2024 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION INDIA

Navigating Instruction Length Decode: TAP into IP using Formal Verification

Vedprakash Mishra
Intel Corporation
vedprakash.mishra@intel.com

Abstract- **Amidst the increasing complexity of computing systems, the precision and integrity of module designs, particularly the Instruction Length Decode unit (ILD) in modern processors, stand as paramount concerns. The ILD's role in identifying instruction boundaries and enabling accurate field extraction becomes more intricate with innovative Byte-Level Speculative Parallel decoding techniques. Traditional verification methods, inadequate for the dynamic nature of modern ILD designs, underscore the need for a comprehensive approach. This paper addresses this challenge by proposing a methodology, the Trilogy Assurance Paradigm (TAP), designed to rigorously validate ILD functionality. Beyond ILD, TAP extends its applicability to diverse complex intellectual properties. Focused on the CPU pipeline, this exploration delves into the ILD's significance and the intricacies of byte-level speculative decoding. TAP's potency lies in its holistic approach, encompassing top-down control path analysis, bottom-up logic scrutiny, and integration assessments for diverse architectural contexts. This paper presents a comprehensive solution to verify intricate modern ILD designs and extends its methodology's applicability to various complex IPs.**

I. Introduction

As computing systems evolve and embrace greater complexity, ensuring the integrity and accuracy of intricate module designs becomes a focal point in the realm of hardware and software development. A notable instance is the Instruction Length Decode unit (ILD), a critical component embedded within modern processors, responsible for precise instruction boundary identification and field extraction. As advancements introduce innovative techniques like Byte-Level Speculative Parallel decoding, the complexity of ILD designs amplifies, requiring a verification approach that can address their unique complexities. This paper introduces a versatile verification approach, the "Trilogy Assurance Paradigm (TAP)," capable of systematically validating a variety of intricate intellectual properties beyond ILD. The paradigm's potency lies in its encompassing nature, spanning top-down analysis of control paths and data transport, bottom-up scrutiny of data manipulation and calculation logic, and integration evaluation for compatibility with diverse architectural contexts. By adopting this approach, the challenges inherent in verifying complex IPs are effectively addressed. The "Trilogy Assurance Paradigm" transcends the boundaries of specific designs, offering a structured framework that can be applied to diverse IPs, assuring reliability, correctness, and seamlessness in integration. This paper outlines the principles underpinning this paradigm, establishing a strong groundwork for the verification and validation of multifaceted module designs across a broad spectrum of computing systems.

A. Problem Statement:

Traditional verification methods, while effective for simpler components, often struggle to comprehensively address the intricate and dynamic nature of modern ILD designs. The dynamic lengths of instructions, coupled with the speculative decoding process, introduce nuances that can lead to undetected errors if not rigorously verified. The absence of a unified approach further compounds the challenges, making it imperative to develop a systematic and comprehensive methodology capable of validating the functionality and correctness of the ILD with Byte-Level Speculative Parallel decoding.

The problem at hand is to devise a verification methodology that can comprehensively and systematically validate the complex operation of the ILD. This methodology should transcend the limitations of traditional verification approaches, encompassing various dimensions of verification such as control path validation, data manipulation/calculation correctness, and integration compatibility. Additionally, the proposed methodology should not be confined to the specific ILD design under consideration but should be extensible to other complex intellectual properties with similar challenges.

This paper presents a comprehensive exploration of the verification challenges inherent in modern computer architectures, with a specific focus on the ILD and its crucial role within the CPU pipeline. The paper briefly delves into the necessity of instructions, the intricacies of the CPU pipeline, the pivotal role of ILD, the innovation of byte-level speculative decoding, and a novel verification approach "TAP" designed to ensure the integrity of complex ILD designs.

