



# 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

## Contents

1. Verification strategy .....	5
2. Test bench.....	6
2.1. Block diagram .....	6
2.2. Configuration.....	7
2.3. Interface .....	8
2.4. Sequence item .....	9
2.5. Agent.....	9
2.5.1. Driver .....	10
2.5.2. Monitor .....	11
2.6. Scoreboard.....	12
2.6.1. C Model .....	12
2.7. Coverage.....	13
3. DUT .....	13
4. Database Structure and Simulation Flow .....	14
4.1. Database structure.....	14
4.2. Simulation Flow.....	15
5. Assertions and checks.....	15
6. Tests.....	16
6.1. UVM Sequences .....	16
Appendix.....	17

## List of figures

<i>Figure 1: AES VIP top level block diagram</i> .....	6
<i>Figure 2: Block diagram of compare_to_c_model function</i> .....	12
<i>Figure 3: Block diagram of DUT aes128</i> .....	13
<i>Figure 4: Directory structure of the VIP environment</i> .....	14

## List of tables

<i>Table 1: Configuration class fields</i> .....	7
<i>Table 2: Interface class fields</i> .....	8
<i>Table 3: Sequence item fields</i> .....	9
<i>Table 4: Sequence item constraints</i> .....	9
<i>Table 5: Driver class fields</i> .....	10
<i>Table 6: Driver functions and tasks</i> .....	10
<i>Table 7: Monitor fields</i> .....	11
<i>Table 8: Monitor functions and tasks</i> .....	11
<i>Table 9: C model main function</i> .....	12
<i>Table 10: Coverpoint list</i> .....	13
<i>Table 11: Assertion list</i> .....	15
<i>Table 12: Summary of tests used to verify the AES core</i> .....	16
<i>Table 13: Sequence list</i> .....	16

## 1. Verification strategy

The AES VIP is an UVM environment 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 VIP will follow the steps below:

- The environment will be written in System Verilog.
- An agent 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 VIP 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.

