VPI Extensions to SystemVerilog

September 2003

SYNOPSYS®
NOTES
1. Top-level modules shall be accessed using \texttt{vpi\_iterate()} with a NULL reference object.
2. Passing a NULL handle to \texttt{vpi\_get()} with types \texttt{vpiTimePrecision} or \texttt{vpiTimeUnit} shall return the smallest time precision of all modules in the instantiated design.
3. The properties \texttt{vpiDefLineNo} and \texttt{vpiDefFile} can be affected by the \texttt{`line} and \texttt{`file} compiler directives. See Clause 19 for more details on these compiler directives.
4. If a module is an element within a module array, the \texttt{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.
5. With SystemVerilog, $\texttt{root}$ is always the top-level module. To get all top-level modules specified in the design, the user must do \texttt{vpiIterate(vpiModuleRoot\_handle)}.

\textbf{Modport}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{modport.png}
\caption{Modport Diagram}
\end{figure}

- \texttt{modport} to \texttt{mpport}
- \texttt{interface} to \texttt{modport}
- \texttt{-> name} to \texttt{str: vpiName}
- \texttt{-> direction} to \texttt{int: vpiDirection}
Task TF Declaration

For `TFDecl` with access type `vpiForkJoin`, there can be more than one task/function definition. Hence, the user needs to use `vpiIterate(vpiTask,TFDecl_handle)` to get to the TF definition.

Mport

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 site, not on the actual.

2. `vpi_get/put` 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 `vpiMpPort` 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.
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 pointer to the original object.

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
2. vpiPort and vpiPortInst is defined only for vpiRefObj where vpiRefObjType is vpiInterface.
endmodule

/***********************************
for port of i1,
    vpiHighConn = vpiRefObj where vpiRefObjType = vpiInterface
for port of i2,
    vpiHighConn = vpiRefObj where vpiFullType = vpiModport
****************************************/
module child1(interface simple s)
    c1 c_1(s);
    c1 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 refObj, where full type is vpiInterface.
for port of c_2:
    vpiHighConn is a refObj, where full type is vpiModport.
NOTES
1. A var select is a word selected from a variable array.
2. The VPI does not provide access to bits of variables. If a handle to bit select of a variable is obtained, the object shall be a \texttt{vpiBitSelect} in the simple expression class. The variable containing the bit can be accessed using \texttt{vpiParent}.
3. The boolean property \texttt{vpiArray} shall be \texttt{TRUE} if the variable handle references an array of variables, and \texttt{FALSE} otherwise. If the variable is an array, iterate on \texttt{vpiVarSelect} to obtain handles to each variable in the array.
4. \texttt{vpi\_handle (vpi\_Index, var\_select\_handle)} shall return the index of a var select in a 1-dimensional array. \texttt{vpi\_iterate (vpi\_Index, 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.
5. \texttt{vpiLeftRange} and \texttt{vpiRightRange} shall apply to variables when \texttt{vpiArray} is \texttt{TRUE}, and represent the array range declaration. These relationships are only valid when \texttt{vpiArray} is \texttt{TRUE}.
6. \texttt{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.
7. \texttt{vpiSize} for a var select shall return the number of bits in the var select. This applies only for packed var select.
8. Variables whose boolean property \texttt{vpiArray} is \texttt{TRUE} do not have a value property.
9. \texttt{vpiBit} iterator applies only for logic, bit, packed struct, and packed union variables.
10. \texttt{vpiIndexType} is valid only for associative array.
11. \texttt{cbSizeChange} will be applicable only for dynamic and associative array.
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

**Instance Arrays (26.6.2)**
Scope (26.6.3)

- vpiInternalScope
  - variables
    - reg
    - reg array
    - memory
    - parameter
  - named event
    - array
  - named begin
    - begin
  - named fork
  - end
  - class

-> name
str: vpiName
str: vpiFullName
IO declaration (26.6.4)

NOTE

vpiDirection returns ref for pass by ref ports.

clocking domain

-> name
   str: vpiName
Class Object Definition

NOTE

1. **ClassDefn** handle is a new concept. It does not correspond to any variable (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

