ELSYS DESIGN

AES VIP
VERIFICATION PLAN

PURPOSE

This document describes the verification of AES VIP. It contains an overview of which AES128 IP features will be verified, verification strategy and objectives. It will also detail the flow, the test bench architecture, and tests used for achieving the complete verification of AES VIP.

Revision History

Rev	Date	Author	Description
0.1	31/10/2024	Vincent Yang	First draft release

References

Title	Reference
Specification for the ADVANCED ENCRYPTION STANDARD (AES)	https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

Contents

List of figures

List of tables

1. Verification strategy

The AES VIP is an UVM agent for all AES protocol-based IPs. According to the specification, this AES VIP supports AES encryption and decryption protocol features below:

- Supports 128-bit plaintext and ciphertext data.
- Supports 128-bit encryption keys.

Verification of AES IP will follow the steps below:

- An UVM environment will be written in System Verilog.
- The VIP will communicate directly to the AES IP ports.
- An AES C model will be used to check the DUT output in the scoreboard with a DPI-C import.
- SV Assertions will be used to check the protocol in the interface class.

2. Test bench

2.1. Block diagram

Figure 1 gives the AES UVM test environment block diagram that shows the structure and flow. Inside the aes_env, an aes_agent is instantiated to drive the aes128 VHDL IP core. The signals generated by the IP core are monitored by the aes_monitor, which gathers these signals and sends them to the scoreboard.

The scoreboard then forwards these signals to a DPI-C function, which compares them to a reference output generated by a C model. This setup allows for real-time verification by comparing the IP core's output against expected values from the C model.

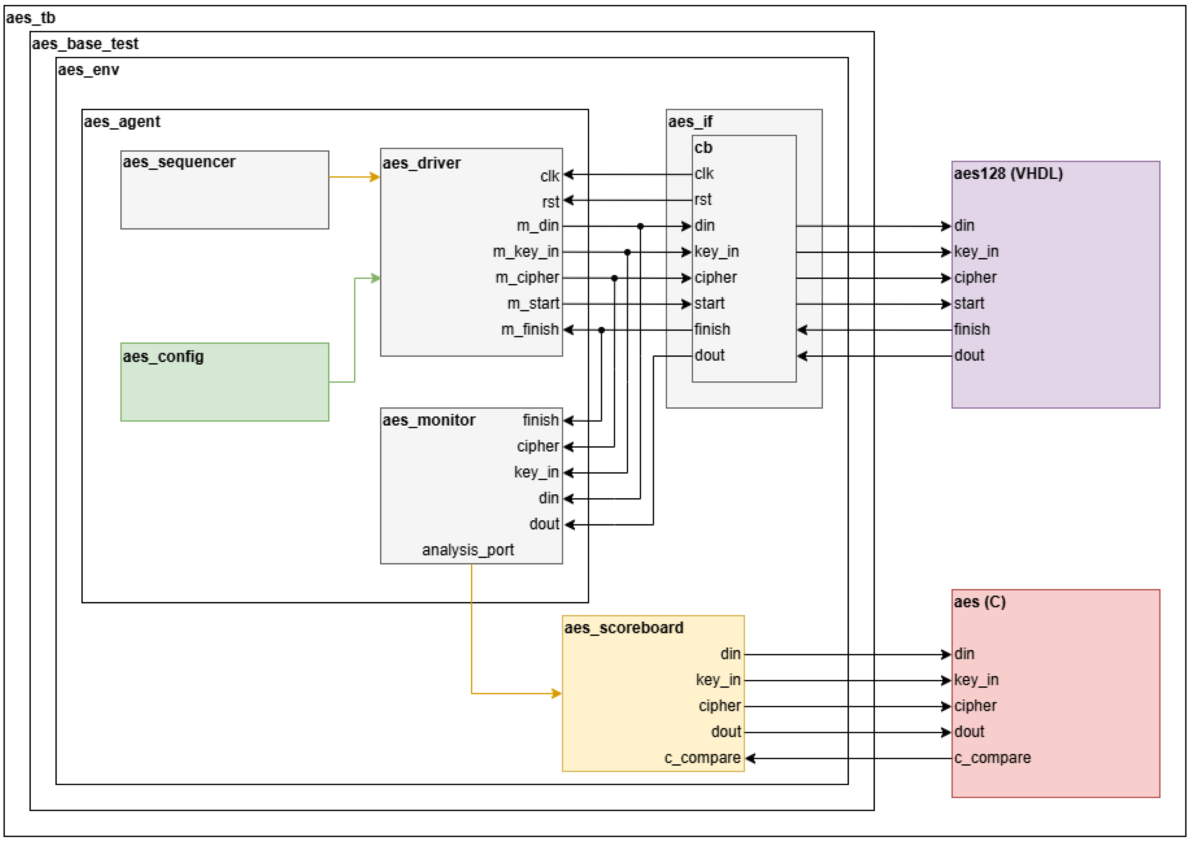


Figure 1: AES UVM test environment block diagram

2.2. Configuration

The configuration aes_config class contains the different settings given to the VIP. Table 1 gives configuration class fields.

