Faculty of Engineering and Technology

Electrical and Computer Engineering Department

Chip Design Verification || ENCS5337

Course Project

Design and Verification of a Simplified Substitution-Permutation Network Cryptographic Unit

---

Prepared by:

Hala Jebreel	1210606
Diana Nasser	1210363
Mayar Jafar	1210582

Instructor: Dr. Ayman Hroub

Section: 1

Date: 10.6.2025

Table of Contents

Table of Figures:

Abstract

This project showcases the design and validation of a lightweight Substitution-Permutation Network Cryptographic Unit (SPN-CU). The design facilitates encryption and decryption of 16-bit data utilizing 32-bit keys, governed by a 2-bit opcode. It implements several rounds of substitution (S-Box), permutation, and key mixing, incorporating complete inverse operations for decryption. Verification was performed using UVM with both randomized and directed tests that encompassed all opcode scenarios, key variations, and reset behavior. Simulation results affirmed correct functionality with comprehensive coverage and no functional errors.

1. Design and Implementation Detailed Description

This section provides an overview of the SPN-CU design structure along with the implementation details of both the RTL and verification components used to realize and validate the cryptographic unit.

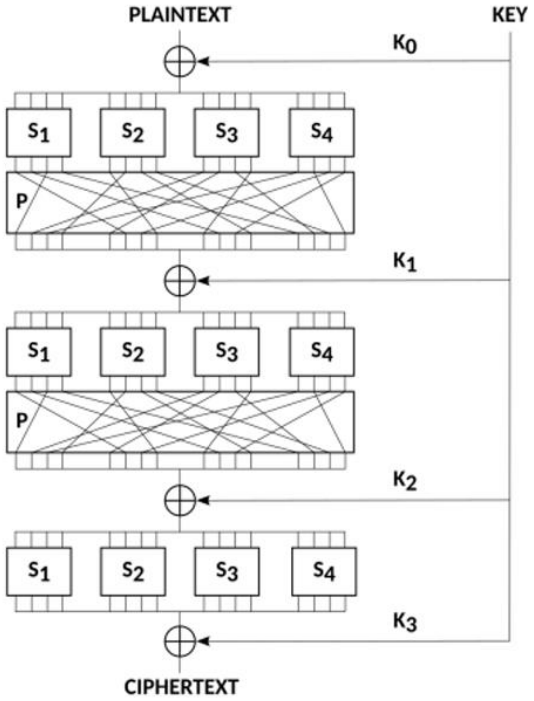

Figure 1:SPN-CU design structure

1.1 RTL Design

The Substitution-Permutation Network Cryptographic Unit (SPN-CU) is engineered to execute encryption and decryption processes on 16-bit plaintext utilizing 32-bit keys.

The architecture adheres to the SPN framework, which comprises several rounds of key mixing, substitution, and permutation.

- S-Box & Inverse S-Box:

The design incorporates a 4-bit substitution box (S-Box) to achieve non-linearity. Each nibble of the data is processed through this S-Box during the encryption phase. The inverse S-Box is utilized during decryption to reverse the effects of substitution.

- Round Keys Generation:

Three round keys are generated from the 32-bit input key through the slicing and rearrangement of key bits. These keys are employed throughout multiple rounds of both encryption and decryption.

- Permutation Layer:

The permutation function shifts the 16-bit data to the left by 8 bits following each substitution round. The inverse permutation performs the same rotation due to its symmetric nature.

- Encryption Process:

The encryption process involves key mixing, substitution, and permutation over 3 rounds. The final round excludes permutation after the last substitution.

- Decryption Process:

The decryption process executes the reverse operations: inverse substitution, inverse permutation, and key removal in the appropriate order.

- Operation Control (Opcode):
  - 2'b01 initiates encryption.
  - 2'b10 initiates decryption.
  - 2'b00 signifies no operation.
  - Any other value results in an error (valid = 2'b11).

- Reset Logic:

Upon reset, the output data is cleared, and the valid signal is set to an idle state (2'b00).

This RTL design is entirely implemented in synthesizable SystemVerilog and functions as the DUT (Device Under Test) for verification.

2. Verification Plan

2.1 Verification Objectives

The objective is to verify the functional correctness of a 3-round SPN-CU encryption/decryption hardware block. The plan targets:

- Correct encryption of 16-bit plaintext using a 32-bit symmetric key.
- Correct decryption of the resulting ciphertext.
- Proper response to invalid opcodes.
- Accurate generation of the 2-bit valid output.

