

Standard Verilog-VHDL Interoperability

Victor Berman

Cadence Design Systems, Inc.

1.0 Abstract

During the last few years HDLs have become the driver behind the move to top down design in the electronic design industry. Two HDLs, VHDL and Verilog HDL have become the dominant de facto industry standard HDLs. Since the industry has made a huge investment in both HDLs and there is every indication that each will retain significant market share for the foreseeable future, it is critical that there exist a standard methodology for interoperability between the two languages. This paper describes the relevant issues for interoperability and suggests solutions where they currently exist. It further summarizes the work which needs to be done for a complete solution and the groups who are involved in achieving this goal. Please note that the emphasis in this paper is on simulation since the semantics of the two languages are specified only for that discipline. While the importance of other disciplines such as logic synthesis cannot be underestimated in the top down design process, the lack of standard language semantics makes general analysis problematic.

2.0 Interoperability Requirements

The problem of interoperability between VHDL and Verilog HDL is bounded by the usage scenarios which are significant to designers. The general problem of unconstrained design development in both languages is troublesome but fortunately rarely occurs in industry. The scenarios which we envision as being of most immediate importance to the designer are:

- A Verilog design which must reference VHDL models

- A VHDL design which must reference Verilog models
- 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 were externally generated. While this situation may change over time most current needs for VHDL/Verilog interoperability are driven by ASIC library availability in Verilog and system level component availability in VHDL. The specification of high level designs and test benches in VHDL which require ASIC library components available in Verilog drive the requirement for verification of synthesized Verilog netlist in a VHDL simulation environment. Focusing on these scenarios allows the problem to be bounded in a manageable way and we divide it into two major segments

- Verilog Import
- VHDL Import

2.1 Verilog Import

Two usage scenarios must be supported. In the first one, the user is incrementally synthesizing Verilog gate-level netlists from the VHDL model input and desires verification of the synthesized netlist with the unsynthesized VHDL; in the second one, the user desires to use a Verilog model as an element of a VHDL design.

In the first scenario, a logic synthesis tool is used to incrementally synthesize portions of a large design into Verilog.

Progressively larger and larger portions of the design will be synthesized. The user will wish to verify the accuracy of the synthesis output through the use of simulation of regression tests using the VHDL design with Verilog gate-level net lists synthesized from progressively more of the VHDL design. The use of Verilog within this process should be hidden as much as possible to the user in this scenario since the user may have little or no knowledge of Verilog. Thus, the VHDL debugger should be able to display and manipulate the synthesized Verilog netlist as if it were a VHDL netlist.

In the second scenario, a user will wish to utilize an existing Verilog design as an element in their VHDL design or test bench. The user in this case will wish to treat the Verilog model as relatively fixed. The user should not be required to know how Verilog works, but may want some limited visibility into the model to display important model data that may be known to exist within the model.

2.2 VHDL Import

The scenario of primary importance to potential users of this functionality is the ability to use externally provided VHDL models as part of a Verilog model simulation. The user in this scenario will, in general, not be knowledgeable of VHDL. It is expected that the VHDL models will be large, fairly high level models of components such as CPUs and that the purpose of the simulation will be to assess the behavior of the Verilog models within the context of a larger system. In this scenario, the VHDL models will be treated as “library modules” and will change infrequently, Verilog models will be the elements under development and will change rather more frequently and simulations with different stimuli will be common. The end user will be unconcerned about VHDL model implementation particulars, but may want visibility to important VHDL model data such as register values. The user will want to access any such values in the same manner as access to other Verilog data is provided.

3.0 Proposed Approach

The general approach taken is to define an interface or encapsulation layer between the unlike language models so that issues such as synchronization, signal resolution, and timing concepts can be handled systematically. Due to the differences in the languages, slightly different strate-

gies are used for the two types of import. They are described below.

3.1 Verilog Import Approach

VHDL models may import Verilog models into a VHDL design through the use of the `foreign` attribute defined in the IEE_1076/93 Verilog models to be incorporated into a VHDL design will have a corresponding entity-architecture pair declared in VHDL Ports must all be of the Verilog logic types as declared in the package `XL_STD` (this package is a VHDL definition of the Verilog logic types), or the types declared in the IEEE 1164 package. In the latter case, conversions between the Verilog type and the IEEE 1164 type will be performed according to mapping rules defined later. It is assumed that tools which implement this approach will have this conversion built in for efficiency. The architecture body must contain an attribute specification for the `foreign` attribute. Semantics associated with any other VHDL language constructs within the entity declaration or architecture body are ignored. The Verilog module name within the Verilog input file being imported must be the same as the VHDL entity name.