Name	Type	Default value	Description
vif	virtual aes_if	None	Virtual interface
name	string	None	Name of the configuration
		General UVM specifications	
is_active	uvm_active_passive_enum	UVM_ ACTIVE	Set the agent configuration to active or passive
coverage_enable	bit	0	Enable coverage
checks_enable	bit	0	Enable checks

Table 1: Configuration class fields

2.3. Interface

The interface aes_if class contains the different wires driven by the AES interface and assertions to check the AES protocol. Section 4 describes assertions.

Table 2 gives interface class fields.

Name	Type	Description
clk	Wire	Clock, actions are performed on its rising edge
arst	Wire	Asynchronous reset, active high. When asserted, set all the signals to its initial state
start	Wire	Start signal. When asserted, it indicates that the input data is ready to be processed. This signal triggers the beginning of the encryption or decryption process
din	Wire [127:0]	Data input signal. It carries the 128-bit plaintext data that needs to be encrypted
key_in	Wire [127:0]	Key input signal. It carries the 128-bit encryption key used for the AES algorithm
cipher	Wire	Indicates which operation is performed. If set to 1, it's an encryption; if set to 0, it's a decryption
dout	Wire [127:0]	Data output signal. It carries the 128-bit ciphertext result after the encryption process is completed.
finish	Wire	The finish signal. When asserted, it indicates that the encryption process is complete and the output data is valid.

Table 2: Interface class fields

2.4. Sequence item

Sequence item aes_tx class contain the different attributes contained in a transaction, constraints on these attributes, and basic functions to print, copy, compare transactions.

Table 3 gives sequence item fields.

Name	Type	Description
m_din	rand bit [127:0]	Data input signal
m_key_in	rand bit [127:0]	Key input signal
m_dout	rand bit [127:0]	Data output signal
m_cipher	rand bit	Encryption or decryption mod signal.
m_delay	rand int	Delay between each operation

Table 3: Sequence item fields

Table 4 gives sequence item constraints.

Name	Min value	Max value	Description
max_delay_rate_c	1	100	Constrains m_delay range

Table 4: Sequence item constraints

2.5. Agent

Agent aes_agent class contains the instantiations of agent blocks: driver, monitor and sequencer.

2.5.1. Driver

Driver aes_driver class drives transactions from sequencer to the interface. They drive interface's control signals regarding from the AES protocol.

Table 5 gives driver fields.

Name	Type	Description
General fields		
vif	virtual aes_if	Virtual aes interface
m_config	aes_config	Configuration set by the agent

Table 5: Driver class fields

Table 6 gives driver functions and tasks.

Name	Type	Arguments	Return	Description
new	Function	string name, uvm_component parent	/	Constructor
build_phase	Function	uvm_phase phase	void	Build phase
run_phase	Task	uvm_phase phase	void	This task gets a new transaction, and call drive item tasks
do_drive	Task	/	/	This task drives sequence items received to virtual interface
handle_reset	Task	/	/	This task set reset values to all data driven if rst is asserted

Table 6: Driver functions and tasks

2.5.2. Monitor

Monitor aes_monitor class detect AES operation at the interface by waiting finish signals. They send sequence item to the scoreboard and to subscribers.

Table 7 gives monitor fields.

Name	Type	Description
vif	virtual aes_if	Virtual AES interface
m_config	aes_config	Configuration set by the agent
analysis_port	uvm_analysis_port #(aes_tx)	Analysis port where monitor writes detected item.
m_trans	aes_tx	Item to receive data from monitor
m_trans_cloned	aes_tx	Item to clone data from monitor to send it to the subscriber

Table 7: Monitor fields

Table 8 gives monitor functions and tasks.

