Unleashing the Power of Whisper for block-level verification in high performance RISC-V CPU

Chenhui Huang
chuang@tenstorrent.com

Yu Sun
ysun@tenstorrent.com

Joe Rahmeh
jrahmeh@tenstorrent.com

Tenstorrent Inc.
7717 Southwest Parkway, Bldg 3
Austin, TX 78735-9007

***Abstract*-This paper describes a framework that uses Whisper, an open-source RISC-V Instruction Set Simulator (ISS), RISC-V assembly and a UVM testbench to verify the memory subsystem of a high-performance RISC-V CPU. Load and store unit and memory management unit are arguably the most complex microarchitectural blocks in any high-performance CPU. Our methodology enables RTL designers to describe a scenario which is then translated into block level stimulus and drivers with the help of Whisper. This not only reduces the effort for DV engineers for crafting stimulus but also provides more granular control that comes with a block level testbench. Using this flow, we will demonstrate how we enabled a design from scratch and how we were able to take high-level language workloads (e.g., operating systems, benchmarks) and seamlessly run them at the block level. This flow also provided a staged enablement of RISC-V architectural extensions in our design. We are in the process of creating an interface that allows stimulus generation via the AI Large Language Model such as ChatGPT. It will effectively allow complex test plan scenarios to be described in plain text and translated into stimulus that can run on a block level testbench.**

I. Introduction

The most effective approach to RTL bring-up involves using block-level test benches and well-defined, reusable direct and self-checking tests. Block-level test benches offer advantages like faster turnaround times, quicker compilation, easier debugging, and precise control over stimulus and timing due to their compact size. Reused direct tests are preferred as they eliminate the need to debug test-related issues and offer better control over specific features. Using both direct and self-checking tests is favored because during RTL bring-up, the test bench may not be fully matured. Reused direct tests prevent issues related to missing and incorrect constraints, while self-checking tests address the absence of checkers in earlier stages.

In CPU verification, assembly tests are widely used and serve as reusable universal test components. The ideal approach is to begin with core-level bring-up tests at the block level, verifying block-level logic before scaling to core-level tests. However, implementing this strategy can be complex due to the extensive RTL logic required to make assembly tests applicable at the block level. The infrastructure needed for assembly tests can be overwhelming and impractical. This is where Whisper's power comes into play, helping overcome these challenges. This paper outlines our unique approach of leveraging existing Whisper architecture simulation logic to make assembly level usable at block level without additional RTL scaffolding. Our example demonstrates how we used Whisper to fill logic gaps during the bring-up of Load and Store RTL units, including fetch, decode, and execution units.

One fortuitous benefit of our approach is that it enabled design engineers to contribute to the bring-up process. Typically, most designers are not trained in writing UVM sequences or SystemVerilog constraints, even though they are experts in their specific designs. Our Whisper-assisted assembly flow effectively removed the dependency between RTL designers and verification engineers. With basic training in assembly test development, RTL designers were empowered to autonomously create tests for micro-architectural scenarios. This newfound independence allowed them to deliver higher-quality RTLs, significantly expediting the bring-up process.

Whisper is a high-performance open-source RISC-V instruction set simulator (ISS) [1] initially developed by Western Digital for the SweRV Core projects [2, 3]. It allows users to step through each instruction of the target RISC-V code while also giving the capability to view or alter the RISC-V registers and simulated system memory. Therefore, it is serving as a golden ISA reference model for performance analysis, debugging and checking architectural states in core-level simulation.

This paper outlines a framework that integrates Whisper ISS as a central element in the Bus Functional Model (BFM) within the LSU block-level Universal Verification Methodology (UVM) testbench. This approach offers RTL designers a unique method for specifying scenarios, which are then converted into block-level stimulus and drivers

via Whisper. Leveraging various Direct Programming Interfaces (DPIs) and Peek/Poke features from Whisper, the testbench can gather all essential information required for accurately driving the tests. Meanwhile, the framework allows for fine-tuned control over microarchitectural events within the testbench. Utilizing this framework not only minimizes the workload for Design Verification (DV) engineers in generating stimulus but also offers the precise control inherent in block-level testbenches.