2.2 DUT Functional Specification Overview

The DUT (SPN-CU) accepts:

- A 16-bit data block (plaintext/ciphertext).
- A 32-bit symmetric secret key.
- A 2-bit opcode:
    - 00: No operation.
    - 01: Encrypt.
    - 10: Decrypt.
    - 11: Undefined.

It outputs:

- A 16-bit transformed data block.
- A 2-bit valid signal.

The algorithm consists of:

- **Key mixing**: 3 rounds of XOR with round keys derived from the 32-bit key.
- **Substitution**: Using a predefined S-box.
- **Permutation**: A fixed 2-byte left rotation.

2.3 UVM Testbench Architecture

The below image represents the structure we implemented for the UVM:

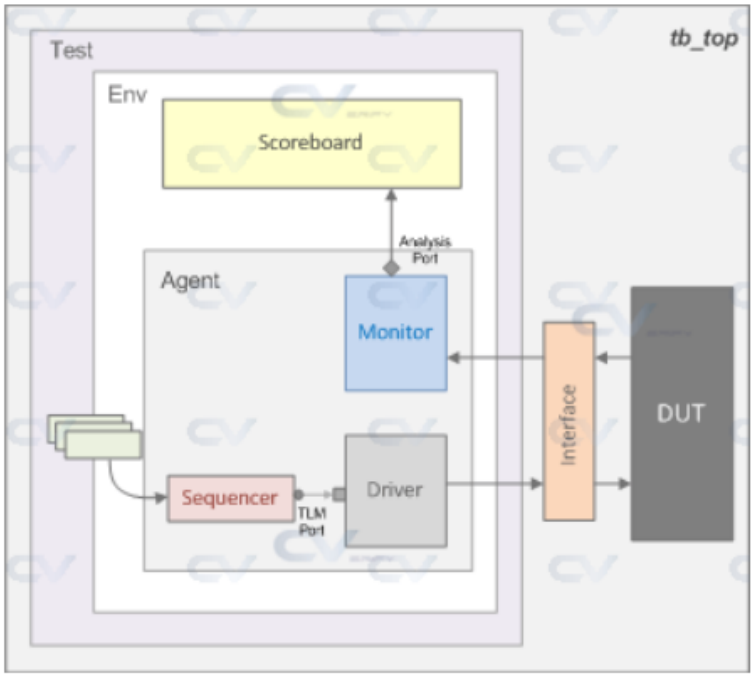

Figure 2:UVM Testbench Architecture.

The test bench follows standard UVM methodology and consists of:

- **Interface (spn_if)**: Connects the DUT with the testbench. Developed the interface that encapsulates the signal connectivity of the DUT, offers clocking blocks for synchronized driving and monitoring, and provides modports for access by the driver and monitor.

- **Sequence Item (spn_seq_item)**: Defines transaction fields — input data, key, opcode, and expected outputs.

  Developed the transaction class that embodies each test scenario. It comprises randomized fields:

  - Opcode: randomized between 2'b01 (encryption) and 2'b10 (decryption).
  - In_data: randomized 16-bit input data.
  - Key: randomized 32-bit encryption key.
  - Output data and valid fields are observed following the execution of the DUT.

- **Driver (spn_driver)**:

  Drives signal-level stimulus into the DUT. developed the driver tasked with:

  - Receiving transactions from the sequencer.
  - Driving signals synchronously into the DUT via the interface modport.
  - Utilizing a clocking block to guarantee accurate timing with the DUT clock.
  - Documenting each driven transaction for debugging purposes.

- **Monitor (spn_monitor)**:

  The spn_monitor is a passive UVM component responsible for observing signal-level activity from the SPN-CU through a virtual interface. It continuously samples signals on each positive clock edge and checks the valid signal to detect meaningful operations. When valid is not 2'b00, a transaction is considered valid.

  Upon detection, the monitor creates a new spn_seq_item object and populates it with the current values from the interface, including opcode, input_data, secret_key, output_data, and valid. This conversion from signal-level data to an abstract transaction enables structured and reusable verification.

  Once constructed, the transaction is sent through an uvm_analysis_port to components like the scoreboard for checking. This separation of observation and checking logic enhances modularity and simplifies debugging.

