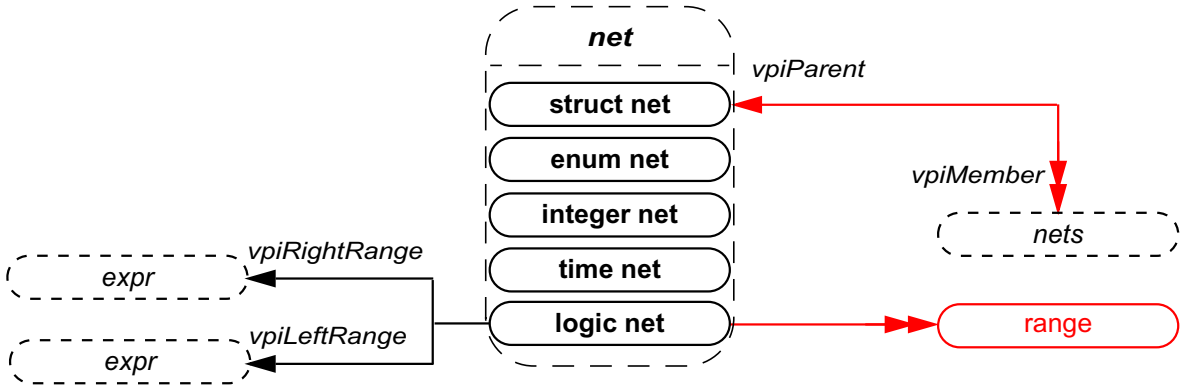
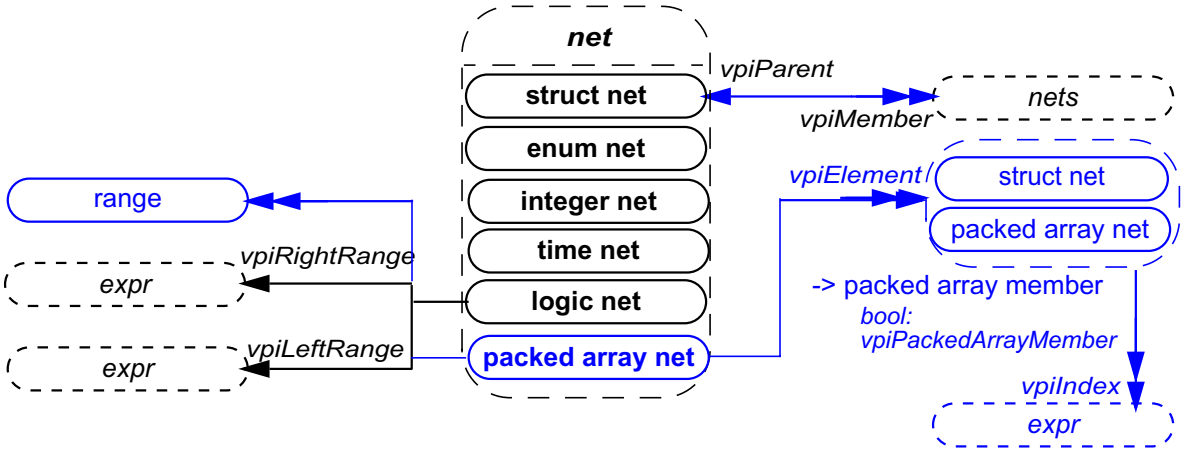
36.15 Nets

REPLACE:



WITH:



36.15 Nets [continued]

REPLACE:

1) Any net declared as an array with one or more unpacked ranges is an array net. The range iterator for an array net returns only the unpacked ranges for the array.

WITH:

1) Any net declared as an array with one or more unpacked ranges is an array net. The range iterator for an array net returns only the unpacked ranges for the array. Any packed struct net declared with one or more explicit packed ranges is a packed array net. The range iterator for a packed array net returns only the explicit packed ranges for such a net. It shall not return the implicit range of the packed struct net elements themselves. For example:

```
// a 34-bit-wide struct net (range iteration not allowed)
wire struct packed { logic [1:0]vec1; integer i1; } psnet;

// a packed array net (ranges “[3:0]” and “[2:1]” returned by range iteration)
wire struct packed { logic [1:0]vec1; integer i1; } [3:0][2:1] panet;

// an array net (ranges “[5:4]” and “[6:8]” returned by range iteration)
wire struct packed { logic [1:0]vec1; integer i1; } [3:0][2:1] anet [5:4][6:8];
```

.....

REPLACE:

2) The boolean property **vpiArray** is deprecated in this standard. The **vpiArrayMember** property shall be TRUE for a net that is an element of an array net. It shall be FALSE otherwise.

WITH:

2) The boolean property **vpiArray** is deprecated in this standard. The **vpiArrayMember** property shall be TRUE for a net that is an element of an array net. It shall be FALSE otherwise. The **vpiPackedArrayMember** property shall be TRUE for a packed struct net or a packed array net that is an element of a packed array net.

.....

