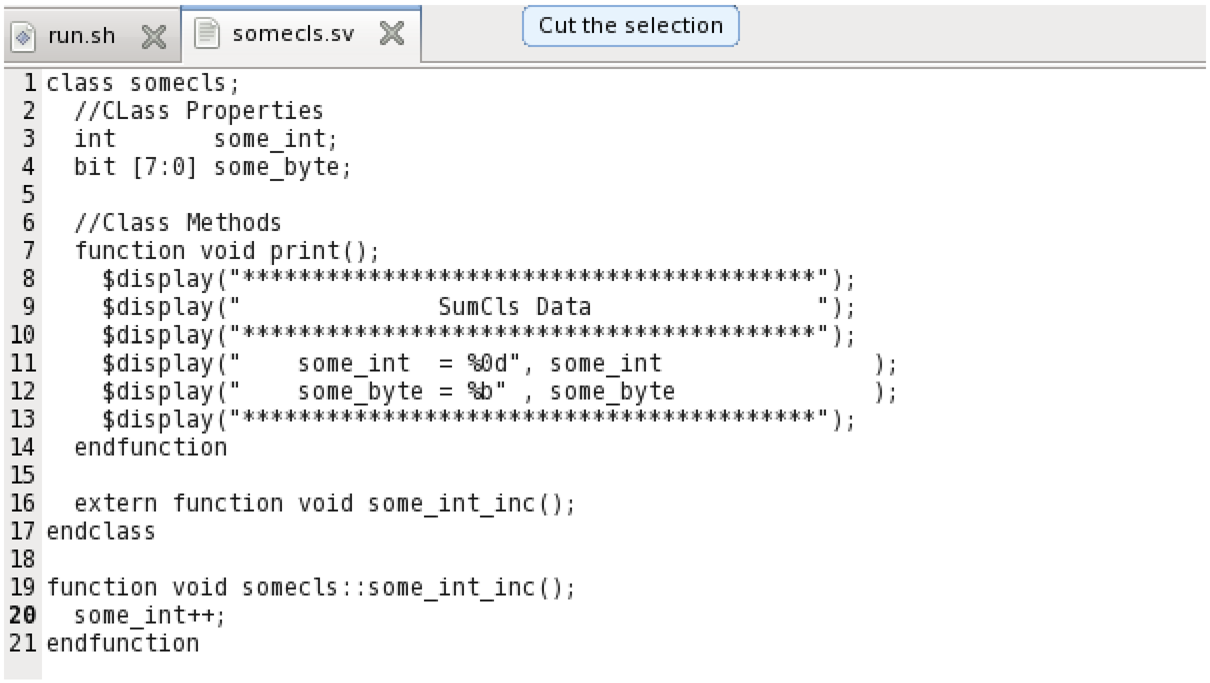
Labwork1

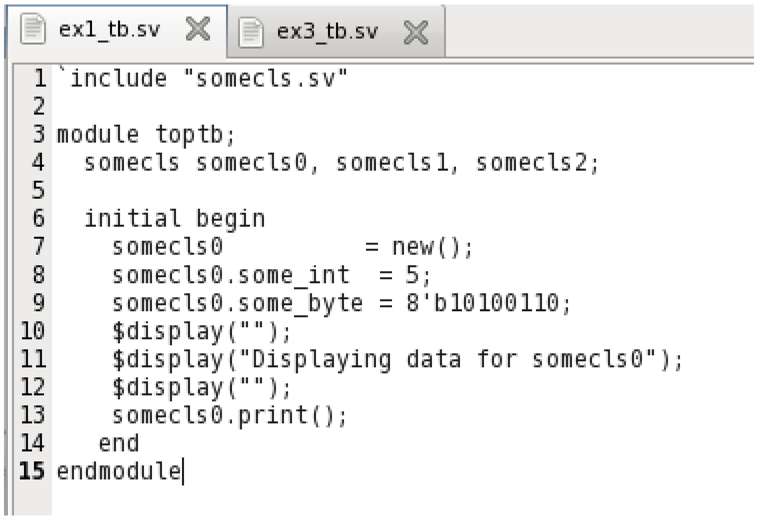
Here we will declare a class object and assigned some values to its variables

Class definition:



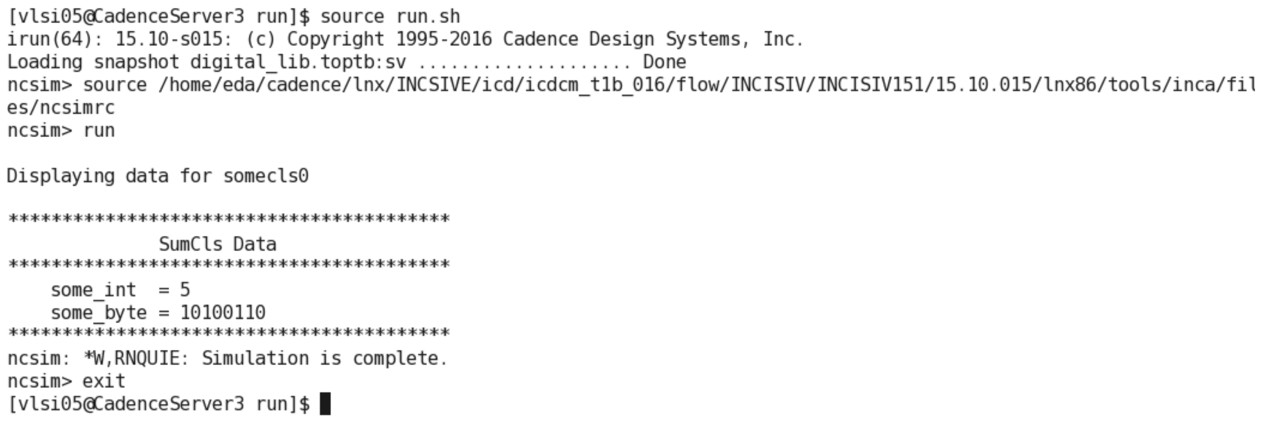
```
class somecls;
  //CLass Properties
  int       some_int;
  bit [7:0] some_byte;

  //Class Methods
  function void print();
    $display("****************************************");
    $display("              SumCls Data               ");
    $display("****************************************");
    $display("    some_int  = %0d", some_int              );
    $display("    some_byte = %b" , some_byte             );
    $display("****************************************");
  endfunction

  extern function void some_int_inc();
endclass

function void somecls::some_int_inc();
  some_int++;
endfunction
```

Testbench Code:



```
`include "somecls.sv"

module toptb;
  somecls somecls0, somecls1, somecls2;

  initial begin
    somecls0          = new();
    somecls0.some_int  = 5;
    somecls0.some_byte = 8'b10100110;
    $display("");
    $display("Displaying data for somecls0");
    $display("");
    somecls0.print();
   end
endmodule
```

Output:



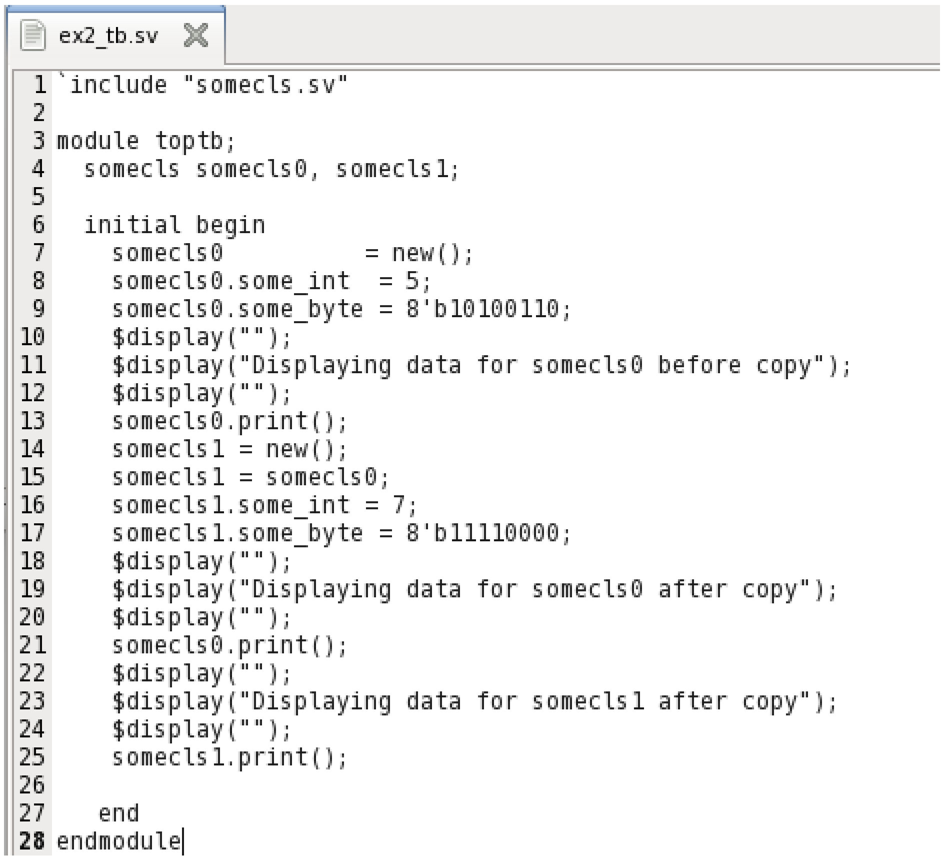
```
[vlsi05@CadenceServer3 run]$ source run.sh
irun(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
Loading snapshot digital_lib.toptb:sv .................... Done
ncsim> source /home/eda/cadence/lnx/INCSIVE/icd/icdcm_t1b_016/flow/INCISIV/INCISIV151/15.10.015/lnx86/tools/inca/fil
es/ncsimrc
ncsim> run

Displaying data for somecls0

*****************************************
              SumCls Data
*****************************************
    some_int  = 5
    some_byte = 10100110
*****************************************
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
[vlsi05@CadenceServer3 run]$ 
```

.

Labwork2

Here we will copy the values from one class object to another where the variables point at the same address in physical memory.

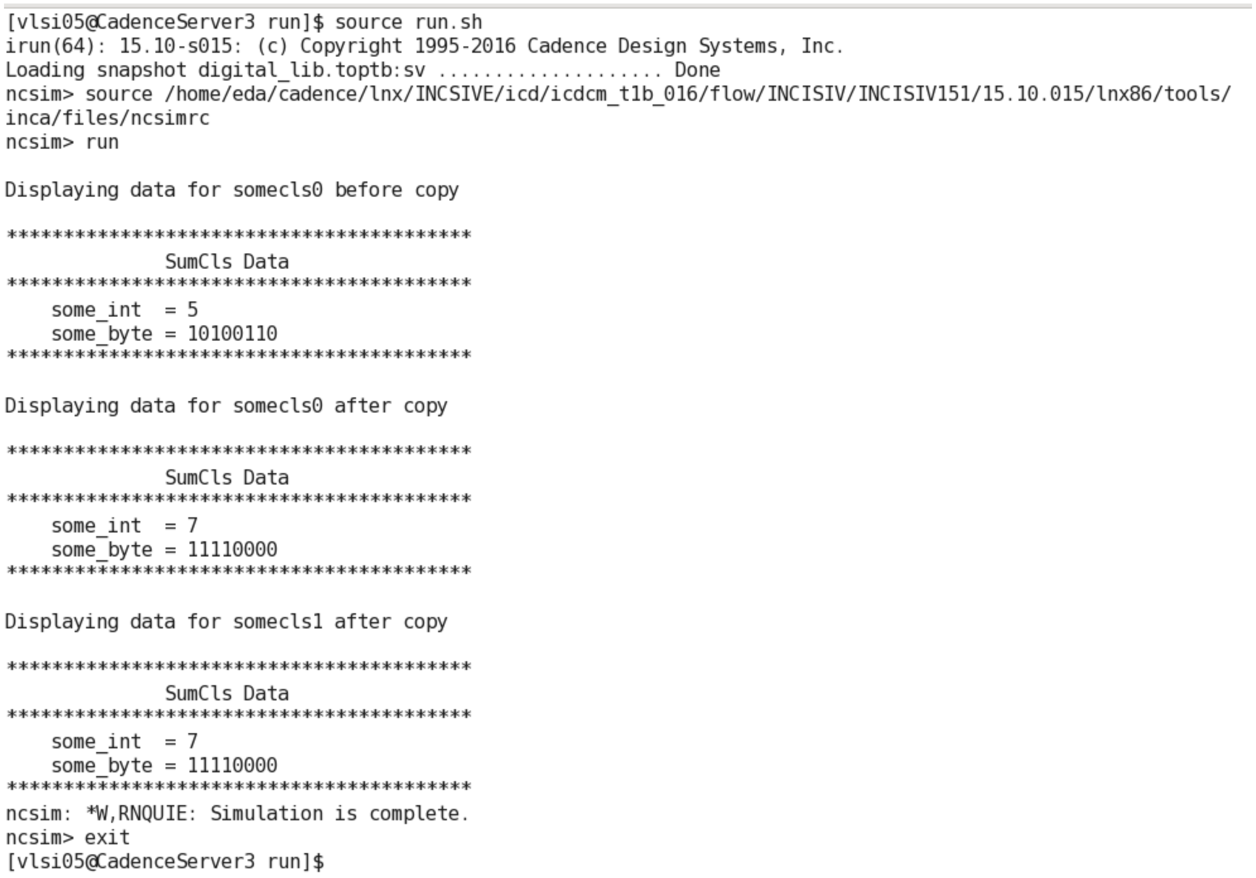
Testbench Code:



ex2_tb.sv

```
`include "somecls.sv"

module toptb;
  somecls somecls0, somecls1;

  initial begin
    somecls0          = new();
    somecls0.some_int  = 5;
    somecls0.some_byte = 8'b10100110;
    $display("");
    $display("Displaying data for somecls0 before copy");
    $display("");
    somecls0.print();
    somecls1 = new();
    somecls1 = somecls0;
    somecls1.some_int = 7;
    somecls1.some_byte = 8'b11110000;
    $display("");
    $display("Displaying data for somecls0 after copy");
    $display("");
    somecls0.print();
    $display("");
    $display("Displaying data for somecls1 after copy");
    $display("");
    somecls1.print();

   end
endmodule
```

