CPM Verification Plan

Project: Configurable Packet Modifier (CPM)

Author: Assaf Afriat

Date: 2026-02-01

Version: 1.0

Status: Planning Phase

Phase 1 - Requirements & Stakeholder Analysis

Before developing the verification plan, the following questions were identified for clarification with relevant stakeholders. Understanding these requirements ensures alignment between the design specification and verification approach.

Digital Design Team - Questions

1. **Fixed vs. Variable Latency:** According to the specification, latency is mode-dependent (0-2 cycles). Does each MODE always result in a fixed latency (PASS=0, XOR=1, ADD=2, ROT=1), or can it vary dynamically based on pipeline state?
2. **Configuration Timing:** If the MODE or PARAMS registers are updated while in_valid is high and a packet is being accepted, does the DUT apply the new configuration to the current packet immediately, or is the configuration captured at the moment of acceptance (in_fire)?
3. **ROT_AMT Parameter:** The ROT mode performs a left rotation, but ROT_AMT is defined as a localparam in the RTL (fixed at 4). Is this intentionally fixed, or should it be configurable via a register field?
4. **Counter Invariant Timing:** The invariant COUNT_OUT + DROPPED_COUNT == COUNT_IN - at what point in simulation is this guaranteed to hold? Only when STATUS.BUSY = 0 (pipeline empty)?

Firmware/Software Team - Questions

1. **Counter Monitoring:** The DROPPED_COUNT register tracks discarded packets. Will your drivers poll this register regularly, or do you expect hardware notification (interrupt) when packets are dropped?
2. **Soft Reset Behavior:** In what scenarios do you plan to use the SOFT_RST bit in the CTRL register? Should the pipeline flush completely, and should counters reset to zero?
3. **Counter Overflow:** What is the expected behavior when COUNT_IN, COUNT_OUT, or DROPPED_COUNT overflow their 32-bit range? Standard wrap-around or saturation?

System Architect - Questions

1. **Traffic Patterns:** What is the expected distribution between different modes? Do you anticipate long bursts of single-mode traffic or frequent mode switches? This affects our randomized test constraints.
2. **Backpressure Scenarios:** If the downstream component fails (permanently low out_ready), how should the CPM behave after its internal 2-slot buffer fills? Block input indefinitely or signal error?
3. **Opcode Usage:** Are certain opcodes reserved or more frequently used in the system? This affects our coverage bin weighting strategy.

Board Design Team - Questions

1. **Clock and Reset:** Are there specific power-up sequencing requirements for clk and rst signals that we should model to ensure the DUT initializes correctly in simulation?

Phase 2 - Verification Plan (vPlan) Development

2.1 Scope Definition

In Scope

1. **Data Path Functionality:** Verification of all four transformation modes (PASS, XOR, ADD, ROT). Validating that out_payload matches the expected result based on in_payload and the PARAMS register values.
2. **Register Functionality:** Validating all 8 registers (CTRL, MODE, PARAMS, DROP_CFG, STATUS, COUNT_IN, COUNT_OUT, DROPPED_COUNT). Verifying that register writes correctly configure DUT behavior and reads return accurate values.
3. **Streaming Protocol:** Validation of the CpmStreamIf interface using valid/ready handshake semantics. Testing backpressure handling when out_ready is deasserted.
4. **Packet Drop Mechanism:** Verification that packets with opcodes matching DROP_CFG.DROP_OPCODE are discarded when DROP_CFG.DROP_EN is set, and that DROPPED_COUNT increments correctly.
5. **Reset Compliance:** Verifying the DUT enters a known state after hardware reset (rst), and that SOFT_RST in CTRL register functions correctly to clear counters and pipeline state.
6. **Counter Invariant:** Verification that COUNT_IN == COUNT_OUT + DROPPED_COUNT holds when STATUS.BUSY = 0.

