

Verification Tutorial of a floating point MAC

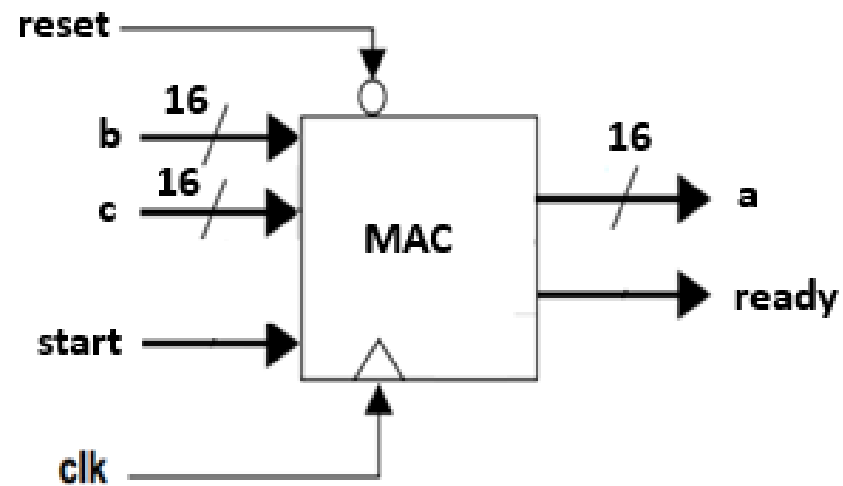
Frank Plasencia Balabarca

Advisors:

MSc Mario Raffo MSc Edward Mitacc

Signal Layer

Input/output port distribution
of the AES



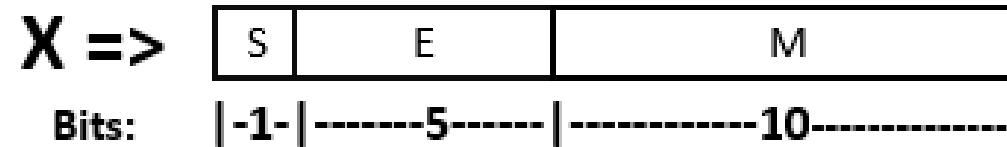
Signal Layer

General DUV behavior

Operation

$$a = b * c + a$$

Bit distribution



S: sign

E: exponent

M: mantissa

$$X = (-1)^S * (M) * e^{E-127}$$

Communication interface

Direct Interconnection

The MAC is connected directly to the verification framework with no need to employ a communication interface since this chip is relatively simple

Stimuli sequences

Working modes

As it was determined in the signal layer stage, there is only one working mode for the MAC module which involves a multiplication followed by a sum.

Stimuli sequences

Corner cases

Overflow

Multiplication of two large numbers

Underflow

Multiplication of two small numbers

Zeros

Multiplication of number with zero or two
zeros

Validation mechanism

SystemVerilog task

The high level behavior of the design is simple and it is equivalent to a single equation so it can easily be represented in a task.

Coverage metrics

Code coverage

Type	Description	Expected ratio (%)
Expression	How many statements were covered	100
Block	How many blocks have been covered	95
Toggle	How many times signals and ports are toggled during simulation	95
FSM	Indicates whether the FSMs in the design reach all possible states or not	95

Functional coverage

Requirement code	Description	Validation type	Priority
start_i_val	After the posedge of start, it must not occur a new posedge of start_i, before the DUV process completeness	Property	1
a_i_val	If start is High, input data must only be 1s or 0s, No Xs or Zs	Property	1
b_i_val	If start is High, key_i must only be 1s or 0s, No Xs or Zs	Property	1
c_o_val	If ready is High, output data must only be 1s or 0s, No Xs or Zs	Property	1
a_i_stb	a must remain stable after the posedge of start, until the negedge of start	Property	1
b_i_stb	b must remain stable after the posedge of start, until the negedge of start	Property	1
c_o_stb	c must remain stable after the negedge of ready, until the posedge of start	Property	1
ready_o_tm	ready must only be High during 1 clock cycle	Property	1
start_i_tm	start must be in High just during 1 clock cycle	Property	1
id_cover	The static variable id, must cover every integer value between 1 and the transaction number defined in the top	Cover-point	2
case_cover	case_stim should cover all cases at least 100 times	Cover-point	2
a_i_cover	a must cover at least 100 value in every sub-range of the 32 sub-ranges	Cover-point	2
b_i_cover	b must cover at least 100 value in every sub-range of the 32 sub-ranges	Cover-point	1
time_out_tm	The time limit between a posedge of start and a posedge of ready is 500 clock cycle	Property	2