Name	Type	Arguments	Return	Description
new	Function	string name, uvm_component parent	/	Constructor. Build and connect phase
run_phase	Task	uvm_phase phase	/	Run phase
do_mon	Task	/	/	Fill a sequence item and send it to subscribers while monitor detects a transfer

Table 8: Monitor functions and tasks

2.6. Scoreboard

The aes_scoreboard class compares AES operations performed by the DUT as observed by the monitor. It verifies that the encryption or decryption done by the DUT is correct by comparing it to a reference AES C model, if not correct, it sends uvm_error error. To do so, it imports a DPI-C function compare_to_c_model from the aes_model.c file. Scoboard also keeps a count of all the transactions processed and all transactions that matched with the C model reference.

At the end of the test, it summarizes all the transactions observed.

2.6.1. C Model

An AES C model aes_model.c has functions to compute AES encryption or decryption.

Table 9 gives aes_model.c main function that compare data computed from the DUT and data computed from the reference.

Name	Type	Arguments	Return	Description
Compare_to_c_model	Function	int data_in[3], int key_in[3], int data_out[3], int cipher_phase	int matrix_compare	Computes AES encryption or decryption depending of argument cipher_phase. Then compare computed golden matrix to data_out from the VHDL DUT. Returns 1 if it matches, else return 0.

Table 9: C model main function

Figure 2 resumes compare_to_c_model operations.

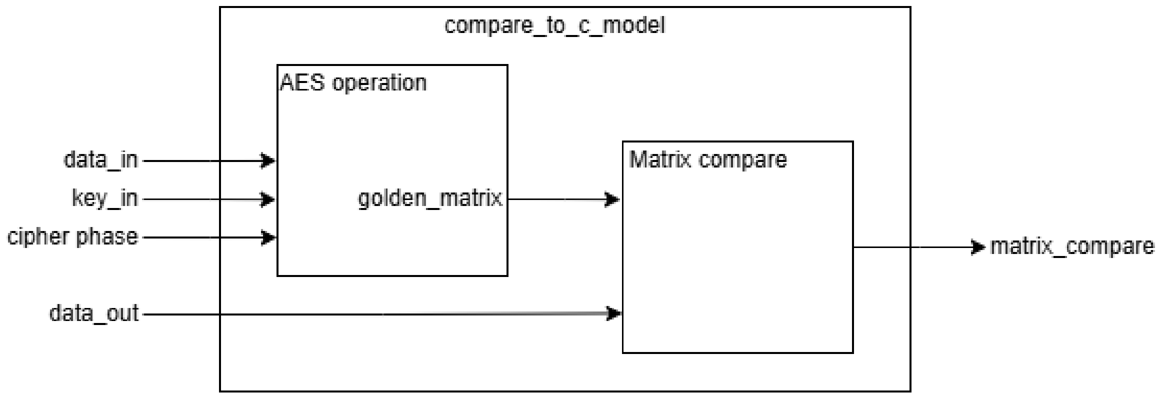


Figure 2: Block diagram of compare_to_c_model function

2.7. Coverage

Table 10 gives coverpoints checked during coverage in aes_coverage class.

Bins	Data covered	Value covered	Description
coverpoint: cp_m_din			
Din_range	bit [127:0] m_din	Auto_bin_max = 1024	Checks that the signal takes values between $[0: 2^{127} - 1]$
coverpoint: cp_m_key_in			
Key_range	bit [127:0] m_key	Auto_bin_max = 1024	Checks that the signal takes values between $[0: 2^{127} - 1]$
coverpoint: cp_m_cipher			
Cipher_range	bit m_cipher	Auto_bin_max = 2	Checks that the signal take values between 0 and 1

Table 10: Coverpoint list

Table 11 gives cross coverpoint checked during coverage in aes_coverage class.

Cross coverpoint name	Coverpoint 1	Coverpoint 2	Description
cross_cp_m_cipher_m_din	cp_m_cipher	cp_m_din	Check that m_din takes all possible bin values, regardless of the AES operation.
cross_cp_m_cipher_m_key_in	cp_m_cipher	cp_m_key_in	Check that m_key_in takes all possible bin values, regardless of the AES operation.

Table 11: Cross coverpoint list

3. DUT

DUT is a VHDL 128 bit AES core IP.

Figure 3: Block diagram of DUT aes128

DUT design is detailed in AES_specification file.

4. Database Structure and Simulation Flow

This section describes simulation directories and simulation commands.

4.1. Database structure

Figure 4 gives the VIP environment structure. Directories, and files inside are detailed below.