An example of a Verilog module and its corresponding VHDL shell are shown below:

```
module vxl_cpu (MemData, MemGrant,
               IOBusy, MemRw, MemReq, IOReq,
               MemAddr, IOAddr, IOOper))

    inout [31:0] MemData;
    input MemGrant, IOBusy;
    output MemRW, MemReq, IOReq;
    output [16:0] MemAddr, IOAddr;
    output [1:0] IOOper;

    ...
endmodule
```

Figure 1: Example Verilog input file module.v.

```

library IEEE;
use IEEE.std_logic_1164.all;
entity vxl_cpu is
port( MemData :
inout std_logic_vector(31 downto 0);
    MemGrant, IOBusy : std_logic;
    MemRw, MemReq, IOReq : out
std_logic;
    MemAddr, IOAddr : out std_log-
ic_vec tor(16 downto 0);
    IOOper : out std_logic_vector(1
downto 0)
    )
end ;

architecture verilog of vxl_cpu is
attribute foreign of verilog:
architecture is "VERILOG: vxl_cpu
model.v";
begin
end

```

Figure 2: Example VHDL design file module.vhd created for Verilog input file of Figure 1.

3.2 VHDL Import Approach

Since Verilog does not have a pre-defined attribute such as Foreign the correspondence between Verilog and VHDL modules is done by convention using an attribute notation as shown below:

```

module vhdl_part
(inA, in8, outA, outB, inout8)

(* integer simulator = "Leapfrog";
integer model=
"mylib.myentity:myarchitecture";*)

    input inA;
    input [7:0] in8;
    output outA, outB;
    inout [7:0] inout8;
endmodule

```

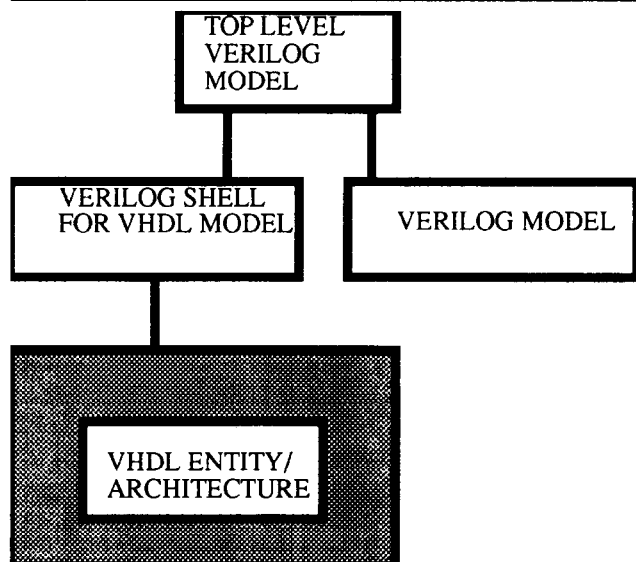
The shell model contains only port definitions. The attribute "simulator" defines the name of the VHDL simulator. The attribute "model" is used to specify the name of the VHDL library, entity, and architecture of your VHDL model.

Ports declared in the root entity must be either scalars or vectors of the IEEE 1164 standard logic type, or of the Verilog logic type as defined in the package VHDL_XL. Support for the Verilog logic type is of secondary importance, but is provided so that VHDL model suppliers who wish to use VHDL port types other than IEEE 1164 may do so without the need for double data conversions.

Port modes are mapped as follows: VHDL ports of mode *in* are mapped to Verilog *input* ports. VHDL ports of mode *inout* are mapped to Verilog *inout* ports. VHDL ports of modes *out* or *buffer* are mapped to Verilog *output* ports. Verilog ports are declared in the same order as they appear in the corresponding VHDL.

Port names and ranges in the generated Verilog are identical, except for any 'extended' identifiers used in the VHDL. VHDL '92 extended identifiers will be mapped to Verilog extended identifier syntax, where each illegal character appearing in the VHDL extended identifier is preceded by a backslash and the backslashes surrounding the extended identifier are removed.

FIGURE 3. Model Organization for VHDL Import



4.0 - SIGNAL RESOLUTION

Since Verilog and VHDL define signals differently, defining signal resolution is an essential part of the interoperability definition. Verilog, signals can be defined as either compact or non-compact. Compact values contain just logically values and no strength information (1, 0, X, or Z). These values are defined as being strong in strength. Non-compact values contain both a logical value and strength information (St1, We0, Pu1, 63X, etc.).

