ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 4, June 2013

Co-Simulation of SystemC with System Verilog: A VCS Tool Approach

Bhargavkumar Tarpara[1], Ajay Tiwari[2], Chintan Shethiya[3], Rutul Bhatt[4]

PG Student[VLSI], U.V. Patel College of Engineering, Ahmedabad, India[1, 2, 3]

Sr. Technical Associate, eInfochips Training And Research Academy, Ahmedabad, India[4]

Abstract: Due to increased complexity of SoC designs, the importance of design reuse, verification, and debugging increased. Theoretically these concepts seem simple and easy to implement, but there are number of challenges that design and verification team must address while practical implementation. For example, one of the significant barriers to IP reuse is the wide variety of design languages used in IP design and verification. Some of the commonly used languages for design and verification are SystemVerilog, SystemC and conventional HDL languages such as Verilog and VHDL. These languages have their unique strengths which make them more suitable for writing certain portions of a design or IP. But for design to be successful, all of its individual components must communicate with each other which are in the different languages. This paper provides guidelines and ways of how to communicate with SystemVerilog and SystemC. It describes different approaches and provides useful insights to help users to integrates IP blocks in a SystemVerilog environment.

Keywords: SystemC, SystemVerilog, Verification, VCS (Verilog compiler simulator).

I. INTRODUCTION

It is possible to develop RTL and verification environment (VIP) independently using different languages. Since each language has its own strengths and weaknesses, it may be appropriate to write the different components of Verification IP in different languages, for example SystemVerilog and SystemC. Thus, there has to be a method of making these languages interacts. Here we will look at methods of communication between SystemVerilog and SystemC.

You will have question in your mind that why SystemC. So here is the answer, SystemC is a C++ class library with an open-source implementation, and it is used as “glue” to stick together component models when building system-level simulations or software virtual platforms.

SystemC has Verilog-like features such as modules, ports, processes, events, time, and concurrency, so it is conceivable that SystemC could be used in place of an HDL. Indeed, hardware synthesis from SystemC is a fast-growing area. However, the primary use case for SystemC today is to create wrappers for existing behavioral models, which could be plain C/C++, in order to bring them into a concurrent simulation environment.

II. METHODS FOR INTERFACING SYSTEMC AND SYSTEMVERILOG

Here we are going to discuss three methods which are used to interface SystemVerilog and SystemC.

A. *Direct Programming Interface*

SystemVerilog introduces the Direct Programming Interface (DPI) which is an easier way to interface with C, C++, or any other foreign language. The SystemVerilog Direct Programming Interface (DPI) is an interface between SystemVerilog and a foreign programming language such as C language. It allows the designer to easily call C functions from SystemVerilog and call SystemVerilog function from C.

DPI have constructs called “import” and “export” which are used to communicate between SystemVerilog and C. The “import” method is used to call C function from SystemVerilog as if it is native SystemVerilog function and “export” method is used to call SystemVerilog function from C.

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 4, June 2013

The native SystemVerilog DPI does not explicitly support SystemC. However, it is possible to use the standard DPI with SystemC in the sense that SystemC is just another C++ application. The import statement defines that the function uses the DPI interface, and contains a prototype of the function name and arguments. For example:

```
import "DPI" function real sin(real in); // sine function in C math library
```

Above example defines the function name `sin` for use in Verilog code. The data type of the function return is a `real` and the function has one input, which is also a `real` data type. Once this C function name has been imported into Verilog, it can be called the same way as a native Verilog language function. For example:

```
always @(posedge clock) begin
      slope <= sin(angle); // call the "sin" function from C
end
```

DPI is mainly used to pass the data between two domains through function arguments and result. The functions which are imported or exported through DPI interface are assumed to complete their execution instantly at zero simulation time. So the biggest disadvantage of the DPI method is that they cannot call blocking SystemC methods which consume time to operate.

B. *Transaction Level Modelling*

The SystemC community has developed the TLM-1.0 and TLM-2.0 standards for Transaction Level Modelling in the context of architectural exploration and creating so-called *virtual platform* models of a hardware platform for early software execution. Meanwhile, for functional verification, SystemVerilog test benches written to the OVM or VMM standards also make use of transaction-level modelling (TLM) for internal communication. So for communication purpose OVM has adopted the TLM-1.0 standard and latest version of VMM has added internal communication features inspired by TLM-2.0. In this paper, we look to develop an approach to communicate between simple SystemVerilog (without any methodologies like OVM, VVM, UVM) and SystemC, so focus is only how tool can solve the purpose (Synopsys VCS).

C. *Wrapper*

Synopsys VCS tool has inbuilt facility to generate a wrapper to interface Verilog and SystemC. The MXVCS SystemC Co-simulation Interface enables VCS MX and the SystemC modeling environment to work together when simulating a system described in the Verilog, VHDL, and SystemC languages.

