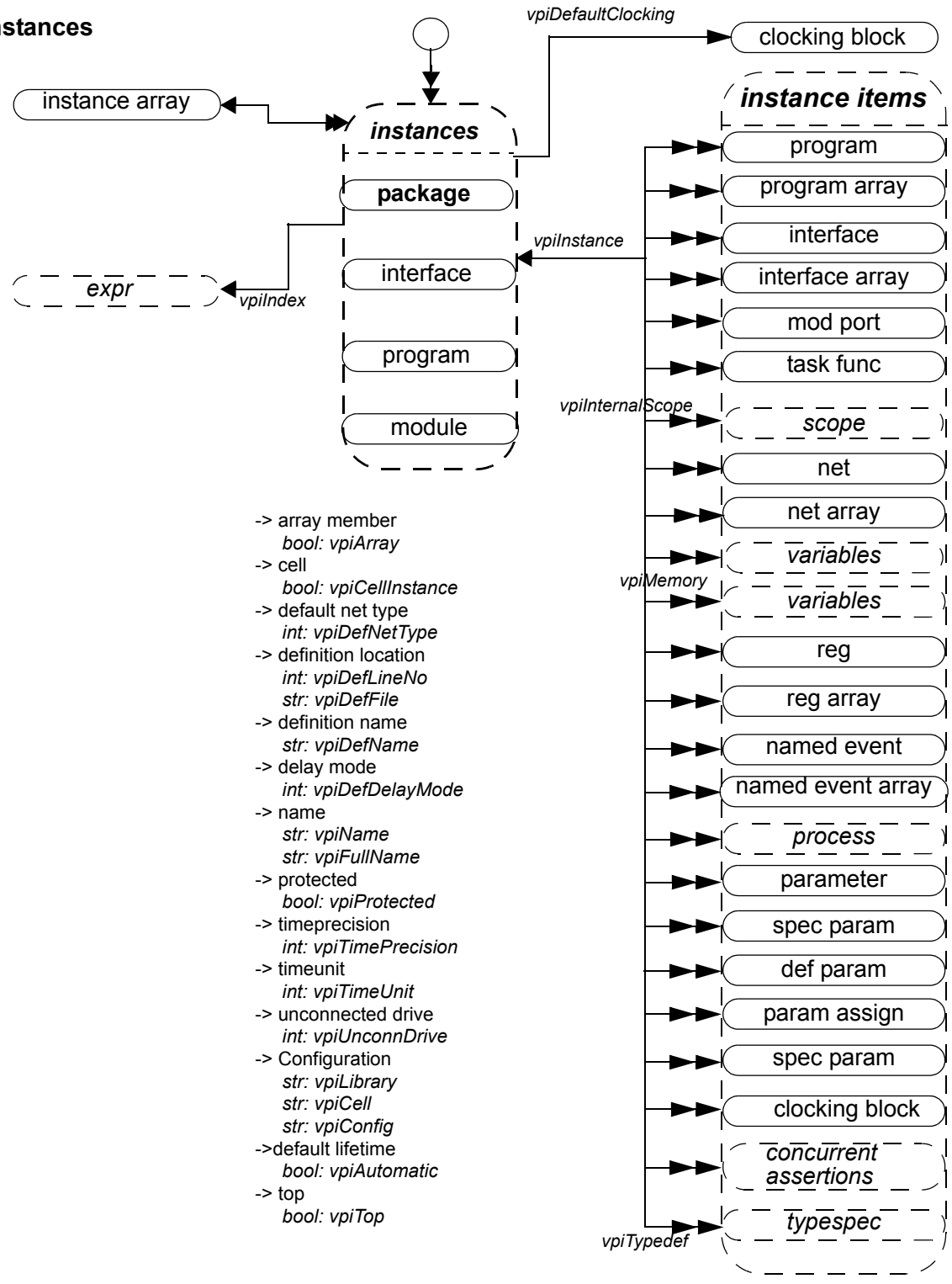
VPI Extensions to SystemVerilog

January 12, 2004

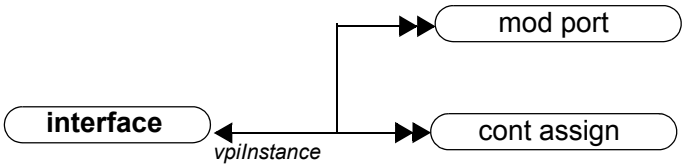
Instances


vpiDefaultClocking
clocking block
instance array
instances
instance items
program
program array
package
interface
interface array
vpiInstance
expr
vpiIndex
interface
mod port
task func
program
vpiInternalScope
scope
module
net
net array
variables
vpiMemory
variables
reg
reg array
named event
named event array
process
parameter
spec param
def param
param assign
spec param
clocking block
concurrent assertions
typespec
vpiTypedef
-> 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

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.
4. 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.

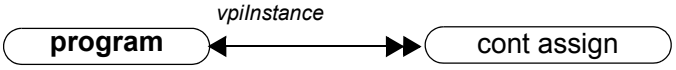
Interface



NOTE

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

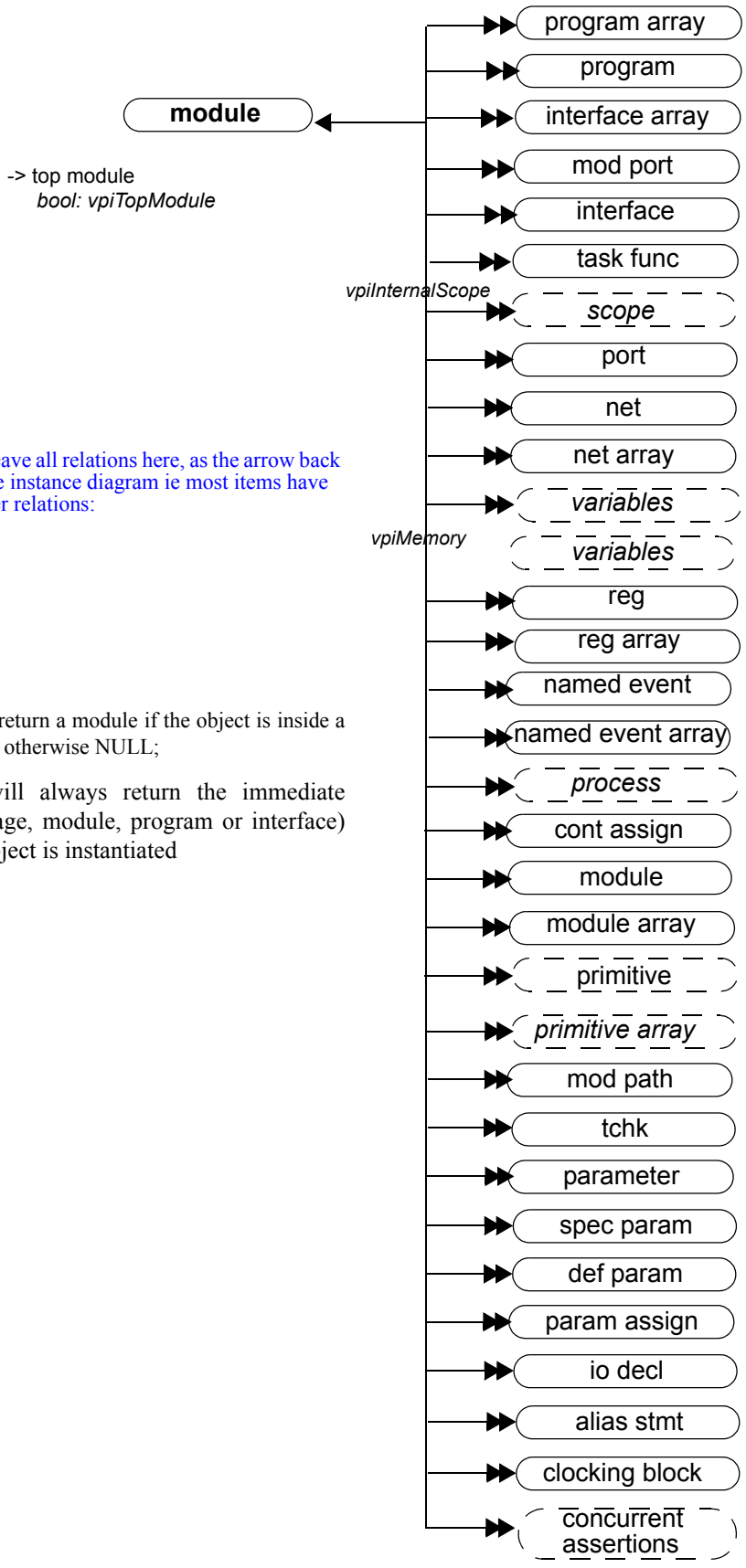
Program



NOTE

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

Module (26.6.1)