output

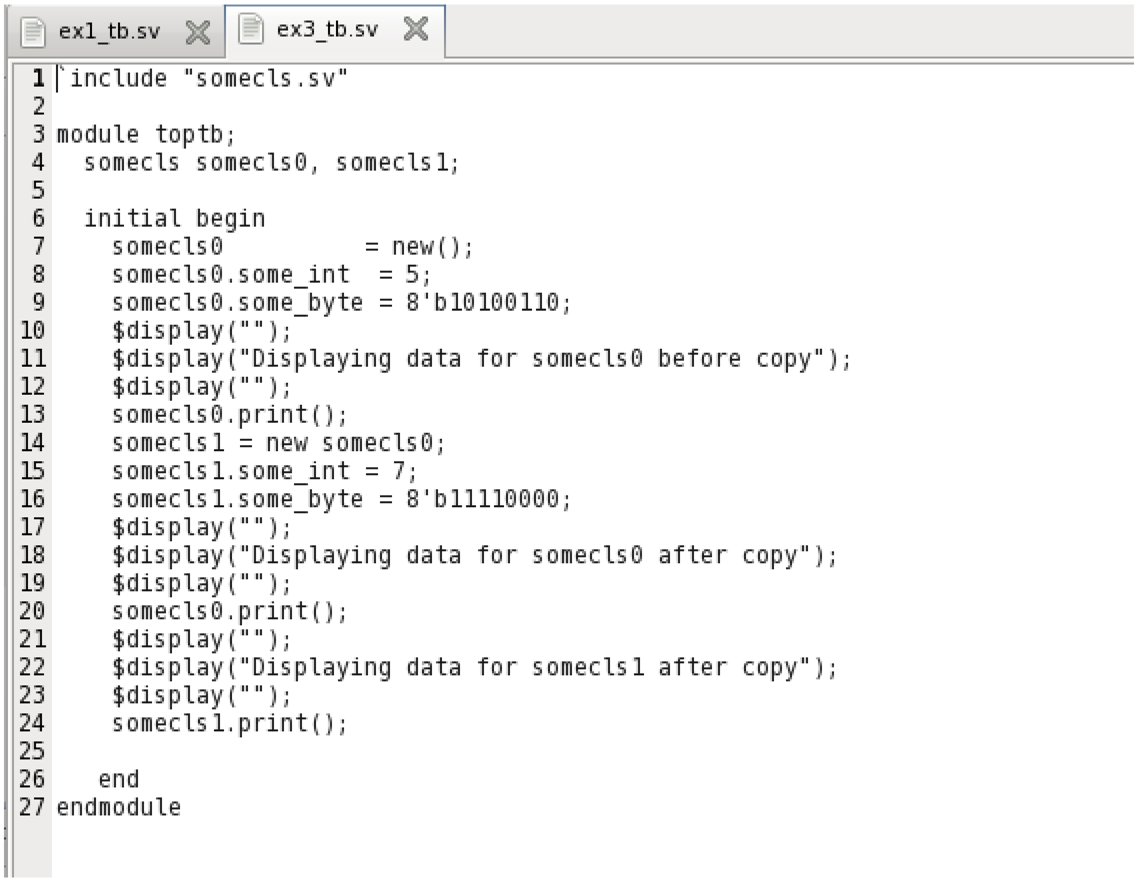


```
[vlsi05@CadenceServer3 run]$ source run.sh
irun(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
Loading snapshot digital_lib.toptb:sv .................... Done
ncsim> source /home/eda/cadence/lnx/INCSIVE/icd/icdcm_t1b_016/flow/INCISIV/INCISIV151/15.10.015/lnx86/tools/
inca/files/ncsimrc
ncsim> run

Displaying data for somecls0 before copy

*****************************************
              SumCls Data
*****************************************
    some_int  = 5
    some_byte = 10100110
*****************************************

Displaying data for somecls0 after copy

*****************************************
              SumCls Data
*****************************************
    some_int  = 7
    some_byte = 11110000
*****************************************

Displaying data for somecls1 after copy

*****************************************
              SumCls Data
*****************************************
    some_int  = 7
    some_byte = 11110000
*****************************************
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
[vlsi05@CadenceServer3 run]$
```

Labwork3

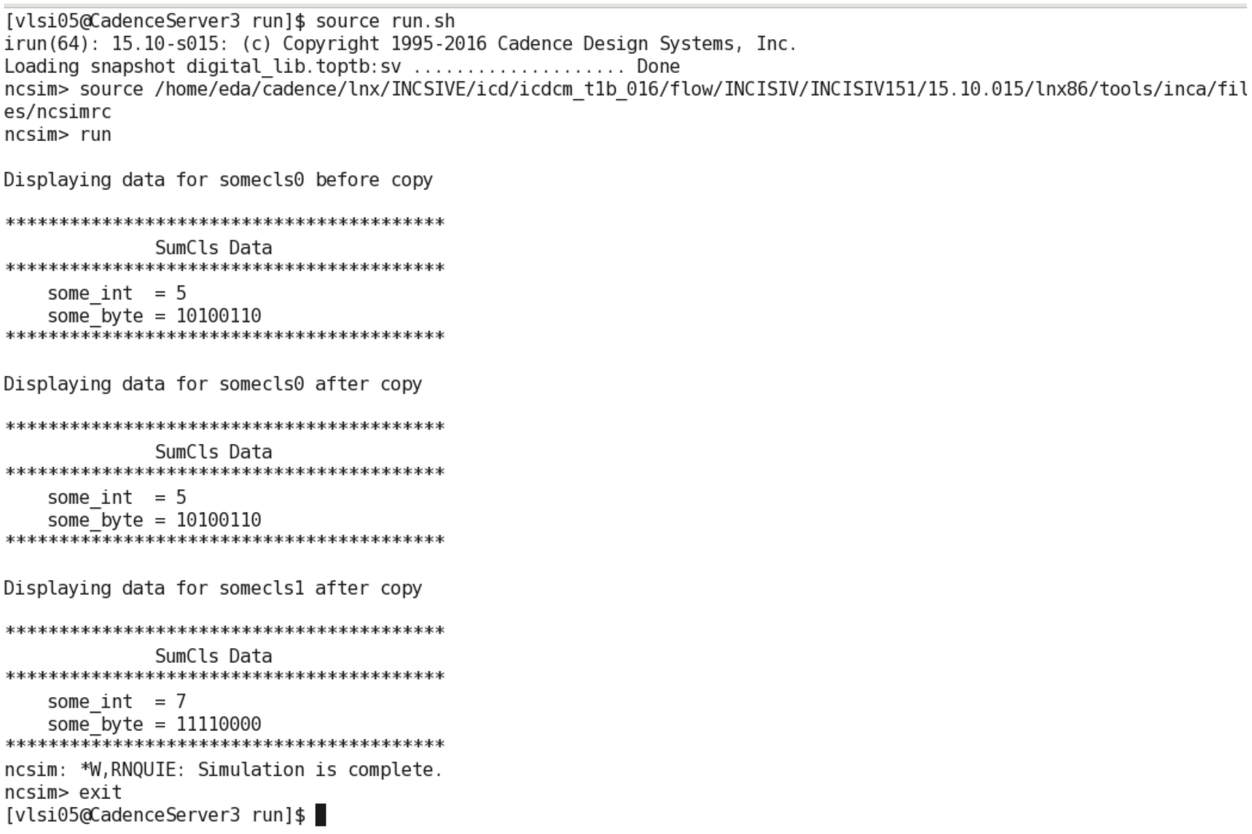
Here we will copy the values from one class object to another where the variables point at the different addresses in physical memory. A new address will be assigned for the variables of the somcls1 by the command new somecls0.

Code



```
`include "somecls.sv"

module toptb;
  somecls somecls0, somecls1;

  initial begin
    somecls0          = new();
    somecls0.some_int  = 5;
    somecls0.some_byte = 8'b10100110;
    $display("");
    $display("Displaying data for somecls0 before copy");
    $display("");
    somecls0.print();
    somecls1 = new somecls0;
    somecls1.some_int = 7;
    somecls1.some_byte = 8'b11110000;
    $display("");
    $display("Displaying data for somecls0 after copy");
    $display("");
    somecls0.print();
    $display("");
    $display("Displaying data for somecls1 after copy");
    $display("");
    somecls1.print();

   end
endmodule
```

Output

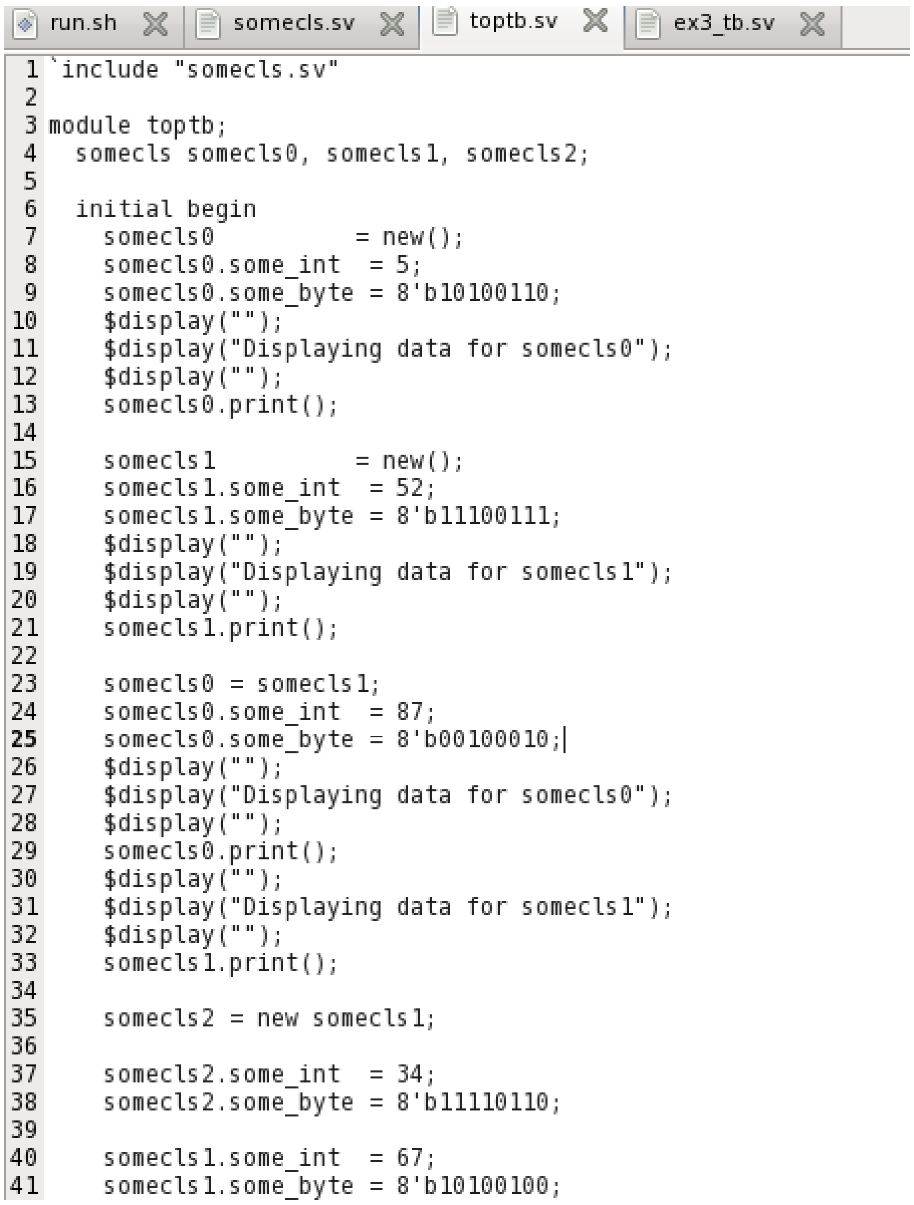


```
[vlsi05@CadenceServer3 run]$ source run.sh
irun(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
Loading snapshot digital_lib.toptb:sv .................... Done
ncsim> source /home/eda/cadence/lnx/INCSIVE/icd/icdcm_t1b_016/flow/INCISIV/INCISIV151/15.10.015/lnx86/tools/inca/fil
es/ncsimrc
ncsim> run

Displaying data for somecls0 before copy

*****************************************
              SumCls Data
*****************************************
    some_int  = 5
    some_byte = 10100110
*****************************************

Displaying data for somecls0 after copy

*****************************************
              SumCls Data
*****************************************
    some_int  = 5
    some_byte = 10100110
*****************************************

Displaying data for somecls1 after copy

*****************************************
              SumCls Data
*****************************************
    some_int  = 7
    some_byte = 11110000
*****************************************
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
[vlsi05@CadenceServer3 run]$ 
```

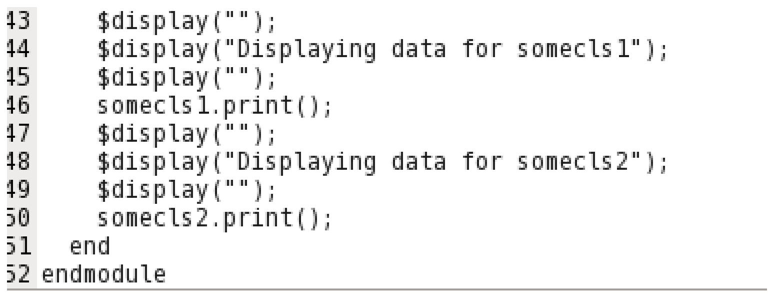
Labwork4

Here we will assign some values to the class objects by shallow and deep copying.

