International Journal of Control and Automation
Vol.7, No.2 (2014), pp.221-230
http://dx.doi.org/10.14257/ijca.2014.7.2.21

System-Level Verification Platform using SystemVerilog Layered Testbench & SystemC OOP

Young-Jin Oh and Gi-Yong Song*

Department of Electronics Enginneering, College of Electrical and Computer Engineering, Chungbuk National University, Cheongju,Chungbuk, 361-763, Korea

goodmen913@cbnu.ac.kr, gysong@cbnu.ac.kr

Abstract

Systems have recently performed multiple functions through a combination of several IPs. SystemVerilog has useful components for modeling and verification at System-level. The OOP of SysemVerilog supports only single inheritance in a verification environment based on a layered testbench of SystemVerilog. It is restricted to construct environment verification. SystemC is a language for system level design at multiple abstraction levels and supports multiple inheritance. We adopt SystemC to design components of a verification platform which employ multiple inheritance, and combine it with the SystemVerilog-based verification platform using SystemVerilog DPI and the ModelSim macro in this paper. Employing the multiple inheritance of SystemC makes the design of a verification environment simple and easy through code reusability. Another characteristic of OOP with SystemVerilog and SystemC is that it can create a reconfigurable verification platform.

Keywords: *SystemVerilog, SystemC, OOP, verification environment, layered testbench*

1. Introduction

When a SoC (system-on-a-chip) grows larger, connections between IPs become more complex. In addition, as contemporary real chips are usually multi-functional, the interaction between various IPs must be verified at a system level. A system-level design and functional verification methodology based on high-level abstraction becomes more important to increase the productivity of a SoC design. In system-level design and verification, hardware/software partitioning of a system through design space exploration affects the structure and performance of the final system, and the importance of verifying functional interaction between hardware part and software part is increasing ever [1-3].

The typical functional verification of hardware mainly uses BFM (bus functional model) because most IPs for a system are connected to and controlled through a bus [1-2]. However, a SoC performs various functions through combination of hardware and software. Co-verification is needed which can verify hardware and software for a functional verification of a system simultaneously. Most previous research on co-verification have used IPC(interprocess communication) with semaphore, pipe or socket for communication between C-code and HDL module[2]. In this case, IPs are added in the form of a library to HDL simulator through Verilog PLI in order to use C-code, with system functions of the IP or kernel called from inside of added system functions[1, 3-5]. However, as hardware and software parts work independently of each other, this goes through a complex process of

* Corresponding Author

ISSN: 2005-4297 IJCA

registration and call. The more complex a system is, the more important the data transmission between the various IPs becomes. In order to ascertain this, verification is essential at system level. If we can proceed to the design and verification phase using one language at a system level, the productivity of a SoC increases by eliminating complex processes.

SystemVerilog and SystemC is a representative language for design and verification as a single language at system level. Since SystemVerilog does not allow multiple inheritance, when an verification environment is configured only with SystemVerilog, it has restrictions [2]. As the internal structure and functionality of environment class objects are alike to each other, the components in the environment class need to be designed with multiple inheritances to increase code reusability.

Currently, a verification environment has a layered structure in order to verify large-scale systems. Each component operates as an independent object and uses communication protocols for data transmission. SystemVerilog and SystemC can implement each object independently and provide the ability to reuse components [2]. This means that there are configuration advantages in the verification platform. Layered structures can make up a reconfigurable environment which can combine implanted components. As SystemVerilog does not allow multiple inheritances [7-9], this paper proposes SystemC design unit employing multiple inheritance, and has it combined with a SystemVerilog-based verification environment. The designed component of environment class is compiled to the shared library with SystemVerilog DPI(direct programming interface) [7, 9-11] and ModelSim macro in order to be visible from the SystemVerilog-based verification environment [15]. SystemVerilog DPI provides a way to interface with any other foreign language. Functions and tasks registered to the shared library using DPI can be called out like native ones. ModelSim recently supports SystemC simulation with built-in compiler for SystemC design unit. In order to simulate SystemC design unit with ModelSim, the SystemC design unit should be modified using some macros provided by ModelSim. The communication protocol can be easily changed by replacing the interface with a different interface in considering the designed IPs can be re-used with the different bus architecture.