The interface between Verilog and the VHDL will only be defined here for two types of VHDL signals. These are the IEEE std_logic and vlbit signal types.

The IEEE std_logic has 9 states: U, X, 0, 1, Z, W, L, H, and -. "U" (un-initialized) is an unknown. "X" (strong unknown), "0" (strong zero), "1" (strong one), and "Z" (high-impedance) are similar to Verilog-XL values. The "W" (weak unknown), "L" (weak zero), and "H" (weak one) are weak states. The "-" state is also unknown.

The vlbit has just 4 states: X, Z, 0, 1. The explanation for these values are the same as for std_logic. The proposed mapping between the two systems is shown below:

std_logic	non-compact	compact
U	StX	X
X	StX	X
0	St0	0
1	St1	1
Z	HiZ	Z
W	WeX	X *
L	We0	0 *
H	We1	1 *
-	St	X
vlbit	non-compact	compact
X	StX	X
Z	HiZ	Z
0	St0	0
1	St1	1

* Weak strength is lost in these cases.

Mapping the Verilog compact values into VHDL values is straightforward as show below:

compact	std_logic	vlbit
0	0	0
1	1	1
X	X	X
Z	Z	Z

In mapping the Verilog non-compact values into VHDL vlbit, the strength information is ignored and just the logical value is retained, as shown in the following chart:

non-compact	vlbit
any 0	0
any 1	1
any X	X
only HiZ	Z

Mapping the Verilog non-compact values to VHDL std_logic is more complex. Verilog strengths of Supply, Strong, and Pull are mapped to strong std_logic values. Verilog-XL strengths of Large, Weak, Medium, and Small are mapped to weak std_logic values. The HiZ strength in Verilog-XL is mapped to a "Z" std_logic value. A summary of this information is shown below:

non-compact strength	std_logic strength
Supply (7)	strong
Strong (6)	strong
Pull (5)	strong
Large (4)	weak
Weak (3)	weak
Medium (2)	weak
Small (1)	weak
HiZZ	

When Verilog values have ambiguous strength, the mapping is less intuitive. In Verilog-XL, a net may have a value of 630 (logical value of 0 with a strength between

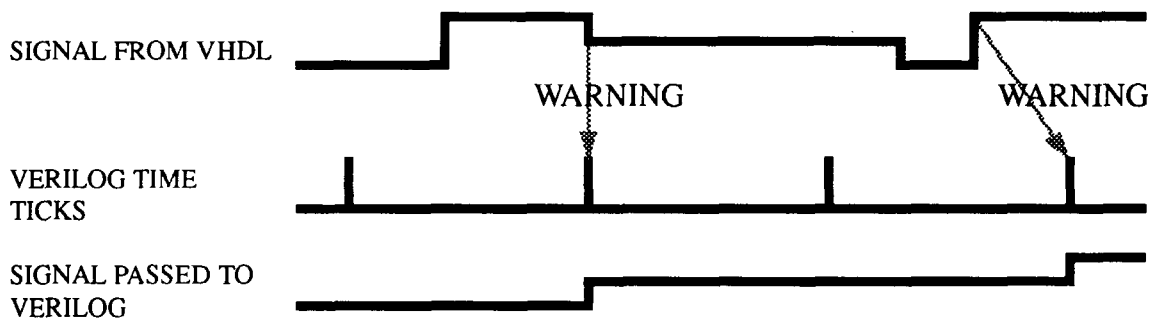
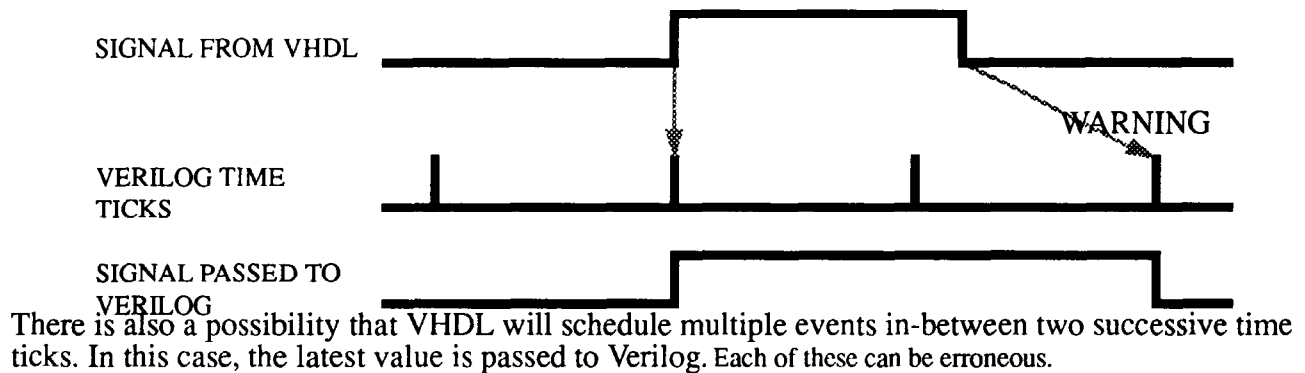
strong and weak). This value could be mapped as either a strong 0 or a weak 0. The mapping uses the maximum strength. In this case the maximum strength is strong, so therefore the result will be a strong 0 in VHDL