Code:

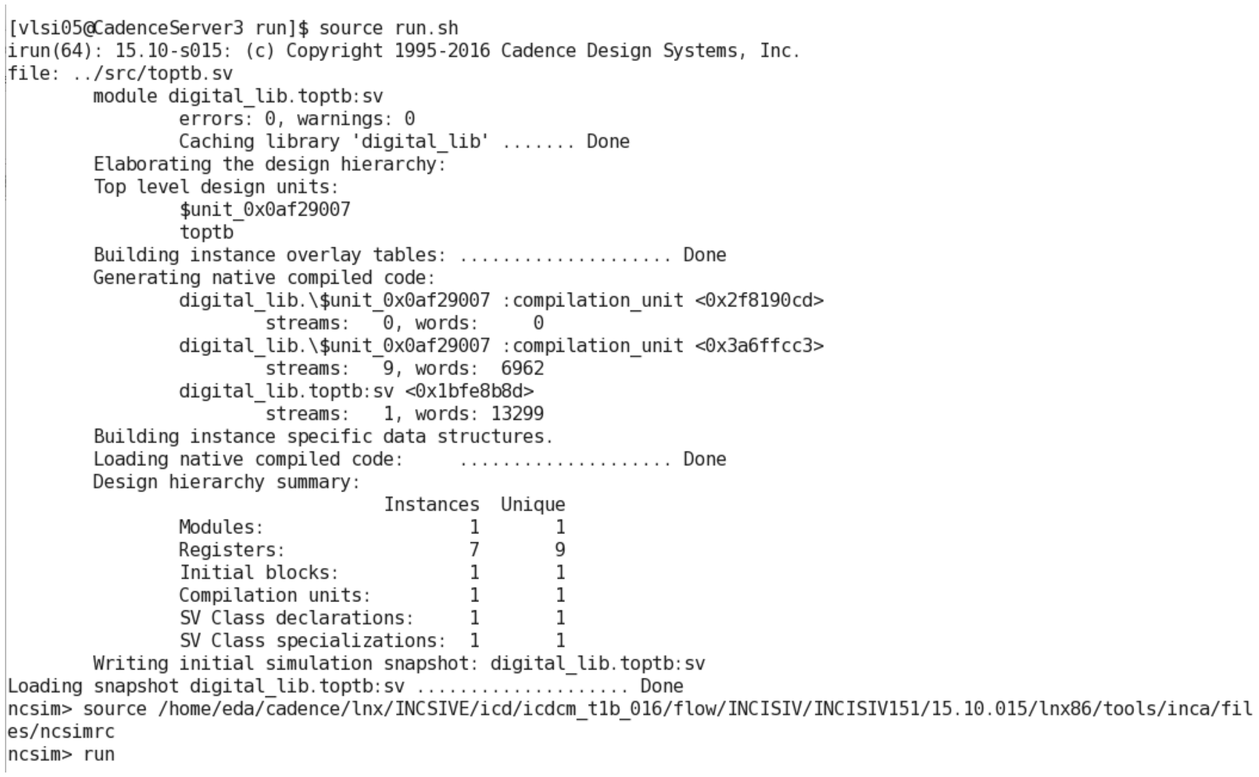


```systemverilog
`include "somecls.sv"

module toptb;
  somecls somecls0, somecls1, somecls2;

  initial begin
    somecls0          = new();
    somecls0.some_int  = 5;
    somecls0.some_byte = 8'b10100110;
    $display("");
    $display("Displaying data for somecls0");
    $display("");
    somecls0.print();

    somecls1          = new();
    somecls1.some_int  = 52;
    somecls1.some_byte = 8'b11100111;
    $display("");
    $display("Displaying data for somecls1");
    $display("");
    somecls1.print();

    somecls0 = somecls1;
    somecls0.some_int  = 87;
    somecls0.some_byte = 8'b00100010;
    $display("");
    $display("Displaying data for somecls0");
    $display("");
    somecls0.print();
    $display("");
    $display("Displaying data for somecls1");
    $display("");
    somecls1.print();

    somecls2 = new somecls1;

    somecls2.some_int  = 34;
    somecls2.some_byte = 8'b11110110;

    somecls1.some_int  = 67;
    somecls1.some_byte = 8'b10100100;
```

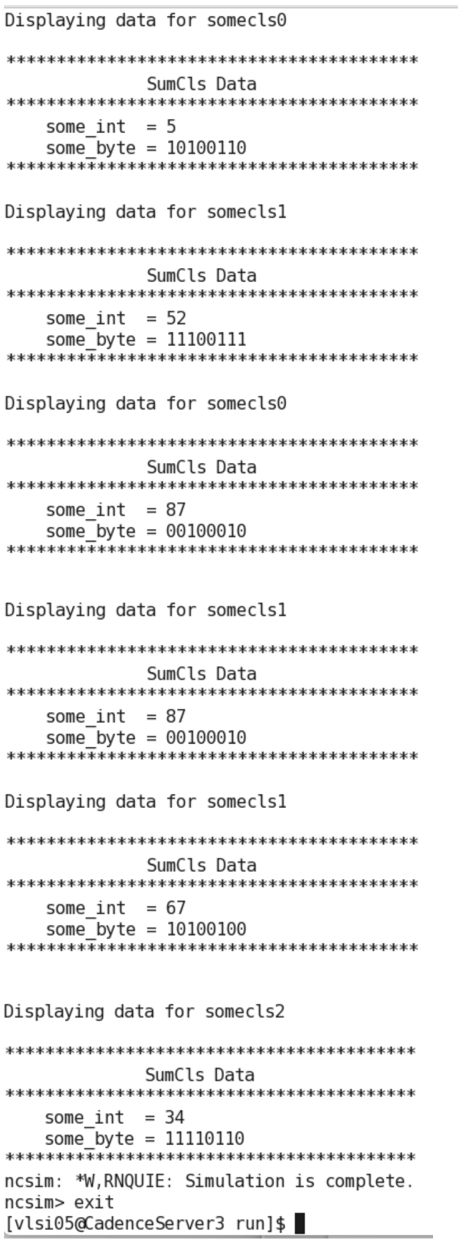


```
    $display("");
    $display("Displaying data for somecls1");
    $display("");
    somecls1.print();
    $display("");
    $display("Displaying data for somecls2");
    $display("");
    somecls2.print();
  end
endmodule
```

Output:



```
[vlsi05@CadenceServer3 run]$ source run.sh
irun(64): 15.10-s015: (c) Copyright 1995-2016 Cadence Design Systems, Inc.
file: ../src/toptb.sv
        module digital_lib.toptb:sv
                errors: 0, warnings: 0
                Caching library 'digital_lib' ....... Done
        Elaborating the design hierarchy:
        Top level design units:
                $unit_0x0af29007
                toptb
        Building instance overlay tables: .................... Done
        Generating native compiled code:
                digital_lib.\$unit_0x0af29007 :compilation_unit <0x2f8190cd>
                        streams:   0, words:     0
                digital_lib.\$unit_0x0af29007 :compilation_unit <0x3a6ffcc3>
                        streams:   9, words:  6962
                digital_lib.toptb:sv <0x1bfe8b8d>
                        streams:   1, words: 13299
        Building instance specific data structures.
        Loading native compiled code:     .................... Done
        Design hierarchy summary:
                                   Instances  Unique
                Modules:                   1       1
                Registers:                 7       9
                Initial blocks:            1       1
                Compilation units:         1       1
                SV Class declarations:     1       1
                SV Class specializations:  1       1
        Writing initial simulation snapshot: digital_lib.toptb:sv
Loading snapshot digital_lib.toptb:sv .................... Done
ncsim> source /home/eda/cadence/lnx/INCSIVE/icd/icdcm_t1b_016/flow/INCISIV/INCISIV151/15.10.015/lnx86/tools/inca/fil
es/ncsimrc
ncsim> run
```



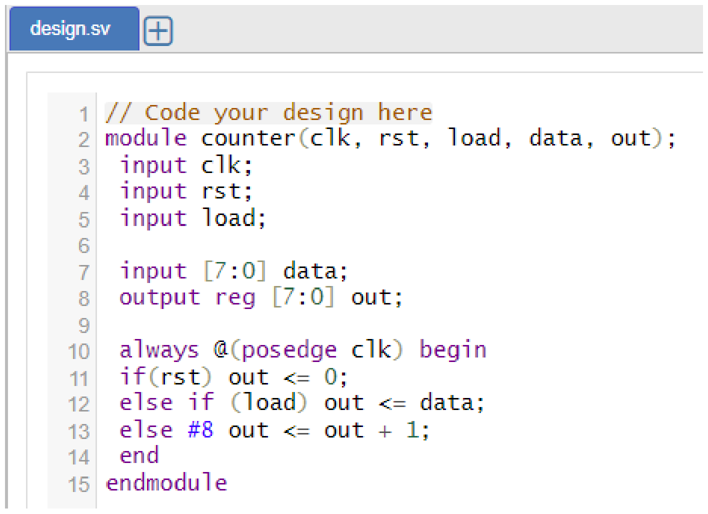
```
Displaying data for somecls0

*****************************************
              SumCls Data
*****************************************
    some_int  = 5
    some_byte = 10100110
*****************************************

Displaying data for somecls1

*****************************************
              SumCls Data
*****************************************
    some_int  = 52
    some_byte = 11100111
*****************************************

Displaying data for somecls0

*****************************************
              SumCls Data
*****************************************
    some_int  = 87
    some_byte = 00100010
*****************************************


Displaying data for somecls1

*****************************************
              SumCls Data
*****************************************
    some_int  = 87
    some_byte = 00100010
*****************************************

Displaying data for somecls1

*****************************************
              SumCls Data
*****************************************
    some_int  = 67
    some_byte = 10100100
*****************************************


Displaying data for somecls2

*****************************************
              SumCls Data
*****************************************
    some_int  = 34
    some_byte = 11110110
*****************************************
ncsim: *W,RNQUIE: Simulation is complete.
ncsim> exit
[vlsi05@CadenceServer3 run]$
```

When we copied somecls1 from somecls0 with shallow copy, the values changed for both the classes together. But when we copied somecls2 from somecls1 with deep copy, assigning value to the first did not change corresponding values for the latter

Labwork5

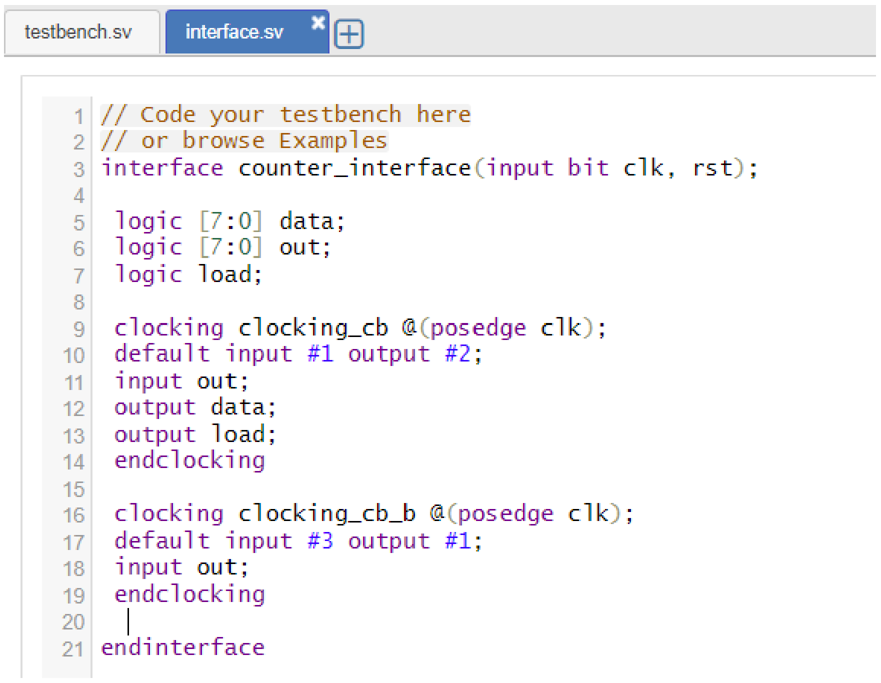
We will now design a simple counter along with a testbench that is connected via an interface.

Design:



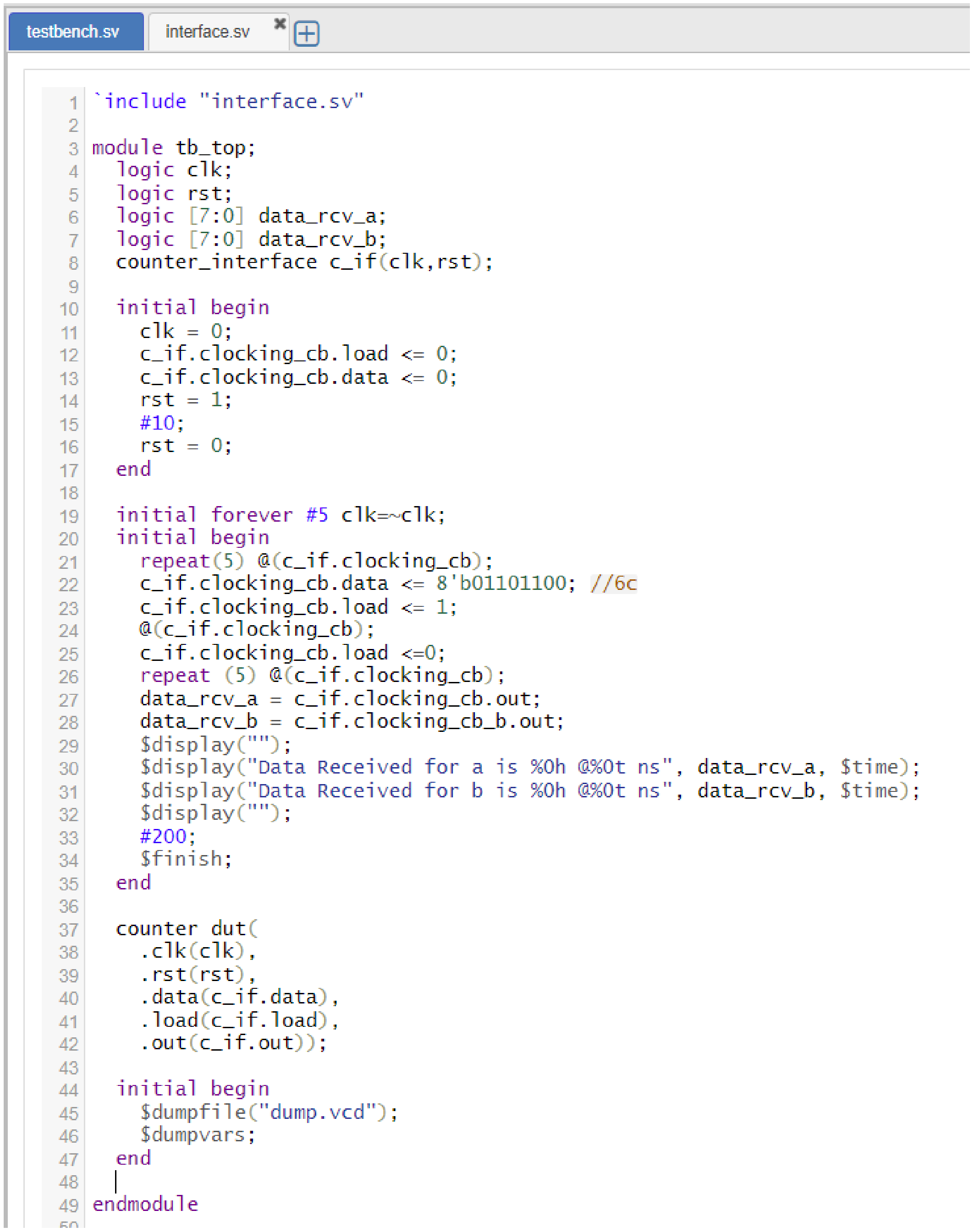
```
// Code your design here
module counter(clk, rst, load, data, out);
 input clk;
 input rst;
 input load;

 input [7:0] data;
 output reg [7:0] out;

 always @(posedge clk) begin
 if(rst) out <= 0;
 else if (load) out <= data;
 else #8 out <= out + 1;
 end
endmodule
```