2. Environment Using System Verilog & System C

2.1. System Verilog-based Layered Test Bench

SystemVerilog is a set of extensions to the Verilog HDL that allows higher level modeling and efficient verification of large digital systems. The enhanced features of SystemVerilog from the verification point are as follows.

- Constrained-random stimulus generation
- Functional coverage
- Higher-level structures, especially OOP
- Multithreading and interprocess communication
- Support for HDL types such as Verilog's 4-state values
- Tight integration with event-simulator for control of the design

A key concept for any modern verification methodology is a layered testbench which helps you control the complexity, which frequently occurs in case of a testbench design itself, by breaking the problem into manageable pieces. The two representative

structures of a layered testbench are introduced in [7-8]. The structure of a layered testbench in [7] is chosen in this paper.

2.2. Multiple Inheritance using OOP of System C

System C is a language for system level design at multiple abstraction levels [8-9]. SystemC adds the capabilities of C++ by enabling modeling of hardware descriptions and hence is powerful enough to describe all kinds of verification environments from the signal level to the transaction level[6-8]. SystemC can implement components of a verification environment which is employed in multiple inheritance. It supports the concepts of time, hardware data type, concurrency, and hierarchy. Because the inheritance, which is one of the important characteristics of OOP, provides polymorphism, it is easy to reconfigure components through a pointer of the base class [9].

A SoC which consists of several devices performs multiple functions. Verification which mimics the circumstance around a real SoC is necessary in order to verify the interaction between devices, because verification environments consists several component classes which have a variety of sub-classes inside. When it verifies an interaction between devices, the device traffic except the main focus of the DUT is generally called background-traffic. Objects of each environment class in [7] have the same structure as shown in Figure 1.

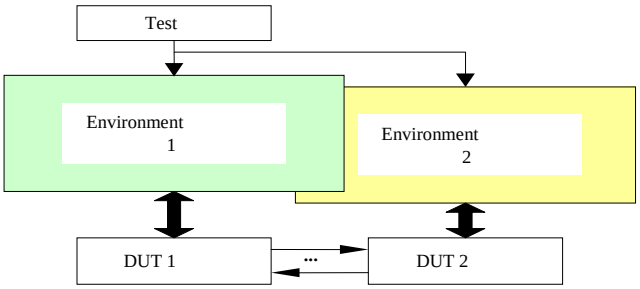

Figure 1. Structure of a system-level verification environment

As classes of environment have similar functions and internal structures, redundant codes are likely to be generated in classes. If we define base-classes for the fundamental operations of components and the verification process, code reusability can significantly be improved by defining a component class of the environment as a derived class out of base sub-classes representing operational diversity. Consequentially, we can reconfigure classes using multiple inheritance. The definition of each component class done in such a way as described above should employ multiple inheritance in the course of class derivation in order to gain code reusability. This allows multiple inheritance to be applied to the design of object processes at the behavioral level without sacrificing the simulation performance. As a result, employing multiple inheritance of SystemC makes the design phase of the verification platform simple and easy. For this reason we selected SystemC for modeling the verification platform employing multiple inheritance.

3. Implement of System-Level Verification Environment

SystemVerilog with OOP characteristics has defined components in which a class contains objects of another class, but when class is implemented by inheritance, it allows only single inheritance [9]. In the case of single inheritance, it is difficult to reuse or add new code

without directly editing the existing code when the DUT is changed. Supposing that the implemented verification platform allows multiple inheritance, we can create new components using extended classes through a combination of necessary methods which are inherited from other classes. The components are designed with SystemC constructs and designed classes are linked to the SystemVerilog-based verification platform in this paper. By combining SystemVerilog methods and components with SystemC, a reconfigurable verification environment is implemented. In an example, Figure 2 shows a hierarchical structure of generator component employing multiple inheritance. The SystemC module basis, sc_module, is for designing a module. And Gen_base class and Env_base class contain fundamental operations of generator component itself and verification process, respectively.

