VPI Extensions to SystemVerilog

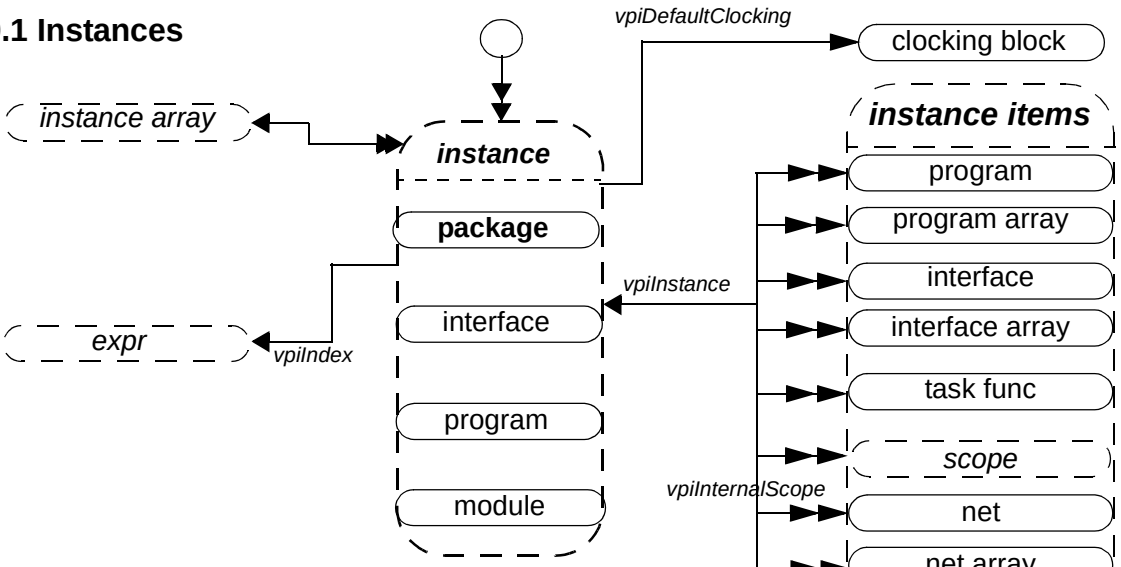
January 21, 2004

30 SystemVerilog VPI Object Model

Contents

Contents

30.1 Instances

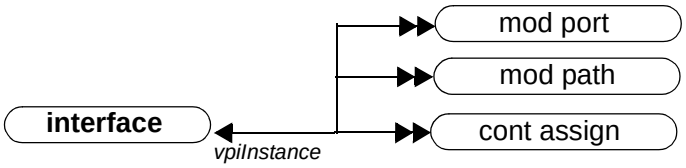


-> array member
bool: vpiArray
-> cell
bool: vpiCellInstance
-> default net type
int: vpiDefNetType
-> definition location
int: vpiDefLineNo
str: vpiDefFile
-> definition name
str: vpiDefName
-> delay mode
int: vpiDefDelayMode
-> name
str: vpiName
str: vpiFullName
-> protected
bool: vpiProtected
-> timeprecision
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.
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.
3. 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.
4. Compilation units are represented as packages that have a **vpiUnit** property set to TRUE. Such implicitly declared packages shall have implementation dependent names.