REPLACE:

15) For **vpiLoad**, **vpiLocalLoad**, **vpiDriver** and **vpiLocalDriver** iterators, if the object is **vpiNet** for an enum net, an integer net, a time net, or for a logic net or struct net for which **vpiVector** is TRUE, then all loads or drivers are returned exactly once as the loading or driving object. That is, if a part-select loads or drives only some bits, the load or driver returned is the part-select. If a driver is repeated, it is only returned once. To trace exact bit-by-bit connectivity pass a **vpiNetBit** object to **vpi_iterate**.

WITH:

15) For **vpiLoad**, **vpiLocalLoad**, **vpiDriver** and **vpiLocalDriver** iterators, if the object is ~~**vpiNet** for~~ a vector net (an enum net, ~~an~~ integer net, ~~a~~ time net, packed array net, or ~~for~~ a logic net or struct net for which **vpiVector** is TRUE), then all loads or drivers are returned exactly once as the loading or driving object. That is, if a part-select loads or drives only some bits, the load or driver returned is the part-select. If a driver is repeated, it is only returned once. To trace exact bit-by-bit connectivity pass a **vpiNetBit** object to **vpi_iterate**.

36.15 Nets [continued]

REPLACE:

21) **vpiSize** for an array net shall return the number of nets in the array. For unpacked structures, the size returned indicates the number of members in the structure. For an enum net, integer net, logic net, time net, or packed struct net, **vpiSize** shall return the size of the net in bits. For a net bit, **vpiSize** shall return 1.

WITH:

21) **vpiSize** for an array net shall return the number of nets in the array. For unpacked structures, the size returned indicates the number of members in the structure. For an enum net, integer net, logic net, time net, ~~or~~ packed struct net, or packed array net, **vpiSize** shall return the size of the net in bits. For a net bit, **vpiSize** shall return 1.

.....

REPLACE:

22) **vpi_iterate(vpiIndex, net_handle)** shall return the set of indices for a net within an array net, starting with the index for the net and working outward. If the net is not part of an array (the **vpiArrayMember** property is FALSE), a NULL shall be returned.

WITH:

22) **vpi_iterate(vpiIndex, net_handle)** shall return the set of indices for a net within an array net, starting with the index for the net and working outward. If the net is not part of an array (the **vpiArrayMember** property is FALSE), a NULL shall be returned. The **vpiIndex** iterator shall work similarly for packed array net elements (packed struct nets or packed array nets whose **vpiPackedArrayMember** property is TRUE). The indices returned shall start with the index of the element and work outward until the **vpiParent** packed array net is reached (see detail 28). The indices retrieved for packed array net elements shall be the same as those shown in the example for detail 29 for each of the sub-elements returned by **vpiElement**. The indices will be retrieved in right-to-left order as they appear in the text.

36.15 Nets [continued]

REPLACE:

25) A logic net without a packed dimension defined is a scalar; and for that object the property **vpiScalar** shall return TRUE and the property **vpiVector** shall return FALSE. A logic net with one or more packed dimensions defined is a vector, and the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). A packed struct net is a vector, and the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). A net bit is a scalar, and the property **vpiScalar** shall return TRUE (**vpiVector** shall return FALSE). The properties **vpiScalar** and **vpiVector** when queried on a handle to an enum net shall return the value of the respective property for an object for which the typespec is the same as the base typespec of the typespec of the enum net. For an integer net or a time net, the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). For an array net, the **vpiScalar** and **vpiVector** properties shall return the values of the respective properties for an array element. The **vpiScalar** and **vpiVector** properties shall return FALSE for all other net objects.

WITH:

25) A logic net without a packed dimension defined is a scalar; and for that object the property **vpiScalar** shall return TRUE and the property **vpiVector** shall return FALSE. A logic net with one or more packed dimensions defined is a vector, and the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). ~~A p~~Packed struct nets and packed array nets are~~is a~~ vectors, and the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). A net bit is a scalar, and the property **vpiScalar** shall return TRUE (**vpiVector** shall return FALSE). The properties **vpiScalar** and **vpiVector** when queried on a handle to an enum net shall return the value of the respective property for an object for which the typespec is the same as the base typespec of the typespec of the enum net. For an integer net or a time net, the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). For an array net, the **vpiScalar** and **vpiVector** properties shall return the values of the respective properties for an array element. The **vpiScalar** and **vpiVector** properties shall return FALSE for all other net objects.

.....

ADD:

29) The **vpiElement** transition shall be used to iterate over the sub-elements of packed array nets. Unlike **vpiNet** iterations for **vpiArrayNet** objects, **vpiElement** shall retrieve elements for only one dimension level at a time. This means that for multi-dimensioned packed array nets, **vpiElement** shall retrieve elements which are themselves also **vpiPackedArrayNet** objects. **vpiElement** can then be used to iterate over the sub-elements of these objects and so on, until the leaf level struct nets are returned. In other words, the datatype of each element retrieved by **vpiElement** is equivalent to the original **vpiPackedArrayNet** object's datatype with one leftmost packed range removed. For example, consider the following **vpiPackedArrayNet** object:

    typedef struct packed { integer i1; logic [1:0][2:3]bvec; } pavartype;

    wire pavartype [0:2][6:3] panet1;

    The **vpiElement** transition applied to `panet1` shall return 3 **vpiPackedArrayNet** objects: `panet1[0]`, `panet1[1]`, and `panet1[2]`. The **vpiElement** transition applied to **vpiPackedArrayNet** `panet1[0]` in turn shall retrieve **vpiStructNet** objects `panet1[0][6]`, `panet1[0][5]`, `panet1[0][4]`, and `panet1[0][3]` respectively. Also, the **vpiParent** transition for all the above-mentioned sub-elements of `panet1` shall return `panet1` (as per detail 28), since `panet1` is "the largest containing packed array net object".

