Digital Design Verification

Lab Manual # 39 – Layered Testbench in SystemVerilog

Name	**M. Hamza Naeem**

Release: 1.0

Date: 31-July-2024

NUST Chip Design Centre (NCDC), Islamabad, Pakistan

Contents

Objective

The objectives of this lab are

- To use a layered testbench for verifying memory features.
- To construct and integrate transaction classes, generators, drivers, monitors, and scoreboards into a verification environment.
- To implement and manage a wrapper environment that manages the verification process effectively.
- To execute the integrated test environment, analyze outputs, and validate the memory behavior against expected outcomes.
- To enhance troubleshooting skills by resolving errors and refining the verification process based on simulation feedback.

In this lab, we will be verifying the same memory module we've worked on before, but this time using a structured layered testbench approach. For memory specifications, you can look back at the earlier labs.

Tools

- SystemVerilog
- Cadence Xcelium

Instructions for Lab Tasks

The submission must follow the hierarchy below, with the folder named after the student (no spaces), and the file names exactly as listed below.

```
./student_name/
      ├── Task/
      │    ├── mem_interafce.sv
      │    ├── transaction.sv
      │    ├── generator.sv
      │    ├── driver.sv
      │    ├── monitor.sv
      │    ├── scoreboard.sv
      │    ├── env.sv
      │    ├── test.sv
      │    ├── testbench.sv
```

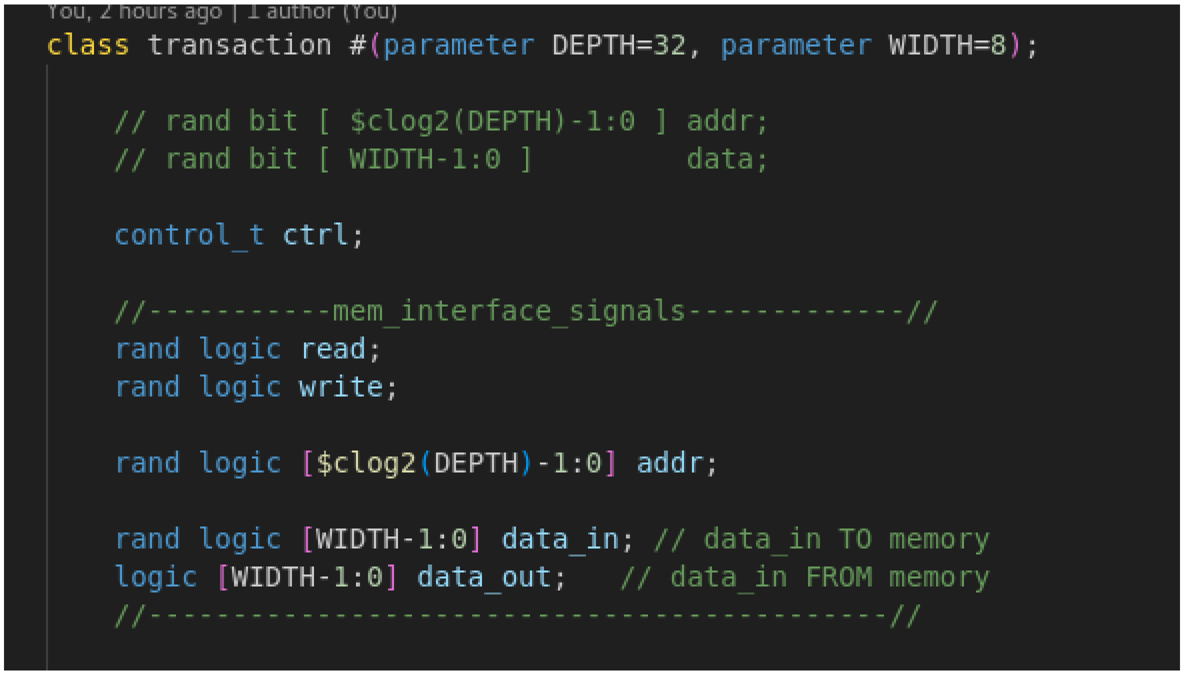
Lab Task 1: Building the Transaction Class

To create a transaction class encapsulating memory interface signals and methods for display and randomization.

Modifying the Transaction class

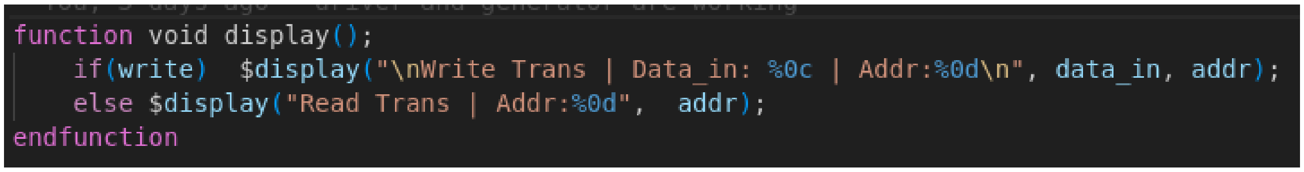
Working in the Task1 directory, modify the transaction.sv file as follows:

1. Add memory interface signals as class properties.
2. Create a class constructor to initialize class properties.



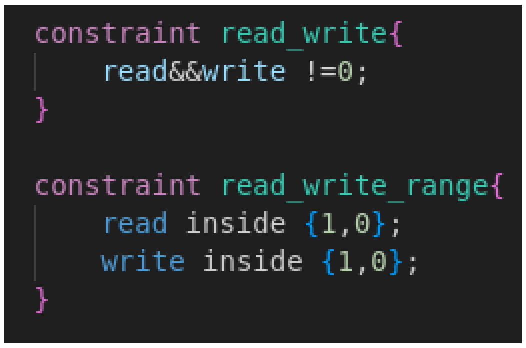
```
You, 2 hours ago | 1 author (You)
class transaction #(parameter DEPTH=32, parameter WIDTH=8);

    // rand bit [ $clog2(DEPTH)-1:0 ] addr;
    // rand bit [ WIDTH-1:0 ]         data;

    control_t ctrl;

    //-----------mem_interface_signals------------//
    rand logic read;
    rand logic write;

    rand logic [$clog2(DEPTH)-1:0] addr;

    rand logic [WIDTH-1:0] data_in; // data_in TO memory
    logic [WIDTH-1:0] data_out;    // data_in FROM memory
    //--------------------------------------------//
```

3. Implement a display method to print out all properties of a transaction class.



```
    You, 3 days ago • driver and generator are working
function void display();
    if(write)  $display("\nWrite Trans | Data_in: %0c | Addr:%0d\n", data_in, addr);
    else $display("Read Trans | Addr:%0d",  addr);
endfunction
```

4. Add constraints to control whether the operation is a write or read, ensuring that each transaction is correctly classified and executed as intended.



```
constraint read_write{
    read&&write !=0;
}

constraint read_write_range{
    read inside {1,0};
    write inside {1,0};
}
```

Lab Task 2: Creating the Generator and Driver classes

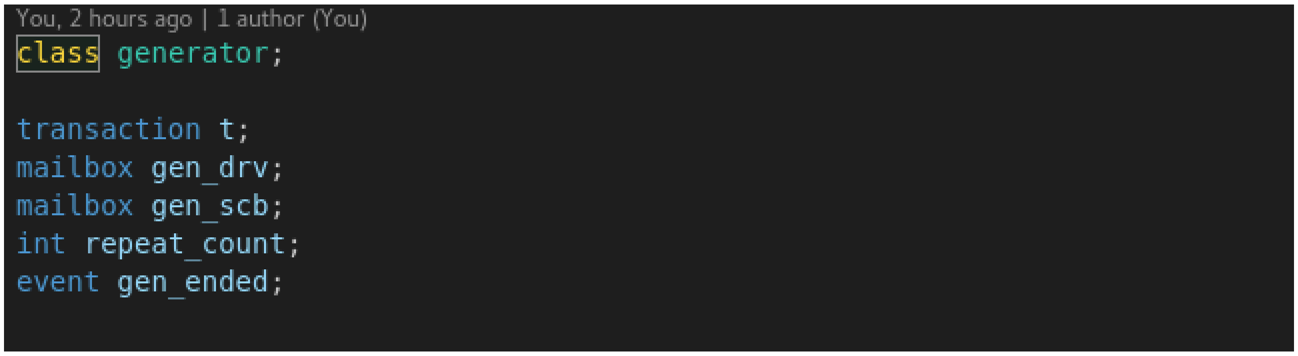
Develop a generator to create different transactions and a driver to drive those transactions to the DUT.

