LAB # 37 Report
Creating a Stimulus Model, Test and Testbench Components

Task 1: Creating Data Item and a Simple Test

1- Defined a yapp_packet class by deriving it from the uvm_sequence_item base class. Then defined the fields according to the specification.

Using uvm macro registered this class in the uvm factory and defined the constructor using the uvm data constructor.

2- added support for randomization of the packet by declaring the length, payload, addr fields as rand. Created a method to calculate and return the correct parity.

Enumeration Declaration:

An enum named parity_type_e was defined outside the class with values GOOD_PARITY and BAD_PARITY, representing parity correctness control.

Parity Control Knob:

A rand property named parity_type of type parity_type_e was declared to randomly control whether the packet uses correct or incorrect parity. set_parity() Method. A method set_parity() was implemented to set the packet's parity. If parity_type is GOOD_PARITY, parity is calculated using the calc_parity() method.If BAD_PARITY, an incorrect parity value is assigned.

post_randomize() Override:

The post_randomize() method was overridden to call set_parity() after successful randomization.

Address Constraint:

A constraint was added to ensure that the address field of the packet remains within valid bounds.

Length and Payload Constraints

The length field of the packet was constrained.

The payload.size() was constrained to match the length, ensuring data consistency.

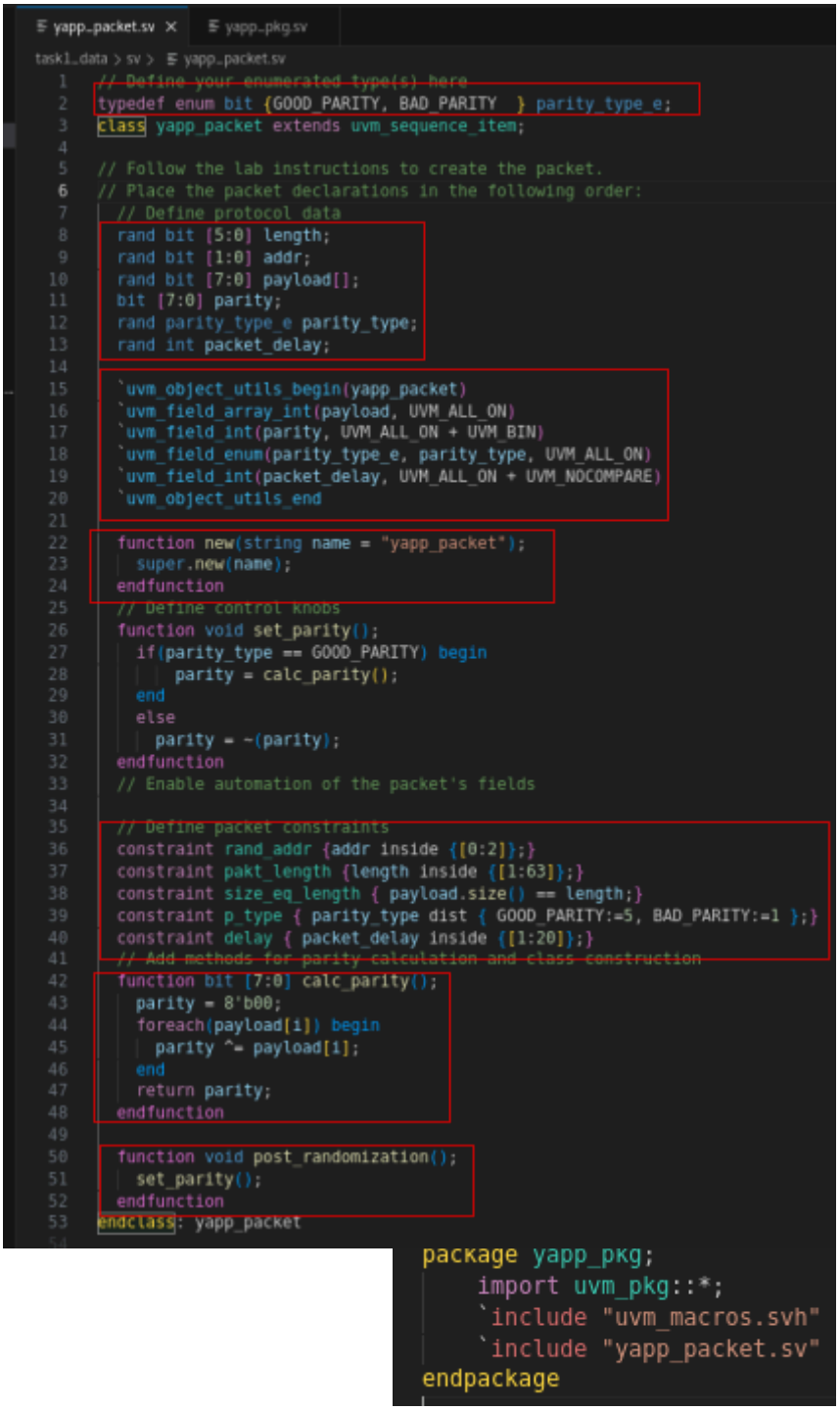
Parity Distribution Constraint:

