design.sv

```
// Code your design here
module alu (out,a,b,s,clk);
	input clk;
		input [8:0]a,b;
		input [3:0]s;
		output [8:0]out;
		reg [8:0]out;
//,flag;
		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
```

testbench.sv

```
`include "uvm_macros.svh"
import uvm_pkg::*;
`include "alu_if.sv"
`include "alu_sequencer.sv"
`include "alu_monitor.sv"
`include "alu_driver.sv"
`include "alu_agent.sv"
`include "alu_coverage.sv"
`include "alu_scoreboard.sv"
`include "alu_env.sv"
`include "alu_random_test.sv"
`include "alu_addition_test.sv"
//`include "alu_multiplication_test.sv"



module testbench;

  // Set run count
  int count = 25;

  // Generate Clock
  bit clk;
  always #5 clk = ~clk;
```

```
//Interface declaration
alu_if vif(clk);

//DUT instance, interface signals are connected to the DUT ports
alu DUT (
  .a(vif.a),
  .b(vif.b),
  .s(vif.s),
  .out(vif.out),
  .clk(clk)
);



initial begin
        //Registers the Interface in the configuration block so that other blocks can use it
  uvm_resource_db#(virtual alu_if)::set(.scope("*"), .name("alu_if"), .val(vif));

  //Registers the run count in the configuration block so that other blocks can use it
  //uvm_resource_db#(int)::set(.scope("*"), .name("count"), .val(count));
  uvm_config_db#(int)::set(uvm_root::get(), "*", "count", count);

        //Executes the test
  run_test("alu_random_test");
end

initial begin
```

```
   $dumpfile("dump.vcd");
   $dumpvars;
  end

endmodule
```

alu_agent.sv

```
class alu_agent extends uvm_agent;
  `uvm_component_utils(alu_agent)

  // Analysis Port
  uvm_analysis_port#(alu_transaction) agent_ap;

  // Component Instances
  alu_sequencer                alu_seq;
  alu_driver                   alu_drv;
  alu_monitor                 alu_mon;



/*****************************************************/
/*                new - Constructor                  */
/*****************************************************/
  function new(string name, uvm_component parent);
            super.new(name, parent);
  endfunction: new

/*****************************************************/
/*                 build_phase                       */
```

```
/*****************************************************/
  function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            agent_ap   = new(.name("agent_ap"), .parent(this));
    if(is_active == UVM_ACTIVE) begin
      alu_seq  = alu_sequencer::type_id::create(.name("alu_seq"), .parent(this));
      alu_drv  = alu_driver::type_id::create(.name("alu_drv"), .parent(this));
    end
    alu_mon  = alu_monitor::type_id::create(.name("alu_mon"), .parent(this));
  endfunction: build_phase


/*****************************************************/
/*                connect_phase                  */
/*****************************************************/
  function void connect_phase(uvm_phase phase);
            super.connect_phase(phase);
    if(is_active == UVM_ACTIVE) begin
      alu_drv.seq_item_port.connect(alu_seq.seq_item_export);
    end
            alu_mon.mon_ap.connect(agent_ap);
  endfunction: connect_phase

endclass: alu_agent
```

