УДК 621.38

Свинцов А.А.

студент магистратуры

НИУ МИЭТ

ИССЛЕДОВАНИЕ ЭФФЕКТИВНОСТИ ВЕРИФИКАЦИИ С ИСПОЛЬЗОВАНИЕМ PYUVM И SYSTEMVERILOG-UVM.

Аннотация: Python, как мультипарадигмальный язык, известный своей простотой интеграции с другими языками, в последнее время завоевал значительное внимание среди инженеров по верификации. Среда верификации на базе Python использует такие открытые библиотеки, как PyUVM, обеспечивающая реализацию UVM 1.2 на базе Python, и PyVSC, способствующая рандомизации с ограничениями и функциональному покрытию. Целью данной работы является оценка эффективности верификации цифровых дизайнов с помощью PyUVM и сравнение возможностей и показателей производительности с устоявшейся методологией SystemVerilog-UVM.

Ключевые слова: верификация, uvm, pyuvm, python, systemverilog.

Svintsov A.A.

master's student

MIET

STUDY OF VERIFICATION EFFICIENCY USING PYUVM AND SYSTEMVERILOG-UVM.

Abstract: Python, as a multi-paradigm language known for its ease of integration with other languages, has recently gained considerable attention among verification engineers. The Python-based verification environment uses

open libraries such as PyUVM, which provides a Python-based implementation of UVM 1.2, and PyVSC, which promotes limited randomization and functional coverage. The purpose of this work is to evaluate the effectiveness of digital design verification using PyUVM and compare capabilities and performance indicators with the established SystemVerilog-UVM methodology.

Keywords: verification, uvm, pyuvm, python, systemverilog.

Введение

По мере непрерывного роста сложности проектов систем на кристалле верификация становится всё более сложной задачей. В результате время, необходимое для верификации, значительно увеличивается. Кроме того, возникает необходимость в более продуктивной и эффективной работе при ограниченном количестве сотрудников.

Различия Python и SystemVerilog

SystemVerilog имеет высокий порог вхождения по сравнению с другими языками программирования. В связи с этим UVM, основанный на SystemVerilog, становится более сложным в использовании. С другой стороны, Python обладает низким порогом вхождения, является одним из самых популярных языков и легко интегрируется с такими библиотеками, как Numpy, Pandas и др.

Таблица 1 – Сравнение SystemVerilog и Python в верификации

Характеристика	**SystemVerilog**	**Python**
Декларация типов данных	Статическая	Динамическая
Поддерживаемые типы логики	0, 1, X, Z	X, Z, U, W
Параметризация и размер переменной	Требуется	Не требуется
Стиль контроля потока	begin, end	Правильный отступ
Функции	Не объекты	Вызываемые

		объекты
Исключения	Не поддерживаются	Поддерживаются
Библиотеки	-	Поддерживаются
Интерпретируемый	Нет	Да
Иерархия дизайна	включает верхний testbench	Не включает верхний testbench

Замечание по таблице для Python:

- Позволяет не объявлять переменные и выполнять операции над ними;
- Поддерживает сложные структуры данных (кортежи и словари);
- Исключения обрабатываются с использованием блоков try, except;
- Легче создать эталонные модели для дизайна, благодаря поддержке множества библиотек;

Время выполнения тестовых сценариев PyUVM медленнее, чем у тестовых сценариев SystemVerilog-UVM. Такое различие во времени выполнения тестовых сценариев связано с их разными подходами. В SystemVerilog для установления связи с симулятором используются директивы и команды моделирования, что приводит к тесной интеграции, которая улучшает выполнение и сокращает время выполнения. В отличие от них, Python взаимодействуют с симулятором с помощью VPI/VHPI, которые менее тесно интегрированы. Эти издержки становятся более значительными по мере увеличения количества транзакций (см. рисунок 1).