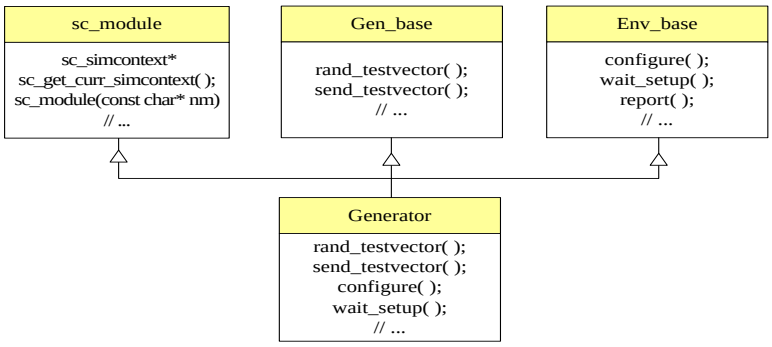

Figure 2. Hierarchical structure of generator component

When a DUT is verified with a platform consisting of many IPs, configuring a verification platform according to the characteristics of the data flow is important. When we compose a verification platform using the SystemC channel, it must be implemented with user-defined channels pursuant to the flow of data to process simulation under conditions that resemble a real environment. For this reason, two types of user-defined channel are implemented: A FIFO channel with blocking and non-blocking modes. To link generator component and FIFO channel to the verification environment, components should be modified with SystemVerilog DPI and ModelSim macro, and compiled to shared library for the SystemVerilog-based verification environment. The SystemVerilog module has to import the methods of generator components which employ the multiple inheritance of SystemC constructs by using a DPI-SC modifier to link it to the verification platform based on SystemVerilog. The partial code of the SystemVerilog top module is shown in Figure 3.

```
module top ;
Generator i_ Gen( );
import "DPI-SC" context function void Gen_rand_testvector (output bit [33:0] i);
import "DPI-SC" context function void Gen_send_testvector (bit [33:0] i);
import "DPI-SC" context function void Gen_configure( );
import "DPI-SC" context function void Gen_wait_setup( );
// …
bit RESETn;
bit CLK;
DUT_if DUT_IF (CLK, RESETn);
DUT1 DUT1(DUT_IF)
// …
endmodule;
```

Figure 3. Partial code of the SystemVerilog top module

The DPI-SC modifier indicates to the SystemVerilog compiler that those methods of generator components are imported functions/task defined in the SystemC shared library. The task which is imported from the DPI-SC modifier compiler calls the prototype implemented with C function in sc_dpiheader.h file. Each variable that is passed through the DPI-SC has two matching definitions; one for the SystemVerilog side, and one for the SystemC side. The sub-modules of the top module are able to reference the imported function/task according to the SystemVerilog search rule.

DUT_if which is defined using the interface constructor includes bus signals as well as read/write tasks which are internally defined to drive signals under bus read/write protocols. The SystemVerilog layered testbench can drive the signals of DUT.

We can configure the environment to verify through selection of components at the simulation phase. Communication components such as the mailbox of SystemVerilog or user-defined FIFO channel modes of SystemC are selected with a constraint ID. We can also change the verification routine using the callback method. Figure 4 shows how we reconfigured the test module using callback and ID as a selection variable.

```
program test(ahb_ifahb_if_);
`include "Environment.sv"
Environment env_;
bit [1:0] ID;
initial begin
env_ = new (ahb_if_);
begin
ID = randomize();
end
env_.gen_cfg ( ID);
env_.build ();
begin
Driver_cbssw = new();
env_.drv_.cbsq.push_back(sw);
end
env_.run ( );
env_.wrap_up ( );
end
endprogram
```

Figure 4. Test module using callback method

4. Experimental Results

A Figure 5 shows the structure of the verification platform including components of the SystemC design units, the generator and FIFO channel, to SystemVerilog-based layered structure.

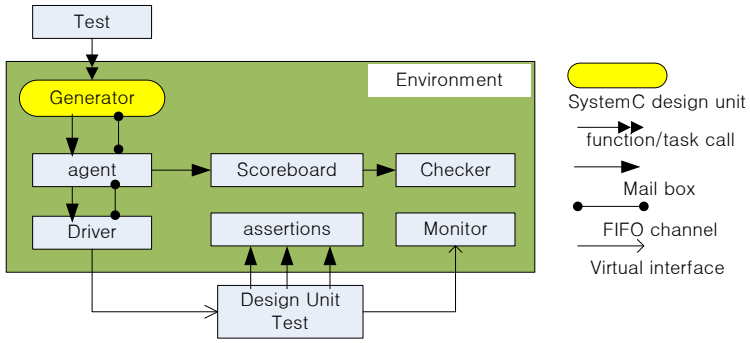

Figure 5. Structure of the reconfigurable verification platform

The designed generator uses the multiple inheritance of SystemC to replace an existing part of SystemVerilog. The execution times of each generator according to the design method are shown in Table 1 for several numbers of testvector.

