

```
import "DPI-C" context function void AES_GOLDEN_MODEL(input int num, input int decrypt_i, input
bit[127:0] key_i, data_i, output bit[127:0] data_o);
```

```
interface bus_itf (input logic clock);
```

```
    //Interface with the verification environment
```

```
    logic    wb_stb_i;
```

```
    logic [31:0] wb_dat_o;
```

```
    logic [31:0] wb_dat_i;
```

```
    logic    wb_ack_o;
```

```
    logic [31:0] wb_adr_i;
```

```
    logic    wb_we_i;
```

```
    logic    wb_cyc_i;
```

```
    logic [3:0] wb_sel_i;
```

```
    //interface with the DUV
```

```
    logic    load_o;
```

```
    logic    decrypt_o;
```

```
    logic [127:0] data_o;
```

```
    logic [127:0] key_o;
```

```
    logic [127:0] data_i;
```

```
    logic    ready_i;
```

```
    logic    reset;
```

```
//Defining the timing cotrol with the bus controller
```

```
clocking cb @(posedge clock);
```

```
    input wb_dat_o, wb_ack_o;
```

```
    output wb_stb_i, wb_dat_i, wb_adr_i, wb_we_i, wb_cyc_i, wb_sel_i, reset;
```

```
endclocking : cb
```

```
modport DUV (output data_i, ready_i, input load_o, decrypt_o, data_o, key_o, reset);
```

```

modport TB (clocking cb);

modport WB (input data_i, ready_i, wb_stb_i, wb_dat_i, wb_adr_i, wb_we_i, wb_cyc_i, wb_sel_i,
reset, output load_o, decrypt_o, data_o, key_o, wb_dat_o, wb_ack_o);

endinterface : bus_itf

//Virtual interface
typedef virtual bus_itf.TB vbus_tb;

//Stimulus sequence
typedef enum logic[1:0]{seq_1,seq_2}case_stim;

class Transaction;

//identifier of the transaction
static int id_counter = 1;
int id;

//random stimulus
randc logic [6:0] sub_cyclic;
rand logic [6:0] sub_range;
rand logic [127:0] key_i;
rand logic [127:0] data_i;
rand logic      decrypt_i;

//Constraints
constraint subRanges_cyclic
{
    sub_range == sub_cyclic;
}
constraint subRanges_key
{

```

```

        key_i inside {[ (sub_range << 121) : 128'hFFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF]};
        (sub_range != 7'b1111111) -> key_i inside {[ (sub_range << 121):(sub_range + 7'h01) << 121)}];

        solve sub_range before key_i;
    }
    constraint subRanges_data
    {
        data_i inside {[ (sub_range << 121) : 128'hFFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF_FFFF]};
        (sub_range != 7'b1111111) -> data_i inside {[ (sub_range << 121):(sub_range + 7'h01) << 121)}];

        solve sub_range before data_i;
    }

    //Result registers
    logic [127:0] data_o_duv;
    logic [127:0] data_o_scb;

    //Partial Result registers
    logic [127:0] duv_part_res [10];
    logic [127:0] scb_part_res [10];

    //seq_1 -> data_i has no constraints and can be normally randomized
    //seq_2 -> data_i must be the previous data_i generated
    rand case_stim stim_seq;

    //Auxiliar variables
    static logic [127:0] prev_data;

    //METHODS

    //Randomization
    function void post_randomize();
        if ((prev_data != 'x) && (stim_seq == stim_seq.last()))

```

```
        data_i = prev_data;
    prev_data = data_i;
endfunction : post_randomize
```

```
//Constructor of the transaction
```

```
function new;
    id = id_counter; //getting the id
    id_counter++;      //incrementing for the next id
endfunction : new
endclass : Transaction
```

```
// Base class to define the Callback structure
```

```
virtual class Callback;
```

```
virtual task pre_TX_RX(ref Transaction r); //before a Transmission or Reception
endtask : pre_TX_RX
virtual task pos_TX_RX(ref Transaction r); //after a Transmission or Reception
endtask : pos_TX_RX
```

```
endclass : Callback
```

```
class Generator;
```