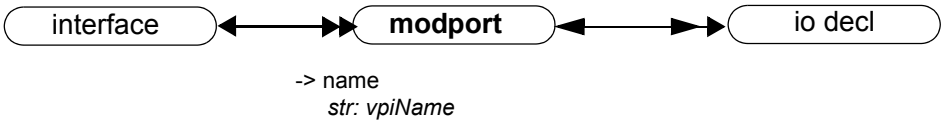


NOTE to reviewers leave all relations here, as the arrow back is different than in the instance diagram ie most items have two possible container relations:
vpiModule
vpiInstance

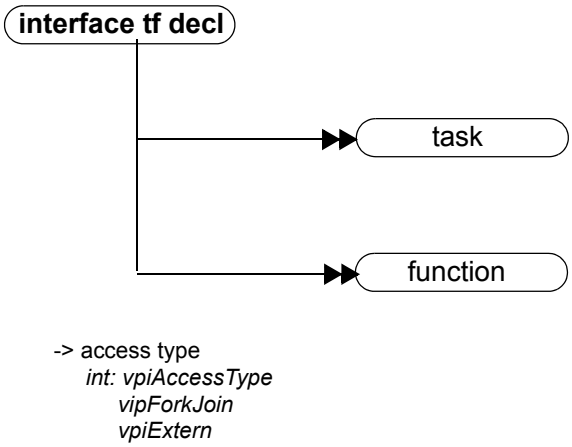
NOTES

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

Modport



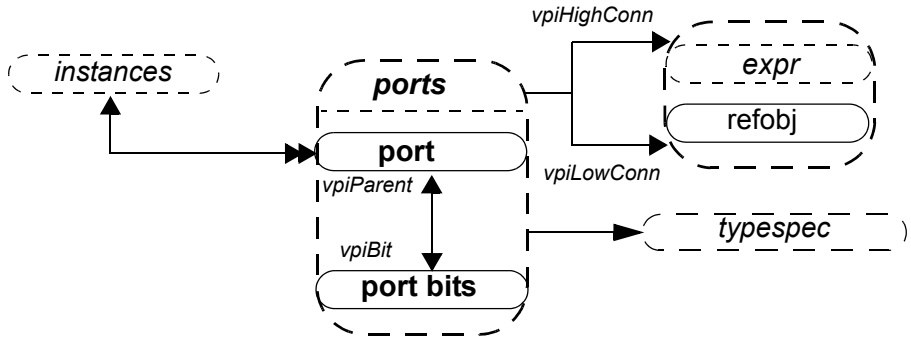
Interface tf decl



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.

Ports (26.6.5)

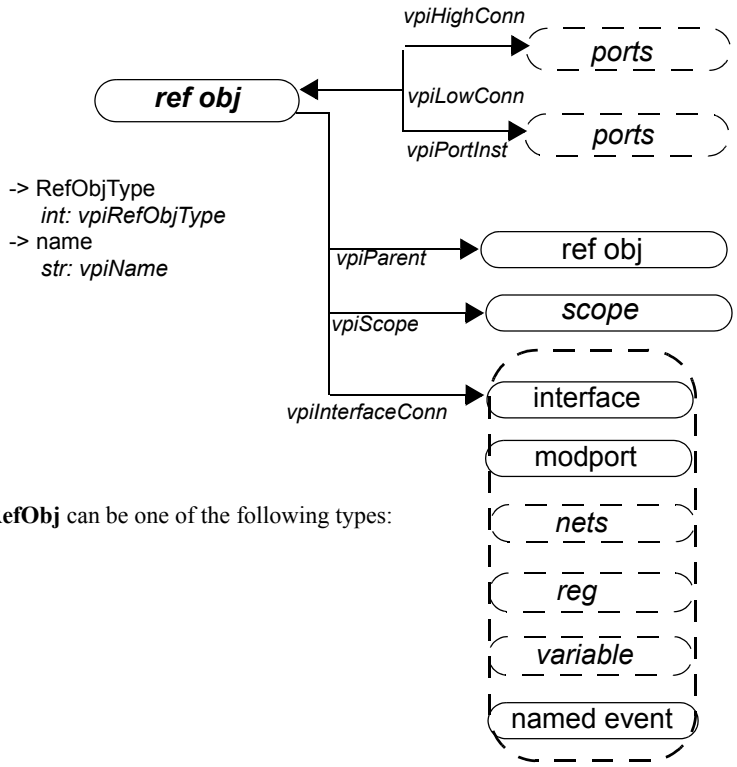


-> connected by name
bool: vpiConnByName
-> delay (mipd)
vpi_get_delays()
vpi_put_delays()
-> direction
int: vpiDirection
-> explicitly named
bool: vpiExplicitName
-> index
int: vpiPortIndex
-> name
str: vpiName
-> port type
int: vpiPortType
-> scalar
bool: vpiScalar
-> size
int: vpiSize
-> vector
bool: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



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

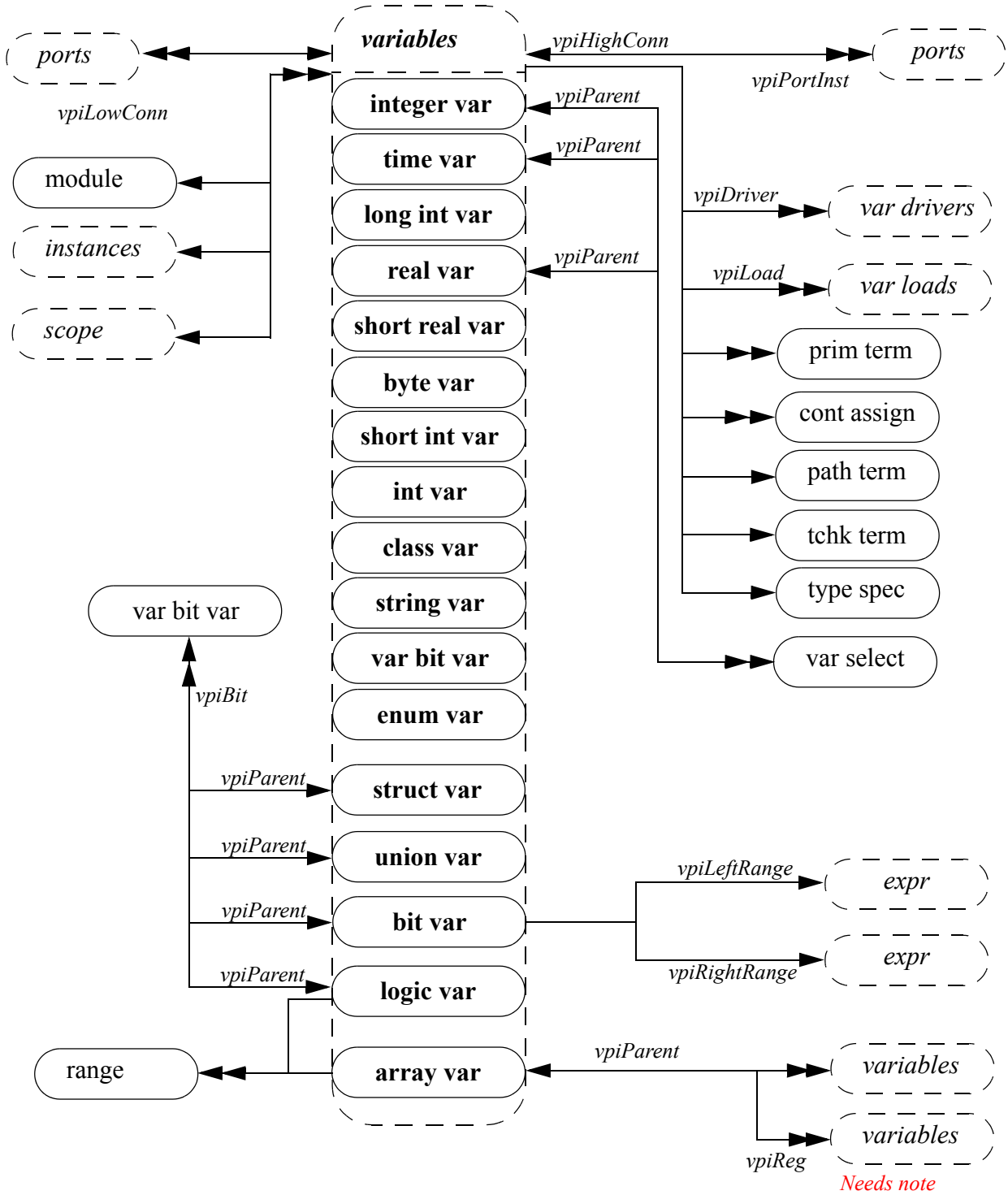
These objects are newly defined objects needed for supporting the full connectivity through ports where the ports are vpiInterface or vpiModport or any object inside modport or interface.

