INSERT:

35.12 VPI Backwards-Compatibility Features and Limitations

The VPI data model has evolved 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.

35.12.1 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**	**2008**
1) **vpiMemory** exists as an object	Y	D	N	N	N
2) **vpiMemoryWord** exists as an object	Y	D	N	N	N
3) **vpiIntegerVar** and **vpiTimeVar** can be arrays	Y	Y	Y	N	N
4) **vpiRealVar** can be an array	N	Y	Y	N	N
5) **vpiVariables** iterations include **vpiReg** and **vpiRegArray**	N	N	N	Y	Y
6) **vpiReg** iterations on **vpiRegArray** include other objects	N	N	N	Y	Y
7) **vpiRegArray** iterations include variable arrays	N	N	N	Y	Y

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 or object not applicable or 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 36.16 detail 19).

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 standard versions (see section 36.18). 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 standard versions (see section 36.18). 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 standard versions, these array types are always represented as **vpiRegArray** objects, and **vpiIntegerVar** and **vpiTimeVar** objects are always non-array variables (see section 36.16).

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 standard versions, these are now exclusively represented as **vpiRegArray** objects (see section 36.16).

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

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 standards, they are both included in **vpiVariables** iterations (see section 36.16).

6) **vpiReg** iterations on **vpiRegArray** include other objects

This is a consequence of **vpiRegArray** objects being used to represent unpacked arrays of non-**vpiReg** elements in 1800 standards (see section 36.16). **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) **vpiRegArray** iterations include variable array objects

This is another consequence of **vpiRegArray** objects being used to represent unpacked arrays of non-**vpiReg** elements in 1800 standards (see section 36.16). In 1364-2001 and 1364-2005 standards **vpiRegArray** iterations only included arrays of **vpiReg** objects, but in 1800 standards this iteration includes arrays of **vpiIntegerVar**, **vpiTimeVar**, and **vpiRealVar**.

35.12.2 VPI Mechanisms to Deal With Incompatibilities

In order to ease the transition to the latest VPI standard for older applications, 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) with which they are compatible. This capability is intended only as an interim measure to allow extra time for applications to be upgraded; it does not provide general emulation of older behaviors for newer design constructs. For example, it does not allow 1364 applications to run on portions of designs requiring 1800-level simulation capability.

As described in sections 35.12.2.1 and 35.12.2.2 below, two mechanisms to support this shall be provided, which can be used in combination.

35.12.2.1 Mechanism 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 (setting it to the numeric constant “1”), or defined on the C-compiler command-line:

```
VPI_COMPATIBILITY_VERSION_1364v1995
VPI_COMPATIBILITY_VERSION_1364v2001
VPI_COMPATIBILITY_VERSION_1364v2005
VPI_COMPATIBILITY_VERSION_1800v2005
VPI_COMPATIBILITY_VERSION_1800v2008
```

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. This ensures that all design information is handled in the same way for a given mode across the entire application. A compilation error will occur during the processsing of vpi_user.h if more than one of the above symbols is defined.

Example:

VPI source code file with a compatibility mode selected:

```
/* VPI application mytask */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define VPI_COMPATIBILITY_VERSION_1364v2001 1
#include “vpi_user.h”
#include “sv_vpi_user.h”

#include “my_appl_header.h”
......
......
```

Alternatively, the same mode selection could be performed by defining the following option on the C-compiler command line:

```
-DVPI_COMPATIBILITY_VERSION_1364v2001
```

When a mode is selected by one of the means above, C-preprocessor constructs in “vpi_user.h” cause the following VPI functions to be redefined to mode-specific versions:

```
vpi_compare_objects
vpi_control
vpi_get
vpi_get_str
vpi_get_value
vpi_handle
vpi_handle_by_index
vpi_handle_by_multi_index
vpi_handle_by_name
vpi_handle_multi
vpi_iterate
vpi_put_value
vpi_register_cb
vpi_scan
```

For example, defining the mode symbol ‘`VPI_COMPATIBILITY_VERSION_1364v2001`’ as shown above will cause ‘vpi_handle’ to be redefined as:

```
vpi_handle_1364v2001
```

This retargets all calls to ‘vpi_handle’ in the recompiled application to this mode-specific variant, achieving mode-compatible behavior. See “vpi_user.h” (1364-2005 Annex G) for the complete set of definitions.