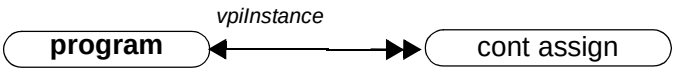
30.2 Interface



NOTE

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

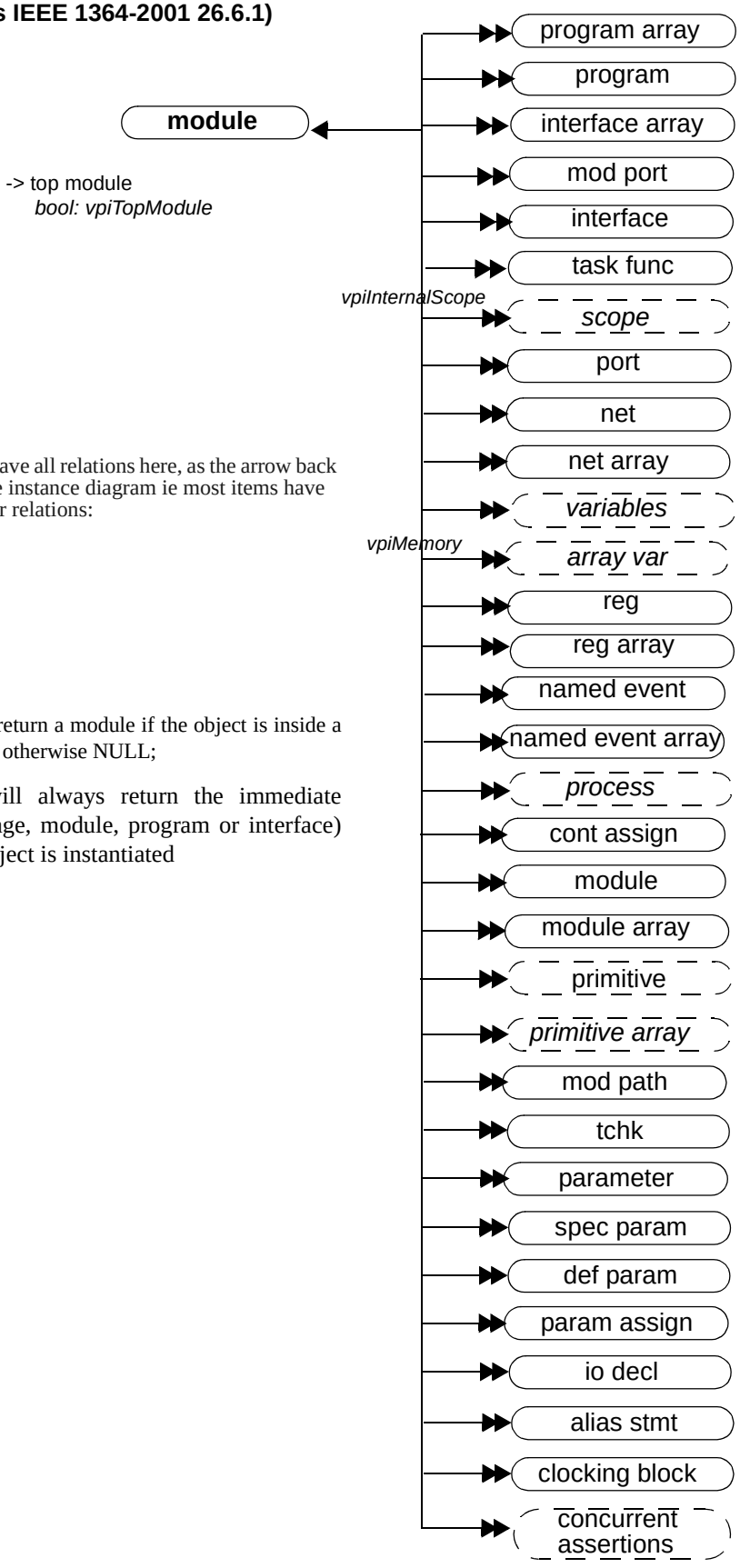
30.3 Program



NOTE

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

30.4 Module (supercedes IEEE 1364-2001 26.6.1)

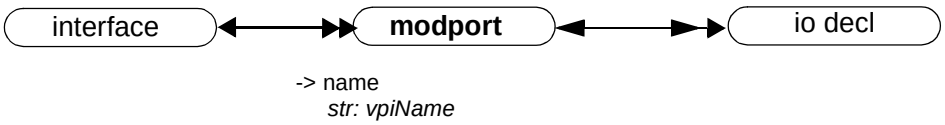


NOTE to reviewers leave all relations here, as the arrow back is different than in the instance diagram ie most items have two possible container relations:
vpiModule
vpiInstance