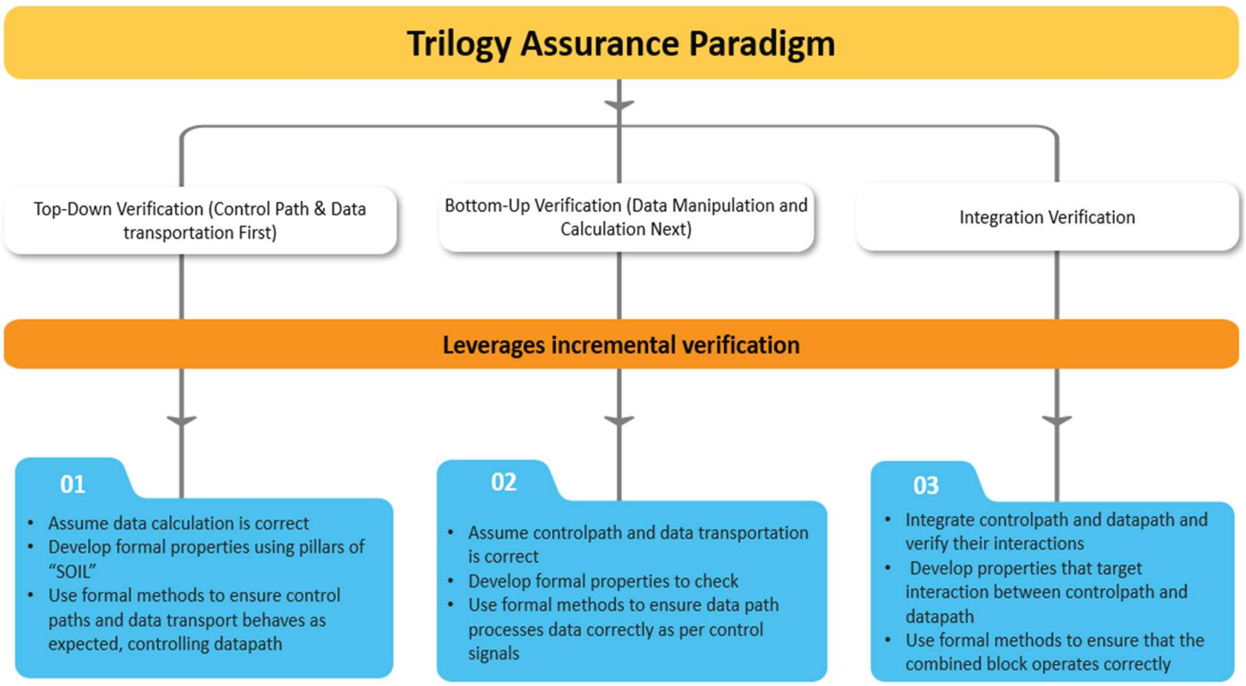

Figure 1: Illustration of TAP methodology

II. DESCRIPTION

A. *Instruction Length Decode: Need, Role & Functioning*[1]

1) *Need for Instructions and the CPU Pipeline:*

In the dynamic landscape of computer systems, instructions serve as the fundamental building blocks that guide the execution of programs. The CPU pipeline, a central aspect of modern processors, enables the concurrent execution of multiple instructions, significantly enhancing performance. However, this concurrency introduces challenges related to hazard detection, data dependencies, and instruction boundary identification.

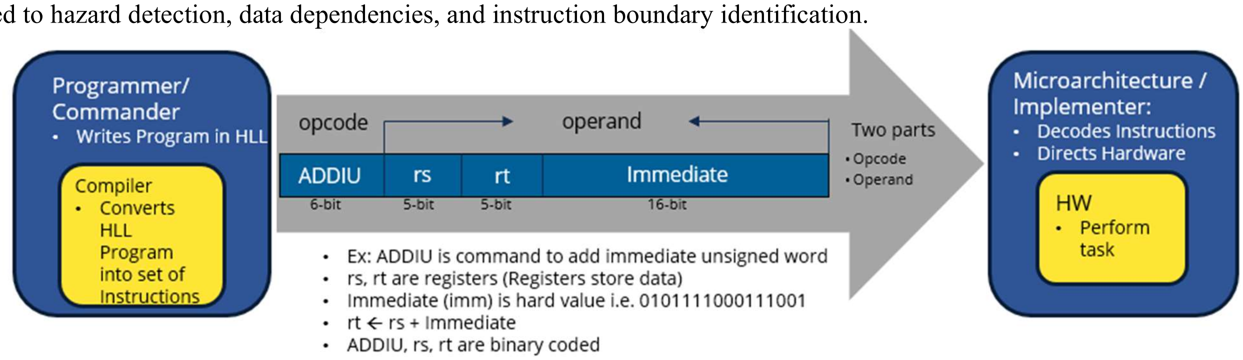

Figure 2: Internals of Instruction

2) Pivotal Role of ILD

At the core of instruction execution lies the ILD unit, responsible for accurately identifying the boundaries of instructions fetched from memory. Proper instruction boundary identification is crucial for the correct execution of subsequent stages in the pipeline. As modern processors adopt variable-length instruction formats, ILD's complexity intensifies, necessitating innovative techniques to ensure accurate instruction decoding. Variable-length instructions allow for a more compact representation of instructions and better utilization of memory.