Interface:



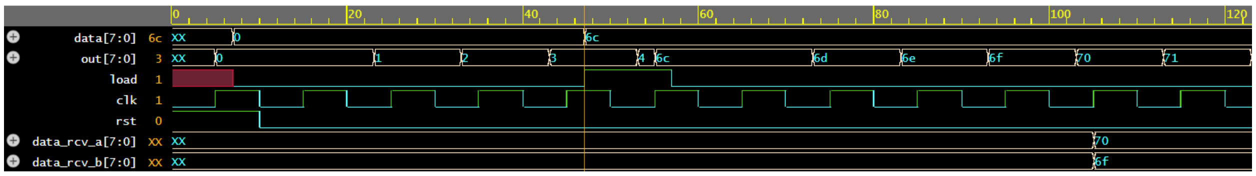
```
// Code your testbench here
// or browse Examples
interface counter_interface(input bit clk, rst);

 logic [7:0] data;
 logic [7:0] out;
 logic load;

 clocking clocking_cb @(posedge clk);
 default input #1 output #2;
 input out;
 output data;
 output load;
 endclocking

 clocking clocking_cb_b @(posedge clk);
 default input #3 output #1;
 input out;
 endclocking

endinterface
```

Testbench:



```
`include "interface.sv"

module tb_top;
  logic clk;
  logic rst;
  logic [7:0] data_rcv_a;
  logic [7:0] data_rcv_b;
  counter_interface c_if(clk,rst);

  initial begin
    clk = 0;
    c_if.clocking_cb.load <= 0;
    c_if.clocking_cb.data <= 0;
    rst = 1;
    #10;
    rst = 0;
  end

  initial forever #5 clk=~clk;
  initial begin
    repeat(5) @(c_if.clocking_cb);
    c_if.clocking_cb.data <= 8'b01101100; //6c
    c_if.clocking_cb.load <= 1;
    @(c_if.clocking_cb);
    c_if.clocking_cb.load <=0;
    repeat (5) @(c_if.clocking_cb);
    data_rcv_a = c_if.clocking_cb.out;
    data_rcv_b = c_if.clocking_cb_b.out;
    $display("");
    $display("Data Received for a is %0h @%0t ns", data_rcv_a, $time);
    $display("Data Received for b is %0h @%0t ns", data_rcv_b, $time);
    $display("");
    #200;
    $finish;
  end

  counter dut(
    .clk(clk),
    .rst(rst),
    .data(c_if.data),
    .load(c_if.load),
    .out(c_if.out));

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end

endmodule
```

EPwave:



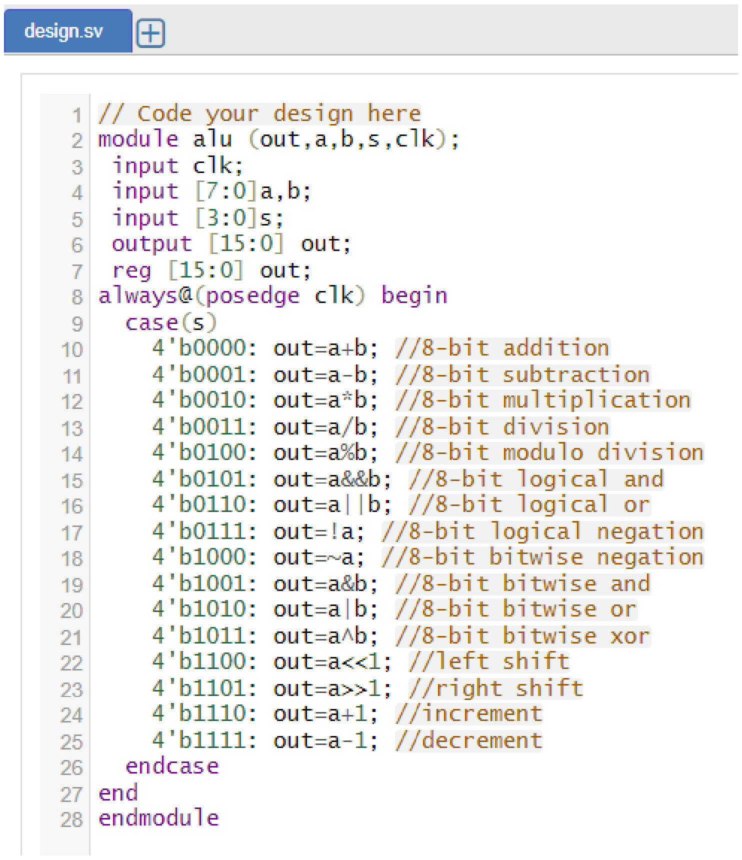
Here, as soon as reset ends, counter starts counting the posedges. The first count is seen after 8ns of the posedge since there was a #8 delay in the counter module. In real designs, delay in design module is not realizable but here we are ignoring it for the experiment.

Counter counts up to 4 in a similar manner and we can see the output in ‘out’ signal from the interface. The data and load are set 2ns after posedge of clk. Once load goes high, the counter loads the data at the next posedge of clock ignoring its current count value 4. Then it keeps counting as before. The data_rcv_a signal is set to sample ‘out’ 1ns before posedge and data_rcv_b is set to sample input 3ns before posedge. Because of this difference the values are different as seen.

Labwork6

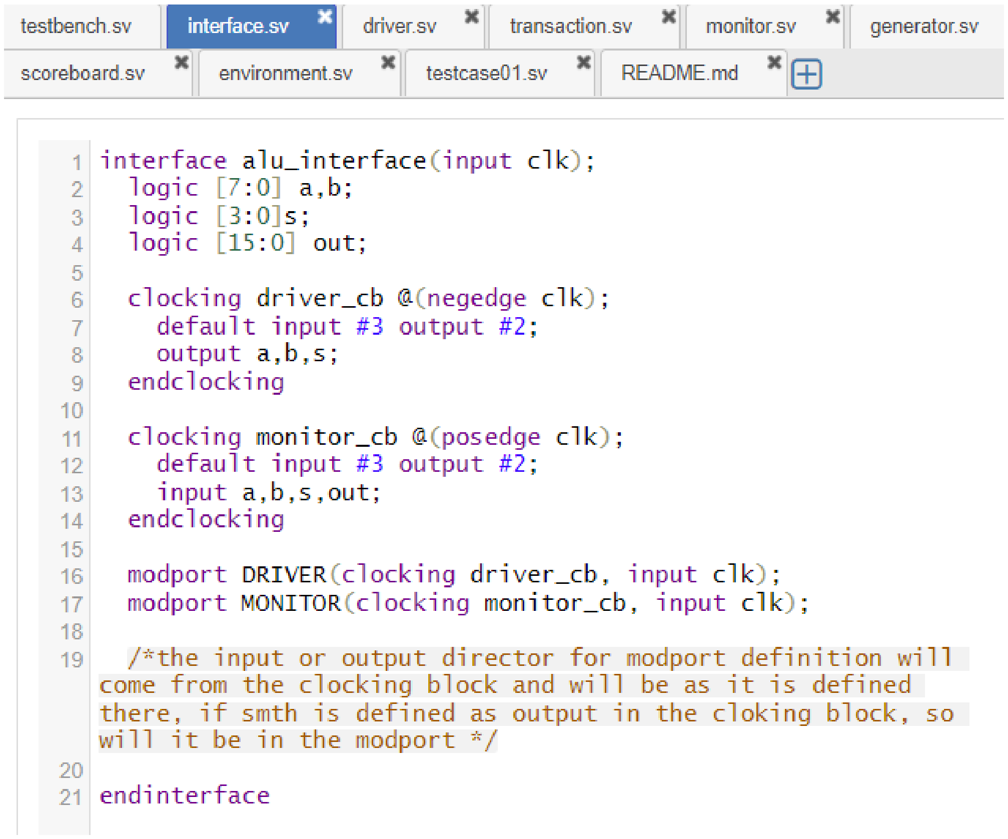
We will now design an ALU module along with a layered testbench

Design:



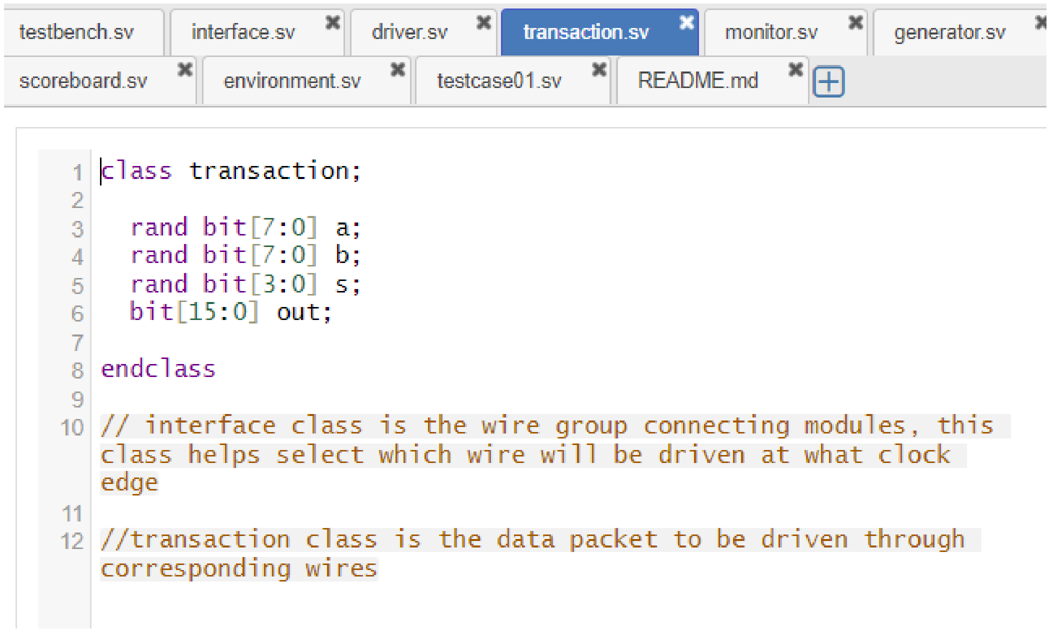
```
// Code your design here
module alu (out,a,b,s,clk);
 input clk;
 input [7:0]a,b;
 input [3:0]s;
 output [15:0] out;
 reg [15:0] out;
always@(posedge clk) begin
  case(s)
    4'b0000: out=a+b; //8-bit addition
    4'b0001: out=a-b; //8-bit subtraction
    4'b0010: out=a*b; //8-bit multiplication
    4'b0011: out=a/b; //8-bit division
    4'b0100: out=a%b; //8-bit modulo division
    4'b0101: out=a&&b; //8-bit logical and
    4'b0110: out=a||b; //8-bit logical or
    4'b0111: out=!a; //8-bit logical negation
    4'b1000: out=~a; //8-bit bitwise negation
    4'b1001: out=a&b; //8-bit bitwise and
    4'b1010: out=a|b; //8-bit bitwise or
    4'b1011: out=a^b; //8-bit bitwise xor
    4'b1100: out=a<<1; //left shift
    4'b1101: out=a>>1; //right shift
    4'b1110: out=a+1; //increment
    4'b1111: out=a-1; //decrement
  endcase
end
endmodule
```

Interface:



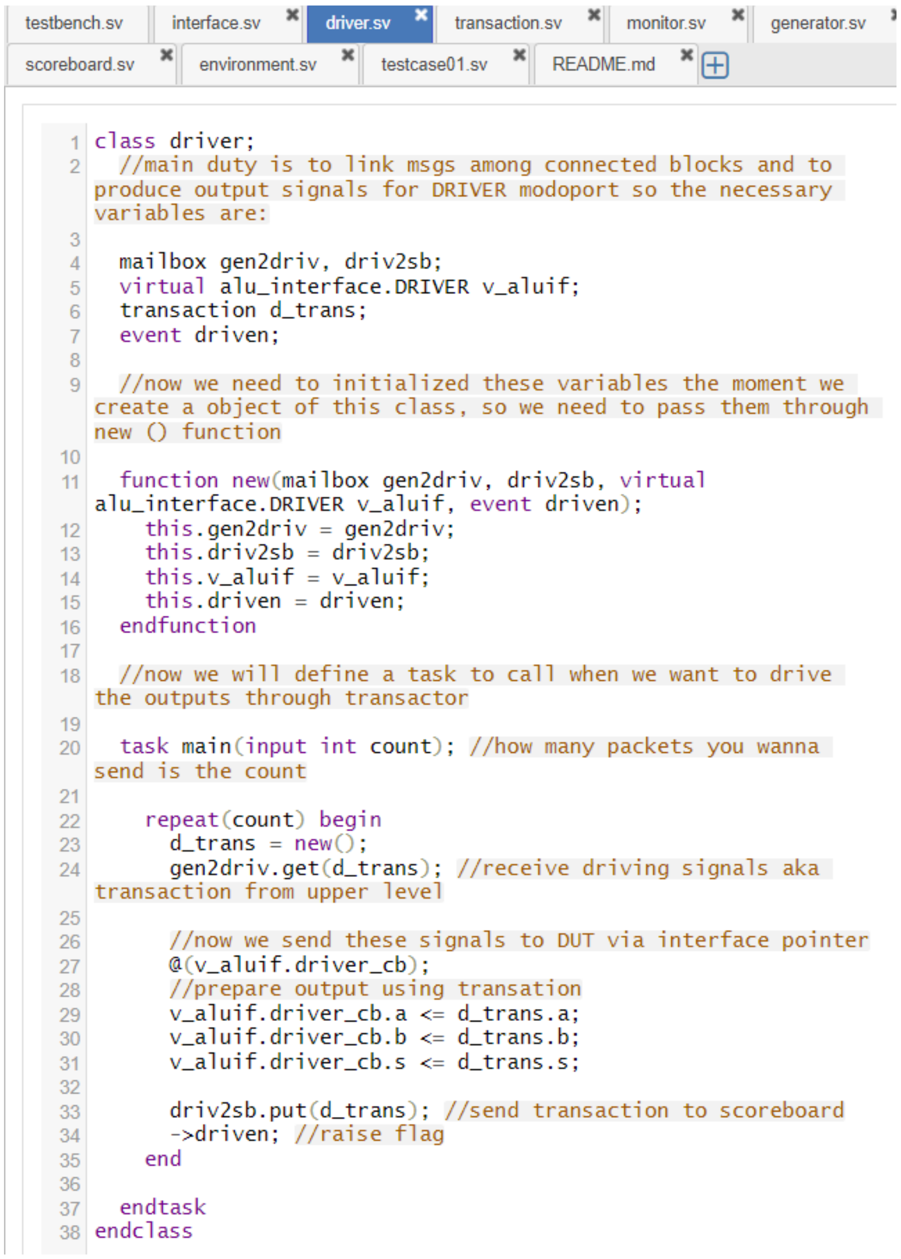
```
interface alu_interface(input clk);
  logic [7:0] a,b;
  logic [3:0]s;
  logic [15:0] out;

  clocking driver_cb @(negedge clk);
    default input #3 output #2;
    output a,b,s;
  endclocking

  clocking monitor_cb @(posedge clk);
    default input #3 output #2;
    input a,b,s,out;
  endclocking

  modport DRIVER(clocking driver_cb, input clk);
  modport MONITOR(clocking monitor_cb, input clk);

  /*the input or output director for modport definition will come from the clocking block and will be as it is defined there, if smth is defined as output in the cloking block, so will it be in the modport */

endinterface
```

Transaction:



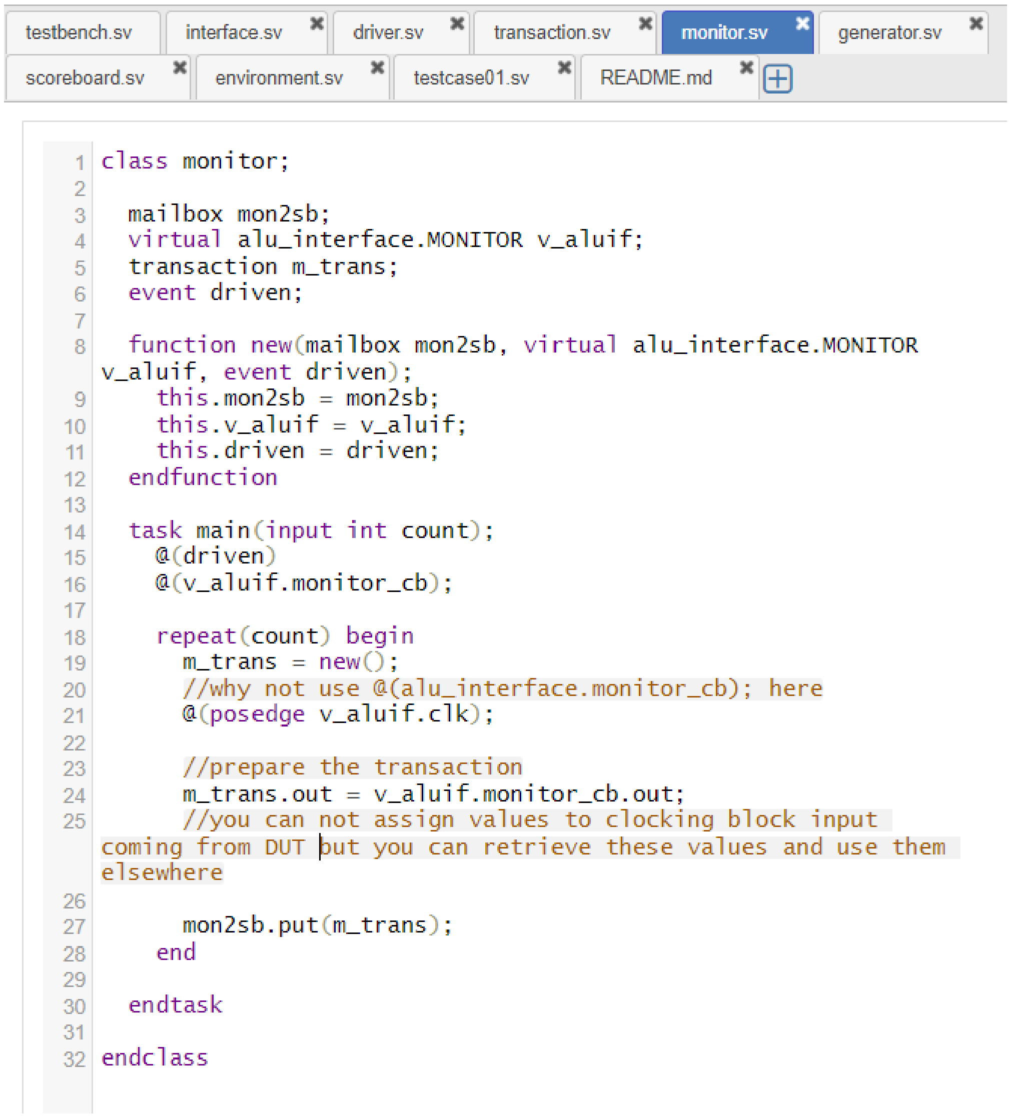
```
class transaction;

  rand bit[7:0] a;
  rand bit[7:0] b;
  rand bit[3:0] s;
  bit[15:0] out;

endclass

// interface class is the wire group connecting modules, this class helps select which wire will be driven at what clock edge

//transaction class is the data packet to be driven through corresponding wires
```

Driver:



```
class driver;
  //main duty is to link msgs among connected blocks and to
produce output signals for DRIVER modoport so the necessary
variables are:

  mailbox gen2driv, driv2sb;
  virtual alu_interface.DRIVER v_aluif;
  transaction d_trans;
  event driven;

  //now we need to initialized these variables the moment we
create a object of this class, so we need to pass them through
new () function

  function new(mailbox gen2driv, driv2sb, virtual
alu_interface.DRIVER v_aluif, event driven);
    this.gen2driv = gen2driv;
    this.driv2sb = driv2sb;
    this.v_aluif = v_aluif;
    this.driven = driven;
  endfunction

  //now we will define a task to call when we want to drive
the outputs through transactor

  task main(input int count); //how many packets you wanna
send is the count

    repeat(count) begin
      d_trans = new();
      gen2driv.get(d_trans); //receive driving signals aka
transaction from upper level

      //now we send these signals to DUT via interface pointer
      @(v_aluif.driver_cb);
      //prepare output using transation
      v_aluif.driver_cb.a <= d_trans.a;
      v_aluif.driver_cb.b <= d_trans.b;
      v_aluif.driver_cb.s <= d_trans.s;

      driv2sb.put(d_trans); //send transaction to scoreboard
      ->driven; //raise flag
    end

  endtask
endclass
```

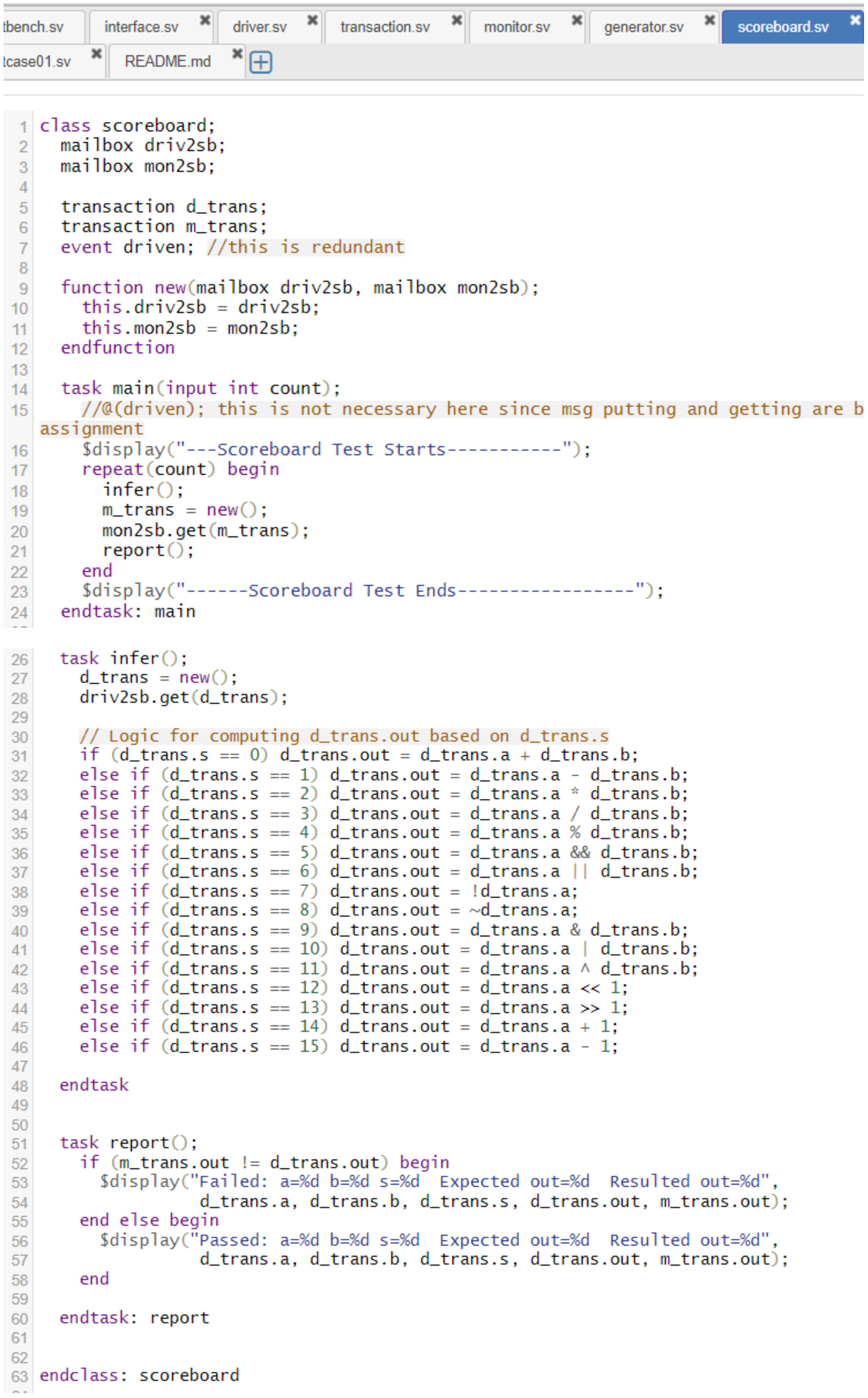
Monitor:



testbench.sv | interface.sv | driver.sv | transaction.sv | monitor.sv | generator.sv | scoreboard.sv | environment.sv | testcase01.sv | README.md