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

```
//METHODS
```

```
//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);

    logic flag; //Indicator

    repeat (n_times) begin //Repeating according to the number of tests
        tr = new; // creating the object of the new transaction
        if (tr.randomize) begin // randomizing the stimuli
            mbG2A.put(tr);
        end
        else begin
            $display("Randomization failed!");
            $finish;
        end
        @this.DoneG2A; // waiting for the Agent to respond
    end
endtask

endclass : Generator

class Agent;

    mailbox #(Transaction) mbG2A,mbA2D; // mailbox to communicate with Driver and Agent
    event DoneG2A, DoneA2D, DoneA2S;

    Transaction tr,tr_aux; // instance of the transaction
    Transaction tr_q[$]; // Q of Transactions
    Callback cbs[$]; // Q of callbacks

    //METHODS

```

```
//Constructor of the Agent class
```

```
function new(input mailbox #(Transaction) mbG2A, mbA2D, input event DoneG2A, DoneA2D,  
DoneA2S);
```

```
  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;
```

```
  this.DoneA2D = DoneA2D;
```

```
  this.DoneA2S = DoneA2S;
```

```
endfunction : new
```

```
task run(input int n_times);
```

```
  fork
```

```
    save(n_times);
```

```
    send(n_times);
```

```
  join
```

```
endtask : run
```

```
task save(input int n_times);
```

```
  logic flag; // Indicator
```

```
  repeat (n_times) begin //Repeating according to the number of tests
```

```
    mbG2A.get(tr_aux);
```

```
    tr_q.push_back(tr_aux);
```

```
    //->this.DoneG2A;
```

```
  end
```

```
endtask : save
```

```
task send(input int n_times);
```

```

repeat (n_times) begin //Repeating according to the number of tests
    do #1; while(tr_q.size() == 0);

    tr = tr_q.pop_front();

    foreach(cbs[i]) cbs[i].pre_TX_RX(tr);//Pre callback method
    mbA2D.put(tr);

    foreach(cbs[i]) cbs[i].pos_TX_RX(tr);//Pos callback method

    ->DoneA2S;

fork

    @this.DoneA2S;

    @this.DoneA2D;

join

    ->this.DoneG2A;

end

endtask : send

endclass : Agent

```

```

class Monitor;

    mailbox #(Transaction) mbM2C;

    vbus_tb bus_tb; // Instance to the interface with the duv

    event DoneM2D, DoneA2D, DoneM2C;

    Transaction r_q[$]; // Q of Transaction

    int tr_counter;

    logic [7:0] data_o_adr;

    logic [31:0] data [4];

    logic [127:0] data_o_duv;

    logic [127:0] data_o_duv_vector [11];

    //Constructor of the Monitor class

```

```
function new(input vbus_tb bus_tb, input event DoneM2D, DoneA2D, DoneM2C, input mailbox
#(Transaction) mbM2C);
```

```
    this.bus_tb  = bus_tb; // Giving direction to interface with the dut
```

```
    this.DoneM2D = DoneM2D;
```

```
    this.DoneA2D = DoneA2D;
```

```
    this.DoneM2C = DoneM2C;
```

```
    this.mbM2C  = mbM2C;
```

```
endfunction : new
```

```
//function to the handler to the current Transaction
```

```
function void save(input Transaction tr);
```

```
    r_q.push_back(tr); // Saving the handler in the Q of Transactions
```

```
endfunction : save
```

```
//Adapter to wishbone
```

```
task receive();
```

```
    logic [31:0] ready;
```

```
    for (int i=0; i <11 ; i++)
```

```
    begin
```

```
        //INITIALIZATION
```

```
        ready = '0;
```

```
        // Wait untill the DUV ends processing
```

```
    do
```

```
    begin
```

```
        //@bus_tb.cb;
```

```
        wait(bus_tb.cb.wb_ack_o == 1'b0);
```

```
        //Write control parameter
```

```
        @bus_tb.cb; // waiting for a posedge clock
```

```
        bus_tb.cb.wb_stb_i <= 1'b1;
```



```

bus_tb.cb.wb_adr_i <= 8'h0;
bus_tb.cb.wb_we_i <= 1'b0;
bus_tb.cb.wb_cyc_i <= 1'b1;
//Wait for the correctness acknowledge
wait(bus_tb.cb.wb_ack_o == 1'b1);
@bus_tb.cb; // waiting for a posedge clock
ready = bus_tb.cb.wb_dat_o;
end

while (ready[1] == 1'b1);
do
begin
//@bus_tb.cb;
wait(bus_tb.cb.wb_ack_o == 1'b0);
//Write control parameter
@bus_tb.cb; // waiting for a posedge clock
bus_tb.cb.wb_stb_i <= 1'b1;
bus_tb.cb.wb_adr_i <= 8'h0;
bus_tb.cb.wb_we_i <= 1'b0;
bus_tb.cb.wb_cyc_i <= 1'b1;
//Wait for the correctness acknowledge
wait(bus_tb.cb.wb_ack_o == 1'b1);
@bus_tb.cb; // waiting for a posedge clock
ready = bus_tb.cb.wb_dat_o;
end

while (ready[1] == 1'b0);

// Get data
//begin
/*@bus_tb.cb; // waiting for a posedge clock

