Scientific Research Publishing

World Journal of Engineering and Technology, 2023, 11, 409-416
https://www.scirp.org/journal/wjet
ISSN Online: 2331-4249
ISSN Print: 2331-4222

Python SystemVerilog (Python SV)

Santhosh Nagaraj Nag

Intel, San Jose, California, USA
Email: Santhoshnnnag@gmail.com

How to cite this paper: Nag, S.N. (2023) Python SystemVerilog (Python SV). *World Journal of Engineering and Technology*, **11**, 409-416.
https://doi.org/10.4236/wjet.2023.113029

Received: May 18, 2023
Accepted: June 27, 2023
Published: June 30, 2023

Copyright © 2023 by author(s) and Scientific Research Publishing Inc.
This work is licensed under the Creative Commons Attribution International License (CC BY 4.0).
http://creativecommons.org/licenses/by/4.0/
Open Access

Abstract

This paper discusses Python SystemVerilog (Python SV), a simulation-based verification approach leveraging the power of Python and SystemVerilog. The use of Python-implemented UVM classes in SystemVerilog enables users to write less code, minimize errors and reduce the verification time. This paper evaluates the use of Python SV in the verification of digital designs, its benefits, limitations, and future prospects. Python-SystemVerilog (Python-SV) is a research area that investigates the feasibility of building a high-level verification environment using Python and SystemVerilog. Python-SV aims to provide a unified framework for the design, simulation, and verification of digital systems, with an emphasis on ease of use and productivity. SystemVerilog is a hardware description and verification language that is widely used for designing digital systems. On the other hand, Python is a powerful, high-level programming language that is widely used in various fields, including software engineering, scientific computing, and data analysis. Python's popularity has grown in recent years, primarily due to its simplicity, ease of use, and wide range of libraries and frameworks. Python-SV research primarily focuses on the following areas: 1) Integration of Python and SystemVerilog: Python-SV aims to seamlessly integrate SystemVerilog and Python, allowing designers to write test benches and verification code in Python and interface them with SystemVerilog modules. This integration simplifies the development process, making it easier to write and maintain large and complex verification environments. 2) Development of Python libraries for verification: Python-SV research focuses on developing Python libraries specifically for digital system verification. These libraries provide a higher-level interface for writing test benches and other functions, such as analysis and visualization of simulation results. 3) Implementation of verification methodologies: Python-SV research investigates the implementation of various industry-standard verification methodologies, such as the Universal Verification Methodology (UVM), in Python. This implementation aims to enable designers to use Python to develop and simulate UVM-compliant test benches. 4) Development of simulation tools: Python-SV also explores the development of simulation tools that extend the capabilities of traditional SystemVerilog simulators. These tools leverage the capabilities of Python for complex data analysis and visualization and provide a more intuitive and user-friendly interface for working with simulation results. Overall, Python-SV research aims to bring the benefits of Python to the world of digital system verification, enabling designers to build more efficient, productive, and flexible verification environments.

Keywords

Python-SystemVerilog (Python-SV), Design, Framework, Simulation

1. Introduction

Python SystemVerilog [1] is an emerging methodology for developing advanced verification environments. Python is a high-level programming language with a wide range of libraries and frameworks [2] that can be utilized for various applications. SystemVerilog, on the other hand, is the hardware description language used to describe digital logic circuits and systems. Python SV offers an innovative approach to digital design verification, taking advantage of both Python and SystemVerilog.

2. Python SV Overview

Python SV facilitates the development of verification environments by allowing users to write structures and functions that can be used directly in SystemVerilog. Python-implemented UVM classes are used to construct verification environments, and the SystemVerilog simulator Simulation [3] is used to execute the verification process.

3. Applications of Python SV

Python SV has been used in various digital design verification environments, such as memory [4] controllers, interfaces, and high-speed serial interfaces. In these applications, Python SV has improved the quality of verification tests, enabling the identification of issues that might have been missed in other verification approaches.

4. Benefits of Python SV

Python SV offers several advantages over other verification methodologies.

1) Reduced Development Time: Python SystemVerilog [5] enables users to write simpler tests by reducing the amount of code required. This reduction in code development significantly reduces the development time.

2) Increased Test Coverage: Python SV has been shown to provide high levels of test coverage, ensuring that all scenarios are covered during the verification process.

3) Improved Debugging Capabilities: Python SV provides a more efficient debugging process than other methodologies, allowing users to find issues quickly and accurately.

