•RTL AES_ENCRYPT VERIFICATION USING UVM.

- **Under supervision of:** Eng. Sherif Hosny

Table of Contents

A class-based UVM Verification for AES module.

AES Module design:

The module to be verified is "AES_Encrypt.v", which describes a combinational circuit the text to be ciphered and the encryption key as hexadecimal inputs, and then outputs the hexadecimal ciphered. The length of the three signals in our case is 128 bits.

RTL design Architecture.

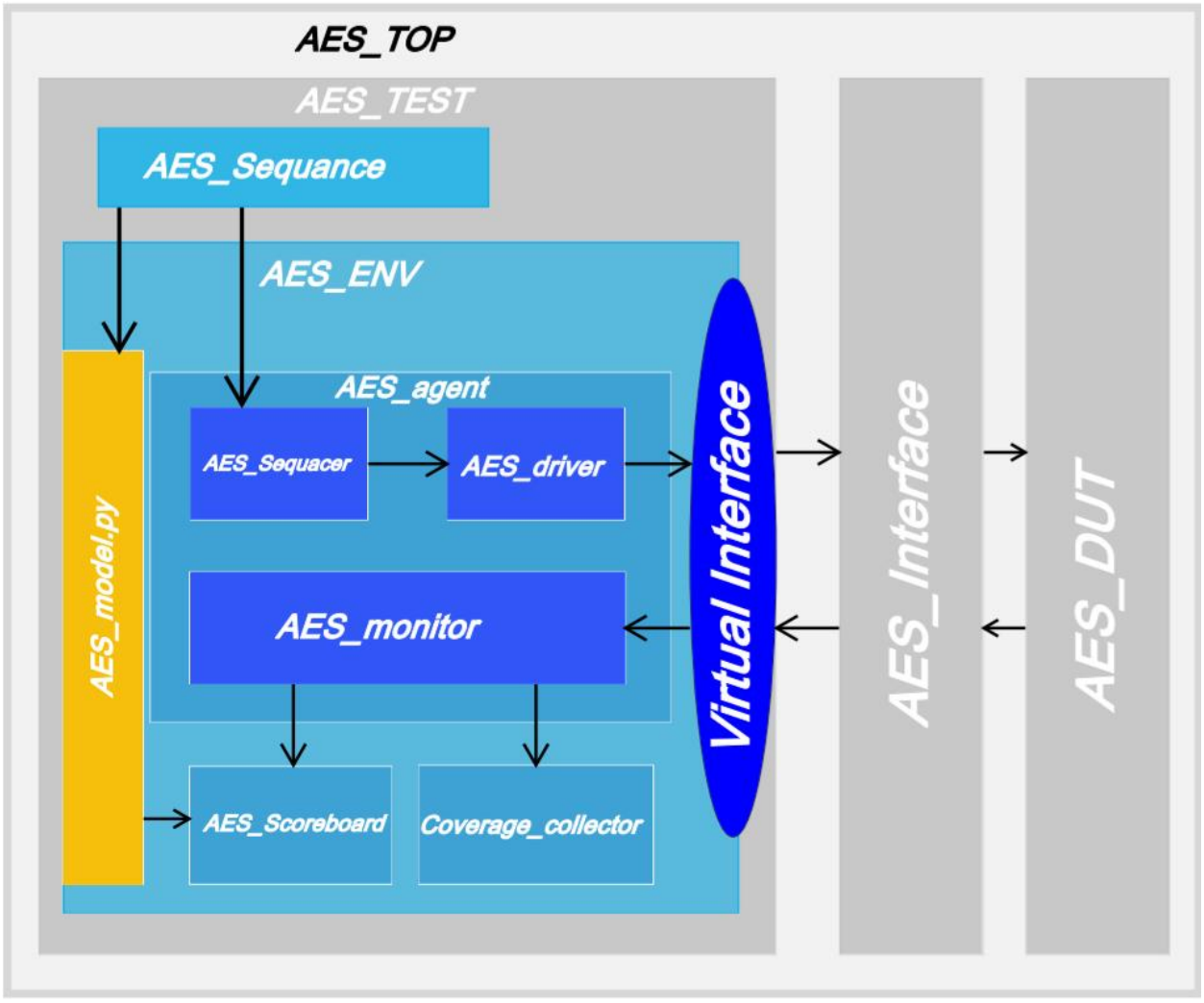
RTL design Modules:

- AES_Encrypt.v

```
module AES_Encrypt#(parameter N=128,parameter Nr=10,parameter Nk=4)(in,key,out);
input [127:0] in;
input [N-1:0] key;
output [127:0] out;
wire [(128*(Nr+1))-1 :0] fullkeys;
wire [127:0] states [Nr+1:0] ;
wire [127:0] afterSubBytes;
wire [127:0] afterShiftRows;

keyExpansion #(Nk,Nr) ke (key,fullkeys);

addRoundKey addrk1 (in,states[0],fullkeys[((128*(Nr+1))-1)-:128]);

genvar i;
generate

    for(i=1; i<Nr ;i=i+1)begin : loop
        encryptRound er(states[i-1],fullkeys[(((128*(Nr+1))-1)-128*i)-:128],states[i]);
```

```
        end
        subBytes sb(states[Nr-1],afterSubBytes);
        shiftRows sr(afterSubBytes,afterShiftRows);
        addRoundKey addrk2(afterShiftRows,states[Nr],fullkeys[127:0]);
            assign out=states[Nr];

endgenerate
endmodule
```

environment:



environment code:

- Top Module.

```
`timescale 1ns/1ps
`include"AES_INTERFACE.sv"
import uvm_pkg::*;
import AES_Package::*;
module  AES_TOP;
parameter Encrypt_count =100;
bit clk;

//interface instantiation
AES_INTERFACE intf();

//dut instantiation
AES_Encrypt AES_Encrypt_1  (intf.in,intf.key,intf.out);
initial begin
    uvm_config_db#(virtual
AES_INTERFACE)::set(null,"uvm_test_top","top2tast",intf);
    run_test("MY_test");
end
endmodule
```

- Pack file

```
package AES_Package;
import uvm_pkg::*;
parameter Encrypt_count =100;
 `include"uvm_macros.svh"
 `include "AES_seq_item.sv"
 `include "AES_Sequancer.sv"
 `include "AES_Driver.sv"
 `include "AES_Monitor.sv"
 `include "AES_Agent.sv"
 `include "AES_Scoreboard.sv"
 `include "AES_coverage_collector.sv"
 `include "AES_Sequance.sv"
 `include "AES_ENV.sv"
 `include "AES_TEST.sv"
endpackage
```

- AES_TEST

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

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

//declare the environment and sequence
AES_env my_env;
AES_Sequence my_seq;

//declare the interface
virtual AES_INTERFACE vif;

//declare the build function
function void build_phase(uvm_phase phase);
super.build_phase(phase);
my_env = AES_env::type_id::create("my_env", this);
my_seq = AES_Sequence::type_id::create("my_seq", this);

if (
    !uvm_config_db #(virtual AES_INTERFACE)::get(this,"", "top2tast", vif)
    ) begin
        `uvm_fatal(get_full_name(), "[MY_test] vif not get");
end
    uvm_config_db #(virtual AES_INTERFACE)::set(this, "my_env", "test2env", vif);
endfunction: build_phase

//declare the connect function
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);

endfunction: connect_phase

// Task: run_phase
task  run_phase(uvm_phase phase) ;
    super.run_phase(phase);
    phase.raise_objection(this);
    `uvm_info("walid_test", "Raising objection", UVM_LOW)
    my_seq.start(my_env.my_agent.m_sequencer);
    `uvm_info("walid_test", "Dropping objection", UVM_LOW)
    phase.drop_objection(this);
```

```
    $display("walid_test_run_phase");
endtask

endclass: MY_test
```

- Sequence class

```
class AES_Sequence extends uvm_sequence;
`uvm_object_utils(AES_Sequence)

//************************************//
///**declare my_trans object**///
    AES_Transaction my_trans;


int file_handle;
int read_handle;
logic  [127:0] py_out_1,py_out_2;
int Encrypt_count = 100;
//************************************//
///**Constructor**///
extern function new(string name = "AES_Sequence");

//************************************//
/////////****body task****/////////
extern task body();
endclass: AES_Sequence


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

//////////////////////////////
/////////****body****/////////
 task AES_Sequence::body();
 for(int i =0;i<Encrypt_count;i++)   //run the sequance 100 times check???
begin
    my_trans = AES_Transaction::type_id::create("my_trans");
    start_item(my_trans);
    $display("[sequance] start body %0d",i);
    if(!my_trans.randomize(in, key)) begin
```

```
    $display("errorrrrrrrrr");
    end


    file_handle = $fopen("in_file.txt","w");


    $fwrite(file_handle,"%h\n%h",my_trans.in,my_trans.key);

    $fclose(file_handle);

    $system("python AES_model.py");  //run the python model ???


