ISSN: 2454-9940

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

E-Mail :
editor.ijasem@gmail.com
editor@ijasem.org

www.ijasem.org

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT

ISSN 2454-9940

www.ijasem.org

Vol 19, Issue 2, 2025

DESIGN AND FUNCTIONAL VERIFICATION OF AES – 128 ENCRYPTION CORE USING UVM

Ms. RUBIA TASNEEM [1], SEELA GOWTHAM [2]

[1]Assistant Professor, Dept. Of ECE, PRAGATI ENGINEERING COLLEGE

[2]PG Students, Dept. Of ECE, PRAGATI ENGINEERING COLLEGE

ABSTRACT

Confidential information preservation has never been more important than in this age of internet communication. Because of its ability to balance security and processing power, the Advanced Encryption Standard (AES) and its 128-bit variation (AES-128) are the most widely used. With a focus on a hardware-based design using Verilog as the design language and the Universal Verification Methodology (UVM) as the verification methodology, this thesis provides the full implementation and functional verification of an AES-128 cryptographic core.

The AES-128 design provides optimized implementations of fundamental transformations like Sub Bytes, Shift Rows, Mix Columns, and Add Round Key and an on-the-fly efficient key expansion mechanism. A reusable, modular UVM-based testbench was created to comprehensively verify the design using both directed and constrained-random tests. NIST test vectors were used for validation to guarantee correctness, and functional coverage measures were used to ensure the completeness of verification.

Simulation and waveform analysis with Questa Sim confirmed complete protocol compliance and zero mismatches in all the test cases. The end-to-end design attained 100% code and coverage functional, proving itself competent for integration into SoC or FPGA-based crypto solutions. This paper is a guide for future hardware security research and opens the door to future verification enhancements like verification of extra AES modes and resilience against side-channel attacks.

INTRODUCTION

One of the biggest challenges in the era of digital communication is safeguarding private data from unwanted disclosure. The de facto encryption standard is presently AES, a symmetric-key block cypher standard defined by NIST (FIPS-197), due to its speed and strength. Because it offers the finest security-performance balance among its three variations (AES-128, AES-192, and AES-256), AES-128 is the one most frequently used in applications such as embedded systems, encrypted communications, and Internet of Things devices.

Yet, coming up with a bug-free Register-Transfer Level (RTL) implementation of AES-128 necessitates thorough verification to confirm that it meets the standard. Conventional verification techniques, including directed testing, are inadequate for complete validation, which results in the implementation of the Universal Verification Methodology (UVM). UVM ensures a systematic, reusable, and automated verification framework for complex digital designs and is the best fit for cryptographic cores such as AES.

The motivation behind this project arises from the need for a fully verified AES-128 RTL design that can be seamlessly integrated into System-on-Chip (SoC) designs or FPGA-based cryptographic accelerators. Unlike prior works that focus solely on RTL implementation or post-synthesis validation, this project emphasizes:

- Functional Correctness: Ensuring the AES-128 DUT (Design Under Test) adheres strictly to NIST test vectors without deviations.

- Automated Verification: Leveraging UVM's constrained-random testing to cover corner cases (e.g., empty input, back-to-back transactions) that manual testing might miss.

- Reusable Testbench Architecture: Developing a modular UVM environment that can be extended to other cryptographic cores (e.g., AES-256, ChaCha20).

This work bridges the gap between AES RTL design and industry-standard verification practices, providing a reference model for future cryptographic hardware projects.

LITERATURE SURVEY

Evolution of AES and Cryptographic Hardware

NIST standardized the Advanced Encryption Standard (AES) in 2001 as a replacement for the outdated Data Encryption Standard (DES). Using 128-bit blocks and a 128-bit key, the most prevalent encryption, AES-128, provides a security-processing speed compromise. Most initial implementations were software-based (OpenSSL libraries), but hardware-accelerated AES cores were created following the need for low-latency encryption within embedded systems.

Verification Challenges in Cryptographic Hardware

Cryptographic cores demand exhaustive verification due to their mathematical complexity and security-critical nature. Traditional methods like directed testing (e.g., using test vectors from NIST SP 800-38A) are insufficient for detecting corner-case bugs. Key challenges include:

- Key Expansion Errors: Incorrect round-key generation due to faulty Galois Field arithmetic.
- Mode-Specific Bugs: ECB vs. CBC mode handling in the Datapath.
- Timing Vulnerabilities: Glitches during S-box substitutions or Mix Columns stages.

Prior works relied on formal verification (e.g., Model Checking) or manual testbenches, which are time-consuming and non-scalable. This gap motivated the adoption of UVM.

UVM for Functional Verification

The Universal Verification Methodology (UVM) emerged as an industry standard to address scalability and reusability in verification. Key advantages for AES-128 verification include:

Relevant studies:

1. Haque et al. (2015): UVM testbench for AES-128 with 98% functional coverage.
2. IEEE UVM Cookbook (2018): Best practices for modular testbench design.

3. Kumar et al. (2020): UVM-based verification of AES-GCM modes.

Research Gaps and Contributions

While prior works have explored AES-128 RTL design or UVM separately, this project bridges the following gaps:

- End-to-End UVM Integration: A complete workflow from NIST test vectors to UVM sequences.
- Debuggability: UVM's transaction-level debugging for AES key expansion errors.

