2025 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE
MUNICH, GERMANY
OCTOBER 14-15, 2025

A Novel Configurable UVM Architecture To Unlock 1.6T Ethernet Verification

Sameh El-Ashry, Cadence Design Systems, Cork, Ireland (*sameha@cadence.com*)

***Abstract*—Verification of high-speed Ethernet controllers poses significant challenges due to the increasing complexity of designs that must support a wide range of configurations and data rates, from 10Mb to 1.6T. This paper presents a novel, fully automated, and configurable UVM-based verification environment tailored for a generic Ethernet controller RTL. The RTL design supports various features sets and configurations, including optional Ultra Ethernet Consortium (UEC) layers, variable SerDes widths, and diverse data rates, all driven by RTL defines generated via a builder tool. To address testbench customization overhead and avoid manual rework for each RTL variant, a Python-based flow was developed to parse RTL defines and dynamically generate a configuration-specific UVM top-level using Jinja2 templates. This produces a testbench-matching Device Under Test (DUT), supporting full-controller and PCS-only modes, with parameterized interface connections and module instantiations. The UVM components—agents, environments, and scoreboards are implemented using a flexible, parameter-driven architecture, enabling rapid adaptation to new RTL builds. Integrated with a Makefile-based flow, the methodology supports one-command generation of both DUT and testbench. Results show a significant reduction in bring-up time, full reuse of verification components, and successful deployment across multiple DUT variants with zero manual edits.**

Keywords—Ethernet; UVM; SystemVerilog; Constrained-Random; Jinja2; Python.

I. INTRODUCTION

As data bandwidth demands grow exponentially in cloud computing, AI accelerators, and high- performance networking, Ethernet technology continues to evolve beyond traditional limits. Modern Ethernet controllers are expected to support a wide spectrum of speeds, ranging from legacy 10Mb up to emerging 1.6T rates, while remaining flexible enough to adapt to diverse application requirements. This introduces significant complexity in both the design and verification of such controllers [1].

An Ethernet controller typically integrates multiple layers, including Application FIFO (AF), UEC layers [2], the Media Access Control (MAC), Physical Coding Sublayer (PCS), Forward Error Correction (FEC) or Reed Solomon Forward Error Correction (RS-FEC), and Physical Medium Attachment (PMA). Depending on the use case, these layers may be enabled or bypassed. Additionally, configurations may vary in the number of Serdes lanes, data bus widths, protocol features, and interface standards. As a result, the RTL is implemented as a generic, highly configurable codebase, with final design instances generated via a builder tool that sets configuration-specific defines.

From a verification perspective, this variability poses a serious challenge: testbenches must either be reworked for every new configuration or built to adapt dynamically without sacrificing maintainability or accuracy. The industry trend is moving toward reusable and parameterized UVM environments, but the top-level testbench remains one of the least automated and most configuration-sensitive components [3].

Building a robust UVM environment is an essential step in verifying any IP, as it enables the development of complex, randomized, and top-level test scenarios that are often portable across projects. In industry practice, it is common for Design Verification (DV) teams to apply a mix of methodologies, including formal verification, UVM-based block-level verification, and even Python or C-based directed tests, depending on the verification scope and IP complexity. Regardless of the strategy, the ultimate objective remains consistent: to implement a comprehensive test plan and close functional and code coverage. Given the resource constraints that most DV teams face, there is a strong push toward automation. Automating repetitive or configuration-dependent infrastructure-such as top-level UVM environments, not only saves significant effort but also frees up engineers to focus on test development, debug, and coverage analysis, all of which are critical to timely and high-quality tapeouts.

This paper addresses this gap by proposing a novel flow that automates the generation of UVM testbench top level based on design-time configuration data, specifically tailored for high-speed Ethernet controllers supporting up to 1.6T.

II. RELATED WORK

The growing complexity of high-speed interfaces such as Ethernet, PCIe, and other advanced interconnects has led to increased demand for scalable and reusable UVM-based verification environments. Traditional UVM testbenches often require manual configurations or duplication to accommodate design variants, especially in parameter rich designs like Ethernet controllers.

Approaches such as those presented in [4], propose using templates to generate consistent verification views and test structures from JSON-based platform descriptions. While effective at a system level, they do not specifically address high-speed Ethernet IP challenges such as lane-to-VIP mapping or full/partial DUT instantiation logic.