Transaction

```
class Transaction;
  //identifier of the transaction
  static int id_counter = 1;
  int id;
  //random stimulus
  randc logic [15:0] b;
  randc logic [15:0] c;
  //Result variables
  logic [15:0] a;
  real a_sc;
  //auxiliar variables
  real a_flo, b_flo, c_flo;
  randc int Sel;
  //Declaring the cases of interest
  rand case_type n_case;

  //Constraints
  constraint cases
  {n_case dist {Normal:=90, Underflow:=10, Overflow:=10,
Zero:=15};} //Defining probabilistic distribution
  constraint n_Sel {Sel>0; Sel<=21;}
```

Cyclic random variables

Probabilistic distribution

Transaction

```
//task used to obtain a value in decimal radix from a value
//in the format that the MAC circuit uses
task convert(input logic [15:0] x, output shortreal x_f);
    int i;
    shortreal x_flo,exp;
    logic [3:0] aux;

    x_flo = 0.0; exp = 0.0; // initial values
    if (x[14:0] == '0) x_f = x_flo; // case of + or - zero
    else begin
        // Processing the mantissa
        for (i=0;i<10;i++)
            x_flo = (x[14-i]==1'b1)? (x_flo + 1.0/(2.0**(i))):x_flo;
        // Processing the exponent
        if (x[4]=='0) // case: positive exponent
            for (i=0;i<4;i++)
                exp = (x[i]==1'b1)? (exp + (2.0**(i))):exp;
        else begin //case: negative exponent
            aux = (~(x[4:0]) + 5'b00001);
            for (i=0;i<5;i++)
                exp = (x[i]==1'b1)? (exp + (2.0**(i))):exp;
            exp = (-1.0)*exp;
        end
        //evaluating mantissa, exponent and sign
        x_flo = ((-1)**(x[15]))*x_flo*(2.0**(exp));
        x_f = x_flo;
    end
endtask : convert
```



Auxiliary task



Transaction

```
//constructor of the transaction
function new;
    id = id_counter; //getting the id
    id_counter++;    //incrementing for the next id
endfunction : new

//function used to get the real representation of b and c
task get_FP;
    convert(this.b,this.b_flo);
    convert(this.c,this.c_flo);
endtask : get_FP

//task used to display data of the stimuli
task display_n;
    get_FP;
    $display("-----");
    $display("----- TRANSACTION %d -----",this.id);
    $display("----- Case = %s -----", this.n_case.name);
    $display("----- B = %8f -----",this.b_flo);
    $display("----- C = %8f -----",this.c_flo);
    $display("-----");
endtask : display_n
```

```
//task to display the results of the stimulation
task display_r;
    convert(this.a,this.a_flo);
    $display("-----");
    $display("----- RESULTS %d -----",this.id);
    $display("----- DUT ----- A    = %8f -----",this.a_flo);
    $display("----- SCB ----- A    = %8f -----",this.a_sc);
    $display("-----");
endtask : display_r
endclass : Transaction
```

Interface

```
interface bus_itf (input logic clock);
    logic      start, ready;
    case_type  n_case;
    logic [15:0] a, b, c;

    //Defining the timing control with the dut
    clocking cb @(posedge clock);
        input  a, ready;
        output b, c, start, n_case;
    endclocking : cb

    modport DUT (output a, ready, input start, b, c);
    modport TB  (clocking cb);

endinterface : bus_itf
```

