VPI Extensions to SystemVerilog

January 12, 2004
NOTES
1. Top-level instances shall be accessed using `vpi_iterate()` with a NULL reference object.
2. Passing a NULL handle to `vpi_get()` with types `vpiTimePrecision` or `vpiTimeUnit` shall return the smallest time precision of all instances in the design.
3. If an instance is an element within an array, the `vpiIndex` transition is used to access the index within the array. If the instance is not part of an array, this transition shall return NULL.
**Interface**

![Diagram of interface relationship]

**NOTE**
All interfaces are instances and all relations and properties in the Instances diagram also apply.

**Program**

![Diagram of program relationship]

**NOTE**
All programs are instances and all relations and properties in the Instances diagram also apply.
Module (26.6.1)

NOTE to reviewers leave all relations here, as the arrow back is different than in the instance diagram ie most items have two possible container relations:

vpiModule
vpiInstance

NOTES

1. **vpiModule** will return a module if the object is inside a module instance, otherwise NULL;

2. **vpiInstance** will always return the immediate instance (package, module, program or interface) in which the object is instantiated
Interface tf decl

NOTE

`vpiIterate(vpiTaskFunc)` can return more than one task/function declaration for modport tasks/functions with an access type of `vpiForkJoin`, because the task or function can be imported from multiple module instances.
Ports (26.6.5)

- `vpiConnByName`: if true, the port is connected by name.
- `vpiDirection`: the direction of the port.
- `vpiExplicitName`: if true, the port is explicitly named.
- `vpiPortIndex`: the index of the port.
- `vpiName`: the name of the port.
- `vpiPortType`: the type of the port.
- `vpiScalar`: if true, the port is scalar.
- `vpiSize`: the size of the port.
- `vpiVector`: if true, the port is vector.

**NOTES**

1. `vpiPortType` shall be one of the following three types: `vpiPort`, `vpiInterfacePort`, and `vpiModportPort`. Port type depends on the formal, not on the actual.

2. `vpi_get_delays`, `vpi_put_delays` delays shall not be applicable for `vpiInterfacePort`.

3. `vpiHighConn` shall indicate the hierarchically higher (closer to the top module) port connection.

4. `vpiLowConn` shall indicate the lower (further from the top module) port connection.

5. `vpiLowConn` of a `vpiInterfacePort` shall always be `vpiRefObj`.

6. Properties scalar and vector shall indicate if the port is 1 bit or more than 1 bit. They shall not indicate anything about what is connected to the port.

7. Properties index and name shall not apply for port bits.

8. If a port is explicitly named, then the explicit name shall be returned. If not, and a name exists, then that name shall be returned. Otherwise, NULL shall be returned.

9. `vpiPortIndex` can be used to determine the port order. The first port has a port index of zero.

10. `vpiHighConn` and `vpiLowConn` shall return NULL if the port is not connected.

11. `vpiSize` for a null port shall return 0.
Ref Obj

These objects are newly defined objects needed for supporting the full connectivity through ports where the ports are vpiInterface or vpiModport or any object inside modport or interface.

RefObjs are dummy objects and they always have a handle to the original object.

```vpi
define
interface simple ()
logic req, gnt;
modport slave (input req, output gnt);
modport master (input gnt, output req);
}
define module top()
interface simple i;
child1 i1(i);
child2 i2(i.master);
```

NOTES

1. `vpiRefObjType` of `vpiRefObj` can be one of the following types:
   - `vpiInterface`
   - `vpiModport`
   - `vpiNet`
   - `vpiReg`
   - `vpiVariable`

12. `vpiPort` and `vpiPortInst` is defined only for `vpiRefObj` where `vpiRefObjType` is `vpiInterface`.

Examples

These objects are newly defined objects needed for supporting the full connectivity through ports where the ports are vpiInterface or vpiModport or any object inside modport or interface.

RefObjs are dummy objects and they always have a handle to the original object.
Accellera

class child1(interface simple s)
endclass

beginmodule
endmodule

/****************************
for port of i1, 
    vpiHighConn = vpiRefObj where vpiRefObjType = vpiInterface

for port of i2 ,
    vpiHighConn =  vpiRefObj  where vpifullType = vpiModport

**************************************************************************/

class child1(interface simple s)
    cl c_1(s);
    cl c_2(s.master);
endclass

endmodule

/****************************
for port of child1, 
    vpiLowConn = vpiRefObj where vpiRefObjType = vpiInterface

for that refObj, 
    vpiPort is  = port of child1.
    vpiPortInst is  = s, s.master
    vpiInterfaceConn  is  = i.