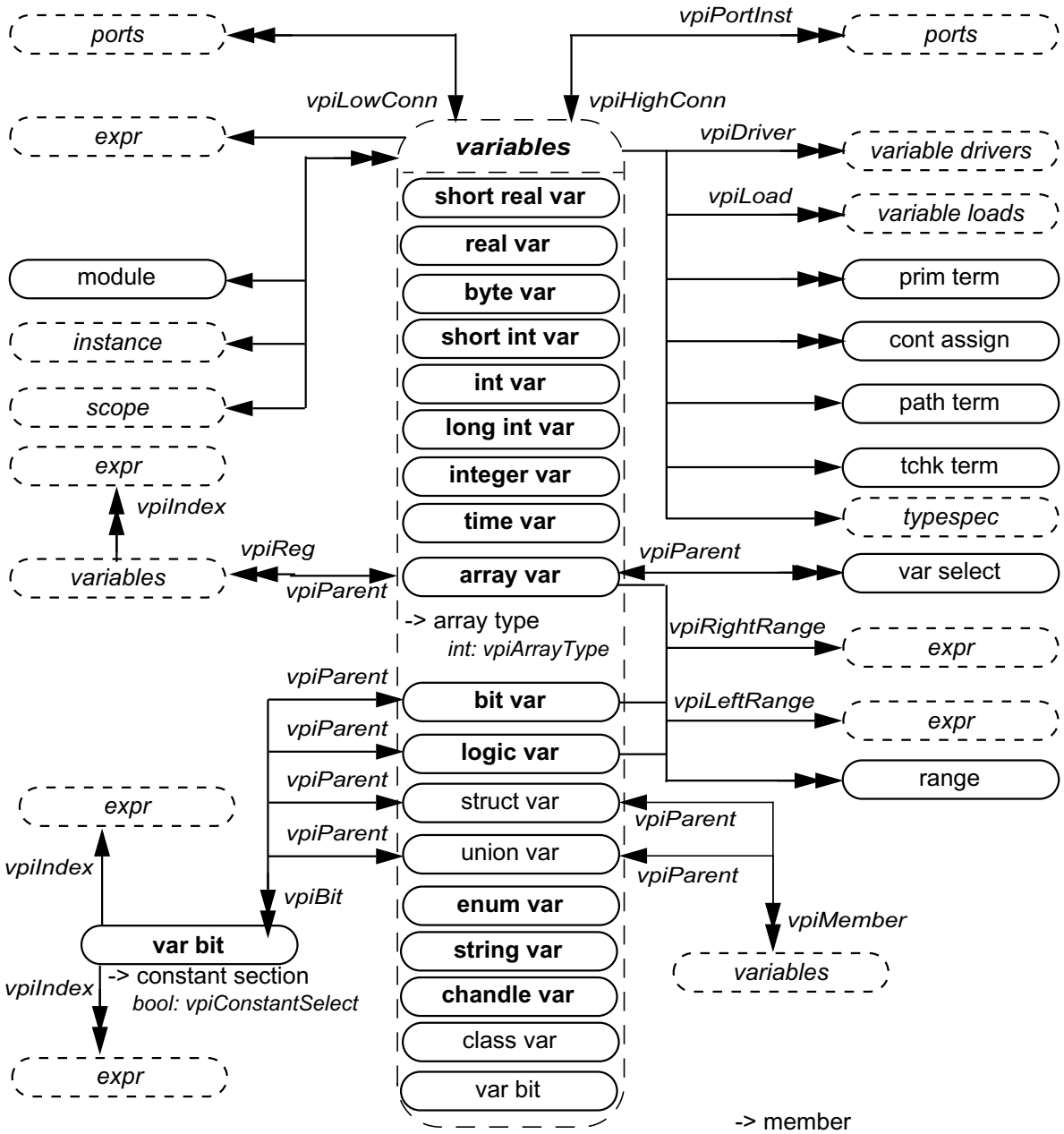
36.15 Nets [continued]

ADD:

30) The **vpiStructUnionMember** property shall be TRUE for any enum net, integer net, time net, struct net, packed array net, or array net that is a direct member of a struct net, i.e. whose **vpiParent** is a struct net (see detail 28). This property shall be FALSE for any net, array net, or net bit whose **vpiParent** is not a struct net.