Control by the testbench

Generator

```
class Generator;
    Transaction r; // instance of the transaction
    mailbox #(Transaction) mbG2A; // mailbox to communicate with the
Agent
    event DoneG2A; // event to synchronize Generator and Agent

    //Constructor of the Generator class
    function new(input mailbox #(Transaction) mbG2A,input event
DoneG2A);
        this.mbG2A      = mbG2A; // Giving direction the mailbox object
        this.DoneG2A    = DoneG2A; // Giving direction to the event object
with the Agent
    endfunction : new

    //Principal task
    task run (input int n_times);
        repeat (n_times) begin //Repeating according to the number of
tests
            r = new; // creating the object of the new transaction
            if (r.randomize) begin // randomizing the stimuli
                $display("at:%t: Value Successfully generated!:",$time);
                mbG2A.put(r);
            end
            else begin
                $display("Randomization failed!");
                $finish;
            end
            @this.DoneG2A; // waiting for the Agent to respond
        end
    endtask
endclass
```



25 años

Verification framework

Agent



```
class Agent;
mailbox #(Transaction) mbG2A,mbA2D;// mailbox to communicate with Driver and Agent
Transaction r; // instance of the transaction
event DoneG2A; // event to synchronize with the Agent
event DoneA2D; // event to synchronize with the Driver
Callback cbs[$]; // Q of callbacks

//constructor of the Agent class
function new(input mailbox #(Transaction) mbG2A, mbA2D, input event DoneG2A, DoneA2D);
    this.mbG2A = mbG2A; // Giving direction to the mailbox object with the Generator
    this.mbA2D = mbA2D; // Giving direction to the mailbox object with the Driver
    this.DoneG2A = DoneG2A; // Giving direction to the event object with the Generator
    this.DoneA2D = DoneA2D; // Giving direction to the event object with the Driver
endfunction : new

//task used to get the appropriate values of b and c according to the case
//It is also evaluated the insertion of de-normalized numbers using Sel random variable
task select;
    logic [15:0] b,c;
    case (this.r.n_case)
        2'b00: begin // Normal
            if (this.r.Sel%2 == '0)
                begin
                    b = (this.r.Sel%3 == '0)? {this.r.b[15:5],5'b00000}:this.r.b;//Subnormal number
                    c = this.r.c;
                end
            else
                begin
                    b = this.r.b;
                    c = (this.r.Sel%3 == '0)? {this.r.c[15:5],5'b00000}:this.r.c;//Subnormal number
                end
            end
        2'b01: // Underflow
            begin
                b = (this.r.Sel%2 == '0)? {this.r.b[15:5],5'b10000} : this.r.b;
```

SVVM

Agent

```

        c = (this.r.Sel%2 == '0')? this.r.c : {this.r.c[15:5],5'b10000};
    end
2'b10: // Overflow
    begin
        b = (this.r.Sel%2 == '1')? {this.r.b[15:5],5'b01111} : this.r.b;
        c = (this.r.Sel%2 == '1')? this.r.c : {this.r.c[15:5],5'b01111};
    end
2'b11: // Zero
    begin
        b = (this.r.Sel%2 == '1')? '0 : this.r.b;
        c = (this.r.Sel%2 == '1')? this.r.c : '0;
    end
default: // in other cases
    begin
        $display("An Error Encountered!");
        $finish;
    end
endcase
this.r.b = b; // put the value of "b" in the transaction
this.r.c = c; // put the value of "c" in the transaction
endtask : select

//Principal function
task run(input int n_times);
    repeat (n_times) begin //Repeating according to the number of tests
        mbG2A.get(r); // getting the transaction from the mailbox with the Generator
        select; // Selecting the appropriate value of B and C
        foreach(cbs[i]) cbs[i].pre_TX_RX(r); //Pre callback method
        mbA2D.put(r); // Putting the transaction in the mailbox with the Driver
        foreach(cbs[i]) cbs[i].pos_TX_RX(r); //Pos callback method
        @this.DoneA2D; // Waiting for Driver to respond
        ->this.DoneG2A; // Triggering event to the Agent
    end
endtask : run
endclass : Agent