Out of Scope

- Gate-level timing and physical verification
- Register bus protocol edge cases (focus is on functional behavior)
- Performance/throughput measurement
- Multi-beat packets (DUT is single-beat only per specification)

2.2 Verification Strategy

2.2.1 Methodology

- **UVM 1.1d Framework:** Modular, reusable testbench architecture following UVM best practices
- **Dual-Agent Architecture:** Separate CpmPacketAgent for streaming interface and CpmRegAgent for register configuration
- **RAL Integration:** CpmRegModel with CpmRegAdapter for register abstraction, enabling portable test sequences

- **Self-Checking:** CpmScoreboard with CpmRefModel for automatic comparison of expected vs. actual outputs

2.2.2 Stimulus Strategy

Constrained Random Verification:

- CpmPacketTxn: Randomized m_id, m_opcode, m_payload within valid ranges
- Timing variation: Random inter-packet delays and backpressure patterns
- Mode switching: Dynamic configuration changes during traffic

Sequence Hierarchy:

Sequence	File Location	Purpose
CpmTopVirtualSeq	sequences/virtual/	Orchestrates complete 8-step test flow
CpmConfigSeq	sequences/ral/	RAL-based register configuration
CpmBaseTrafficSeq	sequences/packet/	Random packet generation
CpmStressSeq	sequences/packet/	Burst traffic patterns
CpmDropSeq	sequences/packet/	Targeted drop testing
CpmCoverageTrafficSeq	sequences/packet/	Coverage-directed traffic (factory override target)

2.2.3 Checking Strategy

Scoreboard Architecture (CpmScoreboard):

- CpmRefModel computes expected output based on mode captured at input acceptance time (m_mode_at_accept)
- Input packets stored in expected queue with transformation prediction
- Output packets matched by m_id + m_opcode key
- Drop tracking reconciled against DROPPED_COUNT register

Reference Model Transformations (CpmRefModel):

PASS: payload_out = payload_in (unchanged)

XOR: payload_out = payload_in ^ PARAMS.MASK (bitwise XOR)

ADD: payload_out = payload_in + PARAMS.ADD_CONST (addition with wrap)

ROT: payload_out = {payload_in[11:0], payload_in[15:12]} (rotate left by 4)

Protocol Assertions (SVA in CpmStreamIf):

Assertion	Description
p_input_stability	Input signals must remain stable when in_valid=1 and in_ready=0
p_output_stability	Output signals must remain stable when out_valid=1 and out_ready=0
p_bounded_liveness	Output must appear within latency bound (disabled for dropped packets)

2.3 Metrics for Success (Exit Criteria)

2.3.1 Functional Coverage Targets

Coverpoint	Target	Description
cp_mode	100%	All 4 modes exercised
cp_opcode	90%+	All 16 opcodes (0-15)
cp_mode_opcode	80%+	64 combinations (MODE x OPCODE)
cp_drop	Hit	Drop event exercised
cp_stall	Hit	Backpressure stall exercised