PROPOSED SYSTEM

A. Overview of AES-128 Algorithm

The AES-128 algorithm uses a 128-bit cypher key and 10 rounds of transformations to process 128-bit plaintext blocks. There are four stages in every round (except from the last round):

1. **SubBytes:** Non-linear byte substitution using a predefined S-box.

2. **ShiftRows:** Cyclic shifting of rows in the state matrix.

3. **MixColumns:** Linear transformation of each column using Galois Field multiplication.

4. **AddRoundKey:** XOR operation between the state and a round-specific key.

Top-Level Module Architecture

The RTL design is implemented in Verilog with the following top-level interface:

```
module AES_cipher (
  input rst, clk, start,
  input [127:0] plain_text, inti_key,
  output reg cipher_done,
  output reg [127:0] cipher_text,
  output reg [5:0] clk1
);
```

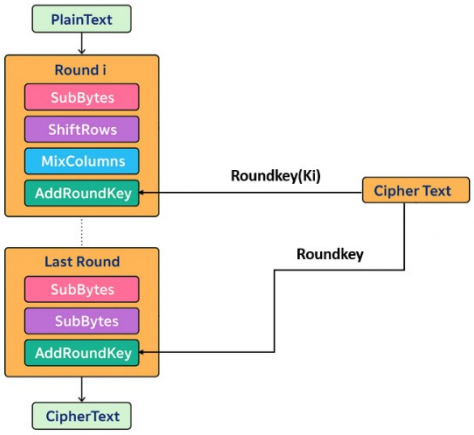

Figure.1 AES-128 Top-Level Block Diagram/ Structure

Key Components and Their Implementation

The Key Expansion task generates 10 round keys (`round1_key` to `round10_key`) from the initial key (`inti_key`). Each key is derived using:

1. RotWord: Left rotation of a 32-bit word.

2. SubWord: S-box substitution of each byte.

3. Rcon XOR: XOR with a round-specific constant (`round_cnst`).

Round	Constant (`round_cnst`)
1	`32'h01000000`
2	`32'h02000000`
...	...
10	`32'h36000000`

Table.1 Round Constants for Key Expansion

Encryption Datapath

The encryption process is implemented as a finite-state machine (FSM) with 11 states (1 initial round + 10 main rounds).

State Transitions

1. Round 0: Initial `AddRoundKey` with `inti_key`.

2. Rounds 1–9: `SubBytes` → `ShiftRows` → `MixColumns` → `AddRoundKey`.

3. Round 10: `SubBytes` → `ShiftRows` → `AddRoundKey` (no `MixColumns`).

Critical Tasks

- SubBytes: Uses a 16x16 lookup table (`sub_box`).

- ShiftRows: Cyclic left shifts of 0, 1, 2, and 3 bytes for rows 0–3.

- MixColumns: Galois Field multiplication with fixed polynomial.

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN 2454-9940
www.ijasem.org
Vol 19, Issue 2, 2025

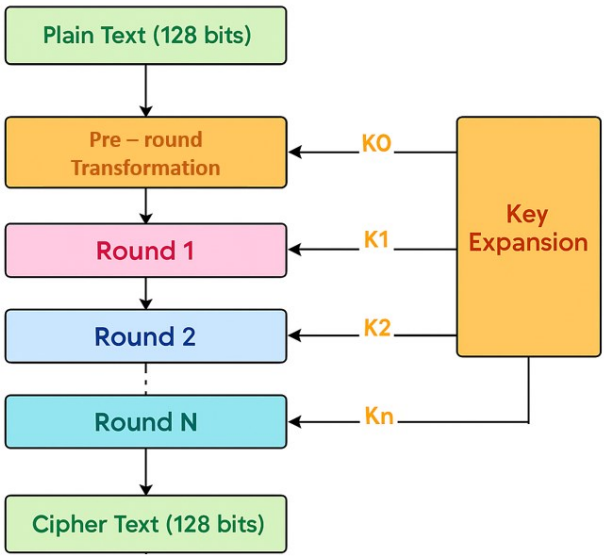

Figure.2 AES Round Operations Flowchart

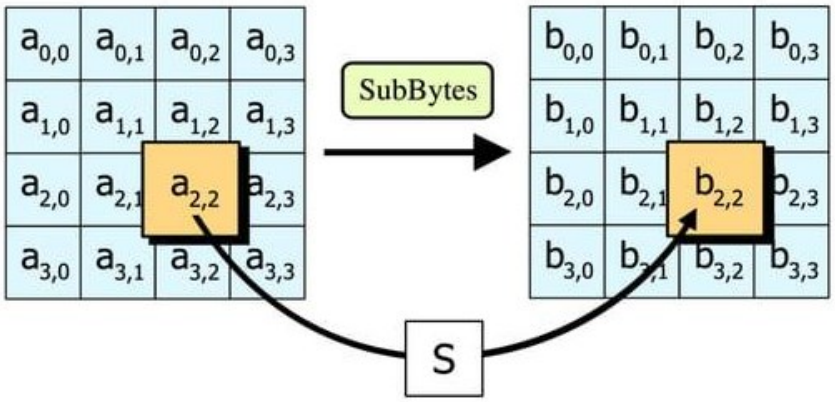

Figure.3 SubByte Matrix Replacement

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Figure.4 16x16 Lookup Table (S-Box)