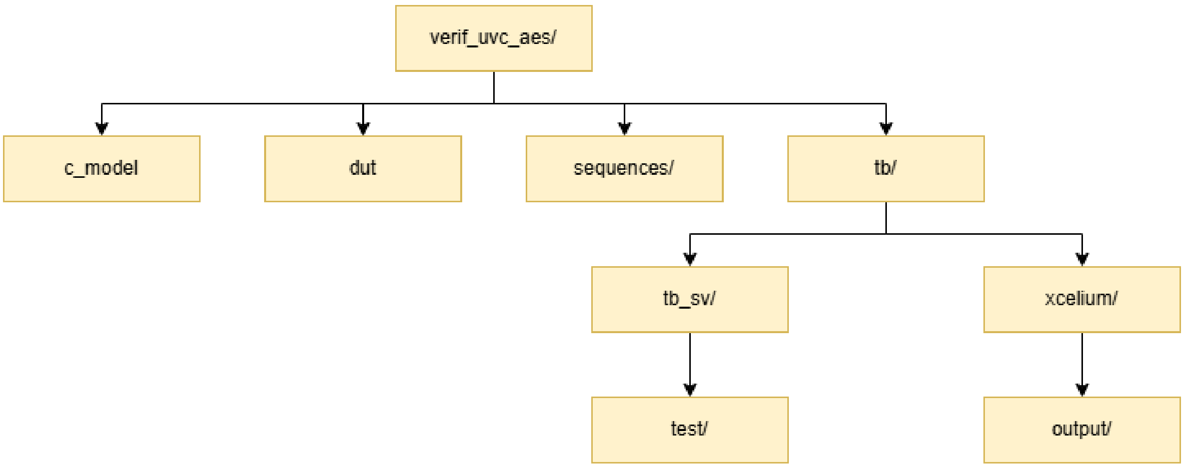


Figure 4: Directory structure of the VIP environment

Directory verif_uvc_aes/ contains:

- UVM file: aes_agent.sv that contains an AES VIP agent class.
- UVM file: aes_config.sv that contains an AES VIP config class.
- UVM file: aes_coverage.sv that contains an AES VIP coverage class.
- UVM file: aes_driver.sv that contains an AES VIP driver class.
- UVM file: aes_if.sv that contains an AES VIP interface.
- UVM file: aes_monitor.sv that contains an AES VIP monitor class.
- UVM file: aes_pkg.sv that contains a package for the VIP.
- UVM file: aes_sequencer.sv that contains an AES VIP sequencer class.
- UVM file: aes_tx.sv that contains an AES VIP item class.

Directory sequences/ contains:

- UVM file: aes_seq_lib.sv that contains different sequence classes used by AES VIP agents.
- UVM file: aes_top_seq_lib.sv that contains different sequences composed of sequences from aes_seq_lib.sv to run tests.

Directory tb/ contains:

- Directory tb_sv/ that contains:
    - Directory test/ that contains:
        - UVM file: test aes_test_coverage.sv
        - UVM file: test aes_test_flush.sv
        - UVM file: test aes_test_syncreq.sv
        - UVM file: test aes_test_transmit_coresight_id.sv
        - UVM file: test aes_test_transmit_locally.sv
        - UVM file: test aes_test_wakeup.sv
        - UVM file: test aes_test_transmit.sv
        - UVM file: test aes_test_transmit_then_flush.sv
        - UVM file: test aes_test_transmit_trace_trigger.sv
    - UVM file: aes_env.sv that contains top level environment of the VIP.
    - UVM file: aes_scoreboard.sv that contains the scoreboard.
    - UVM file: aes_tb.sv that contains the test bench.
    - UVM file: aes_test_base.sv that contains the base test.
    - UVM file: aes_test_pkg.sv that a package for the top-level environment of the VIP.
    - UVM file: aes_th.sv that contains test harness.
- Directory xcelium/ that contains:
    - Directory output/ that contains output simulation logs.
    - Bash file: run that will lauch xrun with given arguments.

4.2. Simulation Flow

<u>TOOL REQUIRED</u>: Cadence Incisive

With the run script “run” you can run a test by giving arguments in the command line.

To run a test:

- Go to verif_uvc_aes/tb/xcelium/ directory.
- Use run bash script with the test wanted in argument.

Execution example:

- Run a test aes_test_cipher:

  `./run +UVM_TESTNAME=aes_test_cipher`

The purpose of the regression script “run_regression” is to run all tests, check if they have passed or failed and then report results to the user in the console and in a log file.

