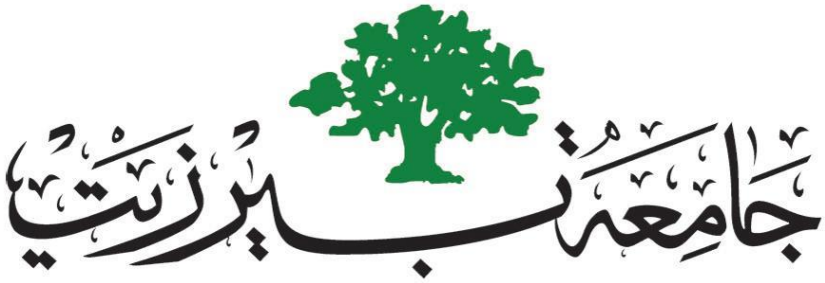


BIRZEIT UNIVERSITY

Birzeit University

Faculty of Engineering & Technology

Department of Electrical & Computer Engineering

ENCS5337

Chip Design Verification

Design and Verification of a Simplified Substitution-Permutation Network Cryptographic Unit " (SPN-CU) Using System Verilog/UVM"

Report

Prepared by:

Abdelrahman Shaheen **Student NO.** 1211753
Mahmoud Awad **Student NO.** 1212677
Robert Bannoura **Student NO.** 1210574

Supervised By: Dr. Ayman Hroub

Data: 10_6_2025

SPN-CU System Design Specification

1. Objective

Design and verify a simplified cryptographic processing unit (SPN-CU) capable of encrypting and decrypting **16-bit plaintext** using a **32-bit key**. The design will follow the Substitution-Permutation Network (SPN) model and be implemented in SystemVerilog with functional verification via a testbench and a golden reference.

2. Functional Requirements

➢ Inputs:

- **data_in:** 16-bit
- **key:** 32-bit
- **opcode:** 2-bit operation code
  - **00** = No Operation (NOP)
  - **01** = Encrypt
  - **10** = Decrypt
  - **11** = Invalid Operation

➢ Outputs:

- **data_out:** 16-bit
- **valid:** 2-bit status flag
  - **00** = No valid output
  - **01** = Successful encryption
  - **10** = Successful decryption
  - **11** = Internal error or undefined operation

➢ processing stages:

- Key scheduling
- Initial round key mixing
- Three rounds of:
  - Substitution
  - Permutation
  - AddRoundKey
  - Final output

➢ Clocking & Reset

- Synchronous design using rising edge of clock
- Active-high synchronous reset

3. Non-Functional Requirements

Category	Requirement
Performance	Encryption/Decryption within a few cycles (bounded latency)
Modularity	Design should be clearly modular (separate Datapath and control logic if possible)
Scalability	Easy to scale to more rounds or longer keys later
Testability	Support verification using a golden model and random test vectors
Synthesizability	Must be fully synthesizable using standard synthesis tools

4. Constraints and Assumptions

- Use only behavioral or structural SystemVerilog (no vendor-specific IPs).
- No pipelining is required at this stage (single-stage SPN-CU).
- SPN rounds and S-boxes will be simplified and fixed.
- Only ECB-like mode for individual block processing.
- Assume all inputs are valid unless opcode is 11.

5. SPN-CU Architecture Overview (High Level)

➢ Key Scheduler

- Derives round keys (k0–k2) from 32-bit input key.

➢ Encryption Path:

- AddRoundKey (initial key)
- 3 rounds of Substitution → Permutation → AddRoundKey

➢ Decryption Path:

- Inverse AddRoundKey
- Inverse Permutation → Inverse Substitution → AddRoundKey (reversed order

6. Files and Modules for the RTL Design

File/Module	Description
spn_cu.sv	Top-level SPN-CU RTL module
key_schedule.sv	key scheduler gives k0 – k3
spn_core.sv	his module handles the full spn logic
sbox_pkg.sv	S-boxes functions package
permute_pkg.sv	permutation functions package

7. Design and Implementation Detailed Description

7.1 Top-Level Architecture

The design of the SPN-CU (Substitution-Permutation Network Cryptographic Unit) is based on a modular SystemVerilog structure, enabling a clean separation between functional blocks such as the key scheduler, core encryption logic, substitution and permutation layers, and top-level control. The top module spn_cu.sv instantiates two main components: the key_schedule module and the spn_core module. These interact to achieve encryption and decryption over a 16-bit data block using a 32-bit symmetric key.

The design operates synchronously with a positive-edge clock and an active-high synchronous reset. The opcode input determines the operation mode (00 = NOP, 01 = encryption, 10 = decryption, 11 = invalid). Based on this opcode, the spn_core performs the required transformation.

7.2 Key Scheduling Unit

The key_schedule module is responsible for deriving the subkeys required by the encryption and decryption processes. From the 32-bit input key, four 16-bit keys (k0 to k3) are extracted as follows:

- **k0 =** {key[23:16], key[7:0]}
- **k1 =** key[15:0]
- **k2 =** {key[7:0], key[31:24]}
- **k3 =** set to 16'd0 (not functionally used but reserved for future flexibility)

This static key schedule allows efficient derivation of round keys without iterative generation, reducing complexity and latency.

7.3 SPN Core Logic

The heart of the SPN-CU is the spn_core module, which implements the main SPN transformation for both encryption and decryption. This module is parameterized by:

- 16-bit input data
- Four subkeys
- A 2-bit opcode
- Clock and reset signals

The operation proceeds as follows:

➢ **Encryption (opcode = 01):**

1. **Initial Key Mixing**: temp = data_in ^ k0
2. **Three SPN Rounds**:
   - temp = round(temp, k1)
   - temp = round(temp, k2)
   - temp = final_round(temp, k3)
3. **Output Assignment**:
   - data_out <= temp
   - valid <= 2'b01

Each round() function applies substitution using an S-box, followed by permutation (left-rotate 2 bytes), and XOR with the respective round key.

➢ **Decryption (opcode = 10):**

1. **Inverse Final Round**: temp = inv_final_round(data_in, k3)
2. **Two Inverse SPN Rounds**:
   - temp = inv_round(temp, k2)
   - temp = inv_round(temp, k1)
3. **Inverse Key Mixing**: temp = temp ^ k0
4. **Output Assignment**:
   - data_out <= temp
   - valid <= 2'b10

Decryption is essentially the reverse of the encryption path, using inverse permutation and inverse substitution functions.

➢ **No Operation / Undefined:**

- If opcode = 00, valid is set to 2'b00, and data_out is left unchanged.
- If opcode = 11, valid is set to 2'b11, indicating an invalid operation.

7.4 Substitution and Permutation Logic

The S-box logic is implemented in sbox_pkg.sv and uses a 16-entry lookup table for substitution. The function sbox() maps a 4-bit nibble to its substituted equivalent as per the project's fixed table. Similarly, inv_sbox() performs the inverse mapping. These are used by substitute() and inv_substitute() to process the full 16-bit data block by applying the S-box transformation to each nibble.

Permutation is handled in permute_pkg.sv. The permute() function performs a left rotation by 8 bits (2 bytes), and inv_permute() simply re-applies the same operation since the permutation is symmetric (i.e., inv_permute = permute). This simplifies hardware implementation and ensures low latency.

7.5 Design Decisions and Considerations

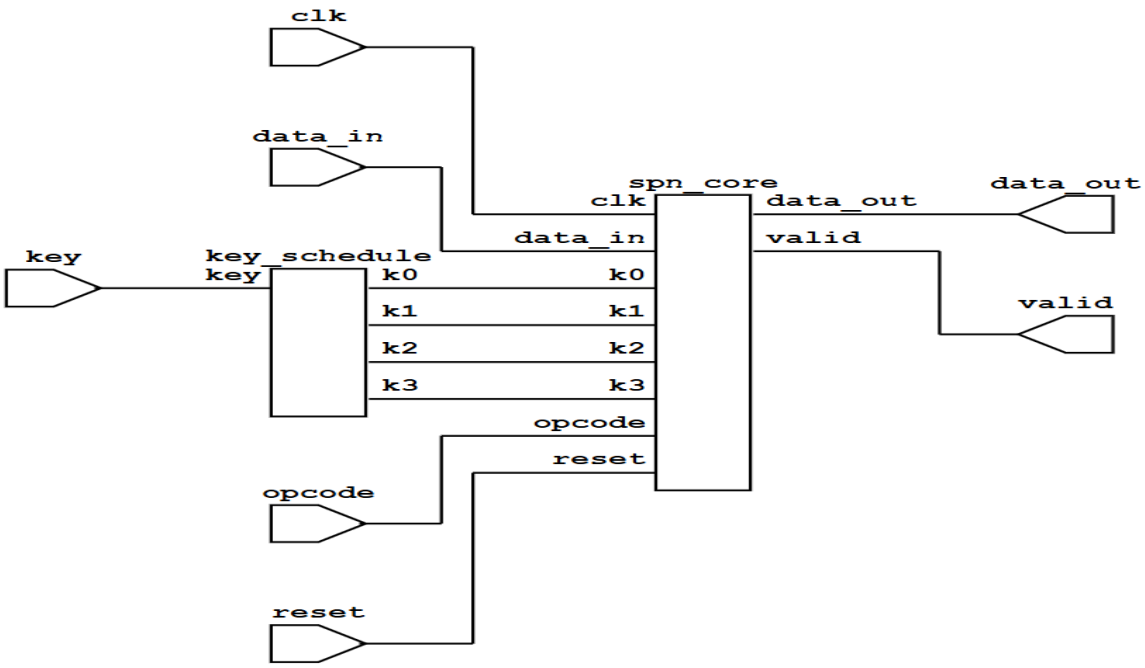
- **K3 Handling**: Although the SPN model used only three keys, k3 was defined and initialized to zero to maintain extensibility. XORing with zero has no effect (x ^ 0 = x), allowing future upgrades without structural changes.
- **Opcode Decoding**: All four possible opcode values are explicitly handled. This ensures that invalid inputs do not propagate unintended behavior and simplifies UVM test coverage.
- **Modularity**: The design is highly modular, with clear separation between key scheduling, SPN logic, substitution/permutation logic, and control paths. This aids verification, debugging, and future enhancements.
- **Synthesizability**: All constructs used (e.g., always_ff, always_comb, packages, functions) are compliant with SystemVerilog synthesis guidelines, making the design fully synthesizable.

7.6 Simulation and Reset Behavior

Upon assertion of the reset signal, the spn_core clears all outputs (data_out = 0, valid = 00). This ensures that the system starts from a known state and avoids glitches or false transitions in early cycles.

The sequential nature of the design ensures that encryption and decryption complete within a deterministic number of cycles — typically one per operation, assuming valid inputs and key are ready.

7.7 SPN-CU Top-Level Block Diagram



8. key Considerations

A. When op is no operation (00). What must the design do?

assuming the customer specified this behavior:

1. Set valid <= 2'b00:

Clearly indicate that the output is not valid. This avoids any confusion downstream in the design or verification environment.