```




25 años



SV/verilog testbench

```
class Monitor;
  mailbox #(Transaction) mbM2C; // mailbox to communicate with the Checker
  vbus_tb bus_tb; // Instance to the interface with the dut
  event ReadyM2D; // event to synchronize with the Driver
  event DoneM2C; // event to synchronize with the Checker
  int tr_counter; // counter of Transactions
  Transaction r_q[$]; // Q of Transaction

  //function to the handler to the current Transaction
  function void save(input Transaction r);
    r_q.push_back(r); // Saving the handler in the Q of Transactions
  endfunction : save

  //Constructor of the Monitor Class
  function new(input mailbox #(Transaction) mbM2C, input vbus_tb bus_tb, input event ReadyM2D, DoneM2C);
    this.mbM2C = mbM2C; // Giving direction to the mailbox object with the Checker
    this.bus_tb = bus_tb; // Giving direction to interface with the dut
    this.ReadyM2D = ReadyM2D; // Giving direction to the event object with the Driver
    this.DoneM2C = DoneM2C; // Giving direction to the event object with the Checker
  endfunction : new

  //task to send the handler to the current Transaction to the Checker
  task send(input int tr_counter);
    Transaction r[$]; //Declaring a Q of Transaction
    r = this.r_q.find(x) with (x.id == tr_counter); // Searching the Transaction according to the id of
    the Transaction
    case (r.size())
      0: begin
        $display("No Match Found for Transaction N° %d",tr_counter);
        $finish;
      end
      1: begin
        r[0].a = bus_tb.cb.a; // storing the output value of the dut in the Transaction
        this.mbM2C.put(r[0]); // putting the Transaction in the Mailbox to the Checker
      end
    end
  end
```

Monitor

```
        default: begin
            $display("Error! Multiple Matches encountered!");
            $finish;
        end
    endcase
endtask : send

//Principal Task
task run(input int n_times);
    tr_counter = 1; // initializing counter of Transactions
    repeat(n_times) begin //Repeating according to the number of tests
        wait(bus_tb.cb.ready); //Waiting for the dut to finish execution
        send(tr_counter); // sending the value to the Checker via Mailbox
        fork
            wait(!bus_tb.cb.ready); // Waiting for the dut to deassert Ready signal
            @DoneM2C; // Waiting for the Checker to respond
        join
        ->ReadyM2D; // Triggering the event for the Driver
        tr_counter++; // incrementing the count of the Transactions
    end
endtask : run
endclass : Monitor
```



25 años

Verification framework

Checker



SV/Verilog T

```
class Checker;
    mailbox #(Transaction) mbM2C; // mailbox to communicate with the Monitor
    Transaction r; // Instance of the Transaction
    event DoneM2C; // event to synchronize with the Monitor
    event DoneC2S; // event to synchronize with the Scoreboard
    static int error_count = 0; // Defining count of errors
    static int tr_counter = 0; // Defining count of Transactions
    real prtg; // Percentage of reliability

    //Constructor of the Checker class
    function new(input mailbox #(Transaction) mbM2C, input event DoneM2C, DoneC2S, input real prtg);
        this.mbM2C = mbM2C; // Giving direction to the mailbox object with the Monitor
        this.DoneM2C = DoneM2C; // Giving direction to the event object with the Monitor
        this.DoneC2S = DoneC2S; // Giving direction to the event object with the Scoreboard
        this.prtg = prtg; // Matching the reliability percentage
    endfunction : new

    // task used to count the errors during Simulation
    task error_ctr;
        real a_dut,a_scb;
        //Taking the magnitude of both values
        a_dut = (this.r.a_flo < 0)? (-1)*this.r.a_flo: this.r.a_flo;
        a_scb = (this.r.a_sc < 0)? (-1)*this.r.a_sc: this.r.a_sc;
        if (a_scb - a_dut > ((this.prtg*a_scb) / 100)) // applying percentage
            begin
                error_count++;
                $display("-----");
                $display("----- In TRANSACTION N* %d : An Error Encountered!----",this.r.id);
                $display("-----");
            end
        $display("\n");
    endtask : error_ctr
```