The Cadence Ethernet 1.6T Verification IP (VIP) offers a solution for protocol compliance across various data rates. While it provides flexibility in verification, it primarily focuses on protocol adherence and does not address automated testbench generation based on dynamic RTL configurations [5].

A study proposed in [6] discusses methodologies for verifying Ethernet subsystems across different platforms. It emphasizes the importance of adaptable verification strategies in heterogeneous environments. However, it does not delve into the automation of testbench generation driven by RTL parameters.

Tools like the Verification Template Engine (VTE) utilize Jinja2 templates to streamline the creation of UVM components. These tools enhance reusability but often lack integration with real-time RTL metadata, necessitating manual intervention for each configuration [7].

The authors in [8] discuss the use of Portable Stimulus Standard (PSS) in conjunction with UVM to verify a highly configurable, high-speed ethernet communication bridge. The approach emphasizes increasing verification efficiency by avoiding scenario flooding and maintaining tight control over the verification space.

Previous work in [9] presents an approach to reduce testbench implementation efforts by automatically generating UVM environments for SystemC models. The methodology enables assertion-based, coverage-driven, and functional verification, streamlining the verification process for SystemC designs. Importantly, it does not address the challenges of highly configurable RTL designs, where verification architecture must adapt dynamically based on pre-processor `define macros.

Few publications tackle the problem of automatically generating UVM testbenches driven by real-time RTL defines, particularly for complex configurations like full-controller versus PCS-only builds or varying SerDes topologies. To our knowledge, the methodology presented in this paper is among the first to apply a Python and Jinja2 flow for dynamically rendering UVM testbench top levels that reflect RTL defined structure, interfaces, and layer inclusion/exclusion in the context of high-speed Ethernet verification.

III. PROPOSED DYNAMIC UVM ARCHITECTURE GENERATION METHODOLOGY

The verification strategy is centered around building a fully configurable and automated UVM environment that adapts to any RTL configuration generated by the designer's builder tool. The methodology bridges the gap between dynamically generated RTL and a reusable, scalable verification environment by introducing an automated scripting flow that configures the UVM top level based on RTL defines as shown in Figure 1.

A. Design Side

On the design side, the DUT is a configurable Ethernet controller designed to support a range of operational speeds and deployment scenarios. Depending on the implementation, the design hierarchy may include layers such as MAC, PCS, FEC, PMA, and other optional protocol or control layers. Based on product requirements, automated

build tools can generate RTL for various configurations, enabling flexible support for different Ethernet standards and topologies.

These configurations may differ in:

- Inclusion or exclusion of UEC and MAC layers (e.g., PCS-only builds)
- Supported Serdes widths and lane counts.
- Ethernet data rates.
- AF data width and interface structure, such as number of AF ports.
- MII data width for PCS-only configuration and the number of MII ports.

Each configuration is captured through RTL define macros and exported in a dedicated header file.