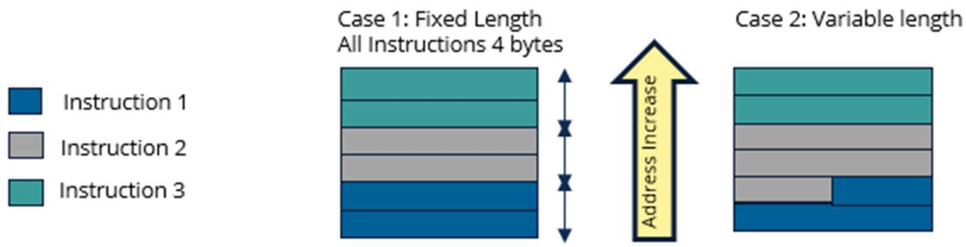

Figure 3: Fetch (Cache-line) level Instruction Packaging

3) Byte-Level Speculative Decoding

One such technique is Byte-Level Speculative Parallel decoding, where each byte within a cache line is considered, a potential instruction starts. This approach aims to leverage parallelism by speculatively decoding multiple instructions simultaneously. While promising, this technique introduces challenges such as identifying valid instruction boundaries, extracting instruction fields accurately, and handling erroneous assumptions like Length Changing Prefix (LCP).

ILD follows a streamlined process for handling requests as shown in Figure 4 & 5:

- Acknowledgement: ILD acknowledges a request based on availability in later stages.
- Prediction & Summation: This stage predicts and sums up something related to the instruction length (possibly related to operand or displacement size).
- Chaining: Potentially involves handling dependencies between requests.

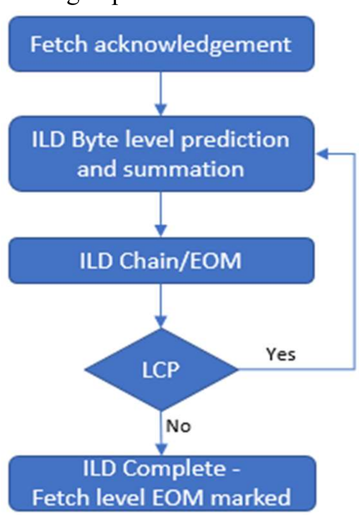

Figure 4: ILD Pipeline

Impact of Length Changing Prefixes (LCP):

- Certain prefixes in the instruction set architecture (ISA) can alter operand or displacement sizes.
- If such a prefix is encountered during prediction, the entire calculation for that request needs to be redone to incorporate the LCP effect.