5.0 TIME RESOLUTION

The Verilog timescale is used to determine the resolution for communication between Verilog-XL and Leapfrog. VHDL simulation does not have any notion of “smallest simulation time unit” other than what is defined in the VHDL language (femto-seconds).

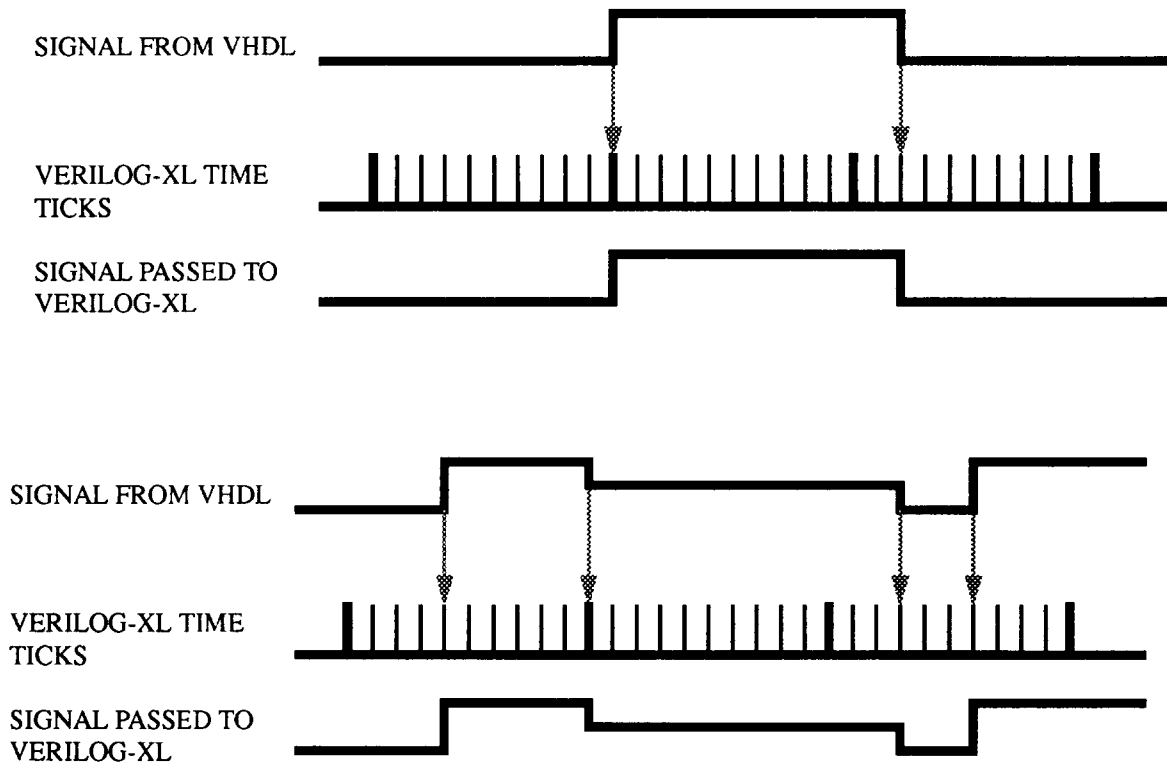
The two possibilities are that Verilog has a time resolution smaller than that used in the VHDL model or that the VHDL model may have its time resolution smaller than that used by Verilog. In the first case, the VHDL simulator does not have any problems if it receives Verilog interrupts more than it sees events in a VHDL model.