4) Supports High-Level Abstractions: Python SV offers support for higher-level abstractions when compared to traditional HDLs. This makes designing, simulating, and testing digital designs more manageable, enabling faster execution.

Python SystemVerilog has various applications in the digital design verification process. Here are some of the best scenario uses of Python SV:

1) Memory Interface Verification: Memory interfaces are digital systems' most complex and critical components. Python SystemVerilog is an ideal verification methodology for memory interfaces, as it is easy to write and execute test cases for different memory operations. Python SV in this application ensures fewer code lines, and high-level abstraction, and provides efficient debugging.

2) High-Speed Serial Interface Verification: Verification of high-speed serial interfaces requires complex verification environments with thousands of test cases. Python SV is an excellent choice for this application because it facilitates the creation of high-speed models, which can simulate and verify the performance of these interfaces with high accuracy.

3) SoC Verification: Python SV verifies SoCs by verifying each module in a system. Python SV simplifies the verification process, by easing the creation of test benches and debug utilities. Python SV used on SoC verification makes it independent of any simulation tool-specific command interface and structured hierarchical connection support.

4) Network-On-Chip (NoC) Verification: Verification of NoCs has become a significant challenge in the design of complex SoCs. Python SV offers several advantages in this scenario, as it facilitates the development of scalable and complex test benches that stimulate different traffic patterns across the NoC topology.

5) Analog-Digital Interface Verification: Python SV is useful for Analog-Digital Interface (ADI) verification as it supports the UVM methodology and facilitates checking the behavior of ADC, DAC, and other analog-digital blocks. This application also highlights the ease of testbench implementation, integrating various Analog Mixed Signal elements in the verification testbench.

Python SV enhances the digital design verification process by providing a powerful and flexible simulation-based verification methodology. Its use cases include each module in a complex SoC design, network-on-chip (NoC) verification, high-speed serial interface verification, and analog-digital interface verification. These best scenario uses to demonstrate the potential of Python SV and highlight its superiority over traditional verification methodologies, such as pure SystemVerilog HDL, VHDL, and C++.

The Python-SV keywords are classified into several categories, including modules, tasks/functions, operators, and control structures. Each category is discussed in detail, providing a thorough understanding of the keywords' functionalities.

The module category includes keywords such as "module", "input", "output", and “wire”, which are used in defining SystemVerilog modules. The task/function category includes keywords such as “task”, “function”, “if”, “else”, and “case”, which are used in defining testbench components, such as stimulus generators and checkers.

The operator category includes keywords such as “module”, “and”, “or”, “not”, and “xor”, which are used in defining Boolean expressions and digital logic gates. The control structure category includes keywords such as “for”, “while”, “repeat”, and “forever”, which are used in controlling the execution flow of a testbench.

5. How to Connect SystemVerilog with Python

Verification of a digital design often requires an interaction between several language domains (SystemVerilog and C, SystemVerilog and Python, SystemVerilog and e-language, etc.). This article shows you how to set up a connection between SystemVerilog and Python.

SystemVerilog is not able to communicate directly with Python. Instead, the SV code first needs to talk to a C code via a DPI-C, with the C code then able to talk to the Python code. A SystemVerilog-Python connection, therefore, needs to follow certain guidelines, otherwise, the communication will fail. A connection of this kind is shown in the diagram below. (**Figure 1**)

This interconnection is made up of 4 layers:

1) The User layer
2) The Client layer
3) The Connection layer
4) The Server layer

5.1. The User Layer

The User layer is where the main verification activity takes place. It is the User layer that initiates a connection with another code written in a different programming language. The first step when initiating a connection is to invoke the *call_client*() function from the Client layer using the address and port number of the server. Optionally, we can provide a message for the Server.

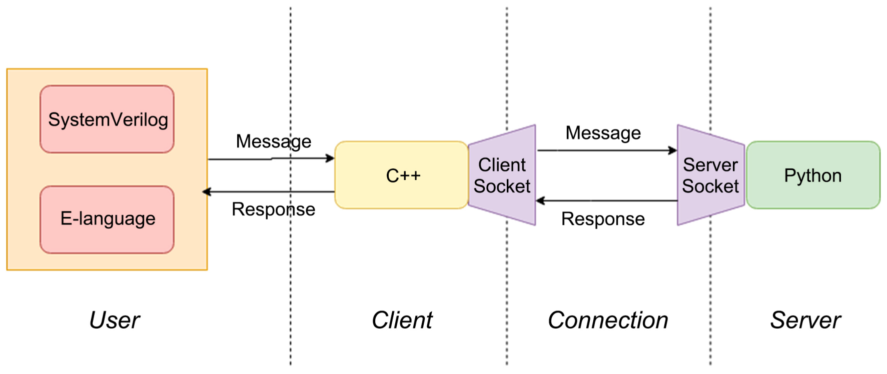

Figure 1. SystemVerilog communication workflow with Python.