    // Open the file for reading
    read_handle = $fopen("in_file.txt", "r");
    assert(read_handle != 0) else `uvm_fatal("file_error", "file not found");

    // Read the output from the file
    void'($fscanf(read_handle, "%h\n%h", py_out_1,py_out_2));
    $fclose(read_handle);

    // Display the read value (optional)
    $display("Read value from in_file.txt 1: %h", py_out_1);
    $display("Read value from in_file.txt 2: %h", py_out_2);
    $display("[sequance] finish body %0d",i);
    finish_item(my_trans);
end
endtask: body
```

- Env class

```
class AES_env extends uvm_env;
`uvm_component_utils(AES_env)

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

//declare the conponents
AES_Agent my_agent;
AES_Scoreboard my_scoreboard;
AES_coverage_collector my_collector;

//declare the virtual interface
virtual AES_INTERFACE my_vif;

//declare the build function
function void build_phase(uvm_phase phase);
super.build_phase(phase);
my_agent = AES_Agent::type_id::create("my_agent", this);
my_scoreboard = AES_Scoreboard::type_id::create("my_scoreboard", this);
my_collector = AES_coverage_collector::type_id::create("my_collector", this);
if(
    !uvm_config_db #(virtual AES_INTERFACE)::get(this, "", "test2env", my_vif)
  )
`uvm_fatal(get_full_name(), "[AES_env] vif not get")

uvm_config_db #(virtual AES_INTERFACE)::set(this, "my_agent", "env2agent", my_vif);
endfunction: build_phase

//declare the connect function
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
my_agent.m_monitor.port.connect(my_collector.analysis_export);
my_agent.m_monitor.port.connect(my_scoreboard.imp);
endfunction: connect_phase

// Task: run_phase
task run_phase(uvm_phase phase);
super.run_phase(phase);
endtask: run_phase
endclass: AES_en
```

- Agent class

```
class AES_Agent extends uvm_agent;
  `uvm_component_utils(AES_Agent)
  // Components
  AES_Driver m_driver;
  AES_Sequencer m_sequencer;
  AES_Monitor m_monitor;

  // Constructor
  function new(string name = "AES_Agent", uvm_component parent = null);
  super.new(name, parent);
  endfunction: new
//interface
  virtual AES_INTERFACE m_vif;

  // Build Phase
    function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    m_driver    = AES_Driver::type_id::create("m_driver", this);
    m_sequencer = AES_Sequencer::type_id::create("m_sequencer", this);
    m_monitor   = AES_Monitor::type_id::create("m_monitor", this);


    if(!uvm_config_db #(virtual AES_INTERFACE)::get(this, "", "env2agent", m_vif))
    `uvm_fatal(get_full_name(), "[AES_Agent] vif not get")

    uvm_config_db #(virtual AES_INTERFACE)::set(this, "m_driver", "driver2agent", m_vif);
    uvm_config_db #(virtual AES_INTERFACE)::set(this, "m_monitor", "monitor2agent", m_vif);
  endfunction: build_phase

  // Connect Phase
    function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    m_driver.seq_item_port.connect(m_sequencer.seq_item_export);
    endfunction: connect_phase

  // Task: run_phase
    task run_phase(uvm_phase phase);
    super.run_phase(phase);
    endtask: run_phase
endclass
```

- Coverage collector

```
class AES_coverage_collector extends uvm_subscriber #(AES_Transaction);
`uvm_component_utils(AES_coverage_collector)

//transaction object
AES_Transaction tran_mon2sub;

//coverage group
covergroup cov;
out:coverpoint tran_mon2sub.out;
endgroup:cov

// Constructor
function new(string name = "AES_coverage_collector", uvm_component parent = null);
super.new(name, parent);
cov  = new();
endfunction: new

//declare the build function
function void build_phase(uvm_phase phase);
super.build_phase(phase);
endfunction: build_phase

//declare the connect function
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
endfunction: connect_phase

// Task: run_phase
task run_phase(uvm_phase phase);
super.run_phase(phase);
endtask: run_phase
// Task: write
function void write(AES_Transaction t);
tran_mon2sub =t;
cov.sample();
endfunction: write

endclass
```

- AES_Driver

```
class AES_Driver extends uvm_driver #(AES_Transaction);
`uvm_component_utils(AES_Driver)

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

//create transaction object
AES_Transaction my_trans;

//declare the virtual interface
virtual AES_INTERFACE vif;

// Build function
function void build_phase(uvm_phase phase);
super.build_phase(phase);
my_trans =AES_Transaction::type_id::create("my_trans");

if(
    !uvm_config_db #(virtual AES_INTERFACE)::get(this, "", "driver2agent", vif)
  )
`uvm_fatal(get_full_name(), "[AES_Monitor] vif not get")
endfunction: build_phase