35.12.2.1 Mechanism 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 shall be selectable for a given simulation run. Additional applications requiring different modes in the same run-time simulation environment must use the compile-based mechanism to do so.

35.12.3 Limitations of VPI Compatibility Mechanisms

When a VPI application uses the compatibility mode mechanism, the application user and application provider should ensure that the design or design partition the application is applied to is consistent with the mode, and does not include constructs that are only supported in other modes. If the design contains unsupported constructs, the behavior of the VPI implementation is undefined. The extent of checking for consistency between constructs and mode is left to the discretion of the VPI implementation.

In general, VPI users and application developers are strongly encouraged to update their applications to the latest VPI version as soon as possible. The compatibility mode feature should be used only as a temporary solution until such upgrades can be completed or become available. It should be expected that older modes will be phased out as new versions of the standard become available.

Annex L

(normative)

vpi_user.h (1364-2005 Annex G)

INSERT after “/****************** FUNCTION DECLARATIONS *****************/” comment:

```
/* Compatibility-mode variants of functions */

#if VPI_COMPATIBILITY_VERSION_1364v1995
#if VPI_COMPATIBILITY_VERSION_1364v2001 || VPI_COMPATIBILITY_VERSION_1364v2005
    || VPI_COMPATIBILITY_VERSION_1800v2005 || VPI_COMPATIBILITY_VERSION_1800v2008
#error “Only one VPI_COMPATIBILITY_VERSION symbol definition is allowed.”
#endif
#define vpi_compare_objects vpi_compare_objects_1364v1995
#define vpi_control vpi_control_1364v1995
#define vpi_get vpi_get_1364v1995
#define vpi_get_str vpi_get_str_1364v1995
#define vpi_get_value vpi_get_value_1364v1995
#define vpi_handle vpi_handle_1364v1995
#define vpi_handle_by_index vpi_handle_by_index_1364v1995
#define vpi_handle_by_multi_index vpi_handle_by_multi_index_1364v1995
#define vpi_handle_by_name vpi_handle_by_name_1364v1995
#define vpi_handle_multi vpi_handle_multi_1364v1995
#define vpi_iterate vpi_iterate_1364v1995
#define vpi_put_value vpi_put_value_1364v1995
#define vpi_register_cb vpi_register_cb_1364v1995
#define vpi_scan vpi_scan_1364v1995
#elif VPI_COMPATIBILITY_VERSION_1364v2001
#if VPI_COMPATIBILITY_VERSION_1364v1995 || VPI_COMPATIBILITY_VERSION_1364v2005
    || VPI_COMPATIBILITY_VERSION_1800v2005 || VPI_COMPATIBILITY_VERSION_1800v2008
#error “Only one VPI_COMPATIBILITY_VERSION symbol definition is allowed.”
#endif
#define vpi_compare_objects vpi_compare_objects_1364v2001
#define vpi_control vpi_control_1364v2001
#define vpi_get vpi_get_1364v2001
#define vpi_get_str vpi_get_str_1364v2001
#define vpi_get_value vpi_get_value_1364v2001
#define vpi_handle vpi_handle_1364v2001
#define vpi_handle_by_index vpi_handle_by_index_1364v2001
#define vpi_handle_by_multi_index vpi_handle_by_multi_index_1364v2001
#define vpi_handle_by_name vpi_handle_by_name_1364v2001
#define vpi_handle_multi vpi_handle_multi_1364v2001
#define vpi_iterate vpi_iterate_1364v2001
#define vpi_put_value vpi_put_value_1364v2001
#define vpi_register_cb vpi_register_cb_1364v2001