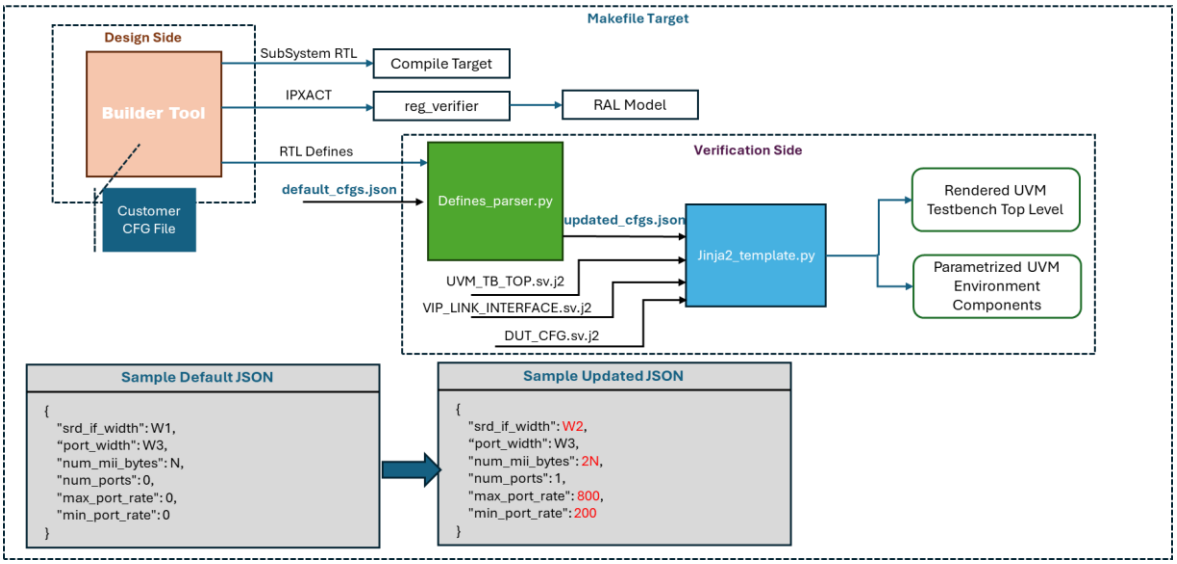

Figure 1: End-to-end automation flow for generating configuration specific UVM environment

B. *Verification Side*

On the verification side, the key innovation lies in an automated flow consisting of the following steps:

1. An initial (default_cfgs.json) file is added as dynamic context data (JSON) and forms a unified, accurate description of the current DUT configurations based on the RTL defines with default values.
2. A python script (Defines_parser.py) parses the RTL defines and generate (updated_cfgs.json) with the updated configurations based on RTL defines. The parser outputs an updated JSON configuration file, which reflects all parameters derived from the DUT build (e.g. number of SerDes lanes, presence of UEC layer, AF width, MII width, etc.) as shown in Figure 2.
3. A Jinja2- based rendering engine uses the adapted JSON file to generate all UVM top-level files, including:
   - DUT instantiation and bind SV assertions modules with multiple instantiations.
   - VIP and interface connections.
   - Clock/reset generators.
   - Lane-to-VIP mapping logic the templates are fully generic, and the rendering process produces a clean, configuration-specific top-level module tailored to the DUT.
4. The UVM components (agents, monitors, drivers, sequences, and environment) are written using parameterized classes and factory overrides. This ensures that the core verification logic remains the same across configurations, minimizing code duplication.
5. As part of the automated UVM infrastructure setup, the Register Abstraction Layer (RAL) model is also generated independently using Reg-Verifier tool. The register description, typically written in IP-XACT or a vendor-specific format, is parsed by Reg-Verifier to produce a complete SystemVerilog UVM RAL model, including register classes, block hierarchies, and access policies. This model is integrated into the

UVM testbench and connected to the DUT's bus interface, allowing for seamless read/write access, register tests, register coverage, and backdoor manipulation during constrained-random sequences. The RAL model generation is typically part of pre-simulation setup flow and complements the top-level environment generation, ensuring full coverage of both datapath and control logic across all DUT configurations.

6. A Makefile target automates the entire process, from RTL configuration parsing to generation of the final UVM top-level, enabling a one-command bring-up for any DUT variant.

This methodology not only simplifies the process of adapting the testbench to new DUT configurations but also decouples the top-level connectivity logic from the environment layer, increasing modularity, and maintainability.

```
Function parse_verilog_defines(file):
        Initialize empty dictionary 'defines'
        For each line in file:
                IF line starts with `define:
                        Extract macro name and optional value
                        If no value provided:
                                Store as: defines[name] = True
                        Else:
                                Convert value to int if possible, otherwise keep as string
                                Store as: defines[name] = Value
        Return defines

Function update_json_file(json_file, defines):
        Load existing JSON config into dictionary 'config'

        For each define in 'defines':
                If key exists in config:
                        Override config[key] = defines[key]

        Generate new file name by appending "_updated.json"
        Save 'config' to this new file

Main:
        Parse command-line arguments: Verilog file, json file
        Defines = parse_verilog_defines (Verilog file)
        Update_json_file (json file, defines)
```

Figure 2: Pseudo-code: updating JSON config based on Verilog defines.