Using this technique, we will demonstrate how we enabled a design from scratch and how we were able to take high level language workloads (e.g., operating systems, benchmarks) and seamlessly run them at the block level. This flow also provided a staged enablement of RISCV architectural extensions in our design. We are in the process of creating an interface that allows stimulus generation via the AI large language model (LLM) such as ChatGPT. It will effectively allow complex test plan scenarios to be described in plain text and translated into stimulus that can run on a block level testbench.

The rest of this paper is organized as follows:

Section 2 will demonstrate a typical testbench architecture for LSU and explain why it is inefficient for feature bring-up at the early stage of the project.

Section 3 will describe how Whisper is used as the stimulus component and how it consumes RISC-V assembly tests in the LSU testbench.

Section 4 will describe how Whisper serves the checker which can verify more than data correctness in the block-level testbench.

Section 5 will introduce a collaborative workflow between Whisper and LLM, which will be used for the assembly test generation given different test cases.

Section 6 will be the discussion for the paper.

II. Block-level Testbench for the Load and Store Unit

Load and Store Unit (LSU), together with Memory Management Unit (MMU) are one of the most complex microarchitectural blocks in any high-performance CPU, primarily because of the critical roles they play in a processor's operation and their involvement in various system-level functionalities. Figure 1 shows a typical block-level testbench architecture including the random stimulus and checker. The following components are the key blocks in the testbench.

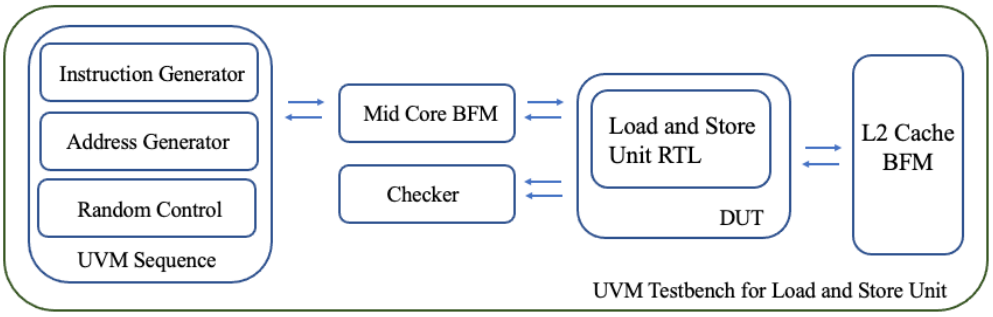

Figure 1. High-level Diagram of block-level testbench architecture for the Load and Store Unit

Instruction Generator: Responsible for generating random load and store instructions with different data size and immediate offset.

Address Generator: Generates various random virtual and physical address space and creates paging translation for the RTL.

Random Control: Complicated scenario control such as pause and flush.

Mid Core Bus Functional Model: Model of the reorder buffer and scheduler behaviors with the instructions to be driven into LSU RTL

Checker: Data checking and hazard checking to keep the fundamental health of the LSU RTL.

L2 Cache Bus Functional Model: Bus functional model for the lower-level cache behavior connecting with LSU RTL.

During the initial phase of RTL bring-up, it is frequently observed that associated testbench components are not entirely developed to full maturity. Employing random stimulus directly during the early stages of RTL bring-up is typically not advisable as it tends to produce an excessive number of illegal stimulus scenarios. This, in turn, demands a substantial degree of fine-tuning to tailor the stimulus to the specific features targeted in the bring-up process, necessitating considerable collaboration between RTL designers and Design Verification (DV) engineers. Moreover, RTL designers, despite their insight knowledge of the microarchitectural design, are often at a disadvantage when it

comes to participating in stimulus generation, since most of them are not trained in writing UVM sequences or SystemVerilog constraints. As a result, it undermines the effectiveness and pace of the RTL feature bring-up in the preliminary stages of the project.

Utilizing direct assembly tests provides a cost-effective and streamlined strategy for the CPU features bring-up. However, their application is limited to situations where the core-level CPU RTL components preceding the Load Store Unit are in a state of complete or partial readiness, enabling the processing of assembly instructions stream issued from the front end. Attaining such a state is typically unfeasible during the preliminary phases of a new CPU project. Additionally, the feature bring-up timeline can become linearly dependent on the developmental milestones of other units, potentially leading to a serialized progression in the overall project.