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

```
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);
```

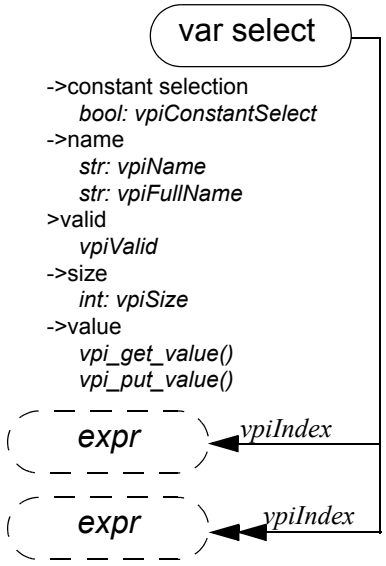
```
endmodule

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

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.
```

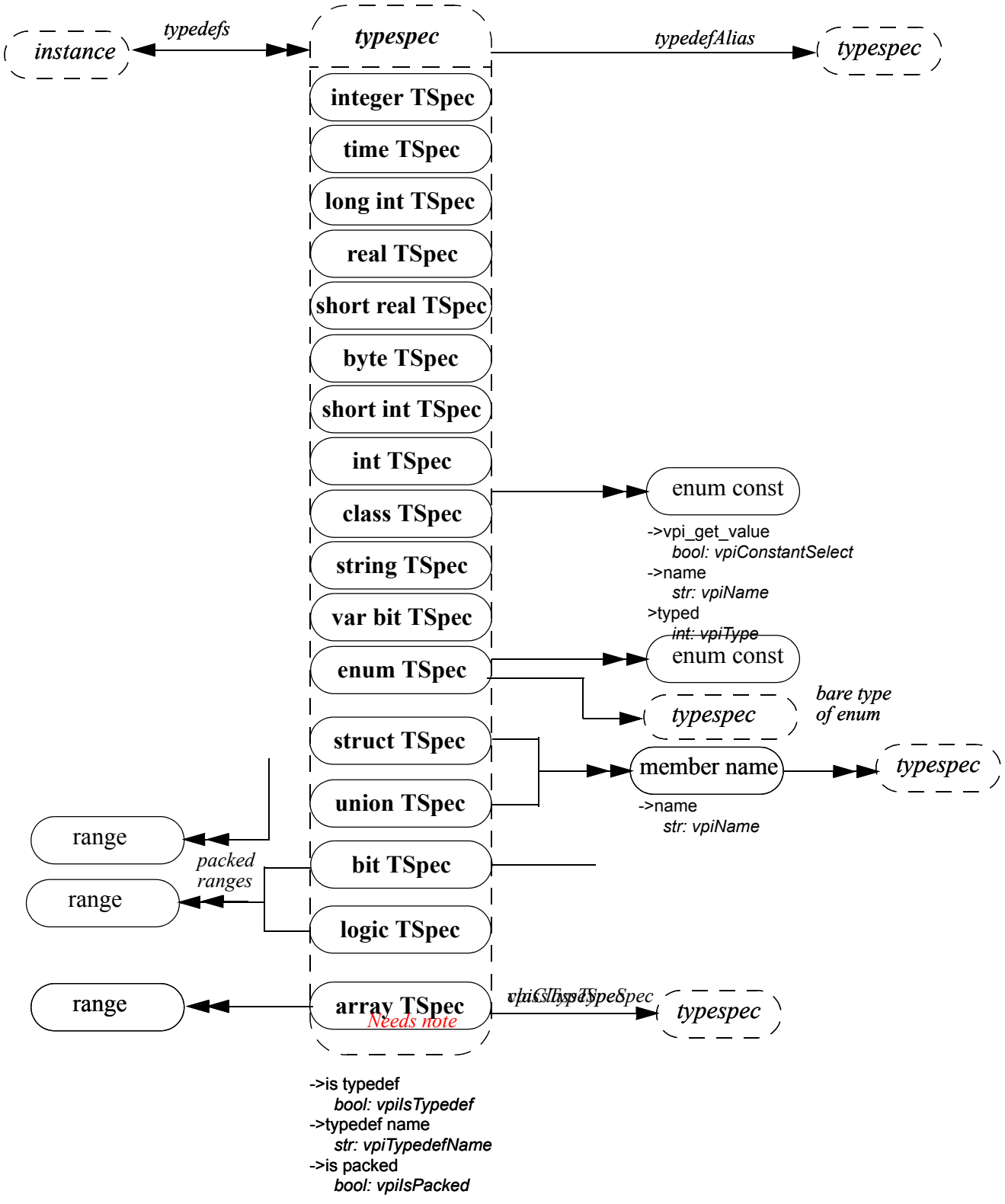
*block*Variable (26.6.8)



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 randomiztion.
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




instance
typedefs
typespec
typedefAlias
typespec
integer TSpec
time TSpec
long int TSpec
real TSpec
short real TSpec
byte TSpec
short int TSpec
int TSpec
class TSpec
string TSpec
var bit TSpec
enum TSpec
struct TSpec
union TSpec
bit TSpec
logic TSpec
array TSpec
enum const
->vpi_get_value
bool: vpiConstantSelect
->name
str: vpiName
>typed
int: vpiType
enum const
typespec
bare type
of enum
member name
typespec
->name
str: vpiName
range
range
packed
ranges
range
Needs note
typespec
->is typedef
bool: vpiIsTypedef
->typedef name
str: vpiTypedefName
->is packed
bool: vpiIsPacked

NOTES

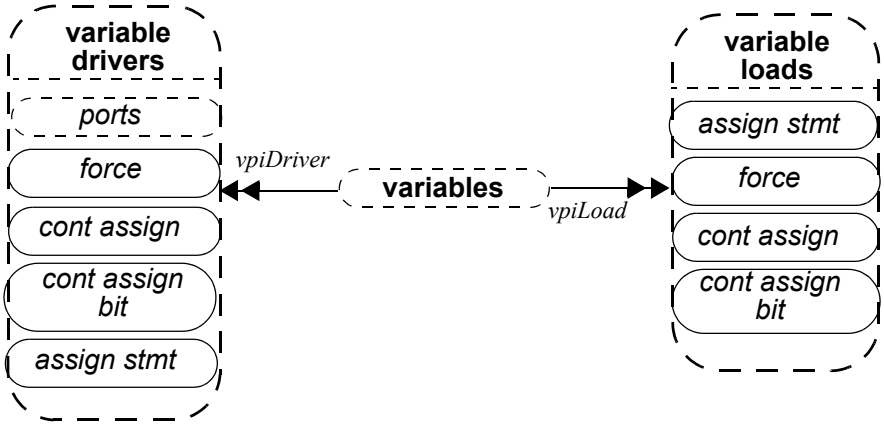
1. Typespec to typespec relation is used when the **vpiTypedefType** is "vpiTypedef", which will be the case for type aliases, for example, typedef a b;

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:

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

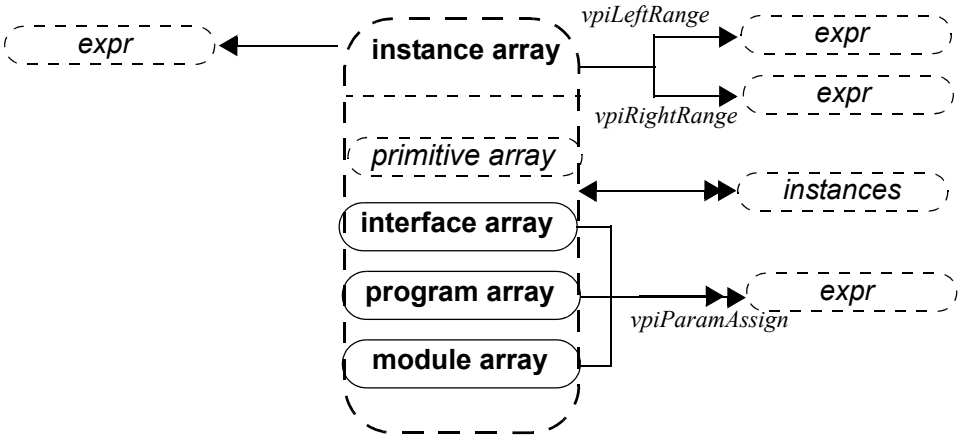
The typedef C has **vpiTypedefType vpiStruct**, a single field named B with **vpiTypedefType vpiStruct**. Obtaining the typedef of field B, you will obtain a typedef with no name and a single field, named "a" with **vpiTypedefType** of **vpiInt**.**Variable Drivers and Loads**



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.