C. Templating UVM Components with Configuration Input

The process of templating the UVM design in this work is not limited to replacing static code fragments with Jinja2 placeholders. Instead, it involves a two-step coupled method:

1. Extracting structural parameters from RTL defines

   A Python-based parser scans the RTL define file generated by the design builder tool. These values represent the fixed configuration of the DUT (e.g., MII port widths). For instance, in a PCS_only configuration, the builder may adjust the MII interface width from a default value to a higher value based on performance requirements. This information is automatically translated into an intermediate JSON format (updated_cfgs.json). Figure 3 illustrated this change, highlighting how the RTL configuration is captured in JSON form.

2. Rendering configuration-specific UVM files

   Instead of hard-coded UVM code, the intermediate JSON (updated_cfgs.json) is then fed into the Jinja2 engine, which renders only those UVM files whose structure depends directly on RTL parameters-primarily the top-level module, interface definitions, and connectivity wrappers. For example, a parsed change in MII interface width is automatically propagated into the generated UVM interface and top-level module, as illustrated in Figure 3.

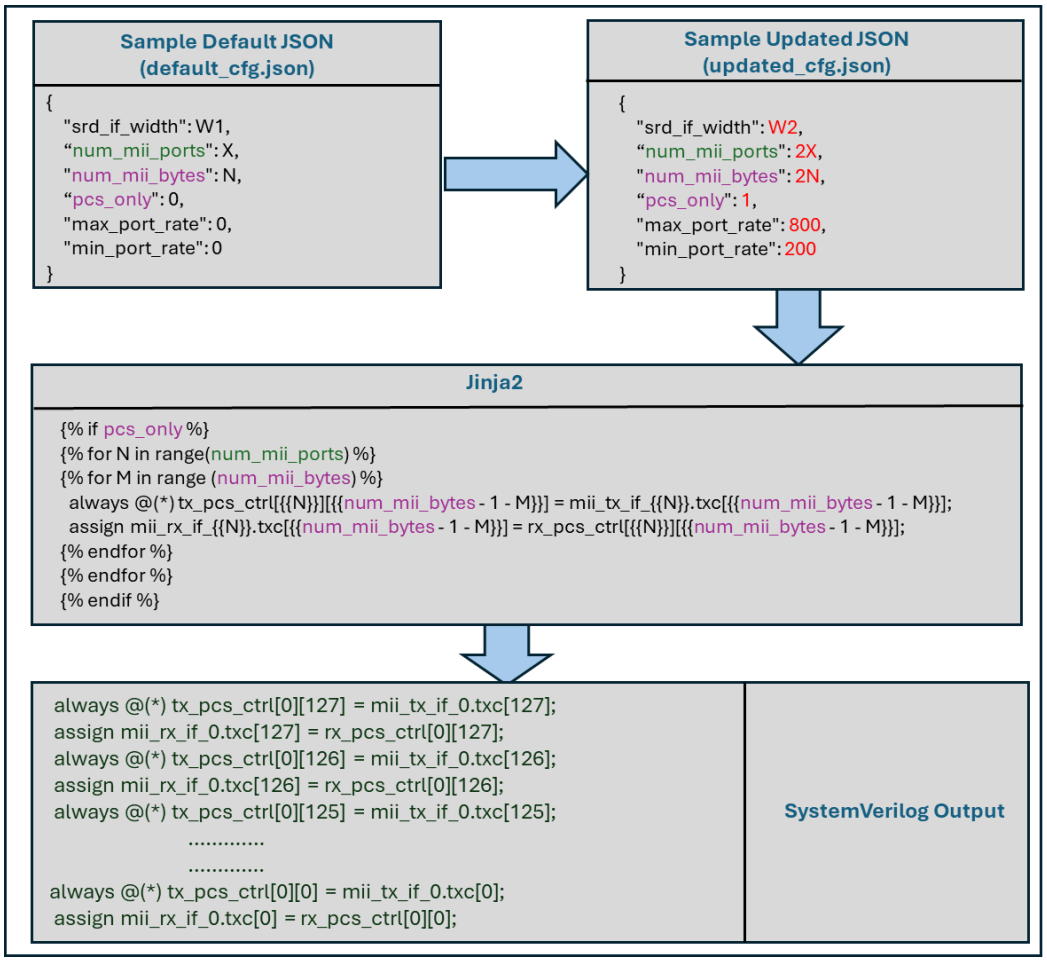

Figure 3: Verification flow of auto-generated UVM snippet showing MII interface width.