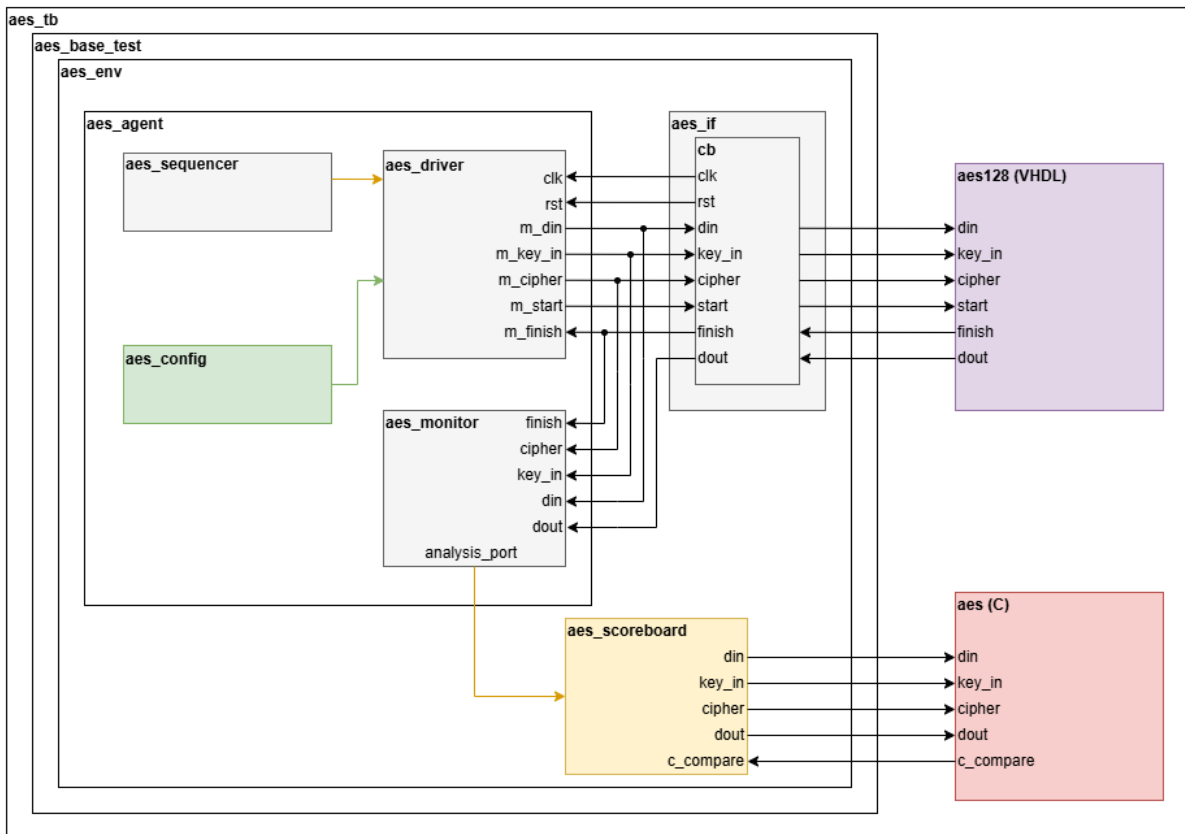


Figure 1: AES VIP top level block diagram

## 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
<b>vif</b>	virtual aes_if	None	Virtual interface
<b>name</b>	string	None	Name of the configuration
<b>General UVM specifications</b>			
<b>is_active</b>	uvm_active_passive_enum	UVM_ACTIVE	Set the agent configuration to active or passive
<b>coverage_enable</b>	bit	0	Enable coverage
<b>checks_enable</b>	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
<b>clk</b>	Wire	Clock, actions are performed on its rising edge
<b>rst</b>	Wire	Synchronous reset, active high. When asserted, set all the signals to its initial state
<b>start</b>	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
<b>din</b>	Wire [127:0]	Data input signal. It carries the 128-bit plaintext data that needs to be encrypted
<b>key_in</b>	Wire [127:0]	Key input signal. It carries the 128-bit encryption key used for the AES algorithm
<b>cipher</b>	Wire	Indicates which operation is performed. If set to 1, it's an encryption; if set to 0, it's a decryption
<b>dout</b>	Wire [127:0]	Data output signal. It carries the 128-bit ciphertext result after the encryption process is completed.
<b>finish</b>	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
<b>m_din</b>	rand bit [127:0]	Data input signal
<b>m_key_in</b>	rand bit [127:0]	Key input signal
<b>m_dout</b>	rand bit [127:0]	Data output signal
<b>m_cipher</b>	rand bit	Encryption or decryption mod signal.
<b>m_delay</b>	rand int	Delay between each operation

*Table 3: Sequence item fields*

Table 4 gives sequence item constraints.

Name	Min value	Max value	Description
<b>max_delay_rate_c</b>	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
<b>General fields</b>		
<b>vif</b>	virtual aes_if	Virtual aes interface
<b>m_config</b>	aes_config	Configuration set by the agent

*Table 5: Driver class fields*

Table 6 gives driver functions and tasks.

Name	Type	Arguments	Return	Description
<b>new</b>	Function	string name, uvm_component parent	/	Constructor
<b>build_phase</b>	Function	uvm_phase phase	void	Build phase
<b>run_phase</b>	Task	uvm_phase phase	void	This task gets a new transaction, and call drive item tasks
<b>do_drive</b>	Task	/	/	This task drives sequence items received to virtual interface
<b>handle_reset</b>	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
<b>vif</b>	virtual aes_if	Virtual AES interface
<b>m_config</b>	aes_config	Configuration set by the agent
<b>analysis_port</b>	uvm_analysis_port #( aes_tx)	Analysis port where monitor writes detected item.
<b>m_trans</b>	aes_tx	Item to receive data from monitor
<b>m_trans_cloned</b>	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
<b>new</b>	Function	string name, uvm_component parent	/	Constructor. Build and connect phase
<b>run_phase</b>	Task	uvm_phase phase	/	Run phase
<b>do_mon</b>	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_fatal` error. To do so, it imports a DPI-C function `compare_to_c_model` from the `aes_model.c` file. Scoreboard 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
<b>Compare_to_c_model</b>	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.

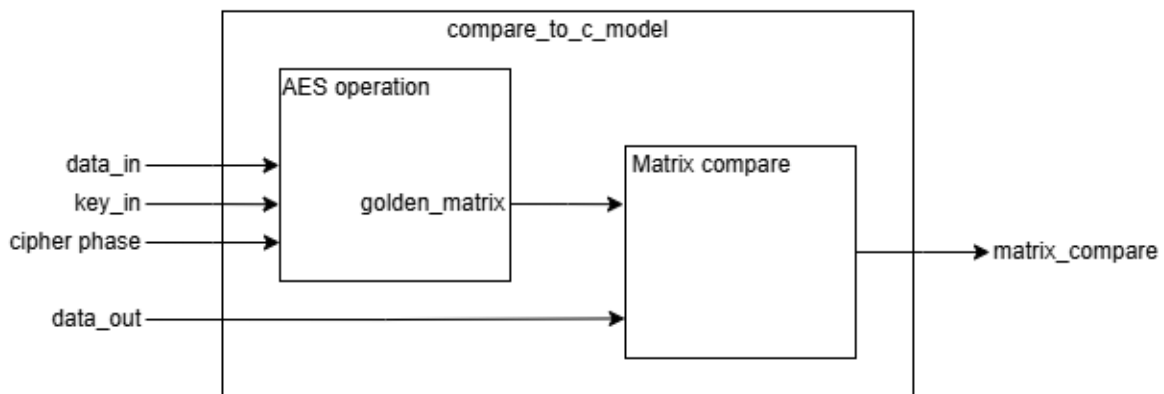


Figure 2: Block diagram of `compare_to_c_model` function

## 2.7. Coverage

Table 10 gives coverpoints checked during coverage in aes\_coverage class.

