VPI Extensions to SystemVerilog

November 2003

SYNOPSYS®
NOTES
1. Top-level modules 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 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.

With SystemVerilog, $root is always the top-level module. To get all modules in $root, the user must do vpi_iterate (vpiModuleRoot_handle).

Modport
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.

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

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 and vpiModPortPort.

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 or a vpiModPortPort 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.
RefObj

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.

```verilog
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);
```
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

Variable bit here may have the same meaning and semantics as bit in 26.6.7
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 array if both value and size change, size changes cb first.
Variable

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 the whole array, part select, or single element.

Instance Arrays (26.6.2)
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**

- `vpiClocking`
- `event control`
- `delay control`
- `vpiDefInputSkew` `vpiDefOutputSkew`
- `clocking domain`
- `clocking i/o decl`
- `expr`
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.
Constraint

> virtual
  - bool: vpiVirtual
  - lifetime (static/automatic)
    - int: vpiLifetime
  - extern
    - bool: vpiExtern
  - name
    - str: vpiName

Constraint ordering

Constraint dist
dist item

- vpiLeftRange
- vpiRightRange
- vpiWeight

-> operation type ( := or :/= )
int: vpiOpType

constraint expr

- implication
- constr if
- constr if else

-> constraint expr type
vpiExpr
vpiImplication
vpiIfExpr
vpiIfElseExpr
Variables (26.6.8)

NOTES
1. **vpiWaitingProcess** 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

Variables

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

NOTES

vpi_get_value/vpi_put_value do not work for unpacked structures or union variables.

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(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

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.
**Alias Statement**

Examples

```plaintext
alias a=b=c=d

Results in 3 aliases:

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

```

d is Rhs for all.
Frames (26.6.20)

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
concurrent assertions

assert property

clocking event
(or null)

disable condition
(or null)

cover property

stmt

vpiSuccessStmt
(or null)

vpiFailStmt
(or null)

definition location
str: vpiDefFile
int: vpiDefLineNo

block identifier
str: vpiName
str: vpiFullName

disable iff
bool: vpiDisableIff

clocking
bool: vpiClkEvent

assertion type
int: vpiAssertionType

vpiProperty

property inst

property spec
disable condition

definition location
  int: vpiDefLineNo
  str: vpiDefFile

clocking event
event control

name (clocking identifier)
  str: vpiName
  str: vpiFullName

definition location
  int: vpiDefLineNo
  str: vpiDefFile

inferred or declared
  bool: vpiInferred

property decl

property spec

property inst

formal list

name
  str: vpiName
  str: vpiFullName

definition location
  str: vpiDefFile
  int: vpiDefLineNo
property spec

- definition location
  - int: vpiDefLineNo
  - str: vpiDefFile
- name
  - str: vpiName
  - str: vpiFullName
- complement
  - bool: vpiNot

property decl

clocking event

disable condition
  (or null)

variables

(! concurrent assertions)

property expr

- vpiRhs
- vpiLhs

operation
  - int: vpiImply
  - int: vpiDelayedImply
- complement consequent (not)
  - bool: vpiNot
multiclock
sequence expr
vpiOperand
clocked seq
clocking
bool: vpiClkEvent
clocked seq
(sequence expr)
clocking event
property inst
arguments
definition location
int: vpiDefLineNo
str: vpiDefFile
property decl
sequence inst \rightarrow \text{sequence decl} \rightarrow \text{sequence spec}

- definition location
  - \text{str: vpiDefFile}
  - \text{int: vpiDefLineNo}
- block identifier
  - \text{str: vpiName}
  - \text{str: vpiFullName}

\text{sequence spec} \rightarrow \text{Sequence}

- \text{sequence expr}
  - \text{multiclock}
  - \text{sequence expr}

\text{formal list} \rightarrow \text{formal list item}

\text{formal list item} \rightarrow \text{identifier}

\text{event expression}

\text{actual arg expr} \rightarrow \text{event expression}

- connected by name
  - \text{bool: vpiConnectByName}
- explicitly named
  - \text{bool: vpiExplicitName}
- argument index
  - \text{int: vpiPortIndex}
- name
  - \text{str: vpiName}
int: vpiSeqOpType is one of:

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