Toffee: an Efficient and Flexible Python Testing Framework for Chip Verification

April 21, 2025

Abstract

Functional verification in IC/ASIC projects typically accounts for up to 60% of the entire development efforts, consuming a lot of resources. Furthermore, traditional verification methodologies are often disconnected from the modern software ecosystem, making it difficult to utilize efficient testing tools and developers in software. This happens mainly for two reasons: low coding efficiency and poor integration for software tools and developers. In this paper, we propose a Python testing framework called TOFFEE. *It introduces three key innovations: asynchronous function modeling, which models hardware designs as software APIs using two types of async functions; a hook-enabled reference model, which employ hooks for efficient communication; and a test-driven execution strategy, allowing test cases to be managed by software testing frameworks. Experiments show that most users can build their first verification environment within 1.8 to 9.8 hours and integrate well with software testing tools. Additionally,* TOFFEE *can reduce the lines of code in the verification environment by up to 86.31%, and cut execution time by up to 89.69%.*

Introduction

Hardware verification plays a crucial role in the chip development process, which can effectively avoid the huge loss caused by the failure of the chip. Research shows that verification accounts for 50 to 60 percent of total development time in IC/ASIC projects. The verification workforce has grown to nearly match the number of design engineers, with ratios reaching as high as 5:1 in certain market segments[1]. This significant resource allocation has driven the industry to persistently seek more efficient verification methodologies.

Verification frameworks have evolved from VMM and OVM to the current UVM[2]. These frameworks support verification engineers by defining standard verification flows and offering reusable code libraries. They are typically built on hardware verification languages (HVLs), such as SystemVerilog, and follow early-stage programming and testing patterns. In contrast, the software development field has advanced rapidly in recent years. software testing tools have evolved a series of fascinating mechanisms to improve testing efficiency and flexibility. For example, pytest[3] can automatically detect and run test functions, reducing the need for manual testing. This aligns with agile development practices and helps increase development speed through quick iterations. Moreover, a survey[4] by StackOverflow shows that hardware developers make up only 0.3% of the total developer population, much less than software developers. The fast pace of new features and the large developer base both highlight the greater activity and efficiency in the software field.

Therefore, in this paper, we ask the question: *can we leverage modern programming practices and developer resources in software field to improve the efficiency of chip verification?* In fact, there have already been efforts in this direction. Cocotb[5] uses VPI to enable simulation of the DUT (design under test) in Python. Pyuvm[6] ports UVM to Python, giving Python a verification framework to support hardware verification. However, pyuvm only translates UVM into a different language, still requiring engineers to follow the traditional paradigm. This leads to two main issues: *1) Low coding efficiency.* UVM environment setup remains inherently complex. Even basic testbenches often need more than ten components, making it difficult to support rapid iteration and continuous delivery common in software development. *2) Poor integration with software tools and developers.* Current approaches use Python but still follow traditional hardware verification logic. They are not integrated with the software testing ecosystem and are difficult for software developers to adopt.

To address these issues, we believe it is necessary to design a new verification framework suited for the software domain. This framework should support fast iteration by making the test environment easier to set up, and it should follow software development conventions to lower the learning curve for software developers. However, reaching this goal is challenging. We identify three core parts that any verification environment must support: interaction, checking, and testing. Each of these parts comes with a specific challenge:

- **C1** There is no efficient way in software field to handle interaction between test cases and the hardware design. *(Interaction mechanism)*

- **C2** Existing methods for reference model communication are inflexible and involve complex interfaces. *(Checking mechanism)*
- **C3** Hardware test cases cannot be managed by software testing frameworks. *(Testing mechanism)*