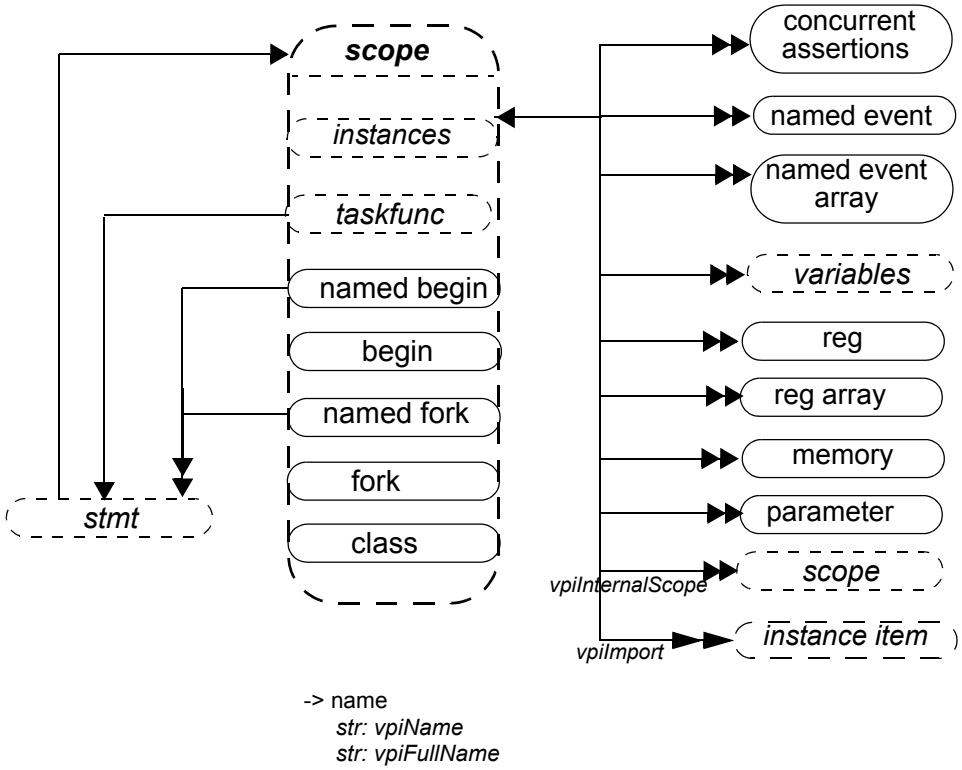
Instance Arrays (26.6.2)



NOTE

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

Scope (26.6.3)

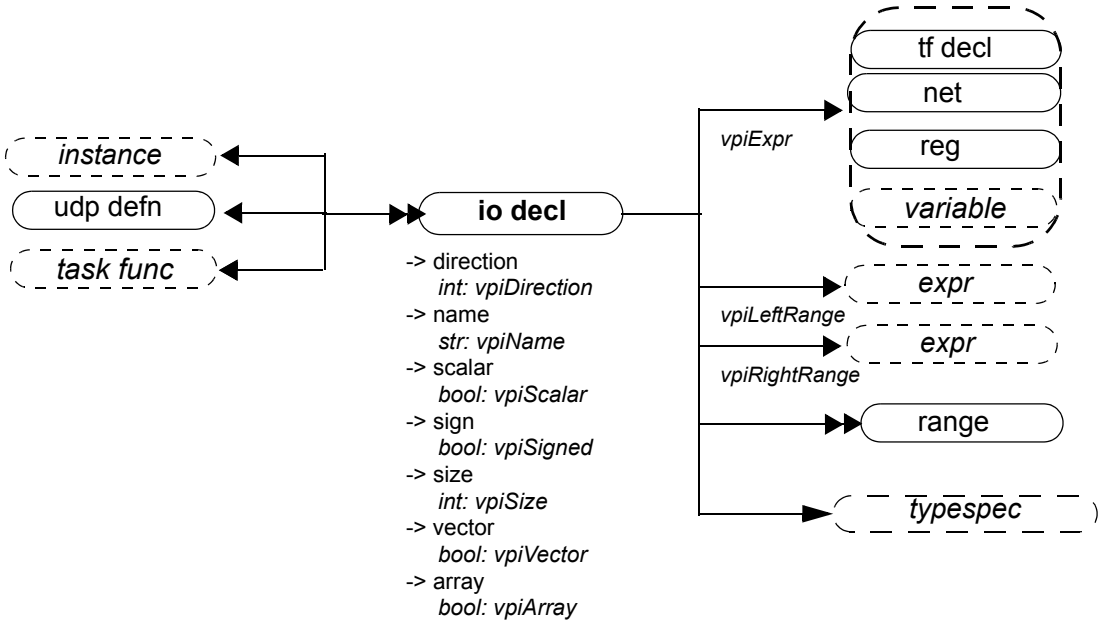


NOTE

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.

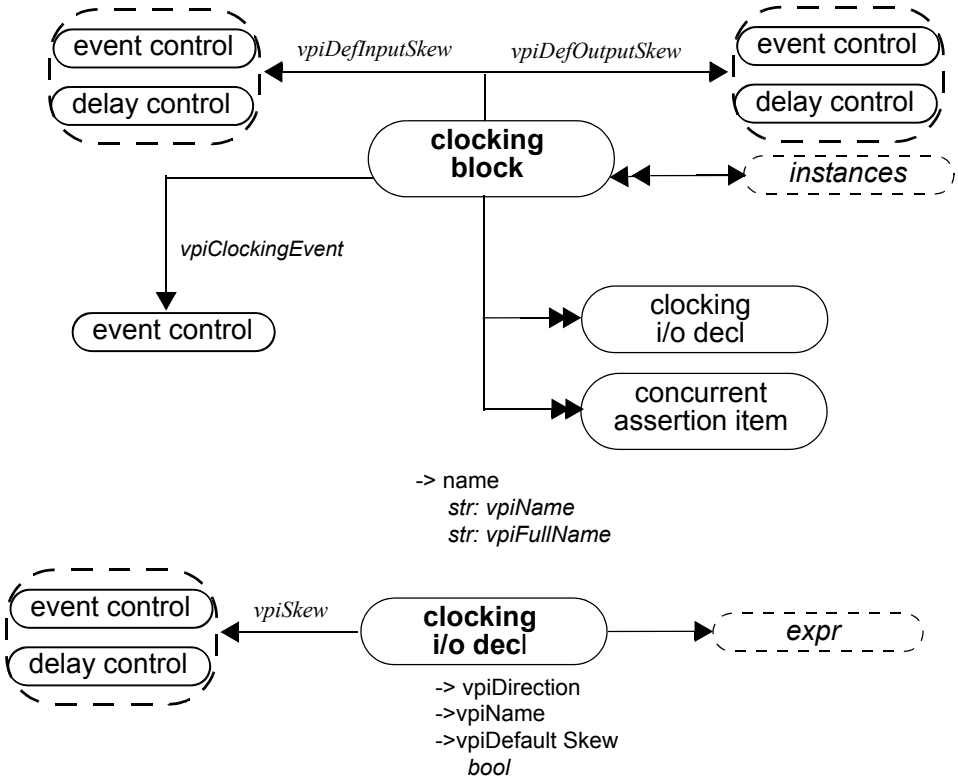
IO declaration (26.6.4)



