Section 31
SystemVerilog VPI Object Model

LRM 183

EDITOR'S NOTE: I added an introduction before the diagrams, drawing from the table of contents in the change documentation.

EDITOR'S NOTE: I still have a lot of work to do on this section. Some diagrams do not fit the page width. The fonts are not consistent. There is spurious text in some of the diagrams. ...

31.1 Introduction (informative)

SystemVerilog extends the Verilog Procedural Interface (VPI) object diagrams to support SystemVerilog constructs. The VPI object diagrams document the properties of objects and the relationships of objects. How these diagrams illustrate this information is explained in Section 26 of the IEEE Std. 1364-2001 Verilog standard. The SystemVerilog extensions to the VPI diagrams are in the form of changes to or additions to the diagrams contained in 1364-2001 Verilog standard. The following table summarizes these changes and additions.

The following table summarizes the changes and additions made to the Verilog VPI object diagrams:

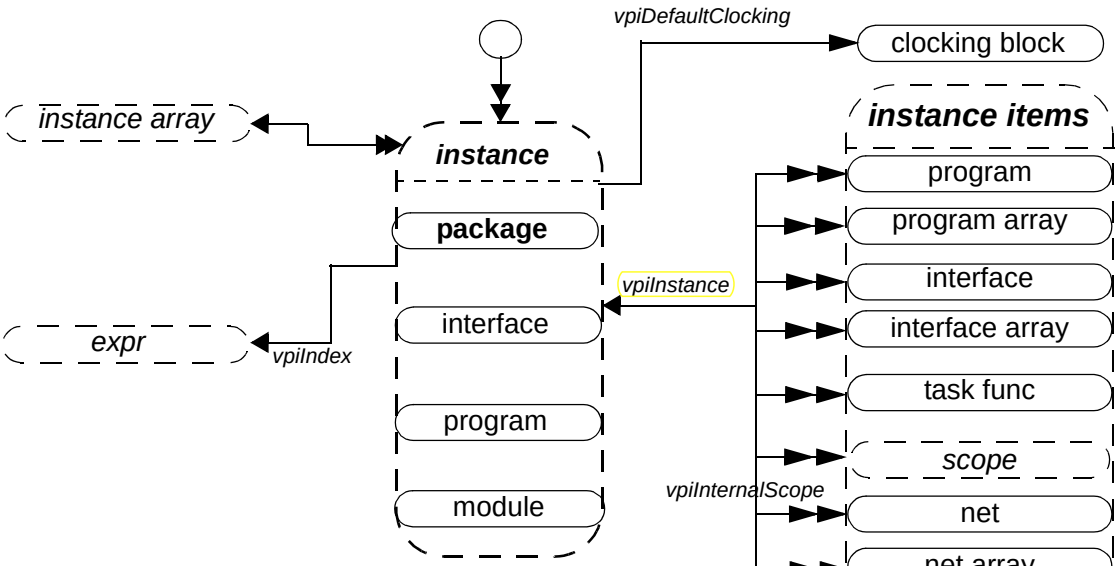
Table 31-4: Verilog VPI object diagram changes and additions

Diagram	Notes
Instances	New
Interface	New
Program	New
Module	Replaces IEEE 1364.2001 section 26.6.1
Modport	New
Interface tf decl	New
Ports	Replaces IEEE 1364.2001, section 26.6.5
Ref Obj	New
Variable	Replaces IEEE 1364.2001 section 26.6.8
Var select	New
Typespec	New
Variable Drivers and Loads	New
Instance Arrays	Replaces IEEE 1364.2001 section 26.6.2
Scope	Replaces IEEE 1364.2001 section 26.6.3
IO Declaration	Replaces IEEE 1364.2001 section 26.6.4
Class Object Definition	New
Constraint	New
Dist Item	New

Table 31-4: Verilog VPI object diagram changes and additions (continued)

Diagram	Notes
Constraint Expression	New
Class Variables	New
Structure/Union	New
Named Events	New
Named Event Array	New
Task, Function Declaration	Replaces IEEE 1364.2001 section 26.6.18
Alias Statement	New
Frames	Replaces IEEE 1364.2001 section 26.6.20
Threads	New
Concurrent Assertions	New
Disable Condition	New
Clocking Event	New
Property Declaration	New
Property Specification	New
Property Expression	New
Multiclock Sequence Expression	New
Sequence Declaration	New
Sequence Expression	New
Instances	New
Atomic Statement	New
if, if-else	Replaces IEEE 1364-2001 section 26.6.35
case	Replaces IEEE 1364-2001 section 26.6.36
return	New
do while	New
waits	Replaces wait in IEEE 1364-2001 section 26.6.32
disables	Replaces IEEE 1364-2001 section 26.6 38
expect	New
foreach	New

31.2 Instances



-> array member
bool: vpiArray
-> cell
bool: vpiCellInstance
-> default net type
int: vpiDefNetType
-> definition location
int: vpiDefLineNo
str: vpiDefFile
-> definition name
str: vpiDefName
-> delay mode
int: vpiDefDelayMode
-> name
str: vpiName
str: vpiFullName
-> protected
bool: vpiProtected
-> timeprecision
int: vpiTimePrecision
-> timeunit
int: vpiTimeUnit
-> unconnected drive
int: vpiUnconnDrive
-> Configuration
str: vpiLibrary
str: vpiCell
str: vpiConfig
->default lifetime
bool: vpiAutomatic
-> top
bool: vpiTop
compile unit
bool: vpiUnit

NOTES

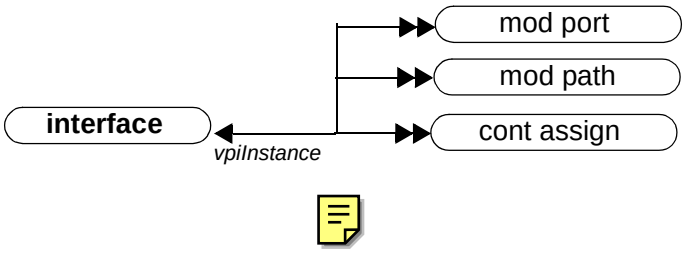
1) Top-level instances shall be accessed using **vpi_iterate()** with a NULL reference object.

1) Passing a NULL handle to **vpi_get()** with types **vpiTimePrecision** or **vpiTimeUnit** shall return the smallest time precision of all instances in the design.

2) 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.

3) Compilation units are represented as packages that have a **vpiUnit** property set to TRUE. Such implicitly declared packages shall have implementation dependent names.

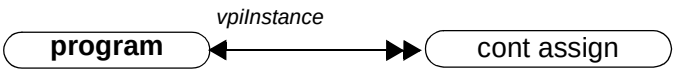
31.3 Interfaces



NOTE

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

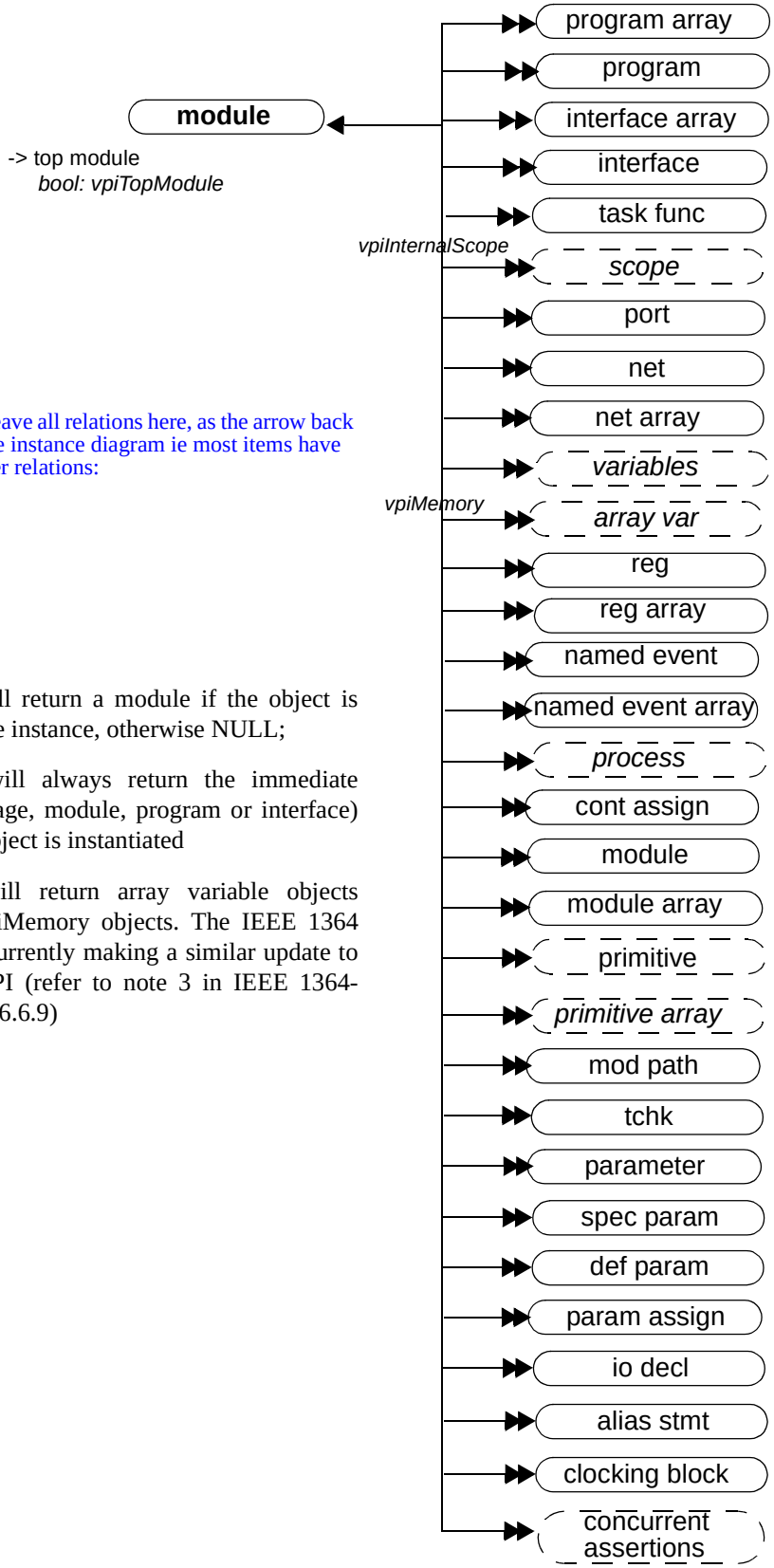
31.4 Program



NOTE

All programs are instances and all relations and properties in the Instances diagram also apply.

31.5 Module (supercedes IEEE 1364-2001 26.6.1)



TE to reviewers leave all relations here, as the arrow back
ifferent than in the instance diagram ie most items have
two possible container relations:
vpiModule
vpiInstance

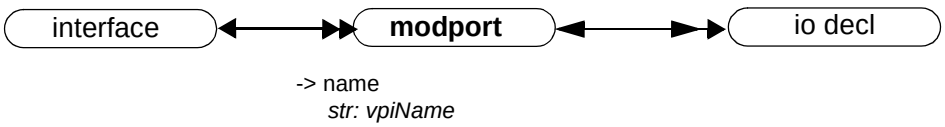
NOTES

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

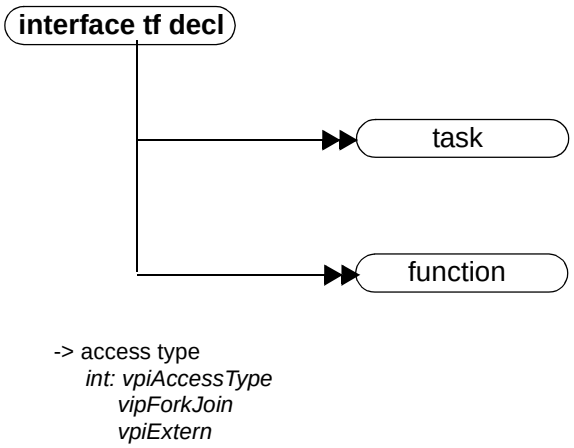
2) **vpiInstance** will always return the immediate instance (package, module, program or interface) in which the object is instantiated

3) vpiMemory will return array variable objects rather than vpiMemory objects. The IEEE 1364 committee is currently making a similar update to the Verilog VPI (refer to note 3 in IEEE 1364-2001, section 26.6.9)

31.6 Modport



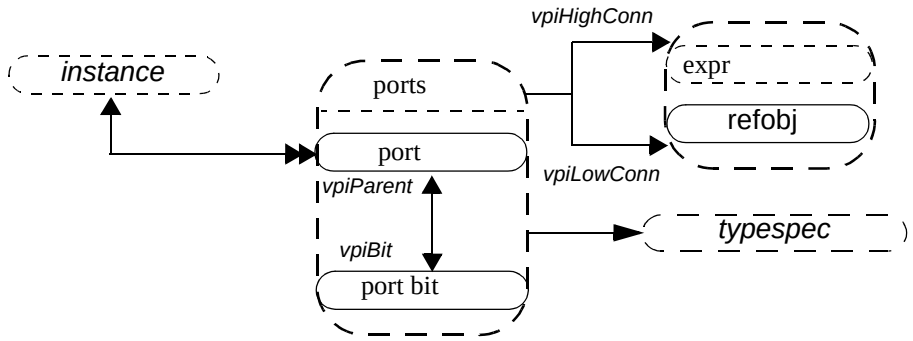
31.7 Interface tf decl



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.

