Proposal for Mantis Item #1477

37.12 Scope

5) A task func can have zero or more statements (see 13.3 and 13.4). If the number of statements is greater than 1, the vpiStmt relation shall return an unnamed begin that contains the statements of the task or function. If the number of statements is zero, the vpiStmt relation shall return NULL.

6) The vpiVirtualInterfaceVar iteration is supported only within elaborated contexts, and is not supported within lexical contexts such as class defns (<ref. to>37.27). If the scope declares an array of virtual interfaces, the vpiVirtualInterfaceVar iteration shall return each element of the array separately. However, the vpiVariables iteration shall return the array declaration as a single vpiArrayVar.
37.13 IO Declaration

REPLACE

2) A ref obj type handle may be returned for the vpiExpr of an io decl if it is passed by reference or if the io decl is an interface or a modport.

3) If the vpiExpr of an io decl is a ref obj and if the vpiActual of the ref obj is an interface or modport declaration, then the vpiDirection of the io decl shall be undefined.

WITH

2) A ref obj type handle may be returned for the vpiExpr of an io decl if it is passed by reference or if the io decl is an interface or a modport. If the io decl is a virtual interface, vpiExpr shall return a vpiVirtualInterfaceVar.

3) If the vpiExpr of an io decl is a ref obj and if the vpiActual of the ref obj is an interface or modport declaration, then the vpiDirection of the io decl shall be undefined. The vpiDirection shall also be undefined if the vpiExpr is a virtual interface var.

37.15 Reference Objects

REPLACE

<table>
<thead>
<tr>
<th>ref obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>-&gt; name</td>
</tr>
<tr>
<td>str: vpiName</td>
</tr>
<tr>
<td>str: vpiFullName</td>
</tr>
<tr>
<td>-&gt; virtual</td>
</tr>
<tr>
<td>bool: vpiVirtual</td>
</tr>
<tr>
<td>-&gt; generic</td>
</tr>
<tr>
<td>bool: vpiGeneric</td>
</tr>
<tr>
<td>-&gt; definition name</td>
</tr>
<tr>
<td>str: vpiDefName</td>
</tr>
</tbody>
</table>

WITH

<table>
<thead>
<tr>
<th>ref obj</th>
</tr>
</thead>
<tbody>
<tr>
<td>-&gt; name</td>
</tr>
<tr>
<td>str: vpiName</td>
</tr>
<tr>
<td>str: vpiFullName</td>
</tr>
<tr>
<td>-&gt; virtual</td>
</tr>
<tr>
<td>bool: vpiVirtual</td>
</tr>
<tr>
<td>-&gt; generic</td>
</tr>
<tr>
<td>bool: vpiGeneric</td>
</tr>
<tr>
<td>-&gt; definition name</td>
</tr>
<tr>
<td>str: vpiDefName</td>
</tr>
</tbody>
</table>
REPLACE

1) A ref obj represents a declared object or sub-element of that object that is a reference to an actual instantiated object. A ref obj exists for ports with ref direction, for an interface port, a modport port, or for formal task function ref arguments. The specific cases for a ref obj are:
   — A variable, named event, named event array that is the lowconn of a ref port
   — Any subelement expression of the above
   — A local declaration of an interface or modport passed through a port or any net, variable, named event, named event array of those
   — A virtual interface declaration in a class definition
   — A ref formal argument of a task or function, or sub-element expression of it

2) A ref obj may be obtained when walking port connections (lowConn, highConn), when traversing an expression that is a use of such ref obj, when accessing the virtual interface of a class, or when accessing the io decl of an instance or task or function.

WITH

1) A ref obj represents a declared object or sub-element of that object that is a reference to an actual instantiated object. A ref obj exists for ports with ref direction, for an interface port, a modport port, or for formal task function ref arguments. The specific cases for a ref obj are:
   — A variable, named event, named event array that is the lowconn of a ref port
   — Any subelement expression of the above
   — A local declaration of an interface or modport passed through a port or any net, variable, named event, named event array of those
   — A virtual interface declaration in a class definition
   — A ref formal argument of a task or function, or sub-element expression of it

2) A ref obj may be obtained when walking port connections (lowConn, highConn), when traversing an expression that is a use of such ref obj, when accessing the virtual interface of a class, or when accessing the io decl of an instance or task or function.
5) The `vpiVirtual` property shall return TRUE if the ref obj is a reference to a virtual interface and FALSE if the ref obj is a reference to an interface that is not a virtual interface. The `vpiVirtual` property shall return `vpiUndefined` for all other kinds of ref obj.