// Connect function
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
endfunction: connect_phase

// Task: run_phase
task run_phase(uvm_phase phase);
    forever begin
    seq_item_port.get_next_item(my_trans);
        vif.in  <= my_trans.in;
        vif.key <= my_trans.key;
    seq_item_port.item_done();
    $display("[driver] done");
    end
endtask: run_phase
endclass: AES_Driver
```

- AES_Monitor

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

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

//declare the interface
virtual AES_INTERFACE vif;

//declare transaction
AES_Transaction my_trans;

//declare analysis port
uvm_analysis_port #(AES_Transaction) port;

logic [127:0]  out;

// Build function
function void build_phase(uvm_phase phase);
super.build_phase(phase);
port = new("port", this);
my_trans =AES_Transaction::type_id::create("my_trans");

if(
        !uvm_config_db #(virtual AES_INTERFACE)::get(this, "", "monitor2agent", vif)
  )
`uvm_fatal(get_full_name(), "[AES_Monitor] vif not get")
endfunction: build_phase

// Connect function
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
endfunction: connect_phase

// Task: run_phase
task run_phase(uvm_phase phase);
        super.run_phase(phase);
        forever begin
        @(vif.out);
        $display("[walid_monitor] the value of dut out %h",vif.out);
        my_trans.out    =  vif.out;
```

```
        port.write(my_trans);
        $display("[walid_monitor] done");
        end
endtask: run_phase

//
endclass: AES_Monitor
```

- AES_INTERFACE

```
interface  AES_INTERFACE();
    logic [127:0]  in ;
    logic [127:0]  key;
    logic [127:0]  out;
endinterface
```

- AES_model.py

```
from Crypto.Cipher import AES

with open('in_file.txt', 'r') as file:
    data_hex = file.readline().strip()  # Read the data hex string
    key_hex = file.readline().strip()  # Read the key hex string

data = bytes.fromhex(data_hex)
key = bytes.fromhex(key_hex)

cipher = AES.new(key, AES.MODE_ECB)

ciphertext = cipher.encrypt(data)

with open('out_file.txt', 'w') as file:
    file.write(ciphertext.hex())
```

- AES_Scoreboard

```
class AES_Scoreboard extends uvm_scoreboard;
`uvm_component_utils(AES_Scoreboard)

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

//declare the analysis imp
uvm_analysis_imp #(AES_Transaction, AES_Scoreboard) imp;

//declare properties
int             file_handle;
logic [127:0]   py_out;
int             count;



//declare the build function
function void build_phase(uvm_phase phase);
super.build_phase(phase);
imp = new("imp", this);
endfunction: build_phase

//declare the connect function
function void connect_phase(uvm_phase phase);
super.connect_phase(phase);
endfunction: connect_phase

// Task: run_phase
task run_phase(uvm_phase phase);
super.run_phase(phase);
endtask: run_phase



//  write_function
function void write (AES_Transaction t);

    file_handle = $fopen("out_file.txt","r");
    if ($fscanf(file_handle, "%h", py_out) == 0) begin
        $display("Error: no output exist");

    end
```

```
    else if (py_out==t.out) begin
        $display("operation ID:%4d, Passed Encryption:\n Expected_py: %h \n
Checked:     %h", count, py_out, t.out);
        $display("--------------------------------");
        //$stop;
    end
    else if (py_out!=t.out) begin
        $display("operation ID:%4d, Failed Encryption:\n Expected_py: %h \n
Checked:     %h", count, py_out, t.out);
        $display("--------------------------------");

    end
    $fclose(file_handle);
endfunction


endclass: AES_Scoreboard
```

