INSERT:

36.1.1 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

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** by 1364-2005, leaving **vpiMemory** only allowed as 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 only the **vpiMemoryWord** transition allowed 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 scalar fixed-width 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). In 1800-2005, they are 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 only were only accessed instead by iterations on **vpiNetArray** (from a scope). 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**, and **vpiRegArray** its value being TRUE meant that these objects represented multidimensional unpacked arrays.

36.1.2 VPI Mechanism 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 this behavior to be run unmodified, as long as they are applied to designs they are compatible with.

This mechanism is based on defining a compiler symbol that binds a particular application to a particular backwards compatibility mode. To invoke such a mode, 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 compile command-line):

```
VPI_COMPATIBILITY_VERSION_1364v1995
VPI_COMPATIBILITY_VERSION_1364v2001
VPI_COMPATIBILITY_VERSION_1364v2005
VPI_COMPATIBILITY_VERSION_1800v2005
```

Only 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. It shall be a compile-time error to define more than one of the above symbols for compiling any VPI application source code.

It is left up to the discretion of the VPI provider to check for consistency between the VPI compatibility level selected and the design the application is applied to. It is assumed that VPI applications that need such backwards compatibility have already been proven in their original VPI environment, and thus do not require rigorous levels of checking.