SV/Verilog Testbe

```
//Principal task
task run(input int n_times);
  repeat(n_times) begin //Repeating according to the number of tests
    mbM2C.get(r); //Getting the Transaction from the mailbox with the Monitor
    fork
      ->DoneM2C; // Triggering event to the Monitor
      ->DoneC2S; // Triggering event to the Scoreboard
    join
    r.display_r; // printing results;
    error_ctr; // Check if there is an error
    tr_counter++; // incrementing the Transaction count
  end
endtask : run

//Review task
task wrap_up;
  //auxiliar variables
  real state,tr,err;
  tr = tr_counter * 1.0;
  err = error_count * 1.0;
  //printing results
  $display("at:%t: Verification Finished",$time);
  $display("-----");
  $display("----- Verification Review -----");
  $display("----- Transactions delivered = %d -----", this.tr_counter);
  $display("----- Errors Found = %d-----", this.error_count);
  //Evaluating whether or not the dut passed the test
  state = 100.0 - ((err / tr) * 100.0);
  if (state >= 90)
    $display("----- TEST PASSED at %d/100 -----",state );
  else
    $display("----- TEST FAILED -----");
    $display("-----");
  endtask : wrap_up
endclass : Checker
```

Driver

SV/Verilog Testbench

```
class Driver;
    Transaction r; // Instance to the Transaction
    mailbox #(Transaction) mbA2D; // mailbox to communicate with the Agent
    event DoneA2D; // event to synchronize with the Agent
    event ReadyM2D; // event to synchronize with the Monitor
    vbus_tb bus_tb; // Instance to the interface with the dut
    Callback cbs[$]; // Q of Callbacks

    //Constructor of the Driver class
    function new(input mailbox #(Transaction) mbA2D, input event DoneA2D, ReadyM2D, input vbus_tb bus_tb);
        this.mbA2D = mbA2D; // Giving direction to the mailbox object with the Agent
        this.DoneA2D = DoneA2D; // Giving direction to the event object with the Agent
        this.bus_tb = bus_tb; // Giving direction to interface with the dut
        this.ReadyM2D = ReadyM2D; // Giving direction to the event object with the Monitor
    endfunction : new

    //task to initialize or send values
    task init_send(input logic init);
        @bus_tb.cb; // waiting for a posedge clock
        bus_tb.cb.start <= '0;
        bus_tb.cb.b <= (init)?'0:this.r.b;
        bus_tb.cb.c <= (init)?'0:this.r.c;
        @bus_tb.cb; // waiting for a posedge clock
        bus_tb.cb.start <= '1;
        @bus_tb.cb; // waiting for a posedge clock
        bus_tb.cb.start <= '0;
    endtask : init_send

    //Principal Task
    task run(input int n_times);
        repeat (n_times) begin //Repeating according to the number of tests
            mbA2D.get(r); //getting Transaction handler from the mailbox with the Agent
            @bus_tb.cb;
            bus_tb.cb.n_case <= this.r.n_case;
            r.display_n; //Displaying the stimuli
            foreach(cbs[i]) cbs[i].pre_TX_RX(r); // pre transmission task
        end
    endtask
endclass
```