2.3.2 Code Coverage Targets

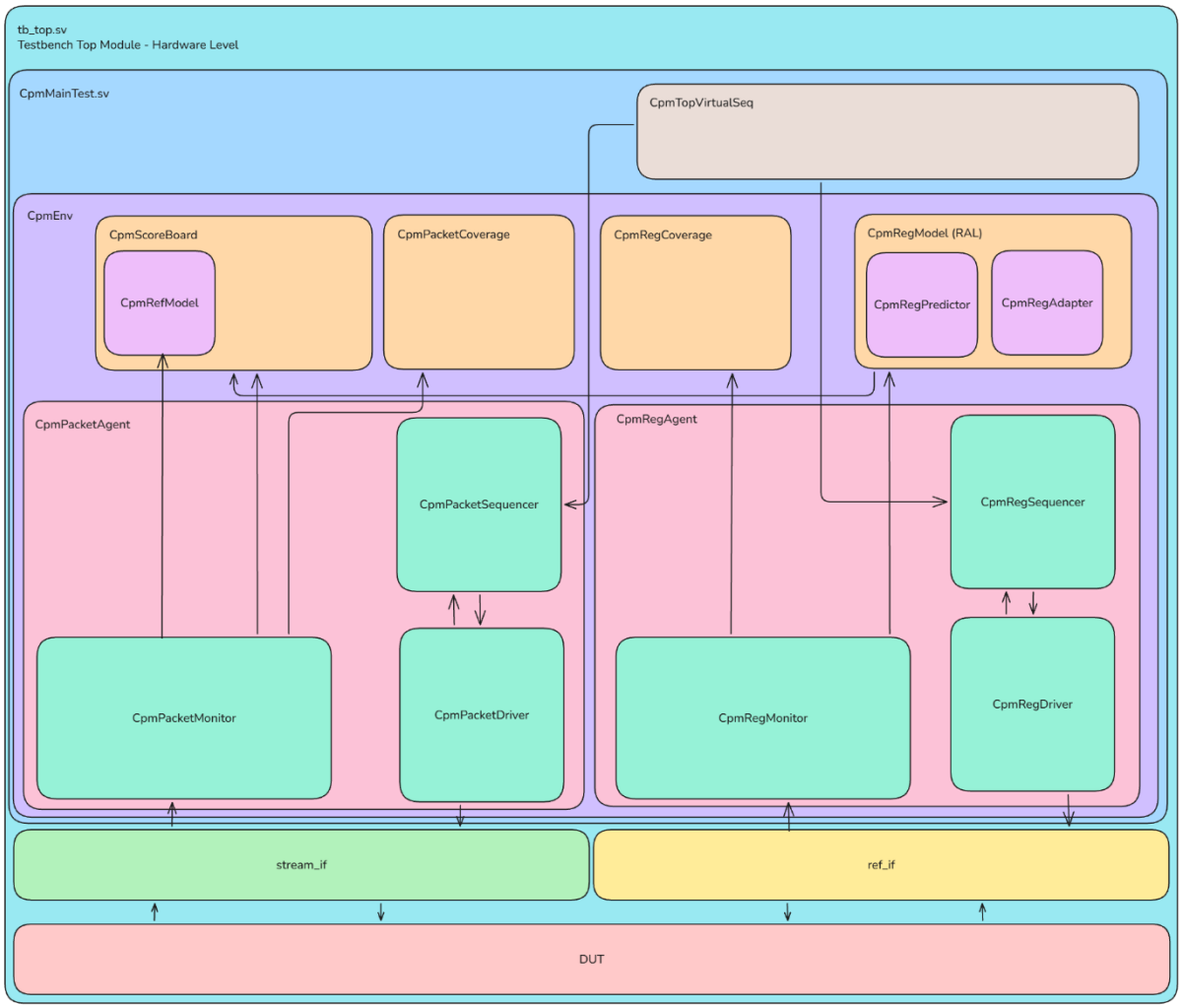
Metric	Target
Statement Coverage	95%+
Branch Coverage	90%+
Expression Coverage	90%+
Condition Coverage	80%+
Toggle Coverage	50%+
Total DUT Coverage	**85%+**

2.3.3 Error-Free Execution

- Zero UVM_ERROR or UVM_FATAL messages
- All SVA assertions pass (0 violations)
- Scoreboard: 0 mismatches between expected and actual
- Counter invariant verified after each test phase

Phase 3 - Testbench Architecture

3.1 Architecture Block Diagram



3.2 Transaction Definitions

CpmPacketTxn (Packet Transaction)

Field	Type	Description
m_id	rand bit [3:0]	Packet identifier for tracking through pipeline
m_opcode	rand bit [3:0]	Opcode for drop mechanism comparison
m_payload	rand bit [15:0]	Data payload for transformation
m_mode_at_accept	cpm_mode_e	Mode captured at input acceptance time
m_mask_at_accept	bit [15:0]	MASK value at input acceptance
m_add_const_at_accept	bit [15:0]	ADD_CONST value at input acceptance
m_timestamp	time	Timestamp for latency measurement
m_expected_payload	bit [15:0]	Expected output (for scoreboard comparison)

CpmRegTxn (Register Transaction)