2. Leave data_out unchanged

Do not assign data_out unless required. This avoids unnecessary transitions on the signal, which:

- Saves power (important in ASIC/FPGA)
- Prevents misinterpretation by downstream modules

B. What must the value of k3 be?

We sat its value to 0 assuming customer's specifications it's the same as there is no k3 since x xor 0 = x so the customer can change its value in future

C. When op is undefined (11). what must the design do?

assuming the customer specified this behavior:

1. Set valid <= 2'b11:
2. leave output unchanged

Verification plan

Specification Review & Requirements Extraction

Overview:

This section captures the functional intent of the **SPN Core Unit (SPN-CU)** based on the design specification. It identifies key features, behaviors, and operational modes that must be verified to ensure correctness, reliability, and compliance with the functional requirements.

Design Description Summary:

The SPN-CU is a 16-bit cryptographic processing unit that implements a 3-round SPN encryption and decryption scheme. It operates on 16-bit plaintext or ciphertext using three 16-bit round keys: K0, K1, and K2. The unit supports four operational modes controlled via a 2-bit opcode, with corresponding control over the valid output signal.
Key properties:

- Based on substitution-permutation cryptography with fixed S-boxes and a fixed permutation layer.
- Sequential encryption: initial XOR with K0, followed by two full rounds (substitution → permutation → XOR with key), then a final substitution stage (no K3).
- Decryption is the logical inverse of the encryption path.
- Supports asynchronous reset and defined behavior for invalid opcodes.

Interface Summary:

Signal	Direction	Description
clk	Input	Clock signal
reset	Input	Asynchronous active-high reset
opcode	Input	2-bit code: encrypt, decrypt, noop, undefined
data_in	Input	16-bit plaintext or ciphertext
key	Input	32-bit symmetric key used for round processing
data_out	Output	16-bit result after encryption/decryption
valid	Output	2-bit status: 01 = encrypt OK, 10 = decrypt OK,00 = no valid output, 11 = internal error or undefined operation

Functional Requirements

Requirement ID	Description
FR1	DUT shall correctly encrypt data_in using keys K0, K1, and K2 when opcode = 2'b01.
FR2	DUT shall correctly decrypt data_in using keys K2, K1, and K0 when opcode = 2'b10.
FR3	When opcode = 2'b00 (NOOP), DUT shall output valid = 2'b00 and not alter data_out.
FR4	When opcode = 2'b11 (UNDEFINED), DUT shall output valid = 2'b11 and avoid computation.
FR5	When reset = 1, DUT shall immediately output data_out = 0 and valid = 2'b00.
FR6	DUT shall apply S-box substitution, permutation, and XOR steps as per the SPN model.
FR7	DUT shall support edge cases such as all-zero/all-one input and key patterns.
FR8	DUT shall only update data_out when valid indicates a completed operation.

Outcome:

This review provides the functional foundation for verification. Each requirement will be addressed by:

- One or more directed or randomized test sequences
- Coverage items
- Assertions and scoreboarding

This ensures traceability between design intent and verification goals.

Feature List & Verification Objectives

Overview:

This section identifies all **verifiable features** of the SPN-CU design. For each feature, we define the corresponding **verification objective** — what must be demonstrated through simulation and checking mechanisms.

Key Features and Their Descriptions

Feature ID	Feature Description
F1	32-bit symmetric key is decoded into round keys K0, K1, K2
F2	Supports four opcodes: ENCRYPT, DECRYPT, NOOP, UNDEFINED
F3	Applies round-based SPN transformation for encryption using substitute, permute, XOR
F4	Performs inverse SPN transformation for decryption
F5	Sets appropriate 2-bit valid signal based on the opcode
F6	Holds previous output during NOOP or invalid opcode
F7	Handles asynchronous reset and clears output/valid immediately
F8	Accepts input synchronously on rising clock edge
F9	Does not produce output when reset = 1 or opcode = 00 or opcode = 11

Verification Objectives

Objective ID	Verification Objective
VO1	Verify correct extraction of K0, K1, K2 from 32-bit key input
VO2	Verify that encryption results match the golden model (ENCRYPT mode)
VO3	Verify that decryption results match the golden model (DECRYPT mode)
VO4	Check that valid = 2'b00 during NOOP and that data_out is stable
VO5	Check that valid = 2'b11 during undefined opcode, with data_out unchanged or zero
VO6	Verify correct reset behavior: data_out = 0, valid = 2'b00 immediately
VO7	Confirm that data_out only changes on a valid encrypt or decrypt command
VO8	Ensure invalid keys or corrupt inputs produce predictable, non-glitched outputs
VO9	Confirm all features are exercised through directed/random tests and covered in metrics

Outcome

This feature-objective mapping guides all test creation, coverage modeling, and check implementation. Each objective will be linked to:

- One or more test cases
- Scoreboard checks
- Functional coverage bins
- Assertions (where applicable)

Verification Schedule

Phase	Start Date	End Date	Duration	Objective
Requirements Analysis	2025-05-26	2025-05-27	1 day	Review the specification and functional behavior of SPN-CU design
RTL Design Implementation	2025-05-27	2025-05-28	1 day	Code and synthesize the SPN-CU encryption/decryption module
Golden Reference Model	2025-06-01	2025-06-02	1 day	Build a trusted model to validate functional correctness of DUT
Basic Testbench (non-UVM)	2025-06-01	2025-06-02	1 day	Create a basic SystemVerilog testbench to perform initial simulation checks
UVM Environment Setup	2025-06-03	2025-06-06	3 days	Create UVM environment: test, env, agent, driver, monitor, etc.
Sequence Development	2025-06-07	2025-06-08	2 days	Develop sanity, corner-case, and constrained-random sequences
Scoreboard and Coverage	2025-06-07	2025-06-08	1 day	Implement self-checking scoreboard and functional coverage metrics
Regression Testing and Debugging	2025-06-08	2025-06-08	1 day	Run full test suite, fix mismatches, verify code and coverage completeness
Final Documentation & Reporting	2025-06-08	2025-06-10	2 days	Compile test results, write verification summary, prepare final report

Testbench Architecture Definition

Overview

This section outlines the **UVM-based testbench architecture** for verifying the SPN-CU. It specifies the verification components required to generate, drive, monitor, and check DUT behavior in a reusable, scalable, and layered structure.

Testbench Hierarchy & Components

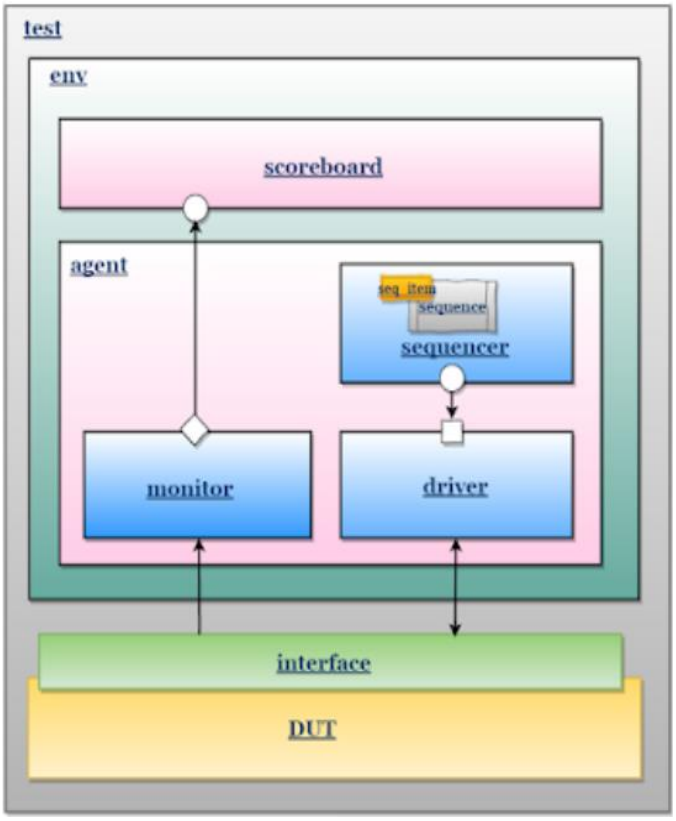


figure.1. UVM hierarchy and components

Component Breakdown

Component	Description
spn_env	Top-level environment connecting all agents, monitors, scoreboard, coverage
spn_agent	Encapsulates driver, sequencer, and monitor; configured as active/passive
spn_sequencer	Sends sequences (data, key, opcode, reset) to the driver
spn_driver	Drives interface signals to the DUT using virtual interface
spn_monitor	Observes DUT outputs and writes transactions to analysis ports
spn_scoreboard	Compares DUT output against golden model to report pass/fail
spn_sequence	Stimulus generator that randomizes input transactions
spn_test	Controls test execution by starting sequences and phases
Coverage (optional)	Collects opcode, key patterns, and functional path coverage

DUT Interface Integration

- A SystemVerilog **virtual interface** (spn_if) will be used to connect the testbench with the DUT signals.
- The interface will include: clk, reset, opcode, data_in, key, data_out, valid.
- This interface will be shared across driver, monitor, and scoreboard.

Outcome

The architecture supports:

- Transaction-level modeling
- Separation of concerns (stimulus, checking, monitoring)
- Reusability and scalability
- Coverage-driven and self-checking testbench design

Stimulus Planning

Overview

Stimulus planning defines the **types of inputs, sequences**, and **scenarios** that will be applied to the SPN-CU DUT to verify all functional behavior. This includes directed tests, constrained random generation, edge cases, and negative scenarios.

Stimulus Type	Description
Directed Tests	Manually defined input data, opcodes, and keys with known expected outputs
Randomized Tests	Constrained-random values for data_in, opcode, and key to explore design space
Edge/Critical Cases	All-zeros, all-ones, alternating bits, min/max values
Illegal Scenarios	Invalid opcodes, reset during operation, unused key bits

Planned Sequences

Sequence Name	Description
spn_sanity_seq	encryption and decryption with known input and key
spn_random_seq	Random opcodes, keys, and data with scoreboard checking
spn_corner_seq	All-zeros, all-ones, alternating bits, min/max values
spn_error_seq	Use opcode=11 and check valid/error response
spn_key_sweep_seq	keep data_in and change key
spn_data_sweep_seq	keep key and change data_in

Constraints & Randomization

- opcode will be randomized over {00, 01, 10, 11}
- data_in and key will be randomized with optional constraints:
  - key[31:0] must not be all-zero or all-one in some tests
  - opcode == 2'b11 => expect no computation
  - When reset == 1, data_out must go to 0 and valid == 00

Outcome

This stimulus plan ensures that:

- All functional paths are exercised
- Edge cases and corner scenarios are not missed
- Reset and illegal conditions are verified
- The DUT is stressed realistically with back-to-back and randomized ops

Coverage Planning

Overview

Coverage planning ensures the **entire design intent is exercised and validated**. It includes both **functional coverage** (verifying all DUT behaviors are triggered) and **code coverage** (ensuring all RTL paths are executed).

Coverage Types

Type	Description
Functional Coverage	Manually defined bins to capture key functional events
Code Coverage	Automatically generated by simulator (e.g., VCS)

Functional Coverage Goals

Coverage Point	Details
Opcode Coverage	All 4 opcodes exercised: 00, 01, 10, 11
Valid Output Values	Valid = 01, 10, 00, 11 depending on opcode and reset
Data Pattern Coverage	Inputs like all-0s, all-1s, alternating bits, random
Key Pattern Coverage	Random and edge-case 32-bit keys

Implemented using SystemVerilog covergroups attached to monitor.