31.8 Ports (supercedes IEEE 1364-2001 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.**

1) **vpiHighConn** shall indicate the hierarchically higher (closer to the top module) port connection.

2) **vpiLowConn** shall indicate the lower (further from the top module) port connection.

3) **vpiLowConn** of a **vpiInterfacePort** shall always be **vpiRefObj**.

4) 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.

5) Properties index and name shall not apply for port bits.

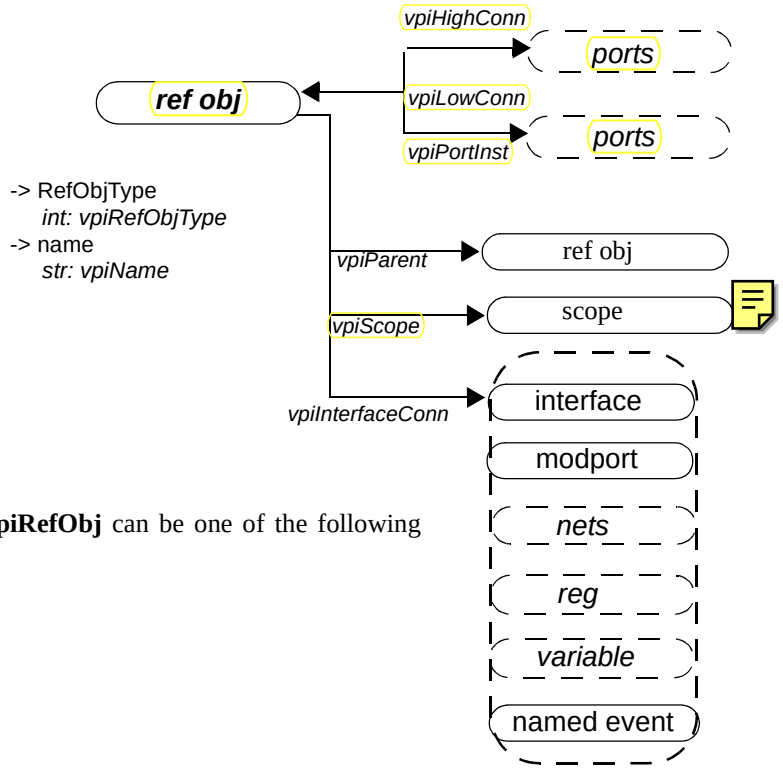
6) If a port is explicitly named, then the explicit name shall be returned. If not, and a name exists, then that name shall be returned. Otherwise, NULL shall be returned.

7) **vpiPortIndex** can be used to determine the port order. The first port has a port index of zero.

8) **vpiHighConn** and **vpiLowConn** shall return NULL if the port is not connected.

9) **vpiSize** for a null port shall return 0.

31.9 Ref Obj



NOTES

1) **vpiRefObjType** of **vpiRefObj** can be one of the following types:

— vpiInterface

— vpiModport

— vpiNet

— vpiReg

10) **vpiPort** and **vpiPortInst** is defined only for **vpiRefObj** where **vpiRefObjType** is **vpiInterface**.

31.9.1 Examples

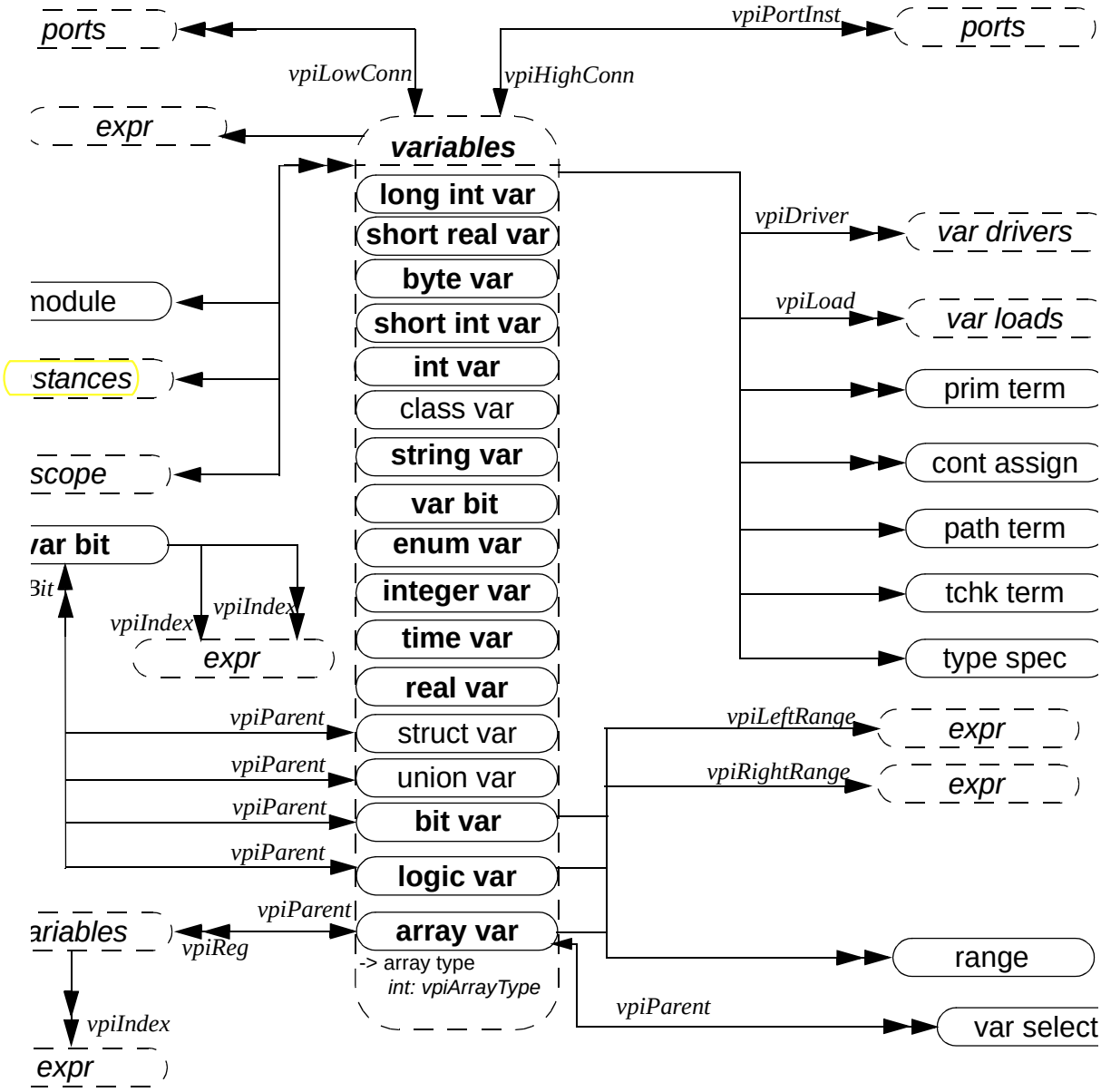
These objects are newly defined objects needed for supporting the full connectivity through ports where the ports are vpiInterface or vpiModport or any object inside modport or interface.

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

```
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)
   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 vpiRefObj, where full type is vpiInterface.
for port of c_2 :
  vpiHighConn is a vpiRefObj, where full type is vpiModport.
```


31.10 Variables (supercedes IEEE 1364-2001 section 26.6.8)



ay member
ol: vpiArray
ne
: vpiName
: vpiFullName
n
ol: vpiSigned
e
: vpiSize
ermine random availability
ol: vpiIsRandomized

-> multi array
bool: vpiMultiArray
-> lifetime
bool: vpiAutomatic (ref. 26.6.20, 1364 2001)
-> constant variable
onstantVariable
-> omiztion type
int: vpiRandType
can be vpiRand, vpiRandC, vpiNotRand

-> member
vpiMember
->value
vpi_get_value()
vpi_put_value()
-> packed array
bool: vpiPacArray
-> scalar
bool: alar
-> visibility
int: vpiVisibility
-> vector
bool: vpiVector

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) To obtain the members of a union and structure, see the relations in section **30.21**

4) The range relation is valid only when **vpiArray** is true. When applied to array vars this relation returns only unpacked ranges. When applied to logic and bit variables, it returns only the packed ranges.

5) 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

6) **vpiLeftRange** and **vpiRightRange** shall only apply if vpiMultiArray is not true, ie if the array is not multi-dimensional.

7) A variable handle of type **vpiArrayVar** represents an unpacked array. The range iterator for array vars returns only the unpacked ranges for the array.

8) If the variable has an initialization expression, the expression can be obtained from **vpi_handle(vpiExpr, var_handle)**

9) **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. For unpacked structures and unions the size returned indicates the number of fields in the structure or union.

10) **vpiSize** for a var select shall return the number of bits in the var select. This applies only for packed var select.

11) Variables whose boolean property **vpiArray** is TRUE do not have a value property.

12) **vpiBit** iterator applies only for logic, bit, packed struct, and packed union variables.

13) **vpi_handle(vpiIndex, var_bit_handle)** shall return the bit index for the variable bit. **vpi_iterate(vpiIndex, var_bit_handle)** shall return the set of indices for a multidimensional variable bit select, starting with the index for the bit and working outwards

14) **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.

15) The property *vpiRandType*, returns the current randomization type for the variable, which can be one of **vpiRand**, **vpiRandC**, and **vpiNotRand**.

16) **vpiIsRandomized** is a property to determine whether a random variable is currently active for randomization.

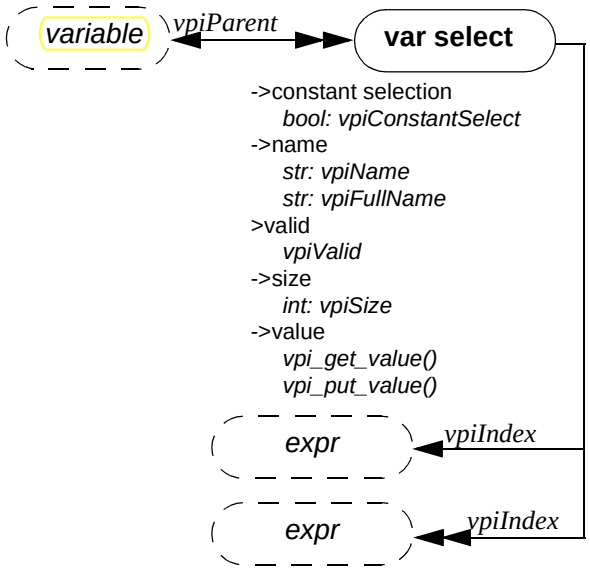
17) When the **vpiMember** property is true, it indicates that the variable is a member of a parent struct or union variable. See also relations in section **30.21**

18) If a variable is an element of an array, the **vpiIndex** iterator will return the indexing expressions that select that specific variable out of the array.