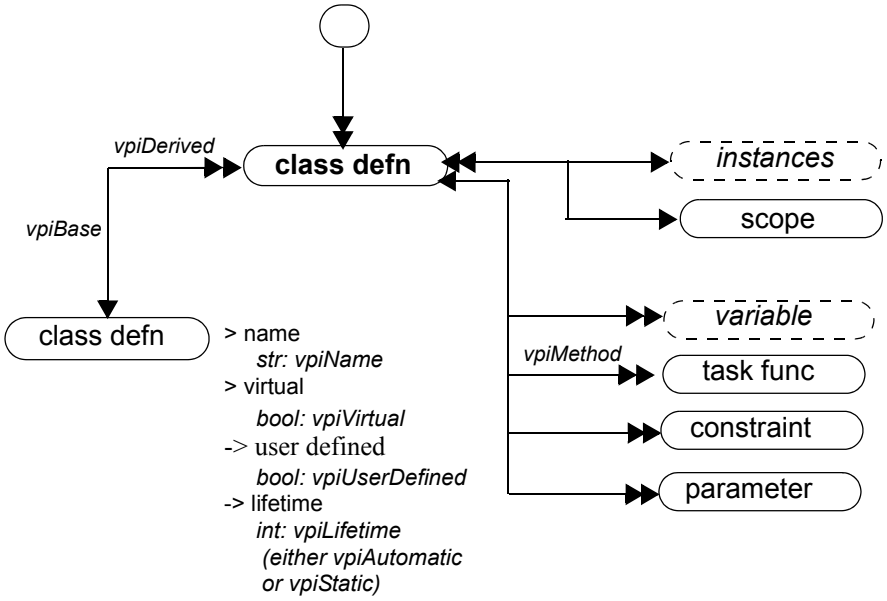
NOTE

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

clocking block



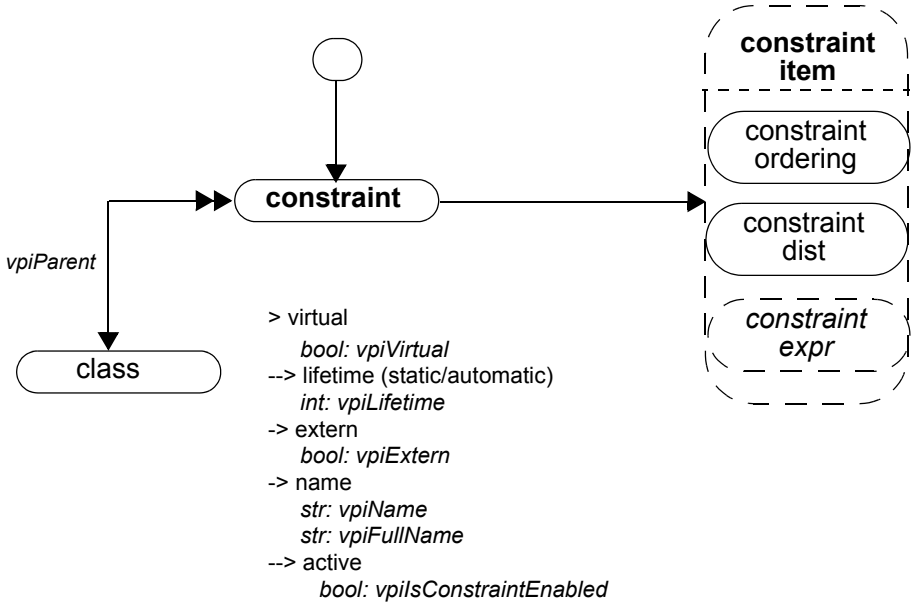
Class Object Definition

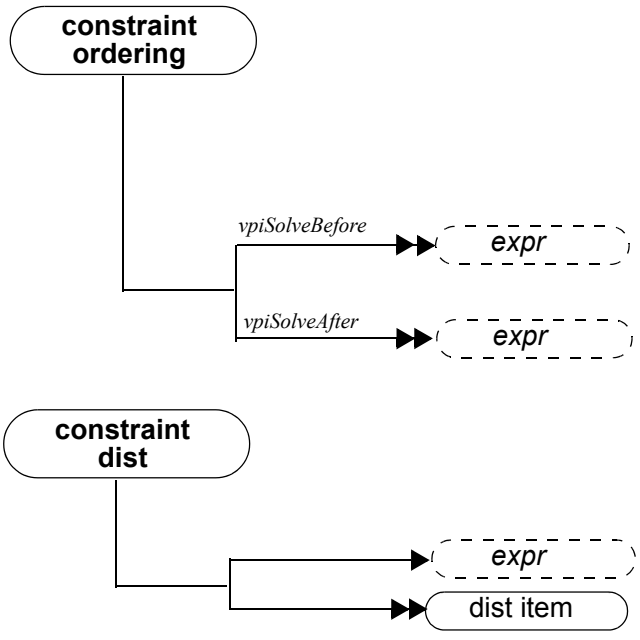


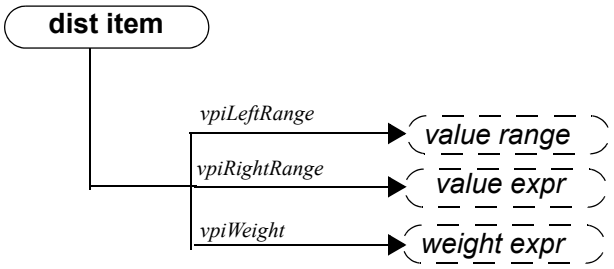
NOTE

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.

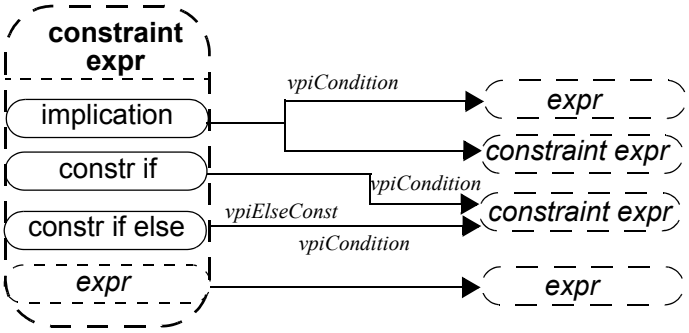
Constraint


constraint item
constraint ordering
constraint dist
constraint expr
constraint
vpiParent
class
> virtual
bool: vpiVirtual
--> lifetime (static/automatic)
int: vpiLifetime
-> extern
bool: vpiExtern
-> name
str: vpiName
str: vpiFullName
--> active
bool: vpiIsConstraintEnabled

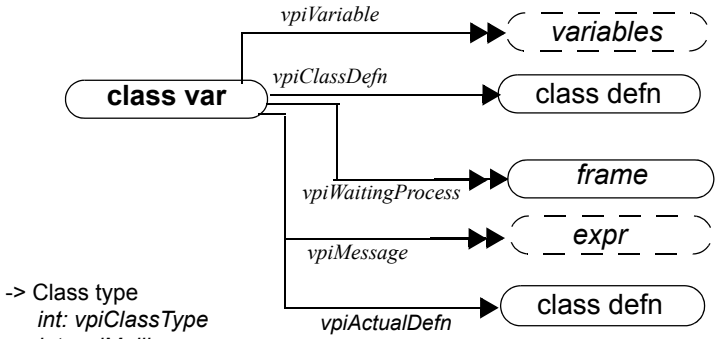

constraint ordering
vpiSolveBefore
expr
vpiSolveAfter
expr
constraint dist
expr
dist item



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



Class Variables

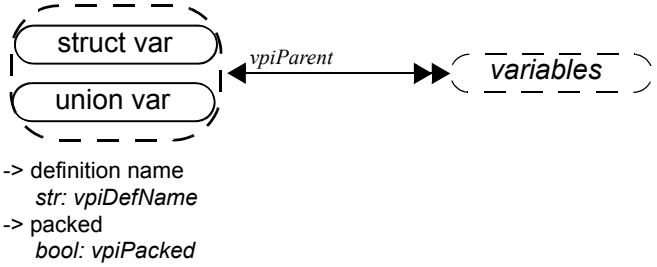


-> Class type
int: vpiClassType
int: vpiMailbox
int: vpiSemaphore
-> associative array
bool: vpiAssociativeArray
-> index type
vpiIndexType
-> dynamic array
bool: vpiDynamicArray
-> access type
int: vpiAccessType
int: vpiPublic
int: vpiPrivate
int: vpiProtected

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.

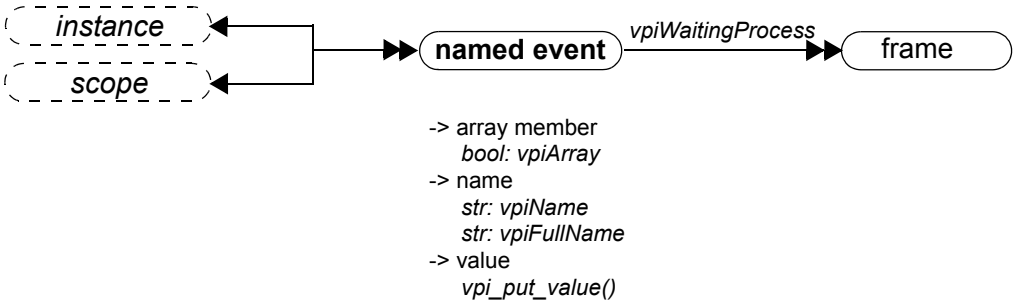
Structure/Union



NOTES

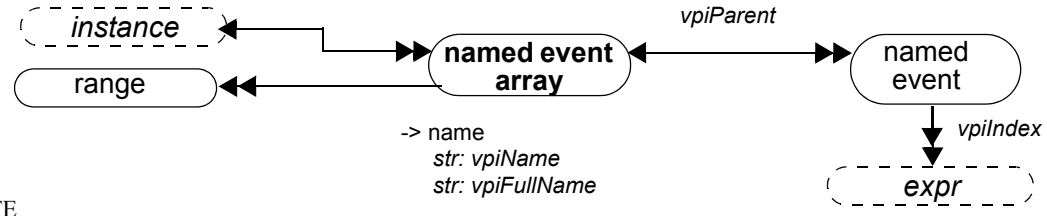
vpi_get_value/vpi_put_value cannot be used to access values of entire unpacked structures and unpacked unions.