6) The `vpiGeneric` property shall return TRUE if the ref obj is a reference to a generic interface and FALSE if the ref obj is a reference to an interface that is not a generic interface. The `vpiGeneric` property shall return `vpiUndefined` for all other kinds of ref obj.

11) Variables of type `vpiArrayVar` or `vpiClassVar` do not have a value property. Struct var and union var variables for which the `vpiVector` property is FALSE do not have a value property.
Example 2: virtual interface declaration in a class definition:

```verilog
interface SBus; // A Simple bus interface
   logic req, grant;
   logic [7:0] addr, data;
endinterface

class SBusTransactor; // SBus transactor class
   virtual SBus bus; // virtual interface of type SBus
function new( virtual SBus s );
   bus = s; // initialize the virtual interface
endfunction

task request(); // request the bus
   bus.req <= 1'b1;
endtask

task wait_for_bus(); // wait for the bus to be granted
   @(posedge bus.grant);
endtask
endclass

module devA( Sbus s ); .... endmodule // devices that use SBus

module devB( Sbus s ); .... endmodule

module top;
   SBus s[1:4] (); // instantiate 4 interfaces
   devA a1( s[1] ); // instantiate 4 devices
   devB b1( s[2] );
   devA a2( s[3] );
   devB b2( s[4] );
   initial begin
      SbusTransactor t[1:4]; // create 4 bus-transactors and bind
      t[1] = new( s[1] );
      t[2] = new( s[2] );
      t[3] = new( s[3] );
      t[4] = new( s[4] );
   end
endmodule

A ref obj is returned for the left hand side expression of the statement “bus = s” in the constructor of the class definition SBusTransactor. The vpiName of that ref obj is “bus” and its vpiDefName is the name of the interface “SBus”. The vpiActual relationship returns the interface instance associated with that particular call to new after the assignment has executed. For example if it was “new ( s[1] )”, vpiActual would return the interface s[1]. If vpiActual is queried before the assignment is executed, the method may return NULL if the virtual “bus” interface is uninitialized. The right hand side expression also returns a ref obj which vpiActual is the interface instance passed to the call to new.

```
37.17 Variables

REPLACE

WITH

virtual interface var
37.23 Typespec

REPLACE

WITH

--- Diagram ---
ADD (after 37.26)

37.27 Virtual interface (NEW section- renumber sections following)

Details:

1) The vpiExpr relation shall return the interface instance assigned to the virtual interface in its declaration, if any; otherwise, vpiExpr shall return NULL.

2) A ref obj may be an interface expr only if it is a local declaration of an interface or modport passed through a port. A constant may be an interface expr only if it has a vpiConstType of vpiNullConst.

Example 1:

```vhdl
interface SBus #(parameter WIDTH=8);
    logic req, grant;
    logic [WIDTH-1:0] addr, data;
    modport phy (input addr, inout data);
endinterface

module top;

    parameter SIZE = 4;

    virtual SBus#(16) V16;
    virtual SBus#(32).phy V32_Array [1:SIZE];

endmodule
```

In this example, V16 is a virtual interface, while V32_Array is an array var. The vpiVariables iteration from module top includes both V16 and V32_Array, while the vpiVirtualInterfaceVar iteration returns
V16 together with the individual elements of V32_Array, that is, V32_Array[1] through V32_Array[4].

Example 2: Virtual interface declaration in a class definition:

```verilog
interface SBus; // A Simple bus interface
  logic req, grant;
  logic [7:0] addr, data;
endinterface

class SBusTransactor; // SBus transactor class
  virtual SBus bus; // virtual interface of type SBus
function new( virtual SBus s );
  bus = s; // initialize the virtual interface
endfunction
task request(); // request the bus
  bus.req <= 1'b1;
endtask
task wait_for_bus(); // wait for the bus to be granted
  @(posedge bus.grant);
endtask
endclass

module devA( SBus s ); ... endmodule // devices that use SBus

module devB( SBus s ); ... endmodule

module top;
  SBus s[1:4] (); // instantiate 4 interfaces
  devA a1( s[1] ); // instantiate 4 devices
  devB b1( s[2] );
  devA a2( s[3] );
  devB b2( s[4] );
  initial begin
    SBusTransactor t[1:4]; // create 4 bus-transactors and bind
    t[1] = new( s[1] );
    t[2] = new( s[2] );
    t[3] = new( s[3] );
    t[4] = new( s[4] );
  end
endmodule
```

