Reflection Report: CPM Verification

Student: Assaf Afriat
Date: February 15, 2026

1. Executive Summary

The objective of this project was to verify a Configurable Packet Modifier (CPM) against Design Specification v1.1. We developed a comprehensive UVM testbench utilizing a dual-agent architecture (Packet Agent, Register Agent), RAL integration, reference model, and functional coverage. The verification environment achieved 100% functional coverage on all mode and opcode combinations while processing over 500 packets across multiple test scenarios. The verification process uncovered two RTL bugs in version 1.0 (COUNT_OUT increment on out_valid instead of out_fire, output instability during backpressure), both of which were fixed in RTL v1.1. The environment also identified and documented 10+ specification ambiguities that required clarification.

2. Expectations vs. Reality

Based on the Design Specification, we expected a compliant packet processor that correctly transforms payloads based on 4 modes (PASS, XOR, ADD, ROT), maintains strict FIFO ordering, handles backpressure gracefully, and preserves the counter invariant COUNT_IN == COUNT_OUT + DROPPED_COUNT.

What We Found (The Reality): The verification environment exposed several deviations and ambiguities:

1. **COUNT_OUT Increment Bug:** The DUT incremented COUNT_OUT on out_valid instead of out_fire, causing multiple counts per packet during backpressure (RTL Bug - Fixed in v1.1).
2. **Output Stability Violation:** Output signals changed while out_valid=1 and out_ready=0, violating the stall stability rule (RTL Bug - Fixed in v1.1).
3. **ROT_AMT Undefined:** Specification mentioned ROT_AMT parameter but never defined its value. Clarified in v1.1 as fixed at 4 bits.
4. **Counter Invariant Timing:** Specification said "when stable" but didn't define the condition. Clarified in v1.1: invariant holds when STATUS.BUSY=0.
5. **Ordering vs. Latency Ambiguity:** Spec stated "no reordering" but different modes have different latencies. Resolved: DUT maintains FIFO regardless of mode latency.
6. **ENABLE Deassertion Behavior:** What happens to in-flight packets when ENABLE goes low? RTL flushes pipeline (implementation-defined).
7. **Counter Overflow:** 32-bit counter wrap behavior undefined. RTL uses standard wrap-around.
8. **SOFT_RST Scope:** Unclear if soft reset flushes pipeline. Spec v1.1 clarified: clears counters and internal state.
9. **in_ready When Disabled:** Spec didn't specify in_ready value when ENABLE=0. RTL correctly deasserts in_ready.

3. Verification Challenge: The "Configuration Timing" Problem

The Concept: The most significant intellectual challenge was verifying that packet transformations used the *correct* configuration. When MODE or PARAMS registers change during traffic, which configuration applies to each packet?

The Insight: A naive implementation would read the RAL mirror at scoreboard prediction time. This caused false mismatches when configuration changed between packet acceptance and output comparison. With high traffic (1000+ packets), packets with identical ID+OPCODE but different configurations were mis-matched.

The Strategy: We implemented "configuration capture at acceptance time" - the monitor reads MODE, MASK, and ADD_CONST from the RAL mirror the instant each packet is accepted (in_fire) and stores these values in the transaction. The reference model then uses these captured values, not the current mirror, for prediction.

The Lesson: This highlighted that verification of pipelined designs requires tracking *when* configuration was sampled, not just *what* the configuration is. The verification engineer must think in terms of packet lifetimes, not instantaneous state.

4. Future Testing Opportunities

To further harden the design and ensure silicon readiness, we recommend expanding the testbench to include:

- **Pipeline Stress Testing:** Rapidly toggling ENABLE while packets are in-flight to verify no data corruption occurs during flush.
- **Soft Reset During Traffic:** Asserting SOFT_RST while the pipeline is full to verify clean state recovery and counter reset.
- **Counter Overflow Verification:** Running extended simulations (4B+ packets) to verify 32-bit counter wrap-around behavior.
- **Mode Switching Under Load:** Changing MODE register every clock cycle while sustaining maximum throughput to stress the configuration sampling logic.
- **Invalid Register Access:** Writing to read-only registers and reading from undefined addresses to verify no side effects.
- **Boundary Value Testing:** Testing edge cases like MASK=0xFFFF, ADD_CONST=0xFFFF, and payload values at boundaries.

5. Final Assessment

The verification environment proved to be highly effective. It achieved 100% coverage of MODE, OPCODE, and their cross-product while detecting two critical RTL bugs that would have caused counter corruption and protocol violations in a real system. The separation of the Scoreboard (data integrity), Reference Model (transformation correctness), and SVA Assertions (protocol compliance) was crucial in isolating distinct failure types. The RAL integration enabled portable, maintainable test sequences, while the callback mechanism provided runtime statistics without modifying core components. The comprehensive bug tracking process documented 12 issues, all of which were resolved through specification clarification or RTL fixes.