- AES_Sequancer

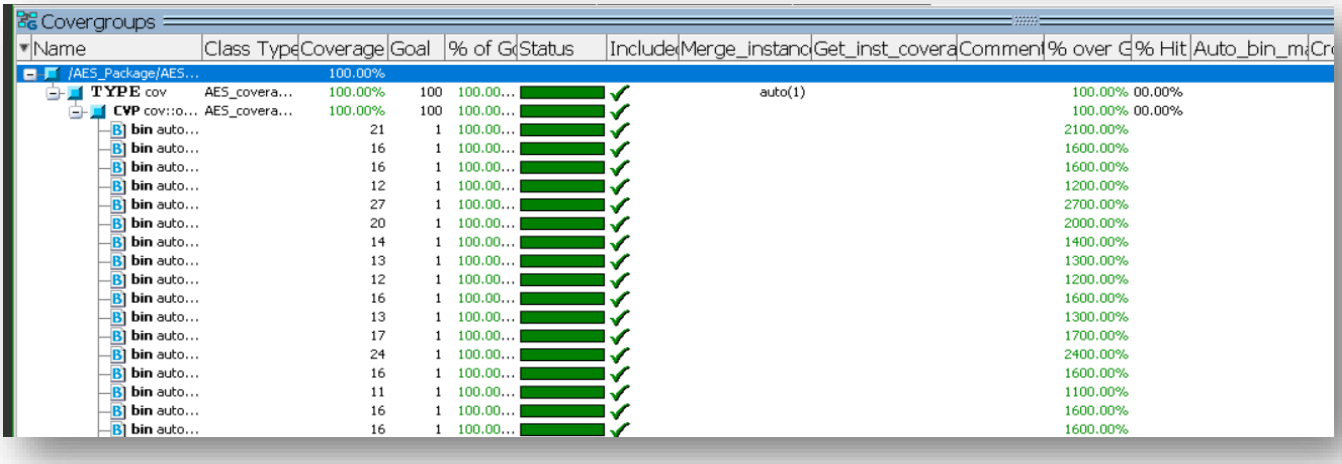
```
class AES_Sequencer extends uvm_sequencer #(AES_Transaction);
    `uvm_component_utils(AES_Sequencer)
    // Constructor
    function new(string name = "AES_Sequencer", uvm_component parent = null);
    super.new(name, parent);
    endfunction: new

    // Build function
    function void build_phase(uvm_phase phase);
    super.build_phase(phase);
    endfunction: build_phase

    // Connect function
    function void connect_phase(uvm_phase phase);
    super.connect_phase(phase);
    endfunction: connect_phase

    // Task: run_phase
    task run_phase(uvm_phase phase);
    super.run_phase(phase);
    endtask: run_phase
endclass: AES_Sequencer
```

- AES_seq_item

```
class AES_Transaction extends uvm_sequence_item;
`uvm_object_utils(AES_Transaction)
// function new
function new(string name = "AES_Transaction");
super.new(name);
endfunction: new

    rand  logic  [127:0]  in ;
    rand  logic  [127:0]  key;
          logic  [127:0]  out;
endclass: AES_Transaction
```

Results:

```
--------------------------------
[AES_monitor] done
[AES_monitor] the value of dut out d84dcb74dca003270a1a0b781c1640a1
operation ID:    0, Passed Encryption:
 Expected_py: d84dcb74dca003270a1a0b781c1640a1
 Checked:     d84dcb74dca003270a1a0b781c1640a1
--------------------------------
[AES_monitor] done
```



- Function coverage:

```
COVERGROUP COVERAGE:
----------------------------------------------------------------------------------------------------
Covergroup                                                 Metric       Goal    Status

----------------------------------------------------------------------------------------------------
 TYPE /AES_Package/AES_coverage_collector/cov            100.00%        100    Covered
    covered/total bins:                                       64         64
    missing/total bins:                                        0         64
    % Hit:                                                100.00%        100
    Coverpoint cov::out                                   100.00%        100    Covered
        covered/total bins:                                   64         64
        missing/total bins:                                    0         64
        % Hit:                                            100.00%        100
        bin auto[0:53169119831396634916152282411213783 03]
                                                              21          1    Covered
        bin auto[53169119831396634916152282411213783 04:106338239662793269832304564822427566 07]
                                                              16          1    Covered
        bin auto[106338239662793269832304564822427566 08:159507359494189904748456847233641349 11]
                                                              16          1    Covered
        bin auto[159507359494189904748456847233641349 12:212676479325586539664609129644855132 15]
                                                              12          1    Covered
        bin auto[212676479325586539664609129644855132 16:265845599156983174580761412056068915 19]
                                                              27          1    Covered
        bin auto[265845599156983174580761412056068915 20:319014718988379809496913694467282698 23]
                                                              20          1    Covered
        bin auto[319014718988379809496913694467282698 24:372183838819776444413065976878496481 27]
                                                              14          1    Covered
        bin auto[372183838819776444413065976878496481 28:425352958651173079329218259289710264 31]
                                                              13          1    Covered
        bin auto[425352958651173079329218259289710264 32:478522078482569714245370541700924047 35]
                                                              12          1    Covered
```

```
TOTAL COVERGROUP COVERAGE: 100.00%  COVERGROUP TYPES: 1

ASSERTION RESULTS:
--------------------------------------------------------------------
Name                 File(Line)                   Failure      Pass
                                                  Count        Count
--------------------------------------------------------------------
/AES_Package/AES_Sequence/body/#anonblk#104709781#32#4#/#ublk#104709781#33/immed__56
                     AES_Sequance.sv(56)             0          1
```