

VHDL-Verilog Cosimulation

Ravi Kumar S V
Rama Kowsalya N
Texas Instruments India (Pvt) Ltd.
150/1.Infantry Road
BANGALORE.INDIA -560 001

Abstract

VHDL and Verilog have gained tremendous popularity in today's CAD world that a lot of CAE vendors are coming up with methodologies and tools to operate both of them in a single environment, commonly referred to as cosimulation. Some of the advantages of this cosimulation are to protect ones investment in the existing models, to provide mixed-language simulation and allow interoperability between simulators. This paper describes cosimulation concepts to exploit the existing Verilog/VHDL descriptions to operate in a coupled manner. Cosimulation can be effectively used to link some of the modeling abstractions described in VHDL and Verilog. The intricacies, usefulness and limitations are described.

1 Introduction to Cosimulation

1.1 Interoperability Requirements

The concept of interoperability comes from reusability, where one would like to protect investment that has already been made. As VHDL and verilog HDL share a lot of common characteristics, interoperability between a VHDL system and Verilog HDL system allows designers to have a less turn around time along with other advantages. CAE vendors used this fact to come up with methodologies and tools to operate both VHDL and verilog based designs in the same environment.

The problem of interoperability between VHDL and Verilog HDL is bound by the usage scenario which are significant to designers. The scenario which we envision as being of most immediate importance to the designer are:

1. A Verilog design which must reference VHDL models

2. A VHDL design which must reference Verilog models
3. A VHDL design where Verilog netlists are synthesized

Design groups tend to adopt a single language for development and then define discrete integration checkpoints to deal with models which are externally generated. While this situation may change over time, most of the current needs for VHDL/Verilog interoperability are driven by ASIC library availability in Verilog and system level component availability in VHDL. Cosimulation, in this perspective allows a mixed language simulation in a tightly coupled manner. In this paper we highlight the first two approaches which are the basic cosimulation techniques.

2 Conventional approach for importing designs

The import of existing verilog or VHDL netlist is mostly a translation process. The verilog design netlist is either translated directly to VHDL or synthesized to obtain a VHDL netlist. This method has disadvantages associated with the translation process, also necessitates the use of synthesis tools, and is generally laborious and time consuming and involves manual editions on the resulting netlist. This method also has limitations while mapping during synthesis, being not able to fully make use of VHDL or verilog features. To quote a few, looping construct using generate block is available in VHDL but not in verilog, similarly treatment of all VHDL signal attributes in verilog is difficult.

Hence forth a treatment of cosimulation arises which alleviates the above problems and allows a coupled interaction between VHDL and Verilog systems. Fig 1 describes a typical synthesis flow.

3 Cosimulation

Cosimulation allows users to have transparent access to Verilog models from a VHDL design netlist or VHDL models from a Verilog netlist via a highly efficient approach which interlocks the kernels of the two simulators.

Fig 2a shows "VHDL import", where existing VHDL models can be reused in a verilog netlist. Fig 2b, explains "Verilog import" where existing Verilog models can be utilized in a VHDL design netlist. The interfaces are simulators specific, which would couple both VHDL and Verilog simulators and take care of synchronization and intercommunication between the simulators.

Cosimulation ensures the following:

- To support ASIC Vendors existing certified Verilog HDL-based libraries in CAE vendors VHDL environment via the VHDL tool's unique Model Import feature Verilog Import
- To support ASIC Vendors existing certified VHDL-based libraries in CAE vendors Verilog environment via the Verilog tool's unique Model Import feature VHDL Import

While performing cosimulations, one should remember and adhere to the following guide lines. These guide lines are mainly from the simulators that support cosimulations, to ease the cosimulation, reduce communication overhead and avoid synchronization problems.

If the user has components available in Verilog(VHDL) for some modules and wants to perform simulations on a VHDL(Verilog) design netlist, containing these components, he/she should make sure that

- The top level of design hierarchy must be in VHDL(Verilog)
- Lower levels of design can be in VHDL or verilog
- All design levels beneath a Verilog module must be in Verilog, similarly all the design levels beneath a VHDL module must be in VHDL. Beneath the top-module, no verilog module should instantiate VHDL model or vice versa. It can be viewed as a tree structure, where the subtree of a particular type, say Verilog, cannot have a branch of type VHDL.

