International Journal of Advanced Information Science and Technology (IJAIST) ISSN: 2319:2682
Vol.3, No.9, September 2014 DOI:10.15693/ijaist/2014.v3i9.45-49

SystemVerilog-based Verification Environment Employing SystemC OOP

Gi-Yong Song
Professor/Department of Semiconductor Engineering College of Electrical Computer Engineering Chungbuk National University, Cheongju, Chungbuk, Korea

Young-Jin Oh
Professor/Department of Semiconductor Engineering College of Electrical Computer Engineering Chungbuk National University, Cheongju, Chungbuk, Korea

***Abstract*—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. SystemVerilog has useful components for modeling and verification at system-level, but the OOP of SystemVerilog supports only single inheritance. So, SystemVerilog poses a limit to constructing verification environment in a diverse manner. SystemC is a language for system level design at multiple abstraction levels and supports multiple inheritance. We adopt SystemC to employ multiple inheritance, and combine it with the SystemVerilog-based verification environment. The environment can select verification routines during verification process using SystemVerilog method such as callback or constraint, making it a reconfigurable one.**

Index terms - Verification environment, SystemVerilog, SystemC OOP, Multiple inheritance,

I. Introduction

Since the early 1980s, when schematic capture was introduced as an efficient way to design very large-scale integration circuits[1], the design methodology has seen a lot of progress, and today's HDL-based designs are portable and independent of technology, allowing designer to modify and re-use designs to keep pace with improvements in technology. When we deal with a complex digital system with several components, system-level design and functional verification methodology based on a high-level abstraction becomes more important to increase the productivity of a digital system design.

We need hardware units for dedicated functions and peripheral devices, linked together by communications network for complete system.

The typical functional verification of hardware mainly uses BFM(bus functional model) of a design because most IPs for a system are connected to and controlled through a bus. As contemporary chips usually are multifunctional, the interactions between the various devices need to be verified at system-level[2][4-5]. For system-level verification, several objects of environment class are used so that each object verifies the functions of corresponding device. So the components of environment class need to be designed with multiple inheritance in order to increase code reusability because the internal structures and functions of environment class objects are alike each other.

SystemVerilog which is an extension to Verilog HDL has characteristics of both hardware description languages and hardware verification language[3][9-13]. SystemC extends C++ by introducing capabilities for modeling hardware. It is a single language for both hardware modeling and software coding[3][7-8][11-12]. As internal structure and functionality of environment class objects are alike each other, the components in environment class need to be designed with multiple inheritance to increase code reusability.

In order to verify a system consisting of several IPs, we need to implement a new verification environment for each communication protocols and bus architectures[2]. If components are implemented using multiple inheritance of SystemC, the verification environment can be reconfigured through partial change of components. In a co-verification of hardware and software, IPC(interprocess communication) is adopted to communicate with other units[11-12]. We use callback to select verification routine with SystemVerilog mailbox or with SystemC FIFO channel.

II. SystemVerilog & SystemC

Looking at the two languages, SystemC and SystemVerilog, it is obvious that SystemVerilog extends the Verilog HDL scope to object orientation, while SystemC extends the C/C++ scope toward hardware. Both languages support such concepts as OOP, events, interface and signal[18].

A. Layerd Testbench of SystemVerilog

SystemVerilog is a set of extensions to the Verilog HDL, allowing higher level modeling and efficient verification of large digital systems [10]-[13]. SystemVerilog adds hardware functional verification constructs such as OOP, randomization, thread, IPC, etc.

A key concept for any modern verification methodology is the layered testbench which helps control the complexity[6][10] which frequently occurs in case of a testbench design itself, by breaking the problem into manageable pieces. Testbench creates stimulus, and applies it to DUT in order to check operations of DUT. And it ascertains whether the behavior is correct using response capture.

SystemVerilog is a set of extensions to Verilog HDL. Those extensions enable us to model hardware at a higher level and to make more efficient verification for large systems. SystemVeilog adds such constructs as OOP, randomization thread and IPC for functional verification of hardware. Still, SystemVerilog allows only single inheritance. Two structures of a layered testbench using SystemVerilog are introduced in [10][11].

We chose the structure of a layered testbench in [11] here. The whole structure is shown in Figure.1.