```

```

bus_tb.cb.wb_stb_i <= 1'b1;
bus_tb.cb.wb_adr_i <= 8'h24;
bus_tb.cb.wb_we_i <= 1'b0;
bus_tb.cb.wb_cyc_i <= 1'b1;
//Wait for the correctness acknowledge
wait(bus_tb.cb.wb_ack_o == 1'b1);
wait(bus_tb.cb.wb_ack_o == 1'b0);
@bus_tb.cb;
data[0] = bus_tb.cb.wb_dat_o;
//end

//begin
@bus_tb.cb; // waiting for a posedge clock
bus_tb.cb.wb_stb_i <= 1'b1;
bus_tb.cb.wb_adr_i <= 8'h28;
bus_tb.cb.wb_we_i <= 1'b0;
bus_tb.cb.wb_cyc_i <= 1'b1;
//Wait for the correctness acknowledge
wait(bus_tb.cb.wb_ack_o == 1'b1);
wait(bus_tb.cb.wb_ack_o == 1'b0);
@bus_tb.cb;
data[1] = bus_tb.cb.wb_dat_o;
//end

//begin
@bus_tb.cb; // waiting for a posedge clock
bus_tb.cb.wb_stb_i <= 1'b1;
bus_tb.cb.wb_adr_i <= 8'h2C;
bus_tb.cb.wb_we_i <= 1'b0;
bus_tb.cb.wb_cyc_i <= 1'b1;

```

```

//Wait for the correctness acknowledge
wait(bus_tb.cb.wb_ack_o == 1'b1);
wait(bus_tb.cb.wb_ack_o == 1'b0);
@bus_tb.cb;
data[2]    = bus_tb.cb.wb_dat_o;
//end
begin
    @bus_tb.cb; // waiting for a posedge clock
    bus_tb.cb.wb_stb_i <= 1'b1;
    bus_tb.cb.wb_adr_i <= 8'h30;
    bus_tb.cb.wb_we_i  <= 1'b0;
    bus_tb.cb.wb_cyc_i <= 1'b1;
    //Wait for the correctness acknowledge
    wait(bus_tb.cb.wb_ack_o == 1'b1);
    wait(bus_tb.cb.wb_ack_o == 1'b0);
    @bus_tb.cb;
    data[3]    = bus_tb.cb.wb_dat_o;
end*/

// get data
for(int j = 0; j < 4; j++)
    begin
        data_o_adr = (j==0)? 8'h24:data_o_adr + 8'h4;
        @bus_tb.cb; // waiting for a posedge clock
        bus_tb.cb.wb_stb_i <= 1'b1;
        bus_tb.cb.wb_adr_i <= data_o_adr;
        bus_tb.cb.wb_we_i  <= 1'b0;
        bus_tb.cb.wb_cyc_i <= 1'b1;
        //Wait for the correctness acknowledge

```

```

        wait(bus_tb.cb.wb_ack_o == 1'b1);
        wait(bus_tb.cb.wb_ack_o == 1'b0);
    @bus_tb.cb;
    data[j]    = bus_tb.cb.wb_dat_o;
    end

    //Save values to Transaction
    this.data_o_duv_vector[i] = {data[0],data[1],data[2],data[3]};
end

endtask : receive

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 accordinf 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
        for (int j=0; j<10; j++)
            begin
                r[0].duv_part_res [j] = this.data_o_duv_vector[j];
            end
        r[0].data_o_duv = this.data_o_duv_vector[10]; // storing the output value of the duv in the
Transaction
        this.mbM2C.put(r[0]); // putting the Transaction in the Mailbox to the Checker
    end
    default: begin

```

```

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

```

```

//Principal Task
task run(input int n_times);
    tr_counter = 1;
    repeat(n_times) begin
        @this.DoneM2D;
        //@this.DoneA2D;
        receive();
        send(tr_counter);
        ->this.DoneM2C;
        @this.DoneM2C; // wait
        ->this.DoneM2D;
        tr_counter++;
    end
endtask : run
endclass : Monitor