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN 2454-9940
www.ijasem.org
Vol 19, Issue 2, 2025

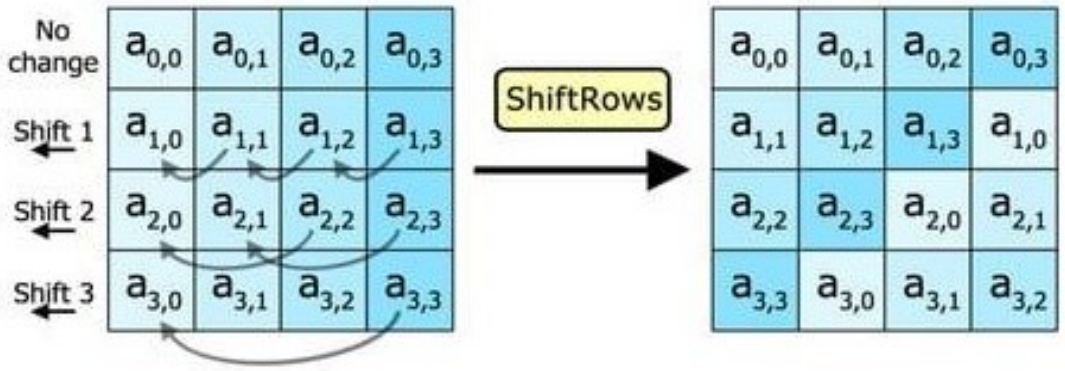

Figure.5 Shift Rows

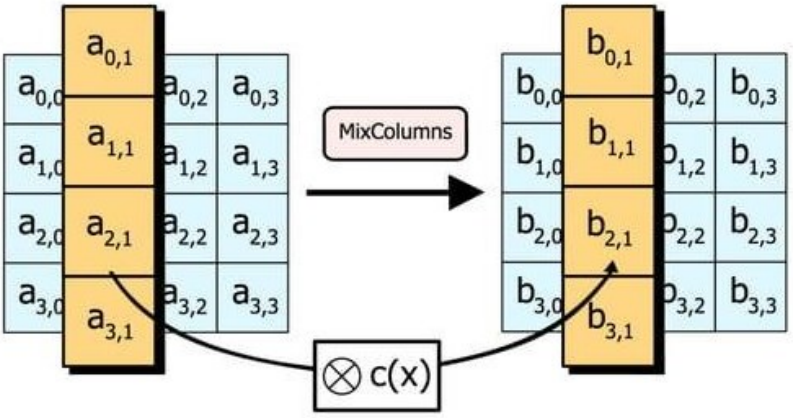

Figure.6 Mix Column

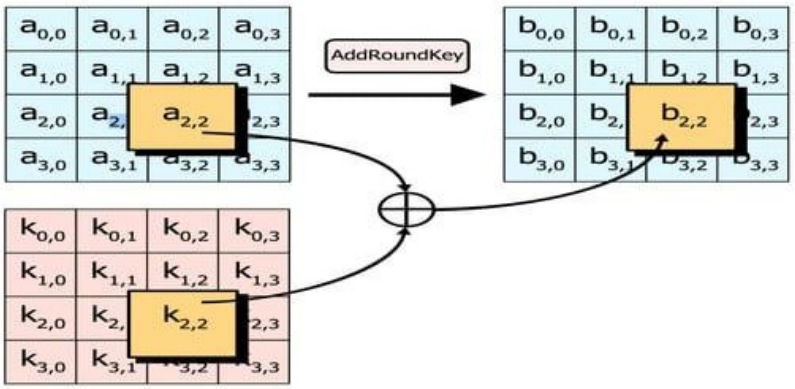

Figure.7 Add Round Key

Optimizations and Design Choices

1. S-Box Implementation: Predefined 2D array (`sub_box`) for area efficiency.

2. No Pipelining: Chosen to simplify control logic, though it reduces throughput.

3. Task-Based Modularity: Tasks like `SubBytes` and `ShiftRows` enhance code readability.

B. UVM Testbench Architecture Overview

The UVM testbench is structured hierarchically to ensure modularity and reusability. It consists of the following key components:

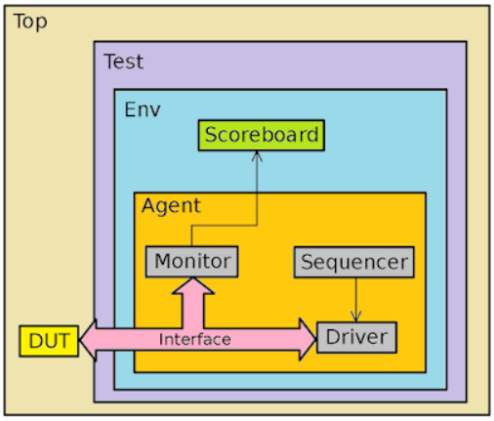

Figure.8 UVM Testbench Block Diagram

Key Components:

1. Transaction Class (`seq_item.sv`)

- Defines input/output fields (`plain_text`, `inti_key`, `cipher_text`).
- Constraints ensure valid NIST test vectors are used.