Named Events



NOTE

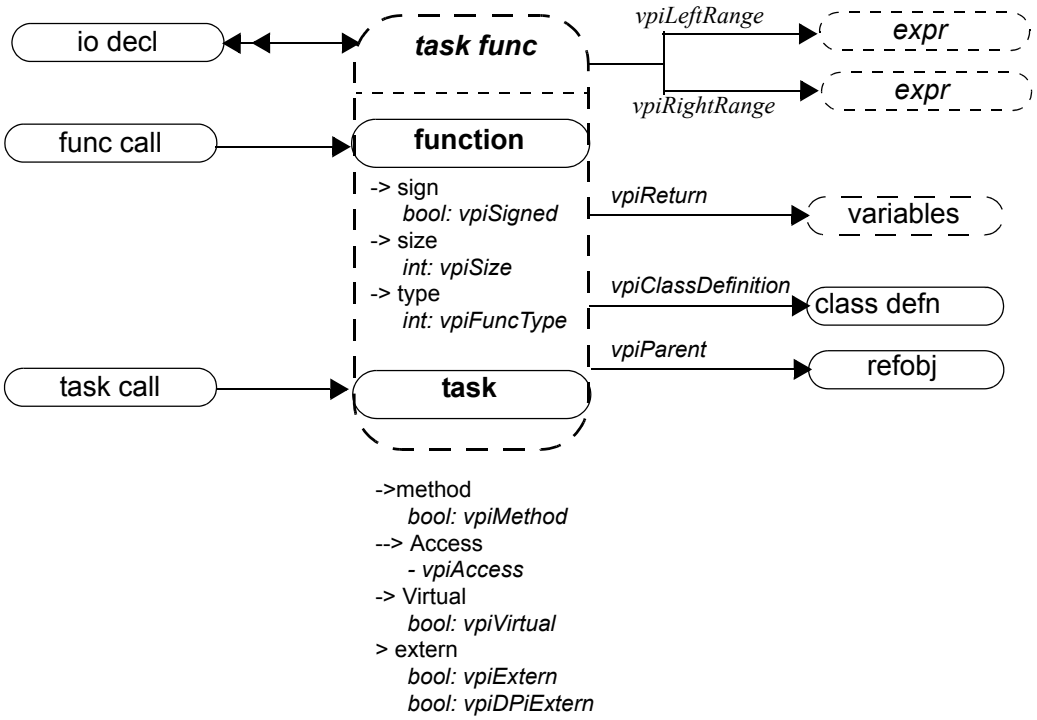
The new iterator (**vpiWaitingProcess**) returns all waiting processes, identified by their frame, for that namedEvent.



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.

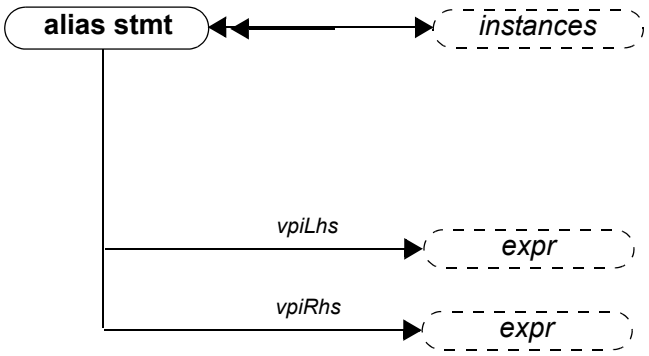
Task, Function Declaration (26.6.18)



NOTE

1. A Verilog HDL function shall contain an object with the same name, size, and type as the function.
2. **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.

Alias Statement



Examples

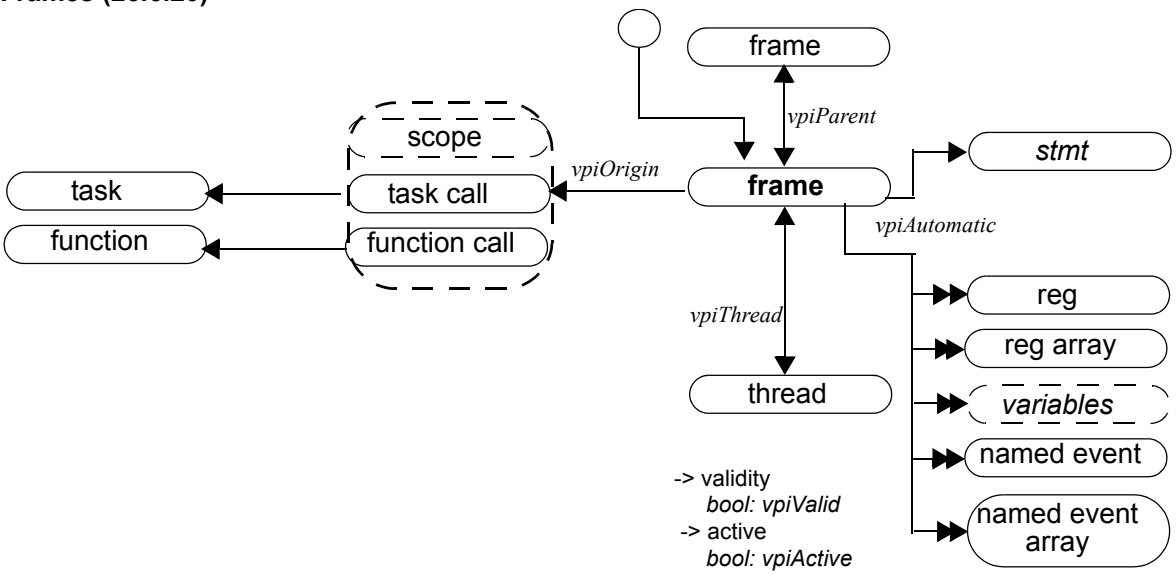
```
alias a=b=c=d
```

Results in 3 aliases:

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

d is Rhs for all.

Frames (26.6.20)

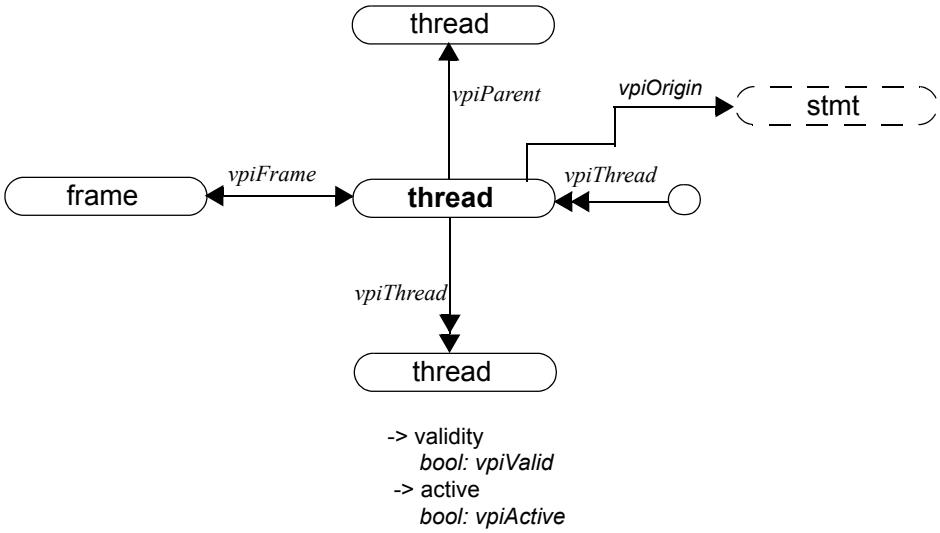


NOTES

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.