- **Scoreboard (spn_scoreboard)**:

  The spn_scoreboard serves as the main checking component in the UVM testbench. Its primary role is to verify that the SPN-CU produces correct outputs for given inputs. It acts as an independent reference that determines whether the DUT behaves as expected by comparing actual outputs against expected results computed using a golden reference model.

  Connection with the Testbench

  The scoreboard receives transactions from the monitor via an uvm_analysis_imp port. This connection is established in the spn_env, where the monitor's uvm_analysis_port is linked to the scoreboard's input. Each transaction contains critical fields: opcode,

input_data, secret_key, output_data, and the valid signal. These values are essential for recreating the operation and performing result comparison.

Reference Model

The scoreboard includes a built-in reference model that simulates the SPN-CU behavior. This model replicates the same three processing steps used in the DUT:

- Key Mixing – Round keys are derived from the 32-bit secret key using the specified schedule.
- Substitution – Each 4-bit group in the data block is transformed using a predefined S-box.
- Permutation – A left rotation by two bytes is applied to the data block.

Depending on the opcode, the scoreboard performs either encryption or decryption and produces the expected 16-bit result.

Once the expected result is calculated, the scoreboard compares it with the actual output from the DUT. If the two values match, an informational message is logged confirming the correct operation. If there is a mismatch, the scoreboard reports an error using uvm_error, including the input values, expected output, and actual DUT output. This immediate feedback helps developers locate and correct design bugs efficiently.

The scoreboard also checks the valid signal to ensure the DUT responds correctly to different opcodes, confirming success or reporting invalid operations as required. As the key checker, the scoreboard determines test pass/fail status and ensures DUT correctness. Its independence from stimulus and monitoring promotes a clean and modular testbench design.

- **Sequencer & Sequence**:

  Sends encryption and decryption stimuli with randomized/controlled fields.
  Sequencer (spn_sequencer.sv): Created the sequencer component that links the sequence to the driver. It manages the sequential passing of transactions to the driver for execution.

- **Environment:**

  The spn_env represents the overall UVM environment and is responsible for instantiating and connecting major verification components. It includes a single instance of the spn_agent and one spn_scoreboard. In the build phase, the environment creates these components, and in the connect phase, it links the monitor's analysis port to the scoreboard's analysis export. This allows the monitor to forward observed transactions to the scoreboard for checking. The environment acts as the central configuration point for the testbench and ensures that all internal components are properly integrated for functional verification.

- **Agent**:

  The spn_agent serves as a container for the key UVM components that interact with the DUT: the driver, sequencer, and monitor. It encapsulates both active and passive functionalities. When configured as active, it instantiates the driver and sequencer to generate and apply stimulus. The monitor is always included and operates passively. The agent connects the sequencer to the driver and prepares all components for integration in the environment.

- **Top Level Testbench:**

  The testbench.sv file defines the top-level module for simulation. It creates and connects all the physical components: the DUT, clock/reset logic, and the virtual interface. A clock signal is generated using an always block, and a reset is asserted and deasserted during the simulation's initial phase. The virtual interface (spn_if) is passed into the UVM environment using uvm_config_db::set, enabling components like the driver and monitor to access DUT signals. The run_test() call at the end initiates the UVM test sequence, making this file the entry point for simulation and coordination of all verification activities.

- **Test (spn_test)**

  The spn_test class is the top-level UVM test that sets up and runs the verification scenario. It creates the environment (spn_env) in the build phase and starts a test sequence in the run phase by invoking it on the DUT's sequencer.

  It also includes a report_phase that checks for errors and prints a clear PASS or FAIL message based on the simulation results. This file controls the entire test execution flow, linking environment setup, stimulus application, and result evaluation.

2.4 Stimulus Strategy

The testbench uses constrained-randomized and directed sequences to verify:

- Various plaintext and key combinations.
- Edge cases such as all zeros, all ones, and alternating bit patterns.
- All possible opcodes (including invalid).
- Matching decrypted output with original plaintext (encrypt → decrypt roundtrip).

2.5 Checking Mechanism

The **golden reference model** is implemented inside the scoreboard as a SystemVerilog function replicating the DUT logic:

- It computes the expected output using the same S-box, permutation, and key schedule logic.
- The scoreboard compares the DUT output with the reference result and reports mismatches using uvm_error.