Driver

```
init_send(1'b0); // sending values to the dut
foreach(cbs[i]) cbs[i].pos_TX_RX(r); // pos transmission task
  @this.ReadyM2D; // waiting for the Monitor to respond
  ->this.DoneA2D; // triggering event to the Agent
end
endtask : run
endclass : Driver
```

```
class Scoreboard;
  Transaction r_q[$]; // Q of Transactions
  static int tr_counter = 1; // Count of Transactions
  static real A = 0.0;
  real b,c;
  event DoneC2S; // event to synchronize with the Checker

  //function to save the Transaction handler in the Transaction Q
  function void save(input Transaction r);
    r_q.push_back(r); // putting the Transaction in the Q
  endfunction : save

  //Calculates the ideal value of A and put this value on the Transaction
  task get_value;
    Transaction r_v[$]; // Declration of a Transaction Q verify existance
    r_v = this.r_q.find(x) with (x.id == this.tr_counter); //Searching existance
    case (r_v.size())
      0: begin
        $display("No Match Found for Transaction N° %d",tr_counter);
        $finish;
      end
      1: begin
        r_v[0].convert(r_v[0].b,this.b); // get the real value of b
        r_v[0].convert(r_v[0].c,this.c); // get the real value of c
        this.A = this.A + this.b * this.c; // get the ideal value of A
        r_v[0].a_sc = this.A; // store the ideal value of A in the Transaction
      end
      default: begin
        $display("Error! Multiple Matches encountered!");
        $finish;
      end
    endcase
  endtask : get_value
```

Scoreboard

```
//Principal task
task run(input int n_times);
  repeat(n_times) //Repeating according to the number of tests
  begin
    @DoneC2S; // Waiting for the Checker to respond
    tr_counter++; //increment the Transaction count
  end
endtask : run

//constructor of the Scoreboard class
function new(input event DoneC2S);
  this.DoneC2S = DoneC2S; // giving the direction of the event object with the Checker
endfunction : new

endclass : Scoreboard
```


Callbacks

```
// Class to define the callback from the Driver to the Monitor
class Driver_cbk extends Callback;
    Monitor Mon; // Monitor class instance
    virtual task pre_TX_RX(ref Transaction r); // before Transmission or Reception
        this.Mon.save(r); //Saving the handler to the current Transaction
    endtask : pre_TX_RX

    //Constructor of the Driver callback class
    function new(input Monitor Mon);
        this.Mon = Mon; // Giving direction the object of the current Monitor
    endfunction : new
endclass : Driver_cbk
```

```
// Class for Agent Callback
class Agent_cbk extends Callback;
    Scoreboard Scb; //Instance to Scoreboard
    virtual task pos_TX_RX(ref Transaction r); // pos transmission or Reception task
        this.Scb.save(r); //save the Transaction handler in the Q
        this.Scb.get_value; //Calculate ideal value of A
    endtask : pos_TX_RX

    // constructor of the Scoreboard class
    function new(input Scoreboard Scb);
        this.Scb = Scb; //Giving the direction of the mailbox object to the Scoreboard
    endfunction : new
endclass : Agent_cbk
```

```
class Environment;
    mailbox #(Transaction) mbG2A; // mailbox to communicate the Generator with the Agent
    mailbox #(Transaction) mbA2D; // mailbox to communicate the Agent with the Driver
    mailbox #(Transaction) mbM2C; // mailbox to communicate the Monitor with the Checker
    Generator Gen; // Instance of the Generator class
    Driver Div; // Instance of the Driver class
    Agent Agt; // Instance of the Agent class
    Monitor Mon; // Instance of the Monitor class
    Checker Chk; // Instance of the Checker class
    Scoreboard Scb; // Instance of the Scoreboard class
    event DoneG2A; // event to synchronize the Generator and the Agent
    event DoneA2D; // event to synchronize the Agent and the Driver
    event ReadyM2D; // event to synchronize the Monitor and the Driver
    event DoneM2C; // event to synchronize the Monitor and the Checker
    event DoneC2S; // event to synchronize the Checker and the Scoreboard
    real prtg; // instance of the reliability percentage
    static int n_times=12; // number of test
    vbus_tb bus_tb; // instance to the current interface
    Driver_cbk Dcbk; // Driver Callback instance
    Agent_cbk Acbk; // Agent Callback instance

    //constructor of the Environment class
    function new(input int n_test, input real prtg);
        this.n_times = n_test; // Matching the number of tests
        this.prtg = prtg; // Matching the percentage of reliability
    endfunction : new

    task build(input vbus_tb bus_tb);
        mbG2A = new; //Creating mailbox object
        mbA2D = new; //Creating mailbox object
        mbM2C = new; //Creating mailbox object
        this.bus_tb = bus_tb; //Directing to the real interface
        Gen = new(mbG2A,this.DoneG2A); //Creating class object
        Agt = new(mbG2A,mbA2D,this.DoneG2A,this.DoneA2D); //Creating class object
        Div = new(mbA2D,this.DoneA2D,this.ReadyM2D,this.bus_tb); //Creating class object
        Mon = new(mbM2C,this.bus_tb,this.ReadyM2D,this.DoneM2C); //Creating class object
```

