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

•RTL AES\_ENCRYPT VERIFICATION using   
UVM.

Table of Contents

[**A class-based UVM Verification for AES module.** 2](#_Toc177220658)

[*AES Module design:* 2](#_Toc177220659)

[*RTL design Architecture.* 2](#_Toc177220660)

[*RTL design Modules:* 2](#_Toc177220661)

[ AES\_Encrypt.v 2](#_Toc177220662)

[*environment:* 3](#_Toc177220663)

[*environment code:* 4](#_Toc177220664)

[ Top Module. 4](#_Toc177220665)

[ Pack file 4](#_Toc177220666)

[ AES\_TEST 5](#_Toc177220667)

[ Sequence class 6](#_Toc177220668)

[ Env class 8](#_Toc177220669)

[ Agent class 9](#_Toc177220670)

[ Coverage collector 10](#_Toc177220671)

[ AES\_Driver 11](#_Toc177220672)

[ AES\_Monitor 12](#_Toc177220673)

[ AES\_INTERFACE 13](#_Toc177220674)

[ AES\_model.py 13](#_Toc177220675)

[ AES\_Scoreboard 14](#_Toc177220676)

[ AES\_Sequancer 15](#_Toc177220677)

[ AES\_seq\_item 16](#_Toc177220678)

[*Results:* 16](#_Toc177220679)

[ Function coverage: 17](#_Toc177220680)

# **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.*

A blue rectangular sign with white text

Description automatically generated

## *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 |

## A diagram of a software system Description automatically generated *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 |

## A computer code with blue text Description automatically generated*Results:*

A screenshot of a computer

Description automatically generated

### Function coverage:

A close-up of a document

Description automatically generatedA screenshot of a document

Description automatically generated