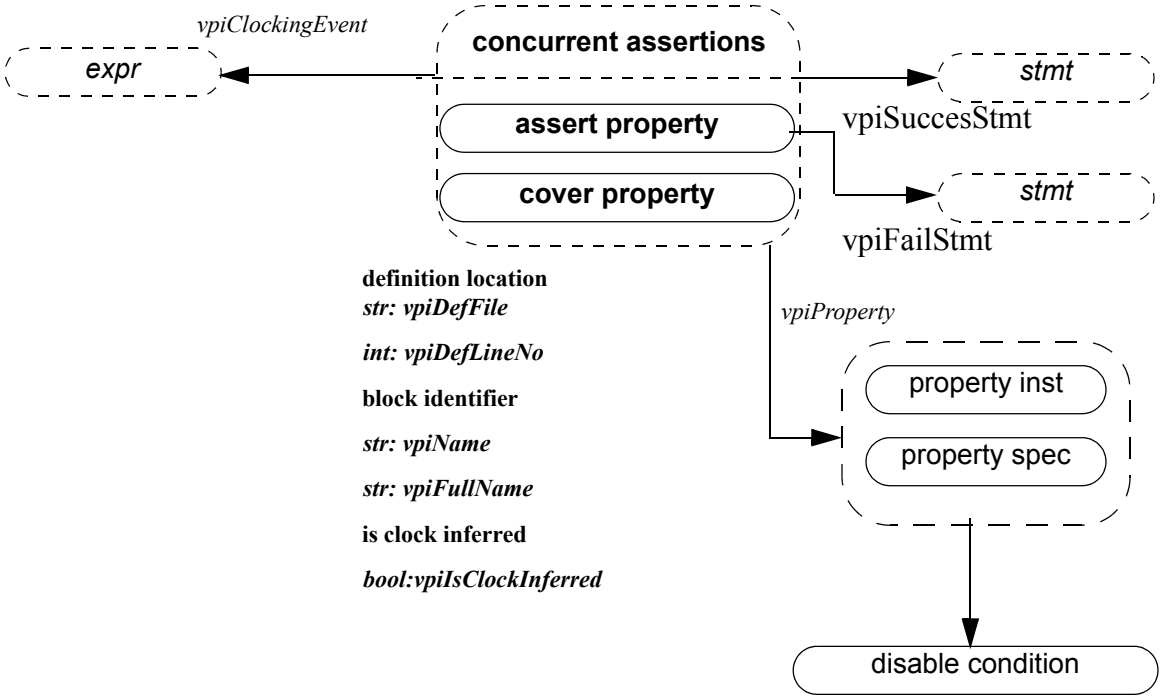
Threads



NOTES

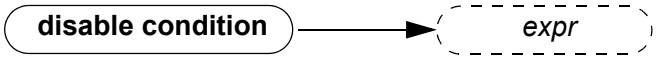
The following callbacks shall be supported on threads

- **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)



definition location
int: vpiDefLineNo
str: vpiDefFile

name (clocking identifier)

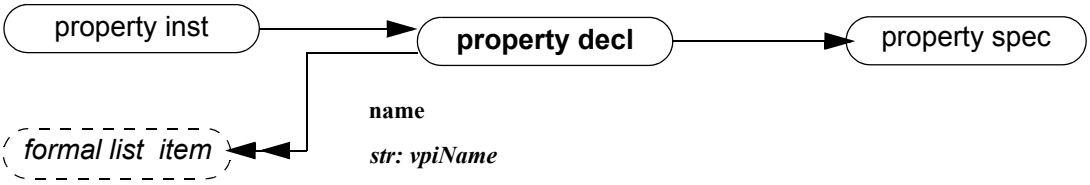
str: vpiName

str: vpiFullName

definition location
int: vpiDefLineNo
str: vpiDefFile

inferred or declared

bool: vpiInferred



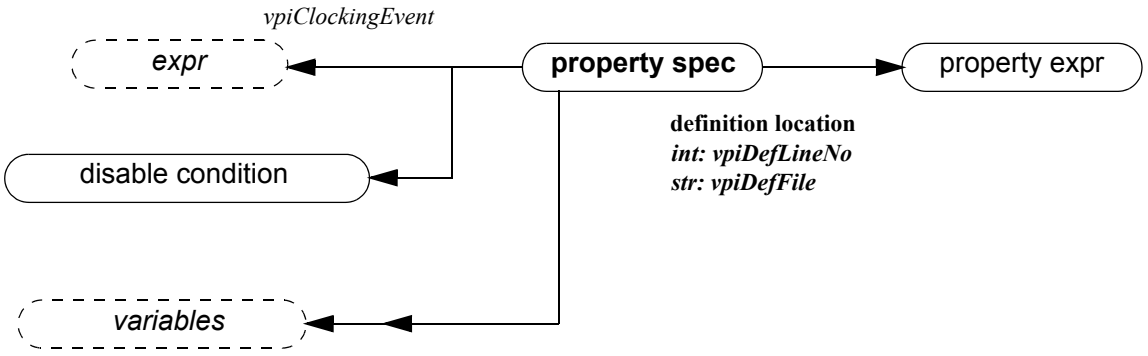
name

str: vpiName

str: vpiFullName

definition location
str: vpiDefFile

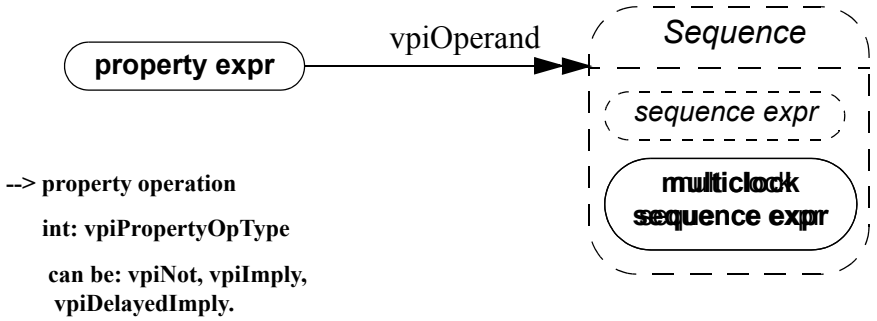
int: vpiDefLineNo

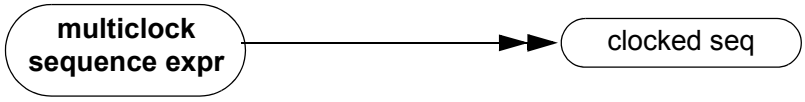


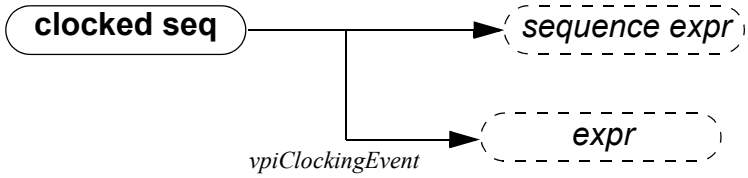
NOTE

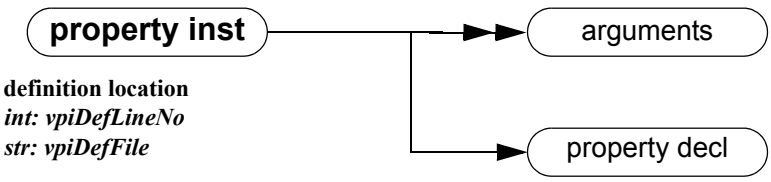
Variables are declarations of property variables. You cannot get the value of these variables.

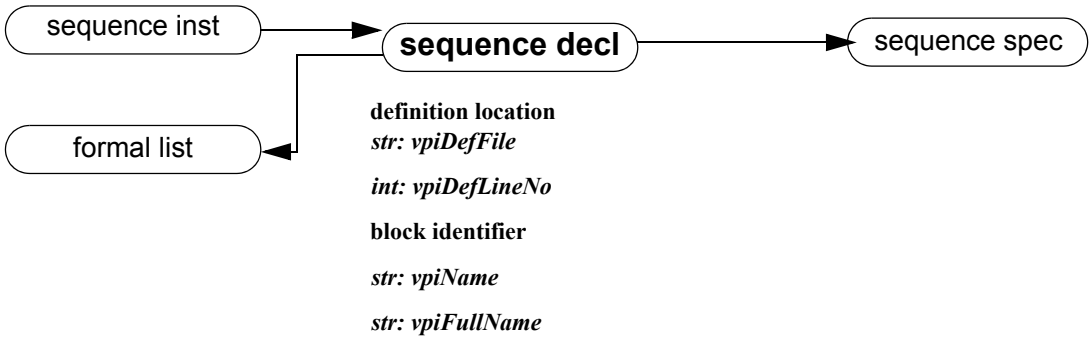
Note that the sequence bubble will be as already drawn in this diagram, but only one of them.