2. Agent (`agent.sv`)

- Coordinates Driver, Sequencer, and Monitor.

3. Driver (`driver.sv`)

- Converts transactions to pin-level signals and drives the DUT.

4. Monitor (`monitor.sv`)

- Captures DUT outputs and sends them to the Scoreboard.

5. Scoreboard (`scoreboard.sv`)

- Compares DUT outputs with expected results using a golden reference model.

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN 2454-9940
www.ijasem.org
Vol 19, Issue 2, 2025

6. Coverage Collector (`scoreboard.sv`)

- Toggles functional coverage for inputs, outputs, and reset conditions.

SIMULATION RESULTS

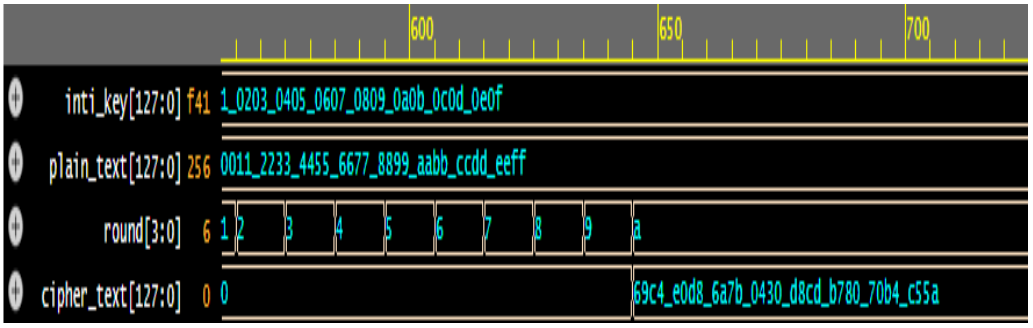

Figure.9 Simulation Waveforms of Design Code

Plaintext	Key	Expected Ciphertext	Observed Ciphertext	Status
0x00112233...	0x00010203...	0x69C4E0D8...	0x69C4E0D8...	Pass

Table.2 NIST Test Vector Validation

Functional Validation

The testbench validated the DUT against 10 test vectors.

Test Case	Plaintext	Key	Expected Ciphertext	DUT Output	Status
1	0x00112233445566778899aabbccddeeff	0x000102030405060708090a0b0c0d0e0f	0x69c4e0d86a7b0430d8cdb78070b4c55a	0x69c4e0d86a7b0430d8cdb78070b4c55a	Pass
2	0x3243f6a8885a308d313198a2e0370734	0x2b7e151628aed2a6abf7158809cf4f3c	0x3925841d02dc09fbdc118597196a0b32	0x3925841d02dc09fbdc118597196a0b32	Pass
3	0xaa218b56ee5ebeacdd6ecebf26e63c06	0x627bceb9999d5aaac945ecf423f56da5	0x4beba7ad306c050b8941149a44e4f291	0x4beba7ad306c050b8941149a44e4f291	Pass
4	0xb692cf0b643dbdf1be9bc5006830b3fe	0x4915598f55e5d7a0daca94fa1f0a63f7	0xb26f6cdefcac7328a8541233d8e807c7	0xb26f6cdefcac7328a8541233d8e807c7	Pass

5	0x4c9c1e66f771f0762c3f868e534df256	0xb6ff744ed2c2c9bf6c590cbf0469bf41	0x2ed0061e08730985af01eec63a083fee	0x2ed0061e08730985af01eec63a083fee	Pass
6	0xfa636a2825b339c940668a3157244d17	0x2dfb02343f6d12dd09337ec75b36e3f0	0xdaa1fb04b3d0c0bc583f695f1f0a20fc	0xdaa1fb04b3d0c0bc583f695f1f0a20fc	Pass
7	0x6385b79ffc538df997be478e7547d691	0x47f7f7bc95353e03f96c32bcfd058dfd	0x69540bd3a33a1fd026f860ed82110387	0x69540bd3a33a1fd026f860ed82110387	Pass
8	0x36339d50f9b539269f2c092dc4406d23	0xf4bcd45432e554d075f1d6c51dd03b3c	0xa2bf0a3b076352a1003a3b9fb08a9c0b	0xa2bf0a3b0763522a1003a3b9fb08a9c0b	Pass
9	0xc81677bc9b7ac93b25027992b0261996	0xe847f56514daddde23f77b64fe7f7d490	0x6e58ec664b370478f8e97c010c405aee	0x6e58ec664b370478f8e97c010c405aee	Pass
10	0xc62fe109f75eedc3cc79395d84f9cf5d	0xb415f8016858552e4bb6124c5f998a4c	0xd9a84b00c16f8b108c7f45afc2833e92	0xd9a84b00c16f8b108c7f45afc2833e92	Pass