```

```

// 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 Checker;

    mailbox #(Transaction) mbM2C; // mailbox to communicate with the Monitor
    Transaction tr; // Instance of the Transaction
    event DoneM2C; // event to synchronize with the Monitor
    static int error_count = 0; // Defining count of errors
    static int tr_counter = 0; // Defining count of Transactions
    static int n_bins = 0; // Defining count of bins
    static int part_error = 0;


//COVERPOINT SIGNALS
int id;

logic [127:0] key_i;
logic [127:0] data_i;
logic    decrypt_i;

//seq_1 -> data_i has no constraints and can be normally randomized
//seq_2 -> data_i must be the previous data_i generated
case_stim stim_seq;


covergroup Tr_Coverage;

coverpoint id

    {bins id[] = {[0:this.n_bins]};

        option.weight = 2;}

```

```
coverpoint key_i
{
    option.auto_bin_max = 128;
    option.weight       = 1;
    option.goal         = 100;}

```

```
coverpoint data_i
{
    option.auto_bin_max = 128;
    option.weight       = 2;
    option.goal         = 100;}

```

```
coverpoint decrypt_i
{
    bins Encryption = {0};
    bins Decryption = {1};
    option.weight   = 2;
    option.goal     = 6400;}

```

```
coverpoint stim_seq
{
    bins Variant_data = {seq_1};
    bins Invariant_data = {seq_2};
    option.weight     = 2;}

```

```
endgroup : Tr_Coverage
```

```
//Constructor of the Checker class
```

```
function new(input mailbox #(Transaction) mbM2C, input event DoneM2C, input int n_bins);
```

```
    this.n_bins = n_bins - 1;
```

```
    this.mbM2C = mbM2C; // Giving direction to the mailbox object with the Monitor
```

```
    this.DoneM2C = DoneM2C; // Giving direction to the event object with the Monitor
```

```
    Tr_Coverage = new;
```

```
endfunction : new
```

```
// Sample input data
```

```

function void sample(input Transaction tr);

    $display("----- COVERAGE SUCCESSFULLY SAMPLED ----- ");

    $display("-----\n\n");

    this.id      = tr.id-1;
    this.key_i   = tr.key_i;
    this.data_i  = tr.data_i;
    this.decrypt_i = tr.decrypt_i;
    this.stim_seq = tr.stim_seq;

    //Sampling the data
    Tr_Coverage.sample();
endfunction : sample

```

```

task error_check;

    int bit_error = 0;

    //Looking for any error bit
    for(int i=0;i<128;i++)
    begin
        if (this.tr.data_o_duv[i] != this.tr.data_o_scb[i])
            bit_error++;
    end

    if ((bit_error > 0) || (part_error > 0))
    begin
        $display("-----");
        $display("----- In TRANSACTION N* %d : An Error Encountered!-----",this.tr.id);
        $display("-----");
        this.error_count++;
    end

    else
    begin

```



```

$display("-----");
$display("----- In TRANSACTION N* %d : NO Errors Encountered -----",this.tr.id);
$display("-----");
end
$display("\n");
endtask : error_check

```

```

task tr_report;

```

```

    int pos = 0;

```

```

    int j, valid;

```

```

//INITIALIZING PART ERROR COUNTER

```

```

this.part_error = 0;

```

```

$display("-----");
$display("at %t: ----- TRANSACTION N* %d -----",$time, this.tr.id);
$display("-----");
$display("----- Has been successfully driven -----");
$display("-----");
$display("----- DATA_IN is: %h -----",this.tr.data_i);
$display("----- KEY_IN is: %h -----",this.tr.key_i);
$display("-----");
if (this.tr.decrypt_i == '1)
$display("----- PROCESS: DECRYPTION -----");
else
$display("----- PROCESS: ENCRYPTION -----");
$display("-----");
$display("----- DATA_OUT_SCB is: %h -----",this.tr.data_o_scb);
$display("----- DATA_OUT_DUV is: %h -----",this.tr.data_o_duv);

