VPI Extensions to SystemVerilog

December 19, 2003
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 modules and packages in the instantiated design.

3. If a module is an element within a module array, the `vpiIndex` transition is used to access the index within the array. If a module is not part of a module array, this transition shall return NULL.
Interface

```
interface
  vpilInstance
  vpiDefaultClocking
  clocking domain
  mod port
  interface
  interface array
  cont assign
```

Program

```
program
  vpilInstance
  vpiDefaultClocking
  clocking domain
  cont assign
```
Module (26.6.1)

NOTE to reviewers leave all relations here, as the arrow back is different than in the instance diagram i.e. 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 instance (package, module, program or interface) that is the container.
Accellera

Modport

![Diagram of Modport]

Interface tf decl

![Diagram of 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.
Accellera

ModPort Ports

NOTES
1. For simple port declaration inside a modport, the HighComm represents the signal in the interface (type expr).
2. For hierarchical port declaration, the HighComm will be a RefObj of type vpiModPort.
3. Direction for hierarchical ports should be vpiUndefined.
Ports (26.6.5)

-> connected by name
   bool: vpiConnByName
-> delay (mipd)
   vpi_get_delays()
   vpi_put_delays()
-> direction
   int: vpiDirection
-> explicitly named
   bool: vpiExplicitName
-> index
   int: vpiPortIndex
-> name
   str: vpiName
-> port type
   int: vpiPortType
-> scalar
   bool: vpiScalar
-> size
   int: vpiSize
-> vector
   bool: vpiVector

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.

```vhdl
interface simple ()
logic req, gnt;
modport slave (input req, output gnt);
modport master (input gnt, output req);
}
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.
endmodule

/*******************************************************************************/
for port of i1,
    vpiHighConn = vpiRefObj where vpiRefObjType = vpiInterface
for port of i2 ,
    vpiHighConn = vpiRefObj where vpifullType = vpiModport
*******************************************************************************/

module child1(interface simple s)
    cl c_1(s);
    cl c_2(s.master);
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.
Variable (26.6.8)

- **ports**
  - **vpiLowConn**
  - **vpiHighConn**

- **scope**
  - **expr**
  - **vpiLeftRange**
  - **vpiRightRange**

- **variables**
  - **logic var**
  - **int var**
  - **integer var**
  - **short int var**
  - **long int var**
  - **byte var**
  - **struct var**
  - **union var**
  - **bit var**
  - **enum var**
  - **real var**
  - **time var**
  - **short real var**
  - **class**
  - **string var**
  - **variable bit**

- **var select**
  - Constant selection
    - **bool: vpiConstantSelect**
      - **name**
        - **str: vpiName**
        - **str: vpiFullName**
      - **valid**
        - **vpiValid**
      - **size**
        - **int: vpiSize**
      - **value**
        - **vpi_get_value()**
        - **vpi_put_value()**

- **vpiIndex**

- **typedef**

For type equivalence rules, see 7.15
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.
Accellera

typedef instances

-> type of typedef
  -> range

int: vpiTypedefType
all the variable types:
  vpiLogic, vpiInt, ..., vpiString,
  plus vpiTypedef

-> name

field

NOTES
1. Typedef to typedef relation is used when the vpiTypedefType is "vpiTypedef", which will be the case for type aliases, for example, typedef a b;

2. If the type of a type is vpiStruct or vpiUnion, then you can iterate over field to obtain the structure of the user-defined type. For each field, if the field is vpiStruct or vpiUnion, the typedef relation from the field 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:

```c
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

Unnamed scopes shall have valid names, though tool dependent.
IO declaration (26.6.4)

NOTE

vpiDirection returns vpiRef for pass by ref ports.

clocking domain

expr

vpiDefInputSkew vpiDefOutputSkew

event control
delay control

clocking domain

vpiClockingEvent

event control

-> name

str: vpiName
str: vpiFullName

concurrent assertion item

clocking i/o decl

expr

clocking i/o decl

-> vpiDirection
-> vpiName
-> vpiDefault Skew
bool
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
dist item

- vpiLeftRange
- vpiRightRange
- vpiWeight

\[ \Rightarrow \text{operation type ( := or :/ )} \]
\[ \text{int: vpiOpType} \]
\[ \Rightarrow \text{vpiDistType} \]
\[ \text{vpiEqualDist} \]
\[ \text{vpiDivDist} \]

constraint expr

- implication
- constr if
- constr if else

\[ \text{expr} \]
\[ \text{constraint expr} \]
\[ \text{vpiCondition} \]
\[ \text{expr} \]
Accellera

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.
Structure/Union

- struct var
- union var

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

NOTES

vpi_get_value/vpi_put_value cannot be used to access values of entire unpacked structures and unpacked unions.

Enum, Enum Constant

- enum var
- enum const

-> vpi_get_value
-> name
  str: vpiName
-> typed
  int: vpiType
Named Events

NOTE

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

NOTE

vpi_iterate(vpilIndex, 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. `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, `vpi_handle` (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. `vpiReturn` will always return a var object, even for simple returns.
Examples

alias a=b=c=d

Results in 3 aliases:

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

d is Rhs for all.
Accellera

Frames (26.6.20)

- task
- task call
- function
- function call

- scope
- vpiOrigin
- frame
- vpiThread
- thread
- vpiAutomatic
- reg
- reg array
- variables
- named event
- named event array
- stmt

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.
Threads

NOTES

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
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)
**disable condition**

Definition location:
- `int: vpiDefLineNo`
- `str: vpiDefFile`

**clocking event**

Name (clocking identifier):
- `str: vpiName`
- `str: vpiFullName`

Definition location:
- `int: vpiDefLineNo`
- `str: vpiDefFile`

Inferred or declared:
- `bool: vpiInferred`

**property decl**

Name:
- `str: vpiName`
- `str: vpiFullName`

Definition location:
- `str: vpiDefFile`
- `int: vpiDefLineNo`
Accellera

property spec

definition location
int: vpiDefLineNo
str: vpiDefFile

disable condition

variables

expr

NOTE
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.

property expr

vpiOperand

--> property operation
int: vpiPropertyOpType
can be: vpiNot, vpiImply, vpiDelayedImply.
sequence decl

definition location
- str: vpiDefFile
- int: vpiDefLineNo

block identifier
- str: vpiName
- str: vpiFullName

variables

sequence spec

Sequence

sequence expr

Sequence

sequence expr

Sequence

multiclock

sequence expr

actual arg expr

expr

connected by name
- bool: vpiConnectByName

explicitly named
- bool: vpiExplicitName

argument index
- int: vpiPortIndex

name
- str: vpiName

formal list item

identifier

event control
sequence decl

sequence inst

sequence expr

operation

operation type

int: vpiSeqOpType

vpiOperand

sequence expr

expr

actual arg expr

assignment

int: vpiSeqOpType is one of:

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

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 domain

Class defn

Constraint

VpiAttribute

-> name
  str: vpiName

-> On definition
  bool: vpiDefAttribute

-> value:
  vpi_get_value()

definition location

str: vpiDefFile
int: vpiDefLineNo
Accellera