alu_assertions.sv

```
/*****************************************************/
/*                Assertions                  */
```

```
/*****************************************************/
 property addition;
   logic [8:0] op_a, op_b;
   @(posedge clk) (s == 0, op_a = a, op_b = b) |=>  (out == (op_a+op_b));
 endproperty
 assert property (addition);
 //else $display("Failed");
// cover property (addition);

 property subtraction;
   logic [8:0] op_a, op_b;
   @(posedge clk) (s == 1, op_a = a, op_b = b) |=>  out == (op_a - op_b);
 endproperty
 assert property (subtraction);

 property multiplication;
   logic [8:0] op_a, op_b;
   @(posedge clk) (s == 2, op_a = a, op_b = b) |=>  out == (op_a * op_b);
 endproperty
 assert property (multiplication);

 property division;
   logic [8:0] op_a, op_b;
   @(posedge clk) (s == 3, op_a = a, op_b = b) |=>  out == (op_a / op_b);
 endproperty
 assert property (division);
```

```
property modulo_division;
  logic [8:0] op_a, op_b;
  @(posedge clk) (s == 4, op_a = a, op_b = b) |=>  out == (op_a % op_b);
endproperty
assert property (modulo_division);

property logical_and;
  logic [8:0] op_a, op_b;
  @(posedge clk) (s == 5, op_a = a, op_b = b) |=> (out == (op_a && op_b));
endproperty
assert property (logical_and);
//else $display("a=%d, b=%d, out=%d",a,b,out);

property logical_or;
  logic [8:0] op_a, op_b;
  @(posedge clk) (s == 6, op_a = a, op_b = b) |=>  out == (op_a || op_b);
endproperty
assert property (logical_or);

property logical_negation;
  logic [8:0] op_a, op_b;
  @(posedge clk) (s == 7, op_a = a, op_b = b) |=>  out == !op_a;
endproperty
assert property (logical_negation);

property bitwise_negation;
  logic [8:0] op_a, op_b;
```

```
  @(posedge clk) (s == 8, op_a = a, op_b = b) |=>  out == ~op_a;
endproperty
assert property (bitwise_negation);

property bitwise_and;
  logic [8:0] op_a, op_b;
  @(posedge clk) (s == 9, op_a = a, op_b = b) |=>  out == (op_a & op_b);
endproperty
assert property (bitwise_and);

property bitwise_or;
  logic [8:0] op_a, op_b;
  @(posedge clk) (s == 10, op_a = a, op_b = b) |=>  out == (op_a | op_b);
endproperty
assert property (bitwise_or);

property bitwise_xor;
  logic [8:0] op_a, op_b;
  @(posedge clk) (s == 11, op_a = a, op_b = b) |=>  out == (op_a ^ op_b);
endproperty
assert property (bitwise_xor);

property left_shift;
  logic [8:0] op_a, op_b;
  @(posedge clk) (s == 12, op_a = a, op_b = b) |=>  out == op_a << 1;
endproperty
assert property (left_shift);
```

```
 property right_shift;
   logic [8:0] op_a, op_b;
   @(posedge clk) (s == 13, op_a = a, op_b = b) |=>  out == op_a >> 1;
 endproperty
 assert property (right_shift);

 property increment;
   logic [8:0] op_a, op_b;
   @(posedge clk) (s == 14, op_a = a, op_b = b) |=>  out == op_a + 1;
 endproperty
 assert property (increment);

 property decrement;
   logic [8:0] op_a, op_b;
   @(posedge clk) (s == 15, op_a = a, op_b = b) |=>  out == op_a - 1;
 endproperty
 assert property (decrement);
```

alu_converge.sv

```
class alu_coverage extends uvm_subscriber#(alu_transaction);
  `uvm_component_utils(alu_coverage)

  alu_transaction  alu_tx_cg;

/*********************************************************/
/*                Covergroup                 */
/*********************************************************/
```

```
covergroup alu_cg;
  option.per_instance = 1;
  selector: coverpoint alu_tx_cg.s {
    bins addition        = {0};
    bins subtraction     = {1};
    bins multiplication  = {2};
    bins division        = {3};
    bins modulo_division = {4};
    bins logical_and     = {5};
    bins logical_or      = {6};
    bins logical_negation = {7};
    bins bitwise_negation = {8};
    bins bitwise_and     = {9};
    bins bitwise_or      = {10};
    bins bitwise_xor     = {11};
    bins left_shift      = {12};
    bins right_shift     = {13};
    bins increment       = {14};
    bins decrement       = {15};
    option.at_least = 1;
  }
  op_a: coverpoint alu_tx_cg.a {
    bins zero_to_255 = {[0:255]};
  }
  cross_a_s: cross op_a,selector;
endgroup: alu_cg
```

```
/*****************************************************/
/*                  Assertions                       */
/*****************************************************/
/*  property addition;
   alu_tx_cg.s == 0 |=> alu_tx_cg.out == alu_tx_cg.a+alu_tx_cg.b;
 endproperty
 assert property (addition);
 cover property (addition);  */




/*****************************************************/
/*                new - Constructor                  */
/*****************************************************/
 function new(string name, uvm_component parent);
   super.new(name, parent);
   alu_cg = new();
 endfunction: new

/*****************************************************/
/*                function - write                   */
/*****************************************************/
 function void write(alu_transaction t);
   alu_tx_cg = t;
   alu_cg.sample();
 endfunction:write
```