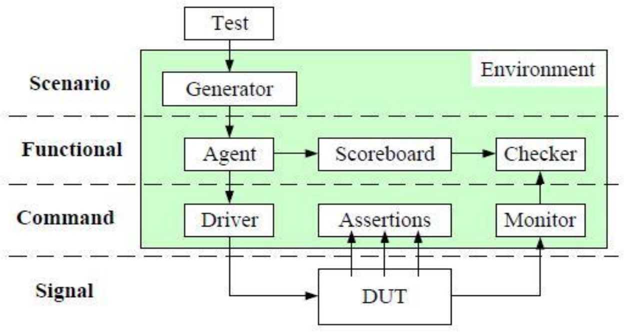


Figure 1. Structure of a layered testbench

B. Multiple Inheritance of SystemC

A SoC performing multiple functions consists of several devices, so it is necessary to verify the interaction between devices. SystemC is a language for system level design at multiple abstraction levels. Being built on standard C/C++ language, the SystemC describes functions and communications at various levels of abstraction. As it supports concepts of time, hardware data type, concurrency, and hierarchy[8-9]. SystemC uses a layered approach that allows for the flexibility of introducing new, higher-level constructs sharing an efficient simulation. The base layer of SyatemC provides an event-driven simulation kernel to work with event and processes in an abstract manner[8].

As multiple 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. By defining a component class of the environment as a derived class out of base sub-classes representing operational diversity, code reusability can significantly be improved. The definition of each component class in such a way as described above should employ multiple inheritance in the course of class derivation in order to gain code reusability. Applying multiple inheritance to the design of an object at behavioral level does not scarify the simulation performance. In the result, employing multiple inheritance of SystemC makes the design phase of verification environment simple and easy.

As an example, Figure. 2 shows a hierarchical structure of generator component employing multiple inheritance. The SystemC module, sc_module, is a base for designing a module. And Gen_base class and Env_base class contain fundamental operations of generator component itself and verification process, respectively.

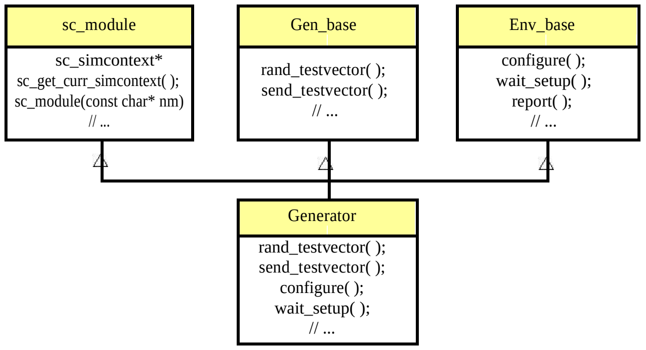


Figure. 2. Hierarchical structure of generator component

III. VERIFICATION ENVIRONMENT WITH SYSTEMVERILOG EMPLOYING SYSTEMC OOP

The components of environment class can be defined as a single flat class that has variables and routines declared and defined using SystemVerilog. However, if class has a flat hierarchy, it is hard to expand upon existing code in an organized way without directly editing the code. The components are designed with SystemC constructs, and classes are linked to the SystemVerilog-based verification environment in this paper.

By combining SystemVerilog methods and components with SystemC, a reconfigurable verification environment is implemented. To link generator component to the verification environment, generator component should be modified with SystemVerilog DPI[18] and ModelSim macro[19], and compiled to shared library for the SystemVerilog-based verification environment.

For this reason, component of environment class is designed with SystemC constructs as SystemVerilog does not allow multiple inheritance, and linked to the SystemVerilog-based verification environment

As shown in Figure. 3 SystemVerilog module has to import the methods of generator component employing multiple inheritance of SystemC constructs, using DPI-SC modifier in order to link it to the SystemVerilog-based verification environment. The DPI-SC modifier indicates to SystemVerilog compiler that those methods of generator component are import functions/tasks defined in the SystemC shared library. The sub-blocks of top-level module are able to reference the imported functions/tasks through SystemVerilog search rule. Each variable that is passed through the DPI-SC has two matching definitions; one for the SystemVerilog side, and one for the SystemC side. It is designer's responsibility to use compatible type

```
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 task Gen_wait_setup( );
// ...

bit RESETn;
bit CLK;
DUT_if DUT_IF (CLK, RESETn);
DUT1 DUT1(DUT_IF)
// ...
endmodule;
```

Figure. 3. Partial code of top module of SystemVerilog

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.

