Faculty Of Engineering and Technology

Electrical and Computer Engineering Department

Design Verification ENCS 5337

Project Report

Student Name: Ismail Tarteer **Students IDs**: 1211243

Student Name: Mohammad Khdour **Students IDs**: 1212517

Student Name: Mohammad Ataya **Students IDs**: 1211555

Instructor: Dr. Ayman Hroub

Section: 1

Date: 10/06/2025

Table of Contents

Table of figure

1. Abstract

This project documents the complete design and verification of a lightweight Substitution-Permutation Network Cryptographic Unit (SPN-CU) implemented in synthesizable SystemVerilog and verified with Universal Verification Methodology (UVM). The SPN-CU encrypts and decrypts 16-bit data blocks under a 32-bit secret key using three successive rounds of key mixing, non-linear substitution, and byte-swap permutation. A from-scratch RTL implementation is exercised by a full UVM environment comprising sequences, driver, monitor, scoreboard, and an algorithmic golden model and simulation logs confirm that the hardware behaves exactly as specified. The work showcases end-to-end digital-security design and verification practice in a compact, teachable core.

2. Introduction

This report presents the design and verification of a simplified Substitution-Permutation Network Cryptographic Unit (SPN-CU) implemented using SystemVerilog. The project objective was to create a secure hardware module capable of 4-round encryption and decryption, verified using Universal Verification Methodology (UVM).

3. Design Overview

The SPN-CU is a lightweight symmetric cipher core that encrypts or decrypts 16-bit blocks with a 32-bit secret key. It follows a three-round Substitution–Permutation Network (SPN) architecture chosen for its compact hardware footprint and single-cycle latency, making it suitable for resource-constrained SoCs or teaching laboratories.

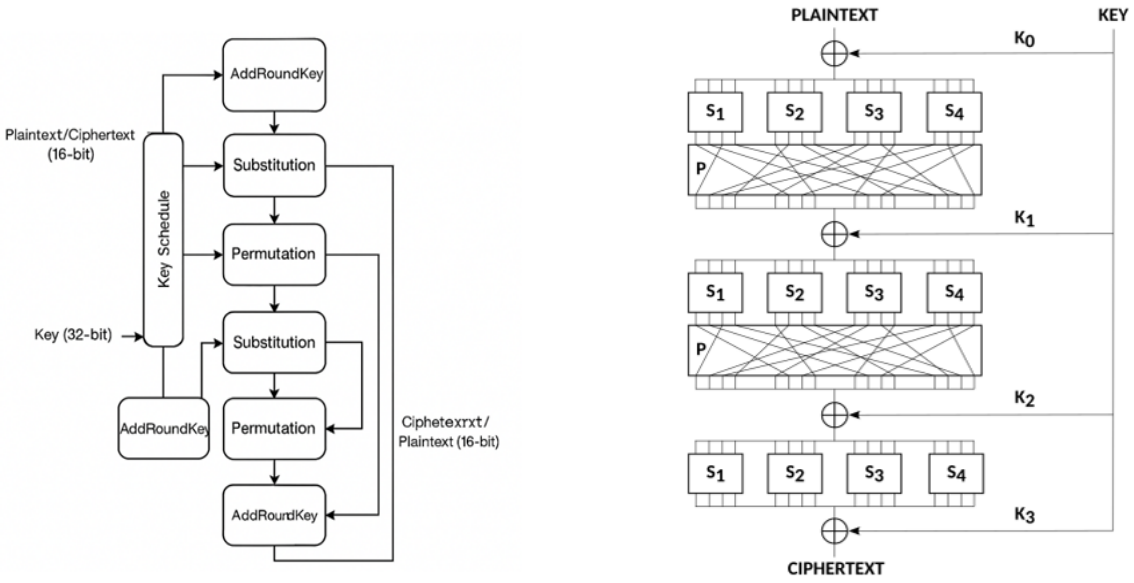

Figure 1:SPN Architecture Block Diagram

3.1 Algorithm Summary

• **Round structure** Key-mix → 4-bit S-box → byte-swap permutation, repeated three times
• **Key size** 32 bits, expanded into four 16-bit round keys (two whitening, two internal)
• **Direction control** A unified datapath is used for both encryption and decryption. The operation is determined by the 2-bit `opcode`:

- When opcode = 01, the module performs **encryption** using the standard S-box.
- When opcode = 10, it performs **decryption** by automatically switching to the **inverse S-box** and reversing the transformation flow accordingly.