endclass : alu_coverage

alu_driver.sv

```
class alu_driver extends uvm_driver#(alu_transaction);
  `uvm_component_utils(alu_driver)

  // Virtual Interface
  virtual alu_if vif;

/*****************************************************/
/*              new - Constructor                    */
/*****************************************************/
  function new(string name, uvm_component parent);
          super.new(name, parent);
  endfunction: new

/*****************************************************/
/*              build_phase                          */
/*****************************************************/
  function void build_phase(uvm_phase phase);
          super.build_phase(phase);

    void'(uvm_resource_db#(virtual alu_if)::read_by_name(.scope("*"), .name("alu_if"), .val(vif)));
  endfunction: build_phase

/*****************************************************/
```

```
/*                run_phase                    */
/*************************************************************/
 task run_phase(uvm_phase phase);
          drive();
 endtask: run_phase

/*************************************************************/
/*                task - drive()                 */
/*************************************************************/
 virtual task drive();
          alu_transaction alu_tx;
   //vif.DRIVER.driver_cb.a <= 0;
   //vif.DRIVER.driver_cb.b <= 0;
   //vif.DRIVER.driver_cb.s <= 0;

          forever begin
    seq_item_port.get_next_item(alu_tx);

    // Print the received transaction
    //`uvm_info("alu_driver", alu_tx.sprint(), UVM_LOW);

    @(vif.driver_cb);
    vif.DRIVER.driver_cb.a <= alu_tx.a;
    vif.DRIVER.driver_cb.b <= alu_tx.b;
    vif.DRIVER.driver_cb.s <= alu_tx.s;

    seq_item_port.item_done();
```

```
            end
  endtask: drive


endclass:alu_driver
```

alu_env.sv

```
class alu_env extends uvm_env;


  // Component Instances
  alu_agent       agents[];
  alu_scoreboard  scb;
  alu_coverage    cov;
  int             num_agents = 2;
  string          inst_name;


  `uvm_component_utils_begin(alu_env)
    `uvm_field_int(num_agents, UVM_ALL_ON)
  `uvm_component_utils_end


/*****************************************************/
/*                new - Constructor                  */
/*****************************************************/
  function new(string name, uvm_component parent);
            super.new(name, parent);
  endfunction: new


/*****************************************************/
```

```
/*                build_phase                */
/*****************************************************/
  virtual function void build_phase(uvm_phase phase);
          super.build_phase(phase);
    if(num_agents == 0)
      `uvm_fatal("NONUM","'num_agents' must be set");
    agents = new[num_agents];
    for(int i=0; i < num_agents; i++) begin
      $sformat(inst_name, "agents[%0d]", i);
      agents[i] = alu_agent::type_id::create(.name(inst_name), .parent(this));
    end
    scb      = alu_scoreboard::type_id::create(.name("scb"), .parent(this));
    cov   = alu_coverage::type_id::create(.name("cov"), .parent(this));

    // Set agents ACTIVE/PASSIVE
    uvm_config_db#(int)::set(uvm_root::get(),"*.agents[0]","is_active", UVM_ACTIVE);
    uvm_config_db#(int)::set(uvm_root::get(),"*.agents[1]","is_active", UVM_PASSIVE);
  endfunction: build_phase

/*****************************************************/
/*                connect_phase                */
/*****************************************************/
  function void connect_phase(uvm_phase phase);
          super.connect_phase(phase);
    agents[0].agent_ap.connect(scb.sb_export_before);
    agents[1].agent_ap.connect(scb.sb_export_after);
    agents[0].agent_ap.connect(cov.analysis_export);
```

```
  endfunction: connect_phase


endclass: alu_env
```