Creating Generator Class

Create the generator class in the file generator.sv as follows:

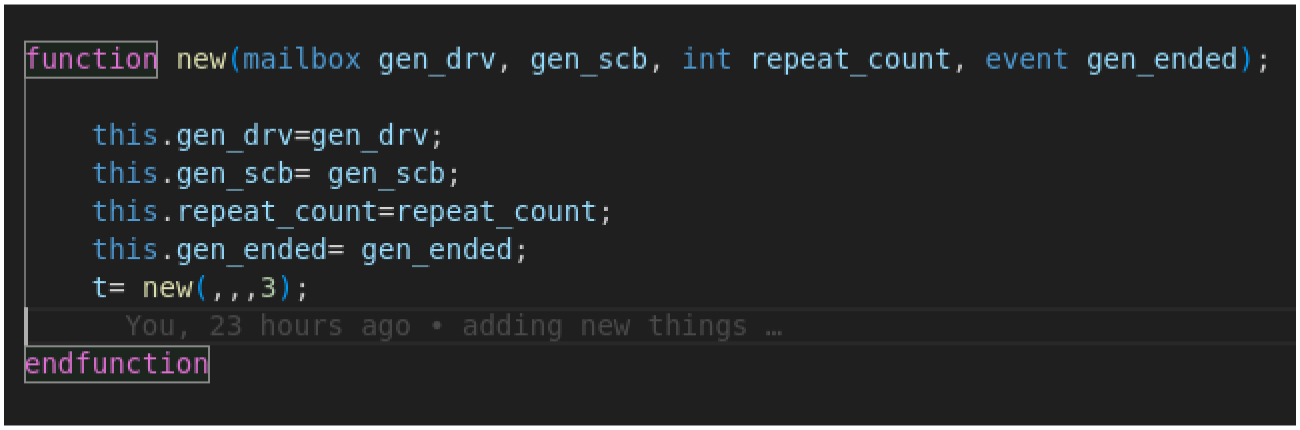
1. Add class properties to the generator class. The properties should include a transaction object,

two mailboxes (one for sending transactions to the driver and one for the scoreboard), an int repeat count to specify the number of transactions to be generated by driver, and an event to signal the completion of transaction generation.



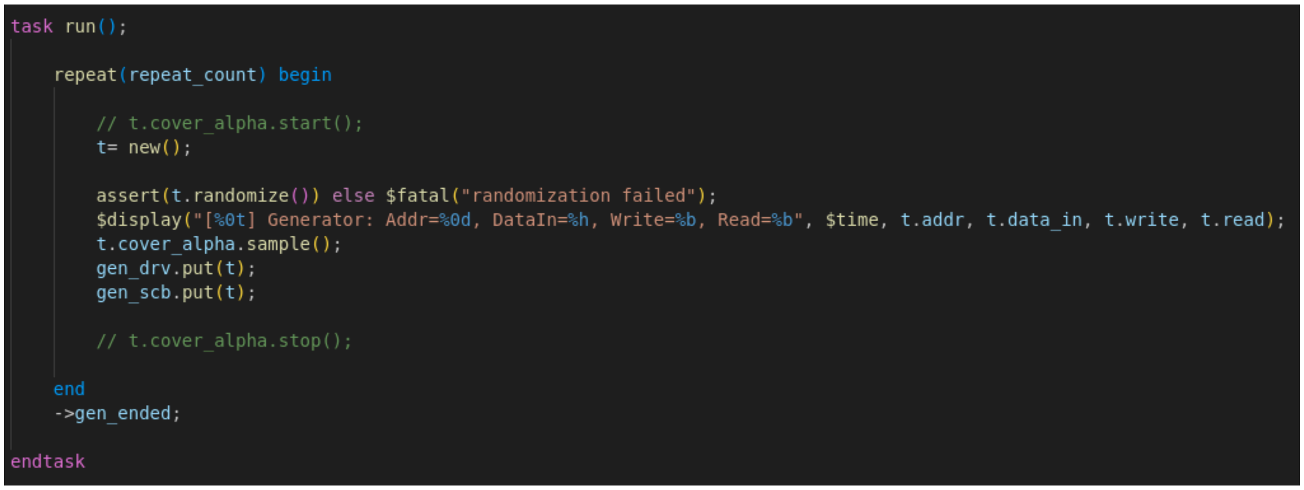
```
You, 2 hours ago | 1 author (You)
class generator;

transaction t;
mailbox gen_drv;
mailbox gen_scb;
int repeat_count;
event gen_ended;
```

2. Write the constructor for the generator class. It should initialize the mailboxes and set the repeat count based on an input argument. Also, instantiate the transaction object.



```
function new(mailbox gen_drv, gen_scb, int repeat_count, event gen_ended);

    this.gen_drv=gen_drv;
    this.gen_scb= gen_scb;
    this.repeat_count=repeat_count;
    this.gen_ended= gen_ended;
    t= new(,,,3);
      You, 23 hours ago • adding new things …
endfunction
```

3. Implement the run task as class method. In this task, generate random transactions up to the specified repeat count using randomize() method. You can control the type of transaction (write or read) using constraints. After randomizing, send different copies of transaction to both the driver and the scoreboard through mailboxes. Once required number of transactions are generated, trigger the completion event.



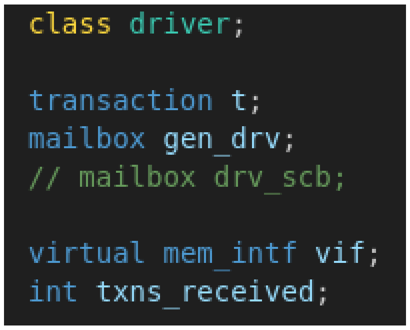
```
task run();

    repeat(repeat_count) begin

        // t.cover_alpha.start();
        t= new();

        assert(t.randomize()) else $fatal("randomization failed");
        $display("[%0t] Generator: Addr=%0d, DataIn=%h, Write=%b, Read=%b", $time, t.addr, t.data_in, t.write, t.read);
        t.cover_alpha.sample();
        gen_drv.put(t);
        gen_scb.put(t);

        // t.cover_alpha.stop();

    end
    ->gen_ended;

endtask
```

Remember to include error handling for the randomize method if the transaction fails to randomize due to conflicting constraints.

Creating Driver Class

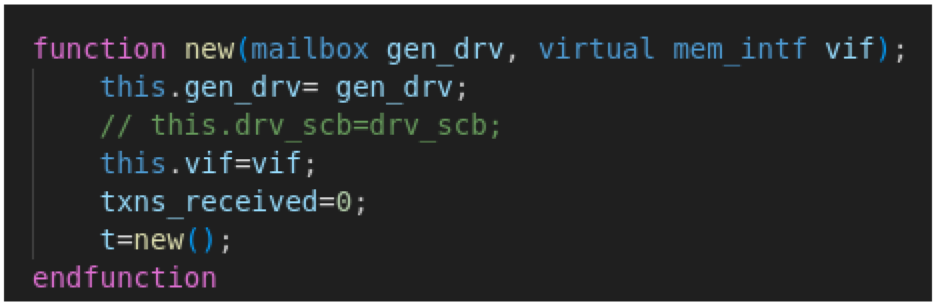
Create the driver class in the file driver.sv as follows:

4. Define class properties including a transaction object, a mailbox for incoming transactions, a virtual interface to the DUT, and an integer for tracking the number of transactions received from generator.



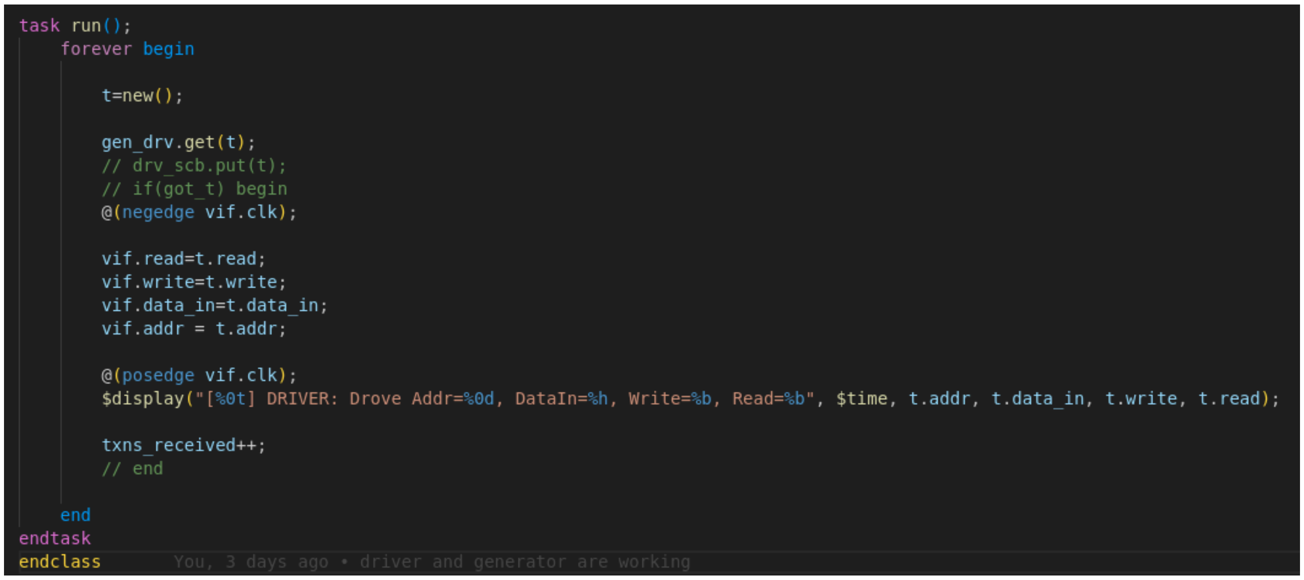
```
class driver;

transaction t;
mailbox gen_drv;
// mailbox drv_scb;

virtual mem_intf vif;
int txns_received;
```