Not all files in the verification environment require templating. UVM components such agents, drivers, monitors, and scoreboards are implemented in a fully parameterized style, allowing them to adapt automatically to configuration changes without regeneration. This hybrid approach ensures maximum reuse of UVM code while reserving templates only for files where structural variations are necessary.

D. Classification of DUT Configuration Parameters

A key advantage of the proposed methodology lies in the significant time saving and reliability it offers during UVM top-level generation. The automation flow, implemented using a combination of Python parsing the RTL defines and Jinja2 templating, executes in just a few seconds-making the runtime virtually negligible compared to traditional manual implementation. This efficiency becomes especially valuable when considering the wide range of RTL configurations that impact the UVM top-level structure and connectivity.

In a manual flow, each configuration variant would require a unique top-level UVM testbench. These variants may include different PMA SerDes interface widths, integration with a SerDes VIP model when operating in active mode, passive loopback scenarios where the DUT transmit path is externally looped back to its receive path, PCS-only configurations with varying numbers of MII ports, and full-controller configurations supporting multiple AF ports. Each of these variations necessitates changes in interface connections, instance parameterization, and port mappings—leading to substantial manual rework.

In practice, implementing and debugging the top-level testbench manually for these configurations can take one or two days per variant, due to human errors, syntax mismatches, or misaligned connectivity with VIP interfaces. With at least ten distinct configuration scenarios verified in our flow, this translates to approximately twenty engineering days of effort for top-level setup alone. Through automation, this task has been reduced to a matter of

seconds, with zero manual intervention. This not only accelerates testbench generation but also eliminates human error, improves consistency, and enables rapid adaptation to future RTL changes or configuration additions.

E. Applicability Beyond Ethernet IPs

While the presented methodology is demonstrated using a highly configurable Ethernet controller design, the core automation principles are generic and can be readily applied to other IP types with similar structural variability. The key enabler of this flexibility is decoupling of DUT configuration from testbench architecture, achieved through the use of a defines-driven metadata model and a templated UVM generator powered by Python and Jinja2. This approach is applicable to any IP block where:

- The RTL is built using parameterization or defines (e.g., bus widths, protocol layers, number of ports, etc.).
- The IP can be delivered in multiple configurations, such as standalone or integrated modes.
- Interface connections, instantiations, or bindings at the UVM top level change with configuration.

Although certain IPs implement their own automation strategies, this methodology remains broadly reusable across:

- Custom memory controllers (with varying data widths, burst sizes, or ECC enablement).
- PCIe Controller (e.g., Configurable as Root Port or Endpoint with varying lane width, speed and data path width).
- SoC peripherals (UART, SPI, I2C with parameterized FIFO depths, number of instances, etc.).
- DMA engines (supporting multiple channels, memory-mapped, and streaming modes).
- AI/ML accelerators (with changing compute array sizes, interfaces, or hierarchy depth).

In each case, the same automation process can be reused with minimal template adjustments: parse RTL defines, extract relevant parameters, and render the corresponding UVM top-level structure automatically. This creates a consistent and scalable verification workflow, reduce duplication, and ensures that the verification environment always remains in sync with the RTL configuration, regardless of the IP domain.
By generalizing this solution, teams can build a library of reusable, parameterized UVM templates that adapt to a wide range of design types, significantly reducing the verification ramp-up effort across new IPs.

IV. TOP-LEVEL UVM ARCHITECTURE

The generated UVM top-level architecture for the full Ethernet controller configuration reflects a comprehensive and modular structure tailored for high configurability and coverage closure. The DUT in this step includes six major layers: AF, UEC, MAC, PCS, RS-FEC, and PMA. Each layer is interfaced with corresponding verification components through a mix of UVM Verification Components (UVCs) in active or passive mode, SystemVerilog models, and VIPs, all orchestrated by the generated top-level environment as shown in Figure 4.

A. UVM Environment Structure

On the transmit side, an AF interface is connected to a UVM AF agent, which drives traffic into the DUT, emulating real-world application input behavior. The PMA output interface is first connected to a SerDes SystemVerilog model, which models the serialization behavior before interfacing with the active Ethernet VIP. This setup closely mimics the actual physical connection, ensuring protocol-level validation from logical layers down to serialized bitstreams.