alu_if.sv

```
interface alu_if(input clk);
  logic [8:0] a,b;
  logic [3:0] s;
  logic [8:0] out;


  clocking driver_cb @(posedge clk);
    default input #1 output #1;
    output a,b,s;
  endclocking


  clocking mon_cb @(posedge clk);
    default input #1 output #2;
    input a,b,s;
    input out;
  endclocking


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


  `include "alu_assertions.sv"


endinterface
```

alu_monitor.sv

```
class alu_monitor extends uvm_monitor;
  `uvm_component_utils(alu_monitor)

  // Analysis Port
  uvm_analysis_port#(alu_transaction) mon_ap;

  // Virtual Interface
  virtual alu_if vif;

  alu_transaction alu_tx;

/*****************************************************/
/*                new - Constructor                  */
/*****************************************************/
  function new(string name, uvm_component parent);
              super.new(name, parent);
  endfunction: new

/*****************************************************/
/*                  build_phase                      */
/*****************************************************/
  function void build_phase(uvm_phase phase);
            super.build_phase(phase);

    void'(uvm_resource_db#(virtual alu_if)::read_by_name(.scope("*"), .name("alu_if"), .val(vif)));
    mon_ap = new(.name("mon_ap"), .parent(this));
  endfunction: build_phase
```

```
/*****************************************************/
/*                run_phase                  */
/*****************************************************/
 task run_phase(uvm_phase phase);
        alu_tx = alu_transaction::type_id::create (.name("alu_tx"), .contxt(get_full_name()));

        forever begin
    @(vif.mon_cb);
           alu_tx.a   = vif.MONITOR.mon_cb.a;
           alu_tx.b   = vif.MONITOR.mon_cb.b;
    alu_tx.s   = vif.MONITOR.mon_cb.s;
    alu_tx.out = vif.MONITOR.mon_cb.out;

    // Print the sampled inputs
    //`uvm_info("alu_monitor", alu_tx.sprint(), UVM_LOW);

    //Send the transaction to the analysis port
    mon_ap.write(alu_tx);
        end
 endtask: run_phase
endclass: alu_monitor
```