- `NULL`
- `constraint`
- `class`
- `scope`
- `instances`

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

- `vpiSolveBefore`
  - `expr`
- `vpiSolveAfter`
  - `expr`

- `vpiExpr`
  - `expr`
- `vpiDistItem`
  - `dist item`
**dist item**

- `vpiLeftRange` -> `expr`
- `vpiRightRange` -> `expr`
- `vpiWeight` -> `expr`

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

**constraint expr**

- `vpiCondition` -> `expr`
- `vpiElseConst` -> `constraint expr`

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

- **vpiClassDefn**
  - `class defn`
  - `int: vpiClassType`
  - `int: vpiMailbox`
  - `int: vpiSemaphore`

- **vpiVariable**
  - `variables`
  - `class defn`

- **vpiWaitingProcess**
  - `-> Class type`
  - `int: vpiClassType`
  - `int: vpiMailbox`
  - `int: vpiSemaphore`

- **vpiMessage**
  - `expr`
  - `class defn`

- **vpiActualDefn**

---

**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.
   - The assumption is that no initial/always block can wait on these objects.
2. **vpiMessage** iterator should return all the messages in a mailbox.
3. **vpiClassDefn** returns the ClassDefn which was used to create the handle.
4. **vpiActualDefn** returns the ClassDefn the handle object points to when the query is made.
5. **vpiClassDefn/vpiActualDefn** both should return null for built-in classes.
Structure/Union

NOTES

\texttt{vpi\_get\_value/vpi\_put\_value} do not work for unpacked structures or union variables.

Enum, Enum Constant
**vpiNamedEvent**

NOTE

The new iterator (vpiWaitingProcess) returns all waiting processes for that namedEvent - the processes here are frames only as we assume that only task/function can wait for a 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. vpiInterfaceTask/vpiInterfaceFunction will be true if task/function is declared inside and interface or a modport off an interface.
3. For function where return type is a user-defined type, vpi_handle (vpiReturn/Function_handle should return a dummy variable handle from which the user can get the details of that UDT.)
Alias Stmt

![Diagram of alias stmt and related components.](image-url)
NOTES

1. The following callbacks should be supported on frames:
   - cbStartOfFrame: triggers whenever any frame gets executed.
   - cbEndOfFrame: triggers when a particular thread is deleted.

2. Please note that we have changed the \texttt{vpiParent} handle from the LRM. \texttt{vpiOrigin} now gives the originating scope or task/function call.
## Threads

NOTES

The following callbacks should be supported on threads

- **cbStartOfThread**: triggers whenever any thread is created
- **cbEndOfThread**: triggers when a particular thread gets deleted
- **cbEnterThread**: triggers whenever a particular thread resumes execution
concurrent assertions

assert property

cover property

stmt

vpiSucceStmt (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

NOTE: All objects have the property int: vpiType
**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
```
multiclock sequence expr

vpiOperand

clocked seq

clocking

bool: vpiClkEvent

clocked seq

(sequence expr)

clocking event

property inst

definition location

int: vpiDefLineNo
str: vpiDefFile

arguments

property decl
sequence decl

- definition location
  - str: vpiDefFile
  - int: vpiDefLineNo
- block identifier
  - str: vpiName
  - str: vpiFullName

sequence spec

- sequence expr
- multiclock

formal list

formal list item

- identifier
- event expression

actual arg expr

- event_expression

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

- `and`
- `intersect`
- `or`
- `first_match`
- `throughout`
- `within`
- `##`
- `$[*]$
- `$[*=]$
- `$[*->]$

Where:

- **operation**: `vpiOperand` is a sequence expression.
- **sequence expr**: `vpiOperand` is a sequence expression.
- **sequence inst**: `vpiOperand` is a sequence instance.
- **definition location**: `int: vpiDefLineNo` and `str: vpiDefFile`.
- **name**: `str: vpiName` and `str: vpiFullName`.
- **expr**: `assignment`.

The diagram illustrates the relationships and components involved in a sequence expression within a design automation context.