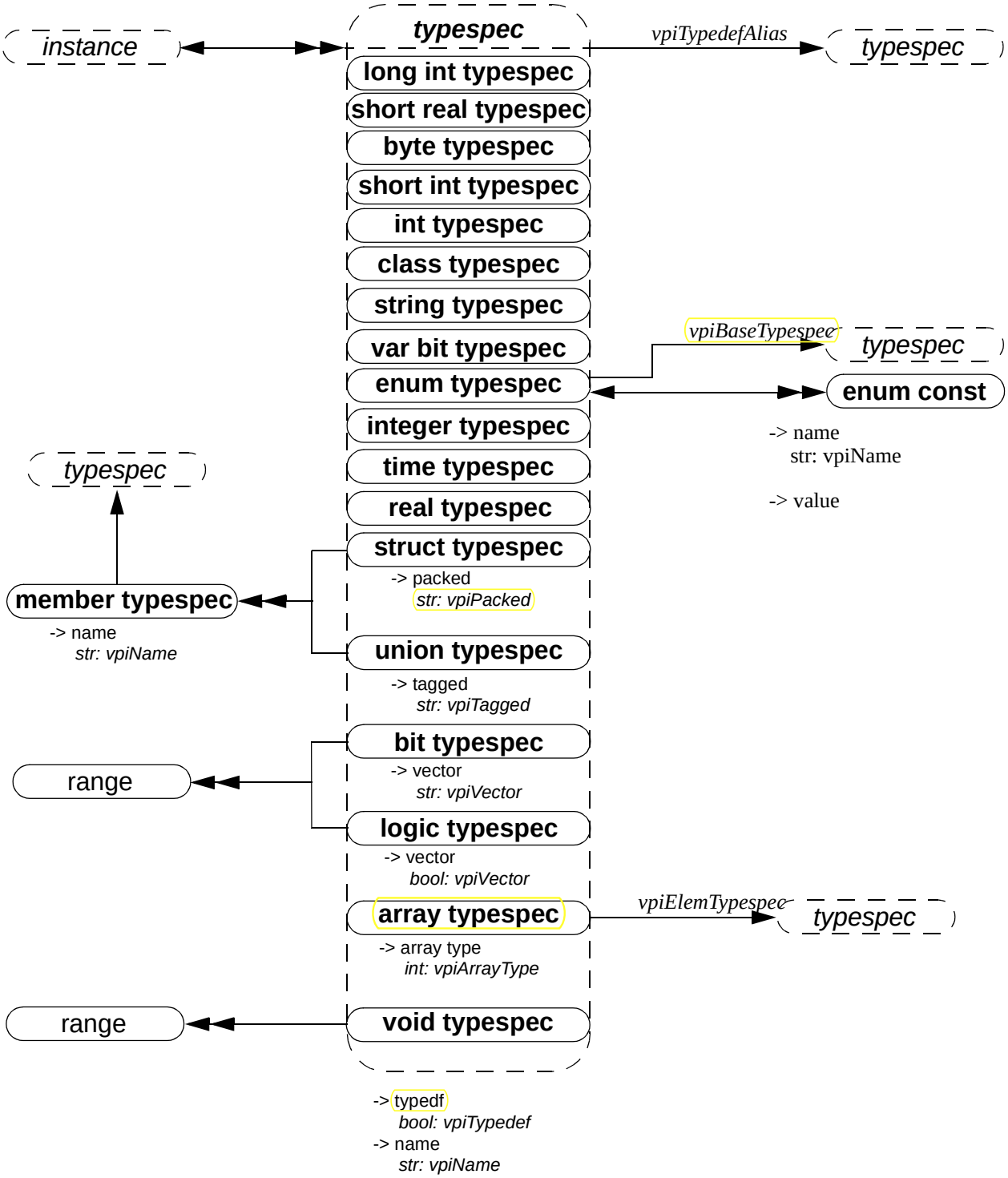
19) Note that:
logic var == reg

var bit var == reg bit
array var == reg array

31.11 Var Select (supercedes IEEE 1364-2001 26.6.8)



31.12 Typespec



NOTES

1) Typespec to typespec relation is used when the **vpiTypedefType** is "vpiTypedef", which will be the case for type aliases, for example, typedef a b;

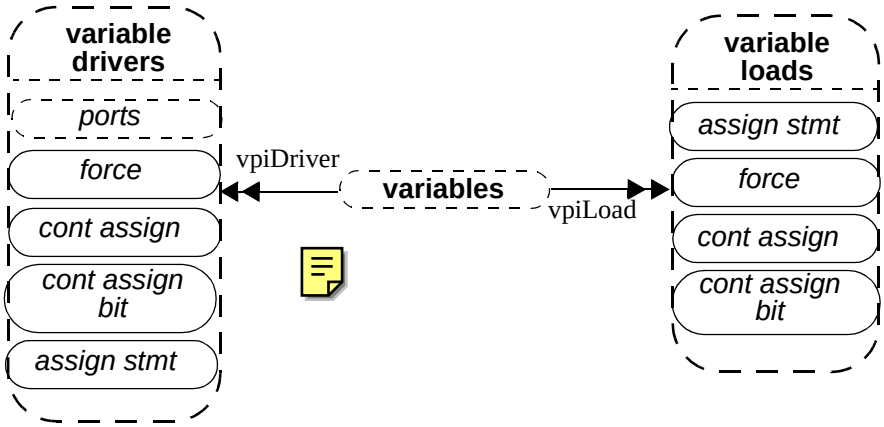
2) If the type of a type is **vpiStruct** or **vpiUnion**, then you can iterate over numbers to obtain the structure of the user-defined type. For each member the typespec relation from the member will detail its type.

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:

4)
```
typedef struct {
  struct
    int a;
  }B
} C;
```

5) 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**.

31.13 Variable Drivers and Loads (supercedes IEEE 1364-2001 26.6.23)



NOTES

1) **vpiDrivers/Loads** for a structure, union, or class variable will include the following:

   — Driver/Load for the whole variable

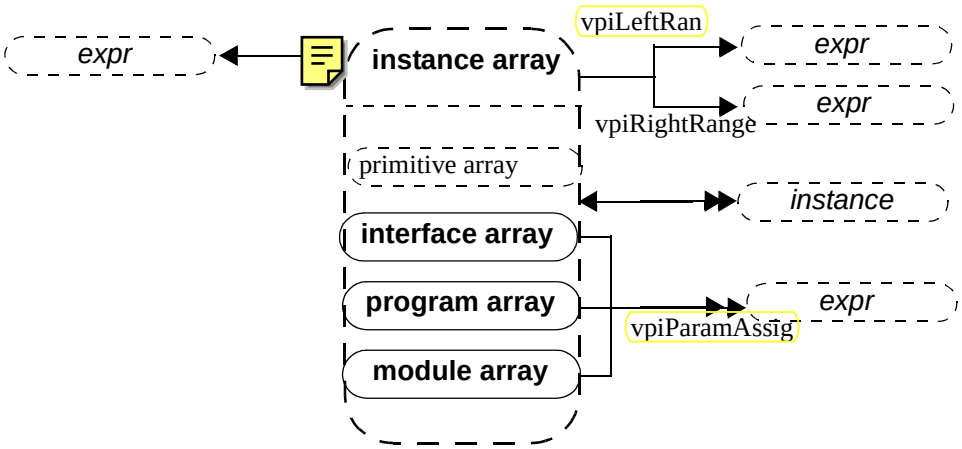
   — Driver/Load for any bit/part select of that variable

   — 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.

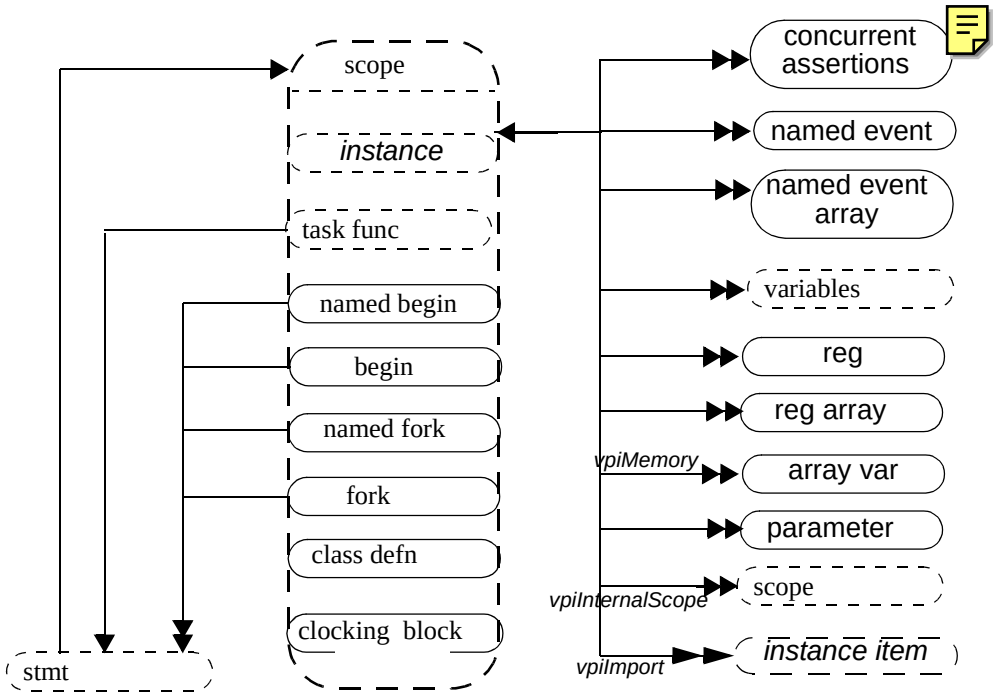
31.14 Instance Arrays (supercedes IEEE 1364-2001 26.6.2)



NOTE

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

31.15 Scope (supercedes IEEE 1364-2001 26.6.3)

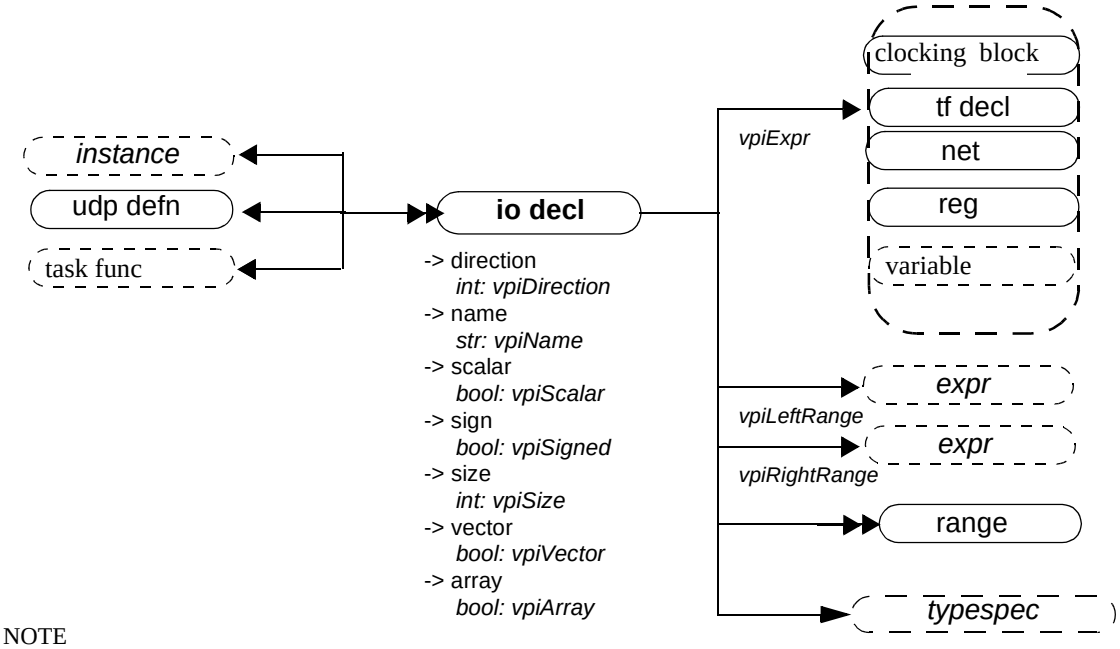


NOTE

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.

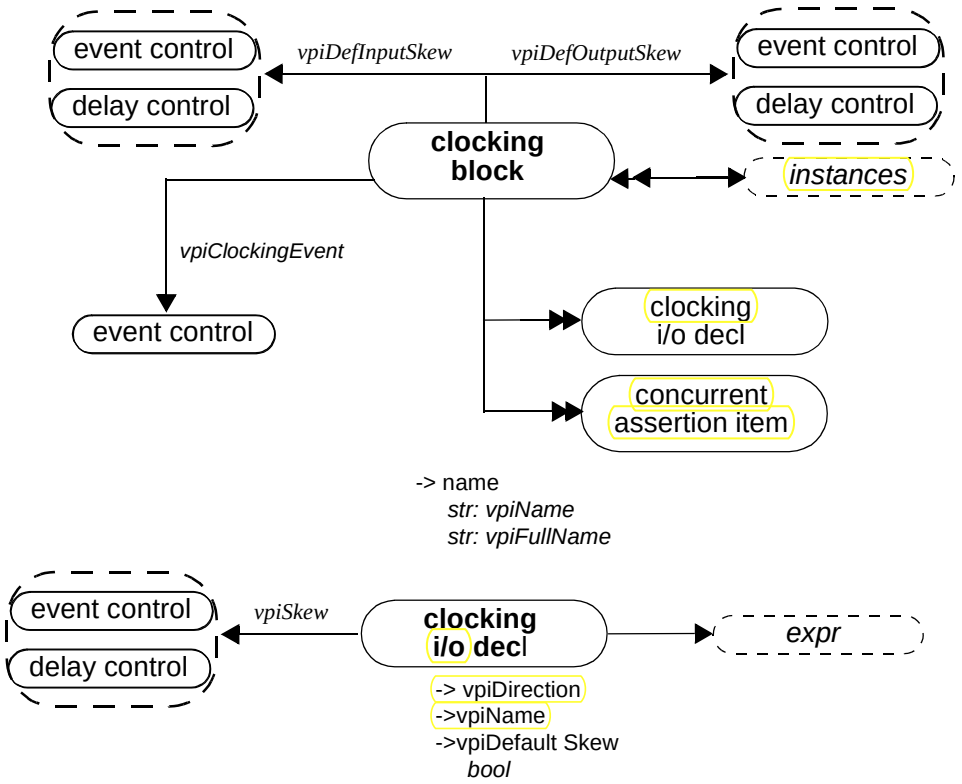
31.16 IO Declaration (supercedes IEEE 1364-2001 26.6.4)