Figure.4 shows the structure of the verification environment including components of the SystemC design units such as generator and FIFO channel. The generator uses the multiple inheritance of SystemC to replace an existing counterpart of SystemVerilog.

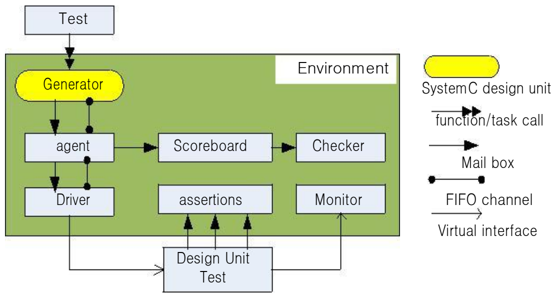


Figure.4 The verification environment including SytemC design units.

We can configure the environment to verify through selection of components at the simulation phase, and also change the verification routine using the callback method. The callback routine can be defined differently in each test. As a result, the test can add new functionality to the driver using callbacks without editing the driver class[3]. A callback method created in the top-level module is called from the driver in the verification environment.

We can configure the environment to verify through selection of components at the three simulation phase[12].

- Step 1 : Build phase
  - Generate configuration
  - Build environment
    allocate and connect the testbench components based on the configuration
  - Reset the DUT
  - Configure the DUT
- Step 2 : Run phase
  -Start environment
    run the testbench components such as BFMs and stimulus generators.
  -Run the test
    start the test and then wait for it to complete.
- Step 3 : Wrap-up phase - Sweep
    after the lowest layer completes, wait for the final transactions to drain out of the DUT.
  -Report
    create the final report on whether the test passed or failed

Figure. 5 shows how we reconfigured the test module using callback and ID as selection variables.

```
 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 5. The callback method of systemVerilog

.

IV. Experimental Results

The generator uses multiple inheritance of SystemC to replace an existing part of SystemVerilog. The execution times of generators corresponding to each design method are shown in Figure. 6 for several number of test vectors.

The performance of each generator, one designed with SystemC and the other designed with SystemVerilog, is very similar, but as the number of test vector grows, simulation time is observed to increase

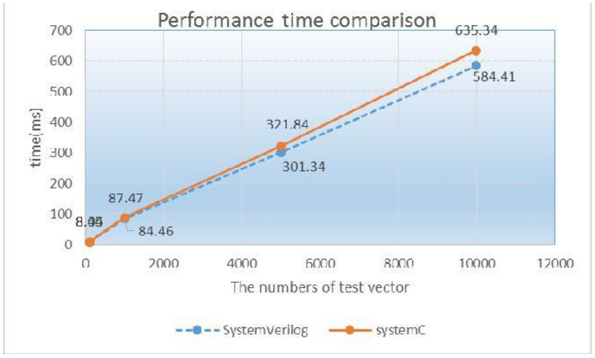
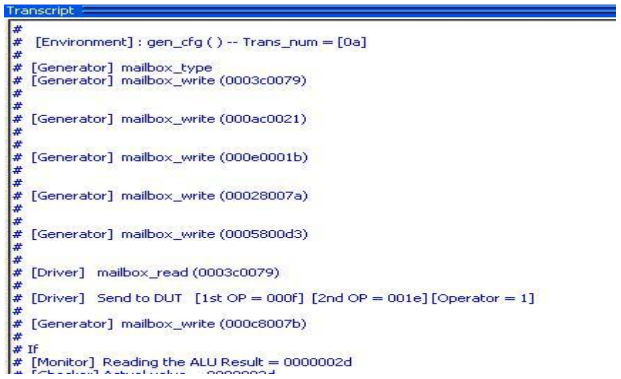


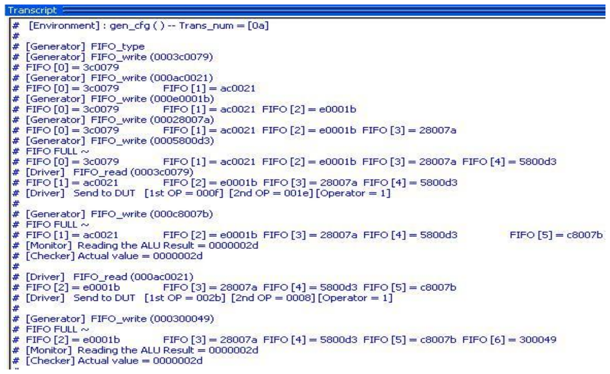
Figure 6. Comparison of performance of design units

The results from each verification environment constructed with mailbox, or FIFO channel is shown in Figure. 7, Figure. 8, respectively.