To run a regression:

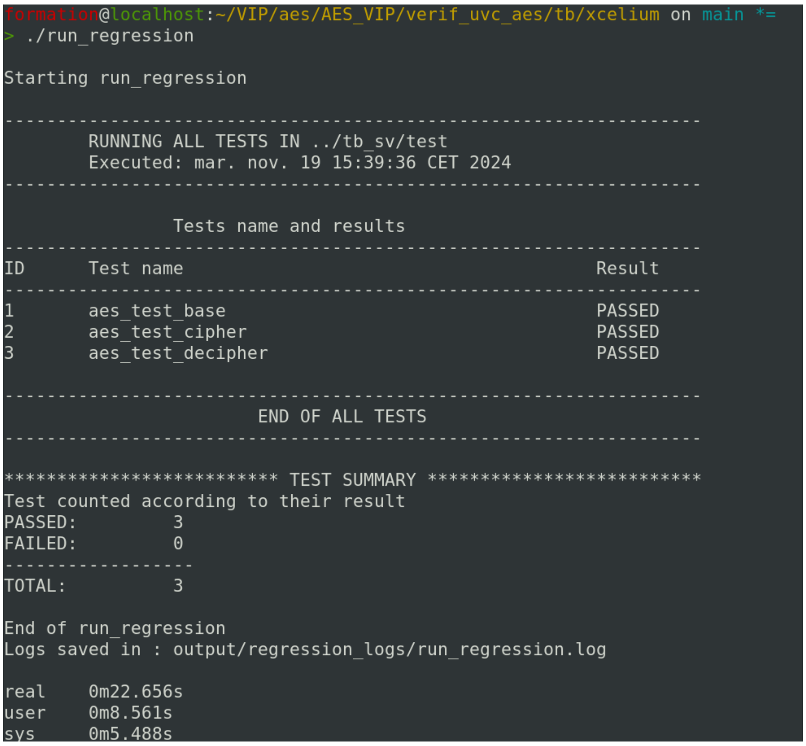
- Go to verif_uvc_aes/tb/xcelium/ directory.
- Use run_regression bash script.

Execution example:

- Run a regression with a regression configuration file:

  `./run_regression`

Logs shown in Figure 5 are stored in verif_uvc_aes/tb/xcelium/output/regression_log/ directory.



```
formation@localhost:~/VIP/aes/AES_VIP/verif_uvc_aes/tb/xcelium on main *=
> ./run_regression

Starting run_regression

------------------------------------------------------------------
        RUNNING ALL TESTS IN ../tb_sv/test
        Executed: mar. nov. 19 15:39:36 CET 2024
------------------------------------------------------------------

                Tests name and results
------------------------------------------------------------------
ID      Test name                                       Result
------------------------------------------------------------------
1       aes_test_base                                   PASSED
2       aes_test_cipher                                 PASSED
3       aes_test_decipher                               PASSED

------------------------------------------------------------------
                        END OF ALL TESTS
------------------------------------------------------------------

************************** TEST SUMMARY **************************
Test counted according to their result
PASSED:         3
FAILED:         0
------------------
TOTAL:          3

End of run_regression
Logs saved in : output/regression_logs/run_regression.log

real    0m22.656s
user    0m8.561s
sys     0m5.488s
```

Figure 5: Regression log example

5. Tests

Feature covered Test name	Cipher	Decipher
aes_test_base	X	X
aes_test_cipher	X	
aes_test_decipher		X

Table 12: Summary of tests used to verify the AES core

These tests will report test passed if they have run all their sequences without any assertion error.

1. aes_test_base

This test does:

- Sends a random number of sequence from its virtual sequencer: **aes_base_seq** which send random sequences.
- In parallel, assert the arst signal a defined number of times to test asynchronous reset functionality.

2. aes_test_cipher

This test does:

- Execute the inherited run_phase but override **aes_base_seq** by **aeb_cipher_seq.**

3. aes_test_decipher

This test does:

- Execute the inherited run_phase but override **aes_base_seq** by **aeb_decipher_seq.**

5.1. UVM Sequences

Name	Description
aes_base_seq	Basic sequence that randomizes a sequence item.
aes_cipher_seq	Cipher sequence that randomizes a sequence item with cipher = 1.
aes_decipher_seq	Decipher sequence that randomizes a sequence item with cipher = 0.

Table 13: Sequence list

Appendix

N/A