

### Highlights

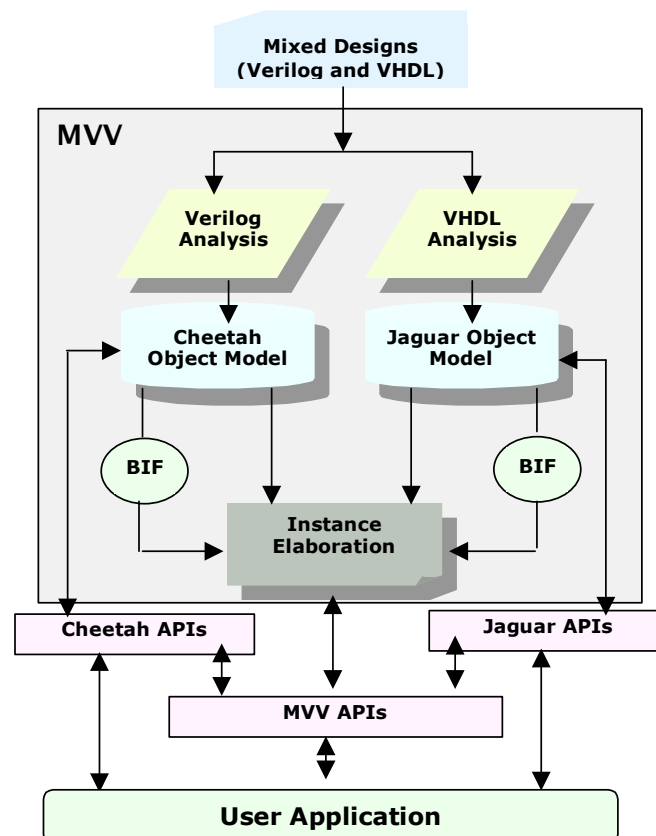
- Mixed design elaboration involving System Verilog and VHDL
- Well-defined, complete set of API functions
- Support for Partial Elaboration
- Support for incomplete designs through black box option
- API functions to propagate defparams across language boundaries
- API functions to support full design elaboration through elaboration, copying instance master elaborated by overriding parameter/generic values, binding instances to a copied master across language boundaries
- API functions to flatten a design having generate blocks and/or array of instances
- Backed by Interra's field-proven expertise in developing HDL analyzers for VHDL and Verilog

Addressing the needs of ASIC/Soc professionals and EDA tool developers who need to verify and simulate designs in mixed Verilog and VHDL, Interra offers MVV, a mixed language Digital Design Elaborator. Targeted as a customizable front-end for System Verilog and VHDL based mixed design applications such as simulation and synthesis, MVV is compliant with industry standard simulators and other tools.

Available as library of "C" APIs, MVV can be seamlessly integrated with "C" or "C++" applications and used as a front-end to various EDA applications.

MVV analyzes mixed designs and elaborates cross-HDL instances. MVV is built over Interra's popular Verilog and VHDL analyzers: Cheetah and Jaguar. MVV users, therefore, have the dual advantage. They can use Cheetah and Jaguar APIs to analyze and modify designs, and use MVV APIs to verify cross-HDL binding of instances. Enabling the user to manage Verilog and VHDL libraries, optimize elaboration, and access information from elaborated masters, MVV's APIs are complete, function-rich, and intuitive.

MVV is available on Solaris, HP-Unix, Linux, and Windows NT platforms.



# The MVV Features

## Complete Language Support

MVV is built over Cheetah and Jaguar analyzers. Cheetah completely supports System Verilog IEEE 1800-2005. Cheetah is backward compatible with IEEE 1364-1995, IEEE 1364-2005, OVI 2.0, and simple subset of IEEE 1850, V1.1 and V1.01.

Jaguar completely supports VHDL 93. Jaguar is backward compatible with VHDL 87.

Cheetah and Jaguar analyze the syntax and semantics of mixed designs. In addition, MVV provides a facility to elaborate mixed digital designs covering different aspects of System Verilog and VHDL.

## C Procedural Interface

MVV's 'C' procedural interface provides the flexibility to integrate MVV with 'C' and 'C++' applications. The intuitive API function names facilitate a short learning curve.

## Complete Set of API Functions

MVV has specific APIs to elaborate mixed designs. These APIs enable the user to find out master of an instance across the language boundary. MVV APIs work on both Cheetah and Jaguar Object Model. Therefore, one can use MVV APIs in addition to Cheetah or Jaguar APIs in their applications. While the Cheetah and Jaguar APIs let users work on Verilog and VHDL designs, respectively, MVV enables the user to elaborate mixed designs. Using MVV APIs, one can work in two modes: the in-memory mode and the dump mode.

## Partial Elaboration

MVV APIs provide the facility for elaborating a master design unit only when required. Further, the APIs can be used to un-elaborate the master design unit and re-elaborate it with a different set of parameter values. With this feature, MVV users can effectively keep memory requirement of the application low.

## Full Elaboration

MVV enables a full elaboration of mixed designs. A master design unit can be elaborated, copied, and instances can be bound to the copied master design unit across language boundaries. API functions are also available to flatten a design that has generate blocks and array of instances. MVV supports the propagation of defparam values and resolution of scope variables across language boundaries. MVV also supports SV Bind construct elaboration where the target scope is both SV and VHDL in full elaboration flow.

## Black Box Elaboration

MVV allows elaboration of an instance whose master is not defined. MVV creates a dummy design unit (Verilog module if the instance language is Verilog or entity if the instance language is VHDL) with a list of input and output ports inferred through corresponding instantiations.

## Customization of Errors

The API functions enable the user to customize the messages to suit application-specific needs. One can also register an error message handler to meet customized needs. In addition, one can use the API functions to suppress error messages and warning messages or change the severity of messages.