INSERT:

35.13 VPI Backwards-Compatibility Features and Limitations

The VPI data model has evolved substantially over many previous versions in order to keep up with corresponding features of the Verilog HDL. Substantial efforts have been made to maintain backwards-compatibility with prior versions whenever possible. However, some critical incompatible changes were needed that could not be avoided. This section identifies those incompatibilities and provides a way for older affected applications to continue to run in newer VPI environments, with some important restrictions.

35.13.1 VPI Incompatibilities With Other Standard Versions

The following table summarizes the VPI incompatibilities with prior IEEE standard versions.

Table 1-1: Summary of VPI Incompatibilities Across Standard Versions

Incompatibility	**1364**			**1800**
See detailed descriptions below	**1995**	**2001**	**2005**	**2005**
1) **vpiMemory** exists as an object	Y	D	N	N
2) **vpiMemoryWord** exists as an object	Y	D	N	N
3) **vpiIntegerVar** and **vpiTimeVar** can be arrays	Y	Y	Y	N
4) **vpiRealVar** can be an array	N	Y	Y	N
5) **vpiVariables** iterations include **vpiReg** and **vpiRegArray** objects	N	N	N	Y
6) **vpiReg** iterations on **vpiRegArray** can result in non-**vpiReg** objects	N	N	N	Y
7) **vpiNet** iterations on scopes and modules include **vpiNetArray** objects	N	N	N	Y
8) **vpiNet** iterations on **vpiNetArray** can result in non-**vpiNet** objects	N	N	N	Y
9) **vpiMultiArray** property available	N	Y	D	N

Table Key:

Y = Behavior, function or object present in that version

D = Behavior, function or object deprecated (present but use discouraged) in that version

N = Behavior, function of object no longer present in that version

For the above table and details below, the types **vpiReg** and **vpiRegArray** are the same as **vpiLogicVar** and **vpiArrayVar**, respectively, as shown in the 1800 VPI data model (see 36.14 detail ‘s’).

Incompatibility Details:

1) **vpiMemory** exists as an object:

Unpacked unidimensional reg arrays were exclusively characterized as **vpiMemory** objects in 1364-1995, and later deprecated in 1364-2001. This object type was replaced by **vpiRegArray** in1364-2005, leaving **vpiMemory** allowed as only a one-to-many transition for 1364-2005 and 1800-2005 (see section 36.16). Note that 1364-2001 allowed *either* **vpiMemory** or **vpiRegArray** types to represent unpacked unidimensional arrays of **vpiReg** objects.

2) **vpiMemoryWord** exists as an object:

Elements of unpacked unidimensional reg arrays were exclusively characterized as **vpiMemoryWord** objects in 1364-1995, and later deprecated in 1364-2001. This object type was replaced by **vpiReg** in 1364-2005, leaving **vpiMemoryWord** allowed only as an iterator for 1364-2005 and 1800-2005 (see section 36.16). Note that 1364-2001 allowed *either* **vpiMemoryWord** or **vpiReg** types to represent elements of unpacked unidimensional arrays of **vpiReg** objects.

3) **vpiIntegerVar** and **vpiTimeVar** can be arrays

vpiIntegerVar and **vpiTimeVar** objects could represent unpacked arrays instead of simple variables in all 1364 standards. In 1800-2005 these array types are always represented as **vpiRegArray** objects, and **vpiIntegerVar** and **vpiTimeVar** objects are always non-array variables (see section 36.14).

4) **vpiRealVar** can be an array

This object type was allowed to represent an unpacked array of such variables in 1364-2001 and 1364-2005 standards (**vpiRealVar** arrays were not yet allowed in 1364-1995). In 1800-2005, these are now exclusively represented as **vpiRegArray** objects (see section 36.14).