Code Coverage Goals

Metric	Description
Statement	All lines of RTL executed at least once
Branch	All if, case, and conditionals evaluated both ways
Toggle	All bits in signals toggle between 0 and 1
Expression	All Boolean expressions fully exercised

Coverage Tools & Collection Points

- **Functional coverage** collected in the UVM monitor and/or scoreboard
- **Code coverage** collected via simulator using RTL instrumentation
- Reports analyzed post-regression to identify unverified areas

Outcome

- A clear measurement of verification completeness
- Confidence that **every feature, corner case, and RTL path** has been exercised
- Foundation for **coverage-driven closure** and sign-off

Checkers & Scoreboarding

Overview

This section defines how the DUT's outputs (data_out, valid) are verified using a self-checking mechanism. A golden reference model is used to compute expected outputs for encryption and decryption modes. All comparisons are performed automatically in the UVM scoreboard.

Core Strategy

Component	Role
Golden Model	Reference SystemVerilog function that implements correct SPN behavior
Scoreboard	Collects outputs from the DUT and compares them with golden model results
Monitor	Observes and transfers DUT inputs/outputs into transaction objects

DUT Output Validation Logic

Opcode	Validation Action
2'b01 (ENCRYPT)	Compare data_out against golden_encrypt(data_in, key) and check valid == 2'b01
2'b10 (DECRYPT)	Compare data_out against golden_decrypt(data_in, key) and check valid == 2'b10
2'b00 (NOOP)	Check valid == 2'b00
2'b11 (UNDEF)	Check valid == 2'b11
reset == 1	DUT must set data_out = 0 and valid = 2'b00 immediately

Implementation Details

Checker Element	Description
spn_scoreboard	UVM component comparing DUT results to golden outputs
write()	Receives transaction from monitor
run_phase()	Continuously compares DUT output to golden model
Pass/Fail Logging	$display, uvm_info, or uvm_error marks match/mismatch
Support for Reset/UNDEF	Check if valid and data_out behave correctly on edge cases

Outcome

This checker mechanism ensures the DUT is:

- Producing correct outputs for all functional modes
- Compliant with timing and reset behavior
- Fully self-verified without human observation

Pass/Fail Criteria

Overview

This section defines how we determine whether the DUT has passed verification. These criteria must be met for each test case individually and for the project as a whole before verification closure.

Per-Test Pass Criteria

Checkpoint	Requirement
Scoreboard Match	data_out from DUT matches golden model output
Valid Signal Accuracy	valid field reflects correct operation mode per spec
No Unexpected Changes	data_out does not change in NOOP or UNDEFINED modes
Reset Behavior	On reset == 1, DUT outputs data_out = 0 and valid = 00
No X/Z Values	Output signals do not contain unknowns (x) or high-Z (z)

Regression-Level Pass Criteria

The verification environment is considered successful if:

Metric	Requirement
All test cases pass	No failing tests in regression logs
Functional coverage	All bins hit or justified (goal: 100%)
Code coverage	Target ≥ 90% for line, branch, toggle, FSM
No simulator errors	No fatal simulation errors, timeouts, or core dumps

Result Reporting

For each test, include in log:

- Test Name
- UVM_PHASE result (passed/failed)
- Scoreboard match or mismatch
- Coverage snapshot

At regression level:

- Generate summary CSV/table of test outcomes
- Generate functional coverage report

Outcome

This step defines objective success, eliminates guesswork, and ensures a clear verification closure path. These criteria will be used to determine when the design is fully verified and sign-off ready.

SPN-CU Verification Summary Report

Verification of SPN Cryptographic Unit (SPN-CU)

1. DUT Overview

Inputs

- **clk:** system clock
- **reset:** active-high reset
- **opcode [1:0]:** operation code (00: nop, 01: encrypt, 10: decrypt, 11: undefined)
- **data_in [15:0]:** plaintext or ciphertext input
- **key [31:0]:** 32-bit symmetric key

Outputs

- **data_out [15:0]:** result of operation
- **valid [1:0]:** status code (00: nop, 01: encryption OK, 10: decryption OK, 11: error)

2. Verification Environment

- **Methodology:** SystemVerilog UVM
- **Testbench Components:**
    - **spn_model_env:** Top-level environment
    - **spn_agent:** Contains driver, sequencer, and monitor
    - **spn_monitor:** Captures transactions from DUT and sends to scoreboard
    - **spn_scoreboard:** Compares DUT output with golden reference model
    - **spn_seq_item:** Encapsulates each transaction
    - **spn_sequences:** Multiple sequences to stimulate DUT under various conditions

3. Stimulus Plan

Sequence Name	Purpose	Status
`spn_sanity_seq`	Sweep opcode/reset combos (8 tests)	Done
`spn_random_seq`	Constrained random inputs (100 tests)	Done
`spn_corner_seq`	Edge-case data and key values	Done
`spn_error_seq`	Illegal opcodes (2'b11)	Done
`spn_key_sweep_seq`	Fixed data, random keys	Done
`spn_data_sweep_seq`	Fixed key, random data	Done

4. Output Checking

- Scoreboard compares DUT output against golden encryption/decryption models
- Mismatches reported via uvm_error
- Match results logged with uvm_info

5. Summary of Results

Sequence	Tests Run	Mismatches Found	Status
Sanity	8	0	Pass
Random	100	0	Pass
Corner	10	0	Pass
Error	10	0	Pass
Key Sweep	20	0	Pass
Data Sweep	20	0	Pass

6. Final Notes

- The DUT behaves correctly across all opcode and reset combinations
- All outputs match the expected golden reference results
- No mismatches or unexpected behaviors observed

Overall Result: PASS

Coverage Summary – SPN-CU Functional Verification

Objective

This section summarizes the functional coverage achieved during the verification of the SPN-CU (Substitution-Permutation Network Cryptographic Unit) using SystemVerilog and UVM methodology.

Functional Coverage Groups

Coverage Group	Description	Achieved (%)
cg_opcode_vs_valid (spn_monitor)	Cross-coverage of opcode and valid output combinations	81.25%
cg_outputs (spn_monitor)	Coverage on final output values (valid, data_out edge cases)	100.00%
cg_inputs (spn_seq_item)	Coverage on all stimulus inputs (opcode, key, data_in, reset, etc.)	100.00%

- **Total Functional Coverage Score: 93.75%**

Covergroup Definitions and Goals

1. **cg_outputs**

   - Captures edge cases of valid output and selected bins from data_out
   - Important for ensuring corner-case output scenarios are covered
   - Partial coverage indicates some edge cases (e.g. certain data_out bins) were not fully triggered

2. **cg_opcode_vs_valid**

   - Crosses opcode and valid output to ensure that each opcode results in the correct valid signal
   - Score < 100% suggests some opcode-valid combinations (opcode=10, opcode = 01) can't result in (valid = 11) since no problems occurred in the enc/dec processes) may not have occurred

3. **cg_inputs**

   - Thoroughly samples all input combinations, including:
     - opcode bins (NOP, ENCRYPT, DECRYPT, UNDEFINED)
     - reset states
     - Key patterns and data_in edge cases
     - Crosses opcode and reset
     - Fully covered

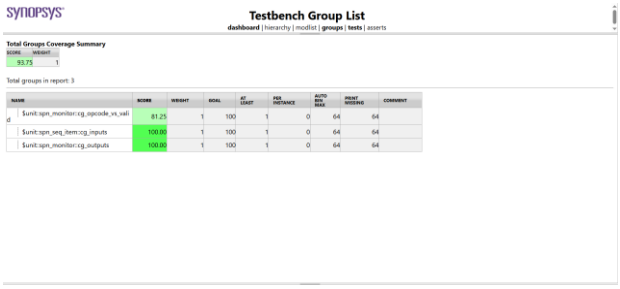
Summary of Achievements

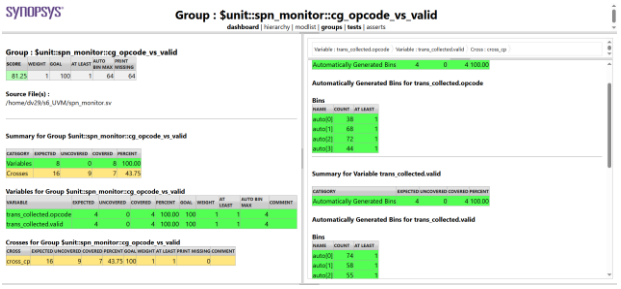
- Functional stimulus applied across all defined inputs
- All input combinations thoroughly tested and covered (100% on cg_inputs)
- All output combinations thoroughly tested and covered (100% on cg_outputs)
- Some rare combinations on opcode-valid cross may require additional stimuli

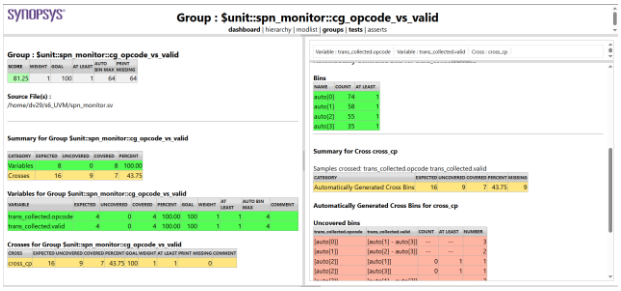
Conclusion

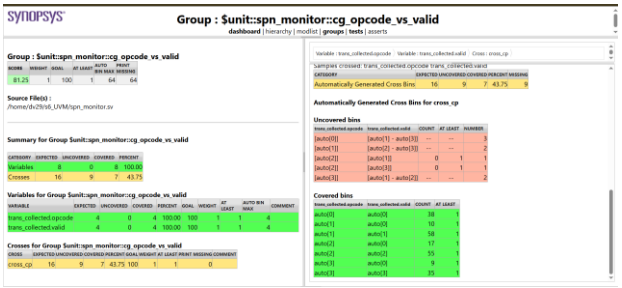
Functional coverage was the main metric used to validate the SPN-CU. With **93.75%** total group coverage, the testbench is close to full completeness, and the cg_inputs and cg_outputs groups are fully satisfied. To reach 100% coverage, focused tests on missed corner-case cross combinations. But it's impossible unless an error on the enc/dec operations occurs.