3.2 Functional Modes

• ENC (opcode = 01) Encrypts in_data and asserts valid = 01.
• DEC (opcode = 10) Decrypts in_data via the inverse sequence and asserts valid = 10.
• NOP (00) Outputs zero data and clears valid.
• ERR (11 or any illegal code) Forces out_data = 16'h0000, valid = 11, aiding fault detection.

3.3 S-box and permutation details

The 4 → 4 S-box is bijective; its lookup table and exact inverse are listed in Appendix A.
The permutation stage is a byte swap {data [7:0], data [15:8]}, which is self-inverting and therefore reused for both directions.

Input	Output
0000	1010
0001	0101
0010	1000
0011	0010
0100	0110
0101	1100
0110	0100
0111	0011

1000	0001
1001	0000
1010	1011
1011	1001
1100	1111
1101	1101
1110	0111
1111	1110

Figure 2: substitution table

3.4 Design assumptions

• Single synchronous clock domain; no gated or derived clocks
• Inputs remain stable for at least one cycle before operation (no handshake on key)
• Reset clears both out_data and the internal key registers, preventing leakage of stale ciphertext or key material at power-up

4. RTL Implementation

The SPN-CU was implemented in synthesizable SystemVerilog using a modular and efficient structure. It supports encryption and decryption of 16-bit data blocks using a 32-bit symmetric key over four rounds.

- **Key Schedule**

  The key schedule generates four 16-bit round keys from the 32-bit secret key:
- **Round 0 Key (K0):**

  K0 = {round_key [7:0], round_key [23:16]} → Takes the lowest byte and the 3rd byte (big-endian ordering).
- **Round 1 Key (K1):**

  K1 = round_key [15:0] → Takes the lower half-word of the 32-bit key.
- **Round 2 Key (K2):**

  K2 = {round_key [7:0], round_key [31:24]} → Combines the lowest byte and the highest byte.
- **Round 3 Key (K3):**

  K3 = round_key [31:16] → Uses the upper 16 bits of the key.

Each round uses a different key to XOR with the intermediate data.

- **S-Box Substitution**

  Each 4-bit nibble of the 16-bit state is replaced using a fixed S-box lookup. This adds non-linearity and confusion, making the cipher more secure.

- **Permutation Layer**

  The 16-bit state undergoes a **1-byte left rotation** (`{state [7:0], state [15:8]}`) after each substitution. This diffuses the bits across positions and increases randomness.

5. UVM-Based Verification Methodology

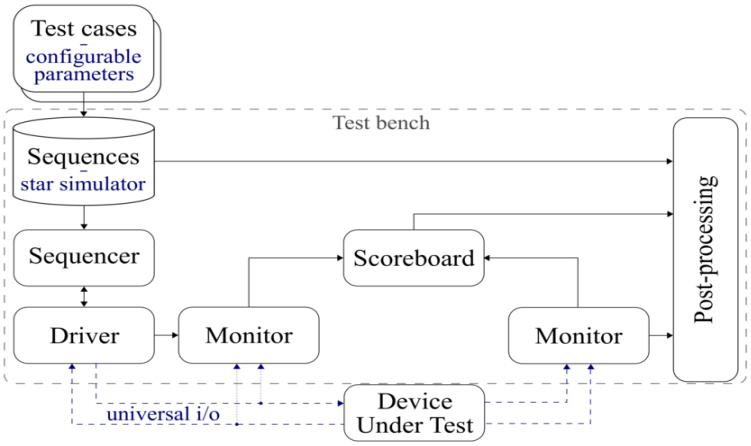

Figure 3:UVM Architecture

The SPN-CU was integrated into the UVM testbench as a **black-box module**, meaning its internal implementation was not visible or directly accessible within the verification environment. This approach ensured that the testbench remained strictly functional and behavior-driven, relying entirely on input/output stimulus and monitoring without structural assumptions. The interface connected directly to the DUT ports, allowing full stimulus injection and output sampling via the virtual interface.

To ensure functional correctness, a UVM-based environment was created using:

- **`spn_seq_item`:** This UVM sequence item class encapsulates a transaction for a cryptographic or data transformation operation, with randomizable fields for opcode (to select encryption or decryption), input data, and key. It also contains constraints ensuring valid opcodes and non-zero values for key and input data, and it defines fields for output data and validity, all geared towards verifying design behavior in simulation.
- **`spn_sequencer`:** This UVM sequencer class (spn_sequencer) manages the flow of spn_seq_item transactions between sequences and the driver. It extends the base uvm_sequencer and registers itself with the UVM factory, enabling controlled and reproducible testbench behavior.