```

```

$display("-----");

this.part_error = 0;
valid = 1;
for(int i=0; i<10; i++)
    begin
        if (valid == 1)
            begin
                if (this.tr.scb_part_res[i] != this.tr.duv_part_res[i])
                    begin
                        this.part_error++;
                        pos = i;
                        valid = 0;
                    end
            end
        end
    end
    end
    if (this.part_error == 0)
        begin
            $display("----- All partial results were correct! -----
-----");
            $display("-----
");
        end
    else
        begin
            $display("----- the partial result NÂ° %d failed -----
-----", pos);
            $display("-----
");
            $display("----- Scb: %h -----",this.tr.scb_part_res[pos]);

```

```

$display("----- DUV: %h -----",this.tr.duv_part_res[pos]);
$display("-----");
");
        end
endtask : tr_report

//Review task
task wrap_up;
    //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);
    $display("-----");
    //Evaluating whether or not the dut passed the test
    if ( error_count == 0)
        $display("----- TEST PASSED -----");
    else
        $display("----- TEST FAILED -----");
        $display("-----");
    endtask : wrap_up

//Principal task
task run(input int n_times);
    repeat(n_times) begin //Repeating according to the number of tests
        @this.DoneM2C; // wait
        mbM2C.get(tr); //Getting the Transaction from the mailbox with the Monitor
        tr_report;
    end
end

```

```

    error_check;

    tr_counter++;

//SAMPLING COVERAGE

    sample(this.tr);

->DoneM2C; // Triggering event to the Monitor

end

endtask : run

endclass : Checker

class Driver;

    Transaction tr; // Instance to the Transaction
    event DoneA2D, DoneM2D;

    mailbox #(Transaction) mbA2D; // mailbox to communicate with the Agent
    vbus_tb bus_tb; // Instance to the interface with the duv
    Callback cbs[$]; // Q of Callbacks

    logic [7:0] key_adr;
    logic [7:0] data_adr;
    logic [31:0] key [4];
    logic [31:0] data [4];

//Constructor of the Driver class

    function new(input mailbox #(Transaction) mbA2D, input vbus_tb bus_tb, input event DoneA2D,
DoneM2D);

        this.mbA2D = mbA2D; // Giving direction to the mailbox object with the Agent
        this.bus_tb = bus_tb; // Giving direction to interface with the dut
        this.DoneA2D = DoneA2D;
        this.DoneM2D = DoneM2D;

```

```
endfunction : new
```

```
//Adapter to wishbone
```

```
task send();
```

```
    //Get Values
```

```
    key = '{this.tr.key_i[127:96],this.tr.key_i[95:64],this.tr.key_i[63:32],this.tr.key_i[31:0]};
```

```
    data = '{this.tr.data_i[127:96],this.tr.data_i[95:64],this.tr.data_i[63:32],this.tr.data_i[31:0]};
```

```
//RESET DESIGN
```

```
begin
```

```
    @bus_tb.cb;
```

```
    #5ns bus_tb.cb.reset <= '0;
```

```
end
```

```
begin
```

```
    @bus_tb.cb;
```

```
    #5ns bus_tb.cb.reset <= '1;
```

```
end
```

```
@bus_tb.cb;
```

```
wait(bus_tb.cb.wb_ack_o == 1'b0);
```

```
begin
```

```
    //Write control parameters
```

```
    @bus_tb.cb; // waiting for a posedge clock
```

```
    bus_tb.cb.wb_dat_i <= {29'h00_00,this.tr.decrypt_i,2'h0};
```

```
    bus_tb.cb.wb_stb_i <= 1'b1;
```

```
    bus_tb.cb.wb_adr_i <= 8'h0;
```

```
    bus_tb.cb.wb_we_i <= 1'b1;
```

```
    bus_tb.cb.wb_cyc_i <= 1'b1;
```

```
//Wait for the correctness acknowledge
```

```
wait(bus_tb.cb.wb_ack_o == 1'b1);
```

```
wait(bus_tb.cb.wb_ack_o == 1'b0);
```

end

// Write data

for(int i = 0; i < 4; i++)

begin

data_adr = (i==0)? 8'h4:data_adr+8'h4;

@bus_tb.cb; // waiting for a posedge clock

bus_tb.cb.wb_dat_i <= data[i];

bus_tb.cb.wb_stb_i <= 1'b1;

bus_tb.cb.wb_adr_i <= data_adr;

bus_tb.cb.wb_we_i <= 1'b1;

bus_tb.cb.wb_cyc_i <= 1'b1;