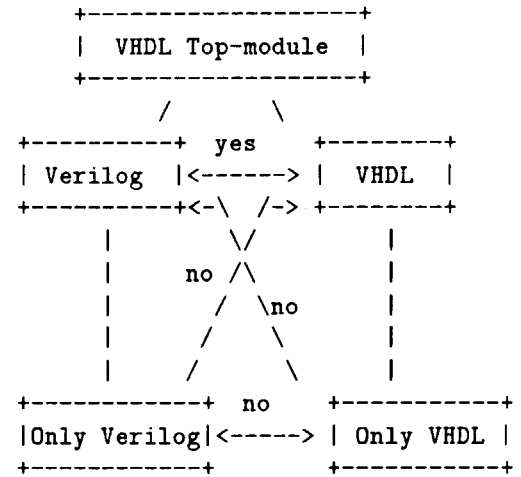


Fig 3. VHDL, Verilog tree structure and the allowed communications

- The VHDL entity name, port names, and port modes must exactly match the Verilog module name, port names and port modes. In general, ports declared in the entity must be either scalars or vectors of IEEE-1164 logic type. VHDL ports of mode in/inout are mapped to verilog input/inout ports respectively. out/buffer ports of VHDL are mapped to out of verilog.
- The boundary module, whose ports directly get connected to VHDL (for Verilog import) or Verilog(for VHDL import), should have no unconnected ports.
- The time scales for both VHDL and Verilog modules should be properly chosen to get accurate timings.

The following sections illustrate "VHDL import" and "Verilog import" with examples from different simulators.

4 Verilog Import

Verilog import has been in demand with the recent trends in design groups to migrate over to VHDL from Verilog.

4.1 Example of Verilog Import in Leapfrog

Leapfrog is VHDL simulator from Cadence Inc. Leapfrog allows to import Verilog models through the use of

'foreign' attribute defined in the IEEE-1076 LRM. Verilog models will have a module declaration corresponding to an entity-architecture pair declared in the VHDL design netlist. Ports must all be of the Verilog logic types as declared in the package XL_STD or the types declared in IEEE-1164 package. XL_STD Cadence package is a VHDL definition of the Verilog logic types.

Semantics associated with any other VHDL language constructs other than the 'foreign' attribute are ignored.

Leapfrog allows to create a vhd shell file, from a verilog model. For example, the following verilog model (functionality not shown) can be easily imported to Leapfrog environment.

verilog model ver_import.v for module 'ver_import'

```
module ver_import ( A , B , C , D , Y );
input A;
input B;
input C;
input D;
output Y;
..
..
endmodule
```

The following command,

```
verilog.exe +vhdl_crshell ver_import.v
```

creates ver_import.vhd file (shown below), through the Verilog compiler directive +vhdl_crshell for the verilog.exe simulator.

shell file ver_import.vhd for Verilog module 'ver_import'

```
library IEEE;
use IEEE.std_logic_1164.all;
entity ver_import is
port(
A : in std_logic := 'U';
B : in std_logic := 'U';
C : in std_logic := 'U';
D : in std_logic := 'U';
Y : out std_logic);
end;
architecture SHELL of ver_import is
attribute foreign of verilog :
architecture is "VERILOG: ver_import ver_import.v";
```

```
begin
end;
```

Once the shell files are created, the design 'TOP' (shown below) can instantiate the vhd shell 'ver_import', which imports 'ver_import' verilog model.

top level VHDL design netlist, which instantiates shell file for ver_import

```
library IEEE;
use IEEE.std_logic_1164.all;
entity TOP;
end;

architecture cosim of TOP is
component ver_import (A : in std_logic := 'U';
B : in std_logic := 'U';
C : in std_logic := 'U';
D : in std_logic := 'U';
Y : out std_logic);
end component;

signal A1,B1,C1,D1,Y1 : in std_logic;
begin
inst: ver_import (A1, B1, C1, D1, Y1)
end;
```

The simulations of TOP will be now taking place both on Leapfrog and Verilog simulators, and are transparent to the user, unless the user likes to debug his designs in Verilog environment, while cosimulation.

4.2 Example of Verilog Import in VSS

VSS (VHDL System simulator) is another VHDL simulator from Synopsys Inc, which supports cosimulation capability, in a different a way.