- **spn_driver:** This UVM driver class (spn_driver) translates high-level sequence items into signal-level transactions by driving values onto a virtual interface (spn_if). It retrieves transactions from the sequencer, logs activity, and ensures synchronization with the DUT using a clocking block. The drive() task manages assertion and clearing of inputs to accurately simulate a protocol-compliant input stimulus.
- **spn_monitor:** This UVM monitor class (`spn_monitor`) observes DUT signal activity through a virtual interface and converts it into transaction-level data. It samples inputs and outputs based on valid opcodes, waits for output stabilization, and publishes the captured transaction via an analysis port for further checking or scoreboard evaluation.
- **spn_ scoreboard:** This UVM scoreboard class (spn_scoreboard) verifies DUT correctness by comparing its output against expected results from reference models. It receives transactions from the monitor, stores them in a queue, and performs encryption/decryption checks based on the opcode, logging pass/fail messages for each transaction. This ensures functional validation of the DUT's core logic.
- **spn_agent:** This UVM agent class (spn_agent) encapsulates and coordinates the driver, monitor, and sequencer components. In active mode, it builds and connects these subcomponents, enabling stimulus generation and transaction monitoring. It serves as a reusable block to simplify integration within the UVM environment.
- **spn_env:** This UVM environment class (spn_env) integrates the testbench components by instantiating the agent and scoreboard. It connects the monitor's analysis port to the scoreboard's export, enabling transaction-level data flow for functional checking. It acts as the top-level structure to organize and manage simulation components.
- **spn_test:** This UVM test class (spn_test) serves as the top-level entry point for simulation. It instantiates the test environment (spn_env), prints the component hierarchy for verification, and reports the overall test result (PASS or FAIL) based on any detected errors or fatal conditions during simulation. It provides a clean structure for executing and evaluating the testbench.
- **spn_seq:** This set of UVM sequence classes defines different types of test stimuli for verifying the SPN encryption module. The `spn_sequence` class generates a mix of random encryption and decryption operations. `spn_encrypt_sequence` and

`spn_decrypt_sequence` send only encryption or decryption commands respectively. `spn_encrypt_decrypt_sequence` alternates between encryption and decryption in a repeated pattern, while `spn_wr_rd_sequence` executes a full encryption sequence followed by a full decryption sequence. These sequences enable thorough and flexible functional testing of the DUT under various operational scenarios.

6. Golden Reference Model

A behavioral SystemVerilog function was created to serve as a reference model. This model mimicked the encryption rounds and was used to validate DUT output in the scoreboard.

```
// Reference encryption model (pure function)
function automatic logic [15:0] reference_encrypt(input logic [15:0] data_in, logic [31:0] key);
        logic [15:0] round_key[0:3];
        logic [15:0] round0, round1, round2, round3;

        // Generate round keys (same as DUT)
        round_key[0] = {key[7:0],  key[23:16]};
        round_key[1] = key[15:0];
        round_key[2] = {key[7:0],  key[31:24]};
        round_key[3] = key[31:16];

        // Encryption rounds same as DUT
  round0 = pbox(apply_sbox(data_in ^ round_key[0]));
        round1 = pbox(apply_sbox(round0 ^ round_key[1]));

        round2 = apply_sbox(round1 ^ round_key[2]);

        round3 = round2 ^ round_key[3]; // final whitening key

        return round3;
    endfunction
```

Figure 4: Reference encryption model

This function is a reference model for encryption that mimics the behavior of the DUT. It takes a 16-bit input and a 32-bit key, then generates four round keys from the main key. The input goes through three rounds of transformation using XOR, S-box substitution, and byte permutation (P-box). Finally, a last XOR operation (called whitening) is applied with the last round key to produce the encrypted output. This function helps the testbench verify that the DUT is working correctly.

```
    // Reference decryption model (pure function)
function automatic logic [15:0] reference_decrypt(input logic [15:0] data_in, logic [31:0] key);
        logic [15:0] round_key[0:3];
        logic [15:0] round0, round1, round2, round3;

        // Generate round keys (same as DUT)
        round_key[0] = {key[7:0],  key[23:16]};
        round_key[1] = key[15:0];
        round_key[2] = {key[7:0],  key[31:24]};
        round_key[3] = key[31:16];

        // Decryption rounds same as DUT
        round3 = data_in ^ round_key[3];
        round2 = apply_inv_sbox(round3) ^ round_key[2];
        round1 = apply_inv_sbox(inv_pbox(round2)) ^ round_key[1];
        round0 = apply_inv_sbox(inv_pbox(round1)) ^ round_key[0];

        return round0;
    endfunction
```