alu_scoreboard.sv

```
//`uvm_analysis_imp_decl(_before)
//`uvm_analysis_imp_decl(_after)

class alu_scoreboard extends uvm_scoreboard;
```

```
 `uvm_component_utils(alu_scoreboard)

 // Analysis Port
 uvm_analysis_export #(alu_transaction) sb_export_before;
 uvm_analysis_export #(alu_transaction) sb_export_after;

 // Analysis FIFO
 uvm_tlm_analysis_fifo #(alu_transaction) before_fifo;
 uvm_tlm_analysis_fifo #(alu_transaction) after_fifo;

 alu_transaction  transaction_before;
 alu_transaction  transaction_after;



/*****************************************************/
/*              new - Constructor                    */
/*****************************************************/
 function new(string name, uvm_component parent);
         super.new(name, parent);

         transaction_before       = new("transaction_before");
         transaction_after        = new("transaction_after");
 endfunction: new

/*****************************************************/
/*              build_phase                          */
/*****************************************************/
```

```
 function void build_phase(uvm_phase phase);
        super.build_phase(phase);

        sb_export_before        = new("sb_export_before", this);
        sb_export_after         = new("sb_export_after", this);

        before_fifo             = new("before_fifo", this);
        after_fifo              = new("after_fifo", this);
 endfunction: build_phase

/*****************************************************/
/*              connect_phase                */
/*****************************************************/
 function void connect_phase(uvm_phase phase);
        sb_export_before.connect(before_fifo.analysis_export);
        sb_export_after.connect(after_fifo.analysis_export);
 endfunction: connect_phase

/*****************************************************/
/*              task - run                   */
/*****************************************************/
 task run();
        forever begin
          before_fifo.get(transaction_before);
          after_fifo.get(transaction_after);

    //predictor();
```

```
            compare();
    predictor();
          end
 endtask: run

/*************************************************************/
/*              function - Compare                 */
/*************************************************************/
          virtual function void compare();
                    if(transaction_before.out == transaction_after.out) begin
       `uvm_info(get_type_name(),$sformatf("a = %0d, b = %0d, s = %0d, Expected = %0d, Resulted = %0d", transaction_before.a, transaction_before.b, transaction_before.s, transaction_before.out, transaction_after.out), UVM_LOW);
       `uvm_info("PASSED", {"Test: PASSED "}, UVM_LOW);
                    end else begin
       `uvm_info(get_type_name(),$sformatf("a = %0d, b = %0d, s = %0d, Expected = %0d, Resulted = %0d", transaction_before.a, transaction_before.b, transaction_before.s, transaction_before.out, transaction_after.out), UVM_LOW);
       `uvm_info("FAILED", {"Test: FAILED "}, UVM_LOW);
                    end
          endfunction: compare


/*************************************************************/
/*              function - predictor()              */
/*************************************************************/
          virtual function void predictor();
```

```
    if     (transaction_before.s==0) transaction_before.out=transaction_before.a +
transaction_before.b;

    else if(transaction_before.s==1) transaction_before.out=transaction_before.a -
transaction_before.b;

    else if(transaction_before.s==2) transaction_before.out=transaction_before.a *
transaction_before.b;

    else if(transaction_before.s==3) transaction_before.out=transaction_before.a /
transaction_before.b;

    else if(transaction_before.s==4) transaction_before.out=transaction_before.a %
transaction_before.b;

    else if(transaction_before.s==5) transaction_before.out=transaction_before.a &&
transaction_before.b;

    else if(transaction_before.s==6) transaction_before.out=transaction_before.a ||
transaction_before.b;

    else if(transaction_before.s==7) transaction_before.out=!transaction_before.a;

    else if(transaction_before.s==8) transaction_before.out=~transaction_before.a;

    else if(transaction_before.s==9) transaction_before.out=transaction_before.a &
transaction_before.b;

    else if(transaction_before.s==10) transaction_before.out=transaction_before.a |
transaction_before.b;

    else if(transaction_before.s==11) transaction_before.out=transaction_before.a ^
transaction_before.b;

    else if(transaction_before.s==12) transaction_before.out=transaction_before.a << 1;

    else if(transaction_before.s==13) transaction_before.out=transaction_before.a >> 1;

    else if(transaction_before.s==14) transaction_before.out=transaction_before.a + 1;

    else if(transaction_before.s==15) transaction_before.out=transaction_before.a - 1;

        endfunction: predictor


endclass: alu_scoreboard
```

alu_sequence.sv

```
class alu_sequence extends uvm_sequence#(alu_transaction);
```

```
 `uvm_object_utils(alu_sequence)

 // Set run count
 int count;

 // Transaction class handle
 rand alu_transaction alu_tx;



/*****************************************************/
/*              new - Constructor                    */
/*****************************************************/
 function new(string name = "alu_sequence");
          super.new(name);
 endfunction: new

/*****************************************************/
/*              task - body()                        */
/*****************************************************/
 task body();
   // Get the count value from testbench top
   //void'(uvm_resource_db#(int)::read_by_name(.scope(""), .name("count"), .val(count)));
   if(!uvm_config_db#(int)::get(uvm_root::get(), "*", "count", count))
     `uvm_fatal("NOCOUNT", {"count must be set for: ", get_full_name(),".count"});

   alu_tx = alu_transaction::type_id::create(.name("alu_tx"), .contxt(get_full_name()));
   repeat(count) begin
```

```
    //alu_tx = alu_transaction::type_id::create(.name("alu_tx"), .contxt(get_full_name()));
    start_item(alu_tx);
    assert(this.randomize(alu_tx));
    //`uvm_info("alu_sequence", alu_tx.sprint(), UVM_LOW);
    finish_item(alu_tx);
          end
  endtask: body
endclass: alu_sequence
```

alu_sequencer.sv

```
`include "alu_transaction.sv"
`include "alu_sequence.sv"

typedef uvm_sequencer#(alu_transaction) alu_sequencer;
```