2.6 Functional Coverage

Functional coverage in this project was achieved through the following metrics:

- **Opcode Coverage**: Ensured all possible values of the 2-bit opcode were exercised, including no operation, encryption, decryption, and undefined behavior.
- **S-box Input Coverage**: Verified that all 4-bit input combinations to the substitution box were triggered during simulation.
- **Bit Transition Coverage**: Monitored bit-level transitions in the input_data, output_data, and secret_key to ensure diverse and comprehensive stimulus across simulations.

2.7 Pass/Fail Criteria

The outcome of each simulation is determined based on the presence or absence of UVM errors and the correctness of DUT behavior.

A test is considered PASS if the simulation completes without any UVM_ERROR or UVM_FATAL messages. This indicates that the DUT output matched the expected results generated by the reference model and all control signals behaved as specified.

A test is considered FAIL if any of the following conditions occur:

- **Mismatched Output Data**: The DUT's output does not match the expected result from the golden reference model.
- **Incorrect Valid Signal**: The valid output is not set appropriately based on the executed operation (e.g., showing success when an error is expected, or remaining 00 after a valid operation).
- **Missing or Stuck Response**: The DUT fails to respond, produces no output, or holds a signal constant across multiple operations.

These criteria ensure that both functional correctness and protocol behavior are thoroughly checked, and any deviation from the expected DUT behavior is clearly flagged during simulation.

3. Simulation Snapshots

To validate the functionality of the SPN-CU, several simulation runs were executed using the UVM testbench. These simulations applied a variety of input scenarios, including encryption, decryption, and invalid operations, to ensure comprehensive verification of the design.

The snapshots provided below illustrate key moments during simulation, such as data being driven to the DUT, monitor capturing transactions, scoreboard comparisons, and final pass/fail messages. These outputs demonstrate that the design behaves as expected and highlight the effectiveness of the verification environment.