Field	Type	Description
m_addr	rand bit [7:0]	Register address (0x00-0x1C)
m_wdata	rand bit [31:0]	Write data
m_rdata	bit [31:0]	Read data (response from DUT)
m_write_en	rand bit	1 = Write operation, 0 = Read operation

3.3 RAL Integration

RAL Flow:

1. CpmConfigSeq calls m_reg_model.m_mode.write(status, value)
2. CpmRegAdapter.reg2bus() converts RAL request to CpmRegTxn
3. CpmRegDriver drives signals on CpmRegIf
4. CpmRegMonitor captures transaction from bus
5. uvm_reg_predictor updates RAL mirror via bus2reg()

RAL Register Map (CpmRegModel)

Instance	Address	Access	Fields	Description
m_ctrl	0x00	RW	ENABLE[0], SOFT_RST[1]	Control register
m_mode	0x04	RW	MODE[1:0]	Operation mode
m_params	0x08	RW	MASK[15:0], ADD_CONST[31:16]	Transformation parameters
m_drop_cfg	0x0C	RW	DROP_EN[0], DROP_OPCODE[7:4]	Drop configuration
m_status	0x10	RO	BUSY[0]	Pipeline status
m_count_in	0x14	RO	COUNT[31:0]	Input packet counter
m_count_out	0x18	RO	COUNT[31:0]	Output packet counter
m_dropped_count	0x1C	RO	COUNT[31:0]	Dropped packet counter

3.4 Component Summary

Environment Components

Component	File	Description
CpmEnv	env/CpmEnv.sv	Top environment container
CpmPacketAgent	agent/CpmPacketAgent.sv	Stream interface agent (driver + monitor + sequencer)
CpmRegAgent	agent/CpmRegAgent.sv	Register bus agent (driver + monitor + sequencer)
CpmScoreboard	scoreboard/CpmScoreboard.sv	Packet comparison checker with expected queue
CpmRefModel	scoreboard/CpmRefModel.sv	Golden reference model for transformations
CpmPacketCoverage	coverage/CpmPacketCoverage.sv	Packet functional coverage collector
CpmRegCoverage	coverage/CpmRegCoverage.sv	Register access coverage collector

Tests

Test	File	Description
CpmBaseTest	tests/CpmBaseTest.sv	Common test base class
CpmMainTest	tests/CpmMainTest.sv	Full flow with factory override + callback demo
CpmSmokeTest	tests/CpmSmokeTest.sv	Quick sanity check
CpmRalResetTest	tests/CpmRalResetTest.sv	RAL reset value verification

Mandatory UVM Features Implementation

Feature	Implementation
RAL (Register Abstraction Layer)	CpmRegModel + CpmRegAdapter + uvm_reg_predictor
Virtual Sequence	CpmTopVirtualSeq with 8-step orchestration flow
Factory Override	CpmBaseTrafficSeq → CpmCoverageTrafficSeq in CpmMainTest
Callbacks	CpmBasePacketCb with pre_drive/post_drive hooks
Functional Coverage	cg_packet covergroup in CpmPacketCoverage
SVA Assertions	p_input_stability, p_output_stability, p_bounded_liveness

Phase 4 - Coverage Plan Development

4.1 Packet Coverage (cg_packet in CpmPacketCoverage.sv)

#	Name	Type	Definition
1	cp_mode	Coverpoint	Bins: mode_pass, mode_xor, mode_add, mode_rot
2	cp_opcode	Coverpoint	16 auto-bins for opcodes 0-15
3	cp_mode_opcode	Cross	MODE × OPCODE (64 combinations, target 80%+)
4	cp_drop	Coverpoint	Bins: drop_hit (packet dropped), no_drop
5	cp_stall	Coverpoint	Bins: stall_hit (backpressure observed), no_stall

4.2 Register Coverage (cg_register in CpmRegCoverage.sv)

#	Name	Type	Definition
1	cp_addr	Coverpoint	8 bins for register addresses (0x00, 0x04, 0x08, 0x0C, 0x10, 0x14, 0x18, 0x1C)
2	cp_op	Coverpoint	Bins: read, write
3	cp_addr_op	Cross	Address × Operation (16 combinations)

Coverage Bin Definitions

Mode Coverage (cp_mode): Four explicit bins ensure each transformation mode is exercised at least once during simulation. This is critical for verifying all data path transformations.