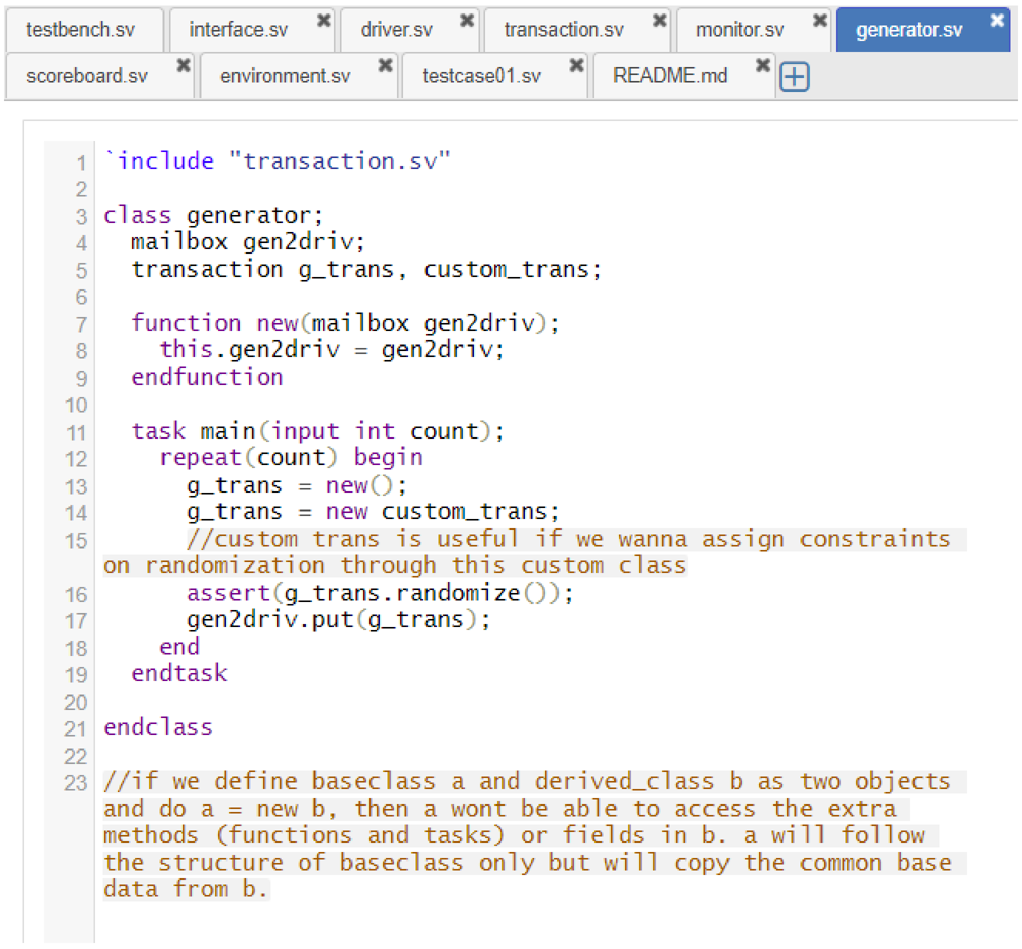
```
class monitor;

  mailbox mon2sb;
  virtual alu_interface.MONITOR v_aluif;
  transaction m_trans;
  event driven;

  function new(mailbox mon2sb, virtual alu_interface.MONITOR
v_aluif, event driven);
    this.mon2sb = mon2sb;
    this.v_aluif = v_aluif;
    this.driven = driven;
  endfunction

  task main(input int count);
    @(driven)
    @(v_aluif.monitor_cb);

    repeat(count) begin
      m_trans = new();
      //why not use @(alu_interface.monitor_cb); here
      @(posedge v_aluif.clk);

      //prepare the transaction
      m_trans.out = v_aluif.monitor_cb.out;
      //you can not assign values to clocking block input
coming from DUT but you can retrieve these values and use them
elsewhere

      mon2sb.put(m_trans);
    end

  endtask

endclass
```

Scoreboard:



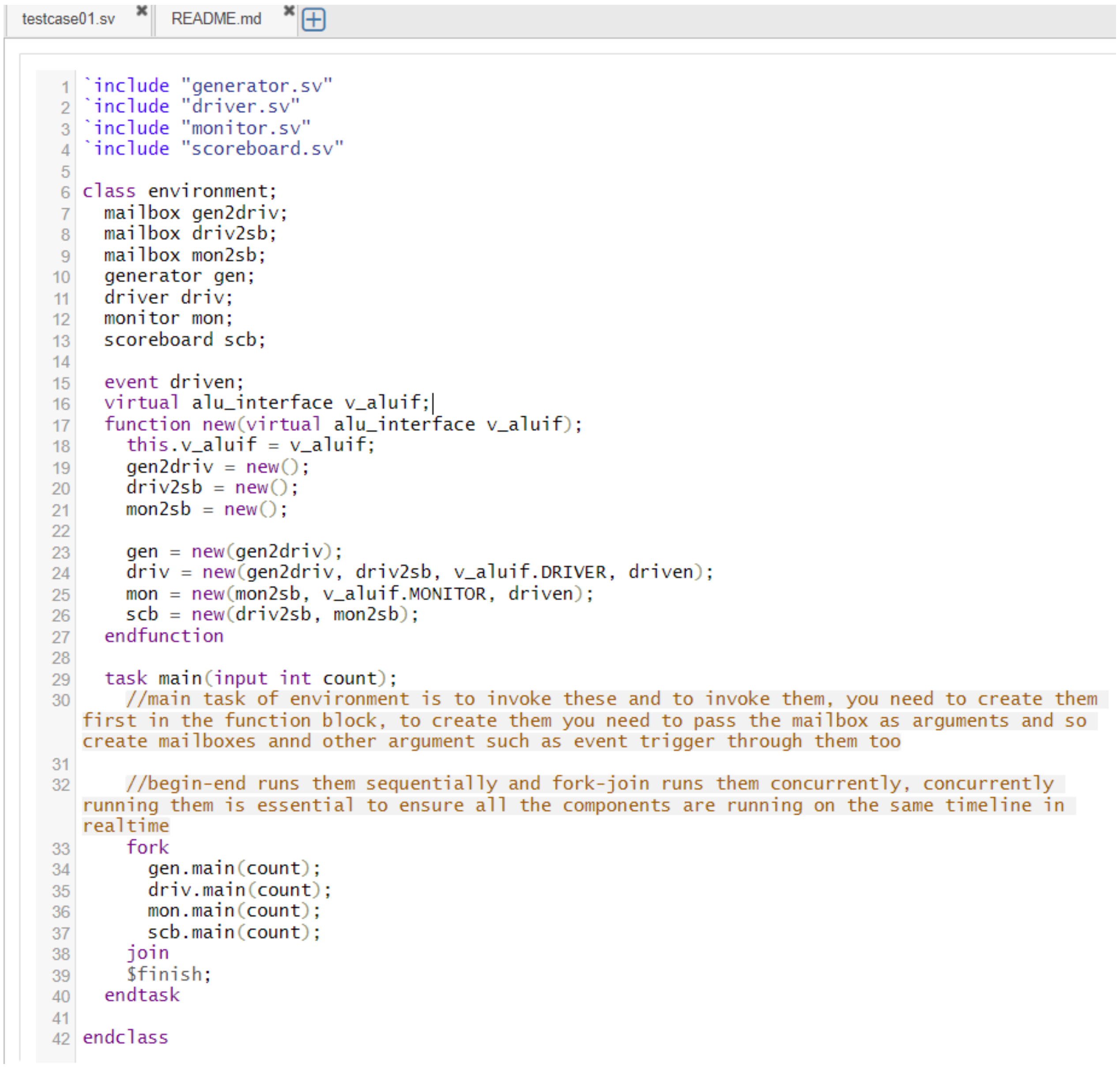
```
class scoreboard;
  mailbox driv2sb;
  mailbox mon2sb;

  transaction d_trans;
  transaction m_trans;
  event driven; //this is redundant

  function new(mailbox driv2sb, mailbox mon2sb);
    this.driv2sb = driv2sb;
    this.mon2sb = mon2sb;
  endfunction

  task main(input int count);
    //@(driven); this is not necessary here since msg putting and getting are b
assignment
    $display("---Scoreboard Test Starts----------");
    repeat(count) begin
      infer();
      m_trans = new();
      mon2sb.get(m_trans);
      report();
    end
    $display("------Scoreboard Test Ends-----------------");
  endtask: main

  task infer();
    d_trans = new();
    driv2sb.get(d_trans);

    // Logic for computing d_trans.out based on d_trans.s
    if (d_trans.s == 0) d_trans.out = d_trans.a + d_trans.b;
    else if (d_trans.s == 1) d_trans.out = d_trans.a - d_trans.b;
    else if (d_trans.s == 2) d_trans.out = d_trans.a * d_trans.b;
    else if (d_trans.s == 3) d_trans.out = d_trans.a / d_trans.b;
    else if (d_trans.s == 4) d_trans.out = d_trans.a % d_trans.b;
    else if (d_trans.s == 5) d_trans.out = d_trans.a && d_trans.b;
    else if (d_trans.s == 6) d_trans.out = d_trans.a || d_trans.b;
    else if (d_trans.s == 7) d_trans.out = !d_trans.a;
    else if (d_trans.s == 8) d_trans.out = ~d_trans.a;
    else if (d_trans.s == 9) d_trans.out = d_trans.a & d_trans.b;
    else if (d_trans.s == 10) d_trans.out = d_trans.a | d_trans.b;
    else if (d_trans.s == 11) d_trans.out = d_trans.a ^ d_trans.b;
    else if (d_trans.s == 12) d_trans.out = d_trans.a << 1;
    else if (d_trans.s == 13) d_trans.out = d_trans.a >> 1;
    else if (d_trans.s == 14) d_trans.out = d_trans.a + 1;
    else if (d_trans.s == 15) d_trans.out = d_trans.a - 1;

  endtask


  task report();
    if (m_trans.out != d_trans.out) begin
      $display("Failed: a=%d b=%d s=%d  Expected out=%d  Resulted out=%d",
                d_trans.a, d_trans.b, d_trans.s, d_trans.out, m_trans.out);
    end else begin
      $display("Passed: a=%d b=%d s=%d  Expected out=%d  Resulted out=%d",
                d_trans.a, d_trans.b, d_trans.s, d_trans.out, m_trans.out);
    end

  endtask: report


endclass: scoreboard
```

Generator:



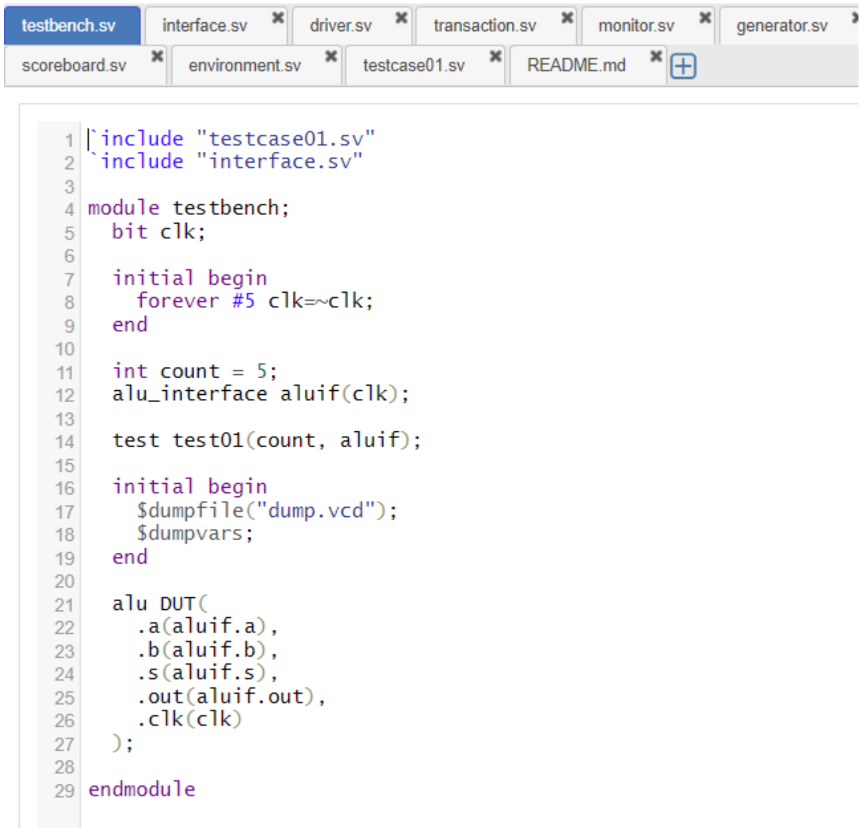
```
`include "transaction.sv"

class generator;
  mailbox gen2driv;
  transaction g_trans, custom_trans;

  function new(mailbox gen2driv);
    this.gen2driv = gen2driv;
  endfunction

  task main(input int count);
    repeat(count) begin
      g_trans = new();
      g_trans = new custom_trans;
      //custom trans is useful if we wanna assign constraints on randomization through this custom class
      assert(g_trans.randomize());
      gen2driv.put(g_trans);
    end
  endtask

endclass

//if we define baseclass a and derived_class b as two objects and do a = new b, then a wont be able to access the extra methods (functions and tasks) or fields in b. a will follow the structure of baseclass only but will copy the common base data from b.
```

Environment:


testcase01.sv README.md

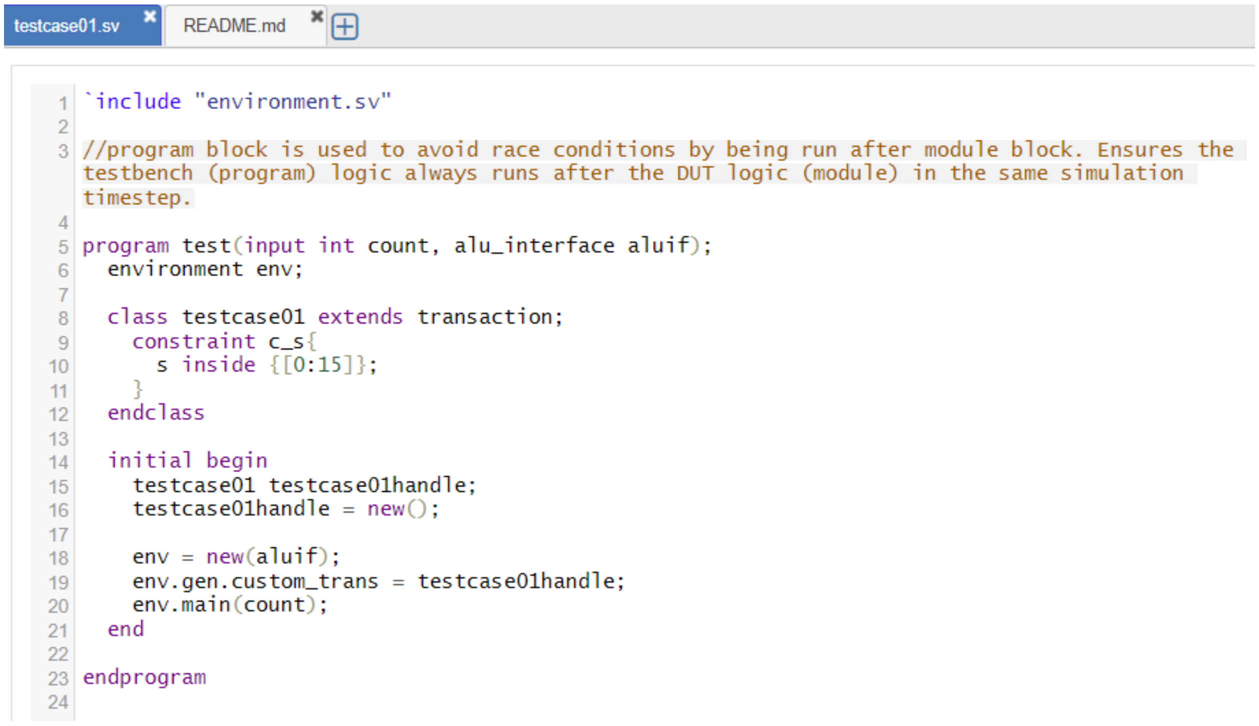
```
`include "generator.sv"
`include "driver.sv"
`include "monitor.sv"
`include "scoreboard.sv"

class environment;
  mailbox gen2driv;
  mailbox driv2sb;
  mailbox mon2sb;
  generator gen;
  driver driv;
  monitor mon;
  scoreboard scb;

  event driven;
  virtual alu_interface v_aluif;
  function new(virtual alu_interface v_aluif);
    this.v_aluif = v_aluif;
    gen2driv = new();
    driv2sb = new();
    mon2sb = new();

    gen = new(gen2driv);
    driv = new(gen2driv, driv2sb, v_aluif.DRIVER, driven);
    mon = new(mon2sb, v_aluif.MONITOR, driven);
    scb = new(driv2sb, mon2sb);
  endfunction

  task main(input int count);
    //main task of environment is to invoke these and to invoke them, you need to create them first in the function block, to create them you need to pass the mailbox as arguments and so create mailboxes annd other argument such as event trigger through them too

    //begin-end runs them sequentially and fork-join runs them concurrently, concurrently running them is essential to ensure all the components are running on the same timeline in realtime
    fork
      gen.main(count);
      driv.main(count);
      mon.main(count);
      scb.main(count);
    join
    $finish;
  endtask

endclass
```