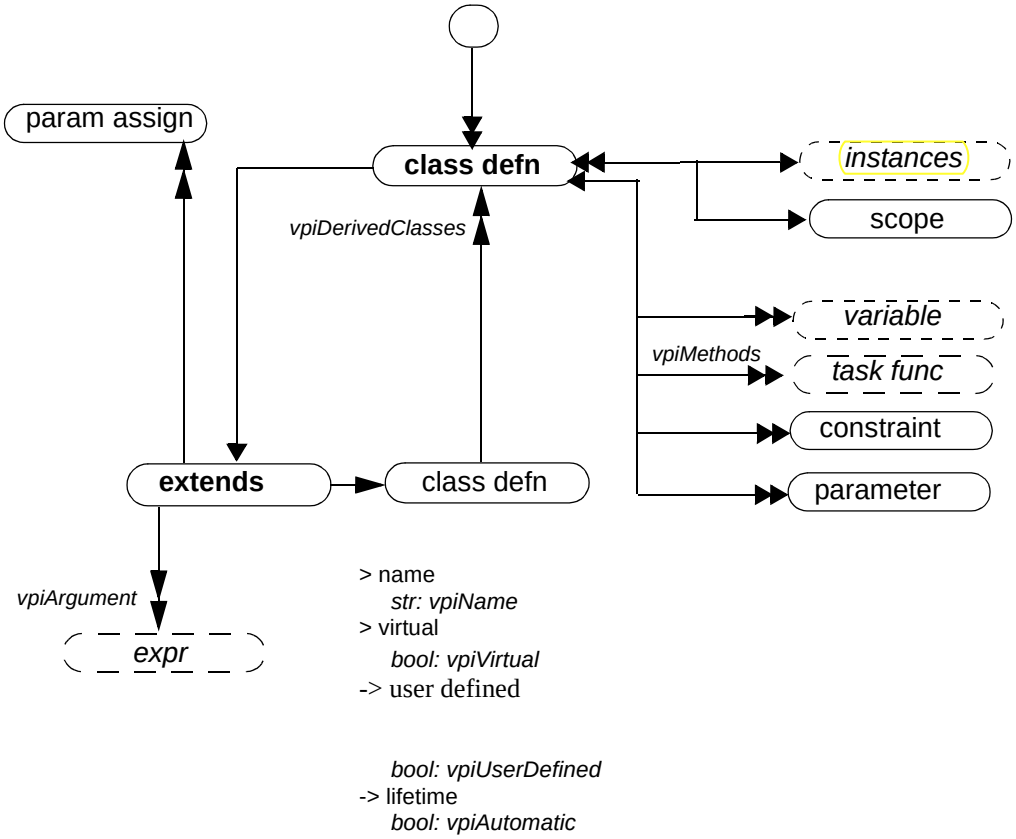
NOTE

vpiDirection returns **vpiRef** for pass by ref ports.

31.17 Clocking Block



31.18 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.

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

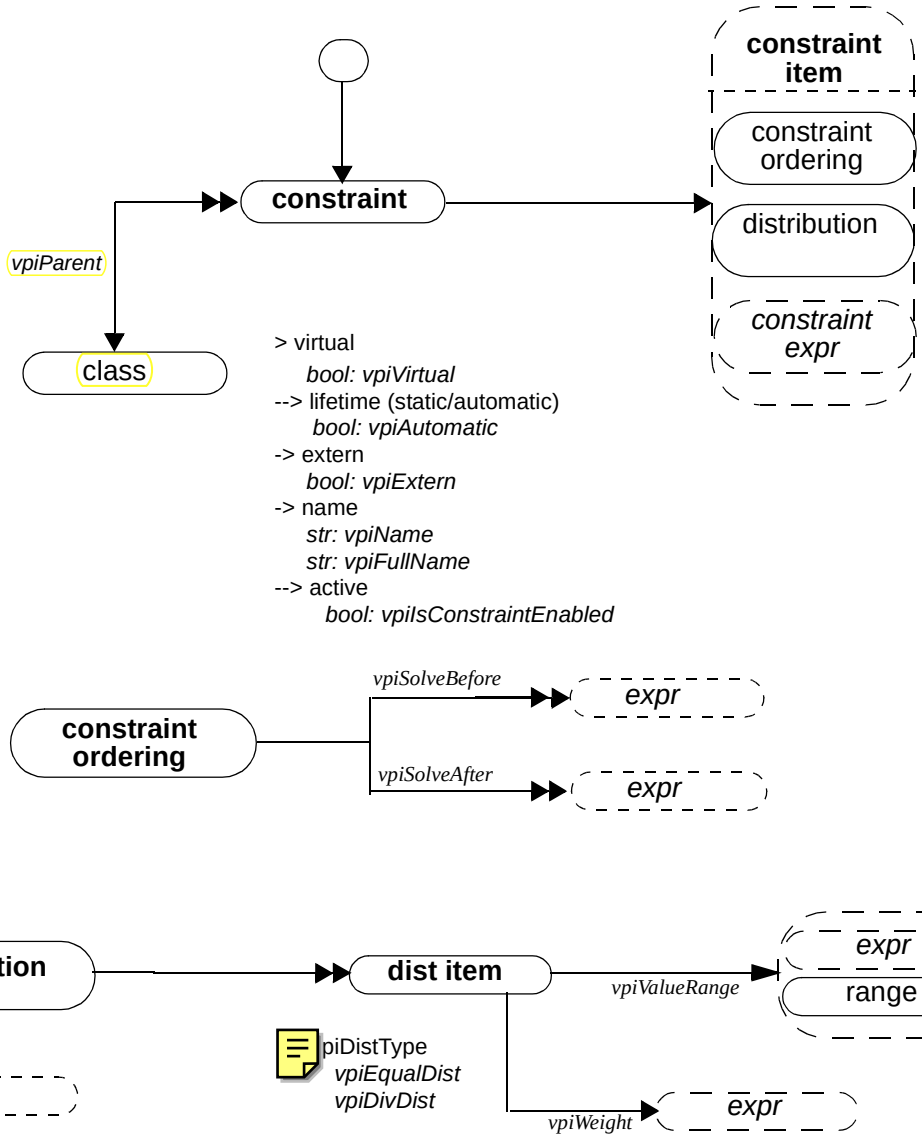
1) Iterator to constraints returns only normal constraints and not inline constraints.

1) To get constraints inherited from base classes, you will need to traverse the extend relation to obtain the base class.

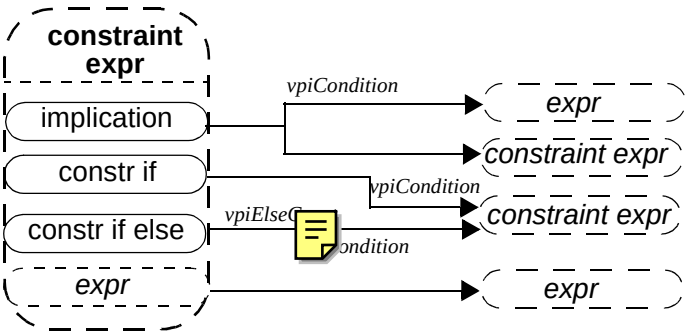
1) The vpiDerivedClasses iterator returns all the classes derived from the given class.

1) The relation to **vpiExtend** exists whenever a one class is derived from another class (ref Section 11.12). The relation from extend to classDefn provides the base class. The iterators from extend to param assign and arguments provide the parameters and arguments used in constructor chaining (ref Section 11.16 and 11.23)

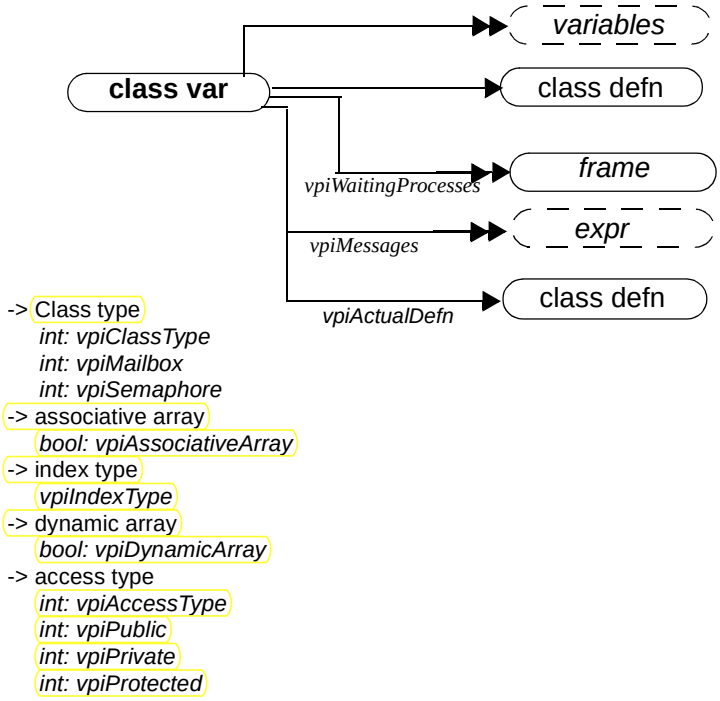
31.19 Constraint, constraint ordering, distribution,



31.20 Constraint expression



31.21 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.

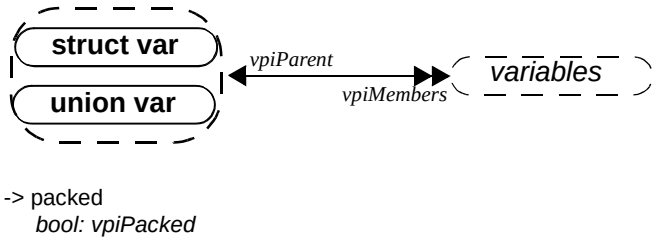
3) **vpiClassDefn** returns the ClassDefn which was used to create the handle. **vpiActualDefn** returns the ClassDefn that handle object points to when the query is made. The difference can be seen in the example below:

```
class Packet
...
endclass : Packet

class LinkedPacket extends Packet
...
endclass : LinkedPacket

LinkedPacket l = new;
Packet p = l;
```

   In this example, the vpiClassDefn of variable "p" is Packet, but the vpiActualDefn is "LinkedPacket".

4) **vpiClassDefn/vpiActualDefn** both shall return NULL for built-in classes.

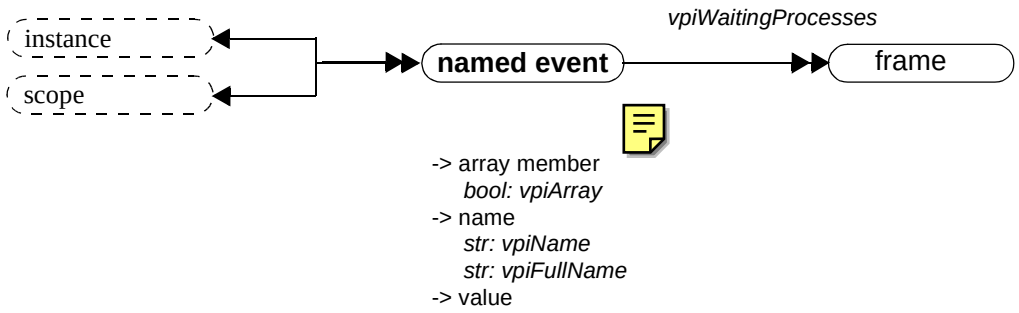
31.22 Structure/Union



NOTES

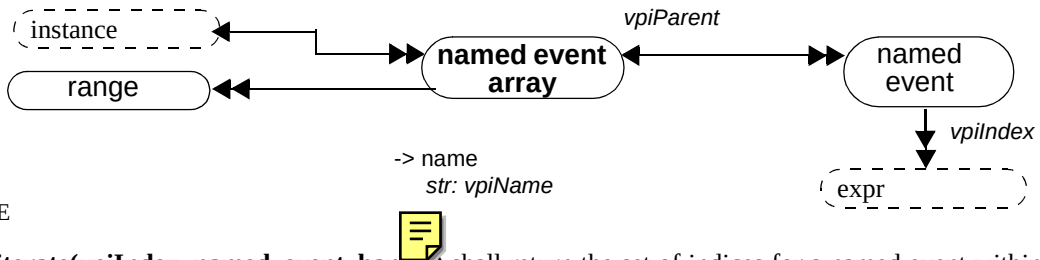
vpi_get_value/vpi_put_value cannot be used to access values of entire unpacked structures and unpacked unions.