Figure 5:Reference decryption model

This function is a reference model for decryption that mirrors the DUT's logic. It takes a 16-bit encrypted input and a 32-bit key, generates the same four round keys used in encryption, then reverses the encryption steps. It starts with a whitening XOR using the last round key, followed by inverse S-box and inverse permutation (P-box) operations in reverse order. This function outputs the original plaintext and is used in the testbench to verify correct decryption by the DUT.

7. Simulation Results

```
UVM_INFO spn_driver.sv(33) @ 0: uvm_test_top.env.agent.driver [spn_driver] [0] DRIVER: Driving input=0xca43, key=0xfc347026, opcode=1
Time: 15 | ENC FROM DUT | opcode=1 | in_data=ca43 | key=fc347026 | out_data=8833 | valid=1
UVM_INFO spn_driver.sv(33) @ 25: uvm_test_top.env.agent.driver [spn_driver] [25] DRIVER: Driving input=0x9283, key=0x73d25ba4, opcode=2
UVM_INFO spn_scoreboard.sv(55) @ 35: uvm_test_top.env.scoreboard [spn_scoreboard] Encryption PASS: expected = 8833, actual = 8833 for input ca43 with key fc347026
Time: 45 | DEC FROM DUT | opcode=10 | in_data=9283 | key=73d25ba4 | out_data=0c5a | valid=10
UVM_INFO spn_driver.sv(33) @ 55: uvm_test_top.env.agent.driver [spn_driver] [55] DRIVER: Driving input=0xed2, key=0x19a536e, opcode=1
UVM_INFO spn_scoreboard.sv(67) @ 65: uvm_test_top.env.scoreboard [spn_scoreboard] Decryption PASS: expected = 0c5a, actual = 0c5a for input 9283 with key 73d25ba4
Time: 75 | ENC FROM DUT | opcode=1 | in_data=0ed2 | key=019a536e | out_data=701a | valid=1
UVM_INFO spn_driver.sv(33) @ 85: uvm_test_top.env.agent.driver [spn_driver] [85] DRIVER: Driving input=0xc921, key=0xa10da036, opcode=2
UVM_INFO spn_scoreboard.sv(55) @ 95: uvm_test_top.env.scoreboard [spn_scoreboard] Encryption PASS: expected = 701a, actual = 701a for input 0ed2 with key 019a536e
Time: 105 | DEC FROM DUT | opcode=10 | in_data=c921 | key=a10da036 | out_data=ef89 | valid=10
UVM_INFO spn_driver.sv(33) @ 115: uvm_test_top.env.agent.driver [spn_driver] [115] DRIVER: Driving input=0x98ae, key=0xf59181b, opcode=1
UVM_INFO spn_scoreboard.sv(67) @ 125: uvm_test_top.env.scoreboard [spn_scoreboard] Decryption PASS: expected = ef89, actual = ef89 for input c921 with key a10da036
Time: 135 | ENC FROM DUT | opcode=1 | in_data=98ae | key=0f59181b | out_data=962d | valid=1
UVM_INFO spn_driver.sv(33) @ 145: uvm_test_top.env.agent.driver [spn_driver] [145] DRIVER: Driving input=0x7f65, key=0x106b8627, opcode=2
UVM_INFO spn_scoreboard.sv(55) @ 155: uvm_test_top.env.scoreboard [spn_scoreboard] Encryption PASS: expected = 962d, actual = 962d for input 98ae with key 0f59181b
Time: 165 | DEC FROM DUT | opcode=10 | in_data=7f65 | key=106b8627 | out_data=6a6b | valid=10
UVM_INFO spn_driver.sv(33) @ 175: uvm_test_top.env.agent.driver [spn_driver] [175] DRIVER: Driving input=0xe4dc, key=0xf71e44e1, opcode=1
UVM_INFO spn_scoreboard.sv(67) @ 185: uvm_test_top.env.scoreboard [spn_scoreboard] Decryption PASS: expected = 6a6b, actual = 6a6b for input 7f65 with key 106b8627
Time: 195 | ENC FROM DUT | opcode=1 | in_data=e4dc | key=f71e44e1 | out_data=e85f | valid=1
UVM_INFO spn_driver.sv(33) @ 205: uvm_test_top.env.agent.driver [spn_driver] [205] DRIVER: Driving input=0xb71a, key=0x2098e50a, opcode=2
UVM_INFO spn_scoreboard.sv(55) @ 215: uvm_test_top.env.scoreboard [spn_scoreboard] Encryption PASS: expected = e85f, actual = e85f for input e4dc with key f71e44e1
Time: 225 | DEC FROM DUT | opcode=10 | in_data=b71a | key=2098e50a | out_data=0f7b | valid=10
UVM_INFO spn_driver.sv(33) @ 235: uvm_test_top.env.agent.driver [spn_driver] [235] DRIVER: Driving input=0xaf74, key=0xe91e7726, opcode=1
UVM_INFO spn_scoreboard.sv(67) @ 245: uvm_test_top.env.scoreboard [spn_scoreboard] Decryption PASS: expected = 0f7b, actual = 0f7b for input b71a with key 2098e50a
Time: 255 | ENC FROM DUT | opcode=1 | in_data=af74 | key=e91e7726 | out_data=41ea | valid=1
UVM_INFO spn_driver.sv(33) @ 265: uvm_test_top.env.agent.driver [spn_driver] [265] DRIVER: Driving input=0x551c, key=0xb44a0087, opcode=2
UVM_INFO spn_scoreboard.sv(55) @ 275: uvm_test_top.env.scoreboard [spn_scoreboard] Encryption PASS: expected = 41ea, actual = 41ea for input af74 with key e91e7726
Time: 285 | DEC FROM DUT | opcode=10 | in_data=551c | key=b44a0087 | out_data=cdd1 | valid=10
UVM_INFO spn_scoreboard.sv(67) @ 305: uvm_test_top.env.scoreboard [spn_scoreboard] Decryption PASS: expected = cdd1, actual = cdd1 for input 551c with key b44a0087
UVM_INFO /apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.2/src/base/uvm_objection.svh(1276) @ 345: reporter [TEST_DONE] 'run' phase is ready to proceed to the 'extract' phase
UVM_INFO spn_test.sv(49) @ 345: uvm_test_top [spn_enc_dec_test] ----------------------------------------
UVM_INFO spn_test.sv(50) @ 345: uvm_test_top [spn_enc_dec_test] ----            TEST PASS            ----
UVM_INFO spn_test.sv(51) @ 345: uvm_test_top [spn_enc_dec_test] ----------------------------------------
UVM_INFO /apps/vcsmx/vcs/U-2023.03-SP2//etc/uvm-1.2/src/base/uvm_report_server.svh(904) @ 345: reporter [UVM/REPORT/SERVER]
--- UVM Report Summary ---
```