//Wait for the correctness acknowledge

wait(bus_tb.cb.wb_ack_o == 1'b1);

wait(bus_tb.cb.wb_ack_o == 1'b0);

end

// Write key

for(int j = 0; j < 4; j++)

begin

key_adr = (j==0)? 8'h14:key_adr + 8'h4;

@bus_tb.cb; // waiting for a posedge clock

bus_tb.cb.wb_dat_i <= key[j];

bus_tb.cb.wb_stb_i <= 1'b1;

bus_tb.cb.wb_adr_i <= key_adr;

bus_tb.cb.wb_we_i <= 1'b1;

bus_tb.cb.wb_cyc_i <= 1'b1;

//Wait for the correctness acknowledge

wait(bus_tb.cb.wb_ack_o == 1'b1);

```

        wait(bus_tb.cb.wb_ack_o == 1'b0);
    end

    //Load values in DUV
begin
    @bus_tb.cb; // waiting for a posedge clock
    bus_tb.cb.wb_dat_i <= {29'h00_00,this.tr.decrypt_i,2'h1};
    bus_tb.cb.wb_stb_i <= 1'b1;
    bus_tb.cb.wb_adr_i <= 8'h0;
    bus_tb.cb.wb_we_i <= 1'b1;
    bus_tb.cb.wb_cyc_i <= 1'b1;
    //Wait for the correctness acknowledge
    wait(bus_tb.cb.wb_ack_o == 1'b1);
    wait(bus_tb.cb.wb_ack_o == 1'b0);
end
endtask : send

//Principal Task
task run(input int n_times);
    repeat (n_times) begin
        mbA2D.get(tr);
        foreach(cbs[i]) cbs[i].pre_TX_RX(tr); // pre transmission task
        send();
        foreach(cbs[i]) cbs[i].pos_TX_RX(tr); // pos transmission task
        ->this.DoneM2D;
        @this.DoneM2D;
        ->this.DoneA2D;

    end
end

```

```

endtask : run

endclass : Driver

class Scoreboard;

Transaction r_q[$]; // Q of Transactions

Transaction tr; // Instance to the Transaction

int tr_counter; // Count of Transactions

event DoneA2S; // event to synchronize with the Checker

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

//Calculates the ideal value of the output
task predict;
    bit [127:0] key_i, data_i, data_o;
    int decrypt_i, i;
    int num = 10;

    //RESET OF THE GOLDEN MODEL
    data_i = '0;
    key_i = '0;
    decrypt_i = '0;
    AES_GOLDEN_MODEL(num, decrypt_i, key_i, data_i, data_o);

    //TRANSFORMATION OF THE INPUTS INTO VALID DATA IN THE C++ MODEL
    //data_i:
    for (i = 0; i < 128 ; i++)

```



```

begin
    if (this.tr.data_i[i] == '1')
        data_i[i] = '1';
    else
        data_i[i] = '0';
    end
//key_i:
for (i = 0; i < 128 ; i++)
begin
    if (this.tr.key_i[i] == '1')
        key_i[i] = '1';
    else
        key_i[i] = '0';
    end
//decrypt_i
if (this.tr.decrypt_i == '1')
    decrypt_i = 1;
else
    decrypt_i = 0;
//CALCULATION OF THE RESULT
for(i=0;i<11;i++)
begin
    AES_GOLDEN_MODEL(num,
1'b1,128'h0000_0000_0000_0000_0000_0000_0000_0000,128'h0000_0000_0000_0000_0000_0000_0000_0000, data_o);
    AES_GOLDEN_MODEL(i, decrypt_i, key_i, data_i, data_o);
    //$display("data_o %d: %h", i, data_o);
    if (i < 10)
        this.tr.scb_part_res[i] = data_o;

```

```

        else
            this.tr.data_o_scb = data_o;
        end
    endtask : predict

task get_handler(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("In Scoreboard: No Match Found for Transaction N* %d",tr_counter);
            $finish;
        end
        1: begin
            tr = r[0];
            end
        default: begin
            $display("Error! In Scoreboard, Multiple Matches encountered!");
            $finish;
        end
    endcase
endtask : get_handler

//Principal task
task run(input int n_times);
    tr_counter = 1;
    repeat(n_times) //Repeating according to the number of tests
        begin

```

```

    @DoneA2S; // Waiting for the Checker to respond
    get_handler(tr_counter);
    predict;
    tr_counter++; //increment the Transaction count
    ->DoneA2S;
end
endtask : run