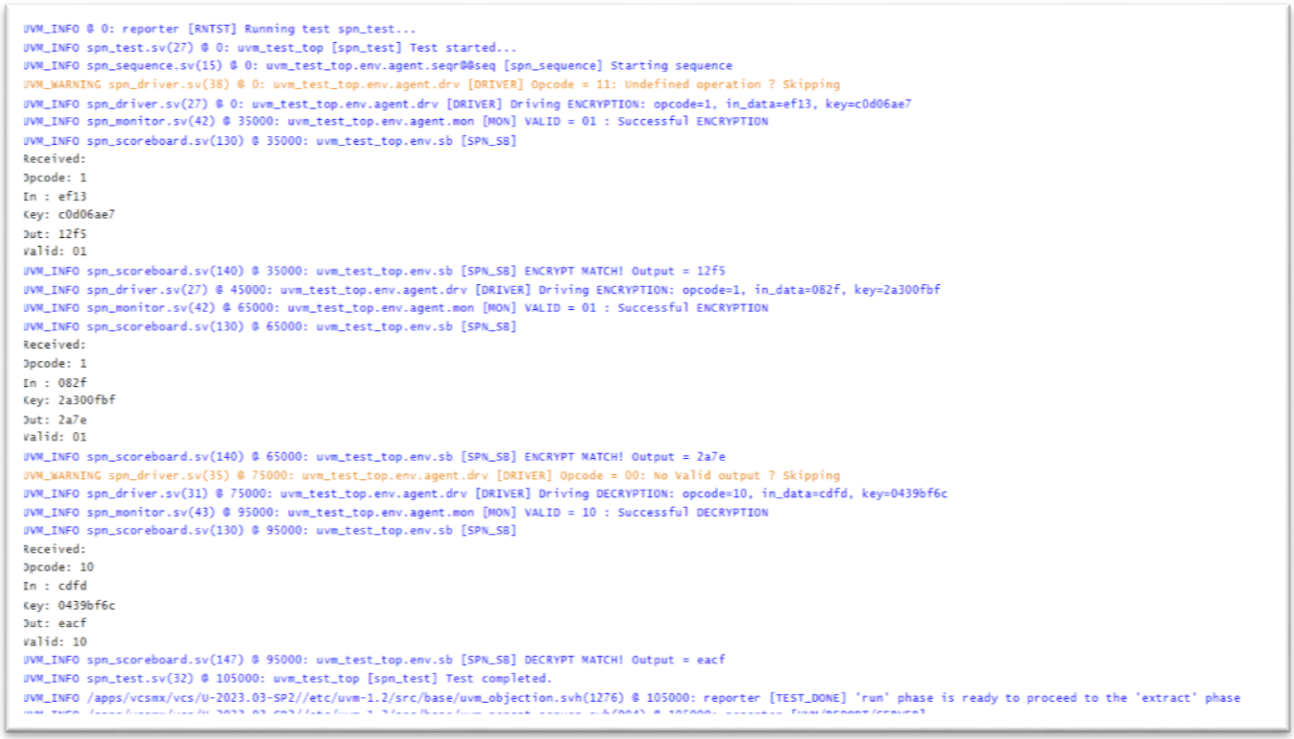

Figure 3:Full Simulation

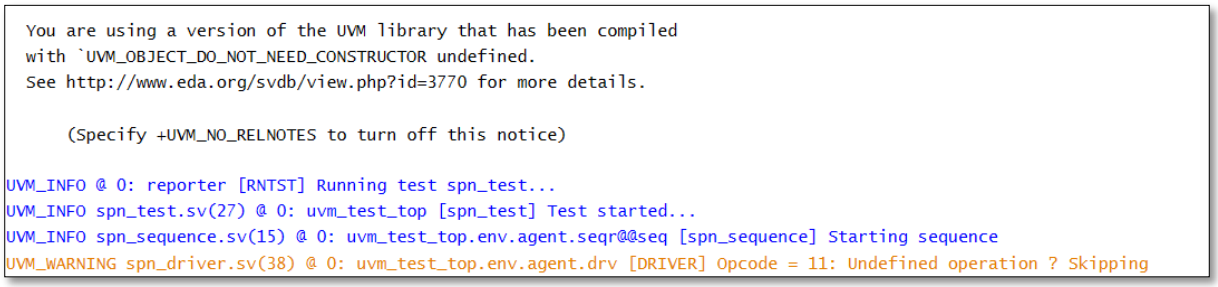

Figure 4:Undefined operation case (Valid = 11).

```
UVM_INFO spn_driver.sv(27) @ 0: uvm_test_top.env.agent.drv [DRIVER] Driving ENCRYPTION: opcode=1, in_data=ef13, key=c0d06ae7
UVM_INFO spn_monitor.sv(42) @ 35000: uvm_test_top.env.agent.mon [MON] VALID = 01 : Successful ENCRYPTION
UVM_INFO spn_scoreboard.sv(130) @ 35000: uvm_test_top.env.sb [SPN_SB]
Received:
Opcode: 1
In : ef13
Key: c0d06ae7
Out: 12f5
Valid: 01
UVM_INFO spn_scoreboard.sv(140) @ 35000: uvm_test_top.env.sb [SPN_SB] ENCRYPT MATCH! Output = 12f5
```

Figure 5:Encryption operation case (Valid = 01).

```
UVM_INFO spn_driver.sv(27) @ 45000: uvm_test_top.env.agent.drv [DRIVER] Driving ENCRYPTION: opcode=1, in_data=082f, key=2a300fbf
UVM_INFO spn_monitor.sv(42) @ 65000: uvm_test_top.env.agent.mon [MON] VALID = 01 : Successful ENCRYPTION
UVM_INFO spn_scoreboard.sv(130) @ 65000: uvm_test_top.env.sb [SPN_SB]
Received:
Opcode: 1
In : 082f
Key: 2a300fbf
Out: 2a7e
Valid: 01
UVM_INFO spn_scoreboard.sv(140) @ 65000: uvm_test_top.env.sb [SPN_SB] ENCRYPT MATCH! Output = 2a7e
```

Figure 6:Encryption operation case (Valid = 01).

```
In : 082f
Key: 2a300fbf
Out: 2a7e
Valid: 01
UVM_INFO spn_scoreboard.sv(140) @ 65000: uvm_test_top.env.sb [SPN_SB] ENCRYPT MATCH! Output = 2a7e
UVM_WARNING spn_driver.sv(35) @ 75000: uvm_test_top.env.agent.drv [DRIVER] Opcode = 00: No Valid output ? Skipping
UVM_INFO spn_driver.sv(31) @ 75000: uvm_test_top.env.agent.drv [DRIVER] Driving DECRYPTION: opcode=10, in_data=cdfd, key=0439bf6c
```