A virtual interface var is returned for the left hand side expression of the statement “bus = s” in the constructor of the class definition SBusTransactor. The vpiName of the virtual interface var is “bus”, and it has a vpiInterfaceTypespec for which the vpiDefName is “SBus”. The vpiActual relationship returns the interface instance associated with that particular call to new after the assignment has executed. For example if it was “new(s[1])”, vpiActual would return the interface s[1]. If vpiActual is queried before the assignment is executed, the method shall return NULL if the virtual interface is uninitialized. In addition, the right-hand side expression of “bus = s” returns a virtual interface var for which vpiActual is the interface instance passed to the call to new.
ADD (after 37.26 and new Virtual Interface section)

37.28 Interface typespec (NEW section- renumber following sections)

Details:

1) The vpiDefName of an interface typespec that represents a modport shall be the mod port identifier. The vpiDefName of an interface typespec that represents an interface shall be the identifier of the interface declaration.

2) For an interface typespec that represents a modport, vpiParent shall return an interface typespec of the corresponding interface. For an interface typespec that represents an interface, vpiParent shall return NULL.

3) In the example below, the first typedef defines an interface typespec corresponding to “virtual SBus#(16)” whose vpiName is SB16. The vpiDefname of this typespec shall be SBus, and the assigned parameter value of 16 shall be derived by iterating on vpiParamAssign. The typedef SBphy, however, is an array typespec for which the vpiElemTypespec returns an interface typespec corresponding to “virtual SBus#(32).phy”.

The vpiTypedef iteration from the module top returns handles to both SB16 and SBphy interface typesspecs.

```vhdl
interface SBus #(parameter WIDTH=8);
  logic req, grant;
  logic [WIDTH-1:0] addr, data;
  modport phy (input addr, inout data);
endinterface

module top;

  parameter SIZE = 4;

type virtual SBus#(16) SB16;
type virtual SBus#(32).phy SBphy [1:SIZE];
  ...
endmodule
```
37.27 Class definition (section # reflects original numbering)

6) The relation to `vpiExtends` exists whenever one class is derived from another class (refer to 8.12). The relation from extends to class typespec provides the base class. The `vpiArgument` iterator from extends shall provide the arguments used in constructor chaining (refer to 8.16).

7) The `vpiInterfaceDecl` iteration returns the virtual interface declarations in the class definition.
37.28 Class typespec *(section # reflects original numbering)*

REPLACE

WITH

REPLACE

2) For a class typespec that represents only a lexical construct, the one-to-many relations vpiVariables, vpiMethods, vpiConstraint, vpiInterfaceDecl, vpiNamedEvent, vpiNamedEventArray, vpiTypedef, and vpiInternalScope are not supported.

WITH

2) For a class typespec that represents only a lexical construct, the one-to-many relations vpiVariables, vpiMethods, vpiConstraint, vpiInterfaceDecl, vpiNamedEvent, vpiNamedEventArray, vpiTypedef, and vpiInternalScope are not supported.
REPLACE

10) The **vpiClassTypespec** iteration from a class defn shall return the class specializations derived directly (and not by inheritance) from that class defn.

11) The **vpiInterfaceDecl** iteration shall return the virtual interface declarations in the class specialization.

WITH

10) The **vpiClassTypespec** iteration from a class defn shall return the class specializations derived directly (and not by inheritance) from that class defn.

11) The **vpiInterfaceDecl** iteration shall return the virtual interface declarations in the class specialization. The **vpiVirtualInterfaceVar** iteration (formerly **vpiInterfaceDecl** - now deprecated in this standard) shall return the virtual interface var declarations in the class specialization (see 37.12 detail 6 (new) ). If an array of virtual interfaces is declared, the **vpiVirtualInterfaceVar** iteration shall return each element of the array separately. However, the **vpiVariables** iteration shall return the array declaration as a single **vpiArrayVar**.
37.29 Class variables and class objects (section # reflects original numbering)

**REPLACE**

6) From a class obj, the iterations over `vpiVariables`, `vpiMethods`, `vpiNamedEvent`, and `vpiNamedEventArray` shall return both static and automatic properties or methods. However, the iteration over `vpiMethods` shall not include built-in methods for which there is no explicit declaration.

7) The `vpiInterfaceDecl` iteration returns the virtual interfaces of the class object.