```

```

//constructor of the Scoreboard class
function new(input event DoneA2S);
    this.DoneA2S = DoneA2S; // giving the direction of the event object with the Checker
endfunction : new

endclass : Scoreboard

```

```

// 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
event DoneG2A, DoneA2D, DoneM2D, DoneM2C, DoneA2S;

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

static int      n_times=12; // numer 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);

    this.n_times = n_test; // Matching the number of tests
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,DoneG2A); //Creating class object
    Agt = new(mbG2A,mbA2D,DoneG2A,DoneA2D,DoneA2S); //Creating class object
    Div = new(mbA2D,this.bus_tb,DoneA2D,DoneM2D); //Creating class object

```

```
Mon = new(this.bus_tb,DoneM2D,DoneA2D,DoneM2C,mbM2C);//Creating class object
Chk = new(mbM2C,this.DoneM2C,this.n_times); //Creating class object
Scb = new(this.DoneA2S);//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 : build
```

```
//Principal task
task run;
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
```

```
program test #(parameter int n_test = 12)(bus_itf bus_tb);
```

```

Environment Env; //instance of the Environment class

initial begin

    Env = new(n_test); // Construct Environment

    Env.build(bus_tb); // Build all classes

    Env.run;          // Simulation

    Env.wrap_up;     // Review of Simulation

end

endprogram : test


module top;

    localparam int n_test = 1280; // number of test


    // main timing control of the DUT

    //logic reset;

    logic clock;

    //bus instance

    bus_itf bus(clock);

    int FSM_state = 0;


    // Instances of test and dut

    test #(n_test) tb (.bus_tb(bus.TB));

    aes_duv
    (.clk(clock),.reset(bus.DUV.reset),.load_i(bus.DUV.load_o),.decrypt_i(bus.DUV.decrypt_o),.data_i(bus.D
    UV.data_o),.key_i(bus.DUV.key_o),.ready_o(bus.DUV.ready_i),.data_o(bus.DUV.data_i));


    wb_aes_controller wb (.data_i(bus.WB.data_i), .ready_i(bus.WB.ready_i),
    .wb_stb_i(bus.WB.wb_stb_i), .wb_dat_i(bus.WB.wb_dat_i), .wb_adr_i(bus.WB.wb_adr_i),
    .wb_we_i(bus.WB.wb_we_i), .wb_cyc_i(bus.WB.wb_cyc_i), .wb_sel_i(bus.WB.wb_sel_i),
    .load_o(bus.WB.load_o), .decrypt_o(bus.WB.decrypt_o), .data_o(bus.WB.data_o),
    .key_o(bus.WB.key_o), .wb_dat_o(bus.WB.wb_dat_o), .wb_ack_o(bus.WB.wb_ack_o),
    .clk(clock),.reset(bus.WB.reset),.*);

```

```
//system clock generator and reset
```

```
initial begin
```

```
    bus.reset = '1;
```

```
    clock = '0;
```

```
    forever
```

```
        #5ns clock = ~clock;
```

```
    //ASSERTIONS ACTIVATION
```

```
    $asserton;
```

```
end
```

```
//ASSERTIONS
```

```
property assn_1;
```

```
    @(posedge clock) $rose(bus.load_o) |-> !($isunknown(bus.data_o));
```

```
endproperty : assn_1
```

```
property assn_2;
```

```
    @(posedge clock) $rose(bus.load_o) |-> !($isunknown(bus.key_o));
```

```
endproperty : assn_2
```

```
property assn_3;
```

```
    @(posedge clock) $rose(bus.ready_i) |-> !($isunknown(bus.data_i));
```

```
endproperty : assn_3
```

```
property assn_4;
```

```
    @(posedge clock) $rose(bus.load_o) |=> $stable(bus.data_o) s_until $fell(bus.load_o);
```

```
endproperty : assn_4
```

```
sequence seq_1;
```

```
        ##1 !($fell(bus.ready_i)) [*1:$];  
endsequence : seq_1
```

```
property assn_5;  
    @(posedge clock) $fell(bus.load_o) | => !$rose(bus.load_o) s_until seq_1 [*11];  
endproperty : assn_5
```

```
property assn_6;  
    @(posedge clock) $rose(bus.ready_i) | => $fell(bus.ready_i);  
endproperty : assn_6
```

```
property assn_7;  
    @(posedge clock) $rose(bus.load_o) | => !($rose(bus.load_o)) s_until (seq_1 [*11]);  
endproperty : assn_7
```

```
property assn_8;  
    @(posedge clock) $rose(bus.load_o) | => $stable(bus.decrypt_o) s_until $fell(bus.load_o);  
endproperty : assn_8
```

```
property assn_9;  
    @(posedge clock) $rose(bus.load_o) | => $fell(bus.load_o);  
endproperty : assn_9
```

```
property assn_10;  
    @(posedge clock) $rose(bus.load_o) | => $stable(bus.key_o) s_until $fell(bus.load_o);  
endproperty : assn_10
```

```
property assn_11;
```



```

        @(posedge clock) (bus.wb_adr_i == 8'h4) | => !$changed(bus.wb_adr_i) [*1:$] | =>
(bus.wb_adr_i == 8'h8) | => !$changed(bus.wb_adr_i) [*1:$] | => (bus.wb_adr_i == 8'hC) | =>
!$changed(bus.wb_adr_i) [*1:$] | => (bus.wb_adr_i == 8'h10);