A passive VIP agent is also instantiated and looped back with the active agent VIP to validate correct synchronization and locking behavior. This loopback mechanism is crucial to ensure that the active VIP maintains protocol lock in the presence of both clean and corrupted traffic. Additionally, the architecture supports a fully external loopback mode, where the DUT's TX path is internally looped back to its RX path, bypassing the VIP, which is set to passive monitoring mode. This allows testing of DUT internal loopback logic and path latency without external traffic injection.

Two independent scoreboards are implemented to verify transmit (TX) and receive (RX) paths. The TX scoreboard checks the correctness of packet framing, timing, and integrity as data leaves the DUT. The RX scoreboard validates complete end-to-end correctness, including packet reconstruction after passing through all DUT layers and VIPs or through the internal loopback.

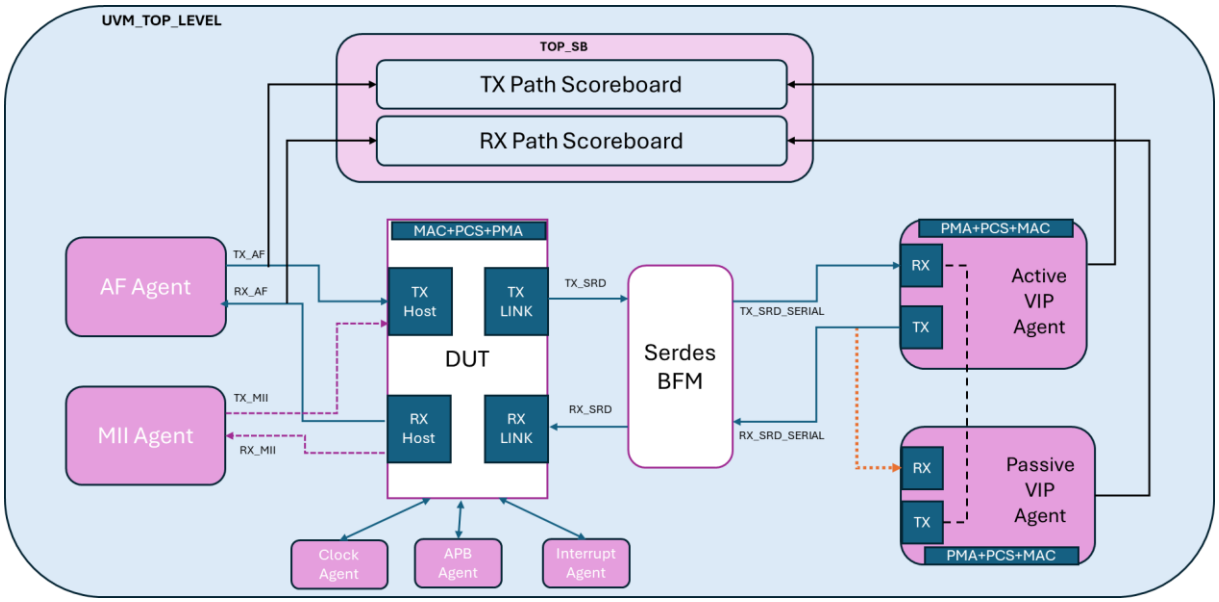

Figure 4: Parameterized UVM top level environment

The environment supports positive and negative test scenarios. Using advanced callback mechanisms in the VIP, the testbench can inject controlled protocol violations such as Start Frame Delimiter (SFD) errors or Alignment Marker (AM) corruptions to evaluate the DUT's resilience and error-handling logic.

Furthermore, the environment includes support of latency measurement, allowing for monitoring of TX latency, RX latency, and round-trip latency, depending on the test scenario. These metrics are essential for performance validation and can be dynamically enabled per test configuration.

The environment is built entirely through the automated template-rendering flow described earlier, which generates the top-level UVM file, interface bindings, and structural connectivity based on the specific DUT configuration. While the top-level structure is fully auto-generated, all UVM components, such as agents, monitors, drivers, and scoreboards are implemented using a highly parameterized architecture. This design allows each component to adapt dynamically to any DUT configuration (e.g., number of SerDes lanes, interface widths, or presence of specific blocks), even though the components themselves are not regenerated per configuration. This hybrid approach ensures scalability and flexibility across a broad set of use cases, without the need to rewrite or duplicate verification components.