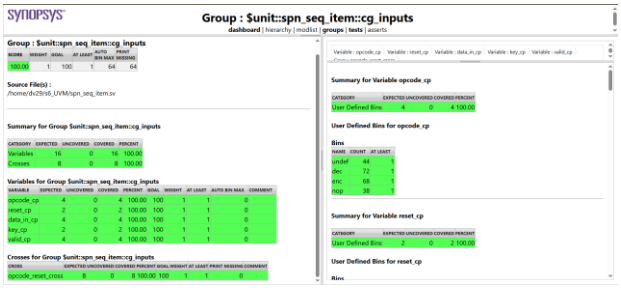
Results:

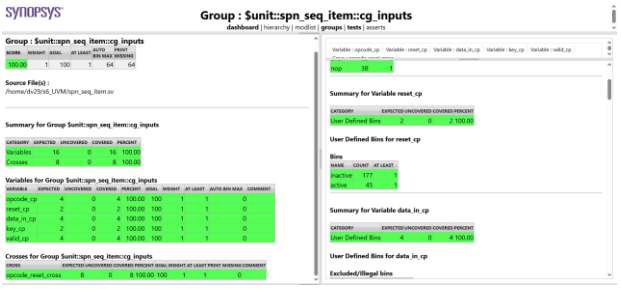

Testbench Group List
dashboard | hierarchy | modlist | groups | tests | asserts
Total Groups Coverage Summary
Total groups in report: 3
$unit::spn_monitor::cg_opcode_vs_valid
$unit::spn_seq_item::cg_inputs
$unit::spn_monitor::cg_outputs

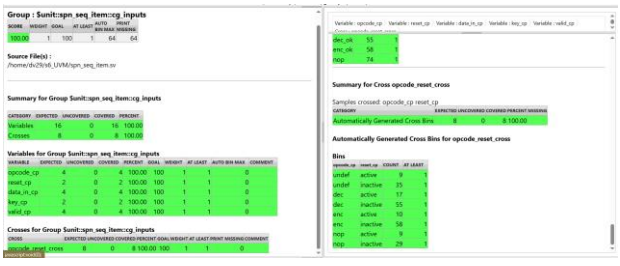

Group : $unit::spn_monitor::cg_opcode_vs_valid
dashboard | hierarchy | modlist | groups | tests | asserts
Group : $unit::spn_monitor::cg_opcode_vs_valid
Source File(s) :
/home/dv29/s6_UVM/spn_monitor.sv
Summary for Group $unit::spn_monitor::cg_opcode_vs_valid
Variables for Group $unit::spn_monitor::cg_opcode_vs_valid
Crosses for Group $unit::spn_monitor::cg_opcode_vs_valid
Automatically Generated Bins for trans_collected.opcode
Summary for Variable trans_collected.valid
Automatically Generated Bins for trans_collected.valid

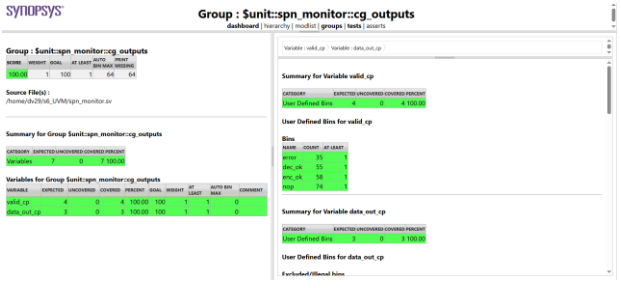

Group : $unit::spn_monitor::cg_opcode_vs_valid
dashboard | hierarchy | modlist | groups | tests | asserts
Group : $unit::spn_monitor::cg_opcode_vs_valid
Source File(s) :
/home/dv29/s6_UVM/spn_monitor.sv
Summary for Group $unit::spn_monitor::cg_opcode_vs_valid
Variables for Group $unit::spn_monitor::cg_opcode_vs_valid
Crosses for Group $unit::spn_monitor::cg_opcode_vs_valid
Summary for Cross cross_cp
Automatically Generated Cross Bins for cross_cp
Uncovered bins
Covered bins


Group : $unit::spn_monitor::cg_opcode_vs_valid
dashboard | hierarchy | modlist | groups | tests | asserts
Group : $unit::spn_monitor::cg_opcode_vs_valid
Source File(s) :
/home/dv29/s6_UVM/spn_monitor.sv
Summary for Group $unit::spn_monitor::cg_opcode_vs_valid
Variables for Group $unit::spn_monitor::cg_opcode_vs_valid
Crosses for Group $unit::spn_monitor::cg_opcode_vs_valid
Automatically Generated Cross Bins for cross_cp
Uncovered bins
Covered bins


Group : $unit::spn_seq_item::cg_inputs
dashboard | hierarchy | modlist | groups | tests | asserts
Group : $unit::spn_seq_item::cg_inputs
Source File(s) :
/home/dv29/s6_UVM/spn_seq_item.sv
Summary for Group $unit::spn_seq_item::cg_inputs
Variables for Group $unit::spn_seq_item::cg_inputs
Crosses for Group $unit::spn_seq_item::cg_inputs
Summary for Variable opcode_cp
User Defined Bins for opcode_cp
Summary for Variable reset_cp
User Defined Bins for reset_cp

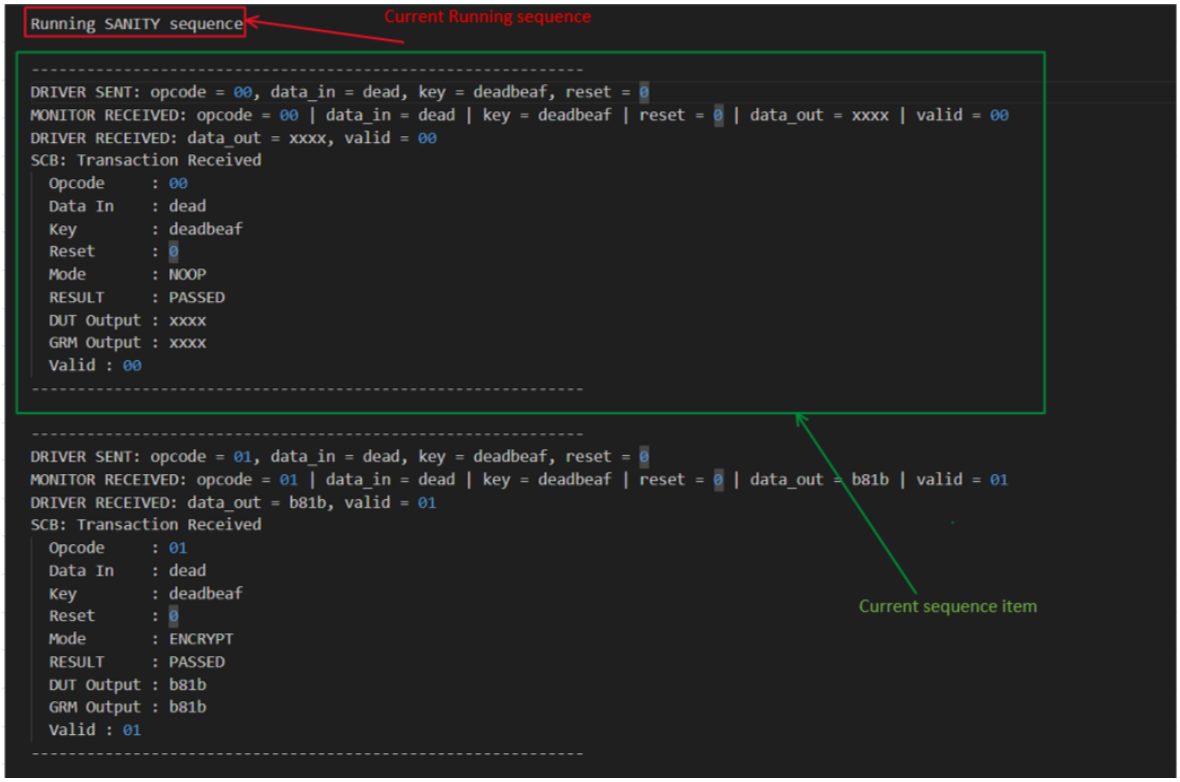

Group : $unit::spn_seq_item::cg_inputs
dashboard | hierarchy | modlist | groups | tests | asserts
Group : $unit::spn_seq_item::cg_inputs
Source File(s) :
/home/dv29/s6_UVM/spn_seq_item.sv
Summary for Group $unit::spn_seq_item::cg_inputs
Variables for Group $unit::spn_seq_item::cg_inputs
Crosses for Group $unit::spn_seq_item::cg_inputs
Summary for Variable reset_cp
User Defined Bins for reset_cp
Summary for Variable data_in_cp
User Defined Bins for data_in_cp
Excluded/Illegal bins


Group : $unit::spn_seq_item::cg_inputs
Source File(s) :
/home/dv29/s6_UVM/spn_seq_item.sv
Summary for Group $unit::spn_seq_item::cg_inputs
Variables for Group $unit::spn_seq_item::cg_inputs
Crosses for Group $unit::spn_seq_item::cg_inputs
Summary for Cross opcode_reset_cross
Automatically Generated Cross Bins for opcode_reset_cross
Bins


Group : $unit::spn_monitor::cg_outputs
dashboard | hierarchy | modlist | groups | tests | asserts
Group : $unit::spn_monitor::cg_outputs
Source File(s) :
/home/dv29/s6_UVM/spn_monitor.sv
Summary for Group $unit::spn_monitor::cg_outputs
Variables for Group $unit::spn_monitor::cg_outputs
Summary for Variable valid_cp
User Defined Bins for valid_cp
Summary for Variable data_out_cp
User Defined Bins for data_out_cp