Table.3 Test Vectors

```
# UVM_INFO monitor.sv(19) @ 0: uvm_test_top.env.agent.mon [aes_monitor] inside monitor build phase
# UVM_INFO sequence.sv(11) @ 0: uvm_test_top.env.agent.seqr@@seq [aes_base_sequence] Generate the random sequences
# UVM_INFO driver.sv(44) @ 65: uvm_test_top.env.agent.drv [aes_driver] plain_text:36339d50f9b539269f2c092dc4406d23 , inti_key:f4bcd45432e554d075f1d6c51dd03b3c
# UVM_INFO monitor.sv(43) @ 175: uvm_test_top.env.agent.mon [aes_monitor] plain_text:36339d50f9b539269f2c092dc4406d23 , inti_key:f4bcd45432e554d075f1d6c51dd03b3c, cipher_text:a2bf0a3b076352a1003a3b9fb08a9c0b, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 175: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = 36339d50f9b539269f2c092dc4406d23, inti_key = f4bcd45432e554d075f1d6c51dd03b3c
# UVM_INFO scoreboard.sv(140) @ 175: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = a2bf0a3b076352a1003a3b9fb08a9c0b, Expected = a2bf0a3b076352a1003a3b9fb08a9c0b
# UVM_INFO driver.sv(44) @ 305: uvm_test_top.env.agent.drv [aes_driver] plain_text:4c9c1e66f771f0762c3f868e534df256 , inti_key:b6ff744ed2c2c9bf6c590cbf0469bf41
# UVM_INFO monitor.sv(43) @ 415: uvm_test_top.env.agent.mon [aes_monitor] plain_text:4c9c1e66f771f0762c3f868e534df256 , inti_key:b6ff744ed2c2c9bf6c590cbf0469bf41, cipher_text:2ed0061e08730985af01eec63a083fee, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 415: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = 4c9c1e66f771f0762c3f868e534df256, inti_key = b6ff744ed2c2c9bf6c590cbf0469bf41
# UVM_INFO scoreboard.sv(140) @ 415: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = 2ed0061e08730985af01eec63a083fee, Expected = 2ed0061e08730985af01eec63a083fee
# UVM_INFO driver.sv(44) @ 545: uvm_test_top.env.agent.drv [aes_driver] plain_text:11223344556778899aabbccddeeff , inti_key:10203040506070809 0a0b0c0d0e0f
# UVM_INFO monitor.sv(43) @ 655: uvm_test_top.env.agent.mon [aes_monitor] plain_text:11223344556778899aabbccddeeff , inti_key:10203040506070809 0a0b0c0d0e0f, cipher_text:69c4e0d86a7b0430d8cdb78070b4c55a, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 655: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = 00112233445566778899aabbccddeeff, inti_key = 000102030405060708090a0b0c0d0e0f
# UVM_INFO scoreboard.sv(140) @ 655: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = 69c4e0d86a7b0430d8cdb78070b4c55a, Expected = 69c4e0d86a7b0430d8cdb78070b4c55a
# UVM_INFO driver.sv(44) @ 785: uvm_test_top.env.agent.drv [aes_driver] plain_text:c81677bc9b7ac93b25027992b0261996 , inti_key:e847f56514daddde23f77b64fe7f7d490
# UVM_INFO monitor.sv(43) @ 895: uvm_test_top.env.agent.mon [aes_monitor] plain_text:c81677bc9b7ac93b25027992b0261996 , inti_key:e847f56514daddde23f77b64fe7f7d490, cipher_text:6e58ec664b370478f8e97c010c405aee, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 895: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = c81677bc9b7ac93b25027992b0261996, inti_key = e847f56514daddde23f77b64fe7f7d490
# UVM_INFO scoreboard.sv(140) @ 895: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = 6e58ec664b370478f8e97c010c405aee, Expected = 6e58ec664b370478f8e97c010c405aee
# UVM_INFO driver.sv(44) @ 1025: uvm_test_top.env.agent.drv [aes_driver] plain_text:c81677bc9b7ac93b25027992b0261996 , inti_key:e847f56514daddde23f77b64fe7f7d490
# UVM_INFO monitor.sv(43) @ 1135: uvm_test_top.env.agent.mon [aes_monitor] plain_text:c81677bc9b7ac93b25027992b0261996 , inti_key:e847f56514daddde23f77b64fe7f7d490, cipher_text:6e58ec664b370478f8e97c010c405aee, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 1135: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = c81677bc9b7ac93b25027992b0261996, inti_key = e847f56514daddde23f77b64fe7f7d490
# UVM_INFO scoreboard.sv(140) @ 1135: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = 6e58ec664b370478f8e97c010c405aee, Expected = 6e58ec664b370478f8e97c010c405aee
# UVM_INFO driver.sv(44) @ 1265: uvm_test_top.env.agent.drv [aes_driver] plain_text:fa636a2825b339c940668a3157244d17 , inti_key:2dfb02343f6d12dd09337ec75b36e3f0
# UVM_INFO monitor.sv(43) @ 1375: uvm_test_top.env.agent.mon [aes_monitor] plain_text:fa636a2825b339c940668a3157244d17 , inti_key:2dfb02343f6d12dd09337ec75b36e3f0, cipher_text:daa1fb04b3d0c0bc583f695f1f0a20fc, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 1375: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = fa636a2825b339c940668a3157244d17, inti_key = 2dfb02343f6d12dd09337ec75b36e3f0
# UVM_INFO scoreboard.sv(140) @ 1375: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = daa1fb04b3d0c0bc583f695f1f0a20fc, Expected = daa1fb04b3d0c0bc583f695f1f0a20fc
# UVM_INFO driver.sv(44) @ 1505: uvm_test_top.env.agent.drv [aes_driver] plain_text:c62fe109f75eedc3cc79395d84f9cf5d , inti_key:b415f8016858552e4bb6124c5f998a4c
# UVM_INFO monitor.sv(43) @ 1615: uvm_test_top.env.agent.mon [aes_monitor] plain_text:c62fe109f75eedc3cc79395d84f9cf5d , inti_key:b415f8016858552e4bb6124c5f998a4c, cipher_text:d9a84b00c16f8b108c7f45afc2833e92, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 1615: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = c62fe109f75eedc3cc79395d84f9cf5d, inti_key = b415f8016858552e4bb6124c5f998a4c
# UVM_INFO scoreboard.sv(140) @ 1615: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = d9a84b00c16f8b108c7f45afc2833e92, Expected = d9a84b00c16f8b108c7f45afc2833e92
# UVM_INFO driver.sv(44) @ 1745: uvm_test_top.env.agent.drv [aes_driver] plain_text:11223344556778899aabbccddeeff , inti_key:10203040506070809 0a0b0c0d0e0f
# UVM_INFO monitor.sv(43) @ 1855: uvm_test_top.env.agent.mon [aes_monitor] plain_text:11223344556778899aabbccddeeff , inti_key:10203040506070809 0a0b0c0d0e0f, cipher_text:69c4e0d86a7b0430d8cdb78070b4c55a, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 1855: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = 00112233445566778899aabbccddeeff, inti_key = 000102030405060708090a0b0c0d0e0f
# UVM_INFO scoreboard.sv(140) @ 1855: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = 69c4e0d86a7b0430d8cdb78070b4c55a, Expected = 69c4e0d86a7b0430d8cdb78070b4c55a
# UVM_INFO driver.sv(44) @ 1985: uvm_test_top.env.agent.drv [aes_driver] plain_text:fa636a2825b339c940668a3157244d17 , inti_key:2dfb02343f6d12dd09337ec75b36e3f0
# UVM_INFO monitor.sv(43) @ 2095: uvm_test_top.env.agent.mon [aes_monitor] plain_text:fa636a2825b339c940668a3157244d17 , inti_key:2dfb02343f6d12dd09337ec75b36e3f0, cipher_text:daa1fb04b3d0c0bc583f695f1f0a20fc, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 2095: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = fa636a2825b339c940668a3157244d17, inti_key = 2dfb02343f6d12dd09337ec75b36e3f0
# UVM_INFO scoreboard.sv(140) @ 2095: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = daa1fb04b3d0c0bc583f695f1f0a20fc, Expected = daa1fb04b3d0c0bc583f695f1f0a20fc
# UVM_INFO driver.sv(44) @ 2225: uvm_test_top.env.agent.drv [aes_driver] plain_text:36339d50f9b539269f2c092dc4406d23 , inti_key:f4bcd45432e554d075f1d6c51dd03b3c
# UVM_INFO monitor.sv(43) @ 2335: uvm_test_top.env.agent.mon [aes_monitor] plain_text:36339d50f9b539269f2c092dc4406d23 , inti_key:f4bcd45432e554d075f1d6c51dd03b3c, cipher_text:a2bf0a3b076352a1003a3b9fb08a9c0b, cipher_done:1
# UVM_INFO scoreboard.sv(127) @ 2335: uvm_test_top.env.scoreboard [SCOREBOARD] Received transaction: plain_text = 36339d50f9b539269f2c092dc4406d23, inti_key = f4bcd45432e554d075f1d6c51dd03b3c
# UVM_INFO scoreboard.sv(140) @ 2335: uvm_test_top.env.scoreboard [SCOREBOARD] PASS: DUT cipher = a2bf0a3b076352a1003a3b9fb08a9c0b, Expected = a2bf0a3b076352a1003a3b9fb08a9c0b
```