A weighted distribution constraint was applied to parity_type to favor

GOOD_PARITY over BAD_PARITY in a 5:1 ratio, ensuring mostly correct parity packets with occasional faults.

Delay Control Knob:

A new randomized integer property packet_delay was added to control transmission delay in clock cycles. It was constrained to lie in the range 1 to 20, simulating realistic timing variations.



```
// Define your enumerated type(s) here
typedef enum bit {GOOD_PARITY, BAD_PARITY  } parity_type_e;
class yapp_packet extends uvm_sequence_item;

// Follow the lab instructions to create the packet.
// Place the packet declarations in the following order:
  // Define protocol data
  rand bit [5:0] length;
  rand bit [1:0] addr;
  rand bit [7:0] payload[];
  bit [7:0] parity;
  rand parity_type_e parity_type;
  rand int packet_delay;

  `uvm_object_utils_begin(yapp_packet)
  `uvm_field_array_int(payload, UVM_ALL_ON)
  `uvm_field_int(parity, UVM_ALL_ON + UVM_BIN)
  `uvm_field_enum(parity_type_e, parity_type, UVM_ALL_ON)
  `uvm_field_int(packet_delay, UVM_ALL_ON + UVM_NOCOMPARE)
  `uvm_object_utils_end

  function new(string name = "yapp_packet");
    super.new(name);
  endfunction
  // Define control knobs
  function void set_parity();
    if(parity_type == GOOD_PARITY) begin
        parity = calc_parity();
    end
    else
      parity = ~(parity);
  endfunction
  // Enable automation of the packet's fields

  // Define packet constraints
  constraint rand_addr {addr inside {[0:2]};}
  constraint pakt_length {length inside {[1:63]};}
  constraint size_eq_length { payload.size() == length;}
  constraint p_type { parity_type dist { GOOD_PARITY:=5, BAD_PARITY:=1 };}
  constraint delay { packet_delay inside {[1:20]};}
  // Add methods for parity calculation and class construction
  function bit [7:0] calc_parity();
    parity = 8'b00;
    foreach(payload[i]) begin
      parity ^= payload[i];
    end
    return parity;
  endfunction

  function void post_randomization();
    set_parity();
  endfunction
endclass: yapp_packet
```

```
package yapp_pkg;
    import uvm_pkg::*;
    `include "uvm_macros.svh"
    `include "yapp_packet.sv"
endpackage
```

Creating a Simple Test:

The top-level test module (top.sv) was modified to include the UVM library, macros, and the custom yapp_pkg. An instance of the yapp_packet was created, and five random packets were generated and printed using UVM's print() method inside an initial block.

The run.f file was updated to include source files and directories, and simulation was performed using xrun -f run.f. UVM automation features—copy(), clone(), and compare()—were explored. Various print formats (table, tree, and line) were demonstrated using uvm_default_*_printer settings to observe structured packet output.