```
Transcript
#
#  [Environment] : gen_cfg ( ) -- Trans_num = [0a]
#
# [Generator] mailbox_type
# [Generator] mailbox_write (0003c0079)
#
#
# [Generator] mailbox_write (000ac0021)
#
#
# [Generator] mailbox_write (000e0001b)
#
#
# [Generator] mailbox_write (00028007a)
#
#
# [Generator] mailbox_write (0005800d3)
#
#
# [Driver]  mailbox_read (0003c0079)
#
# [Driver]  Send to DUT  [1st OP = 000f] [2nd OP = 001e] [Operator = 1]
#
# [Generator] mailbox_write (000c8007b)
#
# If
# [Monitor] Reading the ALU Result = 0000002d
```

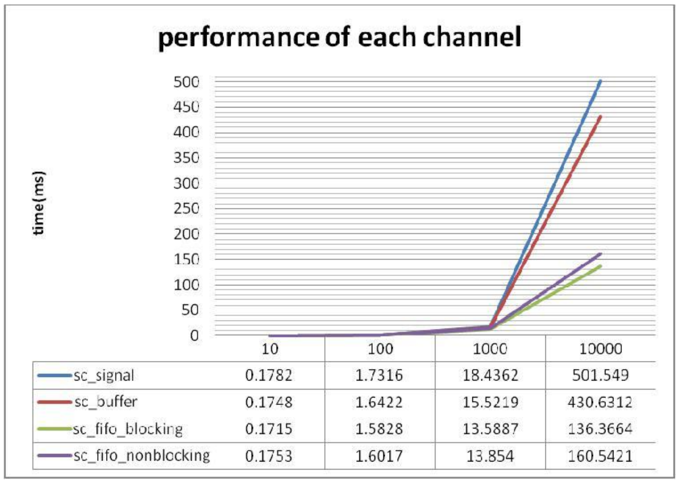
Figure 7. Results in case of a SystemVerilog mailbox