Opcode Coverage (cp_opcode): Sixteen auto-generated bins cover the full range of 4-bit opcode values (0x0 through 0xF). This ensures traffic diversity and validates the drop mechanism across all possible opcodes.

Cross Coverage (cp_mode_opcode): The cross product of mode and opcode creates 64 combination bins (4 modes × 16 opcodes). The target of 80%+ ensures comprehensive interaction testing between modes and opcodes.

Drop Coverage (cp_drop): Two bins track whether the drop mechanism was triggered. The drop_hit bin confirms the drop functionality was exercised; no_drop confirms normal packet flow.

Stall Coverage (cp_stall): Two bins track backpressure scenarios. The stall_hit bin confirms the DUT was tested under backpressure conditions where out_ready was deasserted while out_valid was high.

Phase 5 - Detailed Test Case Design

Test 1: Basic Transformation & RAL Consistency

Goal: Verify data transformation and RAL mirror synchronization.

Steps:

1. CpmConfigSeq: Write m_mode = XOR, m_params = {ADD_CONST, MASK=0xAAAA}
2. Send CpmPacketTxn with m_payload = 0x5555
3. Wait for output (1 cycle latency for XOR mode)
4. CpmScoreboard: Verify output = 0xFFFF (0x5555 XOR 0xAAAA)
5. RAL mirror check: m_reg_model.m_mode.get() matches written value

Pass Criteria: Output matches expected, RAL mirror is synchronized.

Test 2: All Modes Sweep

Goal: Verify all 4 transformation modes produce correct results.

Steps:

1. For each mode in {PASS, XOR, ADD, ROT}:
    - Configure via CpmConfigSeq
    - Send 10+ packets via CpmBaseTrafficSeq
    - CpmRefModel predicts expected output
    - CpmScoreboard verifies all packets match

Pass Criteria: All packets in all modes match expected output.

Test 3: Drop Mechanism & Counter Accuracy

Goal: Verify DROP_CFG functionality and DROPPED_COUNT counter.

Steps:

1. Configure m_drop_cfg: DROP_EN=1, DROP_OPCODE=0x8
2. CpmDropSeq: Send packets with m_opcode=0x8 (should drop)
3. Send packets with other opcodes (should pass)
4. Verify dropped packets do not appear at output
5. RAL read m_dropped_count, verify matches actual drop count

Pass Criteria: Dropped packets not at output, counter accurate.

Test 4: Backpressure & Pipeline Recovery

Goal: Verify out_ready backpressure handling and signal stability.

Steps:

1. Configure MODE = ADD via CpmConfigSeq
2. CpmStressSeq: Start continuous packet burst
3. Deassert out_ready for 20+ cycles
4. Verify in_ready drops (buffer full condition)
5. Reassert out_ready
6. Verify no data loss, correct packet sequence maintained
7. SVA p_output_stability passes (no signal changes during stall)

Pass Criteria: No data loss, SVA passes, correct ordering.

Test 5: Counter Invariant Verification

Goal: Verify COUNT_IN == COUNT_OUT + DROPPED_COUNT holds.

Steps:

1. Enable drop for specific opcode
2. Send mixed traffic (some dropped, some passed)
3. Poll m_status.m_busy until 0 (pipeline empty)
4. RAL read: m_count_in, m_count_out, m_dropped_count
5. Verify invariant: COUNT_IN == COUNT_OUT + DROPPED_COUNT

Pass Criteria: Counter invariant holds after pipeline drain.

Test 6: RAL Reset Verification (MANDATORY - CpmRalResetTest)

Goal: Verify all registers have correct reset values per specification.