31.23 Named Events (supercedes IEEE 1364-2001 26.6.11)



NOTE

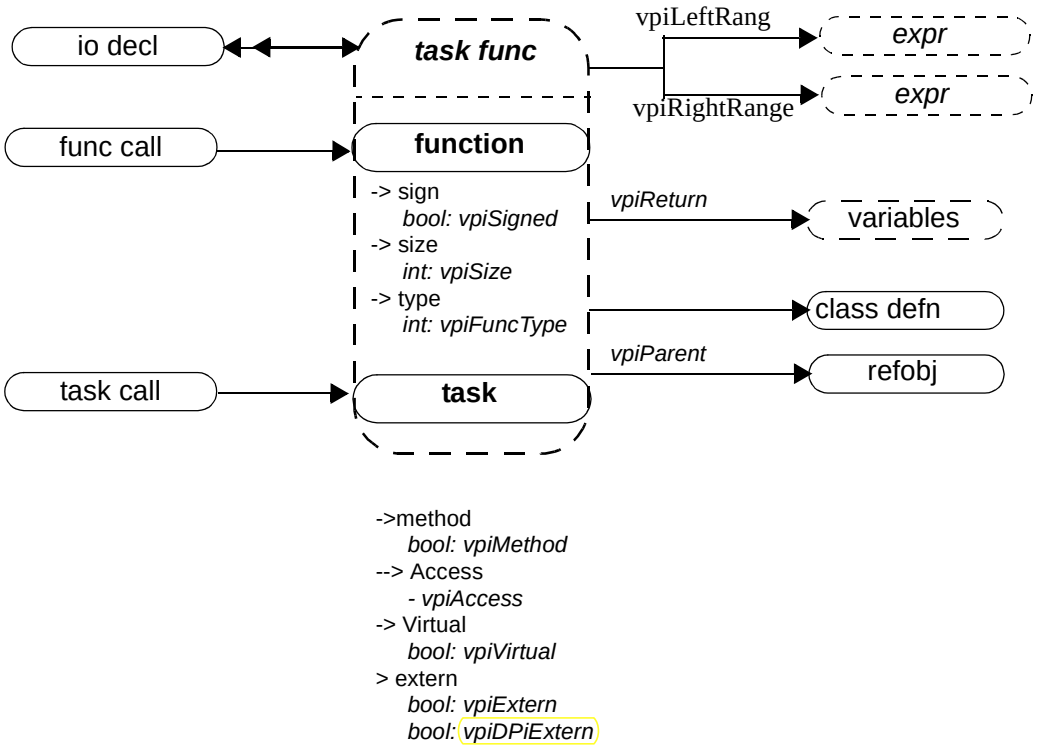
The new iterator (**vpiWaitingProcess**) returns all waiting processes, identified by their frame, for that namedEvent.



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.

31.24 Task, Function Declaration (supercedes IEEE 1364-2001 26.6.18)



NOTE

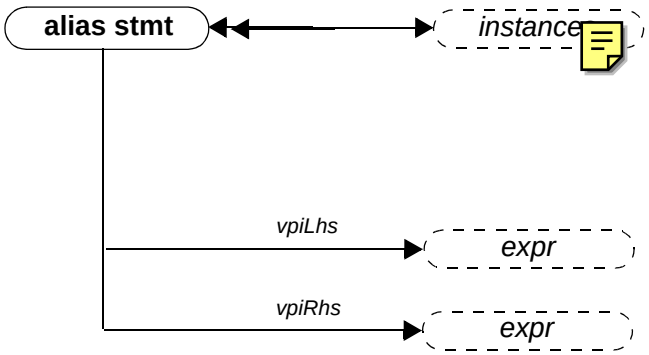
5) A Verilog HDL function shall contain an object with the same name, size, and type as the function.

6) **vpiInterfaceTask/vpiInterfaceFunction** shall be true if task/function is declared inside an interface or a modport of an interface.

7) 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.

8) **vpiReturn** will always return a var object, even for simple returns.

31.25 Alias Statement

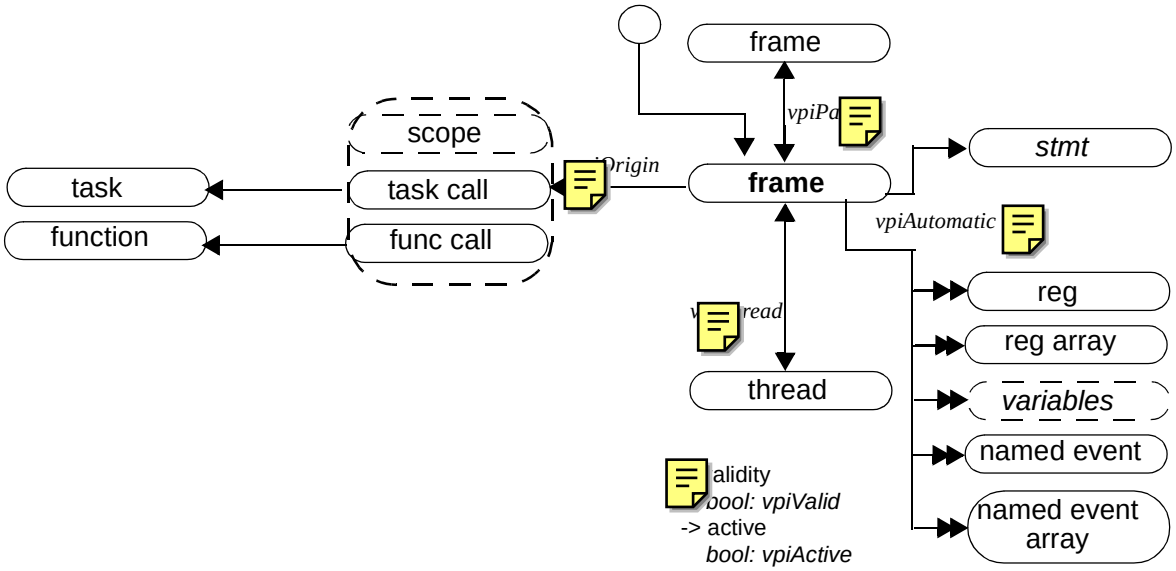


31.25.1 Examples

```
alias a=b=c=d
```
Results in 3 aliases:

```
   alias a=d
   alias b=d
   alias c=d
d is Rhs for all.
```

31.26 Frames (supercedes IEEE 1364-2001 26.6.20)



NOTES

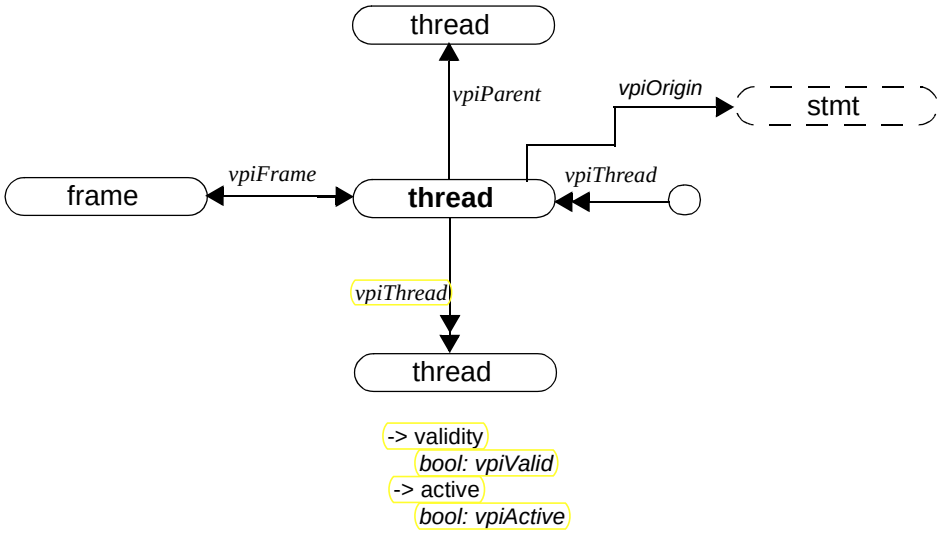
1) The following callbacks shall be supported on frames:

— **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. Note also that this diagram is incompatible with the one in IEEE 1364, but the IEEE is currently also making incompatible changes to that diagram.

31.27 Threads



NOTES

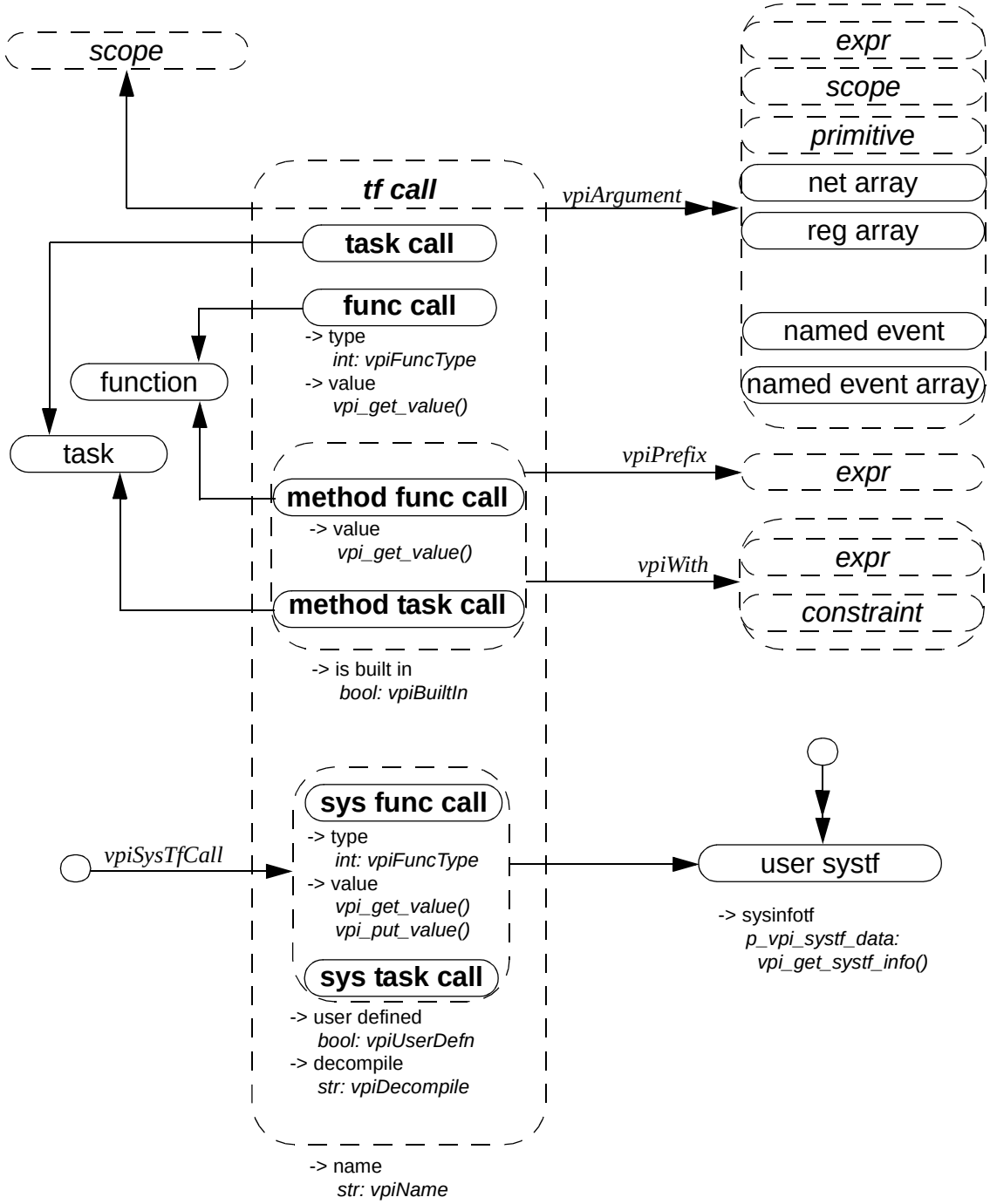
The following callbacks shall be supported on threads

— **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

31.28 tf call (supercedes IEEE 1364-2001 26.6.19)

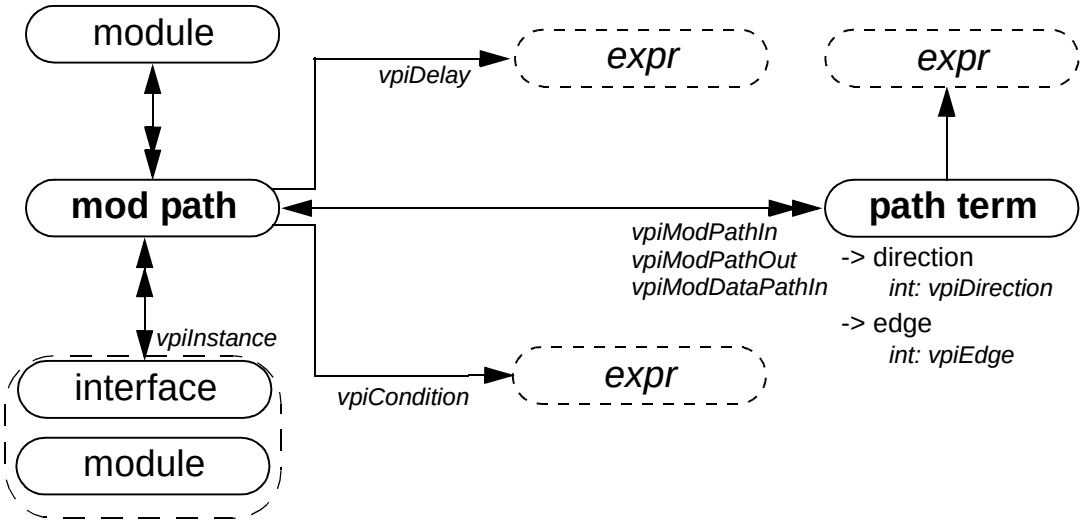


NOTE:

1) the vpiWith relation is only available for randomize methods (see section **12.6**) and for array locator methods (see section **4.15.1**).