5.2. The Connection Layer

The connection between SystemVerilog and Python works like a Client-Server application connecting 2 entities. The handshake and communication flow between the Client and the Server are shown in Figure 2 below.

In this Client-Server architecture, we are going to use a concept known as a socket, which is an endpoint in a network. A socket is bound to a port of the machine where the application is running. In a Client-Server architecture, both parties must have an associated socket. Moreover, both parties must use a common protocol (TCP or UDP) to be able to understand each other, e.g. when sending and receiving data.

Please note that, as depicted in the diagram, the communication must be initiated by the Client. The Server cannot initiate communication on its own.

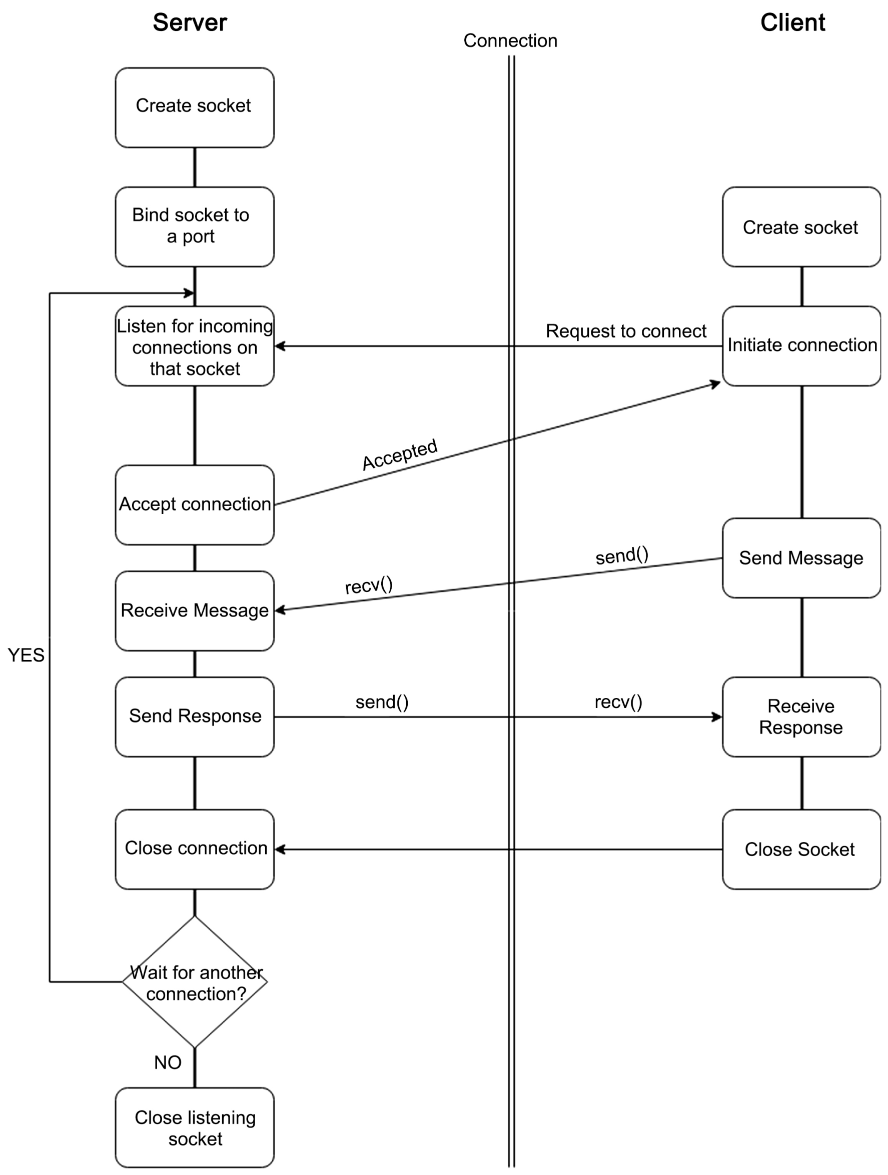

Figure 2. Client-Server communication/handshake flow.