36.16 Variables

REPLACE (diagram next page):



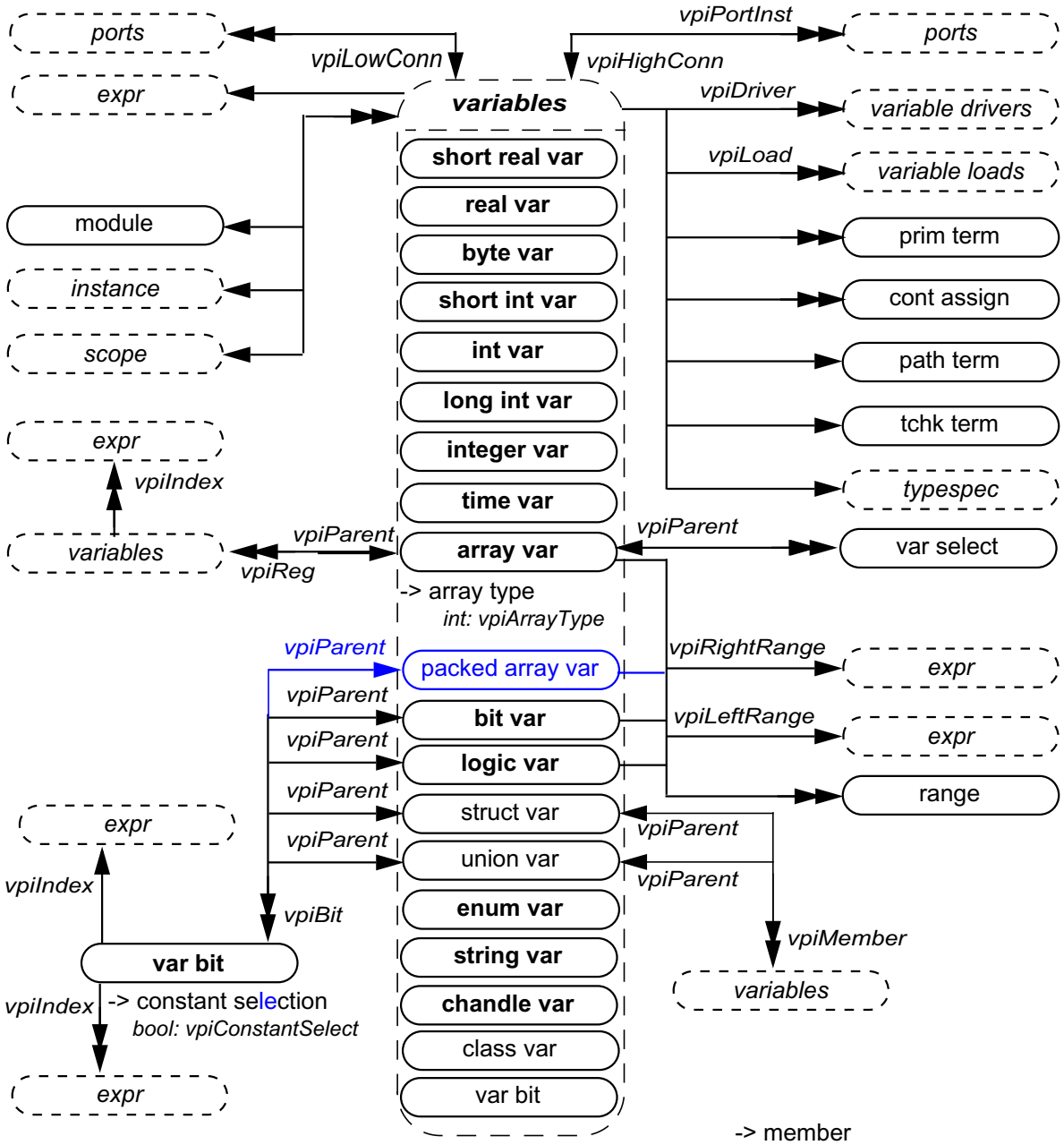
-> access by index
vpi_handle_by_index()
vpi_handle_by_multi_index()
-> array member
bool: vpiArray (deprecated)
bool: vpiArrayMember
-> name
str: vpiName
str: vpiFullName
-> sign
bool: vpiSigned

-> size
int: vpiSize
-> array type
int: vpiArrayType
-> lifetime
bool: vpiAutomatic
-> constant variable
bool: vpiConstantVariable
-> determine random availability
bool: vpiIsRandomized
-> randomization type
int: vpiRandType

-> member
bool: vpiStructUnionMember
->value
vpi_get_value()
vpi_put_value()
-> scalar
bool: vpiScalar
-> visibility
int: vpiVisibility
-> vector
bool: vpiVector
-> validity
int: vpiValid

WITH (diagram next page):

[Note to editor: I made room for the new variable object by shortening the arcs at the top of the diagram. Alternatively, more room would be available after the deletion shown in the properties section. Also, there is a minor spelling error correction in properties section under “var bit” that may be easily missed.]