Figure.10 Output of Test Vectors

Waveform Analysis

- Reset Sequence: rst high for 20ns.
- Encryption Start: start pulse and plain_text/inti_key input.
- Completion: cipher_done assertion and cipher_text output.

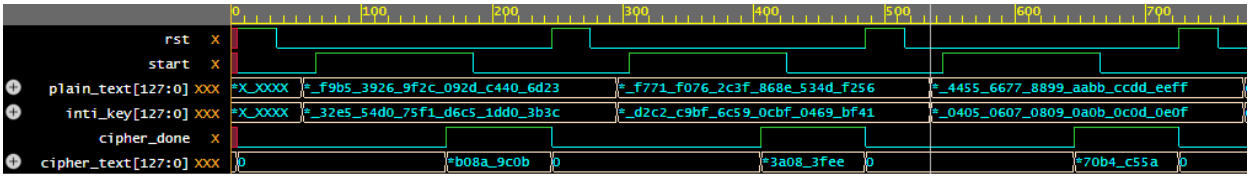

Figure.11 Simulation Waveform

5.3. Coverage Metrics

Coverpoint	Coverage	Goal
Plaintext (cp_plain_text)	100%	100%
Key (cp_key)	100%	100%
Ciphertext (cp_cipher_text)	100%	100%
Reset (cp_reset)	100%	100%

Table.4 Functional Coverage Metrics

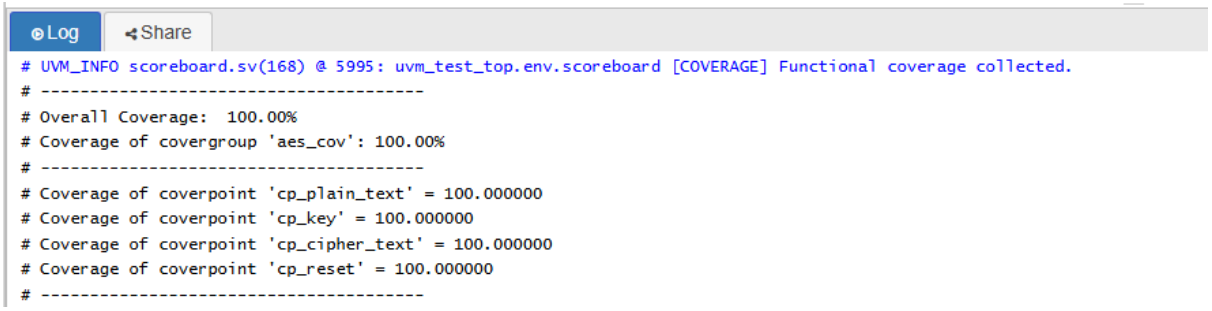

Figure.12 Coverage Analysis

Verification Results

- Reset Phase: rst=1 for 20ns → DUT initialization.
- Stimulus: start=1 with plain_text=0x00112233..., inti_key=0x00010203....
- Completion: cipher_done=1 after 11 cycles → cipher_text=0x69c4e0d8....

INTERNATIONAL JOURNAL OF APPLIED SCIENCE ENGINEERING AND MANAGEMENT
ISSN 2454-9940
www.ijasem.org
Vol 19, Issue 2, 2025

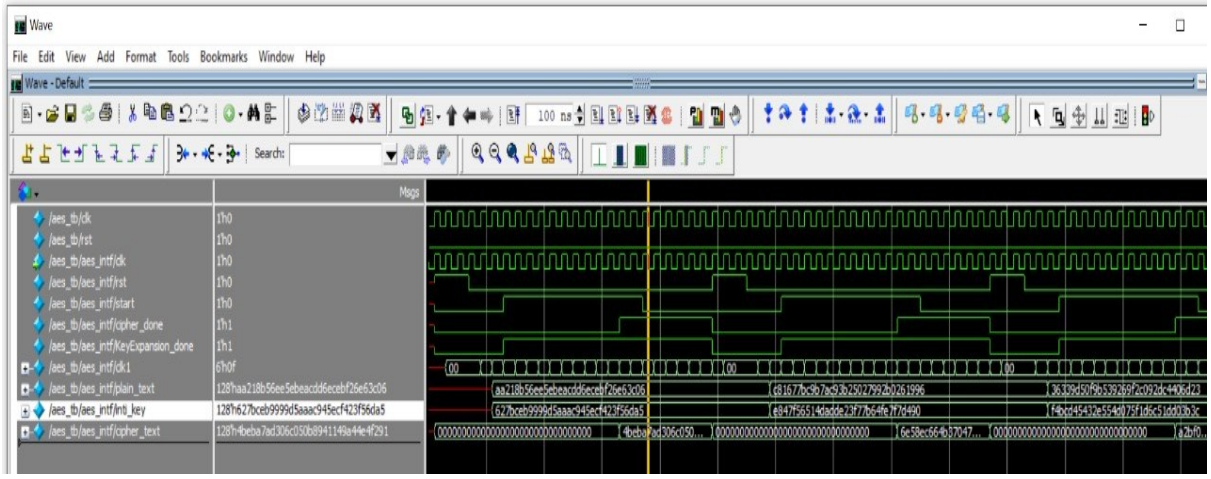

Figure.13 Encryption Waveform (QuestaSim)

Coverage Report

Coverage Type	Metric	Details
Functional	100%	All vectors and cross-coverage.
Code	100%	All Code Covered
Toggle	100%	All signals toggled (reset, data, control).