Testbench:



```
`include "testcase01.sv"
`include "interface.sv"

module testbench;
  bit clk;

  initial begin
    forever #5 clk=~clk;
  end

  int count = 5;
  alu_interface aluif(clk);

  test test01(count, aluif);

  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end

  alu DUT(
    .a(aluif.a),
    .b(aluif.b),
    .s(aluif.s),
    .out(aluif.out),
    .clk(clk)
  );

endmodule
```

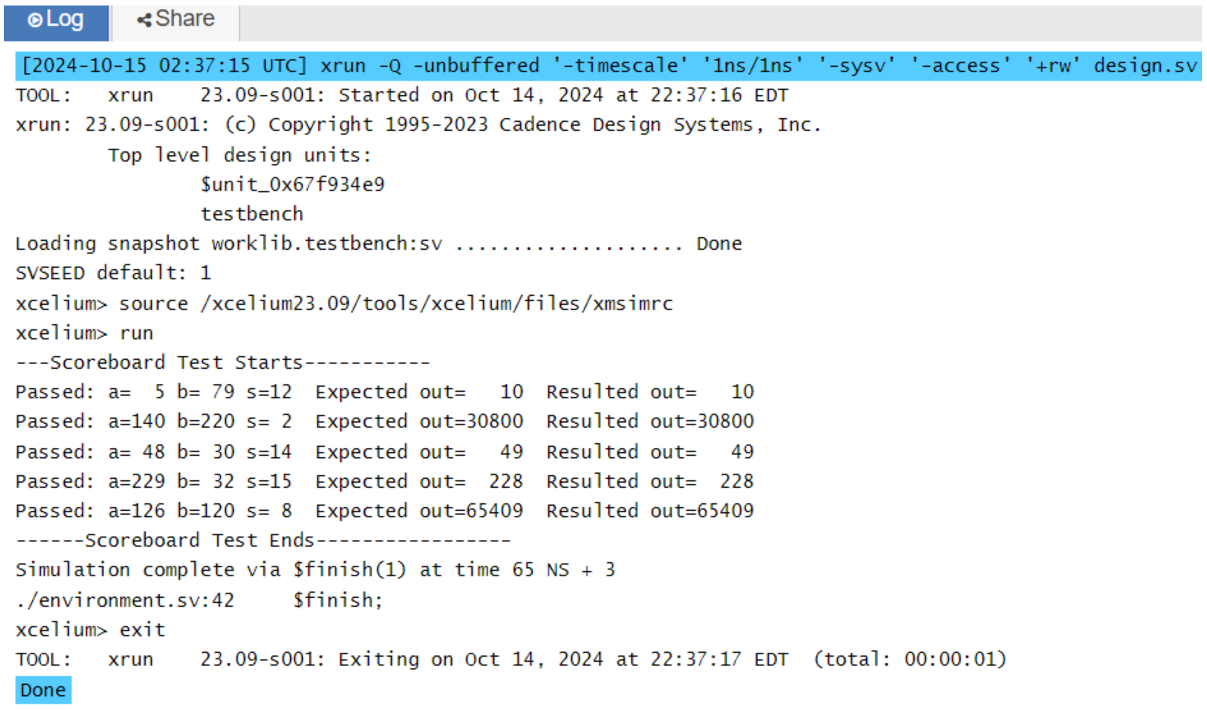
Testcase01:



testcase01.sv README.md

```
`include "environment.sv"

//program block is used to avoid race conditions by being run after module block. Ensures the testbench (program) logic always runs after the DUT logic (module) in the same simulation timestep.

program test(input int count, alu_interface aluif);
  environment env;

  class testcase01 extends transaction;
    constraint c_s{
      s inside {[0:15]};
    }
  endclass

  initial begin
    testcase01 testcase01handle;
    testcase01handle = new();

    env = new(aluif);
    env.gen.custom_trans = testcase01handle;
    env.main(count);
  end

endprogram
```

Output:



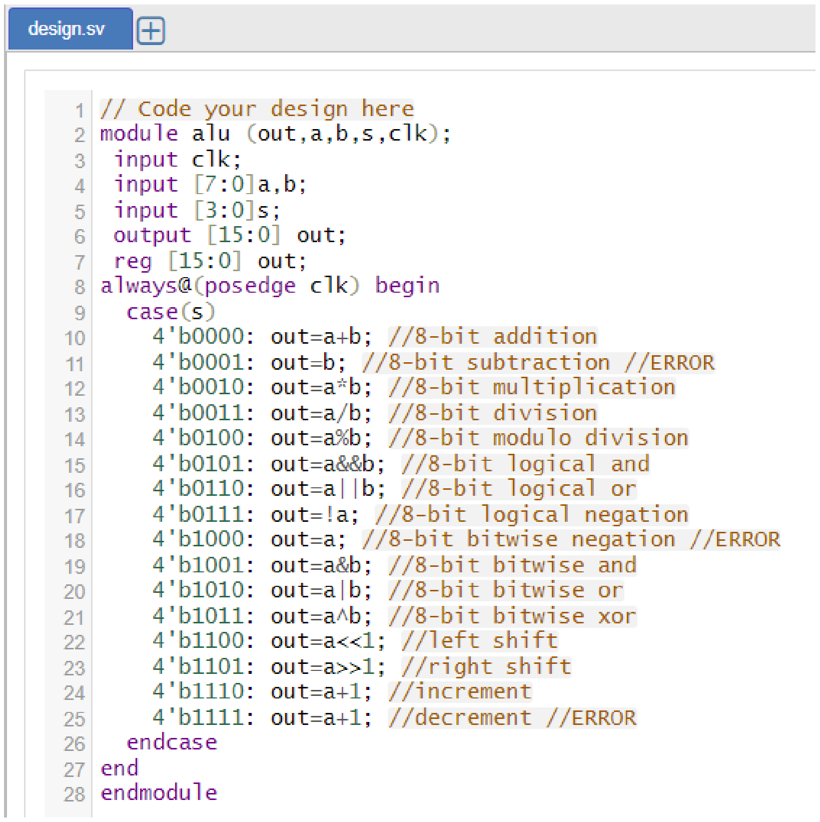
Log Share

```
[2024-10-15 02:37:15 UTC] xrun -Q -unbuffered '-timescale' '1ns/1ns' '-sysv' '-access' '+rw' design.sv
TOOL:	xrun	23.09-s001: Started on Oct 14, 2024 at 22:37:16 EDT
xrun: 23.09-s001: (c) Copyright 1995-2023 Cadence Design Systems, Inc.
	Top level design units:
		$unit_0x67f934e9
		testbench
Loading snapshot worklib.testbench:sv .................... Done
SVSEED default: 1
xcelium> source /xcelium23.09/tools/xcelium/files/xmsimrc
xcelium> run
---Scoreboard Test Starts-----------
Passed: a=  5 b= 79 s=12  Expected out=   10  Resulted out=   10
Passed: a=140 b=220 s= 2  Expected out=30800  Resulted out=30800
Passed: a= 48 b= 30 s=14  Expected out=   49  Resulted out=   49
Passed: a=229 b= 32 s=15  Expected out=  228  Resulted out=  228
Passed: a=126 b=120 s= 8  Expected out=65409  Resulted out=65409
------Scoreboard Test Ends-----------------
Simulation complete via $finish(1) at time 65 NS + 3
./environment.sv:42 	$finish;
xcelium> exit
TOOL:	xrun	23.09-s001: Exiting on Oct 14, 2024 at 22:37:17 EDT  (total: 00:00:01)
Done
```

Report Task1:

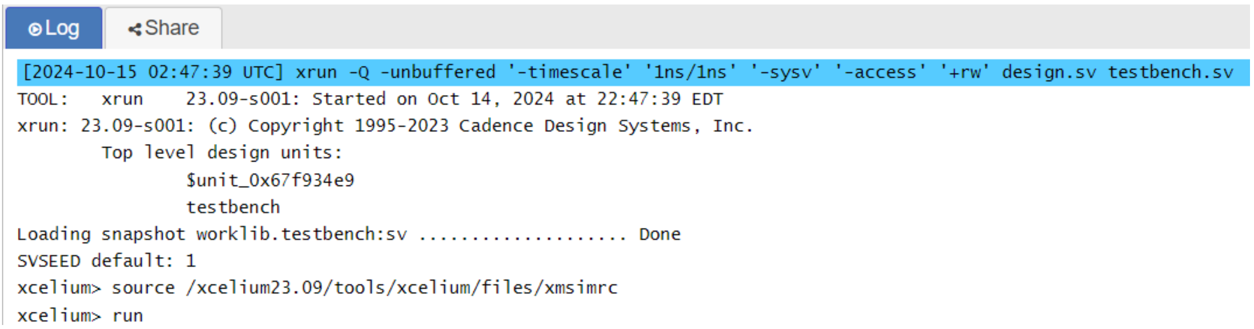
Intentionally introduce some bugs in the RTL code of the ALU and use layered testbench to identify the bugs. Modify your testbench to increase coverage as much as possible and to identify all the errors.

Code changes:

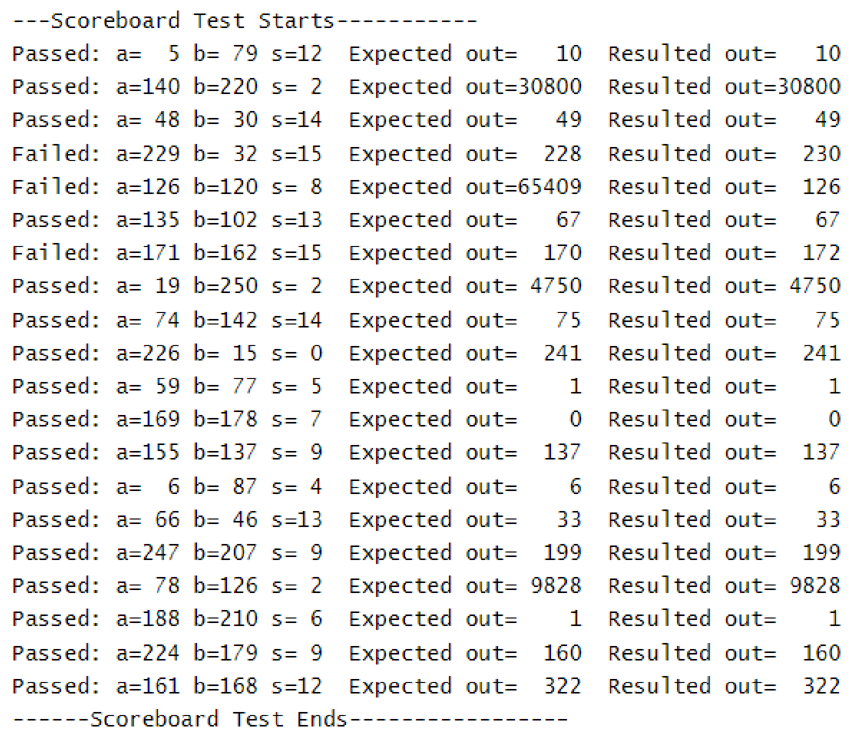


```
// Code your design here
module alu (out,a,b,s,clk);
 input clk;
 input [7:0]a,b;
 input [3:0]s;
 output [15:0] out;
 reg [15:0] out;
always@(posedge clk) begin
  case(s)
    4'b0000: out=a+b; //8-bit addition
    4'b0001: out=b; //8-bit subtraction //ERROR
    4'b0010: out=a*b; //8-bit multiplication
    4'b0011: out=a/b; //8-bit division
    4'b0100: out=a%b; //8-bit modulo division
    4'b0101: out=a&&b; //8-bit logical and
    4'b0110: out=a||b; //8-bit logical or
    4'b0111: out=!a; //8-bit logical negation
    4'b1000: out=a; //8-bit bitwise negation //ERROR
    4'b1001: out=a&b; //8-bit bitwise and
    4'b1010: out=a|b; //8-bit bitwise or
    4'b1011: out=a^b; //8-bit bitwise xor
    4'b1100: out=a<<1; //left shift
    4'b1101: out=a>>1; //right shift
    4'b1110: out=a+1; //increment
    4'b1111: out=a+1; //decrement //ERROR
  endcase
end
endmodule
```