for port of c_1 :
    vpiHighConn is a vpiRefObj, where full type is vpiInterface.

for port of c_2 :
    vpiHighConn is a vpiRefObj, where full type is vpiModport.
blockVariable (26.6.8)
NOTES
1. A var select is a word selected from a variable array.
2. The boolean property vpiArray shall be TRUE if the variable handle references an array of variables, and FALSE otherwise. If the variable is an array, iterate on vpiVarSelect to obtain handles to each variable in the array.
3. vpi_handle (vpiIndex, var_select_handle) shall return the index of a var select in a 1-dimensional array. vpi_iterate (vpiIndex, var_select_handle) shall return the set of indices for a var select in a multidimensional array, starting with the index for the var select and working outward.
4. vpiLeftRange and vpiRightRange shall apply to variables when vpiArray is TRUE, and represent the array range declaration. These relationships are only valid when vpiArray is TRUE.
5. vpiSize for a variable array shall return the number of variables in the array. For non-array variables, it shall return the size of the variable in bits.
6. vpiSize for a var select shall return the number of bits in the var select. This applies only for packed var select.
7. Variables whose boolean property vpiArray is TRUE do not have a value property.
8. vpiBit iterator applies only for logic, bit, packed struct, and packed union variables.
9. vpiIndexType is valid only for associative array.
10. cbSizeChange will be applicable only for dynamic and associative arrays. If both value and size change, the size change callback will be invoked first. This callback fires after size change occurs and before any value changes for that variable. The value in the callback is new size of the array.
11. vpiRandType, vpiRand, vpiRandC, and vpiNotRand add a property to return randomization.
12. vpiIsRandomized adds a property to determine whether a random variable is currently active for randomization.
13. Variable bit may have the same meaning and semantics as bit in 26.6.7. Variable bit relation is available only for logic, bit, and packed structure variables.
14. Note that:
   logic var == reg
   var bit var == reg bit
   array var == reg array
NOTES

1. Typespec to typespec relation is used when the \texttt{vpiTypedefType} is "\texttt{vpiTypedef}"; which will be the case for type aliases, for example, \texttt{typedef a b};

2. If the type of a type is \texttt{vpiStruct} or \texttt{vpiUnion}, then you can iterate over numbers to obtain the structure of the user-defined type. For each member the typespec relation from the member will detail its type.

3. The name of a typedef may be the empty string if the typedef is representing the type of a typedef field defined inline rather than via a typedef. For example:

   ```plaintext
   typedef struct {
       struct
           int a;
   } B;
   } C;
   ```
The typedef C has `vpiTypedefType vpiStruct`, a single field named B with `vpiTypedefType vpiStruct`. Obtaining the typedef of field B, you will obtain a typedef with no name and a single field, named "a" with `vpiTypedefType` of `vpiInt).

**Variable Drivers and Loads**

NOTES

1. **vpiDrivers/Loads** for a structure, union, or class variable will include the following:
   - Driver/Load for the whole variable
   - Driver/Load for any bit/part select of that variable
   - Driver/Load of any member nested inside that variable

2. **vpiDrivers/Loads** for any variable array should include the following:
   - Driver/Load for entire array/vector or any portion of an array/vector to which a handle can be obtained.
Instance Arrays (26.6.2)

NOTE

Param assignments can only be obtained from non-primitive instance arrays.
NOTE

1: Unnamed scopes shall have valid names, though tool dependent.

2: The vpiImport iterator shall return all objects imported into the current scope via import statements. Note that only objects actually referenced through the import shall be returned, rather than items potentially made visible as a result of the import. Refer to section 18.2.2 for more details.
NOTE

vpiDirection returns vpiRef for pass by ref ports.

clocking block
Class Object Definition

NOTE

1. **ClassDefn** handle is a new concept. It does not correspond to any **vpiUserDefined** (class object) in the design. Rather it represents the actual type definition of a class.

2. Should not call **vpi_get_value/vpi_put_value** on the non-static variables obtained from the class definition handle.