In configurations where the DUT is compiled as PCS-only, excluding the AF, MAC, and UEC layers, the same UVM environment is reused with minimal adjustments. Instead of connecting to the AF interface and AF agent, the testbench connects a UVM MII agent to the MII interface of the DUT. This allows packetized Ethernet traffic to be injected directly at the PCS layer. Stimulus is designed with parameterized sequences and virtual sequences; although not auto-generated from templates, adaptation to configuration is achieved through runtime parameters. The majority of the UVM sequences developed for the full controller, including randomized stimulus, latency measurement, and error injection remain applicable and are reused via a virtual sequencer that coordinates between AF-based and MII-based sequences depending on the configuration. This reuse significantly reduces duplication, while maintaining consistent test objectives across DUT variants. The environment's architecture ensures that whether the DUT is a full controller or PCS-only variant, the verification flow remains robust, automated, and efficient.

B. Case Study Highlights

- Case Study 1: Full Controller Configuration.

A verification task involved a full-controller DUT configured for 400G Ethernet, with two ethernet ports, MAC, PCS, and PMA layers, and a wide SerDes interface. The verification goal was to test complex traffic sequences, error conditions (e.g., Start Frame Delimiter corruption), and measure TX/RX latency. Using the automated flow, the UVM top-level was generated with accurate interface wiring between the AF agent, the SerDes model, and

the active/passive VIP instances. All functional sequences were reused without modification. Error injection scenarios were applied via VIP callbacks. Latency measurement features built into the monitor and scoreboard components validated performance. The entire bring-up took minutes, compared to the three to four days typically required for manual setup.

- Case Study 2: PCS-Only Configuration.

This configuration targeted a 100G PCS-only DUT variant with two MII ports, no AF or MAC layer, and featuring a wide PMA interface. The UVM environment automatically switched to connect MII interfaces and instantiated MII agents in place of AF. The same environment supported external loopback, where DUT TX output was routed back to its RX input, and the VIP agent operated in passive mode. Sequences from the full-controller verification were reused through the virtual sequencer, and coverage was collected from both MII ports. This scenario was validated and regression-tested within a single working day, compared to the typical two to three days of manual effort required to adapt a manually built environment.

- Case Study 3: UEC Congestion-based Flow Control (CbFC) Verification.

An advanced testbench scenario involved a full-stack DUT configured for 800G with UEC CbFC enabled. The design included multiple AF ports and protocol layers such as MAC, PCS, and PMA. The UVM flow automatically generated a top-level with the correct AF, PMA, and UEC bindings and instantiated the VIP in active mode. The scenario required stress-testing CbFC behavior under varying congestion and latency conditions. Using VIP callbacks, dynamic pause frame sequences were injected to simulate congestion scenarios. The scoreboard compared UEC pause response latency and verified frame drops and retry behavior. The flexibility of the environment allowed rapid iteration and parameter tuning without any rework to the testbench. This complex scenario, previously estimated to take a full week, was set up and executed within two days.

V. RESULTS

A. Results and Metrics

- As shown in Table I, the manually written UVM top-level file typically spans 800-1000 lines per configuration. The automated solution replaces this with ~150 lines or reusable Jinja2 template code, reducing code duplication by over 85%.
- A single Jinja2 template supports over 10 distinct DUT configurations, compared to 10+ separate hand-coded files in the manual flow.
- Manual top-level implementation resulted in an average of 5 to 7 syntax or connectivity errors per configuration. The automated approach reduced this to zero.
- Bring up time dropped from 2-3 days to less than one hour for new configurations.
- Initial regressions are prone to errors on new configurations due to setup mismatch. With the automated flow, all regressions passed UVM build phase without interface errors.
- The same automated testbench generation framework was reused across rates 10Mb, 100G….to 1.6T projects with the same template structure.
- Instead of tracking multiple static top-level files in version control, only a single set of templates and a configuration file are maintained, reducing review scope by over 70%.