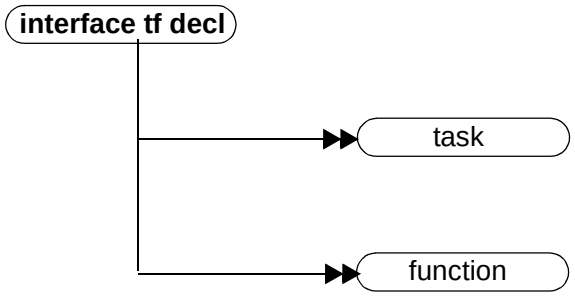
NOTES

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

30.5 Modport



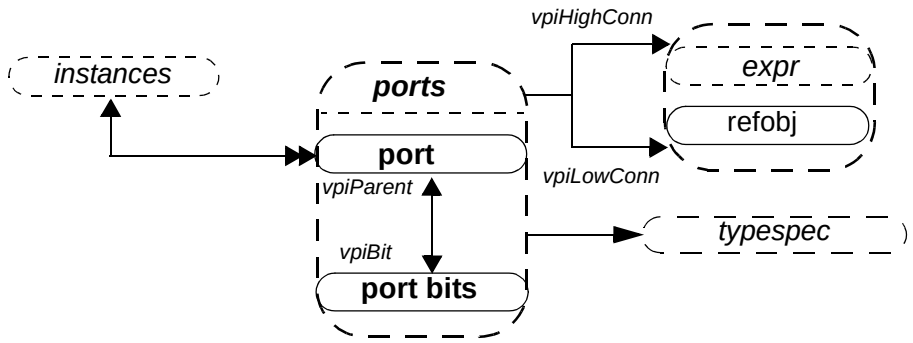
30.6 Interface tf decl



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.

30.7 Ports (supercedes IEEE 1364-2001 26.6.5)

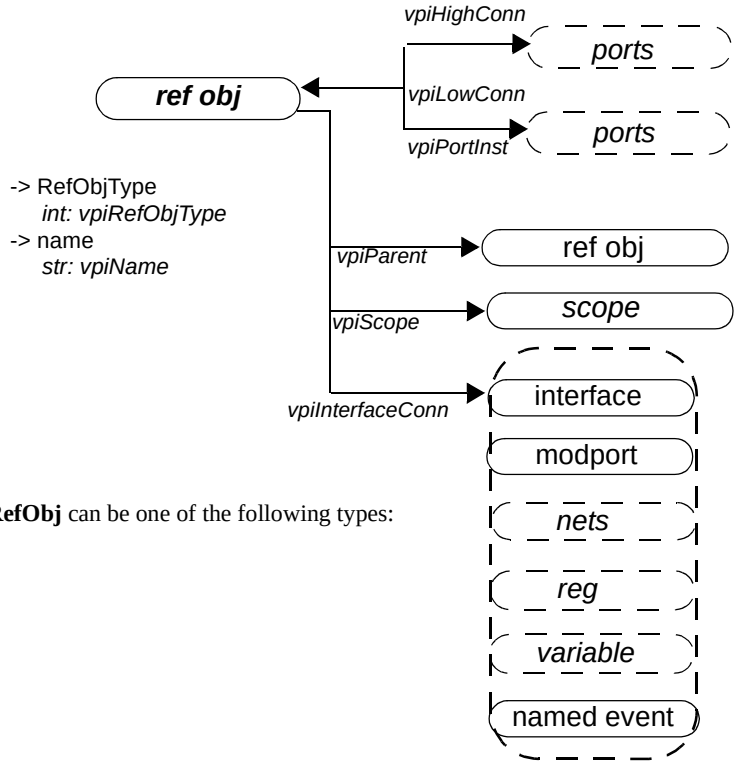


-> connected by name
    bool: vpiConnByName
-> delay (mipd)
    vpi_get_delays()
    vpi_put_delays()
-> direction
    int: vpiDirection
-> explicitly named
    bool: vpiExplicitName
-> index
    int: vpiPortIndex
-> name
    str: vpiName
-> port type
    int: vpiPortType
-> scalar
    bool: vpiScalar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