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

Table 10: Coverpoint list

## 3. DUT

DUT is a VHDL 128 bit AES core IP.

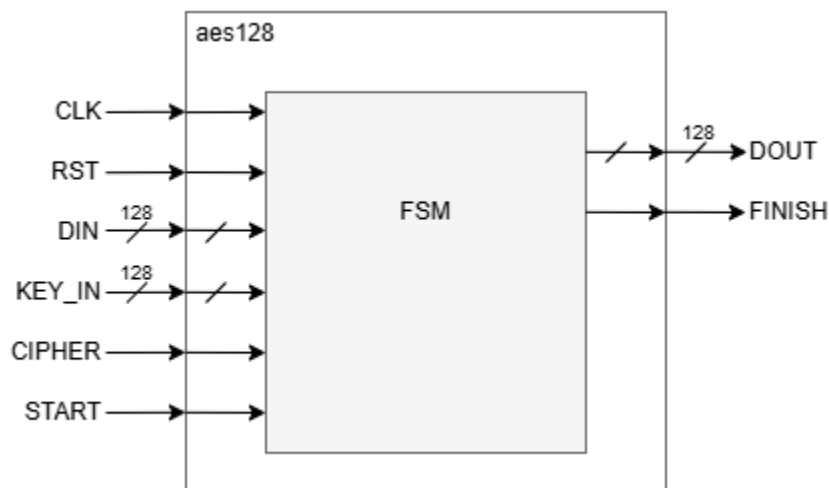


Figure 3: Block diagram of DUT aes128

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.

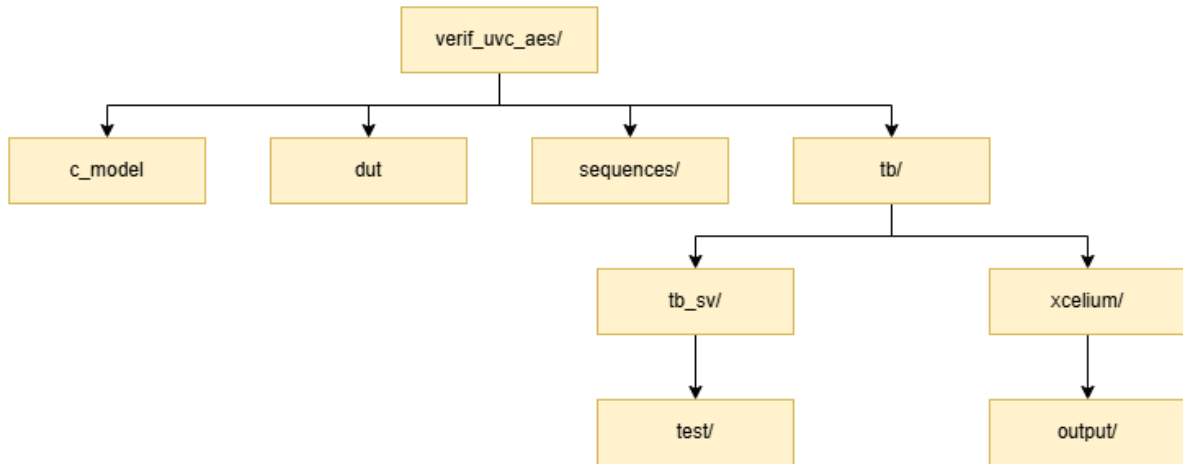


Figure 4: Directory structure of the VIP environment

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:
  - o 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 launch xrun with given arguments.

#### 4.2. Simulation Flow

TOOL REQUIRED: Cadence Incisive

TODO

### 5. Assertions and checks

This section describes the assertions used by the test bench.

Table 11 gives for each assertion its name and its description.

ID	Name	Description
	<b>Assertions located in interface (using interface wires)</b>	
	<b>TBD</b>	

*Table 11: Assertion list*

## 6. Tests

Feature covered	Cipher	Decipher
<b>Test name</b>		
<b>aes_test_base</b>	X	X
<b>atb_test_cipher</b>	X	
<b>atb_test_decipher</b>		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.

### 2. aes\_test\_cipher

This test does:

- Execute the inherited run\_phase but override **aes\_base\_seq** by **atb\_cipher\_seq**.

### 3. aes\_test\_decipher

This test does:

- Execute the inherited run\_phase but override **aes\_base\_seq** by **atb\_decipher\_seq**.

## 6.1. UVM Sequences

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

Table 13: Sequence list



## Appendix

N/A