In the following discussion, “time tick” means the instant when a time unit ends and another time unit begins. There is a problem where the VHDL activity occurs between the Verilog simulation time ticks. In this case, the events from VHDL will be recognized by Verilog at the next simulation time tick.



If there is a problem with events from VHDL occurring too rapidly for the Verilog model, increasing the Verilog time resolution is a remedy. This is done by modifying a `'timescale` directive in one of the Verilog models.

The time resolution for the Verilog simulation could be increased by a factor of 10. Doing this with the previous two examples, would generate the following waveforms:



By increasing the time resolution within Verilog, the signals between the simulators are passed at more accurate times. There is a trade-off with making this adjustment. The memory usage may be increased and the simulation's duration may increase.

If the Verilog models do not contain any `'timescale` directive, then a default timescale of `1SEC/1SEC` is used. Naturally this will cause incorrect results. Therefore the `'timescale` directive must be used when importing models into Verilog.

In such cases a resolution warning message is similar to the following should be issued so that the modeler understands the situation:

Warning! Event(s) happened in a finer resolution.

6.0 SYNCHRONIZATION

Since the model for synchronization of time and event processing in VHDL and Verilog is somewhat different, it is important for the modeler to understand the strategy proposed. Signals which pass between Verilog and VHDL are updated after both simulators have completed all of their currently active events. In response, the Verilog and VHDL simulators will process these incoming events. It is possible for the new events to cause more boundary signals to change. Once all of the events are processed, the boundary signal changes are exchanged again. Once again the simulations will be run in response to the signal changes. This process is repeated until ALL events are processed for the current time unit.

This synchronization strategy has a side effect on the \$monitor task output. When Verilog completes all of its active events, it does not know if it will be called back due to the VHDL simulator passing changes on their port signals. Therefore, the \$monitor statement outputs a line when Verilog completes all of its activity. However, the VHDL simulator may pass back some more data and this can cause some of the monitored signals to change. In which case, the \$monitor statement will produce another line of data. This can produce results similar to the following:

```
450      clk=1 enable=0 data=zz
500      clk=1 enable=1 data=zz
500      clk=1 enable=1 data=52
```

At time 500, Verilog-XL evaluated the signal “enable” and assigned its new value of 1. Verilog completed its evaluations and printed its monitor statement. The signal change on “enable” was passed to the VHDL model. The VHDL model responded by calculating a new value for the data bus. The new value was passed back to Verilog-XL. Since Verilog-XL was also monitoring the data bus, it produces a new output line when it changed.

Another issue concerning synchronization deals with event order dependencies. The events from the VHDL model will always change after all of the Verilog events are processed.

```
my_vhdl    vhdl1(... , data , ...);
my_verilog vlog1(... , clk , ...);
always @(posedge clk) out = data;
```

In the model above, if both “clk” and “data” change during the same time slice, the old value of “data” will always be clocked. If the VHDL model was modelled in Verilog, then it is possible for the new value of “data” to be clocked.

7.0 Conclusions and Future Work

The ideas presented in this paper give a starting point for a standard method of interoperability between VHDL and Verilog. I would like to stress the importance of gaining industry support for such a common methodology before the inevitable market forces lead to disparate proprietary implementation driven approaches which hamper model interoperability. A very significant amount of work in this area has already been done at Cadence to solve the technical problems involved in modeling and simulation for mixed language systems including VHDL and Verilog. It is the intention of Cadence to make as much of this material public as is needed to solve this important problem for industry.

Currently, a working group under the auspices of the IEEE DASC has begun to investigate the issues involved in VHDL Verilog interoperability. We plan to work closely with this group and share our findings with them in the interest of quickly arriving at a useful standard in this area. We feel that a phased approach to the problem has the best chance for early success and therefore have suggested in this paper that the problems which are confronting design-

ers today be solved first before the more general problems of multi-language design be tackled.

We are currently prototyping the ideas discussed in this paper with Cadence's VHDL and Verilog simulators and look forward to reporting on the results of these efforts in the near future.

8.0 References

- [1] "IEEE Standard VHDL Language Reference Manual IEEE Std 1076-1987", 1988, The Institute of Electrical and Electronics Engineers, Inc. New York, NY.
- [2] "IEEE Draft Standard VHDL Language Reference Manual IEEE Std 1076-1992B", 1993, The Institute of Electrical and Electronics Engineers, Inc. New York, NY.
- [3] "Verilog Hardware Description Language Reference Manual, Version 2.0", March 1993, Open Verilog International, San Jose, CA.