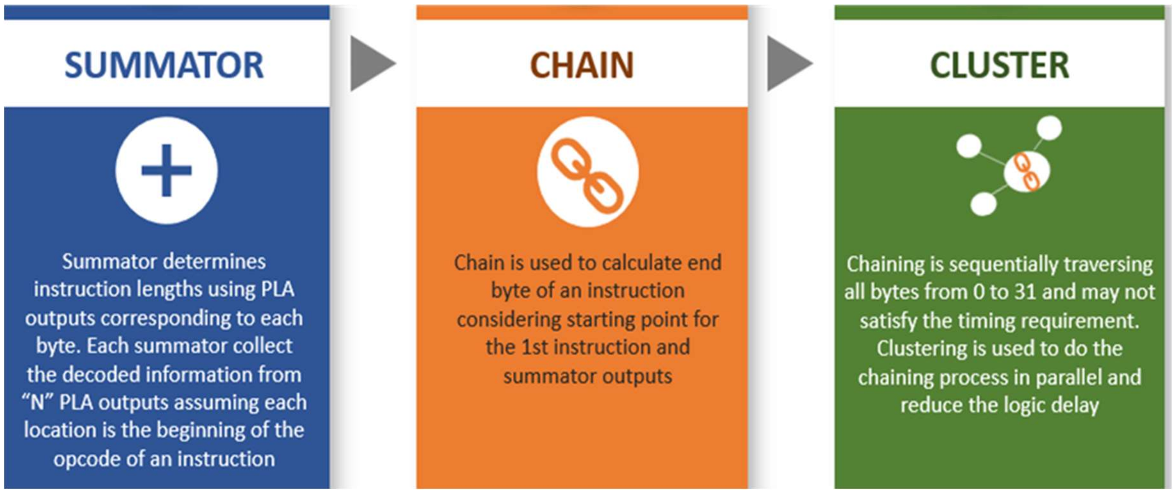

Figure 5: ILD EOM marker mechanism

End-of-Operand (EOM) Markers:

- After potentially redoing calculations and considering LCP, EOM markers are generated for each byte in the fetched instruction packet.
- A value of 1 in the EOM marker indicates the end of an operand within that byte.

Note: This is a simplified explanation. The actual workings of ILD, can be more complex due to factors like:

- Fetch packets being larger than ILD's processing granularity (e.g., 64-byte packets vs. 32-byte processing).
- Dependencies on previous fetch packets for calculations.

III. CASE STUDY: TAP INTO ILD FORMAL VERIFICATION

By combining the three verification phases mentioned in TAP, a wide range of aspects related to the ILD's operation is covered. The top-down approach ensures that the overall control and data flow are correctly established. The bottom-up approach delves into the details of data manipulation and calculation logic. The integration verification phase addresses the ILD's interaction with the pipeline and the generation of correct outputs.

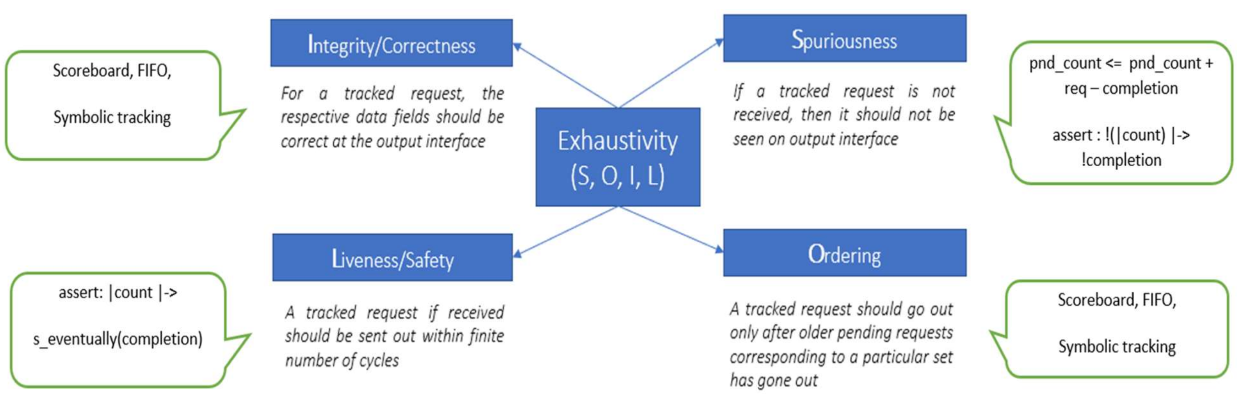

Figure 6: SOIL methodology for control-path verification

1) *Top-Down Verification (Control Path and Data Transportation)[2]:*

- Focuses on the high-level control aspects of the ILD, such as instruction fetching, pipeline stages, and control signals.

- Verifies that the control path is correctly orchestrated, ensuring that instructions are fetched and handled at the appropriate stages using pillars "SOIL"[3]
- Checks that control signals are generated and propagated accurately at each stage to guide the speculative decoding process. Example: summation stage only if acknowledgment received, etc.
- Ensures that data transportation between pipeline stages is correctly managed.

Bug Example: In the context of Figure 4, ILD comprises five stages, starting from acknowledging fetch requests to dispensing ILD complete that define instruction lengths. Notably, in stage 4, there's a check for LCP. When an LCP is detected, stages 2 and 3 are repeated and LCP stall is asserted to stop preceding requests. In a specific bug instance, Fetch Request 2 advanced to stage 1 and received acknowledgment, while Fetch Request 1 reached stage 4 where LCP is checked. Since Request 1 encountered an LCP-related stall, it had to repeat stages 2 and 3. However, due to this LCP stall, Request 1 was erroneously dropped and not held in stage 2 as shown in Figure 7.