Table 1. Performance comparison (time: ms)

# of trans.	Generator designed with SystemVerilog	Generator designed with SystemC
50	0	0
500	46	47
5000	301	321
10000	584	635

The performance of the generator designed with SystemC and the generator designed with SystemVerilog are very similar but as the number of testvectors grows, simulation time is observed to increase.

This phenomenon is presumed to be an overhead due to the simulation of a SystemC design unit on ModelSim which is an HDL-centric simulator. The kernel of ModelSim is used in order to simulate HDL code. It is different from the SystemC simulator kernel. The ModelSim kernel is needed for switching time to recognize the SystemC codes after stopping the HDL code performing temporarily in order to execute SystemC code. Therefore, the more we use SystemC code, the more switching between HDL and SystemC codes occurs. This inevitably produces an increase in total execution time for simulation. We can reduce development time through the partial modification of the existing verification system, obtaining the accuracy and reliability, rather than developing a new verification system.

The result from each verification environment constructed with the mailbox of SystemVerilog, FIFO channel of SystemC in blocking or non-blocking mode is shown in Figure 6. Figure 7 and Figure 8 respectively. We ascertain that various generators can be configured using the multiple inheritance of SystemC, and verification platform can also be reconfigured by selecting routines with SystemVerilog methods.

Figure 6. Result in case of SystemVerilog mailbox

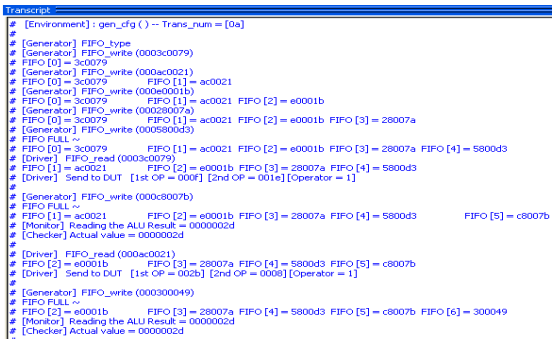

Figure 7. Result in case of FIFO channel in blocking mode

Figure 8. Result in case FIFO channel in non-blocking mode

The verification platform consisted of two environment classes shown in Figure 9. We consider the DUT2 that generates background traffic which causes bus contention. The designed IP components of DUT1 are described as doing simple ALU operations with a bus slave interface. When the bus protocol is switched to another one, only the interface with other communication protocols needs to be replaced with the verification process remaining the same. We confirm that the verification platform is configured using the multiple inheritance of SystemC.