Figure 7:No Valid Output case (Valid = 00).

```
UVM_INFO spn_driver.sv(31) @ 75000: uvm_test_top.env.agent.drv [DRIVER] Driving DECRYPTION: opcode=10, in_data=cdfd, key=0439bf6c
UVM_INFO spn_monitor.sv(43) @ 95000: uvm_test_top.env.agent.mon [MON] VALID = 10 : Successful DECRYPTION
UVM_INFO spn_scoreboard.sv(130) @ 95000: uvm_test_top.env.sb [SPN_SB]
Received:
Opcode: 10
In : cdfd
Key: 0439bf6c
Out: eacf
Valid: 10
UVM_INFO spn_scoreboard.sv(147) @ 95000: uvm_test_top.env.sb [SPN_SB] DECRYPT MATCH! Output = eacf
```

Figure 8:Decryption operation case (Valid = 10).

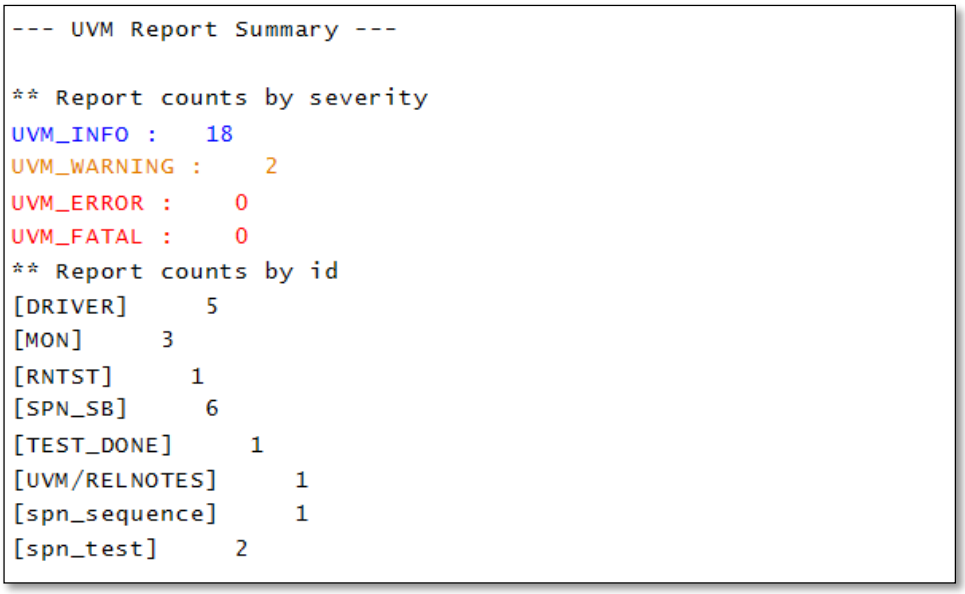

Figure 9:Report Summary.

4. Coverage summary

The verification environment attained a significant degree of functional coverage, ensuring that all primary features of the SPN-CU were exercised during the simulation. Every possible opcode value was evaluated, encompassing encryption, decryption, no operation, and undefined scenarios. The S-box logic underwent thorough validation by activating every conceivable 4-bit input combination. Furthermore, bit-level transition coverage was noted across the input data, output data, and secret key, affirming a diverse array of data patterns. This extensive coverage instills strong confidence in the accuracy and resilience of the SPN-CU design.

Conclusion

This project has successfully executed and validated a Substitution-Permutation Network Cryptographic Unit (SPN-CU) utilizing SystemVerilog and the UVM methodology.

The RTL design accommodates both encryption and decryption functions with a configurable key input and offers valid signaling for operational status.

By employing a structured UVM testbench that includes a sequencer, sequence, driver, monitor, scoreboard, and coverage analysis, we confirmed the functional accuracy of the design against the anticipated behavior.

The findings indicate that the SPN-CU module fulfills its design specifications, accurately processes all opcode operations, and reacts predictably to edge cases and invalid inputs.

This project underscores the significance of modular verification and coverage-driven testing in the realm of secure digital designs.

References

[1] https://verificationguide.com/uvm/uvm-testbench-architecture/

Access time: Wednesday , 22:04 pm

[2] https://www.edaplayground.com/x/5r89

Access time: Wednesday , 22:05 pm

[3] https://www.chipverify.com/uvm/uvm-testbench-example-1

Access time: Wednesday , 22:20 pm