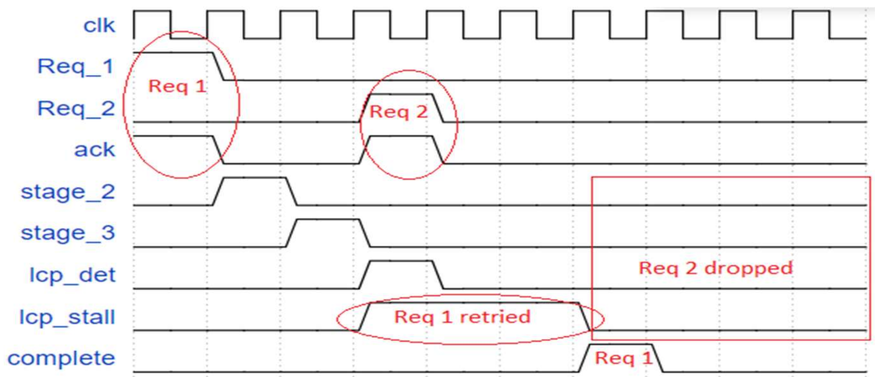

Figure 7: Example of bug caught using Top-down verification.

2) Bottom-Up Verification (Data Manipulation and Calculation Logic):

This section describes the bottom-up approach to Instruction Length Decoding (ILD) as shown in Figure 8. This method focuses on verifying the low-level logic responsible for extracting instruction fields and calculating instruction lengths within a fetch cache line.

Focus on Data Path Correctness:

- Bottom-up ILD assumes the control path is functioning correctly, meaning instruction complete is marked in the last stage and raw bytes are transported properly and EOM markers are available for comparison in final stage by ILD Pipeline.
- It emphasizes verifying the correctness of individual data paths and calculations used to extract instruction information from raw bytes.

Symbolic Instruction Length Calculation:

- While a cache-line can hold multiple instructions, bottom-up ILD avoids calculating lengths for each individual instruction.
- This method leverages the assumption of a functioning control path and EOM markers.
- It symbolically selects one EOM marker within the cache-line and utilizes subsequent bytes to calculate the length of a representative instruction.

By focusing on a symbolic instruction, the bottom-up approach can still detect errors arising from:

- The summator stage responsible for length calculations.
- The chaining stage handling dependencies between instructions.
- The clustering stage potentially grouping instructions.

This symbolic approach avoids the computationally expensive task of calculating individual instruction lengths for every instruction within the cache-line.

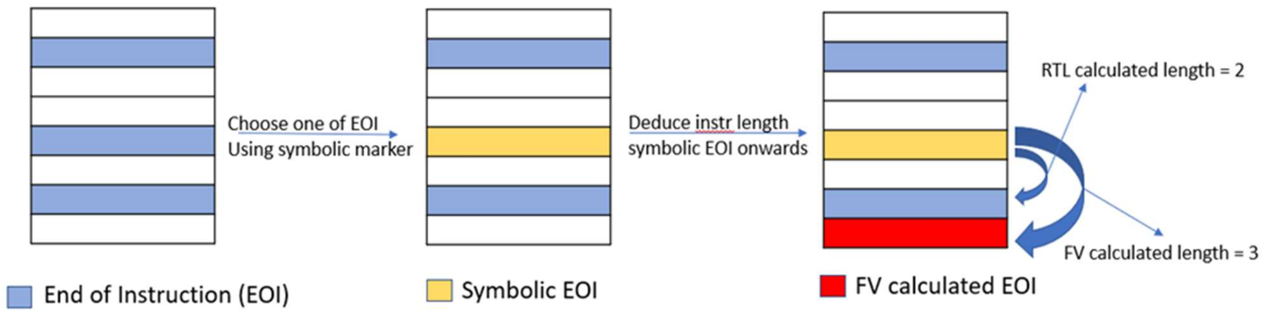

Figure 8: Illustration of symbolic instruction length decoding using symbolic EOM markers.