5) **vpiVariables** iterations include **vpiReg** and **vpiRegArray** objects

In all 1364 standards, **vpiReg** and **vpiRegArray** objects were excluded from **vpiVariables** iterations, and only accessed instead by iterations on **vpiReg** (from a scope or **vpiRegArray**), or **vpiRegArray** (from a scope), respectively. In 1800-2005, they are both included in **vpiVariables** iterations (see section 36.14).

6) **vpiReg** iterations on **vpiRegArray** can result in non-**vpiReg** objects

This is a consequence of **vpiRegArray** objects being used to represent unpacked arrays of non-**vpiReg** elements in 1800-2005 (see section 36.14). **vpiReg** iterations on these array objects can retrieve array elements that are of type **vpiIntegerVar** or **vpiTimeVar** for example, which is not expected in standards 1364-2001 and 1364-2005.

7) **vpiNet** iterations on scopes and modules include **vpiNetArray** objects

In all 1364 standards, **vpiNetArray** objects were excluded from **vpiNet** iterations on scopes and modules, and were only accessed by iterations on **vpiNetArray** (from a scope or module). In 1800-2005, they are included in **vpiNet** iterations (see section 36.13).

8) **vpiNet** iterations on **vpiNetArray** can result in non-**vpiNet** objects

This is a consequence of **vpiNetArray** objects being used to represent net arrays of non-**vpiNet** elements in 1800-2005 (see section 36.13). **vpiNet** iterations on these net array objects can retrieve net elements of type **vpiIntegerNet** or **vpiTimeNet** for example, which is not expected in standards 1364-2001 and 1364-2005.

9) **vpiMultiArray** property available

This is a deprecated property introduced in 1364-2001 that is not referenced in any other standard. For **vpiIntegerVar**, **vpiTimeVar**, **vpiRealVar**, **vpiRegArray**, and **vpiNetArray**, its value being TRUE meant that these objects represented multidimensional unpacked arrays.

35.13.2 VPI Mechanisms to Deal With Incompatibilities

Capability shall be provided to emulate the incompatible VPI behaviors where they conflict with the current standard. This allows older VPI applications dependent on these behaviors to be run unmodified, as long as they are applied only to designs (or portions of designs) they are compatible with. Two mechanisms to support this shall be provided, which can be used in combination:

1) Compile-based binding to a compatibility mode;

This mechanism requires recompilation of the VPI application source code, and is based on defining a compiler symbol that binds a particular application to a particular compatibility mode. To use this scheme, one of the following compiler symbols must be defined prior to compilation of any of the standard VPI include files in the application source code (either using a “#define” in the source code itself, or defined on the C-compiler command-line):

```
VPI_COMPATIBILITY_VERSION_1364v1995
VPI_COMPATIBILITY_VERSION_1364v2001
VPI_COMPATIBILITY_VERSION_1364v2005
VPI_COMPATIBILITY_VERSION_1800v2005
```

No more than one of these symbols shall be defined for a given application, and it must be consistently defined for all of its source code that can access any portion of VPI, including callback functions. A compilation error will occur during the processsing of vpi_user.h if more than one of the above symbols is defined.

2) Selection of default VPI compatibility mode run by the host simulator.

A means to set the default VPI compatibility mode shall be made available by the simulation provider. This shall determine the compatibility mode VPI behavior for *all* applications *not* using the compile-based scheme detailed in mechanism #1. Although VPI applications choosing this mechanism can be run without modification or recompilation, only one such default mode can be selected. Additional applications requiring different modes in the same run-time simulation environment *must* use the compile-based mechanism to do so.

35.13.2 Limitations of VPI Compatibility Mechanisms

The VPI user and VPI application provider should take steps to ensure that VPI applications dependent on these mechanisms are used only for designs or design partitions consistent with the mode selected. Designs should only require simulation versions older or equal to the VPI mode level.The behavior of a VPI application running in a mode that is incompatible with (older than) design objects it is processing is unpredictable, and shall not be guaranteed to be diagnosed by the VPI provider. Strictness of checking for consistency in this regard is left to the discretion of the VPI provider.