : tx_sot high without tx_eot");


    // ---------------------------------------------------------
    // Valid packet transfer - rise of tx_outport_req followed by a tx_vc_gnt followed by tx_sot
     // followed by tx_release_gnt followed by tx_eot. Consider the right timings between each event.
    // ---------------------------------------------------------
    property valid_packet_transfer;
        @(posedge clk) disable iff (!rst_n)


        ($past(tx_outport_req) && $past(tx_vc_req) && $past(tx_vc_gnt) && tx_data && $rose(tx_sot)) |->  (tx_data throughout (##[1:64] ##1 tx_release_gnt || tx_eot));
    endproperty

    assert_valid_packet_transfer : assert property(valid_packet_transfer)                                 //assert the property
    else
        $error("HTAX_TX_INF ERROR : Valid Packet is not transferring properly");
endinterface : htax_tx_interface
```

For coverage analysis, added some cover points and cross cover points in htax_rx_monitor_c

```
10
11      //Analysis port to communicate with Scoreboard
12      uvm_analysis_port #(htax_rx_mon_packet_c) rx_collect_port;
13
14      virtual interface htax_rx_interface htax_rx_intf;
15      htax_rx_mon_packet_c rx_mon_packet;
16      int pkt_len;
17
18      covergroup cover_htax_packet;
19      option.per_instance = 1;
20      option.name = "cover_htax_packet";
21
22
23      //Coverpoint for htax packet field : length
24      LENGTH : coverpoint rx_mon_packet.length  {
25                                                  bins length[16] = {[0:63]};
26                                                                                  illegal_bins len = {[0:2]};
27                                                  }
28
29  endgroup
```

The test code for long packet is given below:

C: > Users > saisu > AppData > Roaming > MobaXterm > slash > RemoteFiles > 985620_2_10 > port_test.sv

```
1
2
3   class port_test extends base_test;
4
5       `uvm_component_utils(port_test)
6
7       function new (string name, uvm_component parent);
8           super.new(name, parent);
9       endfunction : new
10
11      function void build_phase(uvm_phase phase);
12          uvm_config_wrapper::set(this, "tb.vsequencer.run_phase", "default_sequence", random_port_vsequence::type_id::get());
13          super.build_phase(phase);
14      endfunction : build_phase
15
16      task run_phase(uvm_phase phase);
17          super.run_phase(phase);
18          `uvm_info(get_type_name(), "starting new port test",UVM_NONE)
19      endtask : run_phase
20
21  endclass  : port_test
22
23
24  ////////////////////// Virtual Sequence ////////////////////////
25
26  class random_port_vsequence extends htax_base_vseq;
27
28      `uvm_object_utils(random_port_vsequence)
29
30      rand int port;
31      rand int length;
32
33      function new (string name = "random_port_vsequence");
34          super.new(name);
35      endfunction : new
36
37      task body();
38          // Exectuing 10 TXNs on ports {0,1,2,3} randomly
39      //fork
40      repeat(1500) begin
41        port = $urandom_range(0,3);
42        //length = $urandom_range(11,60);
43
44        //`uvm_do_on(req, p_sequencer.htax_seqr[port])
45
46            //USE `uvm_do_on_with to add constraints on req
47
48          `uvm_do_on_with(req, p_sequencer.htax_seqr[port], {req.length inside {[40:63]}; req.delay <= 5;} )
49
50      end
51     //join_none
52
53      endtask : body
54
55  endclass    :   random_port_vsequence
```

The UVM_summary for long packet is

```
--- 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 : 9016
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[SCOREBOARD]  6005
[TEST_DONE]     1
[TOP]     5
[UVMTOP]     1
[htax_tx_driver_c]  3000
[port_test]     1
[random_port_vsequence]     2
Simulation complete via $finish(1) at time 1810150 NS + 45
/opt/coe/cadence/XCELIUM/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_root.svh:457     $finish;
xcelium> exit

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

The test code for short packet data is

```
class short_packet_test extends base_test;

    `uvm_component_utils(short_packet_test)

    function new (string name, uvm_component parent);
        super.new(name, parent);
    endfunction : new

    function void build_phase(uvm_phase phase);
        uvm_config_wrapper::set(this, "tb.vsequencer.run_phase", "default_sequence", short_packet_vsequence::type_id::get());
        super.build_phase(phase);
    endfunction : build_phase

    task run_phase(uvm_phase phase);
        super.run_phase(phase);
        `uvm_info(get_type_name(), "starting new port test",UVM_NONE)
    endtask : run_phase

endclass  : short_packet_test


////////////////////// Virtual Sequence /////////////////////////

class short_packet_vsequence extends htax_base_vseq;

    `uvm_object_utils(short_packet_vsequence)

    rand int port;
    rand int length;

    function new (string name = "short_packet_vsequence");
        super.new(name);
    endfunction : new
```

```
    task body();
       // Exectuing 10 TXNs on ports {0,1,2,3} randomly
    //fork
    repeat(1500) begin
      port = $urandom_range(0,3);
      //length = $urandom_range(3,10);

      //`uvm_do_on(req, p_sequencer.htax_seqr[port])

          //USE `uvm_do_on_with to add constraints on req

        `uvm_do_on_with(req, p_sequencer.htax_seqr[port], {req.length inside {[3:10]}; req.delay <=5;} )

    end
    //join_none

    endtask : body

endclass    :   short_packet_vsequence
```