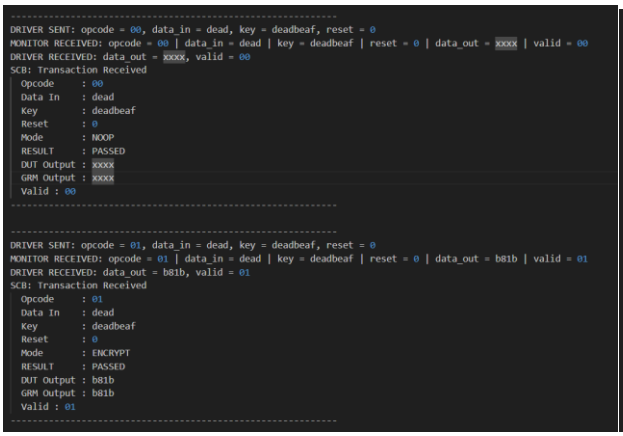
Simulation snapshots

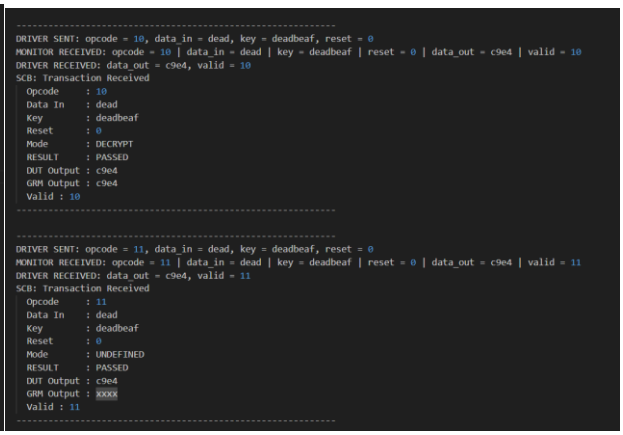
Output organization

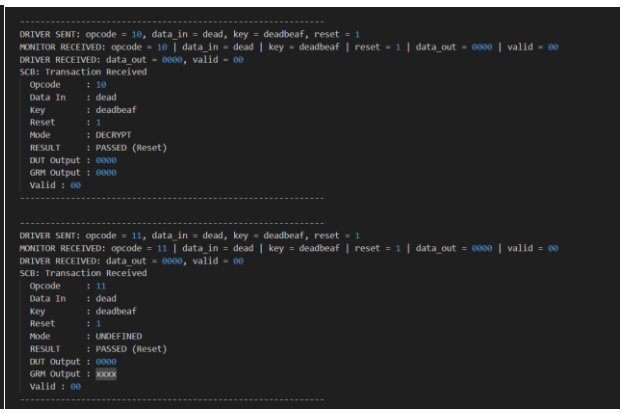


SANITY sequence

All sanity sequence outputs

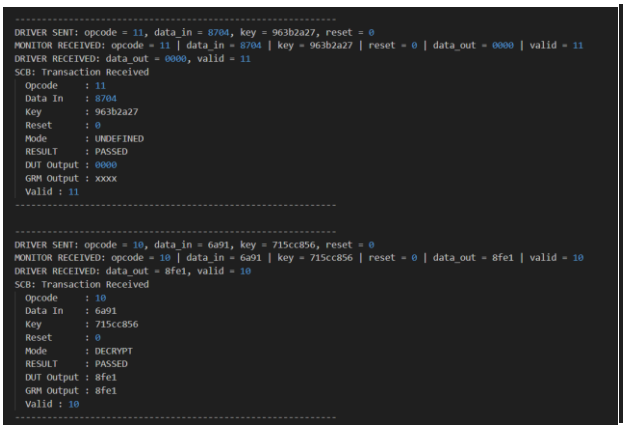






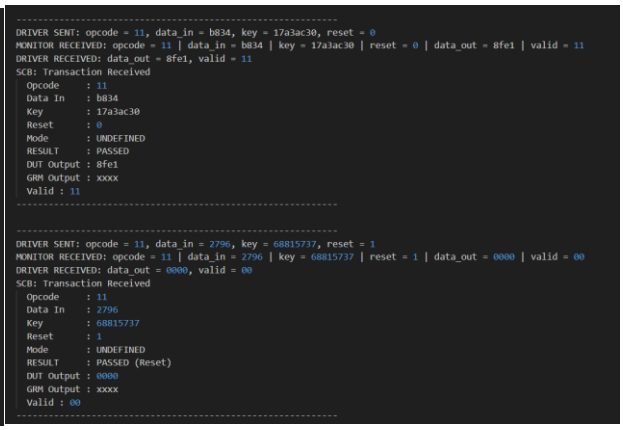
RANDOM sequence

some samples of the random sequence



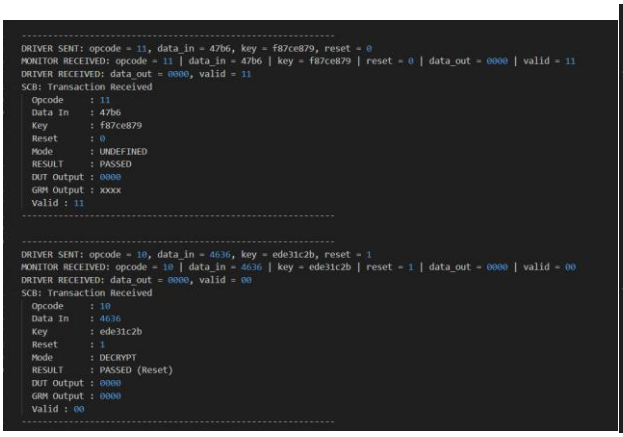
```
DRIVER SENT: opcode = 11, data_in = 8704, key = 963b2a27, reset = 0
MONITOR RECEIVED: opcode = 11 | data_in = 8704 | key = 963b2a27 | reset = 0 | data_out = 0000 | valid = 11
DRIVER RECEIVED: data_out = 0000, valid = 11
SCB: Transaction Received
  Opcode     : 11
  Data In    : 8704
  Key        : 963b2a27
  Reset      : 0
  Mode       : UNDEFINED
  RESULT     : PASSED
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 11
```

```
DRIVER SENT: opcode = 10, data_in = 6a91, key = 715cc856, reset = 0
MONITOR RECEIVED: opcode = 10 | data_in = 6a91 | key = 715cc856 | reset = 0 | data_out = 8fe1 | valid = 10
DRIVER RECEIVED: data_out = 8fe1, valid = 10
SCB: Transaction Received
  Opcode     : 10
  Data In    : 6a91
  Key        : 715cc856
  Reset      : 0
  Mode       : DECRYPT
  RESULT     : PASSED
  DUT Output : 8fe1
  GRM Output : 8fe1
  Valid : 10
```



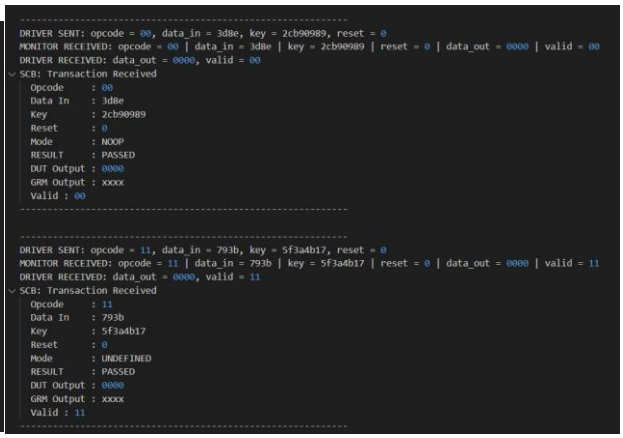
```
DRIVER SENT: opcode = 11, data_in = b834, key = 17a3ac30, reset = 0
MONITOR RECEIVED: opcode = 11 | data_in = b834 | key = 17a3ac30 | reset = 0 | data_out = 8fe1 | valid = 11
DRIVER RECEIVED: data_out = 8fe1, valid = 11
SCB: Transaction Received
  Opcode     : 11
  Data In    : b834
  Key        : 17a3ac30
  Reset      : 0
  Mode       : UNDEFINED
  RESULT     : PASSED
  DUT Output : 8fe1
  GRM Output : xxxx
  Valid : 11
```

```
DRIVER SENT: opcode = 11, data_in = 2796, key = 68815737, reset = 1
MONITOR RECEIVED: opcode = 11 | data_in = 2796 | key = 68815737 | reset = 1 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 11
  Data In    : 2796
  Key        : 68815737
  Reset      : 1
  Mode       : UNDEFINED
  RESULT     : PASSED (Reset)
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 00
```



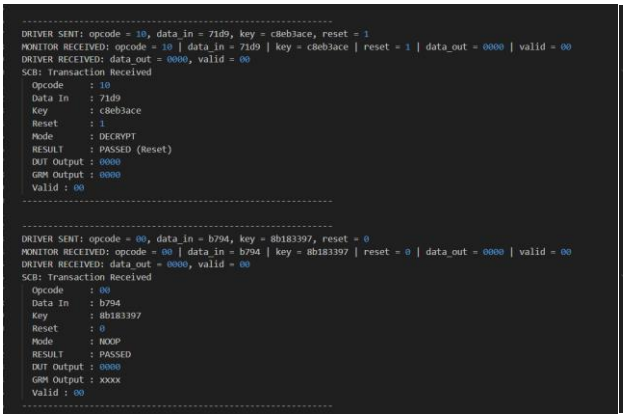
```
DRIVER SENT: opcode = 11, data_in = 47b6, key = f87ce879, reset = 0
MONITOR RECEIVED: opcode = 11 | data_in = 47b6 | key = f87ce879 | reset = 0 | data_out = 0000 | valid = 11
DRIVER RECEIVED: data_out = 0000, valid = 11
SCB: Transaction Received
  Opcode     : 11
  Data In    : 47b6
  Key        : f87ce879
  Reset      : 0
  Mode       : UNDEFINED
  RESULT     : PASSED
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 11
```

```
DRIVER SENT: opcode = 10, data_in = 4636, key = ede31c2b, reset = 1
MONITOR RECEIVED: opcode = 10 | data_in = 4636 | key = ede31c2b | reset = 1 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 10
  Data In    : 4636
  Key        : ede31c2b
  Reset      : 1
  Mode       : DECRYPT
  RESULT     : PASSED (Reset)
  DUT Output : 0000
  GRM Output : 0000
  Valid : 00
```



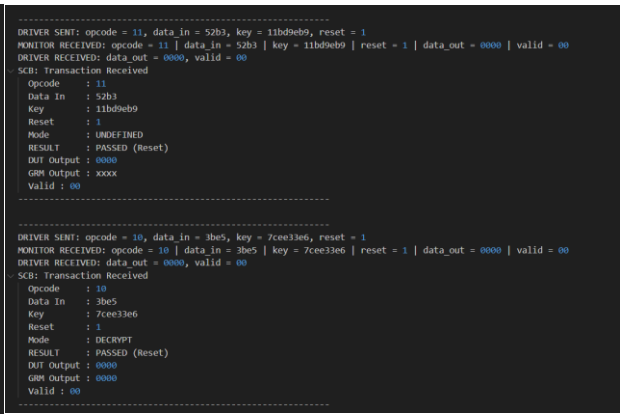
```
DRIVER SENT: opcode = 00, data_in = 3d8e, key = 2cb90989, reset = 0
MONITOR RECEIVED: opcode = 00 | data_in = 3d8e | key = 2cb90989 | reset = 0 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 00
  Data In    : 3d8e
  Key        : 2cb90989
  Reset      : 0
  Mode       : NOOP
  RESULT     : PASSED
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 00
```

```
DRIVER SENT: opcode = 11, data_in = 793b, key = 5f3a4b17, reset = 0
MONITOR RECEIVED: opcode = 11 | data_in = 793b | key = 5f3a4b17 | reset = 0 | data_out = 0000 | valid = 11
DRIVER RECEIVED: data_out = 0000, valid = 11
SCB: Transaction Received
  Opcode     : 11
  Data In    : 793b
  Key        : 5f3a4b17
  Reset      : 0
  Mode       : UNDEFINED
  RESULT     : PASSED
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 11
```



```
DRIVER SENT: opcode = 10, data_in = 71d9, key = c8eb3ace, reset = 1
MONITOR RECEIVED: opcode = 10 | data_in = 71d9 | key = c8eb3ace | reset = 1 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 10
  Data In    : 71d9
  Key        : c8eb3ace
  Reset      : 1
  Mode       : DECRYPT
  RESULT     : PASSED (Reset)
  DUT Output : 0000
  GRM Output : 0000
  Valid : 00
```

```
DRIVER SENT: opcode = 00, data_in = b794, key = 8b183397, reset = 0
MONITOR RECEIVED: opcode = 00 | data_in = b794 | key = 8b183397 | reset = 0 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 00
  Data In    : b794
  Key        : 8b183397
  Reset      : 0
  Mode       : NOOP
  RESULT     : PASSED
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 00
```

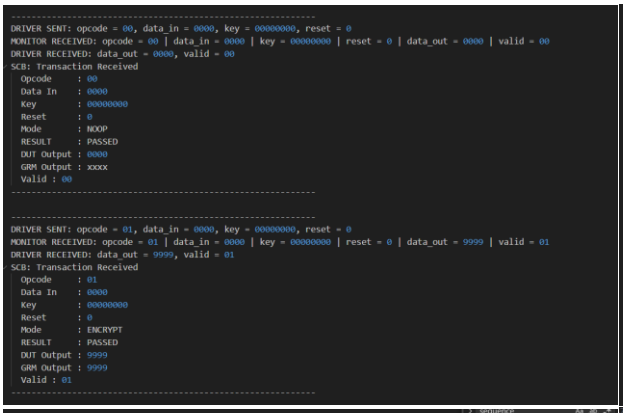


```
DRIVER SENT: opcode = 11, data_in = 52b3, key = 11bd9eb9, reset = 1
MONITOR RECEIVED: opcode = 11 | data_in = 52b3 | key = 11bd9eb9 | reset = 1 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 11
  Data In    : 52b3
  Key        : 11bd9eb9
  Reset      : 1
  Mode       : UNDEFINED
  RESULT     : PASSED (Reset)
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 00
```

```
DRIVER SENT: opcode = 10, data_in = 3be5, key = 7cee33e6, reset = 1
MONITOR RECEIVED: opcode = 10 | data_in = 3be5 | key = 7cee33e6 | reset = 1 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 10
  Data In    : 3be5
  Key        : 7cee33e6
  Reset      : 1
  Mode       : DECRYPT
  RESULT     : PASSED (Reset)
  DUT Output : 0000
  GRM Output : 0000
  Valid : 00
```