Conducting assembly tests within the block-level unit testbench is the preferred approach. To accomplish this, we must develop intricate scaffolding and external behavioral models for the LS unit to simulate the functions of other RTL units. This task is both challenging and labor-intensive, as the behavioral model must adhere to the intricate boundaries of the CPU architecture, which includes a multitude of corner cases. The process would be greatly optimized if an extant model were available to supplement the incomplete aspects of the behavioral model.

III. Using Whisper as the Stimulus in the Testbench

Whisper serves as a high-performance RISC-V Instruction Set Simulator (ISS) and can be integrated as the stimulus component in the LS Testbench to facilitate assembly testing at the block level of the LS unit. This integration leverages Whisper's SystemVerilog Direct Programming Interface (DPI) library, providing an interface that allows for the inspection and modification of RISC-V registers and the simulated system's memory from within the testbench environment. Figure 2 highlights a selection of Whisper's DPI capabilities that are particularly useful for conducting detailed, instruction-by-instruction inspection during RISC-V assembly test simulations.

```
import "DPI-C" function void simpleStepWhisper(input int hart, output longint pc, int opcode, int cha
import "DPI-C" function void stepWhisper(input int hart, input longint iTime, input longint instrTag
import "DPI-C" function void peekWhisperRegister(input int hart, byte resource, longint addr, output
import "DPI-C" function void peekWhisperVecRegister(input int hart, input longint addr, output byte
import "DPI-C" function void pokeWhisperRegister(input int hart, byte resource, longint addr, value,
import "DPI-C" function void pokeWhisperVecRegister(input int hart, input longint addr, input byte va
import "DPI-C" function void getWhisperChange(input int hart, output int resource, output longint ad
import "DPI-C" function void doWhisperReset(input int hart, output bit valid);
......
```

Figure 2. Example of DPI functions Whisper ISS support

Figure 3 illustrates the workflow diagram that outlines the role of Whisper in the process of sequence generation derived from an assembly test. This approach deviates from the method of generating instructions and address spaces randomly described in Figure 1. Instead, the instructions are precisely defined within the assembly code itself. The process commences with the compilation of the assembly code into the Executable and Linkable Format (ELF), which subsequently serves as input for Whisper. When Whisper operates in its lock-step mode, it inspects each state and information instruction by instruction. The sequence items generated during this process are not completely by the SystemVerilog constrain random. Rather, they are informed by comprehensive data obtained via Whisper's DPI calls. Once a sequence item is crafted, it is forwarded to the mid-core BFM. Here, it is scheduled and then executed within the LS RTL in an out-of-order sequence.