2) For methods (method func call, method task call), the **vpiPrefix** relation will return the object to which the method is being applied. For example, for the class method invocation
packet.send();
the prefix for the "send" method is the class var "packet"

31.29 Module path, path term (supercedes IEEE 1364-2001 26.6.15)



mod path properties:

-> delay
vpi_get_delays()
vpi_put_delays()

-> path type
int: vpiPathType

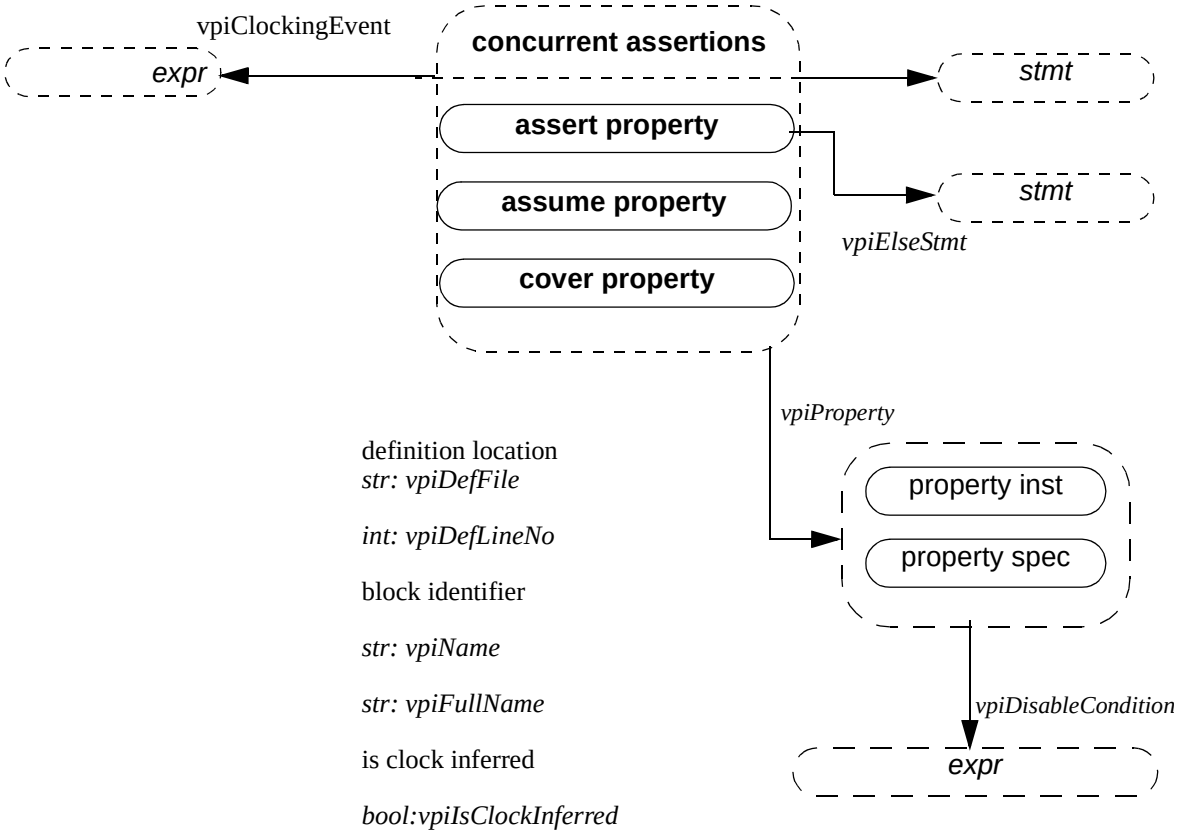
-> polarity
int: vpiPolarity
int: vpiDataPolarity

-> hasIfNone
bool: vpiModPathHasIfNone

NOTE:

1) specify blocks can occur in both modules and interfaces. For backwards compatibility the **vpiModule** relation has been preserved; however this relation will return **NULL** for specify blocks in interfaces. For new code it is recommended that the **vpiInstance** relation be used instead.

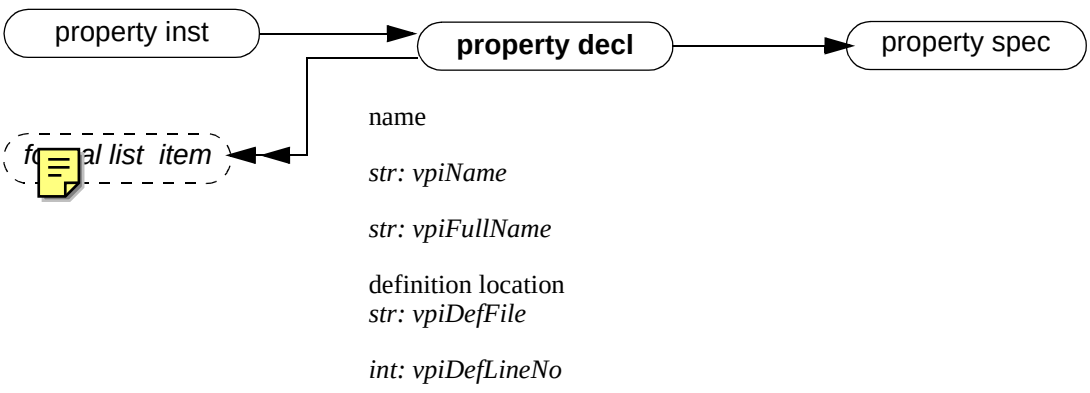
31.30 Concurrent assertions



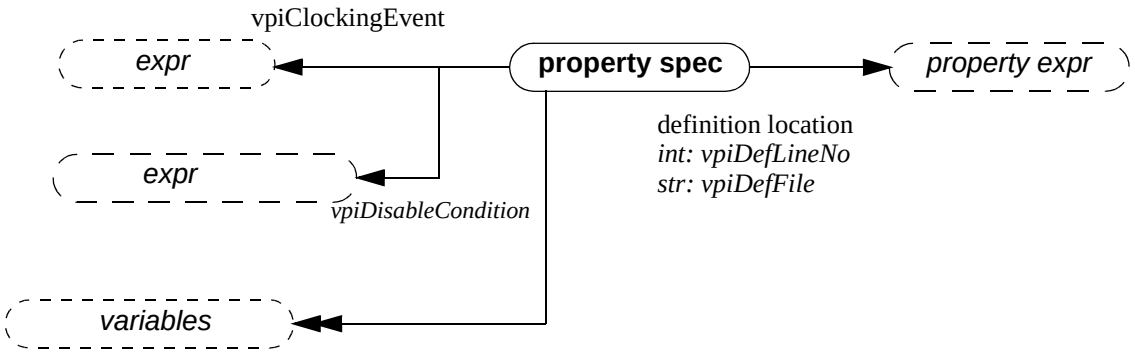
NOTE

Clocking event is always the actual clocking event on which the assertion is being evaluated, regardless of whether this is explicit or implicit (inferred)

31.31 Property Decl

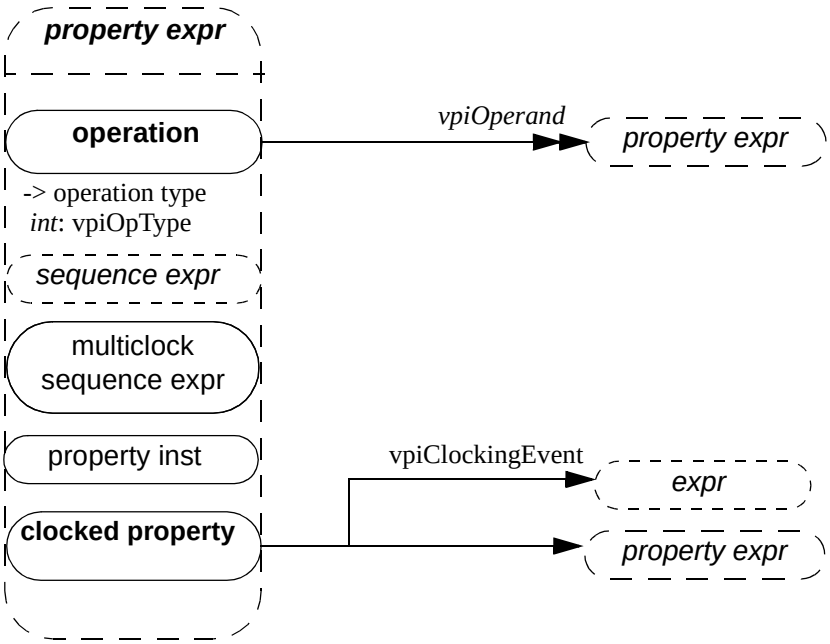


31.32 Property Specification



NOTE

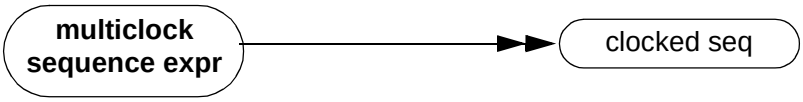
Variables are declarations of property variables. You cannot get the value of these variables.

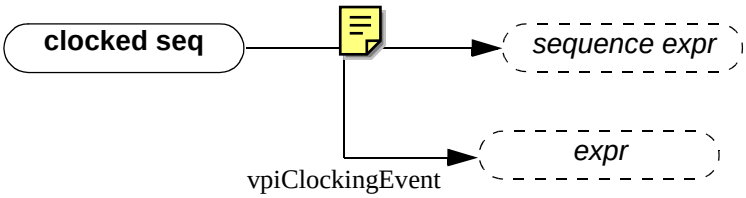


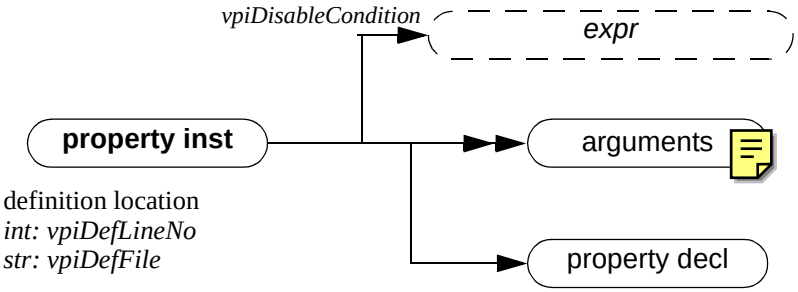
NOTES:

1. within the context of a property expr, **vpiOpType** can be any one of vpiNotOp, vpiImplyOp, vpiDelayedImplyOp, vpiAndOp, vpiOrOp, vpiIfOp, vpiIfElseOp
   Operands to these operations will be provided in the same order as show in the BNF.