Output:



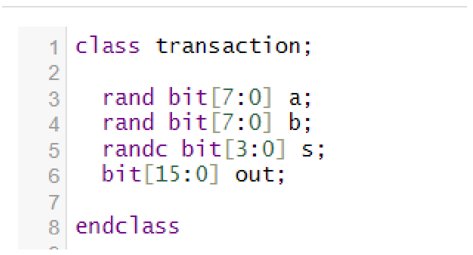
```
[2024-10-15 02:47:39 UTC] xrun -Q -unbuffered '-timescale' '1ns/1ns' '-sysv' '-access' '+rw' design.sv testbench.sv
TOOL:   xrun    23.09-s001: Started on Oct 14, 2024 at 22:47:39 EDT
xrun: 23.09-s001: (c) Copyright 1995-2023 Cadence Design Systems, Inc.
        Top level design units:
                $unit_0x67f934e9
                testbench
Loading snapshot worklib.testbench:sv .................... Done
SVSEED default: 1
xcelium> source /xcelium23.09/tools/xcelium/files/xmsimrc
xcelium> run
```



```
---Scoreboard Test Starts-----------
Passed: a=  5 b= 79 s=12  Expected out=   10  Resulted out=   10
Passed: a=140 b=220 s= 2  Expected out=30800  Resulted out=30800
Passed: a= 48 b= 30 s=14  Expected out=   49  Resulted out=   49
Failed: a=229 b= 32 s=15  Expected out=  228  Resulted out=  230
Failed: a=126 b=120 s= 8  Expected out=65409  Resulted out=  126
Passed: a=135 b=102 s=13  Expected out=   67  Resulted out=   67
Failed: a=171 b=162 s=15  Expected out=  170  Resulted out=  172
Passed: a= 19 b=250 s= 2  Expected out= 4750  Resulted out= 4750
Passed: a= 74 b=142 s=14  Expected out=   75  Resulted out=   75
Passed: a=226 b= 15 s= 0  Expected out=  241  Resulted out=  241
Passed: a= 59 b= 77 s= 5  Expected out=    1  Resulted out=    1
Passed: a=169 b=178 s= 7  Expected out=    0  Resulted out=    0
Passed: a=155 b=137 s= 9  Expected out=  137  Resulted out=  137
Passed: a=  6 b= 87 s= 4  Expected out=    6  Resulted out=    6
Passed: a= 66 b= 46 s=13  Expected out=   33  Resulted out=   33
Passed: a=247 b=207 s= 9  Expected out=  199  Resulted out=  199
Passed: a= 78 b=126 s= 2  Expected out= 9828  Resulted out= 9828
Passed: a=188 b=210 s= 6  Expected out=    1  Resulted out=    1
Passed: a=224 b=179 s= 9  Expected out=  160  Resulted out=  160
Passed: a=161 b=168 s=12  Expected out=  322  Resulted out=  322
------Scoreboard Test Ends-----------------
```

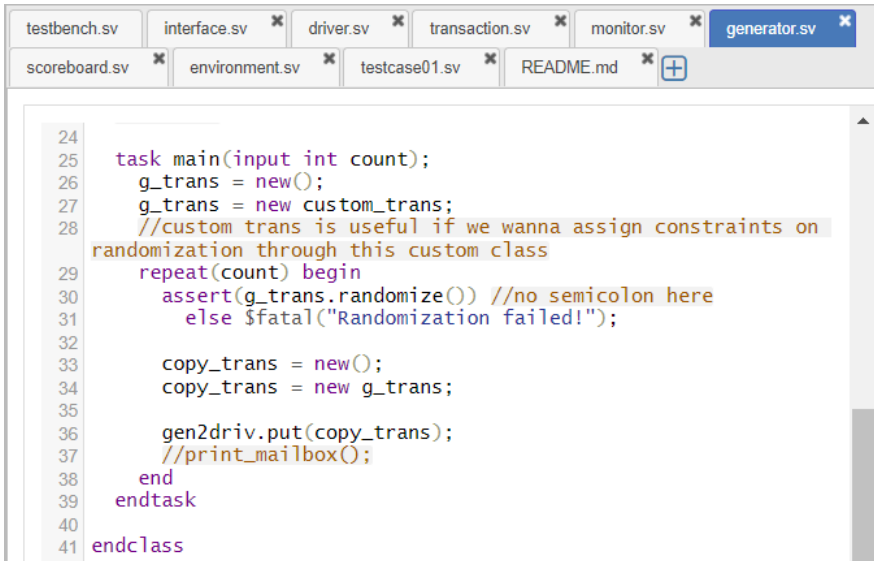
Here, even though we introduced three intentional errors, all the bugs are not identified due to a random selection of test cases. Now we will use cyclic randomization for select signal to ensure we run all the possible ALU operations.

Changed code:



```
class transaction;

  rand bit[7:0] a;
  rand bit[7:0] b;
  randc bit[3:0] s;
  bit[15:0] out;

endclass
```

Here the possible variation for s signal was just 16 and so using cyclic randomization for 20 random testcases ensured we would get errors for all possible ALU operation.



```
  task main(input int count);
    g_trans = new();
    g_trans = new custom_trans;
    //custom trans is useful if we wanna assign constraints on
randomization through this custom class
    repeat(count) begin
      assert(g_trans.randomize()) //no semicolon here
        else $fatal("Randomization failed!");

      copy_trans = new();
      copy_trans = new g_trans;

      gen2driv.put(copy_trans);
      //print_mailbox();
    end
  endtask

endclass
```

We will also need to change the generator function to declare g_trans object out of repeat. If we kept it inside repeat, it would create a new instance every time and the cyclic randomization would not be able to keep track. But declaring the object once would make all the values in gen2driv mailbox same. So, we had to do a deep copy of the object and declare a new object called copy_trans in the scope of repeat.

Output:

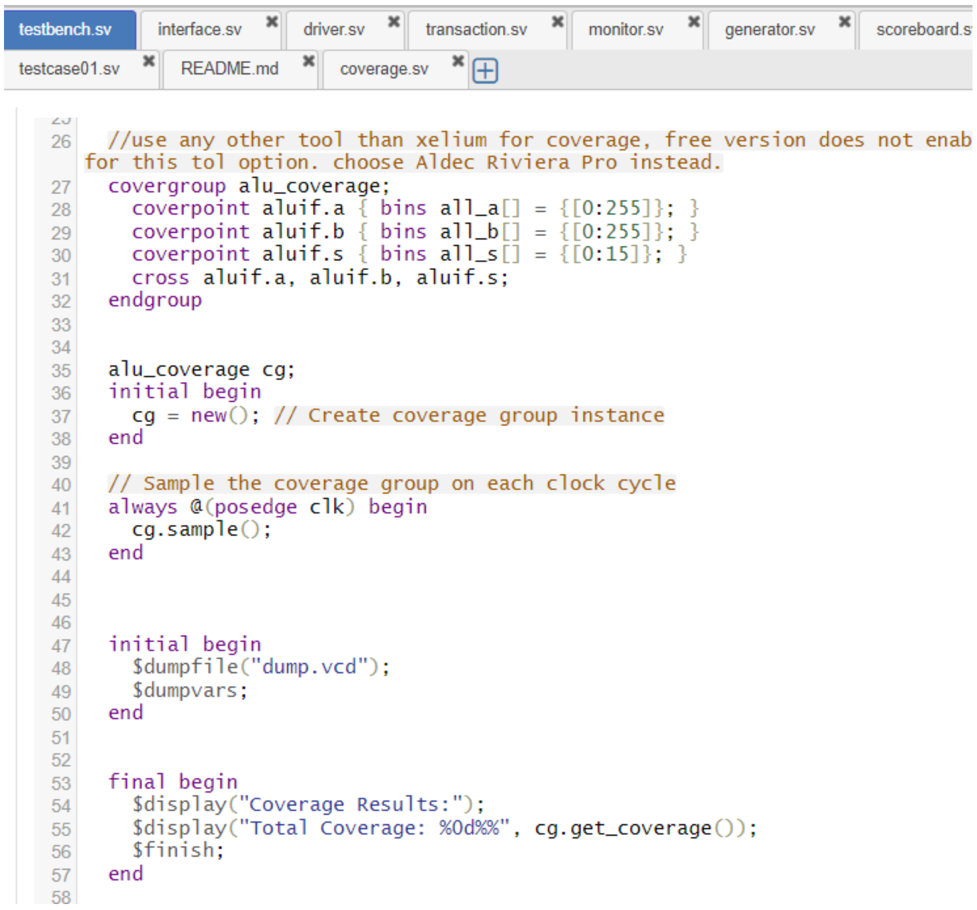
```
---Scoreboard Test Starts-----------
Passed: a=159 b=122 s=12  Expected out=  318  Resulted out=  318
Passed: a=199 b=209 s= 2  Expected out=41591  Resulted out=41591
Passed: a=174 b=160 s= 4  Expected out=   14  Resulted out=   14
Passed: a=  3 b=138 s=14  Expected out=    4  Resulted out=    4
Failed: a=138 b= 20 s=15  Expected out=  137  Resulted out=  139
Passed: a= 22 b= 85 s= 7  Expected out=    0  Resulted out=    0
Passed: a= 99 b=155 s= 0  Expected out=  254  Resulted out=  254
Passed: a=234 b=148 s= 3  Expected out=    1  Resulted out=    1
Passed: a= 20 b= 83 s= 9  Expected out=   16  Resulted out=   16
Passed: a=216 b=124 s= 6  Expected out=    1  Resulted out=    1
Failed: a= 26 b=123 s= 1  Expected out=65439  Resulted out=  123
Passed: a=137 b= 71 s= 5  Expected out=    1  Resulted out=    1
Failed: a=188 b= 96 s= 8  Expected out=65347  Resulted out=  188
Passed: a= 92 b=  1 s=13  Expected out=   46  Resulted out=   46
Passed: a=219 b=137 s=10  Expected out=  219  Resulted out=  219
Passed: a= 99 b= 68 s=11  Expected out=   39  Resulted out=   39
Failed: a=200 b= 87 s=15  Expected out=  199  Resulted out=  201
Passed: a= 76 b=183 s= 2  Expected out=13908  Resulted out=13908
Failed: a=226 b=179 s= 8  Expected out=65309  Resulted out=  226
Failed: a=115 b= 36 s= 1  Expected out=   79  Resulted out=   36
------Scoreboard Test Ends-----------------
```

Report Task2

Report the performance of your testbench and suggest testing strategies that can enhance coverage, reduce verification time and efforts.

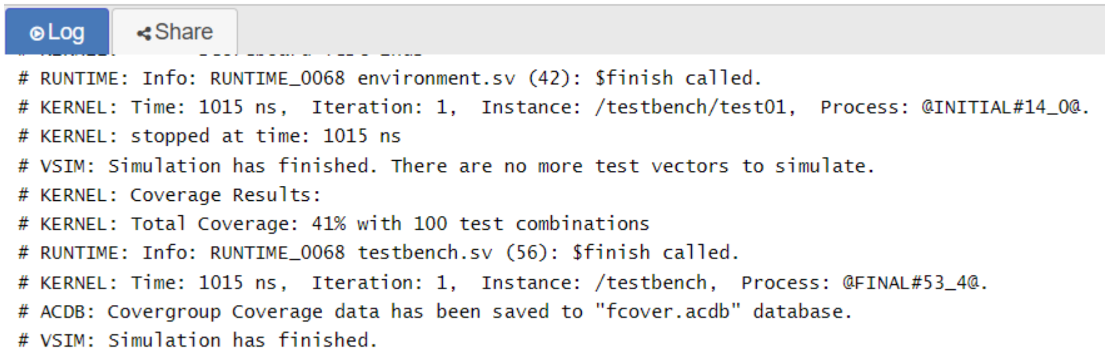
Code changes:

To measure coverage, we modified the testbench to measure from interface signals at posedge of clock. We also used a different tool since the free version of EDA does not allow coverage built-in methods to be used in Xcelium.



```
  //use any other tool than xelium for coverage, free version does not enab
for this tol option. choose Aldec Riviera Pro instead.
  covergroup alu_coverage;
    coverpoint aluif.a { bins all_a[] = {[0:255]}; }
    coverpoint aluif.b { bins all_b[] = {[0:255]}; }
    coverpoint aluif.s { bins all_s[] = {[0:15]}; }
    cross aluif.a, aluif.b, aluif.s;
  endgroup


  alu_coverage cg;
  initial begin
    cg = new(); // Create coverage group instance
  end

  // Sample the coverage group on each clock cycle
  always @(posedge clk) begin
    cg.sample();
  end



  initial begin
    $dumpfile("dump.vcd");
    $dumpvars;
  end


  final begin
    $display("Coverage Results:");
    $display("Total Coverage: %0d%%", cg.get_coverage());
    $finish;
  end
```

Output coverage reports:



```
# RUNTIME: Info: RUNTIME_0068 environment.sv (42): $finish called.
# KERNEL: Time: 1015 ns,  Iteration: 1,  Instance: /testbench/test01,  Process: @INITIAL#14_0@.
# KERNEL: stopped at time: 1015 ns
# VSIM: Simulation has finished. There are no more test vectors to simulate.
# KERNEL: Coverage Results:
# KERNEL: Total Coverage: 41% with 100 test combinations
# RUNTIME: Info: RUNTIME_0068 testbench.sv (56): $finish called.
# KERNEL: Time: 1015 ns,  Iteration: 1,  Instance: /testbench,  Process: @FINAL#53_4@.
# ACDB: Covergroup Coverage data has been saved to "fcover.acdb" database.
# VSIM: Simulation has finished.
```

Here the functional coverage is just 41% with 100 test cases.

Ways to increase functional coverage:

Adding constraint to add more corner cases, i.e max and min values of a and b will help evaluating the performance better, Also, keeping track of a,b and s values to generate unique combinations only will help with the coverage too. We also need to use assertion to ensure only valid and successful cases are included in the functional coverage calculation.