Editors Note: These changes are to be applied to P1364.

28.1 General

Decryption envelopes specify the pragma expressions for decrypting encrypted text regions. A decryption envelope begins in the source text when a **begin_protected** pragma expression is encountered. The end of the decryption envelope occurs at the point where an **end_protected** pragma expression is encountered. The **end_protected** pragma expression is said to close the envelope and shall be associated with the most recent **begin_protected** that has not already been closed. Decryption envelopes may contain other envelopes within their enclosed data block. The number of nested decryption envelopes that can be processed is implementation-specified, however that number shall be no less than 8. Code that is contained within a decryption envelope is said to be protected.

26.3.5 Object protection properties

All objects have a **vpiIsProtected** property, which is not shown in the data model diagrams.

-> IsProtected
bool: vpiIsProtected

Using **vpi_get(vpiIsProtected,** *object_handle***)** returns a boolean constant which indicates whether the object represents code contained in a decryption envelope. The **vpiIsProtected** property shall be TRUE if the *object_handle* represents code that is protected, otherwise FALSE. Unless otherwise specified, access to relationships and properties of a protected object shall be an error. Restrictions on access to complex properties are specified in the function reference descriptions for the corresponding VPI functions. Access to the **vpiType** property and the **vpiIsProtected** property of a protected object shall be permitted for all objects.

NOTE— Protected objects can be returned through object relationships, or by direct lookup using VPI functions that return handles.

26.6.19 Task and function call

8) System task and function calls which are protected shall allow iteration over the **vpiArgument** relationship.

26.6.26 Expressions

3) Expressions which are protected shall permit access to the **vpiSize** property.

27.6 vpi_get()

The VPI routine **vpi_get()** shall return the value of integer and boolean object properties. These properties shall be of type `PLI_INT32`. Boolean properties shall have a value of **1** for TRUE and **0** for FALSE. For integer object properties such as **vpiSize**, any integer shall be returned. For integer object properties that return a defined value, see **Annex G** for the value that shall be returned. Note for object property **vpiTimeUnit** or **vpiTimePrecision**, if the object is `NULL`, then the simulation time unit shall be returned. Unless otherwise specified, calling **vpi_get()** for a protected object shall be an error. Should an error occur, **vpi_get()** shall return **vpiUndefined**.

27.10 vpi_get_str()

The VPI routine **vpi_get_str()** shall return string property values. The string shall be placed in a temporary buffer that shall be used by every call to this routine. If the string is to be used after a subsequent call, the string should be copied to another location. Note that a different string buffer shall be used for string values returned through the `s_vpi_value` structure. Unless otherwise specified, calling **vpi_get_str()** for a protected object shall be an error.

27.16 vpi_handle()

The VPI routine **vpi_handle()** shall return the object of type *type* associated with object *ref*. Unless otherwise specified, calling **vpi_handle()** for a protected object shall be an error. The one-to-one relationships that are traversed with this routine are indicated as single arrows in the data model diagrams.

27.17 vpi_handle_by_index()

The VPI routine **vpi_handle_by_index()** shall return a handle to an object based on the index number of the object within the reference object, *obj*. The reference object shall be an object that has the **access by index** property. Unless otherwise specified, calling **vpi_handle_by_index()** for a protected object shall be an error. For example, to access a net bit, *obj* would be the associated net, to access an element of a reg array, *obj* would be the array. If the selection represented by the index number does not lead to the construction of a legal Verilog index select expression, the routine shall return a `null` handle.

27.18 vpi_handle_by_multi_index()

The VPI routine **vpi_handle_by_multi_index()** shall provide access to an index-selected sub-object of the reference handle. The reference object shall be an object that has the **access by index** property. Unless otherwise specified, calling **vpi_handle_by_multi_index**() for a protected object shall be an error. This routine shall return a handle to a valid Verilog object based on the list of indices provided by the argument *index_array*, and reference handle denoted by *obj*. The argument *num_index* shall contain the number of indices in the provided array index_array.

27.19 vpi_handle_by_name()

The VPI routine **vpi_handle_by_name()** shall return a handle to an object with a specific name. This function can be applied to all objects with a *fullname* property. The *name* can be hierarchical or simple. If *scope* is `NULL`, then *name* shall be searched for from the top level of hierarchy. If a scope object is provided, then search within that scope only. Unless otherwise specified, calling **vpi_handle_by_name()** with a protected scope object shall be an error. If the *name* is hierarchical and includes a protected scope, the call shall be an error.

27.21 vpi_iterate()

The VPI routine **vpi_iterate()** shall be used to traverse one-to-many relationships, which are indicated as double arrows in the data model diagrams. Unless otherwise specified, calling **vpi_iterate()** for a protected object shall be an error. The **vpi_iterate()** routine shall return a handle to an iterator, whose type shall be **vpiIterator**, which can used by **vpi_scan()** to traverse all objects of type *type* associated with object *ref*. To get the reference object from the iterator object use **vpi_handle(vpiUse, iterator_handle)**. If there are no objects of type *type* associated with the reference handle *ref*, then the **vpi_iterate()** routine shall return `NULL`.

Editors Note: These changes are to be applied to P1800. Please add the following notes, assigning the next available note number.

32.27 Task and function call

N+1) System task and function calls which are protected shall allow iteration over the **vpiArgument** relationship.

32.39 Expressions

N+1) Expressions which are protected shall permit access to the **vpiSize** property.