-> access by index
vpi_handle_by_index()
vpi_handle_by_multi_index()

-> array member
bool: vpiArray (deprecated)
bool: vpiArrayMember

-> name
str: vpiName
str: vpiFullName

-> sign
bool: vpiSigned

-> size
int: vpiSize

~~-> array type~~
~~*int: vpiArrayType*~~

-> lifetime
bool: vpiAutomatic

-> constant variable
bool: vpiConstantVariable

-> determine random availability
bool: vpiIsRandomized

-> randomization type
int: vpiRandType

-> member
bool: vpiStructUnionMember

->value
vpi_get_value()
vpi_put_value()

-> scalar
bool: vpiScalar

-> visibility
int: vpiVisibility

-> vector
bool: vpiVector

-> validity
int: vpiValid

36.16 Variables [continued]

REPLACE:

4) The range relation is valid only when the variable is an array var, or the variable is a logic var or a bit var and the property **vpiVector** is TRUE. When applied to array vars this relation returns only unpacked ranges. When applied to logic and bit variables, it returns only the packed ranges.

WITH:

4) The range relation is valid only when the variable is an array var, a packed array var, or the variable is a logic var or a bit var and the property **vpiVector** is TRUE. When applied to array vars this relation returns only unpacked ranges. When applied to logic ~~and,~~ bit, and packed array variables, it returns only the packed ranges.

.....

REPLACE:

6) If a logic var or bit var has more than one packed dimension, **vpiLeftRange** and **vpiRightRange** shall return the bounds of the leftmost packed dimension. If an array var has more than one unpacked dimension, **vpiLeftRange** and **vpiRightRange** shall return the bounds of the leftmost unpacked dimension.

WITH:

6) If a logic var, ~~or~~ bit var, or packed array var has more than one packed dimension, **vpiLeftRange** and **vpiRightRange** shall return the bounds of the leftmost packed dimension. If an array var has more than one unpacked dimension, **vpiLeftRange** and **vpiRightRange** shall return the bounds of the leftmost unpacked dimension.

.....

REPLACE:

12) **vpiBit** iterator applies only for logic, bit, packed struct, and packed union variables.

WITH:

12) **vpiBit** iterator applies only for logic, bit, packed struct, ~~and~~ packed union, and packed array variables.

.....

REPLACE:

17) When the **vpiStructUnionMember** property is TRUE, it indicates that the variable is a member of a parent struct or union variable. See also the relations in 36.22.

WITH:

17) When the **vpiStructUnionMember** property is TRUE, it indicates that the variable is a member of a parent struct or union variable. See also the relations in 36.22 [make this a reference to the **original** 36.22], and 36.22 [make this a reference to the **new** packed array var section] detail 5.

36.16 Variables [continued]

REPLACE:

18) If a variable is an element of an array (the **vpiArrayMember** property is TRUE), the **vpiIndex** iterator shall return the indexing expressions that select that specific variable out of the array.

WITH:

18) If a variable is an element of an array (the **vpiArrayMember** property is TRUE), the **vpiIndex** iterator shall return the indexing expressions that select that specific variable out of the array. See 36.22 [the new section for packed array vars] (and detail 6) for similar functionality available for elements of packed array vars.

.....

REPLACE:

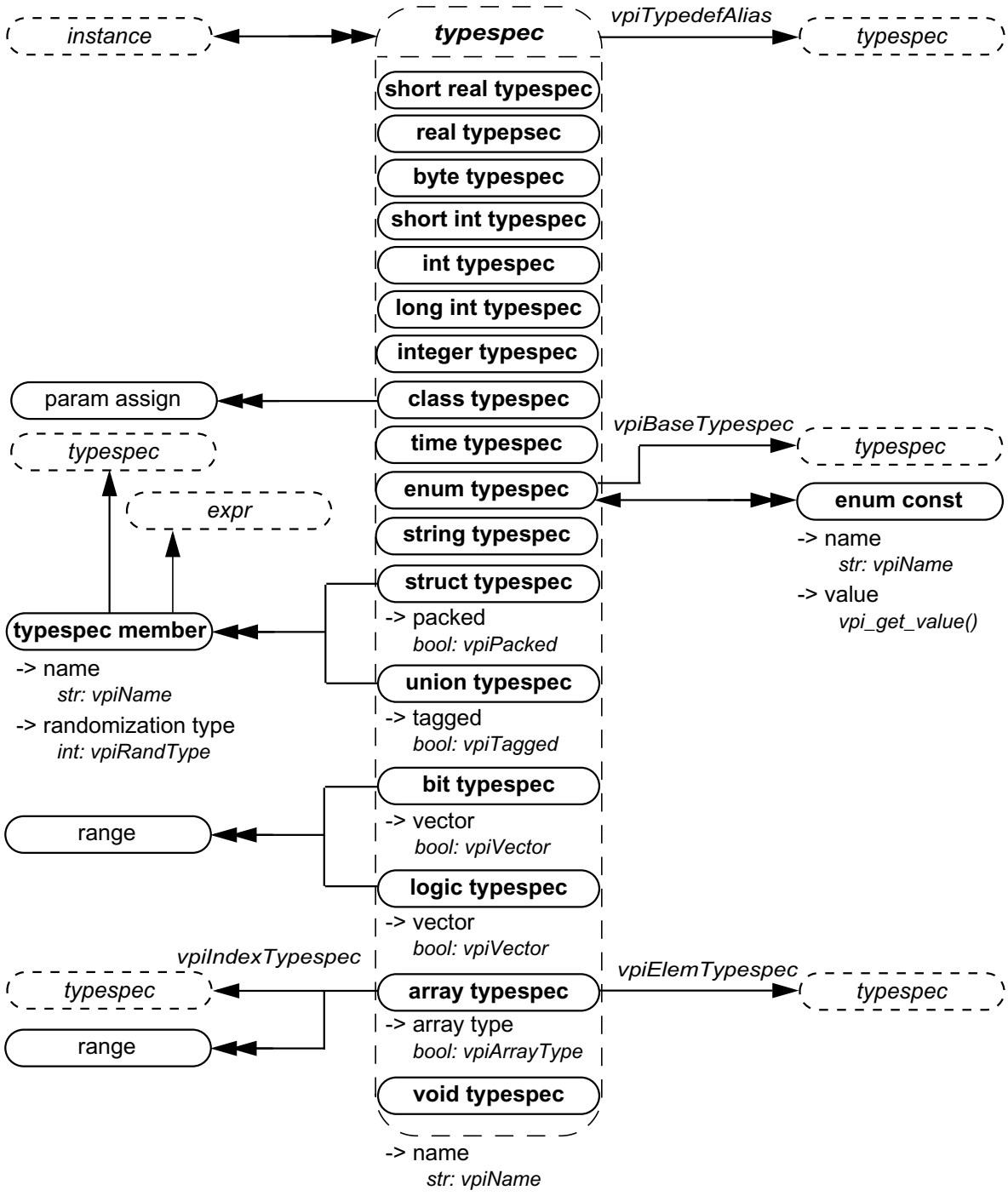
20) A bit var or logic var, without a packed dimension defined, is a scalar and for those objects, the property **vpiScalar** shall return TRUE, and the property **vpiVector** shall return FALSE. A bit var or logic var, with one or more packed dimensions defined, is a vector, and the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). A packed struct var and a packed union var are vectors, and the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). A var bit is a scalar, and the property **vpiScalar** shall return TRUE (**vpiVector** shall return FALSE). The properties **vpiScalar** and **vpiVector** when queried on a handle to an enum var shall return the value of the respective property for an object for which the typespec is the same as the base typespec of the typespec of the enum var. For an integer var, time var, short int var, int var, long int var, and byte var, the property vpiVector shall return TRUE (**vpiScalar** shall return FALSE). For an array var, the **vpiScalar** and **vpiVector** properties shall return the values of the respective properties for an array element. The **vpiScalar** and **vpiVector** properties shall return FALSE for all other var objects.

WITH:

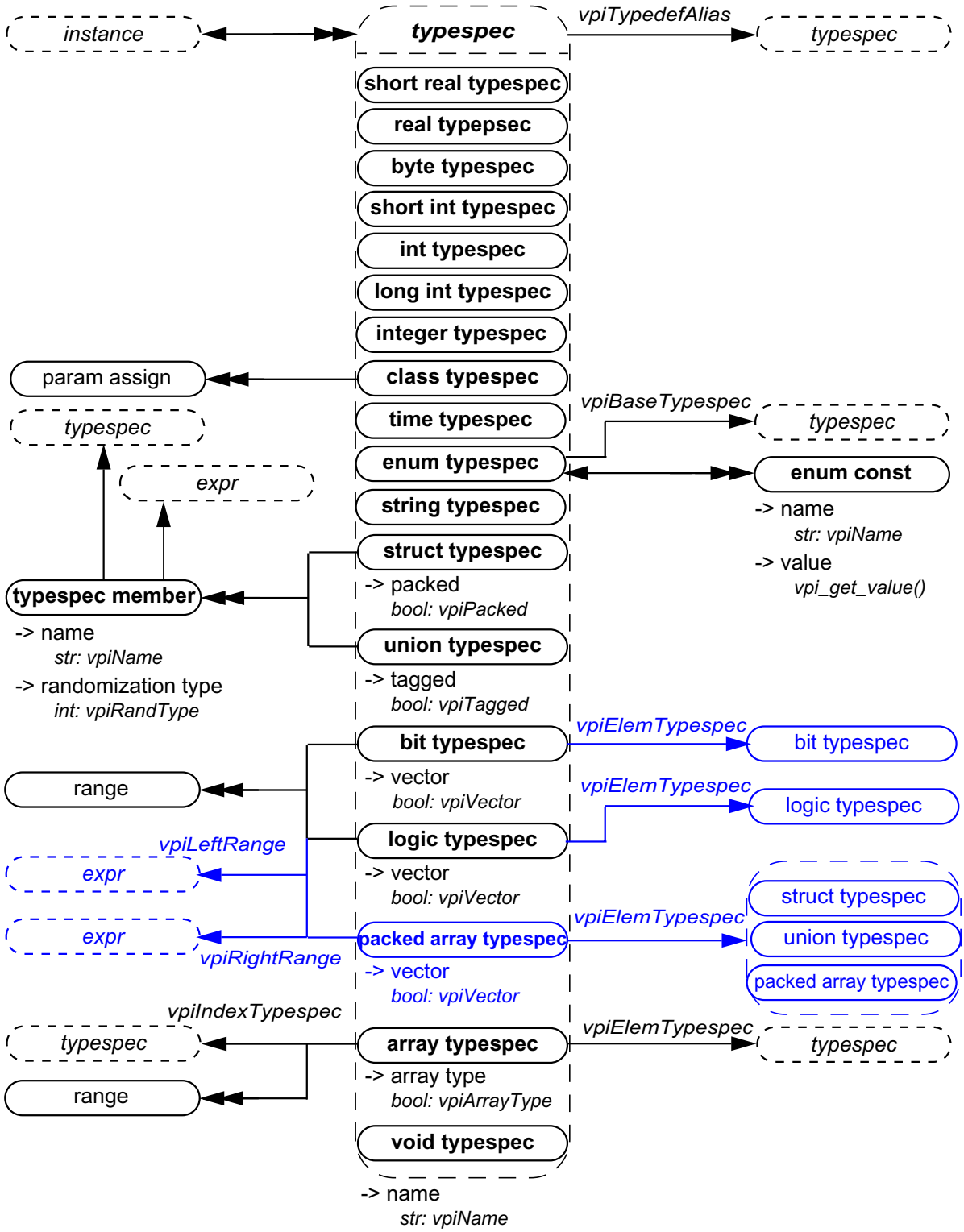
20) A bit var or logic var, without a packed dimension defined, is a scalar and for those objects, the property **vpiScalar** shall return TRUE, and the property **vpiVector** shall return FALSE. A bit var or logic var, with one or more packed dimensions defined, is a vector, and the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). A packed struct var ~~and a~~, packed union var, and a packed array var are vectors, and the property **vpiVector** shall return TRUE (**vpiScalar** shall return FALSE). A var bit is a scalar, and the property **vpiScalar** shall return TRUE (**vpiVector** shall return FALSE). The properties **vpiScalar** and **vpiVector** when queried on a handle to an enum var shall return the value of the respective property for an object for which the typespec is the same as the base typespec of the typespec of the enum var. For an integer var, time var, short int var, int var, long int var, and byte var, the property vpiVector shall return TRUE (**vpiScalar** shall return FALSE). For an array var, the **vpiScalar** and **vpiVector** properties shall return the values of the respective properties for an array element. The **vpiScalar** and **vpiVector** properties shall return FALSE for all other var objects.