**WITH**

6) From a class obj, the iterations over `vpiVariables`, `vpiMethods`, `vpiNamedEvent`, and `vpiNamedEventArray` shall return both static and automatic properties or methods. However, the iteration over `vpiMethods` shall not include built-in methods for which there is no explicit declaration.

7) The `vpiInterfaceDecl` iteration returns the virtual interfaces of the class object. The `vpiVirtualInterfaceVar` iteration (formerly `vpiInterfaceDecl` - now deprecated in this standard) shall return the virtual interface var declarations in the class object. If an array of virtual interfaces is declared, the `vpiVirtualInterfaceVar` iteration shall return each element of the array separately. However, the `vpiVariables` iteration shall return the array declaration as a single `vpiArrayVar`. 
37.38 Task and function call

REPLACE

WITH
37.44 Clocking block

REPLACE

WITH

\begin{center}
\begin{tikzpicture}
  \node[draw,rounded corners] (clocking) {clocking block};
  \node[draw,rounded corners, dashed] (instance) at (clocking.east) {$instance$};
  \node[draw,rounded corners] (clocking_io) at (instance.south west) {clocking io decl};
  \node[draw,rounded corners] (property) at (clocking_io.south west) {property decl};
  \node[draw,rounded corners] (sequence) at (property.south west) {sequence decl};

  \draw[->] (clocking) -- (clocking_io);
  \draw[->] (clocking) -- (property);
  \draw[->] (clocking) -- (sequence);
  \draw[->] (clocking_io) -- (instance);
  \draw[->] (property) -- (instance);
  \draw[->] (sequence) -- (instance);

  \node[draw,rounded corners] (vpiPrefix) at (instance.south west) {$vpiPrefix$};
  \node[draw,rounded corners] (virtual_interface_var) at (vpiPrefix.south west) {virtual interface var};
  \node[draw,rounded corners] (vpiActual) at (virtual_interface_var.south west) {$vpiActual$};
  \node[draw,rounded corners] (clocking_block) at (vpiActual.south west) {clocking block};

  \draw[->] (instance) -- (vpiPrefix);
  \draw[->] (instance) -- (virtual_interface_var);
  \draw[->] (instance) -- (clocking_block);
\end{tikzpicture}
\end{center}
REPLACE

Details:

1) The methods, \texttt{vpiInputSkew} and \texttt{vpiOutputSkew}, and properties \texttt{vpiInputEdge} and \texttt{vpiOutputEdge}, on the clocking block apply to the default constructs. The same methods and properties on the clocking io decl apply to the clocking io decl itself.

2) \texttt{vpiExpr} shall return the expression or ref obj referenced by the clocking io decl. Consider input \texttt{enable = top.mem1.enable}. Here, “enable” is represented by a clocking io decl, and the \texttt{vpiExpr} relation returns a handle to “\texttt{top.mem1.enable}”.

WITH

Details:

1) The methods, \texttt{vpiInputSkew} and \texttt{vpiOutputSkew}, and properties \texttt{vpiInputEdge} and \texttt{vpiOutputEdge}, on the clocking block apply to the default constructs. The same methods and properties on the clocking io decl apply to the clocking io decl itself.

2) The \texttt{vpiPrefix} relation shall be non-NULL when the clocking block represents an expression in the SystemVerilog source code immediately prefixed by a virtual interface.

3) If a prefix of a clocking block is a virtual interface that has no value at the current simulation time, the \texttt{vpiActual} relation shall return NULL.

4) \texttt{vpiExpr} shall return the expression or ref obj referenced by the clocking io decl. Consider input \texttt{enable = top.mem1.enable}. Here, “enable” is represented by a clocking io decl, and the \texttt{vpiExpr} relation returns a handle to “\texttt{top.mem1.enable}”.
37.55 Expressions

REPLACE

WITH
ADD (after 37.56)

### 37.57 Dynamic prefixing (NEW section)

Details:

1) The `vpiPrefix` relation shall be non-NULL when the object represents an expression or task call in the SystemVerilog source code prefixed by a virtual interface or a clocking block, or when the object is all or part of a non-static class property prefixed by a class var.

2) The memory allocation scheme value for an object for which a class var or virtual interface var `vpiPrefix` is non-NULL shall be the same as for the prefix.

3) The property `vpiHasActual` shall return TRUE
   — whenever the prefix object has a corresponding actual at the current simulation time.
   — if the object is all or part of a statically declared object in an elaborated context.
   — if the object is part or all of an automatically allocated variable obtained from a frame (§37.39).