3. Iterator to constraints returns only normal constraints and not inline constraints.

4. To get constraints inherited from base classes, you will need to traverse the class relation to the parent.
Constraint

Constraint

vpiParent

class

> virtual
  bool: vpiVirtual
  --> lifetime (static/automatic)
  int: vplLifetime
  --> extern
  bool: vpiExtern
  --> name
  str: vpiName
  str: vpiFullName
  --> active
  bool: vpIsConstraintEnabled

constraint

constraint item

constraint ordering

constraint dist

constraint expr

constraint expr

constraint expr

expr

expr

expr

dist item
Accellera

**dist item**

- `vpiLeftRange`
- `vpiRightRange`
- `vpiWeight`

-> operation type ( := or :/ )

- int: `vpiOpType`
- `vpiDistType`
  - `vpiEqualDist`
  - `vpiDivDist`

**constraint expr**

- `implication`
- `constr if`
- `constr if else`

- `vpiCondition`
- `expr`

- `vpiElseConst`
- `expr`

- `constraint expr`
Class Variables

NOTES

1. **vpiWaiting/Process** iterator on mailbox/semaphores will show the processes waiting on the object:
   - Waiting process means either frame or task/function handle.
2. **vpiMessage** iterator shall return all the messages in a mailbox.
3. **vpiClassDefn** returns the ClassDefn which was used to create the handle.
4. **vpiActualDefn** returns the ClassDefn that handle object points to when the query is made.
5. **vpiClassDefn/vpiActualDefn** both shall return NULL for built-in classes.
Accellera

Structure/Union

> struct var
> union var
> vpiParent
> variables

-> definition name
  str: vpiDefName
-> packed
  bool: vpiPacked

NOTES

\texttt{vpi\_get\_value/vpi\_put\_value} cannot be used to access values of entire unpacked structures and unpacked unions.
Named Events

NOTE
The new iterator (vpiWaitingProcess) returns all waiting processes, identified by their frame, for that namedEvent.

NOTE
vpi_iterate(vpiIndex, named_event_handle) shall return the set of indices for a named event within an array, starting with the index for the named event and working outward. If the named event is not part of an array, a NULL shall be returned.
Task, Function Declaration (26.6.18)

NOTE
1. A Verilog HDL function shall contain an object with the same name, size, and type as the function.
2. \texttt{vpiInterfaceTask/vpiInterfaceFunction} shall be true if task/function is declared inside an interface or a modport of an interface.
3. For function where return type is a user-defined type, \texttt{vpi_handle} (\texttt{vpiReturn,Function_handle}) shall return the implicit variable handle representing the return of the function from which the user can get the details of that user-defined type.
4. \texttt{vpiReturn} will always return a var object, even for simple returns.
Alias Statement

Examples

alias a=b=c=d

Results in 3 aliases:

alias a=d
alias b=d
alias c=d

d is Rhs for all.
NOTES
1. The following callbacks shall be supported on frames:
   — cbStartOfFrame: triggers whenever any frame gets executed.
   — cbEndOfFrame: triggers when a particular thread is deleted after all storage is deleted.

Comment to editors: Please note that we have changed the vpiParent handle from the LRM. vpiOrigin now gives the originating scope or task/function call.
The following callbacks shall be supported on threads

- **cbStartOfThread**: triggers whenever any thread is created
- **cbEndOfThread**: triggers when a particular thread gets deleted after storage is deleted.
- **cbEnterThread**: triggers whenever a particular thread resumes execution
Accellera

NOTE

Clocking event is always the actual clocking event on which the assertion is being evaluated, regardless of whether this is explicit or implicit (inferred)
Variables are declarations of property variables. You cannot get the value of these variables.

Note that the sequence bubble will be as already drawn in this diagram, but only one of them.
Accellera
int: vpiSeqOpType is one of:

and, intersect, or,
first_match,
throughout, within,
##,
[*], [*=], [*->]
vpiAttribute

-> name
  str: vpiName

-> On definition
  bool: vpiDefAttribute

-> value:
  vpi_get_value()

definition location
  str: vpiDefFile
  int: vpiDefLineNo

instances

port
net
reg
memory

named event
prim term
path term
mod path
tchk

param assign
spec param
taskfunc
variables
primitive

table entry
stmt
process
operation

concurrent assertions

sequence decl
property decl
clocking block
class defn
constraint
### vpi_handle_by_name()

**Synopsis:**
Get a handle to an object with a specific name.

**Syntax:**
```
vpi_handle_by_name(name, refhandle)
```

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>vpiHandle</td>
<td>Handle to an object</td>
</tr>
</tbody>
</table>

**Arguments:**

<table>
<thead>
<tr>
<th>Type</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PLI_BYTE8 *</td>
<td>name</td>
<td>A character string or pointer to a string containing the name of an object</td>
</tr>
<tr>
<td>vpiHandle</td>
<td>refhandle</td>
<td>Can be a HDI scope or a typedefinition object or a class/structure/union instance handle</td>
</tr>
</tbody>
</table>

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. The name should be searched in the refHandle provided.