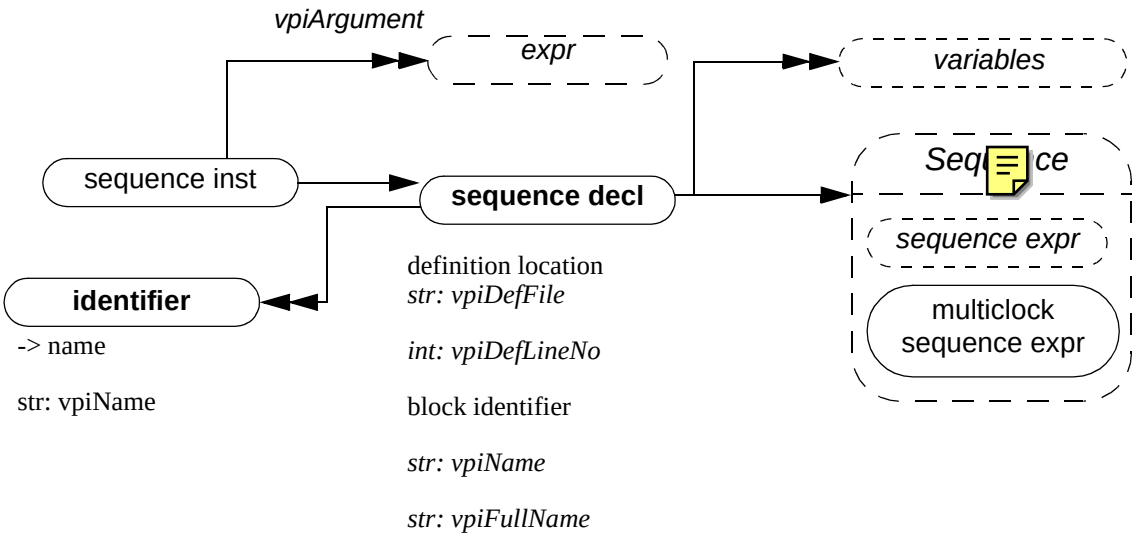
31.33 Multiclock Sequence Expression







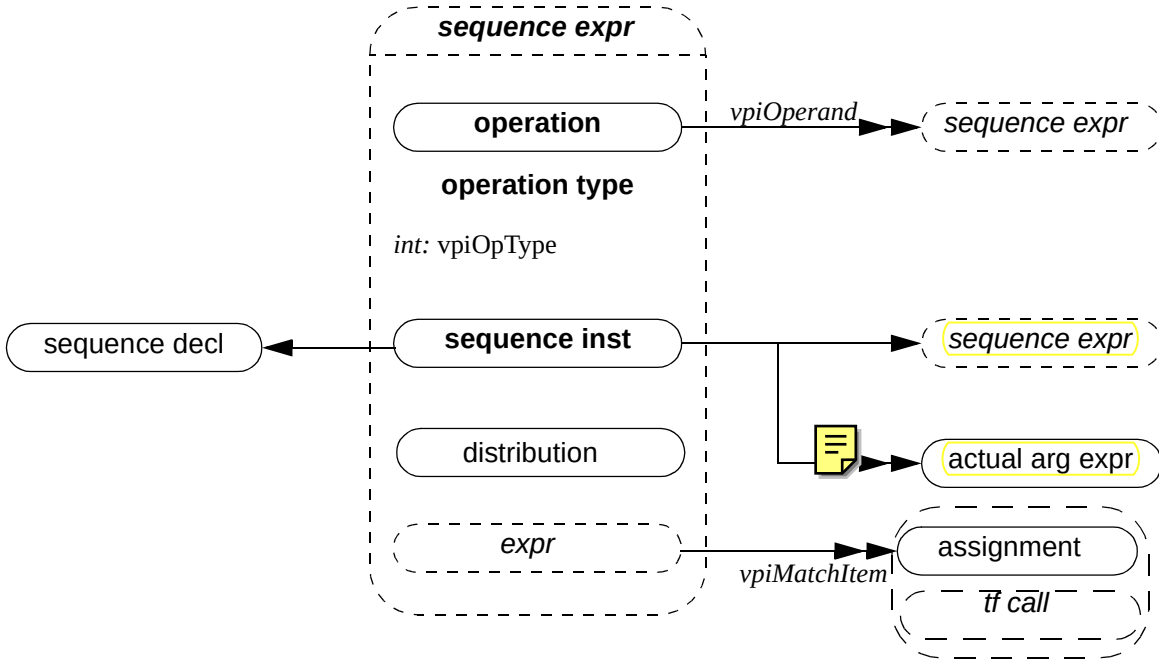
31.34 Sequence Declaration



NOTE:

the vpiArgument iterator shall return the sequence instance arguments in the order that the formals for the sequence are declared, so that the corresponce between each argument and its respective formal can be made. If a formal has a default value, that value will appear as the argument should the instantiation not provide a value for that argument.

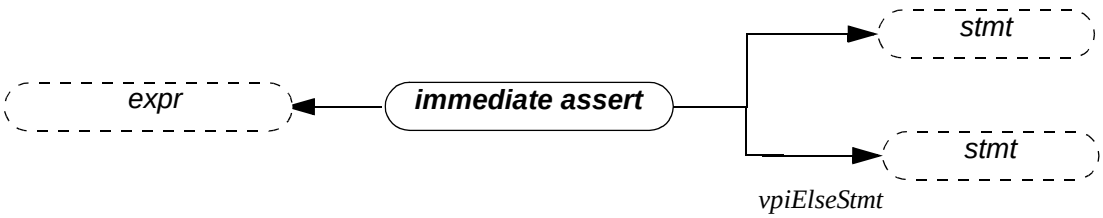
31.35 Sequence Expression



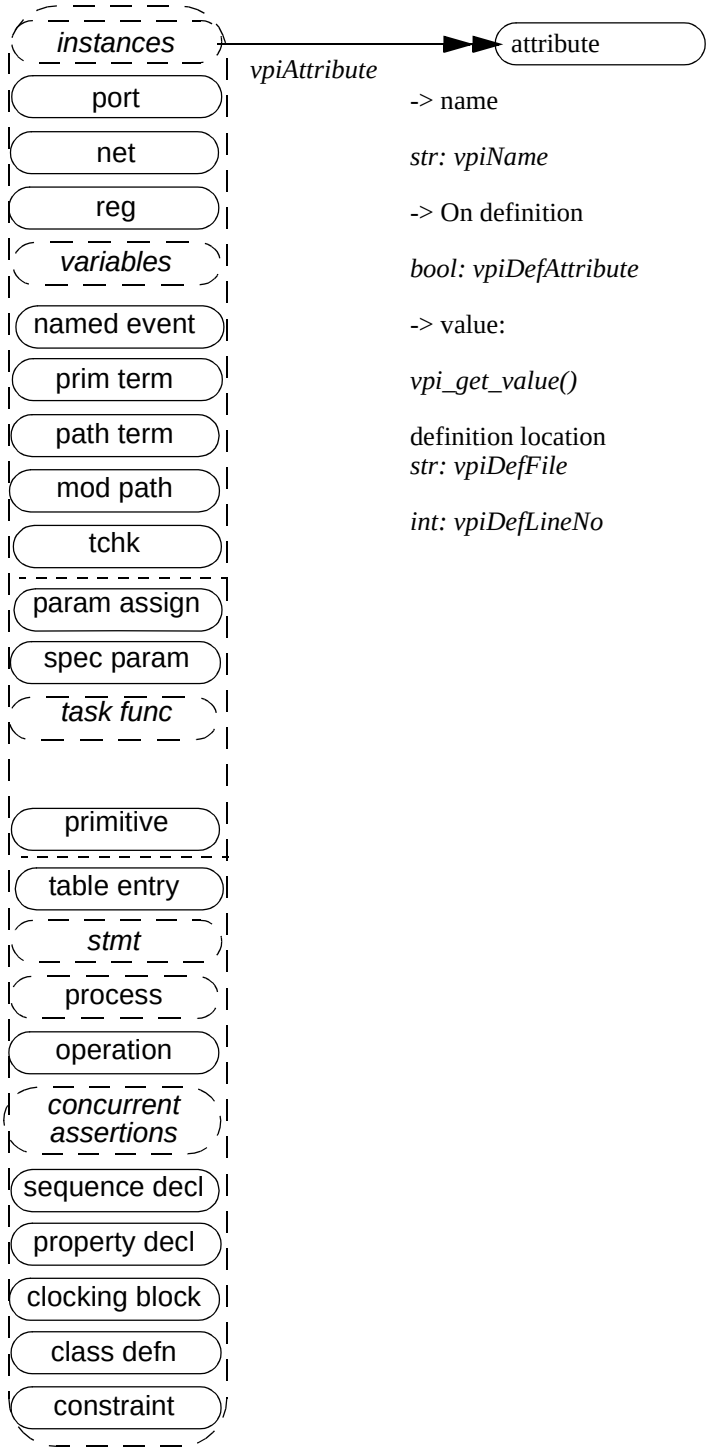
Notes:

2. Within a sequence expression, **vpiOpType** can be any one of vpiAndOp, vpiIntersectOp, vpiOr, vpiFirstMatchOp, vpiThroughoutOp, vpiWithinOp, vpiUnaryCycleDelayOp, vpiCycleDelayOp, vpiRepeatOp, vpiConsecutiveRepeatOp or vpiGotoRepeatOp.
3. For operations, the operands are provided in the same order as the operands appear in BNF, with the following exceptions:
   vpiUnaryCycleDelayOp: arguments will be: sequence, left range, right range. Right range will only be given if different than left range.
   vpiCycleDelayOp: argument will be: lhs sequence, rhs sequence, left range, right range. Right range will only be provided if different than left range.
   all the repeat operators: the first argument will be the sequence being repeated, the next argument will be the left repeat bound, followed by the right repeat bound. The right repeat bound will only be provided if different than left repeat bound.

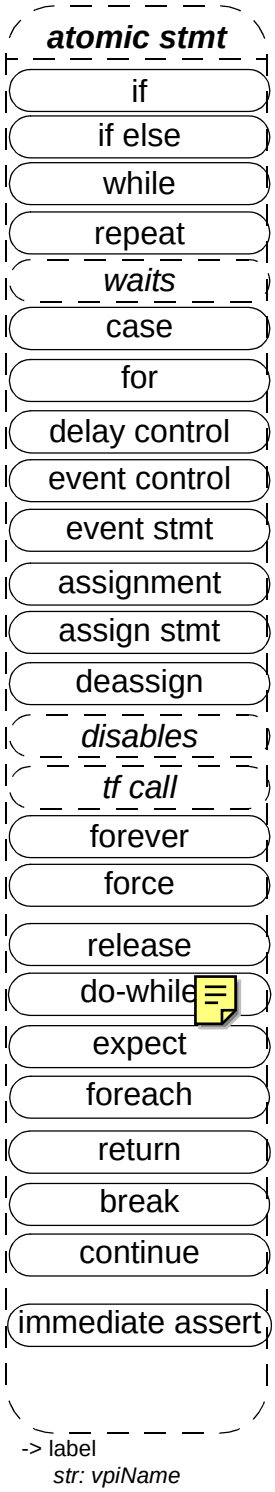
and, intersect, or,
first_match,
throughout, within,
##,
[*], [*=], [*->]



31.36 Attribute (supercedes IEEE 1364-2001 26.6.42)

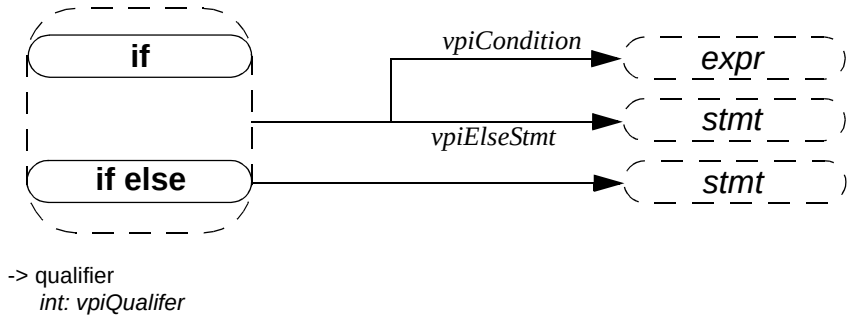


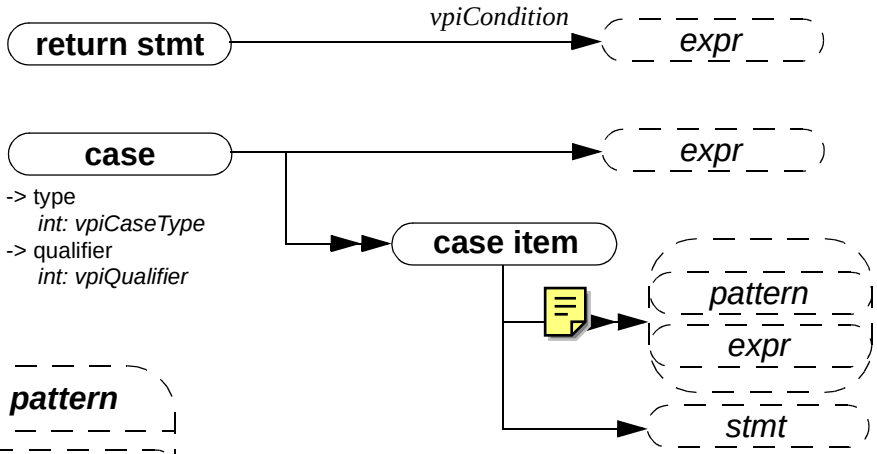
31.37 Atomic Statement (supercedes IEEE 1364-2001 26.6.27)

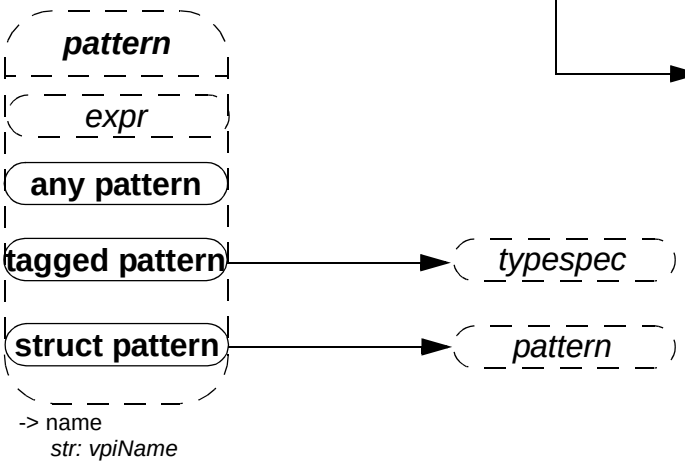


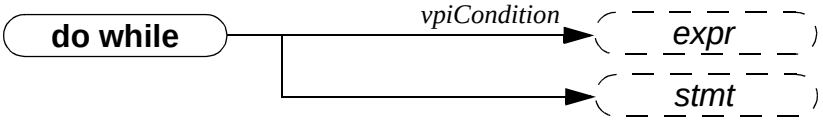
The vpiName property provides the statement label if one was given, otherwise the name is NULL.