```

```

endproperty : assn_11

```

```

property assn_12;

```

```

        @(posedge clock) bus.wb_adr_i == 8'h14 | => !$changed(bus.wb_adr_i) [*1:$] | => bus.wb_adr_i
== 8'h18 | => !$changed(bus.wb_adr_i) [*1:$] | => bus.wb_adr_i == 8'h1C | => !$changed(bus.wb_adr_i)
[*1:$] | => bus.wb_adr_i == 8'h20;

```

```

endproperty : assn_12

```

```

property assn_13;

```

```

        @(posedge clock) bus.wb_adr_i == 8'h24 | => !$changed(bus.wb_adr_i) [*1:$] | => bus.wb_adr_i
== 8'h28;

```

```

endproperty : assn_13

```

```

property assn_14;

```

```

        @(posedge clock) $fell(bus.load_o) | => seq_1 [*11];

```

```

endproperty : assn_14

```

```

property assn_aux;

```

```

        @(posedge clock) $fell(bus.load_o) | => seq_1;

```

```

endproperty : assn_aux

```

```

data_i_val: assert property (assn_1)

```

```

    else

```

```

        begin

```

```

            $error("at %t: Invalid data_i value driven!", $time);

```

```

            $finish;

```

```

        end

```

key_i_val: assert property (assn_2)

else

begin

\$error("at %t: Invalid key_i value driven!", \$time);

\$finish;

end

data_o_val: assert property (assn_3)

else

begin

\$error("at %t: Invalid data_o value received!", \$time);

\$finish;

end

data_i_stb: assert property (assn_4)

else

begin

\$error("at %t: Stability failure on data_i signal!", \$time);

\$finish;

end

key_i_stb: assert property (assn_10)

else

begin

\$error("at %t: Stability failure on key_i signal!", \$time);

\$finish;

end

data_o_stb: assert property (assn_5)

else

begin

\$error("at %t: Stability failure on data_o signal!", \$time);

\$finish;

```

        end
ready_o_tm: assert property (assn_6)
    else
        begin
            $error("at %t: Timing failure on ready_o signal!", $time);
            $finish;
        end
load_i_val: assert property (assn_7)
    else
        begin
            $error("at %t: Invalid load_i value occurred!", $time);
            $finish;
        end
decrypt_i_stb: assert property (assn_8)
    else
        begin
            $error("at %t: Stability failure on decrypt_i signal!", $time);
            $finish;
        end
load_i_tm: assert property (assn_9)
    else
        begin
            $error("at %t: Timing failure on load_i signal!", $time);
            $finish;
        end
dti_adr_seq: assert property (assn_11)
    else
        begin
            $error("at %t: Write over Whishbone failed at data_i!", $time);

```

```

            $finish;
        end
kyi_adr_seq: assert property (assn_12)
    else
        begin
            $error("at %t: Write over Whishbone failed at key_i!", $time);
            $finish;
        end
dto_adr_seq: assert property (assn_13)
    else
        begin
            //$error("at %t: Read from Whishbone failed at data_o!", $time);
            $finish;
        end
FSM_stuck: assert property (assn_14)
    FSM_state = 0;
    else
        begin
            $error("at %t: DUV FSM got stuck at %d!", $time, FSM_state);
            $finish;
        end
FSM_stuck_aux: assert property (assn_aux)
    FSM_state++;

endmodule: top

```