CORNER CASE sequence

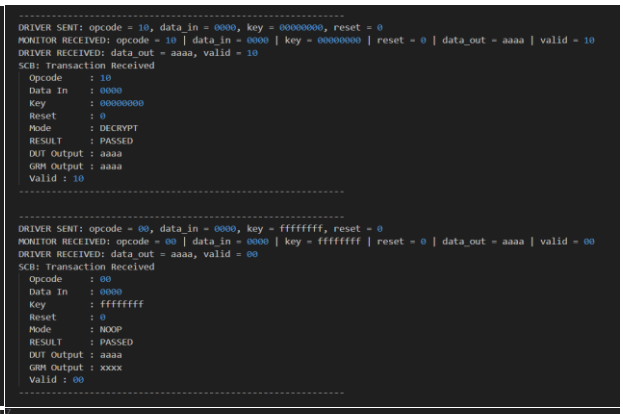
some samples of the corner case sequence



```
DRIVER SENT: opcode = 00, data_in = 0000, key = 00000000, reset = 0
MONITOR RECEIVED: opcode = 00 | data_in = 0000 | key = 00000000 | reset = 0 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 00
  Data In    : 0000
  Key        : 00000000
  Reset      : 0
  Mode       : NOOP
  RESULT     : PASSED
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 00
```



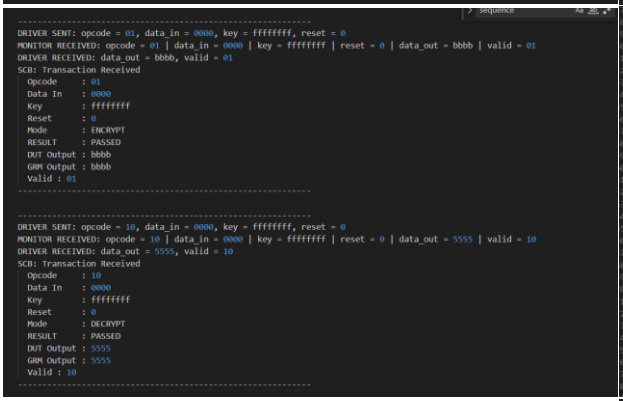
```
DRIVER SENT: opcode = 01, data_in = 0000, key = 00000000, reset = 0
MONITOR RECEIVED: opcode = 01 | data_in = 0000 | key = 00000000 | reset = 0 | data_out = 9999 | valid = 01
DRIVER RECEIVED: data_out = 9999, valid = 01
SCB: Transaction Received
  Opcode     : 01
  Data In    : 0000
  Key        : 00000000
  Reset      : 0
  Mode       : ENCRYPT
  RESULT     : PASSED
  DUT Output : 9999
  GRM Output : 9999
  Valid : 01
```



```
DRIVER SENT: opcode = 10, data_in = 0000, key = 00000000, reset = 0
MONITOR RECEIVED: opcode = 10 | data_in = 0000 | key = 00000000 | reset = 0 | data_out = aaaa | valid = 10
DRIVER RECEIVED: data_out = aaaa, valid = 10
SCB: Transaction Received
  Opcode     : 10
  Data In    : 0000
  Key        : 00000000
  Reset      : 0
  Mode       : DECRYPT
  RESULT     : PASSED
  DUT Output : aaaa
  GRM Output : aaaa
  Valid : 10
```

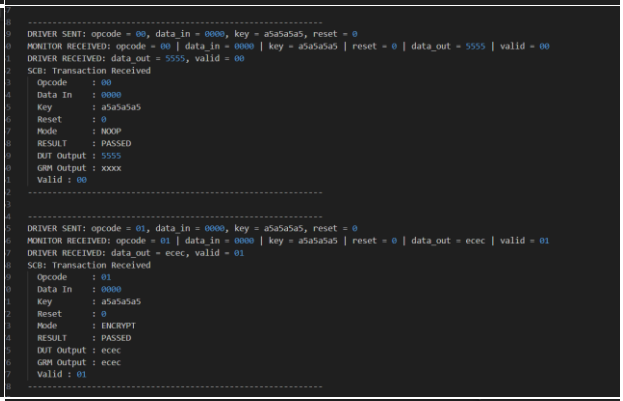


```
DRIVER SENT: opcode = 00, data_in = 0000, key = ffffffff, reset = 0
MONITOR RECEIVED: opcode = 00 | data_in = 0000 | key = ffffffff | reset = 0 | data_out = aaaa | valid = 00
DRIVER RECEIVED: data_out = aaaa, valid = 00
SCB: Transaction Received
  Opcode     : 00
  Data In    : 0000
  Key        : ffffffff
  Reset      : 0
  Mode       : NOOP
  RESULT     : PASSED
  DUT Output : aaaa
  GRM Output : xxxx
  Valid : 00
```



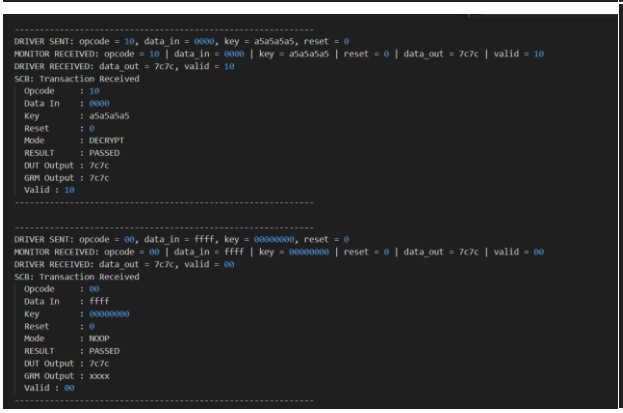
```
DRIVER SENT: opcode = 01, data_in = 0000, key = ffffffff, reset = 0
MONITOR RECEIVED: opcode = 01 | data_in = 0000 | key = ffffffff | reset = 0 | data_out = bbbb | valid = 01
DRIVER RECEIVED: data_out = bbbb, valid = 01
SCB: Transaction Received
  Opcode     : 01
  Data In    : 0000
  Key        : ffffffff
  Reset      : 0
  Mode       : ENCRYPT
  RESULT     : PASSED
  DUT Output : bbbb
  GRM Output : bbbb
  Valid : 01
```

```
DRIVER SENT: opcode = 10, data_in = 0000, key = ffffffff, reset = 0
MONITOR RECEIVED: opcode = 10 | data_in = 0000 | key = ffffffff | reset = 0 | data_out = 5555 | valid = 10
DRIVER RECEIVED: data_out = 5555, valid = 10
SCB: Transaction Received
  Opcode     : 10
  Data In    : 0000
  Key        : ffffffff
  Reset      : 0
  Mode       : DECRYPT
  RESULT     : PASSED
  DUT Output : 5555
  GRM Output : 5555
  Valid : 10
```



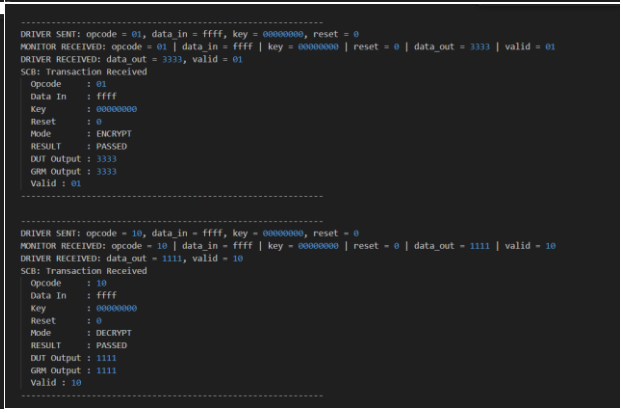
```
DRIVER SENT: opcode = 00, data_in = 0000, key = a5a5a5a5, reset = 0
MONITOR RECEIVED: opcode = 00 | data_in = 0000 | key = a5a5a5a5 | reset = 0 | data_out = 5555 | valid = 00
DRIVER RECEIVED: data_out = 5555, valid = 00
SCB: Transaction Received
  Opcode     : 00
  Data In    : 0000
  Key        : a5a5a5a5
  Reset      : 0
  Mode       : NOOP
  RESULT     : PASSED
  DUT Output : 5555
  GRM Output : xxxx
  Valid : 00
```

```
DRIVER SENT: opcode = 01, data_in = 0000, key = a5a5a5a5, reset = 0
MONITOR RECEIVED: opcode = 01 | data_in = 0000 | key = a5a5a5a5 | reset = 0 | data_out = ecec | valid = 01
DRIVER RECEIVED: data_out = ecec, valid = 01
SCB: Transaction Received
  Opcode     : 01
  Data In    : 0000
  Key        : a5a5a5a5
  Reset      : 0
  Mode       : ENCRYPT
  RESULT     : PASSED
  DUT Output : ecec
  GRM Output : ecec
  Valid : 01
```



```
DRIVER SENT: opcode = 10, data_in = 0000, key = a5a5a5a5, reset = 0
MONITOR RECEIVED: opcode = 10 | data_in = 0000 | key = a5a5a5a5 | reset = 0 | data_out = 7c7c | valid = 10
DRIVER RECEIVED: data_out = 7c7c, valid = 10
SCB: Transaction Received
  Opcode     : 10
  Data In    : 0000
  Key        : a5a5a5a5
  Reset      : 0
  Mode       : DECRYPT
  RESULT     : PASSED
  DUT Output : 7c7c
  GRM Output : 7c7c
  Valid : 10
```

```
DRIVER SENT: opcode = 00, data_in = ffff, key = 00000000, reset = 0
MONITOR RECEIVED: opcode = 00 | data_in = ffff | key = 00000000 | reset = 0 | data_out = 7c7c | valid = 00
DRIVER RECEIVED: data_out = 7c7c, valid = 00
SCB: Transaction Received
  Opcode     : 00
  Data In    : ffff
  Key        : 00000000
  Reset      : 0
  Mode       : NOOP
  RESULT     : PASSED
  DUT Output : 7c7c
  GRM Output : xxxx
  Valid : 00
```