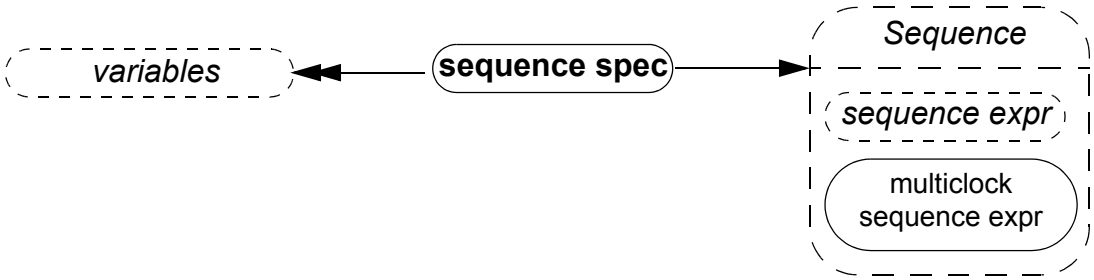


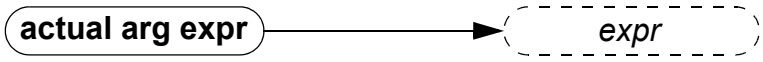

multiclock
sequence expr
clocked seq


clocked seq
sequence expr
expr
vpiClockingEvent


property inst
arguments
property decl
definition location
int: vpiDefLineNo
str: vpiDefFile







connected by name
bool: vpiConnectByName

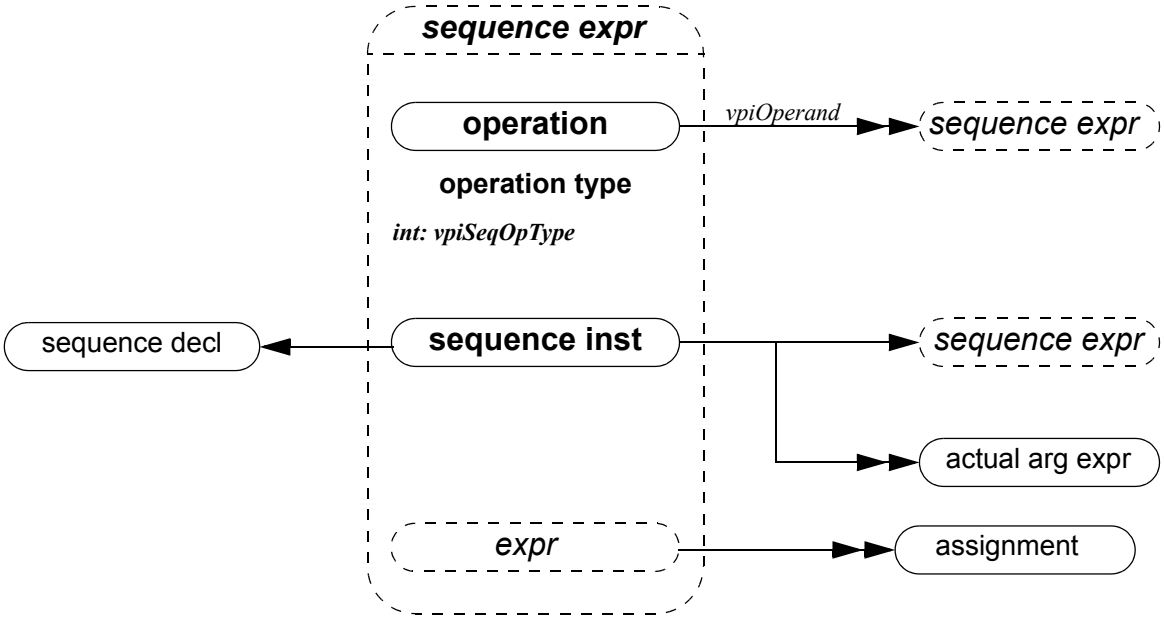
explicitly named

bool: vpiExplicitName

argument index

int: vpiPortIndex

name



***int: vpiSeqOpType* is one of:**

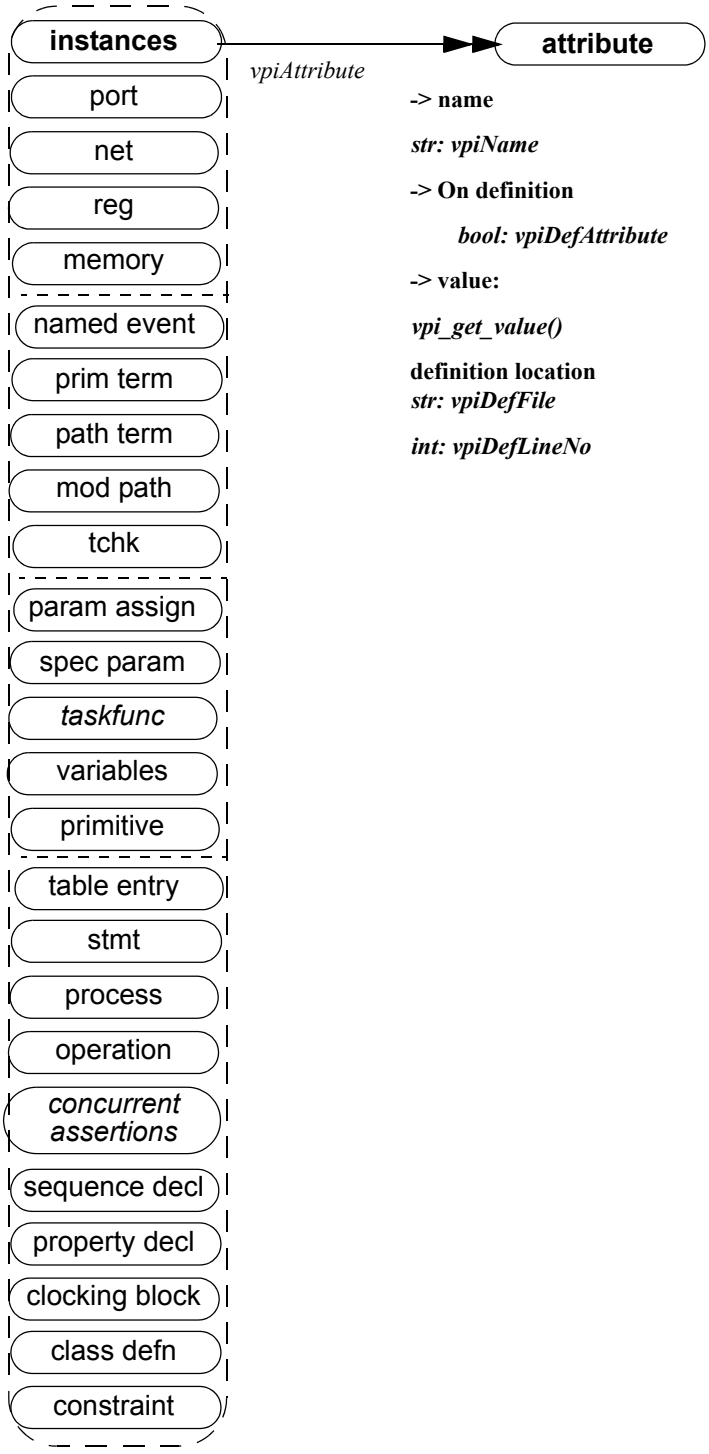
and, intersect, or,

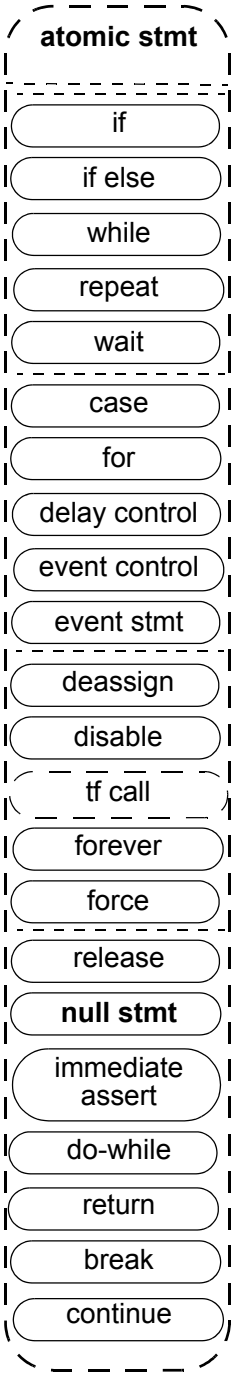
first_match,

throughout, within,

##,

[*], [*=], [*->]


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
vpiAttribute
attribute
-> name
str: vpiName
-> On definition
bool: vpiDefAttribute
-> value:
vpi_get_value()
definition location
str: vpiDefFile
int: vpiDefLineNo


atomic stmt
if
if else
while
repeat
wait
case
for
delay control
event control
event stmt
deassign
disable
tf call
forever
force
release
null stmt
immediate assert
do-while
return
break
continue

vpi_handle_by_name()			
Synopsys:	Get a handle to an object with a specific name.		
Syntax:	`vpi_handle_by_name(name, refhandle)`		
Returns:	**Type**	Description	
	vpiHandle	Handle to an object	
Arguments:	**Type**	**Name**	Description
	PLI_BYTE8 *	name	A character string or pointer to a string containing the name of an object
	vpiHandle	refhandle	Can be a HDl scope or a typedefinition object or a class/structure/union instance handle

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.