SV/VERIV

Environment

```
Chk = new(mbM2C,this.DoneM2C,this.DoneC2S,this.prtg); //Creating class object
Scb = new(this.DoneC2S); //Creating class object
Dcbk = new(this.Mon); //Creating class object
this.Div.cbs.push_back(Dcbk); //Saving class handler to the callback Q
Acbk = new(this.Scb); //Creating class object
this.Agt.cbs.push_back(Acbk); //Saving class handler to the callback Q
endtask

//Principal task
task run;
fork
    Div.init_send(1'b1); //Initialize the signals
join
fork // execute all Principal tasks in parallel
    Gen.run(n_times);
    Agt.run(n_times);
    Div.run(n_times);
    Mon.run(n_times);
    Chk.run(n_times);
    Scb.run(n_times);
join
endtask : run

task wrap_up; // execute the review of the simulation
    Chk.wrap_up; // Call statistics of the simulation
endtask : wrap_up

endclass : Environment
```

Test

```
//TEST
program automatic test #(parameter int n_test = 12, parameter real prtg = 1.0)(bus_itf bus_tb);
  Environment Env; //instance of the Environment class
  initial begin
    Env = new(n_test,prtg); // Construct Environment
    Env.build(bus_tb);      // Build all classes
    Env.run;                // Simulation
    Env.wrap_up;            // Review of Simulation
  end
endprogram : test
```

Top

```

module top;
  localparam int n_test = 100; // number of test
  localparam real prtg = 70; // percentage of reliability
  // main timing control of the DUT
  logic reset;
  logic clock;
  bus_itf bus(clock);

  //coverage
  covergroup g1 @(posedge bus.start);
    cv_case: coverpoint bus.n_case;
  endgroup

  g1 cp_case = new;

  // Instances of test and dut
  test #(n_test,prtg) tb (.bus_tb(bus.TB));
  mac dut (.a(bus.a),.b(bus.b),.c(bus.c),.ready(bus.ready),.start(bus.start),.*);

  //Properties
  property inputs;
    bus.start |-> !((bus.b == 'x) || (bus.b == 'z) || (bus.c == 'x) || (bus.c == 'x));
  endproperty : inputs

  property outputs;
    bus.ready |-> !((bus.a == 'x) || (bus.a == 'z));
  endproperty : outputs

  //system clock generator and reset
  initial begin
    reset = '0;
    clock = '0;
    #5ns reset = '1;
    #5ns clock = '1;
    #5ns reset = '0;
    clock = '0;
    forever
      #5ns clock = ~clock;
  end
endmodule: top

```

Script

In cadence incisive 15.0 this Script should be executed

```
HDL_FILES="mac.sv"  
HDL_TEST_FILES="testbench.sv"  
#covopts="+nccovdut+aes +nccovtest+top +nccoverage+B+E+T+U+F"  
covopts="+nccoverage+B+E+T+U+F"  
ncverilog $HDL_FILES $HDL_TEST_FILES +gui +ncaccess+rpc +sv $covopts &
```

For Code Coverage Results

For Functional Coverage Result

Then type run in console and the simulation starts