**Hardware Verification Plan Report: Ethernet Packet Processing Unit**

**1. Executive Summary**

This document outlines the verification strategy for a complex Ethernet packet-processing Design Under Test (DUT). The plan introduces a **hybrid verification methodology** that leverages **Python’s Scapy library** as a protocol-accurate converter between UDP transactions and XGMII-level representations. Scapy is used for both **stimulus generation** and **response validation**, enabling a unified packet interpretation model. The verification environment integrates Scapy into a **SystemVerilog UVM testbench** through a **C-wrapper and DPI-C bridge**, ensuring precise, bidirectional communication between Python and the testbench. This integration allows for **protocol-accurate verification**, reproducible results, and comprehensive validation across a wide spectrum of Ethernet, IP, and UDP test scenarios including compliance checks, error injection, and high-throughput stress tests. The primary goal is to ensure that the DUT correctly handles all forms of packet traffic, maintaining fidelity and performance across various operational conditions.

A diagram of a design under test

AI-generated content may be incorrect.

**2. Verification Architecture and Methodology**

The testbench employs a **hybrid co-simulation architecture**, combining Python’s flexibility with the robustness and modularity of SystemVerilog/UVM. The approach ensures seamless coordination between protocol-level modeling (Python/Scapy) and transaction-level verification (UVM).

**Key Components**

* **XGMII Handler:**  
  The Scapy-based Python module is responsible for generating, decoding, and validating Ethernet frames. It acts as the *single source of truth* for all protocol behaviours, ensuring that both stimulus and reference models share a common packet interpretation.
* **Integration Bridge:**  
  A **C-wrapper** built using the Python C API enables Scapy functions to be invoked from C++. This bridge is connected to the SystemVerilog environment using **DPI-C interfaces**, providing a smooth data exchange pipeline between UVM sequences and Scapy routines.

## **2. Test Environment Overview**

The test environment is built using the **Universal Verification Methodology (UVM)** and encapsulates various verification components such as **agents**, **scoreboard**, and **virtual sequencer**.  
The environment drives and monitors DUT interfaces, collects functional data, and validates correctness.

A diagram of a software company

AI-generated content may be incorrect.

**3. Agents**

Two agents — **XGMII Agent** and **UDP Agent** — are instantiated within the environment.

**XGMII Agent**

Interfaces directly with the DUT’s physical MAC interface (**TxC/D** and **RxC/D**).  
It can operate in **active mode** (driving stimuli) or **passive mode** (monitoring signals).  
The agent transmits and receives XGMII sequence items and sends actual data to the scoreboard.

A diagram of active and active

AI-generated content may be incorrect.

**UDP Agent**

Connects to the streaming interfaces (**s\_udp** and **m\_udp**) of the DUT.   
It models packet-level UDP transactions and also operates in **active** or **passive** mode. The agent provides both expected and actual UDP packets to the scoreboard.

A diagram of a complex activity

AI-generated content may be incorrect.

**4. Virtual Sequencer**

The **virtual sequencer** coordinates activity across multiple agents.  
It provides a unified control mechanism allowing synchronized or parallel stimulus generation between the **XGMII** and **UDP** agents.  
This is crucial for **full-duplex testing** and **concurrent packet exchanges**.

**5. Scoreboard, Predictor, and Evaluator**

The **scoreboard** is responsible for checking data consistency between expected and actual results.

* **Predictor:** Generates expected reference results based on stimulus sequences. It predicts what the DUT output should be for a given input.
* **Evaluator:** Compares the actual outputs (captured from monitors in both agents) with the predicted outputs and reports mismatches.

This mechanism ensures **end-to-end data integrity** between the **XGMII** and **UDP** layers.

A diagram of a process

AI-generated content may be incorrect.

**3. Test Plan & Testcases**

Testcases are prioritized to implement and run, progressing from basic functionality to complex scenarios.

| **Priority** | **Testcase ID** | **Testcase Name** | **Description** | **Verification Objective** |
| --- | --- | --- | --- | --- |
| P0 | TC\_BASE\_LOOPBACK | Basic Environment Bring-Up | Initializes the UVM environment and verifies basic connectivity and agent handshakes. Sends minimal packets to ensure monitor/driver/sequencer connectivity. | Environment sanity, connectivity check, basic data flow. |
| P0 | TC\_ARP\_HANDSHAKE | ARP / Handshake Test | Executes ARP handshake sequences to populate and verify ARP cache entries. Checks for correct request/reply frame exchange and address resolution. | Verify ARP protocol behavior and link bring-up. |
| P0 | TC\_UDP\_PATH | UDP Functional Test | Sends UDP packets through DUT and validates correct transmission, checksum, and metadata preservation. | Validate UDP transmit/receive correctness and integrity. |
| P1 | TC\_UDP\_BACK2BACK | Back-to-Back UDP Packets | Drives consecutive UDP packets with minimal inter-frame gap to stress DUT buffering and flow control. | Verify throughput handling, FIFO behavior, and packet ordering. |
| P1 | TC\_XGMII\_LAYER | XGMII Protocol Compliance | Sends XGMII frames to verify start/end control characters, CRC correctness, and inter-frame gap behavior. | Validate XGMII protocol and MAC/PHY compliance. |
| P1 | TC\_XGMII\_BACK2BACK | Back-to-Back XGMII Frames | Transmits continuous XGMII frames with minimal IFG to test link-level throughput and boundary detection. | Test sustained high-rate operation and frame boundary accuracy. |
| P2 | TC\_UDP\_XGMII\_PARALLEL | Concurrent UDP and XGMII Traffic | Runs UDP and XGMII sequences simultaneously through virtual sequencer to stress inter-agent coordination and full-duplex functionality. | Validate concurrent traffic handling and synchronization. |
| P2 | TC\_VARIABLE\_IP | Variable IP Payload Test | Sends IP/UDP packets with varying payload sizes and header configurations. Includes min/max and odd-length payloads. | Validate length, checksum correctness, and MTU boundary behavior. |
| P3 | TC\_RANDOM\_STRESS | Randomized Stress Test (Planned) | Planned constrained-random test generating diverse traffic across all layers with randomized headers and payloads. | Broad coverage, robustness, and corner-case exploration. |