Interoperability of Verilog/VHDL Procedural Language Interfaces to build a mixed language GUI

Françoise Martinolle, CharlesDawson, Debra Corlette, Mike Floyd

Cadence Design Systems, Inc.

1. The mixed language design and verification problem

1.1 Statement of the problem

Nowadays, designs become more and more complex and are often developed by several teams that may use different Hardware Description Languages. Mixed Verilog/VHDL designs are proliferating because teams choose different languages or because they are importing intellectual property libraries of VHDL or Verilog cells. The authors of paper [1] which compares Verilog and VHDL semantics conclude that there is enough overlap between the two languages for tools vendors to consider building bilingual tools such as compilers, simulators etc… In this paper we analyze the requirements for a mixed language GUI and explain how the compatibility and interoperability of the standard Verilog and VHDL language procedural interfaces was key to build SimVision, Cadence NCSimulator^{TM} graphical debugging and verification environment. The principles highlighted in this paper can also be applied to other kinds of applications such as Verilog/VHDL co-simulation, testbench generation etc… In the following, we will focus on the problems inherent to a Verilog/VHDL GUI development.

In a mixed language design, the language boundaries are very clearly defined and language construct mixing exist only at the instance level. VHDL can be instantiated from within a Verilog module and Verilog modules can appear as sub-instances of a VHDL instance. Usually, models are integrated by the generation of a shell or by direct instantiation. The shell methodology seems to be the preferred way for importing foreign models. A shell is the interface to the model and is written in the HDL language of the instantiating parent scope. A shell does not add an extra level of design hierarchy; it simply places the behavior of the imported model in the design. The shell bridges the differences in the language syntax and creates correspondence between the language constructs. All communication issues between the languages are localized in the shell. In the case where a shell interfaces a VHDL description to a Verilog environment, the shell is a Verilog compliant module definition equivalent to the VHDL entity description. The shell only specifies the interface (ports, parameters) while the behavior is left expressed in VHDL. This constitutes a typical mixed language use model. In this model, the challenges for a debug and verification environment are:
1) to have the capability to refer to, select any object, whether it belongs to a Verilog or VHDL instance,
2) to be able to navigate through the hierarchy and follow net connections in either language domains,
3) to be able to set, get the value of any object in the design,
4) to have a language sensitive graphical debug context (menus, messages) which adapts to the user actions.

1.2 A need for a mixed language procedural interface

In such a context, it is important to have a software layer between the GUI and the simulator or elaborator that understands this use model in order to provide a clear, simple view of the design. For example, shells should not create an extra level of hierarchy and should be recognized as means of interfacing foreign models. Port connections should not show the artificial shell ports. Object values of any VHDL or Verilog type should be representable. This stresses the point that it is critical to define a Procedural Interface that can access static and runtime information independently of the language. It is desirable:
1) that a mixed language design for which artifacts of integration can be hidden, is presented to the user,

2) to have a common methodology to access information or to interact with the tool independently of the language,

3) to provide a smooth crossing of language domain,

4) that the unified design of the interfaces does not prevent access to HDL specific information.

Therefore the PLI software layer should provide:

1) a *simple post-elaborated view* of the mixed HDL design,

2) a *cohesive* and unified way of getting language information *independently of the language,*

3) a clear definition of the language boundaries and

behavior of the interface functions at the boundaries,

4) *complete support* of the HDL specific semantical information.

At Cadence, we had taken these requirements into account and designed a Procedural Language Interface that can handle both languages. For that, we chose to build a VHDL Procedural Interface highly compatible with the standard Verilog Procedural Interface (VPI) [2], [4]. The VHDL Procedural Interface that we call VHPI is based on the same concepts as the Verilog Procedural Interface, shares the same architecture and software components [3], [6], [7]. From the user perspective, there is really only one interface to learn since the mechanisms for accessing or modifying HDL data are the same. A formal graphical information model expresses specific language data access. Interface operations are defined to work off the underlying information model. The interoperability of the Procedural Interfaces is expressed by the intersection of the language specific information models. It relies on the definition of the interface semantics at the language boundaries and on the use of the same access mechanisms and Procedural Interface functions regardless of the language. This results in a very powerful interface that provides a smooth crossing of the language domains. The next section covers the advantages in using the VPI and VHPI interfaces to develop a mixed language GUI. Details about the VPI and VHPI interfaces can be found in [3], [4].

2. SimVision, a VHPI and VPI application

Our simulator GUI acquires all of its instantiated design information, source information, value information, and connectivity information from VPI and VHPI. VPI was initially chosen because of the fact that an existing standard interface provided all of the information required to produce a graphical debug environment for Verilog. By using a standard interface, additional manpower was not required to duplicate existing functionality.

Both VPI and VHPI provide an information model and a small, uniform, well-defined interface for obtaining this data. The use of abstract handles to objects and consistent mechanisms for accessing both handles and information about the handles allowed for a clean, homogeneous architectural boundary to be implemented between the GUI and VPI/VHPI. The use of this clean architectural layer is important. The consistency between the way VHDL and Verilog data is described and accessed by the interfaces allows this layer to be easily extended to cover both languages. The traversal of objects and hierarchy in both VPI and VHPI is identical. This allows for maximum code reuse and minimal code additions in order for hierarchy traversal to function for both languages. Since the information models and access methods are nearly identical, rapid development of a mixed language user interface could take place. For example, the Navigator window that displays the design hierarchy was extended to handle VHDL and mixed language with one-man week of effort. The amount of code changes for access to design data was minimal and straightforward. Most of the code involved was in properly displaying objects from the different languages. The fact that VPI and VHPI navigation functions return an abstract handle regardless of the language allows for a mixed language design traversal to happen. For each handle returned, the language of the handle can be obtained by looking at the handle type. Once the language is obtained, VPI or VHPI calls for more specific information about the object can be made. This allows the GUI to recognize a Verilog Module that is instantiated in a VHDL instance and to adjust accordingly when the user scopes into this module. When a language boundary is crossed, the GUI automatically modifies the menus for Verilog or VHDL specific terminology and the scope change is a transparent operation to the user.

3. Conclusion

In this paper we have shown the reasons behind the development and use of compatible standard interfaces for developing a mixed language GUI. We have determined VPI and VHPI requirements and extensions to handle mixed language applications. This can enable third party tool developers or internal CAD groups to easily write code which can operate on a mixed language design. We are confident that VHPI and VPI compatibility is a very promising aspect for mixed language application development. The VHPI Cadence interface is currently used for input to the specification of a standard VHDL Procedural Interface under the IEEE standards organization.

References:

[1] VIUF spring 1996, "Verilog: Dialect of VHDL " John Willis, Gabe Moretti, Paul Menchini.

[2] IEEE Std 1364-1995, IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language PLI TF and ACC interfaces Chapters 17-21, VPI interface Chapters 22 and 23.

[3] IVC/VIUF 1998 "A Procedural Interface for VHDL and its Typical Applications", March 16-18 1998, Françoise Martinolle, Adam Sherer.

[4] IVC 1996, "The Verilog Procedural Interface for the Verilog Hardware Description Language" Charles Dawson, Sathyam Pattanam, David Roberts.

[5] IVC 1995, "INCA: A Next-Generation Architecture for Simulation " Jay Lawrence, Cary Ussery.

[6] VIUF fall 1996 "VHPI, A Programming Language Interface for VHDL" Doug Dunlop.

[7] BMAS 1997 IEEE/VIUF International Workshop on Behavioral Modeling and Simulation1, "VHPI a VHDL Procedural Interface" Françoise Martinolle, Sathyam Pattanam, Debra Corlette.