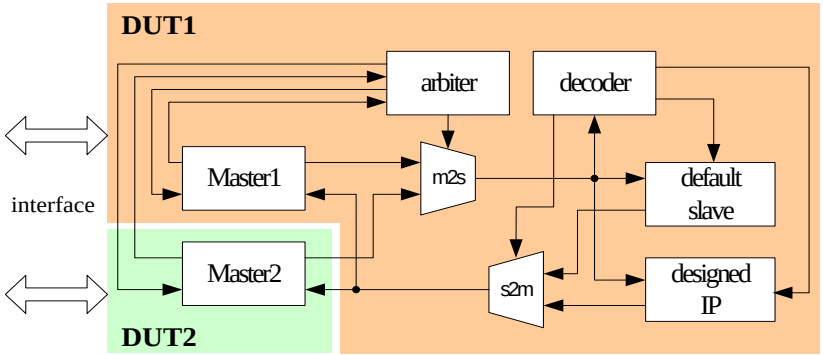

Figure 9. Verification environment consisted of two environment classes

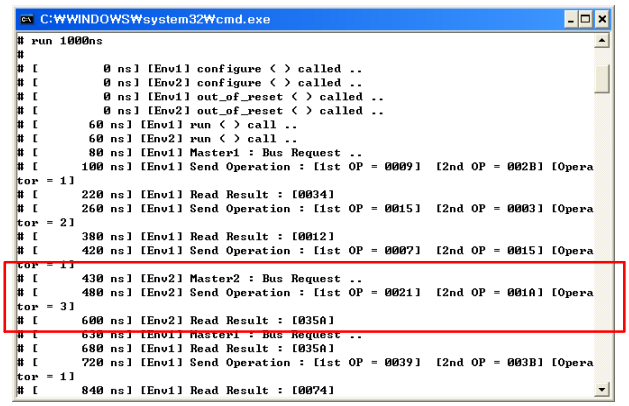

Figure 10. Result of a functional verification

The function verification result of the DUT is shown in Figure 10. The message highlighted in the box says that master2 in DUT2 writes and read data to or from the designed IP after driving the bus request signal while master1 in DUT1 performs data transmission.

5. Conclusions

Because the OOP of SystemVerilog does not allow multiple inheritance, we have constraints on the configuration of the verification platform. However SystemC can secure polymorphism by allowing multiple inheritance. We adopt SystemC to design a component of the class which employs multiple inheritance and link SystemC design units to the verification environment using SystemVerilog DPI and ModelSim macro. We created various verification platforms through a combination of multiple inheritance applied to components of SystemC and SystemVerilog-based verification platform in this paper.

The generator and user-defined channel were designed with SystemC constructs employing multiple inheritance, and were then applied as part of a SystemVerilog-based verification platform. The operation of the environment with SystemC design units was validated through simulation on a DUT, and the performance of a SystemC generator is roughly the same as a SystemVerilog one.

Applying multiple inheritance to the design of an object raises source code reusability without sacrificing the simulation performance allows reconfiguration of the verification platform through a combination of necessary components using SystemC multiple inheritance. The reconfigurability of the verification environment based on OOP is gained easily due to the characteristic of OOP.

Acknowledgements

“This work was supported by the research grant of the Chungbuk National University in 2011”.

References

[1] A. Ki, “SoC Design and Verification Methodologies and Enviroments, HongreungScience”, Seoul, Korea, **(2008).**
[2] M. -K. You and G. Y. Song, “SystemVerilog-based Verification Environment Employing Multiple Inheritance of SystemC”, IEICE TRANS**,** vol. E-93A, no. 5, **(2010)**, pp. 989-992.
[3] S. Sutherland, “Modeling FIFO Communication Channels Using SystemVerilog Interface”, SUNG Boston, **(2004).**

[4] J. Bergeron, "Writing Testbench using systemVerilog", Springer, New York, **(2006).**
[5] S. Yoo and A. A. Jerraya, "Hardware/software cosimulation from interface perspective", Computers and Digital Techniques' *IEE* Proceedings, vol. 152, issue3, **(2005).**
[6] S. Sutherland, S. Davidmann and P. Flake, "SystemVerilog for Design (2nd Edition)", A Guide to Using SystemVerilog for Hardware Design and Modeling, Springer, New York, **(2006).**
[7] T. Grotker, S. Liao, G. Martin and S. Swan, "System Design with SystemC", Kluwer Academic Publishers, Netherlands, **(2002)**.
[8] C. Spear and G. Tumbush, "SystemVerilog for Verification (2nd Edition)", A Guide to Learning the Testbench Language Features, Springer, LLC, **(2012).**
[9] M. Mintz and R. Ekendahl, "Hardware Verification with SystemVerilog", An Object-Oriented Framework, Springer, LLC, **(2007).**
[10] S. Sutherland, "Integrating SystemC Models with Verilog and SystemVerilog Models Using the SystemVerilog Direct Programming Interface", SNUG, Boston, **(2004).**
[11] M. –K. You, Y. –J. Oh and G. –Y. Song, "System-level Hardware Function Verification System", Journal of The Institute of Signal Processing and Systems, vol. 11, no 2, **(2010),** pp177-182.
[13] SystemC Language Reference Manual, http://www.systemc.org, **(2012)** October.
[14] ModelSim SE User's Manual, http://www.mentor.com, **(2011)** October.
[15] http://www.soccentral.com, **(2012)** October.
[16] SystemVerilog3.1a Language Reference Manual: Accellera's Extensions to Verilog, Accellera, Napa, California, **(2004)**.