To address these challenges, we introduce TOFFEE, a software testing framework designed for hardware verification. TOFFEE's verification environment is organized into three main components: the DUT proxy, the model hub, and the test scenario layer, each supporting one of the core functions of verification. In the DUT proxy, we propose asynchronous function modeling, which models hardware operations as simple software APIs. This approach makes it easier for test code to interact with the DUT (*C1 addressed*). In the model hub, we introduce the hook-enabled reference model, which offers a clean and flexible way to connect reference models with the verification environment (*C2 addressed*). In the test scenario layer, we present a test-driven execution strategy, allowing hardware test cases to be fully managed by software testing frameworks (*C3 addressed*). Results show that most users are able to set up their first verification environment within 1.8 to 9.8 hours and give TOFFEE positive feedback for its integration with software tools. TOFFEE reduces the complexity of building verification environments, cutting lines of code by up to 86.31%. Compared to similar frameworks, it also reduces execution time by up to 89.69%. TOFFEE provides an effective solution to the problems of complex setup and limited support for software testing tools and developers.

Overall, our work makes the following key contributions:

- We introduce asynchronous function modeling, which abstracts complex hardware operations using two types of asynchronous functions. This approach allows TOFFEE to bridge test code and the DUT using a software-style method.
- We present the hook-enabled reference model, which enables simple and effective communication with reference models, avoiding redundant interface connections.
- We design the test-driven execution strategy, which replaces the traditional pull model with a push model. This shift gives control to the test cases, aligns better with software testing practices, and allows full integration with software testing frameworks.
- We prototyped the TOFFEE framework using Python, along with picker and pytest. With this implementation, TOFFEE is capable of completing full verification tasks.

Discussion

We began by identifying two major issues in existing approaches: low coding efficiency and poor compatibility with software testing tools and developers. Has TOFFEE addressed these problems effectively? For the first issue, our evaluation shows that the LOC across four case studies directly reflect TOFFEE's improved coding efficiency. Python naturally offers much higher productivity than SystemVerilog, which is also evident in pyuvm having significantly fewer lines than UVM using the same architecture. TOFFEE introduces a new verification environment structure that further enhances this efficiency. In the cases we tested, TOFFEE reduced LOC by up to 86.31%. Notably, TOFFEE showed excellent performance in small environments and maintained strong efficiency in larger designs, demonstrating both flexibility and scalability.

For the second issue, feedback from eight participants with software backgrounds showed that TOFFEE integrates well with software ecosystems. Most users were able to set up their first verification environment within 1.8 to 9.8 hours, and gave positive feedback on its simplicity and software integration. Compared to previous work, our ability to combine hardware verification with tools like pytest and hypothesis marks a major step forward. In terms of execution, TOFFEE performs on par with UVM and even outperforms it in some complex scenarios, making it more practical. These results show that TOFFEE effectively addresses both the efficiency and usability challenges found in traditional frameworks.

References

[1] Siemens Verification Horizons Blog. *Part 8: The 2022 Wilson Research Group Functional Verification Study.* Accessed: 2024-11-20. 2024. URL: `https://blogs.%20sw.siemens.%20com/%20verificationhorizons%20/2022/12/12/part%20-8-the-2022-wilson%20-research-group%20-functional-%20verification%20-study`.

[2] Accellera Systems Initiative. *Universal Verification Methodology (UVM) 1.1 User's Guide.* Accessed: 2024-11-20. 2011. URL: `https://www.accellera.org/images/downloads/standards/uvm/%20uvm%5C_users%5C_guide%5C_1.1.pdf`.

[3] pytest-dev. *pytest.* Accessed: 2024-11-20. 2024. URL: `https://github.com/pytest-dev/pytest`.

[4] Stack Overflow. *2024 Developer Survey.* Accessed: 2025-4-20. 2024. URL: `https://survey.stackoverflow.co/2024/developer-profile/`.

[5] cocotb Developers. *cocotb.* 2024. URL: `https://github.com/cocotb/cocotb`.

[6] pyuvm Developers. *pyuvm.* Accessed: 2024-11-20. 2024. URL: `https://github.com/pyuvm/pyuvm`.