36.21 Typespec

REPLACE:



WITH:



36.21 Typespec [continued]

ADD:

8) The **vpiElemTypespec** transition shall be used to unwind the typespec of an unpacked array (array typespec) or a packed array (packed array typespec, or a bit or logic typespec with one or more dimensions) one dimension level at a time. This means that for a multi-dimensional array typespec (a typespec with more than one unpacked range), **vpi_handle(vpiElemTypespec, array_typespec_handle)** shall initially retrieve a **vpiArrayTypespec** equivalent to the original typespec with its leftmost unpacked range removed. Subsequent calls to the **vpiElemTypespec** method continue the unwinding until a typespec object is retrieved that has no unpacked ranges remaining. Similarly, when the **vpiElemTypespec** is applied to a typespec of a multi-dimensional packed array object, a **vpiPackedArrayTypespec** (or **vpiBitTypespec** or **vpiLogicTypespec**) is retrieved which is equivalent to the original typespec with its leftmost packed range removed, and so on, until a typespec without an explicit packed range is retrieved. When the **vpiElemTypespec** relation is applied to a **vpiStructTypespec**, **vpiUnionTypespec**, or a **vpiBitTypespec** or **vpiLogicTypespec** with no ranges present, it shall return NULL. This allows packed or unpacked array typespecs constructed with multiple typedefs to be unwound without losing name information. Consider the complex array typespec defined below for "arr":

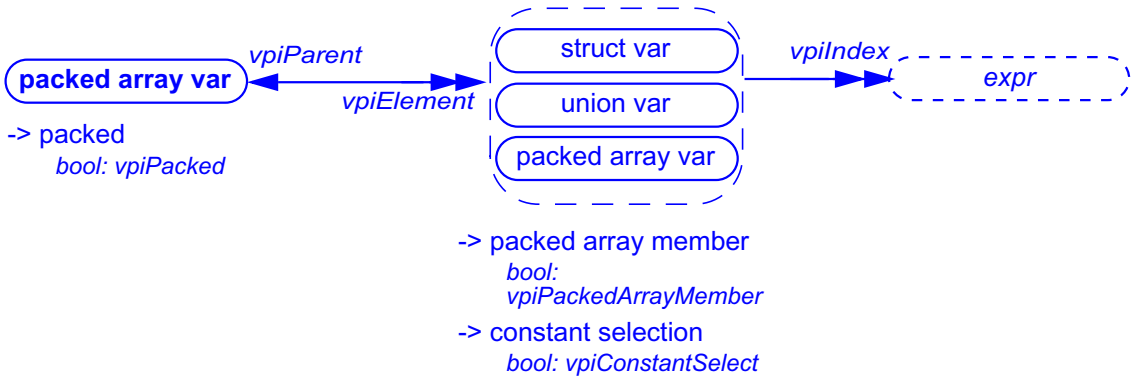
```
typedef struct packed { int i1; bit bvec; } [1:3] parrtype;
typedef parrtype [2:1] parrtype2;
typedef parrtype2 unparrtype [6:4];
unparrtype arr [3:0];
```