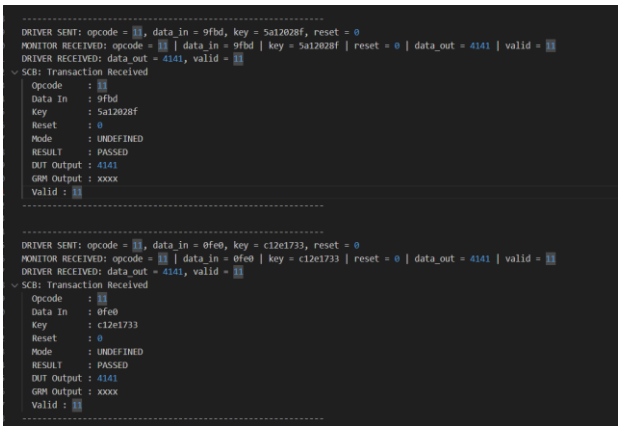


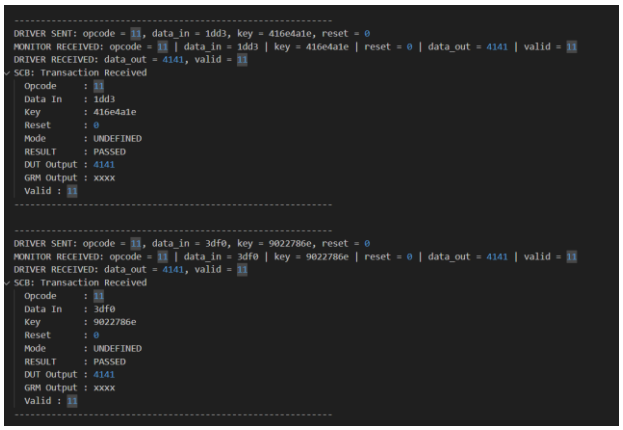
```
DRIVER SENT: opcode = 01, data_in = ffff, key = 00000000, reset = 0
MONITOR RECEIVED: opcode = 01 | data_in = ffff | key = 00000000 | reset = 0 | data_out = 3333 | valid = 01
DRIVER RECEIVED: data_out = 3333, valid = 01
SCB: Transaction Received
  Opcode     : 01
  Data In    : ffff
  Key        : 00000000
  Reset      : 0
  Mode       : ENCRYPT
  RESULT     : PASSED
  DUT Output : 3333
  GRM Output : 3333
  Valid : 01
```

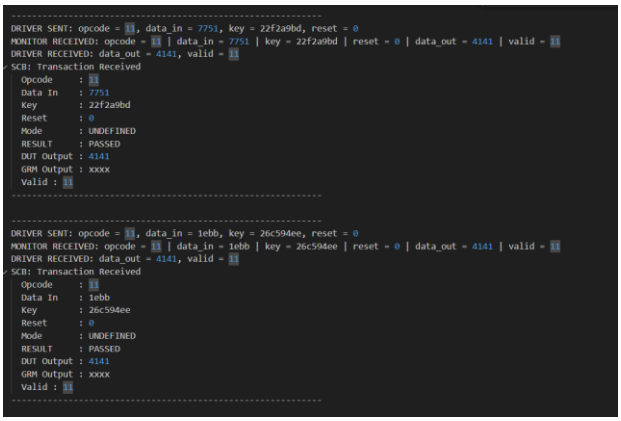
```
DRIVER SENT: opcode = 10, data_in = ffff, key = 00000000, reset = 0
MONITOR RECEIVED: opcode = 10 | data_in = ffff | key = 00000000 | reset = 0 | data_out = 1111 | valid = 10
DRIVER RECEIVED: data_out = 1111, valid = 10
SCB: Transaction Received
  Opcode     : 10
  Data In    : ffff
  Key        : 00000000
  Reset      : 0
  Mode       : DECRYPT
  RESULT     : PASSED
  DUT Output : 1111
  GRM Output : 1111
  Valid : 10
```