5.3. The Client Layer

The Client layer acts as a proxy between the User layer and the Server layer. A connection with the Server is initiated from within this layer. The Client and the User layers are connected via the DPI-C. For more information on the DPI-C, please refer to this article.

Information (port number, hostname, message to send) received from the User layer is packed into a container struct (**client_config**) inside the *call_client*() function. The *call_client*() function is responsible for:

creating the client socket

managing the connection handshake

handling connection errors

returning the server response to the User layer

5.4. The Server Layer

The Server layer is responsible for providing a response to a user request.

After establishing a connection with the Client, the Server decodes the message received from the Client and generates the message to be sent back to the User. The Server layer and the Client layer are connected via the Connection layer.

In the current implementation, the Client is responsible for closing the connection. The Server catches this event and closes the associated connection handle. There may be different requirements for other applications (the Server is responsible for closing the connection, etc.).

6. Limitations of Python SV

While Python-SystemVerilog (Python-SV) is a promising framework for the design, simulation and verification of digital systems, there are some limitations that need to be considered. The following are some of the major limitations of Python-SV:

1) Learning curve: Python-SV requires programmers to learn two languages—Python and SystemVerilog. The syntax and semantics of these languages are very different and learning both to a high level of proficiency can be time-consuming and challenging.

2) Performance: Python-SV is an interpreted language and is generally slower than compiled SystemVerilog code. This can be a disadvantage when working with very large designs or handling large amounts of data, where performance is critical.

3) Compatibility: Not all SystemVerilog features are supported in Python-SV, which can limit the flexibility of the framework. Python-SV also requires specialized tools that may not be compatible with all SystemVerilog simulators or design tools.

4) Debugging and testing: Debugging Python-SV code can be more challenging than debugging SystemVerilog code due to the two languages' different na-

tures. Additionally, testing Python-SV code requires specialized tools that are not always readily available.

5) Availability of libraries: While Python has a vast library of useful modules and packages, the same cannot be said for Python-SV. The community around Python-SV is not as extensive as that of Python, and there may be a limited number of libraries available for certain applications.

6) Limited FPGA Support: Some FPGAs do not yet support Python SV verification, limiting the scalability of the methodology. This limits the scope of Python SV with designs that can only be implemented on specific FPGA devices.

7) Backend Compatibility: Python SV's simulator compatibility is limited. Hence, it cannot work with all available backend system simulators like NC-Verilog.

7. Future Prospects

The use of Python SV is on the rise and could revolutionize the digital design verification process. Several broader adoption frameworks, like Cocotb, enhance Python SV's capabilities, providing a simulation interchange format between Python and SystemVerilog. Further improvements on Python SV aim to expand its support for a wider range of FPGA devices, improving its applicability.

8. Conclusions

Python SV offers a unique approach to digital design verification, offering faster development, higher test coverage, and more efficient debugging. Its limitations must be taken into account. Nevertheless, its popularity is on the rise, and broader adoption frameworks like Cocotb enhance its application domain. The future potential of Python SV is vast, and further advancements on the development roadmap would increase its overall effectiveness.

Overall, Python-SV is a promising framework for the design and verification of digital systems, offering several benefits over traditional design methodologies. Python-SV is a strengthening technology in the field of digital system design and verification. Its potential has been proven through various papers, and it offers several benefits over traditional design methodologies.

While challenges exist, the ongoing research in this area is expected to overcome these challenges, enabling Python-SV to become a mainstream methodology for digital system design and verification.

Conflicts of Interest

The author declares no conflicts of interest regarding the publication of this paper.

References

[1] https://github.com/Kuree/pysv

[2] Jiang, S.N., Pan, P.T., Ou, Y.H. and Batten, C. (2020) PyMTL3: A Python Frame-

work for Open-Source Hardware Modeling, Generation, Simulation, and Verification. *IEEE Micro*, **40**, 58-66. https://doi.org/10.1109/MM.2020.2997638

[3] Shahzad, F. (2016) Pymote 2.0: Development of an Interactive Python Framework for Wireless Network Simulations. *IEEE Internet of Things Journal*, **3**, 1182-1188. https://doi.org/10.1109/JIOT.2016.2570220

[4] Huggi, S. and Jamuna, S. (2020) Design and Verification of Memory Elements Using Python. 2020 *IEEE International Conference on Electronics, Computing, and Communication Technologies* (*CONECCT*), Bangalore, 2-4 July 2020, 1-4. https://doi.org/10.1109/CONECCT50063.2020.9198470

[5] https://github.com/topics/SystemVerilog