The typespec of the object "arr" is an unpacked 4 x 3 array typespec with a `NULL` **vpiName** property. The typespec retrieved by applying **vpiElemTypespec** to this is a 2-element unpacked array typespec with a **vpiName** property of "unparrtype". The typespec retrieved by using **vpiElemTypespec** on this in turn yields a 2 x 3 packed array typespec (of packed struct objects) with a **vpiName** property of "parrtype2". Using **vpiElemTypespec** again in turn yields another packed array typespec (of 3 packed struct objects) with a **vpiName** property of "parrtype". One more application of **vpiElemTypespec** to this result yields a struct typespec, a non-array typespec for which no further array sub-elements exist (the unwinding is done).

9) If a logic typespec, bit typespec, or packed array typespec has more than one packed dimension, **vpiLeftRange** and **vpiRightRange** shall return the bounds of the leftmost packed dimension. If an array typespec has more than one unpacked dimension, **vpiLeftRange** and **vpiRightRange** shall return the bounds of the leftmost unpacked dimension.

ADD ((*) choose appropriate new section number):

36.22 (*) Packed array variables



Details:

1) **vpiPackedArrayVar** objects shall represent packed arrays of packed struct var and union var objects. The properties **vpiVector**, and **vpiPacked** for these objects and their underlying struct var or union var elements shall always be TRUE (see 36.16).

2) For consistency with other variable-width vector objects, the **vpiSize** property for **vpiPackedArrayVar** objects shall be the number of bits in the packed array, not the number of struct or union elements. The total number of struct var or union var elements for a packed array var can be obtained by computing the product of the **vpiSize** property for all of its packed ranges.

3) The **vpiElement** transition shall be used to iterate over the sub-elements of packed array variables. Unlike **vpiVarSelect** or **vpiReg** transitions for **vpiArrayVar** objects, **vpiElement** shall retrieve elements for only one dimension level at a time. This means that for multi-dimensioned packed arrays, **vpiElement** shall retrieve elements which are themselves also **vpiPackedArrayVar** objects. **vpiElement** can then be used to iterate over the sub-elements of these objects and so on, until the leaf level struct or union vars are returned. In other words, the datatype of each element retrieved by **vpiElement** is equivalent to the original **vpiPackedArrayVar** object's datatype with the leftmost packed range removed. For example, consider the following **vpiPackedArrayVar** object:

```
typedef struct packed { int i1; bit [1:0][2:3]bvec; } pavartype;
pavartype [0:2][6:3] pavar1;
```

The **vpiElement** transition applied to `pavar1` shall return 3 **vpiPackedArrayVar** objects: `pavar1[0]`, `pavar1[1]`, and `pavar1[2]`. The **vpiElement** transition applied to **vpiPackedArrayVar** `pavar1[0]` in turn shall retrieve **vpiStructVar** objects `pavar1[0][6]`, `pavar1[0][5]`, `pavar1[0][4]`, and `pavar1[0][3]` respectively. Also, the **vpiParent** transition for all the above-mentioned sub-elements of `pavar1` shall return `pavar1` (as per detail 29 of 36.16, since `pavar1` is "the largest containing packed array object").

4) The **vpiPackedArrayMember** property shall be `TRUE` for any struct var, union var, or packed array var whose **vpiParent** is a packed array var (see detail 29 of 36.16).

36.22 (*) Packed array variables [continued]

5) The **vpiStructUnionMember** property shall be TRUE only for packed array vars that are direct members of struct or union vars, i.e. whose **vpiParent** is a struct or union var (see detail 29 of 36.16). This property shall be FALSE for all sub-elements (as returned by the **vpiElement** iterator) of such packed array vars.

6) **vpi_iterate(vpiIndex, packed_array_var_handle)** shall return the set of indices for a sub-element of a packed array variable (relative to its **vpiParent**), starting with the index for the sub-element and working outwards. The indices retrieved shall be the same as those shown in the example for detail 3 for each of the sub-elements returned by **vpiElement**. The indices will be retrieved in right-to-left order as they appear in the text.

N.2 Source code [appendix N, sv_vpi_user.h]

REPLACE:

```
#define vpiChandleVar 622
```

WITH:

```
#define vpiChandleVar 622
#define vpiPackedArrayVar 623 [or TBD]
```

Editor: Values noted as "[or TBD]" may be assigned alternate unused values to avoid conflicts with other new ones.

.....

REPLACE:

```
#define vpiTypespecMember 644
```

WITH:

```
#define vpiTypespecMember 644
#define vpiPackedArrayTypespec 692 [or TBD]
```

.....

REPLACE:

```
#define vpiContinue 685
```

WITH:

```
#define vpiContinue 685
#define vpiPackedArrayNet 693 [or TBD]
```

.....

REPLACE:

```
#define vpiMember 742
```

WITH:

```
#define vpiMember 742
#define vpiElement 743 [or TBD]
```

N.2 Source code [continued]

REPLACE:

```
/* 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
```

WITH:

```
/* 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

#define vpiPackedArrayMember 655
```