5. Create the driver class constructor to initialize the virtual interface and transaction mailbox.



```
function new(mailbox gen_drv, virtual mem_intf vif);
    this.gen_drv= gen_drv;
    // this.drv_scb=drv_scb;
    this.vif=vif;
    txns_received=0;
    t=new();
endfunction
```

6. Implement a run task that uses a forever loop to continuously receive transactions from the mailbox and drive them to the DUT via the virtual interface, ensuring that transactions are driven on the inactive edge of the clock. At the end of each iteration, increment the int that stores the number of transactions received.



```
task run();
    forever begin

        t=new();

        gen_drv.get(t);
        // drv_scb.put(t);
        // if(got_t) begin
        @(negedge vif.clk);

        vif.read=t.read;
        vif.write=t.write;
        vif.data_in=t.data_in;
        vif.addr = t.addr;

        @(posedge vif.clk);
        $display("[%0t] DRIVER: Drove Addr=%0d, DataIn=%h, Write=%b, Read=%b", $time, t.addr, t.data_in, t.write, t.read);

        txns_received++;
        // end

    end
endtask
endclass        You, 3 days ago • driver and generator are working
```

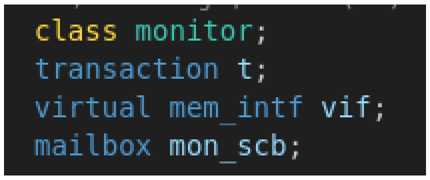
Lab Task 3: Creating the Monitor and Scoreboard Classes

Develop a monitor to observe the DUT responses and a scoreboard to compare those responses against expected results.

Creating Monitor Class

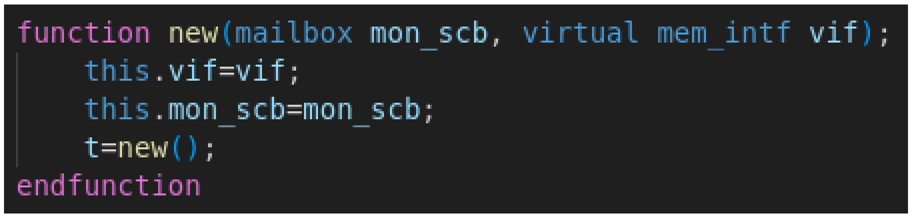
Create the monitor class in the file monitor.sv as follows:

1. Define class properties for the monitor class. These should include a virtual interface to the DUT and a mailbox to send monitored transactions to the scoreboard.



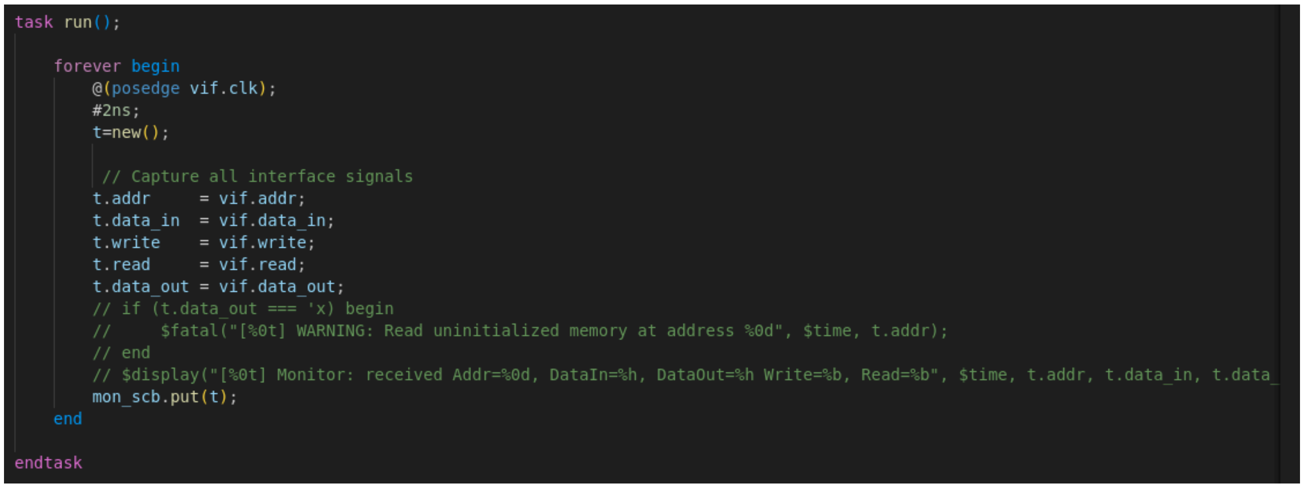
```
class monitor;
transaction t;
virtual mem_intf vif;
mailbox mon_scb;
```

2. Write the constructor for the monitor class to initialize the virtual interface.



```
function new(mailbox mon_scb, virtual mem_intf vif);
    this.vif=vif;
    this.mon_scb=mon_scb;
    t=new();
endfunction
```

3. Implement the run method. This method should continuously monitor the DUT's output using the virtual interface. For each observed response, package the data into a transaction object and send it to the scoreboard mailbox for verification.



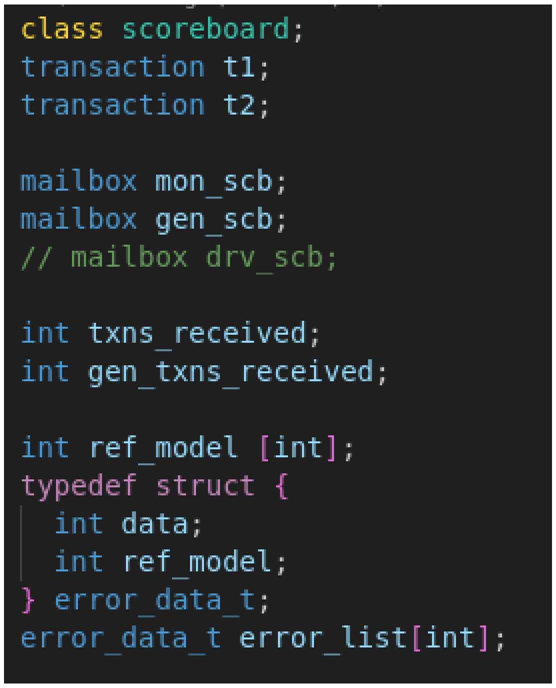
```
task run();

    forever begin
        @(posedge vif.clk);
        #2ns;
        t=new();

         // Capture all interface signals
        t.addr     = vif.addr;
        t.data_in  = vif.data_in;
        t.write    = vif.write;
        t.read     = vif.read;
        t.data_out = vif.data_out;
        // if (t.data_out === 'x) begin
        //     $fatal("[%0t] WARNING: Read uninitialized memory at address %0d", $time, t.addr);
        // end
        // $display("[%0t] Monitor: received Addr=%0d, DataIn=%h, DataOut=%h Write=%b, Read=%b", $time, t.addr, t.data_in, t.data_
        mon_scb.put(t);
    end

endtask
```

Ensure the monitor captures the DUT responses at the correct times for accurate results, avoiding transient states that could lead to verification errors.

Creating Scoreboard Class

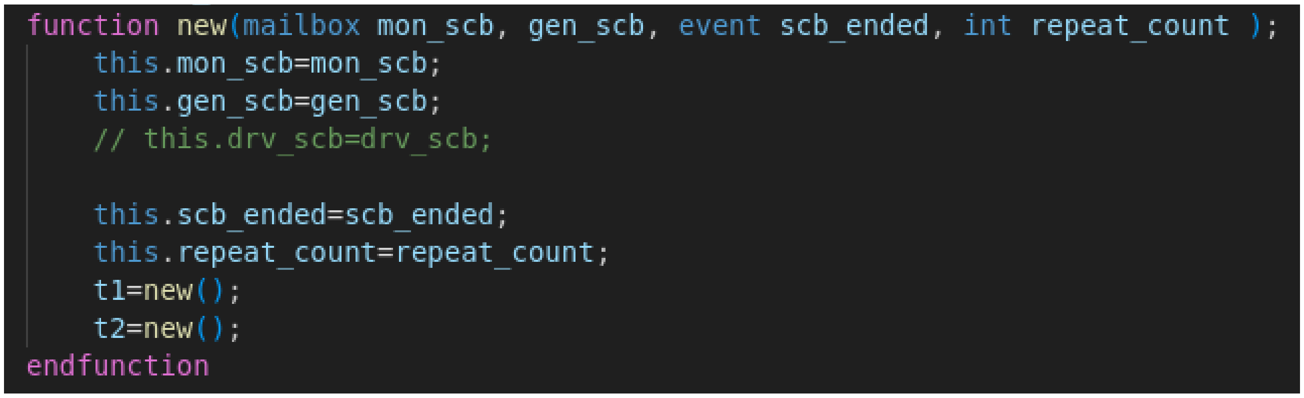
Create the scoreboard class in the file scoreboard.sv as follows:

4. Define class properties for the scoreboard class. The properties should include two mailboxes: one to receive transactions from the generator and one to receive transactions from the monitor. Also add an int for tracking number of transactions received, associative array to model the memory behavior and an int to count the number of errors.



```
class scoreboard;
transaction t1;
transaction t2;

mailbox mon_scb;
mailbox gen_scb;
// mailbox drv_scb;

int txns_received;
int gen_txns_received;

int ref_model [int];
typedef struct {
  int data;
  int ref_model;
} error_data_t;
error_data_t error_list[int];
```

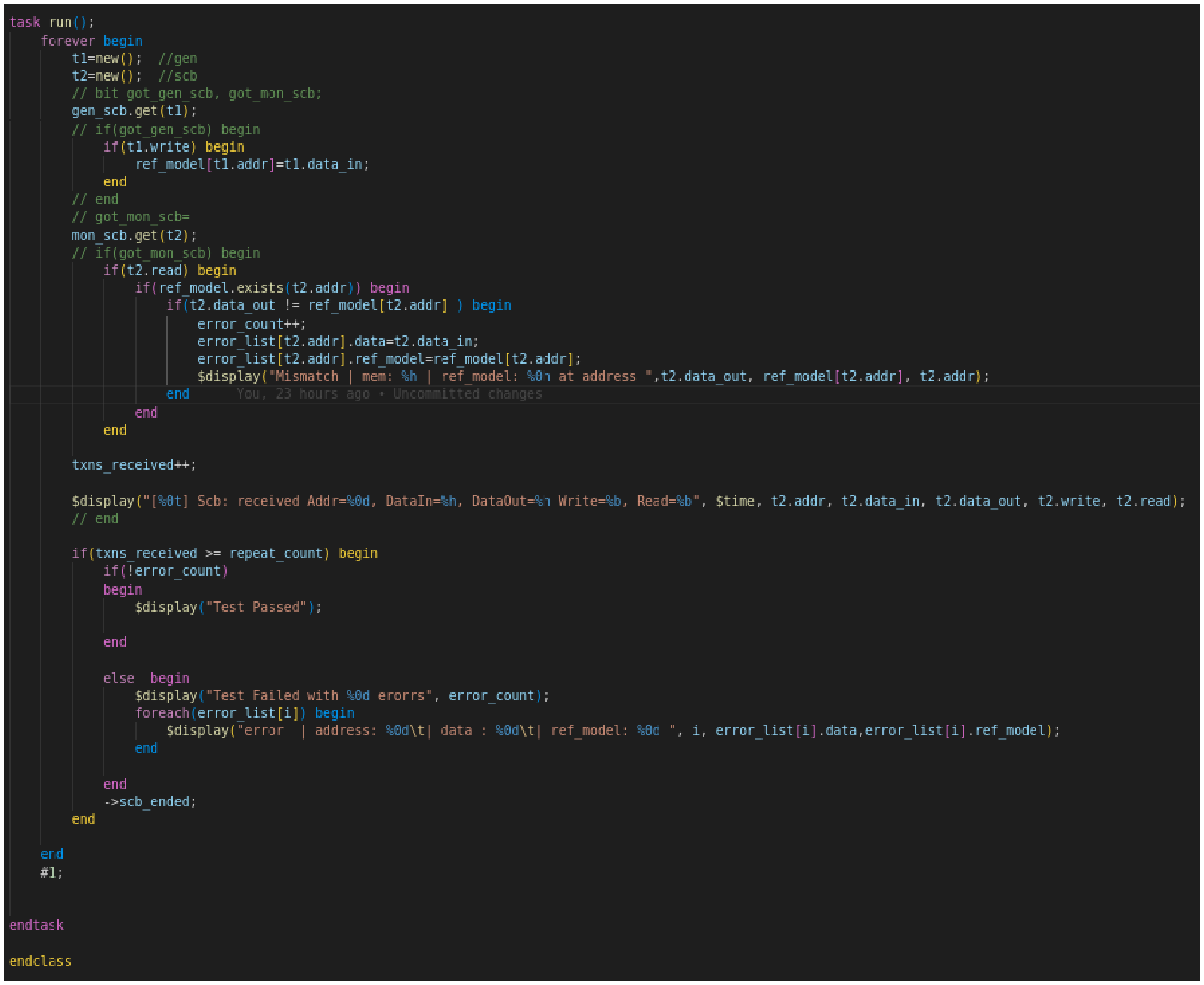
5. Write the constructor for the scoreboard class that initializes both mailboxes.



```
function new(mailbox mon_scb, gen_scb, event scb_ended, int repeat_count );
    this.mon_scb=mon_scb;
    this.gen_scb=gen_scb;
    // this.drv_scb=drv_scb;

    this.scb_ended=scb_ended;
    this.repeat_count=repeat_count;
    t1=new();
    t2=new();
endfunction
```

6. Implement a run task that uses a forever loop to continuously receive transactions from the generator and monitor through corresponding mailboxes. If write transaction is received from generator then write corresponding data to the associative array.

If read transaction is received from the monitor the compare the read data with the data stored in associative array at
corresponding address, increment error count if the data doesn’t match. At the end of each



```
task run();
    forever begin
        t1=new();  //gen
        t2=new();  //scb
        // bit got_gen_scb, got_mon_scb;
        gen_scb.get(t1);
        // if(got_gen_scb) begin
            if(t1.write) begin
                ref_model[t1.addr]=t1.data_in;
            end
        // end
        // got_mon_scb=
        mon_scb.get(t2);
        // if(got_mon_scb) begin
            if(t2.read) begin
                if(ref_model.exists(t2.addr)) begin
                    if(t2.data_out != ref_model[t2.addr] ) begin
                        error_count++;
                        error_list[t2.addr].data=t2.data_in;
                        error_list[t2.addr].ref_model=ref_model[t2.addr];
                        $display("Mismatch | mem: %h | ref_model: %0h at address ",t2.data_out, ref_model[t2.addr], t2.addr);
                    end      You, 23 hours ago • Uncommitted changes
                end
            end

        txns_received++;

        $display("[%0t] Scb: received Addr=%0d, DataIn=%h, DataOut=%h Write=%b, Read=%b", $time, t2.addr, t2.data_in, t2.data_out, t2.write, t2.read);
        // end

        if(txns_received >= repeat_count) begin
            if(!error_count)
            begin
                $display("Test Passed");

            end

            else  begin
                $display("Test Failed with %0d erorrs", error_count);
                foreach(error_list[i]) begin
                    $display("error  | address: %0d\t| data : %0d\t| ref_model: %0d ", i, error_list[i].data,error_list[i].ref_model);
                end

            end
            ->scb_ended;
        end

    end
    #1;


endtask

endclass
```

iteration, increment the int that stores the number of transactions received.

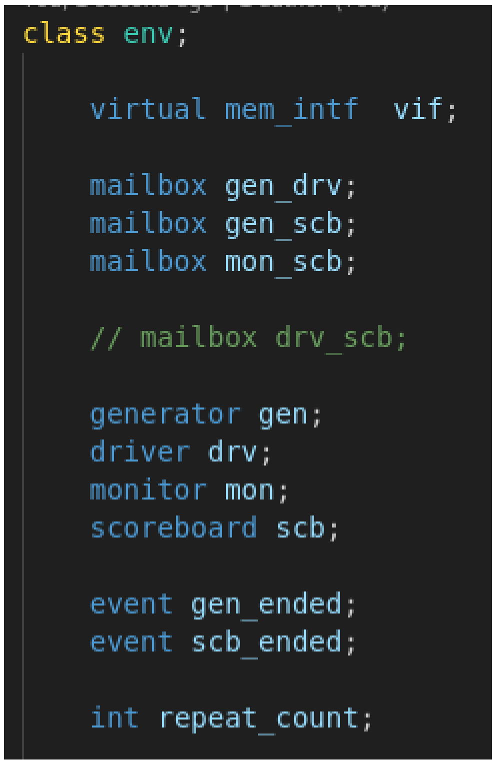
Lab Task 4: Building the Environment and Testbench Program

Develop a wrapper environment class that integrates the generator, driver, monitor, and scoreboard classes, and construct a testbench program to initiate and control the verification process.