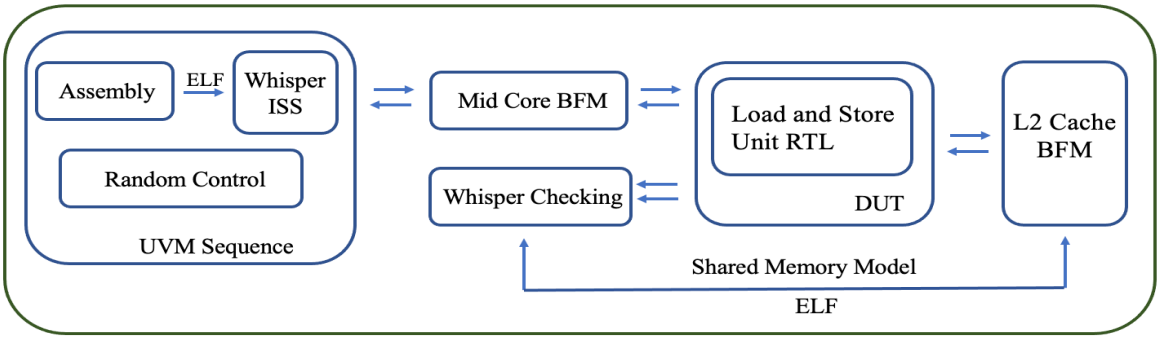

Figure 3. Using Whisper in block-level testbench for the Load and Store Unit

The assembly sequence flow depicted serves as a sophisticated translation interface between the assembly test inputs and the corresponding sequence items that are subsequently driven into the LS RTL. This framework permits RTL designers to construct intricate test scenarios and forge self-checking tests that can be deployed on block-level testbenches. Remarkably, this is achievable without requiring in-depth knowledge of UVM or associated testbench concepts. The adoption of the Whisper-assisted assembly workflow has effectively eliminated the reliance of RTL designers on verification engineers. With foundational training in the development of assembly tests, RTL designers have gained the capability to independently craft tests targeting specific microarchitectural conditions. It has facilitated the generation of higher-caliber RTL designs and has substantially accelerated the feature bring-up phase.

IV. Using Whisper as the Checker in the Testbench

The Whisper ISS functions as a golden reference model for the Instruction Set Architecture of RISC-V, inherently can be used for checking purposes. It provides a straightforward approach to implement an end-to-end checker at the point of instructions retirement or any other architectural boundaries, which requires minimal efforts. When Whisper and L2 BFM share the same memory generated from ELF of RISC-V assembly, we can call the "peek" DPI to get the value of changed architectural states and memory data in the lock-step mode, which can be used to check against the RTL value. This capability allows for consistent and prolonged verification of the LS Unit's integrity, serving as a reliable interim solution until more sophisticated and complex data checking mechanisms are developed and integrated throughout the bring-up phase.

Furthermore, Whisper's capabilities extend beyond simple data integrity verification. It encompasses support for memory ordering and memory consistency model (MCM) checks, specifically verifying the compliance with Preserved Program Order (PPO) rules as dictated by the RISC-V Weak Memory Ordering (RVWMO) [4, 5] model in multi-core environments. This functionality proves advantageous not only during the initial bring-up phase but also continues to add value to the Load Store (LS) unit's operations thereafter.

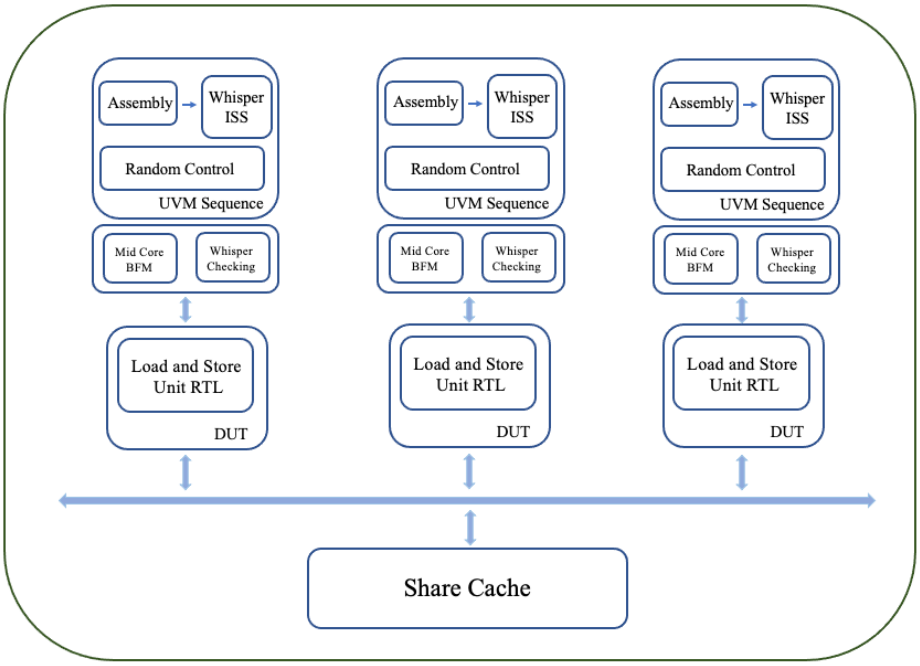

Figure 4. Multiple Load and Store Units with Share Cache

Figure 4 depicts the testbench diagram illustrating the interconnection of multiple LS units that share an L2 Cache. We have reused the testbench environment originally designed for a single LS unit to accommodate this multi-unit configuration. Within this testbench, we launch three instances of RISC-V assembly code and generate corresponding sequences to be fed into the three LS RTL instances. These RTL modules execute diverse load and store operations at different timestamps and interact with separate or overlapping address spaces. The primary objective of constructing this testbench is to validate cache coherency test cases at the block-level. But we leverage Whisper's MCM checker to perform PPO rules verification, leveraging Whisper's MCM DPI calls from the testbench. Whisper's MCM checker scrutinizes load and store values as well as memory ordering events, during instruction retirement or when all bytes

associated with a store or atomic instruction are encompassed by merge-buffer write operations. This approach empowers us to identify and address potential bugs at an earlier stage of the project at the unit-level testing phase.