Consider a case where we have design in the SystemC and verification environment in SystemVerilog. We cannot directly instantiate SystemC design in SystemVerilog environment. For this we need a wrapper that converts data between SystemVerilog and SystemC. Sysnopsys VCS tool has inbuilt utility called **`syscan`** which can be used to generate a Verilog wrapper for SystemC design. As SystemVerilog is an extension of a Verilog, we can instantiate this Verilog wrapper in SystemVerilog environment.

Following are the steps to co-simulate the design in SystemC with Verilog/SystemVerilog testbench.

1. Wrapper Generation
2. Elaboration
3. Simulation

C.1 *Wrapper generation:*

We can use the `syscan` utility to generate the wrapper of Verilog on SystemC module and interface files for co-simulation. `syscan` utility generate csrc directory in the current directory. The `syscan` utility generate wrapper and interface files in subdirectories in ./csrc directory. The syntax for the `syscan` command line is as follows:

`syscan` *filename*[:*modulename*]

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 4, June 2013

Where: *filename*[:*modulename*] [*filename*[:*modulename*]]* Specifies all the SystemC files in the design. There is no limit to the number of files. Include :*modulename*, for those SystemC modules which are directly instantiated in your Verilog/SystemVerilog design.

Following command generate a Verilog wrapper.

`syscan` adder.cpp:adder

Here adder .cpp contains simple 3-bit adder design which is in SystemC language and adder is module name of SystemC module which is directly called in Verilog/SystemVerilog testbench.

C.2 *Elaboration:*

Here the command is as below

`vcs –sysc –sverilog` `top.sv`

Here we used `–sysc` flag to link our top module which is in SystemVerilog language containing generated wrapper file of adder.cpp. We use `–sverilog` flag because our module is in SystemVerilog language. And top.sv file contains various design modules in Verilog/SystemVerilog & wrapper file instance.

After using this command it generate simv file which is used for simulation.

C.3 *Simulation:*

For simulation we just type below command

`./simv`

It gives simulation result.

III. RESULT

Here as describe above we have taken a simple example of 3 bit adder which is in SystemC. For its verification we have made entire environment in SystemVerilog. So for using above method we just follow below steps.

1. Create whole environment for verification of our design.
2. Create Verilog wrapper of SystemC design.
3. Then take instance of this wrapper in our top module of environment where we need dut.
4. Then elaborate this design which gives executable simulation file simv. Then run this simv file, observe the result. It creates dumped waveform file adder.vcd. So you also observe result in form of waveform with using *DVE (Discovery Visualization Environment*).

Here as per our environment which is in SystemVerilog, generator generates random values of a and b. after that we pass these value through interface in generated wrapper. So our design gives out put which is seen in below Fig 1.

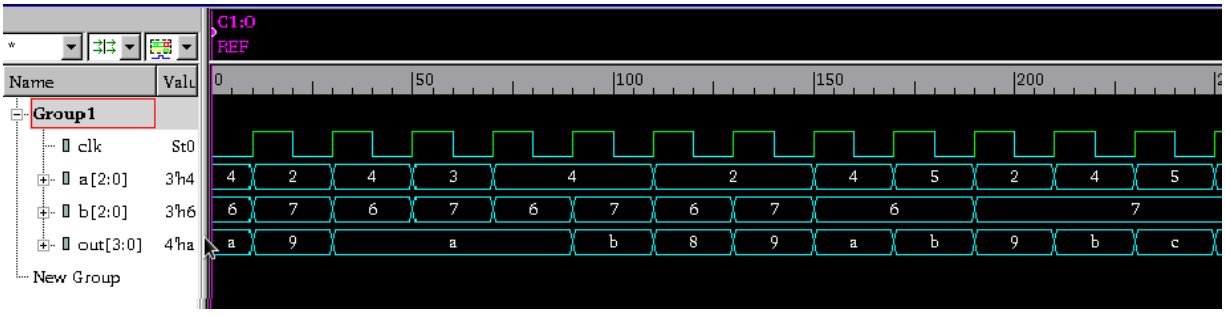

Fig 1- output from DVE Tool

ISSN (Print) : 2320 – 9798
ISSN (Online): 2320 – 9801

International Journal of Innovative Research in Computer and Communication Engineering
Vol. 1, Issue 4, June 2013

IV. CONCLUSION

Interface between SystemC and SystemVerilog can be done by above methods. But as per our research on communication between SystemC and SystemVerilog we have found that wrapper generation method is the easiest tool based method to generate the wrapper for the same.

REFERENCES

[1]. John Aynsley, "SystemVerilog Meets C++: Re-use of Existing C/C++ Models Just Got Easier".
[2]. VCS® MX/VCS MXi™ User Guide Version F-2011.12, December 2011.
[3]. SystemC 2.0.1 Language Reference Manual, Revision 1.0.
[4]. http://www.vmmcentral.org/vmartialarts/category/tlm/ [information regarding Transaction Level Modelling].