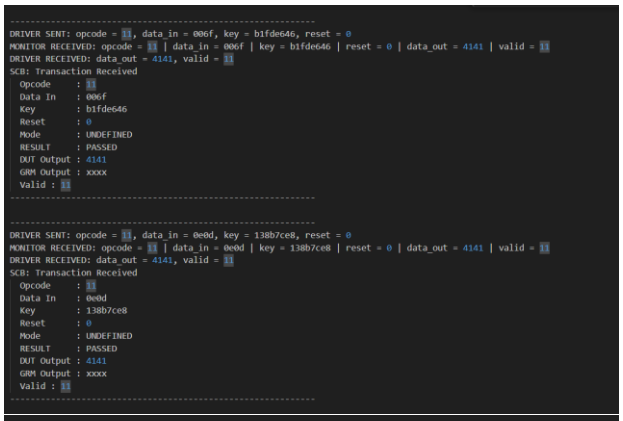
Error sequence

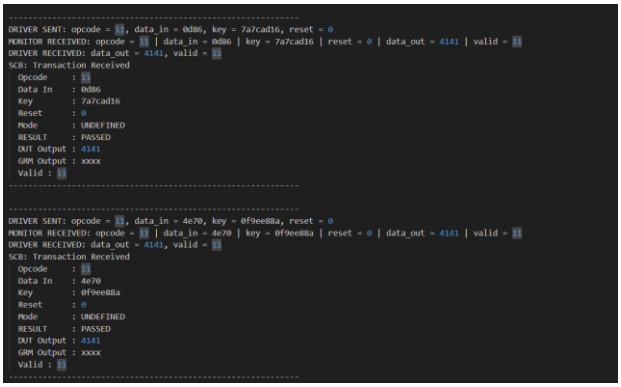
some samples of the error sequence

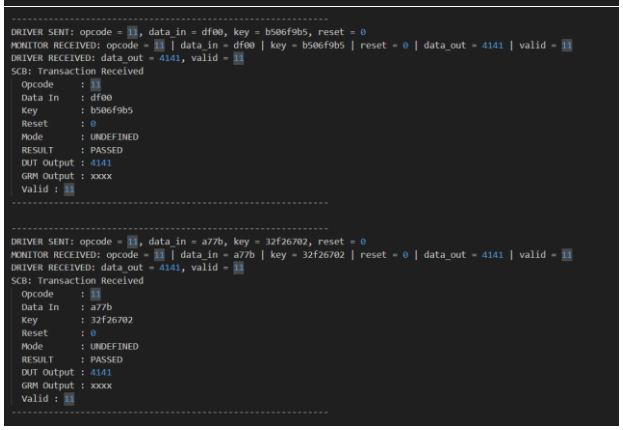






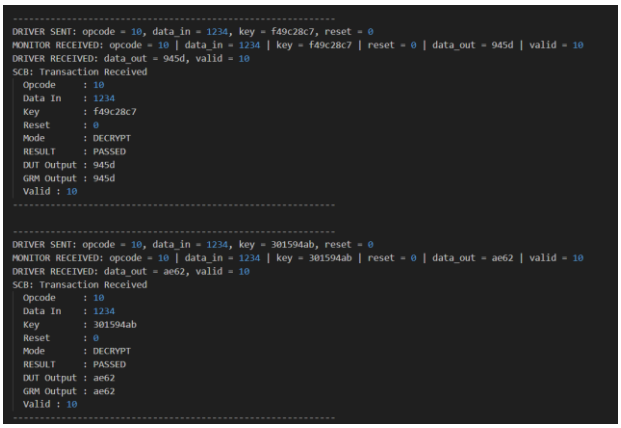


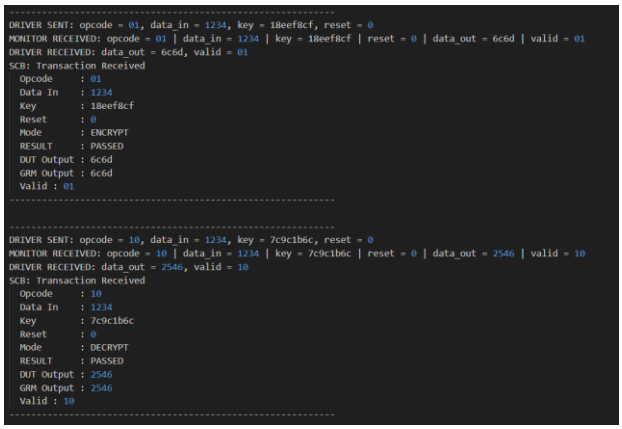




key sweep sequence

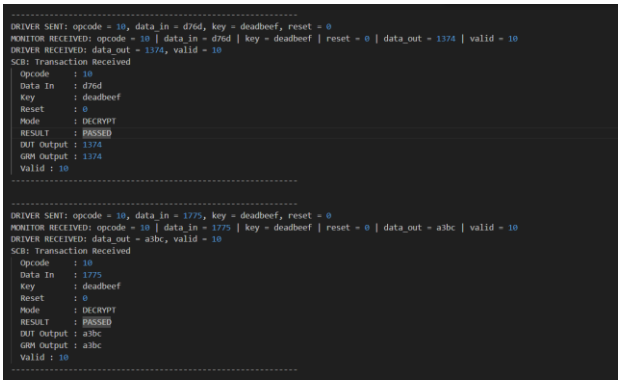
some samples of the key sweep sequence

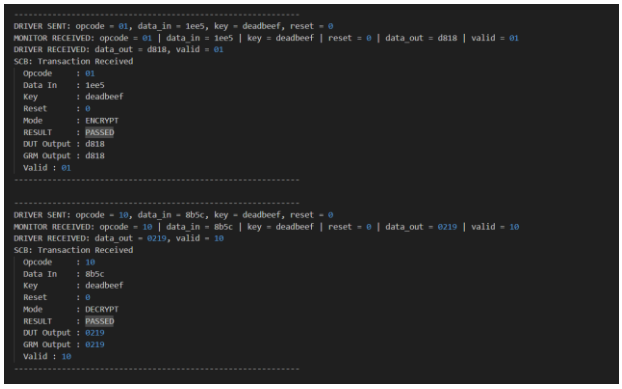




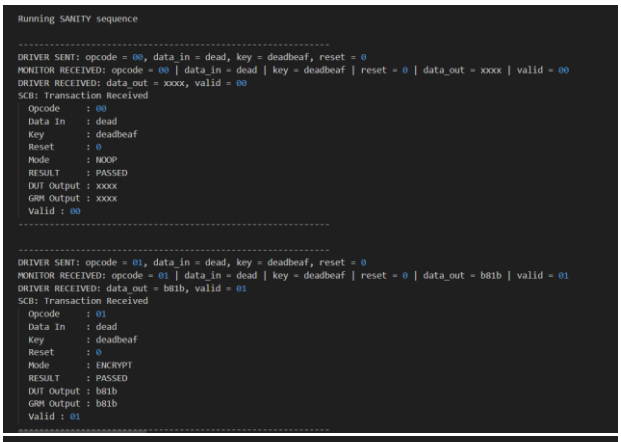
Data sweep sequence

some samples of the data sweep sequence

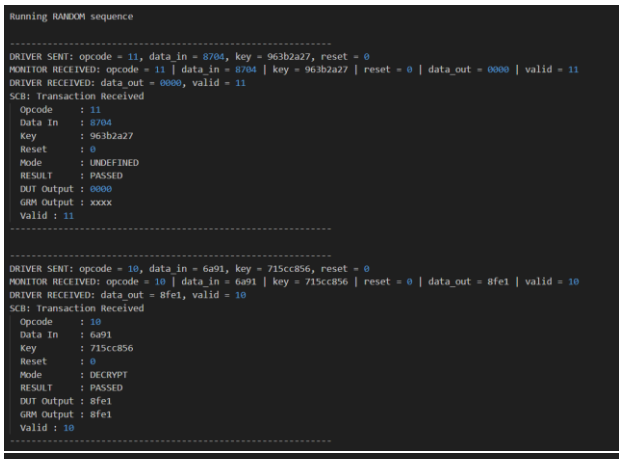




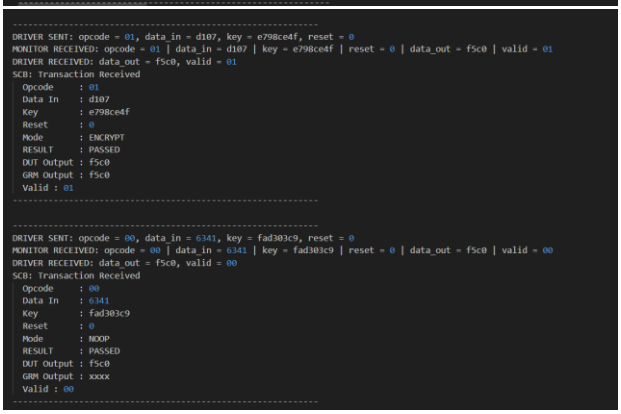
Some random shots



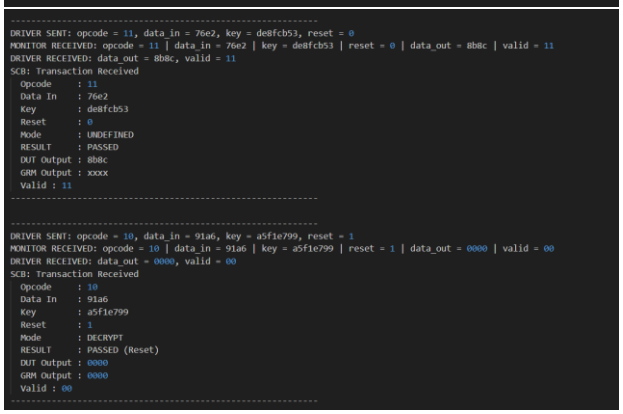
```
Running SANITY sequence

------------------------------------------------------------
DRIVER SENT: opcode = 00, data_in = dead, key = deadbeaf, reset = 0
MONITOR RECEIVED: opcode = 00 | data_in = dead | key = deadbeaf | reset = 0 | data_out = xxxx | valid = 00
DRIVER RECEIVED: data_out = xxxx, valid = 00
SCB: Transaction Received
  Opcode     : 00
  Data In    : dead
  Key        : deadbeaf
  Reset      : 0
  Mode       : NOOP
  RESULT     : PASSED
  DUT Output : xxxx
  GRM Output : xxxx
  Valid : 00
------------------------------------------------------------


------------------------------------------------------------
DRIVER SENT: opcode = 01, data_in = dead, key = deadbeaf, reset = 0
MONITOR RECEIVED: opcode = 01 | data_in = dead | key = deadbeaf | reset = 0 | data_out = b81b | valid = 01
DRIVER RECEIVED: data_out = b81b, valid = 01
SCB: Transaction Received
  Opcode     : 01
  Data In    : dead
  Key        : deadbeaf
  Reset      : 0
  Mode       : ENCRYPT
  RESULT     : PASSED
  DUT Output : b81b
  GRM Output : b81b
  Valid : 01
------------------------------------------------------------
```



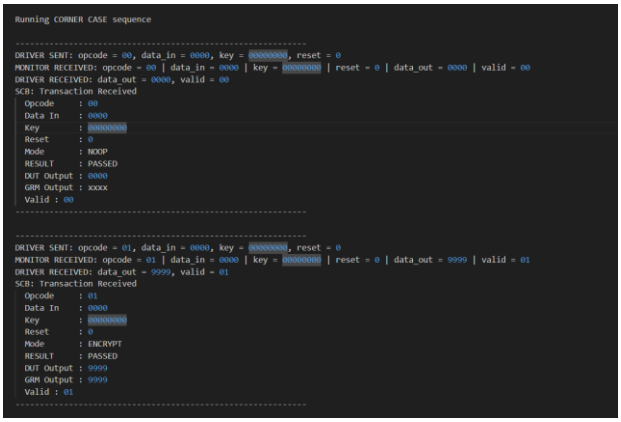
```
Running RANDOM sequence

------------------------------------------------------------
DRIVER SENT: opcode = 11, data_in = 8704, key = 963b2a27, reset = 0
MONITOR RECEIVED: opcode = 11 | data_in = 8704 | key = 963b2a27 | reset = 0 | data_out = 0000 | valid = 11
DRIVER RECEIVED: data_out = 0000, valid = 11
SCB: Transaction Received
  Opcode     : 11
  Data In    : 8704
  Key        : 963b2a27
  Reset      : 0
  Mode       : UNDEFINED
  RESULT     : PASSED
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 11
------------------------------------------------------------


------------------------------------------------------------
DRIVER SENT: opcode = 10, data_in = 6a91, key = 715cc856, reset = 0
MONITOR RECEIVED: opcode = 10 | data_in = 6a91 | key = 715cc856 | reset = 0 | data_out = 8fe1 | valid = 10
DRIVER RECEIVED: data_out = 8fe1, valid = 10
SCB: Transaction Received
  Opcode     : 10
  Data In    : 6a91
  Key        : 715cc856
  Reset      : 0
  Mode       : DECRYPT
  RESULT     : PASSED
  DUT Output : 8fe1
  GRM Output : 8fe1
  Valid : 10
------------------------------------------------------------
```



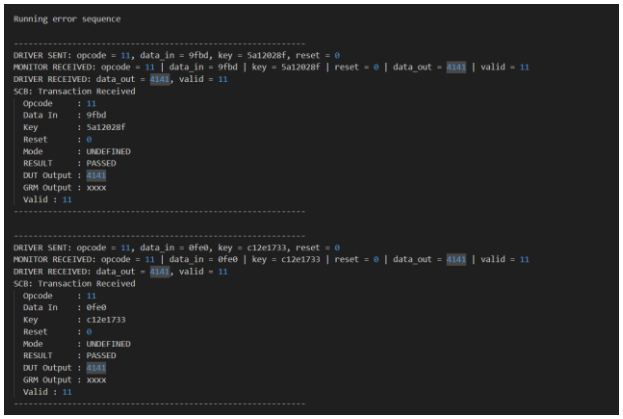
```
------------------------------------------------------------
DRIVER SENT: opcode = 01, data_in = d107, key = e798ce4f, reset = 0
MONITOR RECEIVED: opcode = 01 | data_in = d107 | key = e798ce4f | reset = 0 | data_out = f5c0 | valid = 01
DRIVER RECEIVED: data_out = f5c0, valid = 01
SCB: Transaction Received
  Opcode     : 01
  Data In    : d107
  Key        : e798ce4f
  Reset      : 0
  Mode       : ENCRYPT
  RESULT     : PASSED
  DUT Output : f5c0
  GRM Output : f5c0
  Valid : 01
------------------------------------------------------------


------------------------------------------------------------
DRIVER SENT: opcode = 00, data_in = 6341, key = fad303c9, reset = 0
MONITOR RECEIVED: opcode = 00 | data_in = 6341 | key = fad303c9 | reset = 0 | data_out = f5c0 | valid = 00
DRIVER RECEIVED: data_out = f5c0, valid = 00
SCB: Transaction Received
  Opcode     : 00
  Data In    : 6341
  Key        : fad303c9
  Reset      : 0
  Mode       : NOOP
  RESULT     : PASSED
  DUT Output : f5c0
  GRM Output : xxxx
  Valid : 00
------------------------------------------------------------
```



```
------------------------------------------------------------
DRIVER SENT: opcode = 11, data_in = 76e2, key = de8fcb53, reset = 0
MONITOR RECEIVED: opcode = 11 | data_in = 76e2 | key = de8fcb53 | reset = 0 | data_out = 8b8c | valid = 11
DRIVER RECEIVED: data_out = 8b8c, valid = 11
SCB: Transaction Received
  Opcode     : 11
  Data In    : 76e2
  Key        : de8fcb53
  Reset      : 0
  Mode       : UNDEFINED
  RESULT     : PASSED
  DUT Output : 8b8c
  GRM Output : xxxx
  Valid : 11
------------------------------------------------------------


------------------------------------------------------------
DRIVER SENT: opcode = 10, data_in = 91a6, key = a5f1e799, reset = 1
MONITOR RECEIVED: opcode = 10 | data_in = 91a6 | key = a5f1e799 | reset = 1 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 10
  Data In    : 91a6
  Key        : a5f1e799
  Reset      : 1
  Mode       : DECRYPT
  RESULT     : PASSED (Reset)
  DUT Output : 0000
  GRM Output : 0000
  Valid : 00
------------------------------------------------------------
```



```
Running CORNER CASE sequence

------------------------------------------------------------
DRIVER SENT: opcode = 00, data_in = 0000, key = 00000000, reset = 0
MONITOR RECEIVED: opcode = 00 | data_in = 0000 | key = 00000000 | reset = 0 | data_out = 0000 | valid = 00
DRIVER RECEIVED: data_out = 0000, valid = 00
SCB: Transaction Received
  Opcode     : 00
  Data In    : 0000
  Key        : 00000000
  Reset      : 0
  Mode       : NOOP
  RESULT     : PASSED
  DUT Output : 0000
  GRM Output : xxxx
  Valid : 00
------------------------------------------------------------


------------------------------------------------------------
DRIVER SENT: opcode = 01, data_in = 0000, key = 00000000, reset = 0
MONITOR RECEIVED: opcode = 01 | data_in = 0000 | key = 00000000 | reset = 0 | data_out = 9999 | valid = 01
DRIVER RECEIVED: data_out = 9999, valid = 01
SCB: Transaction Received
  Opcode     : 01
  Data In    : 0000
  Key        : 00000000
  Reset      : 0
  Mode       : ENCRYPT
  RESULT     : PASSED
  DUT Output : 9999
  GRM Output : 9999
  Valid : 01
------------------------------------------------------------
```



```
Running error sequence

------------------------------------------------------------
DRIVER SENT: opcode = 11, data_in = 9fbd, key = 5a12028f, reset = 0
MONITOR RECEIVED: opcode = 11 | data_in = 9fbd | key = 5a12028f | reset = 0 | data_out = 4141 | valid = 11
DRIVER RECEIVED: data_out = 4141, valid = 11
SCB: Transaction Received
  Opcode     : 11
  Data In    : 9fbd
  Key        : 5a12028f
  Reset      : 0
  Mode       : UNDEFINED
  RESULT     : PASSED
  DUT Output : 4141
  GRM Output : xxxx
  Valid : 11
------------------------------------------------------------


------------------------------------------------------------
DRIVER SENT: opcode = 11, data_in = 0fe0, key = c12e1733, reset = 0
MONITOR RECEIVED: opcode = 11 | data_in = 0fe0 | key = c12e1733 | reset = 0 | data_out = 4141 | valid = 11
DRIVER RECEIVED: data_out = 4141, valid = 11
SCB: Transaction Received
  Opcode     : 11
  Data In    : 0fe0
  Key        : c12e1733
  Reset      : 0
  Mode       : UNDEFINED
  RESULT     : PASSED
  DUT Output : 4141
  GRM Output : xxxx
  Valid : 11
------------------------------------------------------------
```