V. Applying AI Large Language Model for Test Generation with Whisper

Whisper not only serves as golden reference model for checking purposes, but it also has the capability to generate a variety of RISC-V assembly tests tailored to specific testing plans and scenarios. To make it work, the generation process is facilitated by the AI large language models (LLM) such as ChatGPT.

Figure 5 illustrates the collaborative workflow between Whisper and the LLM in creating RISC-V assembly tests. This process is structured into a three-step iterative cycle. Initially, the LLM produces an assembly test, which is then compiled into a binary file. This compilation phase is crucial for detecting any assembly syntax errors. If errors are identified, the specific error line number is relayed back to the LLM, prompting it to generate a corrected version. Subsequently, the compiled binary file is executed using Whisper in standalone mode. This phase is designed to ensure the absence of illegal instructions under the specified architectural scenario. Should any illegal instructions be detected, the instruction number ID or Program Counter (PC) for the offending instruction is communicated back to the LLM. This triggers the generation of a legal alternative. The final stage leverages Whisper's internal coverage functionality. Operating as an Instruction Set Simulator, Whisper generates architectural coverage data for the executed RISC-V test. Once the assembly test has been run and the coverage data obtained, this information is fed back to the AI LLM. This step assesses whether any functionalities are missing in relation to the targeted test plan and scenario. The process iterates until all three steps are satisfactorily completed to generate a targeting RISC-V assembly test.

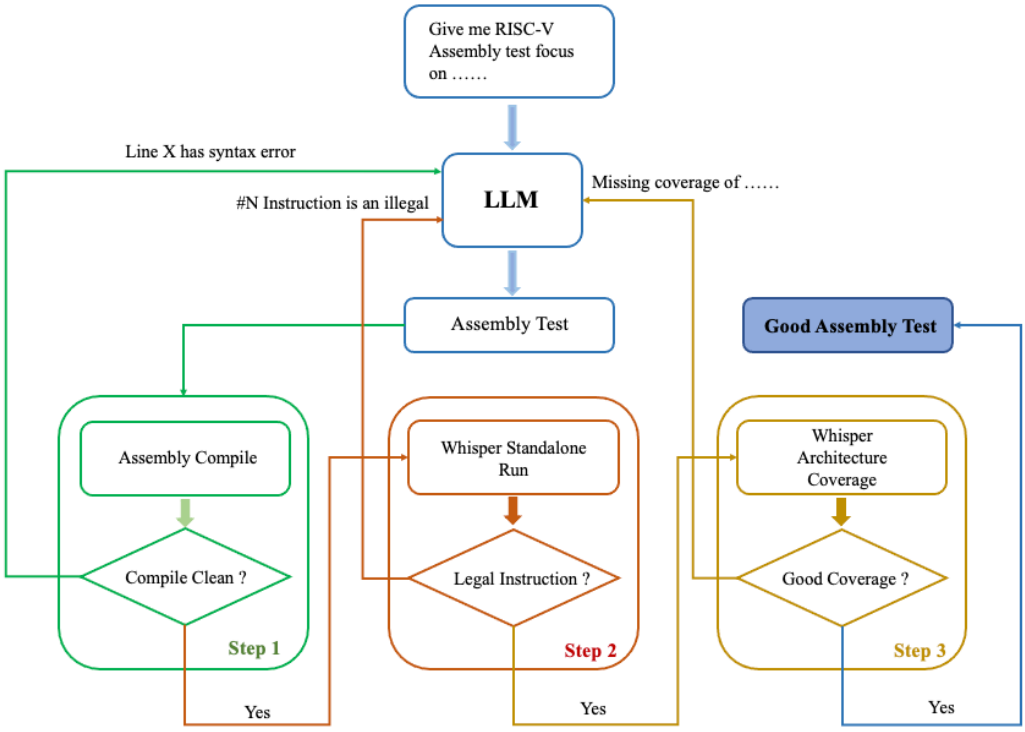

Figure 5. Step flow diagram of using LLM for Assembly test generation.