The UVM summary for short packet test is

```
--- 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 : 9016
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[SCOREBOARD]  6005
[TEST_DONE]     1
[TOP]     5
[UVMTOP]     1
[htax_tx_driver_c]  3000
[short_packet_test]     1
[short_packet_vsequence]     2
Simulation complete via $finish(1) at time 458050 NS + 45
/opt/coe/cadence/XCELIUM/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_root.svh:457     $finish;
xcelium> exit
```

The code for parallel_port testcase is shown below:

```
1   ////////////////////////////////////////////////////////////////////////////
2   // Texas A&M University
3   // CSCE 616 Hardware Design Verification
4   // Created by  : Prof. Quinn and Saumil Gogri
5   ////////////////////////////////////////////////////////////////////////////
6
7
8   class parallel_port extends base_test;
9
10      `uvm_component_utils(parallel_port)
11
12      function new (string name, uvm_component parent);
13          super.new(name, parent);
14      endfunction : new
15
16      function void build_phase(uvm_phase phase);
17          uvm_config_wrapper::set(this,"tb.vsequencer.run_phase", "default_sequence", parallel_port_vsequence::type_id::get());
18          super.build_phase(phase);
19      endfunction : build_phase
20
21      task run_phase(uvm_phase phase);
22          super.run_phase(phase);
23          `uvm_info(get_type_name(),"Starting parallel port test",UVM_NONE)
24      endtask : run_phase
25
26  endclass : parallel_port
27
28
29
30  ////////////////////////////// VIRTUAL SEQUENCE ////////////////////////////
31
32
33  class parallel_port_vsequence extends htax_base_vseq;
34
35    `uvm_object_utils(parallel_port_vsequence)
```

```
36
37    htax_packet_c req[5];
38    rand int port, length;
39    rand int i, j;
40
41    function new (string name = "parallel_port_vsequence");
42      super.new(name);
43      for(int i=0; i<5; i++) begin
44          req[i] = new();
45      end
46    endfunction : new
47
48    task body();
49          // Exectuing 10 TXNs on ports {0,1,2,3} randomly
50      repeat(1500) begin
51
52        //port = $urandom_range(0,3);
53        //i = $urandom_range(0,7);
54        //j = $urandom_range(0,7);
55        length = $urandom_range(3,10);
56              fork begin
57              //USE `uvm_do_on_with to add constraints on req
58              //`uvm_do_on(req, p_sequencer.htax_seqr[port])
59              `uvm_do_on_with(req[0], p_sequencer.htax_seqr[0], {req[0].delay<=5; req[0].length <= 10;})
60              end
61              begin
62              `uvm_do_on_with(req[1], p_sequencer.htax_seqr[1], {req[1].delay<=5; req[1].length <= 10;})
63              end
64              begin
65              `uvm_do_on_with(req[2], p_sequencer.htax_seqr[2], {req[2].delay<=5; req[2].length <=10;})
66              end
67              begin
68              `uvm_do_on_with(req[3], p_sequencer.htax_seqr[3], {req[3].delay<=5; req[3].length <=10;})
69              end
70              join
```

```
70              join
71              //`uvm_do_on(req, p_sequencer.htax_seqr[port])
72              //`uvm_do_on_with(req[7], p_sequencer.htax_seqr[3], {req[7].delay<=5; req[7].length == length;})
73              //`uvm_do_on_with(req[6], p_sequencer.htax_seqr[2], {req[6].delay<=5; req[6].length== length;})
74              //`uvm_do_on_with(req[5], p_sequencer.htax_seqr[1], {req[5].delay<=5; req[5].length== length;})
75              `uvm_do_on_with(req[4], p_sequencer.htax_seqr[0], {req[4].delay<=5; req[4].length<=10;})
76      end
77
78    endtask : body
79
80  endclass : parallel_port_vsequence
81
```

The UVM error free for the parallel_port testcase is shown below

```
--- 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 :45016
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[SCOREBOARD] 30005
[TEST_DONE]     1
[TOP]     5
[UVMTOP]     1
[htax_tx_driver_c] 15000
[parallel_port]     1
[parallel_port_vsequence]     2
Simulation complete via $finish(1) at time 1170930 NS + 45
/opt/coe/cadence/XCELIUM/tools/methodology/UVM/CDNS-1.1d/sv/src/base/uvm_root.svh:457     $finish;
xcelium> exit
```

Please refer to the bug report and closure report