alu_transaction.sv

```
class alu_transaction extends uvm_sequence_item;
  rand bit [8:0] a;
  rand bit [8:0] b;
  randc bit [3:0] s;
  bit [8:0] out;



/*************************************************************/
/*               new - Constructor                 */
/*************************************************************/
  function new(string name = "alu_transaction");
          super.new(name);
  endfunction: new
```

```
/*****************************************************/
/*              Utility and Field macros             */
/*****************************************************/
  `uvm_object_utils_begin(alu_transaction)
    `uvm_field_int(a,UVM_ALL_ON)
    `uvm_field_int(b,UVM_ALL_ON)
    `uvm_field_int(s,UVM_ALL_ON)
    `uvm_field_int(out,UVM_ALL_ON)
  `uvm_object_utils_end

/*  constraint inputs {
    a inside {[1:10]};
    b inside {[1:10]};
    s == 0;
  }
*/

endclass: alu_transaction
```

alu_addition_test.sv

```
class addition_constraint extends alu_sequence;
  `uvm_object_utils(addition_constraint)

/*****************************************************/
/*                new - Constructor                  */
/*****************************************************/
  function new(string name = "addition_constraint");
```

```
            super.new(name);
  endfunction: new

/*****************************************************/
/*                  Constraints                      */
/*****************************************************/
    constraint inputs {
    alu_tx.a inside {[1:10]};
    alu_tx.b inside {[1:10]};
    alu_tx.s == 0;   // Addition
  }

endclass: addition_constraint



//-------------------------------------------------------------------------------//



class alu_addition_test extends alu_random_test;
  `uvm_component_utils(alu_addition_test)

/*****************************************************/
/*                new - Constructor                  */
/*****************************************************/
  function new(string name = "alu_addition_test", uvm_component parent = null);
    super.new(name, parent);
  endfunction: new
```

```
/*****************************************************/
/*           start_of_simulation_phase               */
/*****************************************************/
 function void start_of_simulation_phase(uvm_phase phase);
   set_type_override_by_type(alu_sequence::get_type(), addition_constraint::get_type());
 endfunction

endclass : alu_addition_test
```

alu_multiplication_test.sv

```
class multiplication_constraint extends alu_sequence;
 `uvm_object_utils(multiplication_constraint)

/*****************************************************/
/*               new - Constructor                   */
/*****************************************************/
 function new(string name = "multiplication_constraint");
          super.new(name);
 endfunction: new

/*****************************************************/
/*                 Constraints                       */
/*****************************************************/
   constraint inputs {
   alu_tx.a inside {[1:10]};
   alu_tx.b inside {[1:10]};
```

```
   alu_tx.s == 2;   // Multiplication
  }

endclass: multiplication_constraint



//-------------------------------------------------------------------------------//



class alu_multiplication_test extends alu_random_test;
  `uvm_component_utils(alu_multiplication_test)

/*************************************************************/
/*                new - Constructor                  */
/*************************************************************/
  function new(string name = "alu_multiplication_test", uvm_component parent = null);
    super.new(name, parent);
  endfunction: new



/*************************************************************/
/*            start_of_simulation_phase              */
/*************************************************************/
  function void start_of_simulation_phase(uvm_phase phase);
    set_type_override_by_type(alu_sequence::get_type(), multiplication_constraint::get_type());
  endfunction
```

```
endclass : alu_multiplication_test
```

alu_random_test.sv

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

  // Component Instances
  alu_env env;

/*************************************************************/
/*                new - Constructor                  */
/*************************************************************/
  function new(string name, uvm_component parent);
            super.new(name, parent);
  endfunction: new

/*************************************************************/
/*                  build_phase                      */
/*************************************************************/
  function void build_phase(uvm_phase phase);
            super.build_phase(phase);
            env = alu_env::type_id::create(.name("env"), .parent(this));
  endfunction: build_phase

/*************************************************************/
/*                  run_phase                        */
/*************************************************************/
  task run_phase(uvm_phase phase);
```

```
        alu_sequence seq;


        phase.raise_objection(.obj(this));
          seq = alu_sequence::type_id::create(.name("seq"), .contxt(get_full_name()));
    //assert(seq.randomize());
    seq.start(env.agents[0].alu_seq);
        phase.drop_objection(.obj(this));
 endtask: run_phase


endclass: alu_random_test
```