31.38 If, if else, return, case, do while (supercedes IEEE 1364-2001 26.6.35, 26.6.36)

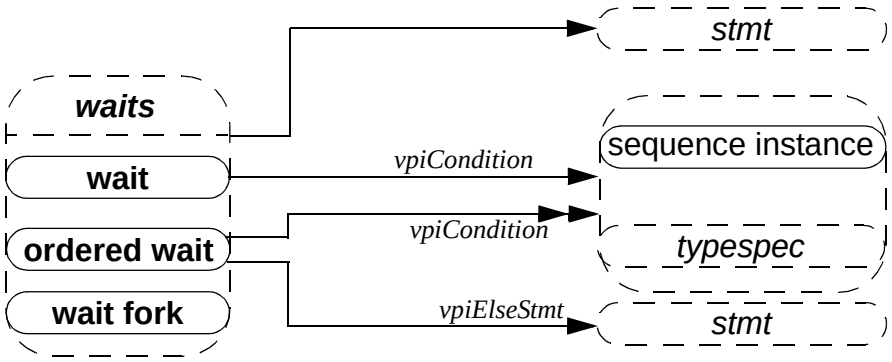


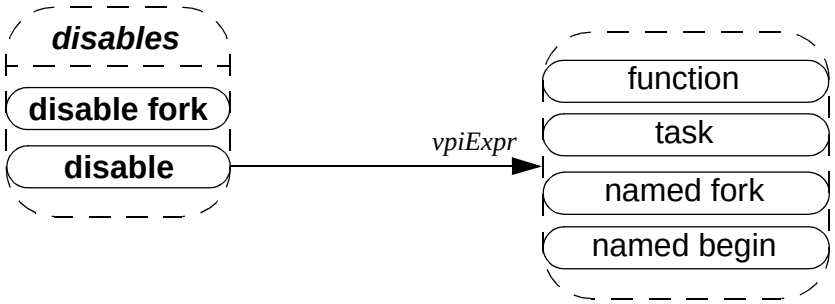


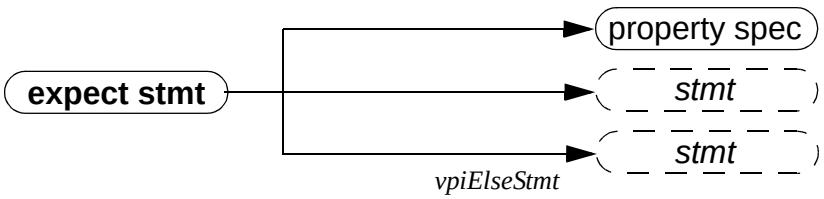


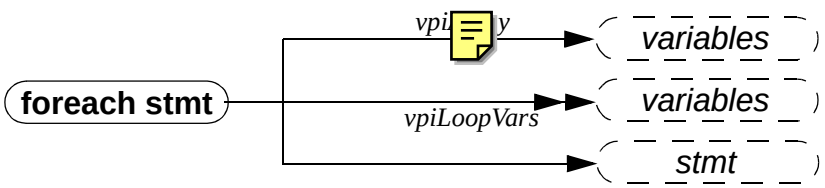


31.39 waits, disables, expect, foreach (supercedes IEEE 1364 26.6.38)

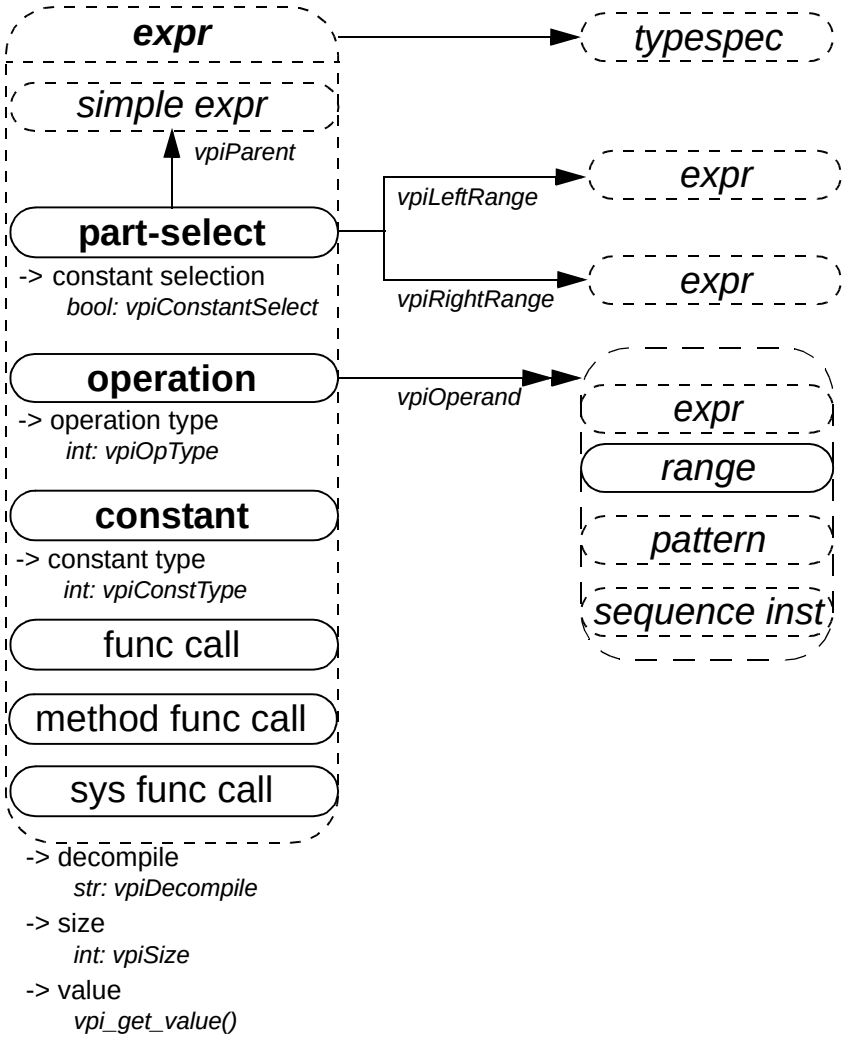








31.40 Expressions (supercedes IEEE 1364-2001 26.6.26)



NOTES:

1) For an operator whose type is **vpiMultiConcat**, the first operand shall be the multiplier expression. The remaining operands shall be the expressions within the concatenation.

2) The property **vpiDecompile** will return a string with a functionally equivalent expression to the original expression within the HDL. Parenthesis shall be added only to preserve precedence. Each operand and operator shall be separated by a single space character. No additional white space shall be added due to parenthesis.

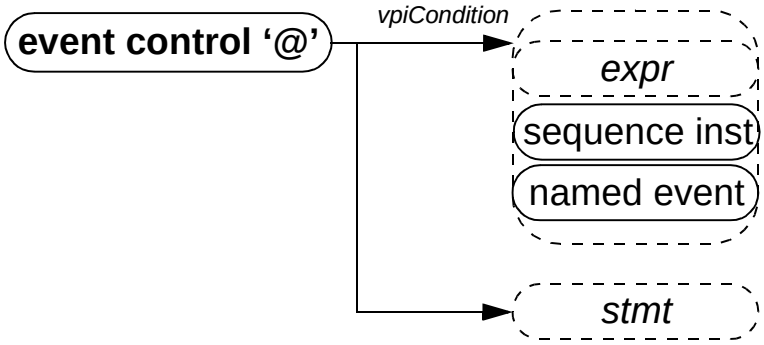
3) new vpiOpTypes: vpiInsideOp, vpiMatchOp, vpiCastOp, vpiPreIncOp, vpiPostIncOp, vpiPreDecOp, vpiPostDecOp, vpiIffOp, vpiCycleDelayOp. The cast operation is represented as a unary operation, with its sole argument being the expression being cast, and the typespec of the cast expression being the type to which the argument is being cast.

4) new vpiConstType: vpiNullConst, vpiOneStepConst, vpiUnboundedConst. The constant vpiUnboundedConst represents the **$** value used in assertion ranges.

5) the one to one relation to typespec must always be available for vpiCastOp operations and for simple expressions. For other expressions it is implementation dependent whether there is any associated typespec.

6) Variable slices are represented by part-selects whose parent simple expression is an array variable.

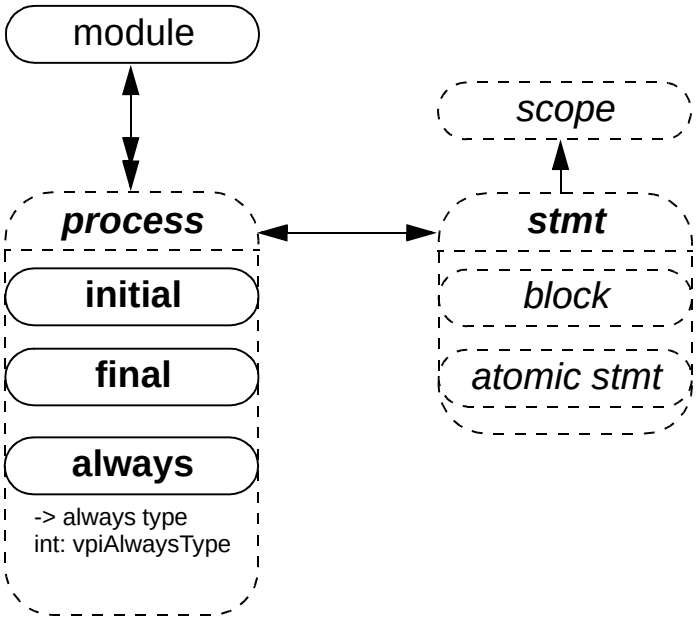
31.41 Event control (supercedes IEEE 1364-2001 26.6.30)



NOTE—For event control associated with assignment, the statement shall always be `NULL`.

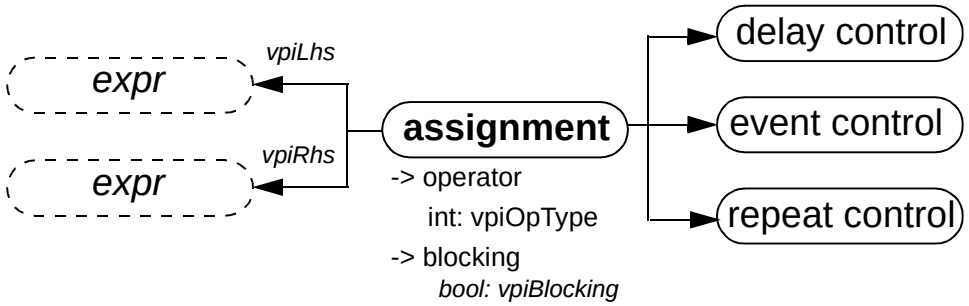
31.42 Event stmt (supercedes IEEE 1364-2001 26.6.27)

31.43 Process (supercedes IEEE 1364-2001 26.6.27)



NOTE- vpiAlwaysType can be one of: vpiAlwaysComb, vpiAlwaysFF, vpiAlwaysLatch

31.44 Assignment (supercedes IEEE 1364-2001 26.6.28)



NOTE: vpiOpType will return vpiAssignmentOp for normal non-blocking ’=’ assignments, and the operator combined with the assignment for the operators described in section 7.3.

For example, the assignment

```
a[i] += 2;
```

will return vpiAddOp for the vpiOpType property.