Creating Environment Class

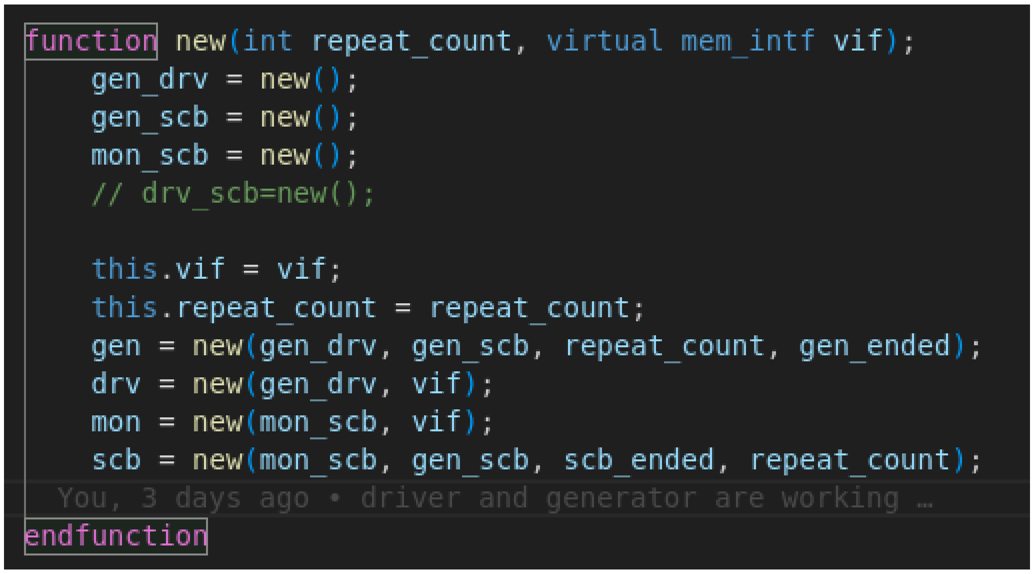
Create the environment class in the file `env.sv` as follows:

1. Add class properties including a virtual memory interface (virtual mem_intf), three mailbox handles for generator to driver (gen2driv), generator to scoreboard(gen2scb) and monitor to scoreboard (mon2scb), instances of generator, driver, monitor, and scoreboard, and an event (gen_ended) to signal the end of transaction generation.



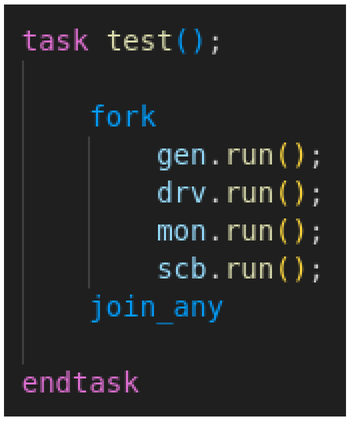
```
class env;

    virtual mem_intf  vif;

    mailbox gen_drv;
    mailbox gen_scb;
    mailbox mon_scb;

    // mailbox drv_scb;

    generator gen;
    driver drv;
    monitor mon;
    scoreboard scb;

    event gen_ended;
    event scb_ended;

    int repeat_count;
```

2. Write the constructor, The constructor should take two inputs virtual interface and repeat count. In this constructor first create instances of mailboxes and then construct object of generator, driver, monitor and scoreboard using the virtual interface, repeat count, end of generator event and the mailboxes.



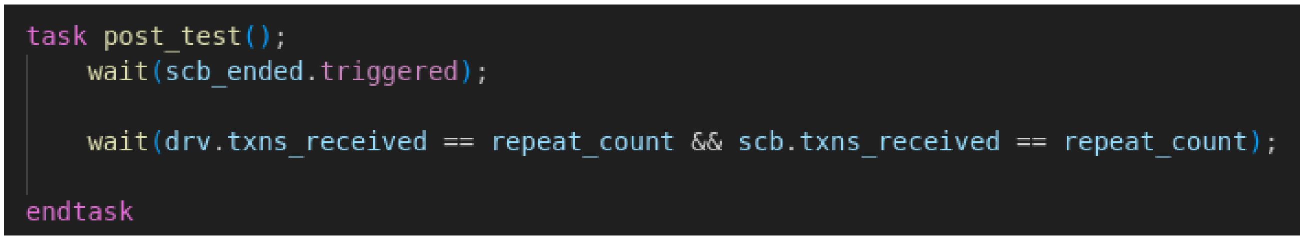
```
function new(int repeat_count, virtual mem_intf vif);
    gen_drv = new();
    gen_scb = new();
    mon_scb = new();
    // drv_scb=new();

    this.vif = vif;
    this.repeat_count = repeat_count;
    gen = new(gen_drv, gen_scb, repeat_count, gen_ended);
    drv = new(gen_drv, vif);
    mon = new(mon_scb, vif);
    scb = new(mon_scb, gen_scb, scb_ended, repeat_count);
      You, 3 days ago • driver and generator are working …
endfunction
```

3. Create the test task to start the generator, driver, monitor, and scoreboard using a fork...join_any construct, allowing them to execute in parallel.



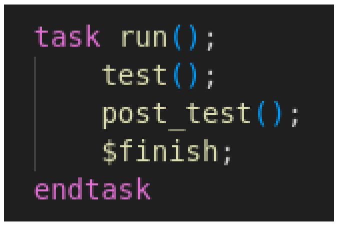
```
task test();

    fork
        gen.run();
        drv.run();
        mon.run();
        scb.run();
    join_any

endtask
```

4. Create the post_test task to first wait until generator has finished producing all transactions, and that both the driver and the scoreboard have received the correct number of transactions equal to the repeat count. At the end, use error count of scoreboard object to display the pass/failed status of test.



```
task post_test();
    wait(scb_ended.triggered);

    wait(drv.txns_received == repeat_count && scb.txns_received == repeat_count);

endtask
```

5. Implement the run method. This method should sequentially executes test(), post_test() and then finish the simulation using $finish.

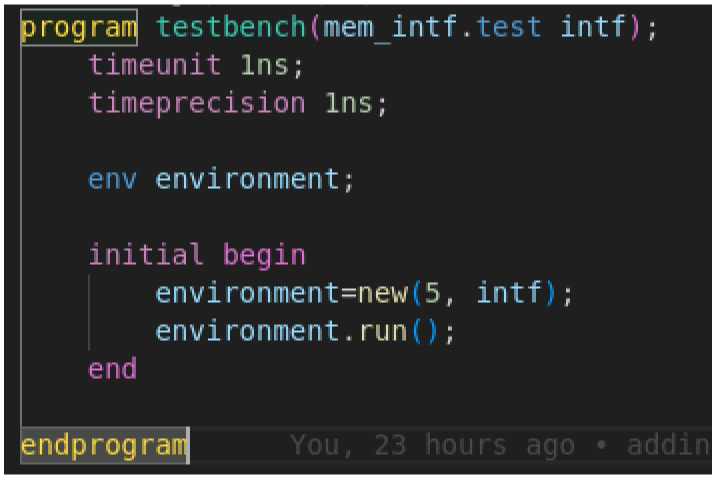


```
task run();
    test();
    post_test();
    $finish;
endtask
```

Creating the Testbench Program

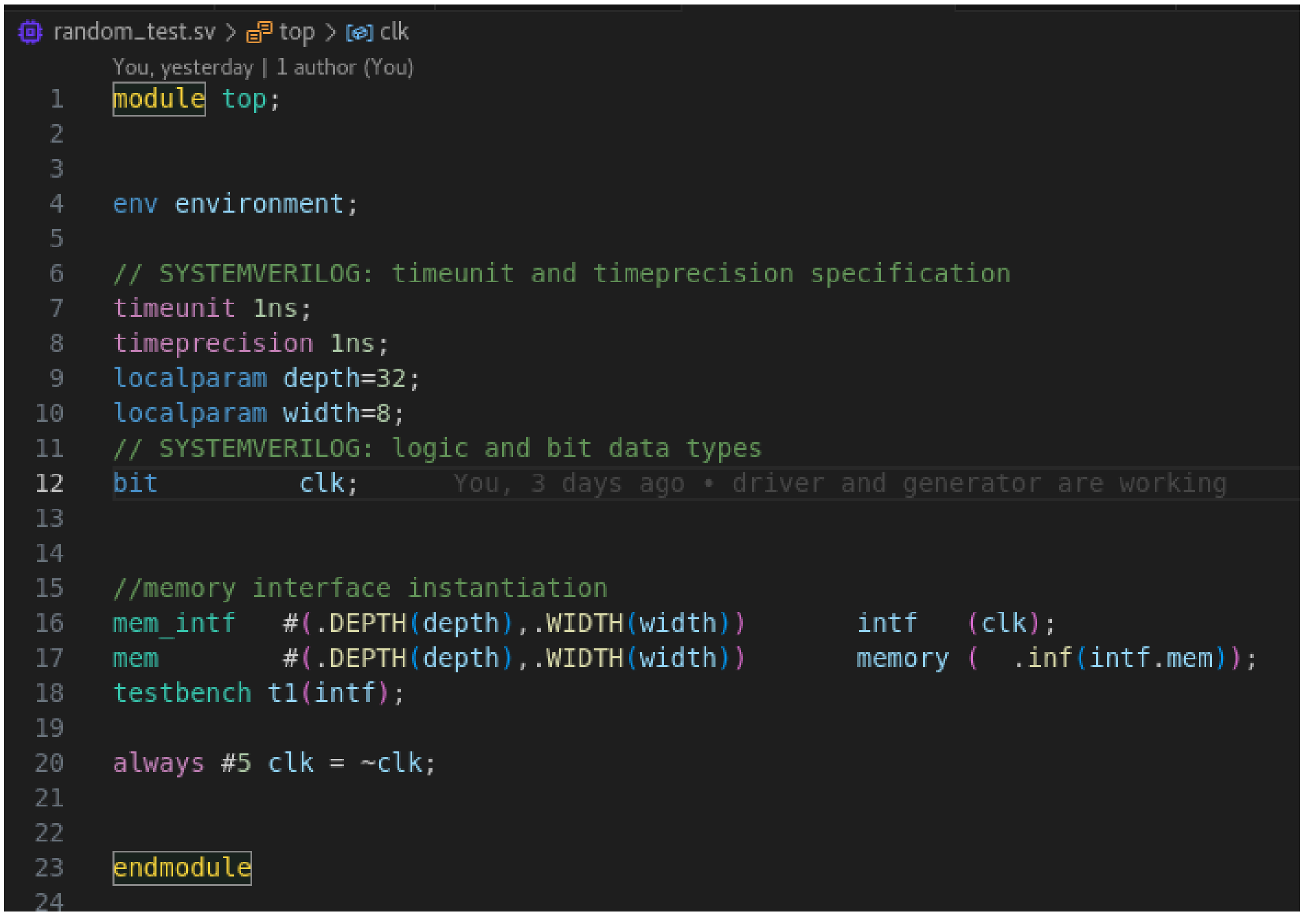
Compose the testbench program in the file `random_test.sv` as follows:

6. Declare an environment class instance within the test program. Instantiate the environment class, passing it the virtual interface and the repeat count.



```
program testbench(mem_intf.test intf);
    timeunit 1ns;
    timeprecision 1ns;

    env environment;

    initial begin
        environment=new(5, intf);
        environment.run();
    end

endprogram        You, 23 hours ago • addin
```

7. Within an initial block, simply call the run task of the environment instance.



random_test.sv > top > clk

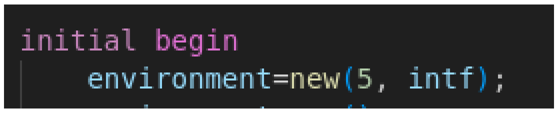
You, yesterday | 1 author (You)

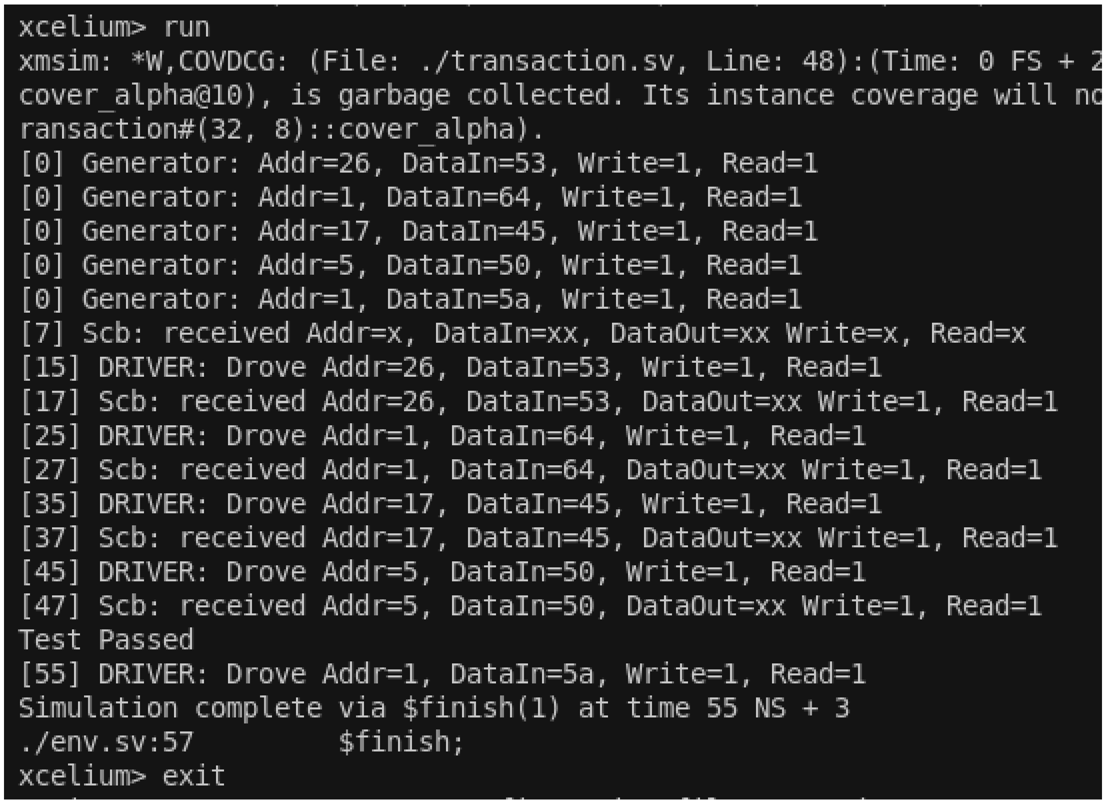
```
module top;


env environment;

// SYSTEMVERILOG: timeunit and timeprecision specification
timeunit 1ns;
timeprecision 1ns;
localparam depth=32;
localparam width=8;
// SYSTEMVERILOG: logic and bit data types
bit         clk;      You, 3 days ago • driver and generator are working


//memory interface instantiation
mem_intf    #(.DEPTH(depth),.WIDTH(width))       intf    (clk);
mem         #(.DEPTH(depth),.WIDTH(width))       memory (   .inf(intf.mem));
testbench t1(intf);

always #5 clk = ~clk;


endmodule
```

Running the Simulation

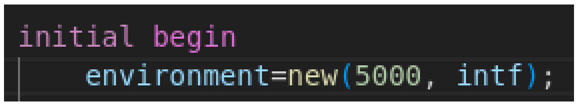
8. Run the simulation. Monitor the process through waveforms and console outputs to observe and verify the DUT's responses.

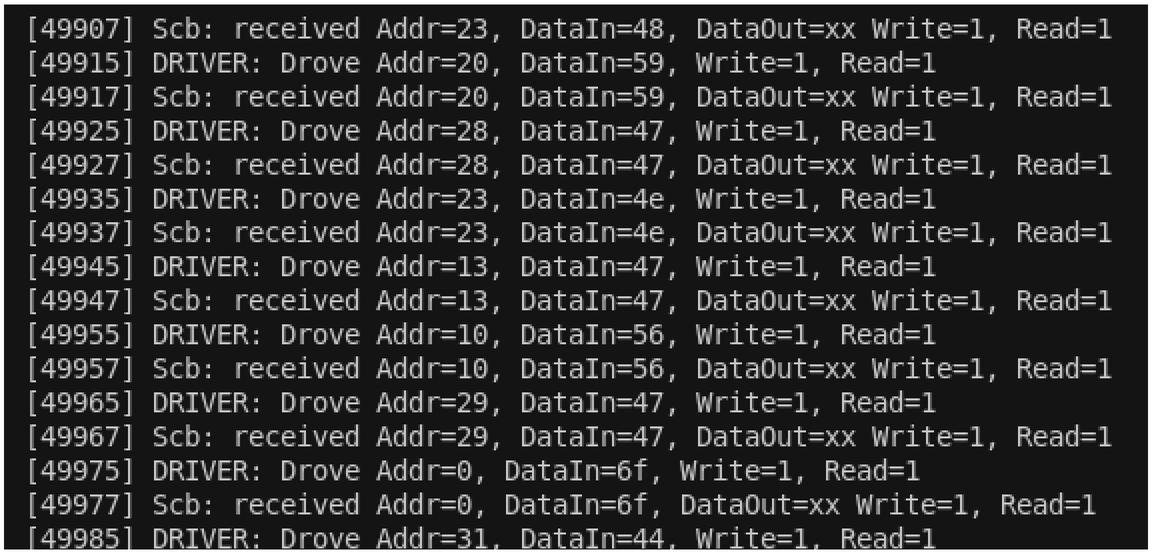
For 5 transactions:

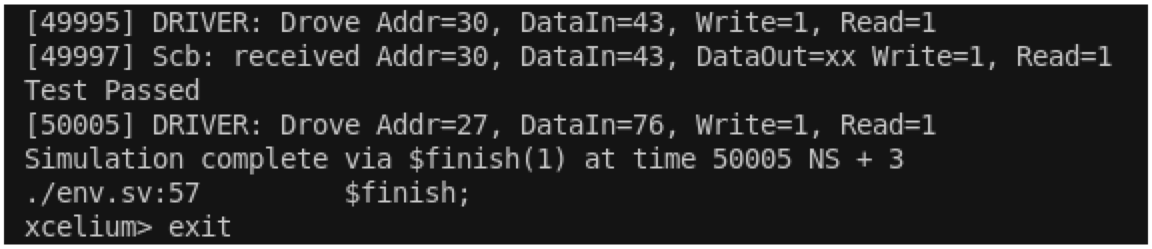

```
initial begin
    environment=new(5, intf);
```


```
xcelium> run
xmsim: *W,COVDCG: (File: ./transaction.sv, Line: 48):(Time: 0 FS + 2
cover_alpha@10), is garbage collected. Its instance coverage will no
ransaction#(32, 8)::cover_alpha).
[0] Generator: Addr=26, DataIn=53, Write=1, Read=1
[0] Generator: Addr=1, DataIn=64, Write=1, Read=1
[0] Generator: Addr=17, DataIn=45, Write=1, Read=1
[0] Generator: Addr=5, DataIn=50, Write=1, Read=1
[0] Generator: Addr=1, DataIn=5a, Write=1, Read=1
[7] Scb: received Addr=x, DataIn=xx, DataOut=xx Write=x, Read=x
[15] DRIVER: Drove Addr=26, DataIn=53, Write=1, Read=1
[17] Scb: received Addr=26, DataIn=53, DataOut=xx Write=1, Read=1
[25] DRIVER: Drove Addr=1, DataIn=64, Write=1, Read=1
[27] Scb: received Addr=1, DataIn=64, DataOut=xx Write=1, Read=1
[35] DRIVER: Drove Addr=17, DataIn=45, Write=1, Read=1
[37] Scb: received Addr=17, DataIn=45, DataOut=xx Write=1, Read=1
[45] DRIVER: Drove Addr=5, DataIn=50, Write=1, Read=1
[47] Scb: received Addr=5, DataIn=50, DataOut=xx Write=1, Read=1
Test Passed
[55] DRIVER: Drove Addr=1, DataIn=5a, Write=1, Read=1
Simulation complete via $finish(1) at time 55 NS + 3
./env.sv:57          $finish;
xcelium> exit
```

For 5000 transactions:


```
initial begin
    environment=new(5000, intf);
```

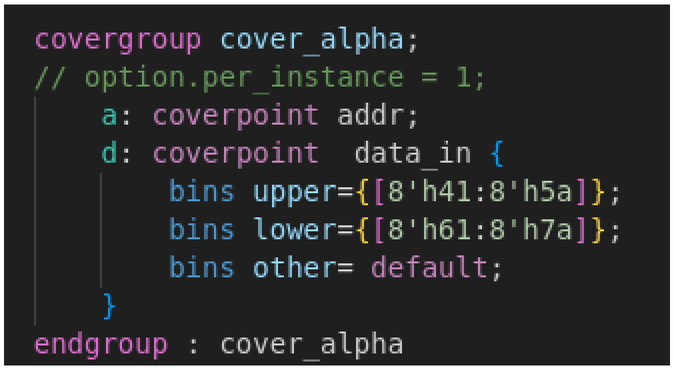

```
[49907] Scb: received Addr=23, DataIn=48, DataOut=xx Write=1, Read=1
[49915] DRIVER: Drove Addr=20, DataIn=59, Write=1, Read=1
[49917] Scb: received Addr=20, DataIn=59, DataOut=xx Write=1, Read=1
[49925] DRIVER: Drove Addr=28, DataIn=47, Write=1, Read=1
[49927] Scb: received Addr=28, DataIn=47, DataOut=xx Write=1, Read=1
[49935] DRIVER: Drove Addr=23, DataIn=4e, Write=1, Read=1
[49937] Scb: received Addr=23, DataIn=4e, DataOut=xx Write=1, Read=1
[49945] DRIVER: Drove Addr=13, DataIn=47, Write=1, Read=1
[49947] Scb: received Addr=13, DataIn=47, DataOut=xx Write=1, Read=1
[49955] DRIVER: Drove Addr=10, DataIn=56, Write=1, Read=1
[49957] Scb: received Addr=10, DataIn=56, DataOut=xx Write=1, Read=1
[49965] DRIVER: Drove Addr=29, DataIn=47, Write=1, Read=1
[49967] Scb: received Addr=29, DataIn=47, DataOut=xx Write=1, Read=1
[49975] DRIVER: Drove Addr=0, DataIn=6f, Write=1, Read=1
[49977] Scb: received Addr=0, DataIn=6f, DataOut=xx Write=1, Read=1
[49985] DRIVER: Drove Addr=31, DataIn=44, Write=1, Read=1
```



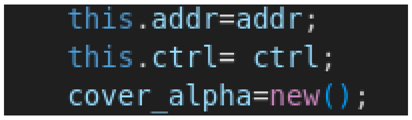
```
[49995] DRIVER: Drove Addr=30, DataIn=43, Write=1, Read=1
[49997] Scb: received Addr=30, DataIn=43, DataOut=xx Write=1, Read=1
Test Passed
[50005] DRIVER: Drove Addr=27, DataIn=76, Write=1, Read=1
Simulation complete via $finish(1) at time 50005 NS + 3
./env.sv:57         $finish;
xcelium> exit
```

9. If errors occur, troubleshoot by examining the waveforms and log files. Adjust the testbench components if necessary to resolve any issues.

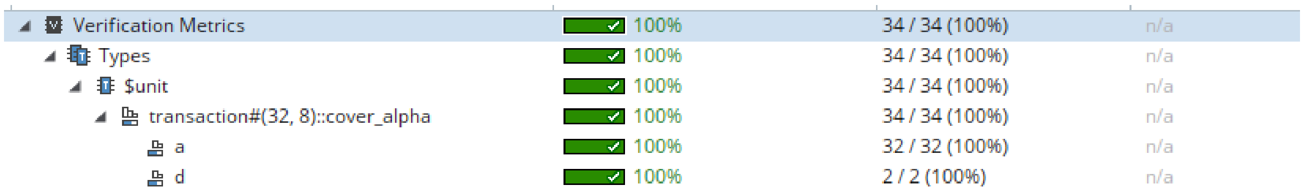
   Coverage:



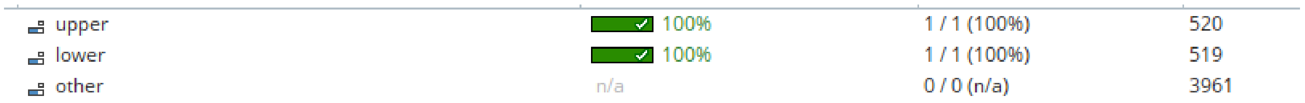
```
covergroup cover_alpha;
// option.per_instance = 1;
    a: coverpoint addr;
    d: coverpoint  data_in {
        bins upper={[8'h41:8'h5a]};
        bins lower={[8'h61:8'h7a]};
        bins other= default;
    }
endgroup : cover_alpha
```



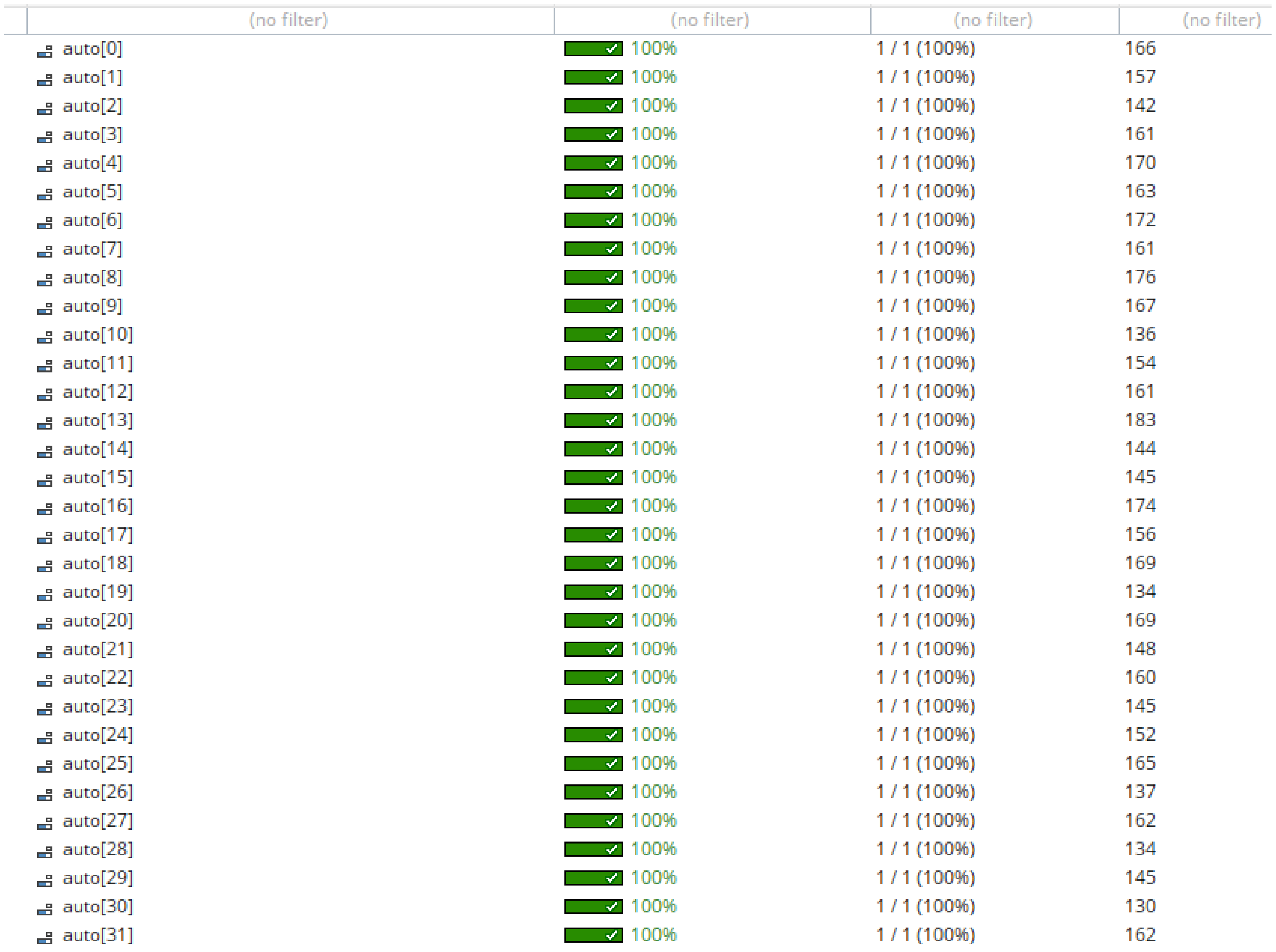
```
    this.addr=addr;
    this.ctrl= ctrl;
    cover_alpha=new();
```