Simulation Results using Built_in UVM compare method and using default table printer

```
module top;
// import the UVM library
// include the UVM macros
import uvm_pkg::*;
`include "uvm_macros.svh"
// import the YAPP package
import yapp_pkg::*;
// generate 5 random packets and use the print
// to display the results

initial begin
   yapp_packet p1, p2, p3;
   p1 = new("p1");
   p2 = new("p2");

      assert(p1.randomize());
      p1.print();
      // p1.print(uvm_default_tree_printer);
      // p1.print(uvm_default_line_printer);
      assert(p2.randomize());
      // p2.copy(p1);
      p2.print();
      $cast(p3 ,p1.clone()); // cloning
      p3.print();
      if(p1.compare(p2))
         $display("P1 = P2 Equal");
      else
         $display("P1 != P2 Not Equal");
   end
// experiment with the copy, clone and compare
endmodule : top
```

```
----------------------------------------------
Name            Type           Size  Value
----------------------------------------------
p1              yapp_packet    -     @2540
  payload       da(integral)   4     -
    [0]         integral       8     'h86
    [1]         integral       8     'hca
    [2]         integral       8     'he7
    [3]         integral       8     'h2
  parity        integral       8     'b0
  parity_type   parity_type_e  1     GOOD_PARITY
  packet_delay  integral       32    'h2
----------------------------------------------
----------------------------------------------
Name            Type           Size  Value
----------------------------------------------
p2              yapp_packet    -     @2552
  payload       da(integral)   20    -
    [0]         integral       8     'h71
    [1]         integral       8     'h10
    [2]         integral       8     'he9
    [3]         integral       8     'ha8
    [4]         integral       8     'hf
    ...         ...            ...   ...
    [15]        integral       8     'h36
    [16]        integral       8     'hd9
    [17]        integral       8     'h1c
    [18]        integral       8     'hb3
    [19]        integral       8     'hef
  parity        integral       8     'b0
  parity_type   parity_type_e  1     GOOD_PARITY
  packet_delay  integral       32    'h2
----------------------------------------------
----------------------------------------------
Name            Type           Size  Value
----------------------------------------------
p1              yapp_packet    -     @2635
  payload       da(integral)   4     -
    [0]         integral       8     'h86
    [1]         integral       8     'hca
    [2]         integral       8     'he7
    [3]         integral       8     'h2
  parity        integral       8     'b0
  parity_type   parity_type_e  1     GOOD_PARITY
  packet_delay  integral       32    'h2
----------------------------------------------
UVM_INFO @ 0: reporter [MISCMP] Miscompare for p1.payload:
UVM_INFO @ 0: reporter [MISCMP] 1 Miscompare(s) for object
P1 != P2 Not Equal
xmsim: *W,RNQUIE: Simulation is complete.
```

Task 2: Creating Test and Testbench Components

A UVM-based testbench for the router module was developed. A router_tb environment class was created by extending uvm_env, with the required uvm_component_utils macro, constructor, and a build_phase() method that includes a uvm_info message to indicate build-time activity.
A separate test class base_test was implemented in router_test_lib.sv by extending uvm_test. It includes the standard UVM macros, a constructor, and a build_phase() where the environment is instantiated. An informative uvm_info message was added to confirm execution, and uvm_top.print_topology() was used in end_of_elaboration_phase() to display the testbench hierarchy.
The top-level module top.sv was updated to include both router_tb.sv and router_test_lib.sv in the correct order. Previous packet randomization was removed, and run_test() was added in an initial block to start the UVM test sequence.

Router_tb Environment with base as uvm_env class

```
class router_tb extends uvm_env;
    `uvm_component_utils(router_tb)


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

    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        `uvm_info("BUILD_PHASE", "Build phase of the testbench is being executed", UVM_HIGH)
    endfunction
endclass
```

Base_test extended from uvm_test

```
class base_test extends uvm_test;
    `uvm_component_utils(base_test)

    function new (string name, uvm_component parent);
        super.new(name, parent);
    endfunction
    router_tb tb;
    virtual function void build_phase(uvm_phase phase);
        super.build_phase(phase);
        tb = new("tb
        ", this);
        `uvm_info("BUILD_PHASE", "Build Phase of Testbench is being executed", UVM_HIGH);
    endfunction

    function void end_of_elaboration_phase(uvm_phase phase);
        uvm_top.print_topology();
    endfunction

endclass
```

Test2 extended from base_test

```
class test2 extends base_test;
    `uvm_component_utils(test2);

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

endclass
```

Simulation with base_test class:

At first the simulation was run with ‘base_test’ and UVM_verbosity level HIGH, the simulation results shows that the **printed topology is the same as expected**, because the test class which is executed is ‘**base_test’** and build phase report shows that there is no error and uvm_info macro has been accessed two time during execution.

```
UVM_INFO @ 0: reporter [RNTST] Running test base_test...
UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
-----------------------------------
Name          Type       Size  Value
-----------------------------------
uvm_test_top  base_test  -     @2555
  tb          router_tb  -     @2604
-----------------------------------


--- 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 :    2
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[UVMTOP]     1
Simulation complete via $finish(1) at time 0 FS + 179
```

Simulation with test2 class:

Now the topology shows that **‘test2’** class is being executed.

```
UVM_INFO @ 0: reporter [RNTST] Running test test2...
UVM_INFO @ 0: reporter [UVMTOP] UVM testbench topology:
----------------------------------
Name          Type       Size  Value
----------------------------------
uvm_test_top  test2      -     @2555
  tb          router_tb  -     @2604
----------------------------------


--- 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 :    2
UVM_WARNING :    0
UVM_ERROR :    0
UVM_FATAL :    0
** Report counts by id
[RNTST]     1
[UVMTOP]     1
Simulation complete via $finish(1) at time 0 FS + 179
```