30.8 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

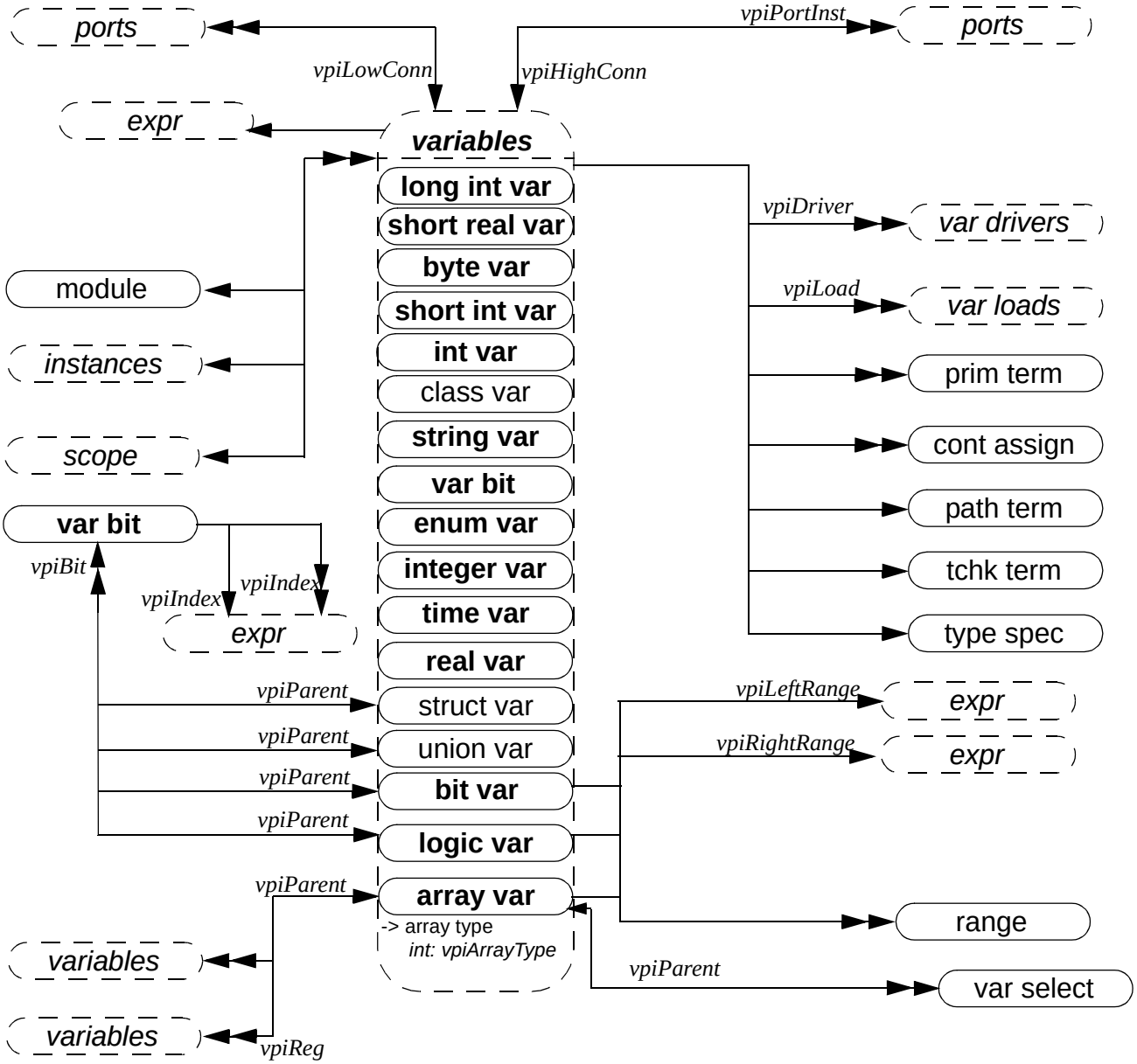
These objects are newly defined objects needed for supporting the full connectivity through ports where the ports are vpiInterface or vpiModport or any object inside modport or interface.