Figure 6:Simulation Terminal

Depending on the figure 5

At Time 15:

An encryption operation is triggered with the input data 0xCA43 and key 0xFC347026, using opcode 01 (which indicates encryption).

- The DUT produces the output 0x8833.
- The scoreboard compares this with the expected result calculated by the reference model and confirms that the encryption result is correct.
- Status: Encryption PASS

At Time 45:

A decryption operation is initiated with the input data 0x9283 and key 0x7D325BAD, using opcode 10 (which indicates decryption)

- The DUT generates output 0x0C5A.
- The scoreboard verifies this against the expected decryption result and confirms a match.
- Status: Decryption PASS

8. Conclusion

The project produced a fully synthesizable Substitution Permutation Network Cryptographic Unit together with a complete UVM verification environment, illustrating an end-to-end hardware design flow for lightweight symmetric encryption. The RTL implementation satisfies every stated requirement accurate key scheduling, three-round data transformation, and robust error handling while the UVM testbench exercised all operational modes and confirmed bit-for-bit agreement with the golden reference model. This work sharpened practical skills in cryptographic architecture, modular SystemVerilog coding, and advanced verification techniques, yielding a compact, reliable core ready for integration or future enhancements such as additional rounds, pipelining, or formal property checking.

9. References

1. Wikipedia. *Substitution–Permutation Network*. https://en.wikipedia.org/wiki/Substitution%E2%80%93permutation_network
2. Bergeron, J. *Writing Testbenches using SystemVerilog*, Springer, 2006.
3. IEEE Std 1800™-2017 – SystemVerilog Language Reference Manual
4. Mentor Graphics. *UVM Cookbook*.
5. Synopsys. *VCS User Guide for SystemVerilog Simulation*.