In VSS, the shell file needs to be created by the user, unlike in Leapfrog where they are generated. The architecture of the shell file will have the attribute "BACKPLANE" indicating to VSS that the architecture is defined in another simulator (in this case Verilog) connected to the VSS backplane. The attribute "VERILOG_FILES_FOR_THIS_ARCHITECTURE" indicates the names of the Verilog files in the subtree. As the verilog models within the subtree don't interact with any of the VHDL models, they don't require any shell files.

An example of a shell file is shown below.

An example of a shell file in VSS

```
library IEEE;
use IEEE.std_logic_1164.all;
library synopsys;
use synopsys.attributes.all;

entity ver_import is
port(
A : in std_logic := 'U';
B : in std_logic := 'U';
C : in std_logic := 'U';
D : in std_logic := 'U';
Y : out std_logic
);
end;

architecture SHELL of ver_import is
attribute BACKPLANE of SHELL : Architecture is
VERILOG;
attribute VERILOG_FILES_FOR_THIS_
ARCHITECTURE of SHELL:
Architecture is "ver_import.v";
begin
end ver_import;
```

The final VHDL netlist TOP will be similar to Leapfrog VHDL netlist, but to run cosimulation, a new Verilog executable needs to be build, to communicate with the VSS with verilog interface.

The approaches for leapfrog and VSS are different for the implementation of interfaces, though they share the common concept of shell file creation. VSS uses synopsys attributes to define the backplane, where as leapfrog makes use of IEEE-1076 foreign attribute. Hence, the same shell files cannot be shared.

5 VHDL Import

The use of "VHDL Import" is in equal demand, owing to the applications requiring VHDL behavioral, RTL level, gate level descriptions in the verilog based designs.

The example given below is importing VHDL models in Leapfrog environment.

Since Verilog doesn't have a pre-defined attribute such as "foreign" the correspondence between Verilog

and VHDL modules is done using an attribute notation as below:

An example of VHDL Import in Verilog

```
module vhdImport (A,B,C,D,Y)
(integer simulator = "Leapfrog"; // Simulator name
integer model = "library.vhdImport:cosim";*)
// library.entity:architecture
input A;
input B;
input C;
input D;
output Y;
endmodule
```

The shell model contains only port definitions. The attribute "simulator" defines the name of the VHDL simulator. The attribute "model" is used to specify the VHDL library, entity and architecture of VHDL model.

The verilog top level netlist can then be simulated, with leapfrog interface using verilog simulator.

6 Cosimulation Pros and Cons

The concept of cosimulation has got a lot of advantages.

- A shorter design turnaround for ASIC designers who chose the ASIC Vendor's technologies in conjunction with using CAE Vendor's existing VHDL/Verilog solution
- Allow mixed language simulations
- Protect one's investment
- Avoid entirely rewriting existing simulator or writing a new simulator
- It's nearly impossible to map all the semantics of VHDL onto an existing system
- As both verilog and vhdI follow the same standard back annotation flow based on OVI (Open Verilog International) sdf (standard delay file), no extra processing is required in the design flows. Though, cosimulation allows the simulations to be performed in a single environment, it has the following problems.

- Communication overhead between the simulators
- Slow, the more the transactions between VHDL, verilog models, the slower the simulation speed.
- Problems with timing and synchronization
- Emerging standards like VITAL may try to eliminate cosimulation
- Extra burden of building up interfaces
- Is not cost-effective
- All data types not allowed in cosimulation environment
For example real types are not allowed in leapfrog VHDL import.
- Unresolved signal strength mappings
For example, VHDL signals are based on 1164 MVL9 logic, where as verilog signals can be defined as compact or non-compact. Compact values contain only logic values (0,1,X or Z) without any strength information. Non-compact values contain both logic value and strength(Strong 1,Weak 0, Pull 1.63 X, etc...). Not all the VHDL simulators which support cosimulation allow all the verilog strength values.

7 Conclusions

This paper described cosimulation concepts to exploit the existing Verilog/VHDL descriptions to operate in a coupled manner. Until VHDL becomes a true multi-level design language with all the tools support, cosimulation methodology will persist in the design environment.

References

1. Synopsys Methodology Notes , Premier Issue September 1991
2. VSS Expert Interface Manual
3. Leapfrog Reference Manual