Verification Metrics	✓ 100%	34 / 34 (100%)	n/a
Types	✓ 100%	34 / 34 (100%)	n/a
$unit	✓ 100%	34 / 34 (100%)	n/a
transaction#(32, 8)::cover_alpha	✓ 100%	34 / 34 (100%)	n/a
a	✓ 100%	32 / 32 (100%)	n/a
d	✓ 100%	2 / 2 (100%)	n/a

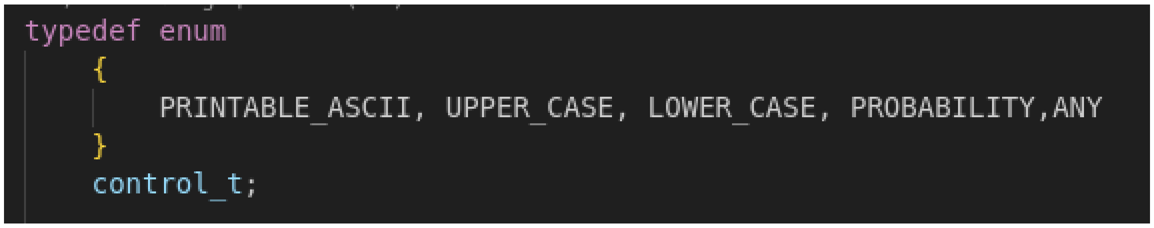


upper	✓ 100%	1 / 1 (100%)	520
lower	✓ 100%	1 / 1 (100%)	519
other	n/a	0 / 0 (n/a)	3961

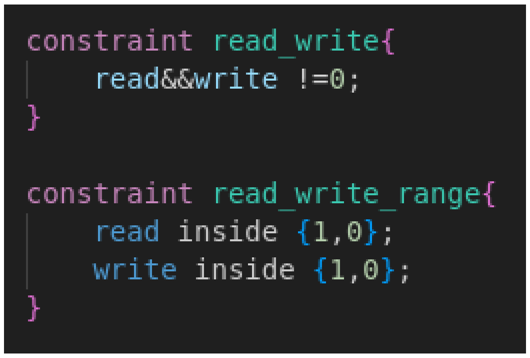


(no filter)	(no filter)	(no filter)	(no filter)
auto[0]	✓ 100%	1 / 1 (100%)	166
auto[1]	✓ 100%	1 / 1 (100%)	157
auto[2]	✓ 100%	1 / 1 (100%)	142
auto[3]	✓ 100%	1 / 1 (100%)	161
auto[4]	✓ 100%	1 / 1 (100%)	170
auto[5]	✓ 100%	1 / 1 (100%)	163
auto[6]	✓ 100%	1 / 1 (100%)	172
auto[7]	✓ 100%	1 / 1 (100%)	161
auto[8]	✓ 100%	1 / 1 (100%)	176
auto[9]	✓ 100%	1 / 1 (100%)	167
auto[10]	✓ 100%	1 / 1 (100%)	136
auto[11]	✓ 100%	1 / 1 (100%)	154
auto[12]	✓ 100%	1 / 1 (100%)	161
auto[13]	✓ 100%	1 / 1 (100%)	183
auto[14]	✓ 100%	1 / 1 (100%)	144
auto[15]	✓ 100%	1 / 1 (100%)	145
auto[16]	✓ 100%	1 / 1 (100%)	174
auto[17]	✓ 100%	1 / 1 (100%)	156
auto[18]	✓ 100%	1 / 1 (100%)	169
auto[19]	✓ 100%	1 / 1 (100%)	134
auto[20]	✓ 100%	1 / 1 (100%)	169
auto[21]	✓ 100%	1 / 1 (100%)	148
auto[22]	✓ 100%	1 / 1 (100%)	160
auto[23]	✓ 100%	1 / 1 (100%)	145
auto[24]	✓ 100%	1 / 1 (100%)	152
auto[25]	✓ 100%	1 / 1 (100%)	165
auto[26]	✓ 100%	1 / 1 (100%)	137
auto[27]	✓ 100%	1 / 1 (100%)	162
auto[28]	✓ 100%	1 / 1 (100%)	134
auto[29]	✓ 100%	1 / 1 (100%)	145
auto[30]	✓ 100%	1 / 1 (100%)	130
auto[31]	✓ 100%	1 / 1 (100%)	162

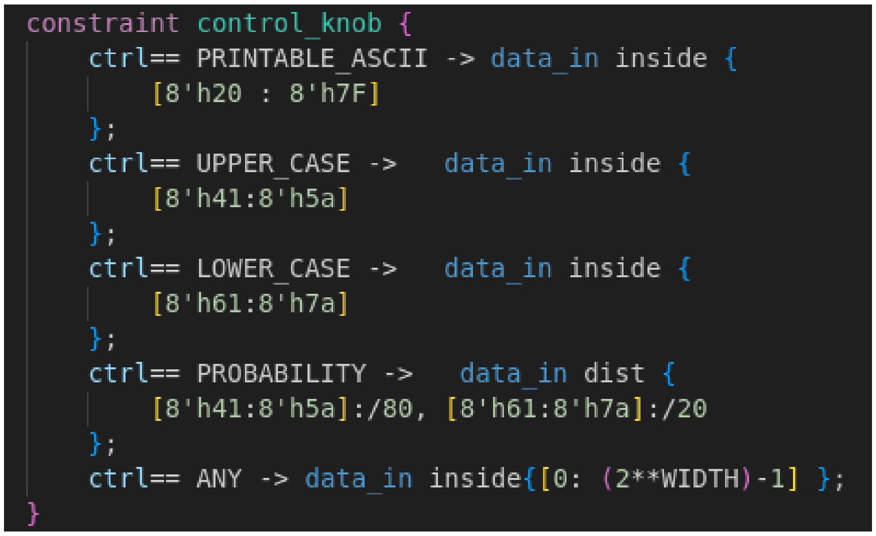
Constraints:



```
typedef enum
    {
        PRINTABLE_ASCII, UPPER_CASE, LOWER_CASE, PROBABILITY,ANY
    }
    control_t;
```



```
constraint read_write{
    read&&write !=0;
}

constraint read_write_range{
    read inside {1,0};
    write inside {1,0};
}
```



```
constraint control_knob {
    ctrl== PRINTABLE_ASCII -> data_in inside {
        [8'h20 : 8'h7F]
    };
    ctrl== UPPER_CASE ->   data_in inside {
        [8'h41:8'h5a]
    };
    ctrl== LOWER_CASE ->   data_in inside {
        [8'h61:8'h7a]
    };
    ctrl== PROBABILITY ->   data_in dist {
        [8'h41:8'h5a]:/80, [8'h61:8'h7a]:/20
    };
    ctrl== ANY -> data_in inside{[0: (2**WIDTH)-1] };
}
```

Problems and Bugs faced:

- Due to the incorrect connections of the mailboxes, simulation was stalling.
- Error occurred because of the incorrect constraints but I resolved by correcting the constraints.
- Also faced problems while collecting coverage as it was taking coverage for each instance of transactions even those that were only for the copying of transactions. I resolved that by turning off the per instance option in the transaction coverage.

Conclusion:

In this lab we learned about how transactions and flow of them works in the layered testbench as it would be helpful in the UVM testbench. We also applied the concepts learned in the coverage and constraints.

While in the lab we were instructed to pass the transactions from the generator to the scoreboard but a better practice is to transfer from the driver as it only passes the transactions that are drived successfully. I also did this in my code and the results were successful.