#define vpi_scan vpi_scan_1364v2001
#elif VPI_COMPATIBILITY_VERSION_1364v2005
#if VPI_COMPATIBILITY_VERSION_1364v1995 || VPI_COMPATIBILITY_VERSION_1364v2001
    || VPI_COMPATIBILITY_VERSION_1800v2005 || VPI_COMPATIBILITY_VERSION_1800v2008
#error “Only one VPI_COMPATIBILITY_VERSION symbol definition is allowed.”
#endif
#define vpi_compare_objects vpi_compare_objects_1364v2005
#define vpi_control vpi_control_1364v2005
#define vpi_get vpi_get_1364v2005
```

```
#define vpi_get_str vpi_get_str_1364v2005
#define vpi_get_value vpi_get_value_1364v2005
#define vpi_handle vpi_handle_1364v2005
#define vpi_handle_by_index vpi_handle_by_index_1364v2005
#define vpi_handle_by_multi_index vpi_handle_by_multi_index_1364v2005
#define vpi_handle_by_name vpi_handle_by_name_1364v2005
#define vpi_handle_multi vpi_handle_multi_1364v2005
#define vpi_iterate vpi_iterate_1364v2005
#define vpi_put_value vpi_put_value_1364v2005
#define vpi_register_cb vpi_register_cb_1364v2005
#define vpi_scan vpi_scan_1364v2005
#elif VPI_COMPATIBILITY_VERSION_1800v2005
#if VPI_COMPATIBILITY_VERSION_1364v1995 || VPI_COMPATIBILITY_VERSION_1364v2001
    || VPI_COMPATIBILITY_VERSION_1364v2005 || VPI_COMPATIBILITY_VERSION_1800v2008
#error “Only one VPI_COMPATIBILITY_VERSION symbol definition is allowed.”
#endif
#define vpi_compare_objects vpi_compare_objects_1800v2005
#define vpi_control vpi_control_1800v2005
#define vpi_get vpi_get_1800v2005
#define vpi_get_str vpi_get_str_1800v2005
#define vpi_get_value vpi_get_value_1800v2005
#define vpi_handle vpi_handle_1800v2005
#define vpi_handle_by_index vpi_handle_by_index_1800v2005
#define vpi_handle_by_multi_index vpi_handle_by_multi_index_1800v2005
#define vpi_handle_by_name vpi_handle_by_name_1800v2005
#define vpi_handle_multi vpi_handle_multi_1800v2005
#define vpi_iterate vpi_iterate_1800v2005
#define vpi_put_value vpi_put_value_1800v2005
#define vpi_register_cb vpi_register_cb_1800v2005
#define vpi_scan vpi_scan_1800v2005
#elif VPI_COMPATIBILITY_VERSION_1800v2008
#if VPI_COMPATIBILITY_VERSION_1364v1995 || VPI_COMPATIBILITY_VERSION_1364v2001
    || VPI_COMPATIBILITY_VERSION_1364v2005 || VPI_COMPATIBILITY_VERSION_1800v2005
#error “Only one VPI_COMPATIBILITY_VERSION symbol definition is allowed.”
#endif
#define vpi_compare_objects vpi_compare_objects_1800v2008
#define vpi_control vpi_control_1800v2008
#define vpi_get vpi_get_1800v2008
#define vpi_get_str vpi_get_str_1800v2008
#define vpi_get_value vpi_get_value_1800v2008
#define vpi_handle vpi_handle_1800v2008
#define vpi_handle_by_index vpi_handle_by_index_1800v2008
#define vpi_handle_by_multi_index vpi_handle_by_multi_index_1800v2008
#define vpi_handle_by_name vpi_handle_by_name_1800v2008
#define vpi_handle_multi vpi_handle_multi_1800v2008
#define vpi_iterate vpi_iterate_1800v2008
#define vpi_put_value vpi_put_value_1800v2008
#define vpi_register_cb vpi_register_cb_1800v2008
#define vpi_scan vpi_scan_1800v2008
#endif
```

Annex M

(normative)

sv_vpi_user.h (1800-2005 Annex I)

INSERT (after “#define vpiGeneric 653”):

```
/* Compatibility-mode property and values (object argument == NULL) */
#define vpiCompatibilityMode 654
#define vpiMode1364v1995 1
#define vpiMode1364v2001 2
#define vpiMode1364v2005 3
#define vpiMode1800v2005 4
#define vpiMode1800v2008 5
```