   The property `vpiHasActual` shall return FALSE
   — whenever the prefix object has no corresponding actual at the current simulation time.
   — if the object is obtained from a lexical context, such as from a class defn (§37.27).
   — if the object is part or all of a nonstatic class property variable referenced relative to its class typespec (§37.28).
   — if the object is part or all of a automatically allocated variable obtained from a task or function declaration (§37.37).
37.59 Assignment

REPLACE

WITH

interface expr
C.4 Constructs identified for deprecation

ADD

C.4.3 VPI definitions

Certain object, relationship, and property definitions have been deprecated to implement corrections and improvements to VPI. Some have been inherited from IEEE Std 1364 (see 36.12.1) and some have been changed or removed to maintain consistency with related improvements, as follows:

1) vpiMemory (as an object type)

The vpiArrayVar (vpiRegArray) object type has been generalized to include vpiMemory and all other arrays of variables. vpiMemory therefore no longer represents a VPI object type, except under certain backwards compatibility modes (see 36.12.1). However, it is still in use as a transition (see 37.20 diagram and detail 1).

2) vpiMemoryWord (as an object type)

This was exclusively used to represent elements of vpiMemory objects in IEEE Std 1364. Since vpiArrayVar (vpiRegArray) has replaced the definition of vpiMemory, and variable object types now represent their elements, this is represented by vpiLogicVar (vpiReg) types. Therefore, it no longer represents a VPI object type, except under certain backwards compatibility modes (see 36.12.1). It is still in use as a transition, however (see 37.20 diagram and detail 1).

3) vpiArray property

In IEEE Std 1364, variable types vpiIntegerVar, vpiTimeVar, and vpiRealVar could represent single variable objects or arrays of those objects. The vpiArray property was required to distinguish those cases (the property returned TRUE when they were arrays). Also, the property indicated when vpiReg types represented elements of vpiRegArrays. These two uses became conflicting and unnecessary when vpiRegArrays and arrays of integer, time, and real variables were generalized as vpiArrayVar (vpiRegArray) objects. To distinguish when any variable is an element of an array, the vpiArrayMember property is now used, thus replacing the original use of vpiArray for reg types. The vpiArray property now has only limited use in IEEE 1364 backwards compatibility modes when vpiIntegerVar, vpiTimeVar, and vpiRealVar could represent arrays (see 36.12.1).

4) vpiValid property

Significant revisions to VPI have rendered the original vpiValid property inconsistent with its original purpose, which was to determine the extent to which a transient object represented by a VPI handle was active or “alive” (see 37.2.4 and 37.3.7). Since the VPI object model no longer supports maintaining handles to objects whose lifetimes have ended, such “validity” is implicit in their existence, and their status must be determined by other means (see 38.36.1).

5) vpiInterfaceDecl one-to-many relationship

This relationship was used to return vpiRefObj objects representing virtual interface variables from any scope. Its definition has been made equivalent to vpiVirtualInterfaceVar, which instead returns vpiVirtualInterfaceVar object types. This was done to correctly reflect the true variable-like characteristics of these objects (see 37.28 (adjust to new section #) detail 11).
**M.2 Source code** [appendix M, sv_vpi_user.h ]

**REPLACE:**

```c
#define vpiPackedArrayVar 623
```

**WITH:**

```c
#define vpiPackedArrayVar 623
#define vpiVirtualInterfaceVar 728
```

**REPLACE:**

```c
#define vpiInterfaceDecl 728
```

**WITH:**

```c
#define vpiInterfaceDecl vpiVirtualInterfaceVar /* interface decl deprecated */
```

**REPLACE:**

```c
#define vpiVirtual 635
```

**WITH:**

```c
#define vpiVirtual 635
#define vpiHasActual 636
```

**REPLACE:**

```c
/*********************** task/function properties ***********************
#define vpiOtherFunc 6 /* returns other types; for property vpiFuncType */
// vpiValid, vpiValidTrue, vpiValidFalse are deprecated in 1800-2009 */

/******************** value for vpiValid ********************
#define vpiValidUnknown 2 /* Validity of variable is unknown */
```

**WITH:**

```c
/*********************** task/function properties ***********************
#define vpiOtherFunc 6 /* returns other types; for property vpiFuncType */

/******************** value for vpiValid ********************
#define vpiValidUnknown 2 /* Validity of variable is unknown */
```