The AI LLM, such as ChatGPT, comes with a variety of Python APIs which make this workflow viable. We have customized this flow to facilitate the generation of vector-focused tests during the initial bring-up of the Vector Extension (V-Ext) for our RISC-V core. ChatGPT's LLM has successfully produced substantial RISC-V assembly tests, specifically for load and store vector instructions. These tests are integral to the introduction of vector capabilities in the LS unit. However, it's noteworthy that access to OpenAI's APIs is on a paid basis, with costs accruing according to usage and the number of tokens per API request. A more sustainable approach would be to adopt a local LLM for RISC-V assembly test generation, thus bypassing ongoing API costs. Exploring and possibly adapting open-source LLMs for our specific use case is an avenue we are considering for future development.

VI. Discussion

The integration of Whisper into the RISC-V block-level verification process represents a substantial advancement in CPU testing methodologies, enhancing the verification workflow, empowering RTL designers with new capabilities, and paving the way for future breakthroughs, particularly within AI-assisted testing and development frameworks.

Whisper's primary role as a stimulus component within the block-level testbench is transformative. It acts as the intermediary, translating scenario-level assembly tests into transaction-level packages that are driven into the RTL. This Whisper-assisted assembly flow allows designers to independently generate tests for specific micro-architectural scenarios, thereby expediting the RTL bring-up phase. Designers are equipped to proactively identify and rectify potential design issues promptly. Whisper's facilitation in converting high-level scenarios into block-level stimulus and drivers reduces the workload on DV engineers and imparts a more refined degree of control over the testing procedures.

Furthermore, Whisper functions as the definitive golden ISA reference model for RISC-V, inherently operating as a checker. It is almost effortless to develop an end-to-end checker, which is particularly important during the early stages of RTL bring-up, where the testbench may not be fully matured. Beyond ensuring data integrity, the Whisper-enabled assembly testbench offers scalability, readily extendable to multi-core settings. This adaptability is advantageous even beyond the initial bring-up phase, leveraging MCM checkers to verify PPO rules under the RVWMO for RISC-V cores, thereby advancing the verification process to earlier project stages.

Finally, the collaborative workflow between Whisper and LLM is a pioneering approach, where tests are systematically generated, devised, compiled, and honed according to specific test cases and scenarios. Nonetheless, the efficacy of the generated tests is contingent upon the LLM's training for those scenarios. A potential challenge is that the iterative process may become protracted, failing to yield the anticipated outcomes. Moreover, the cost of accessing well-trained LLMs via APIs, such as ChatGPT, is a consideration, leading to the exploration and potential adoption of open-source LLMs or the maintenance of in-house LLMs as more pragmatic and economical alternatives.

Reference

[1] Tenstorrent. (2023). *Whisper*. Available at: https://github.com/tenstorrent/whisper

[2] Marena, Ted. "RISC-V: high performance embedded SweRV™ core microarchitecture, performance and CHIPS Alliance." Western Digital Corporation (2019).

[3] Lei, Z., Cai, F., Zhou, J., & Guo, Z. (2022, December). A Floating-Point Unit Architecture Based on SweRV EH1 Core. In 2022 IEEE 16th International Conference on Anti-counterfeiting, Security, and Identification (ASID) (pp. 1-5). IEEE.

[4] Xu, Y., Yu, Z., Tang, D., Chen, G., Chen, L., Gou, L., Jin, Y., Li, Q., Li, X., Li, Z. and Lin, J., 2022, October. Towards developing high performance RISC-V processors using agile methodology. In 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO) (pp. 1178-1199). IEEE.

[5] RISC-V International. (2023). *RISC-V Instruction Set Manual, Volume I: User-Level ISA, Version 2.2*. Section: Preservation of Program Order (PPO) rules for RVWMO