Bug Example: Using symbolic instruction length decoding, a bug depicted in Figure 9 was found for cross instruction between 2 fetch cache lines:

- ILD request (Req 2) is sequentially following another ILD request (Req 1)
- ILD PLA will use high beat of Req 1 if there is an instruction crossing between the 2 requests.
- ILD should remember the EOM of last instruction and marks the entry offset of an instruction (if instr is incomplete) and it uses the saved entry offset to calculate the instruction once the next beat is received/available

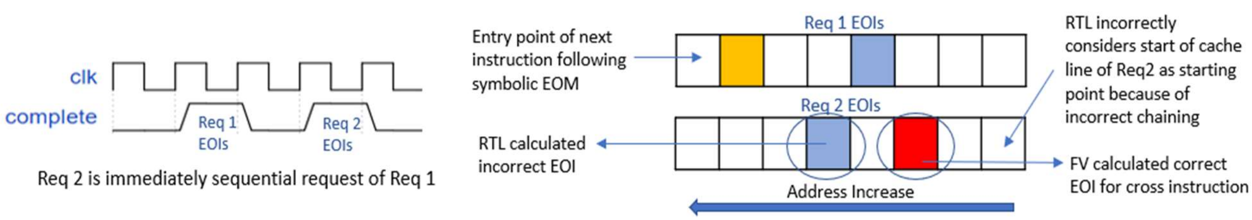

Figure 9: Example of bug caught using Bottom-up verification.

3) Integration Verification (Outputs for Symbolic Fetch Cache Lines):

This section addresses the limitations of bottom-up and top-down verification approaches, which assume correctness of either the control path or data path during verification of the other. Here, we propose a method to ensure seamless integration of the Instruction Length Decoder (ILD) within the larger pipeline context. This verification strategy focuses on a "master property" that validates the integrity of the interface between the control path and data path. It aims to guarantee that no errors leak through this critical interaction point. So overall it aims to achieve following points:

- Addresses the integration of the ILD within the larger pipeline context.
- Verifies that the ILD generates correct outputs for symbolic fetch cache lines.
- Checks that the speculative decoding results are effectively communicated to subsequent pipeline stages.
- Ensures that the ILD's behavior within the pipeline aligns with its intended functionality.
- **Aim**: Utilize symbolic values for inputs, match them against defined patterns, and ensure consistent outputs for those patterns.
- **Methodology**: Symbolic Pattern Matching and Consistency
    - Define symbolic patterns that incoming inputs should match.
    - Use symbolic values to represent input variables.

- Verify that whenever inputs matching the symbolic patterns are received, the corresponding outputs remain consistent.

Bug Example: As shown in figure 10, symbolic patterns/attributes were defined using the attributes (fetch bytes, entry point, etc.), To determine a symbolic request, the attributes of incoming request were compared with the symbolic attributes. In the example, Req1 and Req3 are symbolic requests so the expectation was the EOI markers companied with complete signal of Req1 and Req3 will also match but the bug caused it to mismatch.