Steps:

1. Apply hardware reset (rst = 1)
2. Execute uvm_reg_hw_reset_seq on m_reg_model
3. Sequence reads each register and compares against RAL-defined reset values
4. Verify each register matches specification reset value

Pass Criteria: All registers match reset values, no UVM_ERROR.

Test 7: Full Virtual Sequence Flow (MANDATORY - CpmMainTest)

Goal: Demonstrate all mandatory UVM features in a comprehensive test.

Steps (CpmTopVirtualSeq 8-step flow):

1. **Reset Wait:** Wait for rst deassert
2. **Configure:** CpmConfigSeq programs registers via RAL
3. **Traffic:** CpmBaseTrafficSeq sends 200+ packets
4. **Reconfigure:** Cycle all 4 modes with traffic per mode
5. **Stress:** CpmStressSeq burst traffic with backpressure
6. **Drop Test:** CpmDropSeq targeted drop verification
7. **Readback:** RAL read all counters
8. **End Check:** Verify counter invariant

Mandatory Features Demonstrated:

- Factory
  Override: CpmBaseTrafficSeq::type_id::set_type_override(CpmCoverageTrafficSeq::get_type())
- Callback: CpmPacketStatsCb registered on CpmPacketDriver

Pass Criteria: All 8 steps complete, 0 errors, factory override logged, callback stats printed.

Phase 6 - Closure & Reporting Strategy

6.1 Key Performance Indicators (KPIs)

KPI	Target	Measurement
Test Pass Rate	100%	Zero UVM_ERROR or UVM_FATAL in simulation log
Scoreboard Mismatches	0	All expected vs actual comparisons pass
SVA Violations	0	All assertions pass throughout simulation
Functional Coverage	All targets met	Coverage report meets defined targets
Code Coverage	85%+ total	DUT code coverage report

6.2 Verification Sign-off Criteria

Functional Coverage Sign-off

Metric	Target	Status
cp_mode	100%	[ ] Pass
cp_opcode	90%+	[ ] Pass
cp_mode_opcode	80%+	[ ] Pass
cp_drop	Hit	[ ] Pass
cp_stall	Hit	[ ] Pass

Code Coverage Sign-off

Metric	Target	Status
Statement Coverage	95%+	[ ] Pass
Branch Coverage	90%+	[ ] Pass
Total DUT Coverage	85%+	[ ] Pass

Mandatory Features Checklist

Feature	Verification Method	Status
RAL	CpmRalResetTest passes, all registers accessible	[ ] Complete
Virtual Sequence	CpmTopVirtualSeq completes 8-step flow	[ ] Complete
Factory Override	Log confirms override in CpmMainTest	[ ] Complete
Callbacks	CpmBasePacketCb hooks execute, stats printed	[ ] Complete
SVA Assertions	0 violations for all assertions	[ ] Complete
Functional Coverage	Report meets all targets	[ ] Complete
Counter Invariant	Verified via RAL at end of each test	[ ] Complete

6.3 Deliverables

Deliverable	Location	Description
Verification Plan	deliverables/CPM_Verification_Plan.docx	This document
UVM Testbench	verification/	Complete environment source code
Test Suite	tests/	CpmSmokeTest, CpmMainTest, CpmRalResetTest
Coverage Report	coverage/	Functional and code coverage results
Bug Reports	tracking/bug_tracker.csv	All discovered issues and resolutions
Sign-off Document	docs/SIGNOFF.md	Final verification sign-off

6.4 Risk Assessment & Mitigation

Risk	Impact	Mitigation Strategy
Spec ambiguity (ROT_AMT, counter timing)	Medium	Document assumptions in bug_tracker, clarify with design team
DUT bugs blocking progress	High	Track in bug_tracker.csv, prioritize critical issues
Coverage gaps	Medium	Add directed sequences, use coverage-driven traffic
Timing issues in scoreboard	Medium	Capture config at acceptance time, drain pipeline before checks

End of Document
CPM Verification Plan | Version 1.0
Author: Assaf Afriat | Date: 2026-02-01

This verification plan is written prior to implementation and describes the planned approach, targets, and expected outcomes for the CPM verification project.