Packed-Arrays and Related Properties & Methods

C. Berking 4/26/07

A) The Case For New Objects to Represent Packed-Arrays of Structs and Unions

The current VPI model has no separate objects to designate packed-arrays of variable-width net and variable objects. The implications of this are that the object net/variable types themselves must represent both scalar and/or multi-dimensional vector (packed-array) forms. For example:

```
logic logicElem;
logic [1:0] logic1dim;
logic [1:0][2:5] logic2dim;

bit bitElem;
bit [1:0] bit1dim;
bit [1:0][2:5] bit2dim;

struct packed { bit [7:0] b1; logic [2:0] r1 } structElem;
struct packed { bit [7:0] b1; logic [2:0] r1 } [1:0] struct1dim;
struct packed { bit [7:0] b1; logic [2:0] r1 } [1:0][2:5] struct2dim;
```

There is no alternative in 1800-2005 to representing the above declarations as follows:

logicElem, logic1dim, logic2dim:	vpiLogicVar (vpiReg)
bitElem, bit1dim, bit2dim:	vpiBitVar
structElem, struct1dim, struct2dim:	vpiStructVar

All the above objects with one or more dimensions have “vector-like” properties, since they are packed-arrays. However, their elements- as represented by their declarations without ranges present- are very different. The “logic” and “bit” vector types (“logic1dim”, “logic2dim”, “bit1dim”, and “bit2dim”) have elements that are always a single bit, whereas the struct types with ranges (“struct1dim”, “struct2dim”) have elements that are structs (vectors of width greater than 1 bit). Even though they can be treated as vectors (bit-indexed), they are not declared using ranges, i.e. the range of a struct variable is *implicit.* Packed-arrays of structs and unions are thus sufficiently different from bit and logic vector types to justify a new type of object- a hybrid of a vector type (vpiLogic or vpiBit) and an unpacked array type (vpiArrayVar).

The following new object types should be considered for adding to the standard to represent packed-arrays of struct and union variable and net types:

vpiPackedArrayVar
vpiPackedArrayNet

And their corresponding typespec object type:

vpiPackedArrayTypespec

The following transitions and properties that apply to vpiArrayVar and vpiArrayNet should apply similarly to these new types, with some implications as follows:

vpiSize	This property would return the total number of struct or union elements.
vpiVector	This property will always return '1' (true) for the new packed-array types (and thus vpiScalar would return '0' (false) accordingly).
vpiArrayMember	This property will return '1' (true) for all base-level struct variables of the packed-array objects (Q: What about intermediate packed-array levels ?)
vpiMember	This transition would *not* apply to the new packed-array objects directly, but would clearly be applicable to a fully-selected struct variable from the array**.**

B) Problem: What Are Packed-Array Elements and How Are They Accessed ?

Suggested Essential Terms for Purposes of Discussion
Slice An indexed contiguous part of an array that includes one or more dimensions itself, i.e.,from declaration "logic [1:0][2:5][6:8] myvar", a slice could be myvar[1], or myvar[0][2:3], etc. Note that this is not to be confused with the proposed 'vpiSlice' object, which is intended to represent only the latter form (i.e. with the part-select-like range).
Subpart A complete **slice**, i.e. with no explicit range specified or required, as in the first form described above, e.g. 'myvar[1]', or 'myvar[0][3]'.
Element Three potential ways to view this term have arisen from various sources:
Definition (1): [Traditional] The smallest indexable part of an array whose vpiParent is still the array;
Definition (2): [Traditional + Subarrays] In addition to #1, any **subpart**- i.e. any portion of an array that can be specified by indexing it (with vpiParent == the array).
Definition (3): [Incremental] Any of the largest **subpart**s of the array (i.e. any single-indexed member of its vpiParent array).
Definition (4): [New]The smallest unit of an array that has a datatype (implicit or user-defined) consistent with its declaration, i.e. with either all explicit *unpacked* ranges removed (if array is unpacked) or all explicit *packed* ranges removed.
For example, when arrays are constructed out of one or more typedefs, the element is the next lower **subpart** with a user-defined type. E.g.

```
typedef struct packed { logic [0:3] m4; bit [0:1] m2; } [3:0] pstype;
pstype [7:0] pstype2dim;  // An 8x4 array of structs.
// Element of 'pstype2dim' == pstype2dim[7] -> pstype2Dim[0]
// because their type is user-defined as 'pstype'.
```

1800 LRM HDL perspective:
Definition (1) applies for unpacked arrays, and Definition (2) applies for packed arrays (!).

1800 VPI data model:
Definition (1) applies for vpiReg and vpiVarSelect methods (iterations).

vpiElemTypespec is NOT yet defined; it has been suggested that definition #3 should apply. Or, should it be the same definition for vpiArrayTypespec as vpiReg and vpiVarSelect use for vpiArrayVar (definition #1) ?

Questions to resolve:

1) What definition should we use for vpiElemTypespec ? [How should it operate on vpiArrayTypespec and vpiPackedArrayTypespec objects]

2) What access method is appropriate for vpiPackedArrayVar/Net objects ?

3) Should we add a new iteration method to allow **subpart** access to vpiArrayVar's (unpacked arrays), and the new vpiPackedArrayVar/Net object (perhaps definition #3, but by another name) ? [Some suggestions: vpiElement, vpiChild, vpiSlice, vpiSubpart]

4) Or, if we choose element definition #4 (for whatever **element** access method is appropriate for #2), is this sufficient for access to any **subpart** that would be of interest ?

B) New Property Recommendations

Given the proliferation of new objects, and their possible multi-dimensional combinations, it is surprising that there is no property that relates directly to dimensionality. It seems grossly inefficient to require the application developer to create a vpiRange iteration and their respective handles to determine if there is one or more dimensions, or worse, determine how many.

The following two new properties are proposed to address this need:

vpiUnpackedDims	Returns number of *unpacked* dimensions of vpiArrayVar, vpiArrayNet, or vpiArrayTypespec
vpiPackedDims	Returns number of packed dimensions of any packed-array object (vpiLogicVar, vpiLogicNet, vpiBitVar, vpiPackedArrayVar, or vpiPackedArrayNet) or typespec (vpiLogicTypespec, vpiBitTypespec, or vpiPackedArrayTypespec). In addition, it is allowed on *unpacked* array objects (listed above) as well, in order to characterize packed-dimensions *underneath* unpacked ones.

Examples:

```
bit [1:0][2:5] mybit;                 // vpiPackedDims == 2

bit [1:0][2:5] mybundles [3:0];   //  vpiUnpackedDims == 1, and vpiPackedDims == 2

struct packed {
    bit [7:0] b1;
```

```
        logic [2:0] r1 } [1:0][2:5] struct2dim;    // vpiPackedDims == 2
```

Note that vpiPackedDims for ‘struct2dim’ does *not* include the *implicit* vector dimension of the struct itself