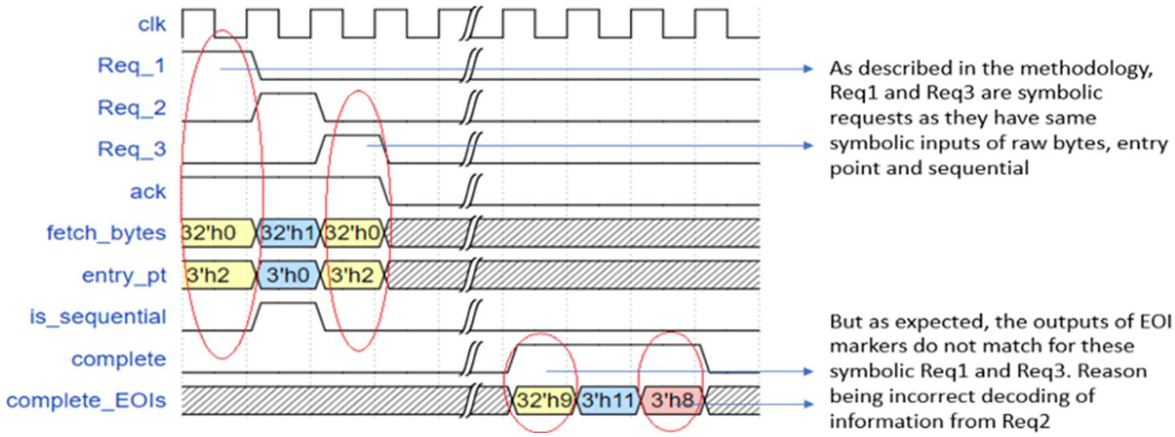

Figure 10: Example of bug causing integration verification

IV. RESULTS AND DISCUSSION

TAP emerges as a solution to the limitation of Formal Verification (FV), where a single bug can disrupt multiple end-to-end properties. TAP divides complex units like the Instruction Length Decode (ILD) into three segments, enabling focused bug identification without being stalled by a single category. Speculative byte-level decoding poses challenges, with RTL focusing on byte-level prediction, while bottom-up verification shifts to accurate instruction-level analysis. TAP's effectiveness lies in its three-pronged approach.

Bottom-up and integration verification streamline summation, chaining, and clustering stage challenges, employing symbolic checks for accuracy. The top-down approach ensures stage integrity, LCP handling and local control Path handling without diving into stage level complexities. Compared to traditional FV, TAP avoids replicating RTL mechanisms, simplifying symbolic handling and data manipulation complexities.

Through TAP's three-pronged approach, total 50 issues have been identified in ILD until now with 40% issues belonging to Top-down, 40% to bottom-up and 10% belonging to integration verification as shown in Figure 8. TAP's influence not only stops to reference modelling and bug hunting but also to coverage. 100% code coverage has been achieved on control-path and data transportation (i.e., top-down) and 100% code coverage on data manipulation/calculation (i.e., bottom-up verification) with waivers.

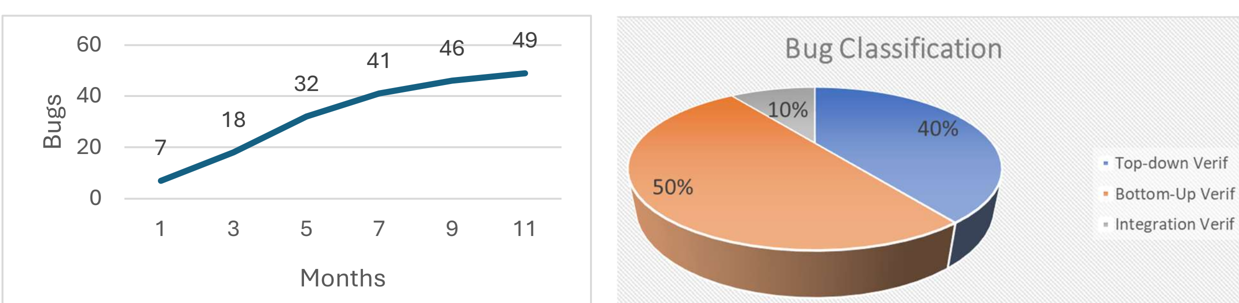

Figure 11: ILD HSD Summary

V. CONCLUSION & FUTURE PLAN

The paper presents the "Trilogy Assurance Paradigm" as an innovative solution to the verification challenges of modern ILD designs. TAP proves its worth by bypassing the limitations of FV and providing a systematic method to verify complex components like the ILD. The division into segments, encompassing bottom-up, integration, and top-down approaches, streamlines verification and identifies distinct categories of bugs. The results signify TAP's potential in transforming how complex designs are verified. By adopting this approach, we anticipate achieving a more accurate and comprehensive assessment of ILD functionality, enabling the identification and resolution of potential errors. Furthermore, this paper serves as a cornerstone for future research in the verification of complex intellectual properties beyond ILD, contributing to the enhancement of system reliability and correctness in diverse architectural contexts.

V. REFERENCES

[1] (2004). Case Study: Synthesis of an Instruction Length Decoder. In: SPARK: A Parallelizing Approach to the High-Level Synthesis of Digital Circuits. Springer, Boston, MA. https://doi.org/10.1007/1-4020-7838-2_11

[2] M, Achutha KiranKumar, Erik Seligman & Tom Schubert, Book on " Formal Verification – An Essential Toolkit for VLSI Design", 2023

[3] OIL check of PCIe with Formal Verification; DVCON 2022; Vedprakash Mishra, Carlston Lim, Zhi Feng Lee, Jian Zhong Wang, Anshul Jain and Achutha KiranKumar V M