```
Transcript
#  [Environment] : gen_cfg ( ) -- Trans_num = [0a]
#
# [Generator] FIFO_type
# [Generator] FIFO_write (0003c0079)
# FIFO [0] = 3c0079
# [Generator] FIFO_write (000ac0021)
# FIFO [0] = 3c0079          FIFO [1] = ac0021
# [Generator] FIFO_write (000e0001b)
# FIFO [0] = 3c0079          FIFO [1] = ac0021  FIFO [2] = e0001b
# [Generator] FIFO_write (00028007a)
# FIFO [0] = 3c0079          FIFO [1] = ac0021  FIFO [2] = e0001b  FIFO [3] = 28007a
# [Generator] FIFO_write (0005800d3)
# FIFO FULL ~
# FIFO [0] = 3c0079          FIFO [1] = ac0021  FIFO [2] = e0001b  FIFO [3] = 28007a  FIFO [4] = 5800d3
# [Driver]  FIFO_read (0003c0079)
# FIFO [1] = ac0021          FIFO [2] = e0001b  FIFO [3] = 28007a  FIFO [4] = 5800d3
# [Driver]  Send to DUT  [1st OP = 000f] [2nd OP = 001e] [Operator = 1]
#
# [Generator] FIFO_write (000c8007b)
# FIFO FULL ~
# FIFO [1] = ac0021          FIFO [2] = e0001b  FIFO [3] = 28007a  FIFO [4] = 5800d3          FIFO [5] = c8007b
# [Monitor] Reading the ALU Result = 0000002d
# [Checker] Actual value = 0000002d
#
# [Driver]  FIFO_read (000ac0021)
# FIFO [2] = e0001b          FIFO [3] = 28007a  FIFO [4] = 5800d3  FIFO [5] = c8007b
# [Driver]  Send to DUT  [1st OP = 002b] [2nd OP = 0008] [Operator = 1]
#
# [Generator] FIFO_write (000300049)
# FIFO FULL ~
# FIFO [2] = e0001b          FIFO [3] = 28007a  FIFO [4] = 5800d3  FIFO [5] = c8007b  FIFO [6] = 300049
# [Monitor] Reading the ALU Result = 0000002d
# [Checker] Actual value = 0000002d
```

Figure 8. Results in case of a SystemC FIFO channel

We ascertain that various generators can be configured using multiple inheritance of SystemC, and verification environment also can be reconfigured by selecting routines with SystemVerilog methods. Communication among various modules is made using elementary SystemC channel such as sc_signal, sc_buffer or sc_fifo. The sc_signal and sc_buffer channel perform an operation of input or output per data instead of processing through data grouping. Comparison of performance of each channel is shown in Figure. 9.



	10	100	1000	10000
sc_signal	0.1782	1.7316	18.4362	501.549
sc_buffer	0.1748	1.6422	15.5219	430.6312
sc_fifo_blocking	0.1715	1.5828	13.5887	136.3664
sc_fifo_nonblocking	0.1753	1.6017	13.854	160.5421

Figure 9. Comparison of performance of each channel

V. CONCLUSIONS

As 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 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 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 SystemC OOP such as multiple inheritance to the design of an object raises source code reusability without sacrificing the simulation performance. Also, it 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.

ACKNOWLEDGEMENT

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

REFERENCES

[1]. Weng Fook Lee, “Verilog Coding for Logic Synthesis”, Wiley & Sons inc. 2011.

[2]. Jason R. Andrews, "Co-Verification of Hardware and Software for ARM SoC Design",Elsevier Inc., 2005.
[3]. Myoung-Kenu You, Gi-Yong Song, "SystemVerilog-based Verification Environment Employing Multiple Inheritance of SystemC". IEICE TRANS. Vol E-93A, No 5, May 2010, pp.989-992.
[4]. Ando Ki, "SoC Design and Verification : Methodologies and Environments". Hongreung Science, 2008.
[5]. S. Yoo, A.A. Jerraya, "Hardware/software cosimulation from interface perspective, Computers and Digital Techniques IEE Proceedings". vol.152, issue3, 2005.
[6]. T. Jozawa, L. Huang, T. Sakai, S. Takeuchi, M. Kasslin, "Heterogeneous co-simulation with SDL and SystemC for protocol modeling". RWS, 2006, pp.603-606.
[7]. S. Chikada, S. Honda, H. Tomiyama, H. Takada, "Cosimulation of ITRON-based embedded software with SystemC". HLDVT, 2005, pp.71-76.
[8]. Thorsten Grotker, Stan Liao, Grant Martin, Stuart Swan, "System Design with SystemC". Kluwer Academic Publishers, 2002.
[9]. Mike Mintz, Robert Ekendahl, "Hardware Verification with SystemVerilog : An Object-Oriented Framework". Springer, 2007.
[10]. Stuart Sutherland, Simon Davidmann and Peter Flake, "SystemVerilog for Design (2nd Edition): A Guide to Using SystemVerilog for Hardware Design and Modeling". Springer, 2006.
[11]. Chris Spear, "SystemVerilog for Verification (2nd Edition): A Guide to Learning the Testbench Language Features", Springer, 2008.
[12]. Stuart Sutherland, Integrating SystemC Models with Verilog and SystemVerilog Models Using the SystemVerilog Direct Programming Interface, SNUG, USA, Boston, 2004
[13]. Stuart Sutherland, "SystemVerilog, ModelSim, and You," MentorUser2User,2004.
[14]. Myoung-Kenu You , Young-Jin Oh, Gi-Yong Song, "System-level Hardware Function Verification System". Journal of The Institute of Signal Processing and Systems, Vol. 11, No 2, April 2010. pp177-182.
[15]. ModelSim SE User's Manual, http://www.mentor.com
[16]. SystemC Language Reference Manual, http://www.systemc.org
[17]. SystemVerilog3.1aLanguageReference Manual:Accellera's Extensions to Verilog, Accellera, Napa,California,2004.
[18]. http://www.soccentral.com.
[19]. ModelSimSEUser'sManual, http://www.mentor.com

Authors Profile

Gi-Yong Song (corresponding author**)** received his Ph.D degree in Computer Engineering from University of Louisiana, Lafayette in 1995. He is now working as a professor at the Department of Semiconductor Engineering, Chungbuk National University, Korea. His research interests includes design and verification of digital systems, high-level synthesis.

Young-Jin Oh received his Ph.D degree in Semiconductor Engineering from Chungbuk National University, Korea, in 2013. His research interests include the areas of digital system design, embedded system design, design and verification of SoC.