Table I. Results: Manual vs. Automated Flow

Metric	**Manual Flow**	**Proposed Automated Flow**
Top-level testbench creation time (per config)	1–2 days	seconds
Human errors in port/interface connections	High (frequent debug cycles)	Negligible
Effort to switch between full and PCS-only mode	Manual rework	Zero rework
Number of supported DUT configurations	1–2 feasible manually	10+ handled with ease
Reuse of sequences between configurations	Partial	High, via virtual sequencer
Required testbench maintenance	High (per variant)	Centralized and minimal
Parameterized UVM agent/scoreboard support	Manual setup per config	Pre-built and reusable

B. *Challenges of the Proposed Methodology*

While the proposed methodology delivers substantial gains in scalability, automation, and configuration accuracy, a few implementation challenges merit discussion. The flow is optimized for compile-time RTL configurations driven by defines and static JSON context, which may not fully capture dynamic behavior, but all dynamic DUT configurations will be handled using the UVM testbench configuration implemented by the developer. As the number and complexity of supported configurations increase, the Jinja2 template base may require ongoing maintenance, and errors coming from outdated templates or misaligned defines could impact reliability. Additionally, the current flow lacks deep RTL semantic parsing, which might lead to missed wiring errors if a new block is added in RTL but not reflected in the defines file. Generated files might be difficult to debug, especially when failures are due to incorrect template logic. Users may find it harder to trace a UVM connection issue back to the source define or template block.

It is important to note that while the automation flow reduces the per-configuration bring-up time from several days to seconds, there is an up-front investment in developing and validating the templates. In practice, this effort is limited to a one-time activity during environment bring-up and was completed within approximately two to three days for the controller. Once validated, the same templates were reused across all subsequent DUT configurations without modification. Therefore, while the error-free automation results from the fact that templates themselves are tested, this testing effort is orders of magnitude smaller than testing each independent UVM instantiation manually.

VI. CONCLUSION

This paper presents a novel and fully automated methodology for generating configuration specific UVM testbenches tailored to highly parameterized Ethernet controller RTL. By integrating a Python-based parser with Jinja2 templating system, the proposed flow dynamically adapts to RTL build outputs, including defines and contextual metadata, to render a complete UVM top-level environment that matches the DUT configuration. The approach supports a wide range of design variants from full controller to PCS only modes with different Serdes widths, AF widths, and optional protocol layers, all while maintaining a reusable and scalable UVM component architecture. By decoupling top-level connectivity from the environment layer and driving testbench generation directly from design configuration file, this methodology reduces bring-up time, minimizes human error, and maximizes reusability across projects. The proposed flow has been successfully deployed on multiple DUT configurations, demonstrating its robustness, flexibility, and suitability for verifying state-of-the-art 1.6T Ethernet IP.

REFERENCES

[1] "IEEE Standard for Ethernet," in IEEE Std 802.3-2022 (Revision of IEEE Std 802.3-2018) , vol., no., pp.1-7025, 29 July 2022, doi: 10.1109/IEEESTD.2022.9844436.

[2] UEC Credit-Based Flow Control Specification, v0.5, Ultra Ethernet Consortium. [Online]. Available: https://ultraethernet.org/.

[3] El-Ashry, Sameh, et al. "On error injection for NoC platforms: a UVM-based generic verification environment." IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 39.5 (2019): 1137-1150.

[4] Alberto Allara, Rinaldo Franco, " Web Template Mechanisms in SOC Verification," in Proceedings of Design and Verification Conference (DVCON), Munich, Germany, 2015.

[5] Ethernet Accelerated VIP User Guide". 2025. [Online]. Available: https://support.cadence.com/.

[6] Tijana Misic, "SoC Verification of Ethernet Subsystem in Multi-Platform Environments," CDNLive, Silicon Valley, USA, 2023. [Online]. Available: https://www.thevtool.com/wp-content/uploads/2023/09/SoC_Verification_of_Ethernet_Subsystem_in_Multi-Platform_Environments_CDNLive-2023.pdf.

[7] https://github.com/fvutils/vte.

[8] Vintila, Andrei, et al. "Portable Stimulus Driven SystemVerilog/UVM verification environment for the verification of a high-capacity Ethernet communication endpoint." Proceedings of the 2018 DVCON Conference and Exhibition Europe, Munich, Germany. 2018.

[9] Mefenza, Michael, Franck Yonga, and Christophe Bobda. "Automatic uvm environment generation for assertion-based and functional verification of systemc designs." 2014 15th international microprocessor test and verification workshop. IEEE, 2014.