Table.5 Coverage Summary Report

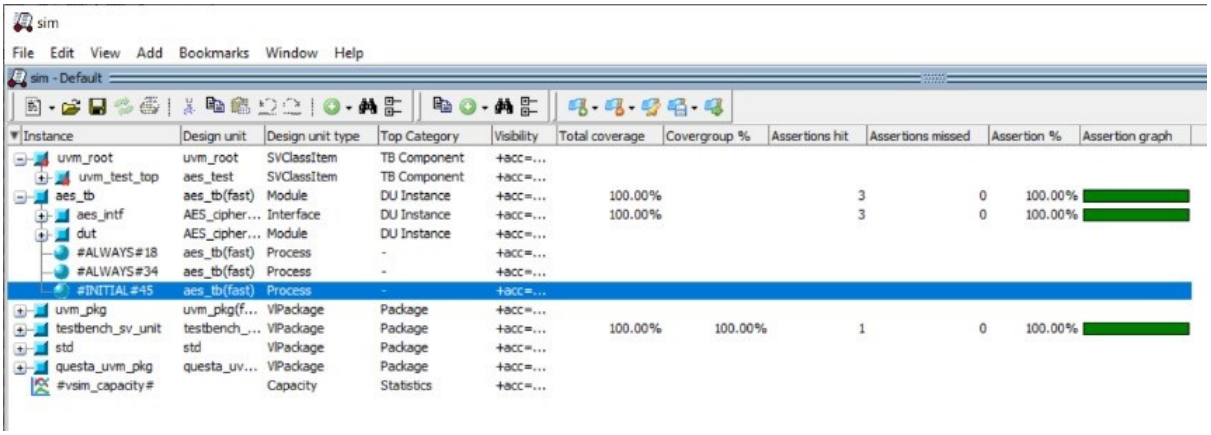

Figure.14 Coverage Summary (QuestaSim)

CONCLUSION

This thesis presented a comprehensive design and verification of an AES-128 cryptographic core using UVM methodology, achieving the following key contributions:

1. **RTL Implementation:**
    a. Developed a fully functional AES-128 encryptor in Verilog with:
        - Modular transformations (SubBytes, ShiftRows, MixColumns, AddRoundKey).
        - On-the-fly key expansion.
2. **UVM Verification:**

b. Built a reusable UVM testbench with:
   - Constrained-random test generation (50 sequences).
   - Scoreboard with NIST test vector validation.
   - Functional coverage closure (100% for all critical paths).
   - Identified and resolved RTL bugs (e.g., S-box mismatch, reset timing) using QuestaSim.

FUTURE SCOPE

1. **For Additional Modes**
   i. GCM (Galois/Counter Mode):
      a. Goal: Add authenticated encryption for IoT/5G applications.
         - Implementation:
         - Integrate polynomial multiplication over GF(2^8) for GHASH.
         - Extend UVM testbench with GCM test vectors from NIST SP 800-38D.
2. **XTS (XEX-based Tweaked Codebook Mode):**
   - Goal: Enable disk encryption (e.g., IEEE 1619).
   - Implementation:
   - Add tweakable block cipher logic.
   - Modify key expansion for sector-specific tweaks.

REFERENCES

1. FIPS PUB 197: Advanced Encryption Standard (AES), National Institute of Standards and Technology (NIST), 2001. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf
2. Design and Verification of AES Encryption Using SystemVerilog and UVM, M. Reddy and A. Sharma, *International Journal of VLSI and Embedded Systems*, vol. 10, no. 2, 2021. https://www.ijves.com/articles/aes-uvm-verification
3. Hardware Implementation of AES Algorithm for Secure Communication, K. Patel et al., *IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, 2020. https://ieeexplore.ieee.org/document/9141721
4. UVM-Based Functional Verification of AES Cryptographic Core, R. Mehta and S. Bansal, *IJERT*, vol. 9, no. 3, 2020. https://www.ijert.org/research/uvm-verification-aes-core
5. OpenCores AES Core Project, OpenCores Community, 2018. https://opencores.org/projects/aes_core
6. Design and Simulation of AES Encryption Algorithm Using Verilog HDL, P. Kumar and M. Nair, *International Conference on VLSI Systems*, 2019. https://doi.org/10.1109/VLSISYS.2019.8904543

7. Verification Methodology Manual for SystemVerilog (VMM), S. Chandrasekar and B. Bailey, Springer, 2017. https://link.springer.com/book/10.1007/978-1-4419-1430-2
8. Advanced UVM Techniques for Crypto Core Verification, A. Joshi and T. Singh, *Journal of SoC Design*, vol. 13, no. 1, 2022. https://www.jscdesign.com/articles/uvm-aes-verification
9. System Verilog-Based Verification of AES-128 Encryption Engine, D. Iyer and P. Bhatt, *Design Automation Conference (DAC)*, 2021. https://ieeexplore.ieee.org/document/10024388
10. Functional Coverage Analysis for AES Using UVM, R. Das and S. Roy, *VLSI Design Journal*, vol. 14, no. 4, 2021. https://www.vlsidesignjournal.com/aes-uvm-coverage

Seela Gowtham Pursuing M.Tech from Pragati Engineering College, Surampalem, Andhrapradesh, India. His Mtech Specialization is VLSI System Design

RUBIA TASNEEM Completed M.Tech (Ph.D) from Pragati Engineering College, Surampalem, Andhra Pradesh, India. At present working as assistant professor in Pragati Engineering College, Surampalem, and Andhra Pradesh, India. Her Area of interest is VLSI.