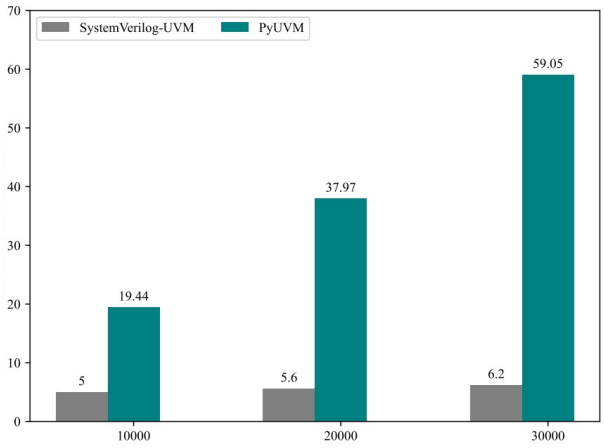

Рисунок 1 – Время тестирования от количества транзакций

С точки зрения использования оперативной памяти, Python потребляет на 30-35% больше памяти. В симуляторе Questa Sim при длительных симуляциях наблюдаются утечки памяти в Python, где потребление памяти могло быть в несколько раз выше, чем в SystemVerilog (см. рисунок 2).

Рисунок 2 – Потребление памяти в разных симуляторах

Покрытие в Python оказалось меньше на 1,6%, разница начала проявляется после 400 операций. Это показывает, что по мере увеличения количества операций, PyVSC чаще выдавал случайные значения, которые уже были обработаны (см. рисунок 3).

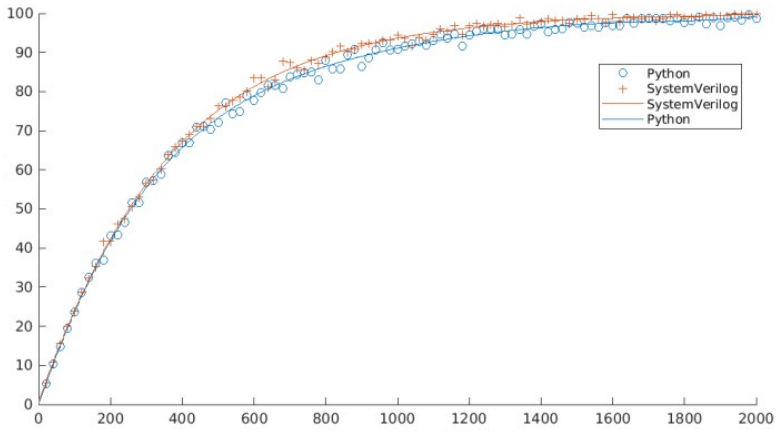

Рисунок 3 – Покрытие от операций записи для Python и SystemVerilog

Можно выделить то, что один и тот же тест в SystemVerilog имеет больше строк кода. Важно отметить, что увеличение количества строк кода может влиять на читаемость, поддержку и тестирование программных решений.

Заключение

Исследование верификации DUT с использованием окружений написанных на PyUVM и SystemVerilog-UVM проводилось по различным параметрам (объем кода, время симуляции, потребления памяти, покрытие). Несмотря на то, что моделирование на Python занимает больше времени и потребляет больше памяти, оно может быть более эффективным при условии, что генерация тактовых импульсов перенесена на сторону DUT. Моделирование с помощью PyUVM позволяет собирать входные данные и покрытии в удобном формате. Их можно проанализировать для создания новых методологий на основе методов машинного обучения, которые ещё больше ускорят процесс верификации.

Список литературы

1. H. Foster, “2022 Wilson Research Group Functional Verification Study,” Siemens Digital Industries Software, Tech. Rep., Oct. 2022.

2. D. Gadde, S. Kumari, A. Kumar, “Towards Efficient Design Verification -- Constrained Random Verification using PyUVM”, Cornell University, May 2024.
3. M. Sinerva, “UVM testbench in Python: feature and performance comparison with SystemVerilog implementation”, University of Oulu, June 2023.
4. Quinn, “Constrained Random Stimulus Generation using Python,” DVClub Europe, 2021.
5. M. DSU, PY-UVM Framework for RISC-V Single Cycle Core, May 8, 2023 (Accessed: August 4, 2023).