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.
```

30.9 Variable (supercedes IEEE 1364-2001 section 26.6.8)



-> array member
bool: vpiArray
-> name
str: vpiName
str: vpiFullName
-> sign
bool: vpiSigned
-> size
int: vpiSize
-> determine random availability
bool: vpiIsRandomized

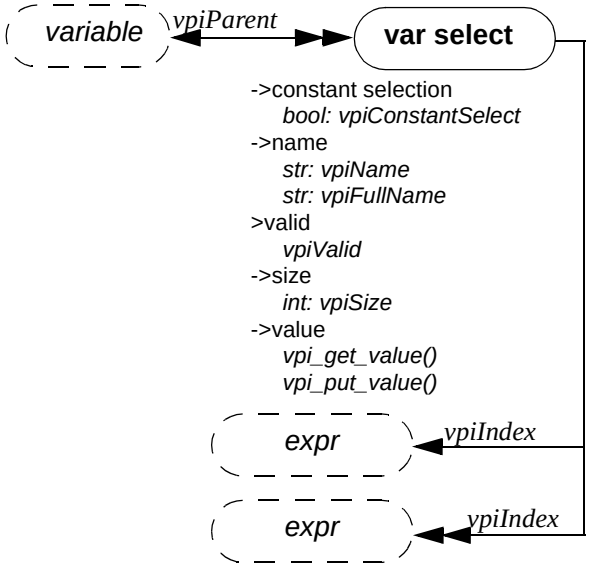
-> multi array
bool: vpiMultiArray
-> lifetime
bool: vpiAutomatic (ref. 26.6.20, 1364 2001)
-> constant variable
vpiConstantVariable
-> randomiztion type
int: vpiRandType
can be vpiRand, vpiRandC, vpiNotRand

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

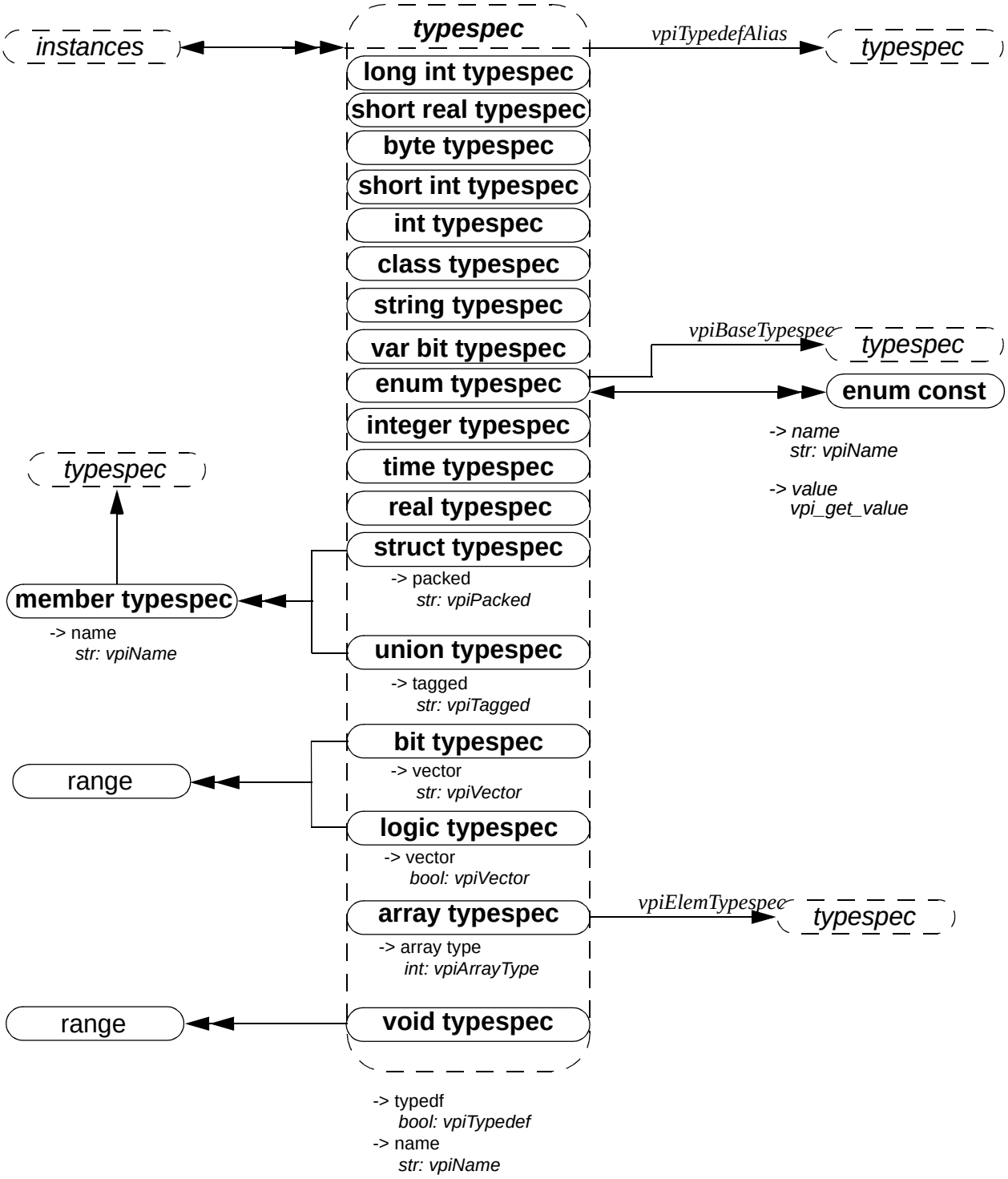
NOTES

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. Array var handle 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. vpiMember indicates that the variable is a member of a parent struct or union variable. See also relations in section **30.21**
18. Note that:
    logic var == reg
    var bit var == reg bit
    array var == reg array

30.10 Var Select (supercedes IEEE 1364-2001 26.6.8)



30.11 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;

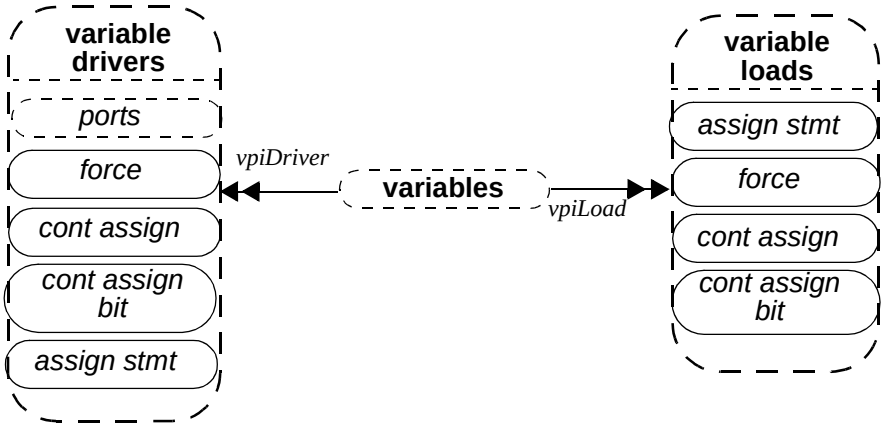
2. If the type of a type is **vpiStruct** or **vpiUnion**, then you can iterate over numbers to obtain the structure of the user-defined type. For each member the typespec relation from the member will detail its type.

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:

```
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**.

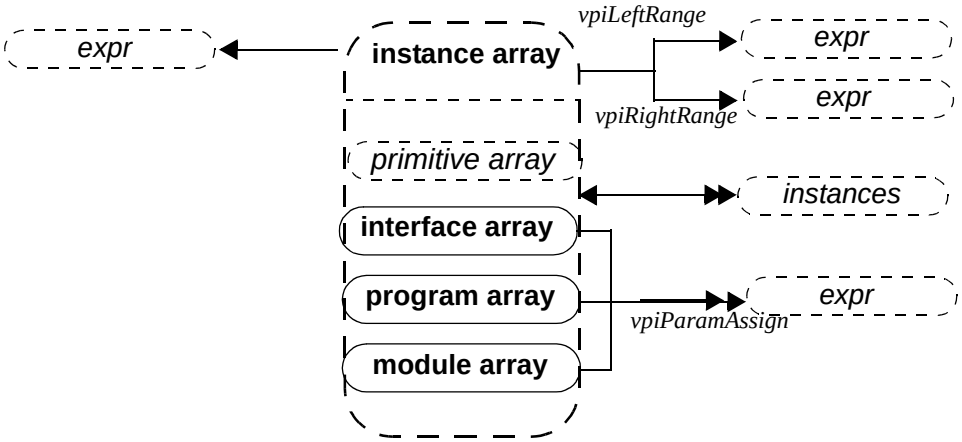
30.12 Variable Drivers and Loads (supercedes IEEE 1364-2001 26.6.23)



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.

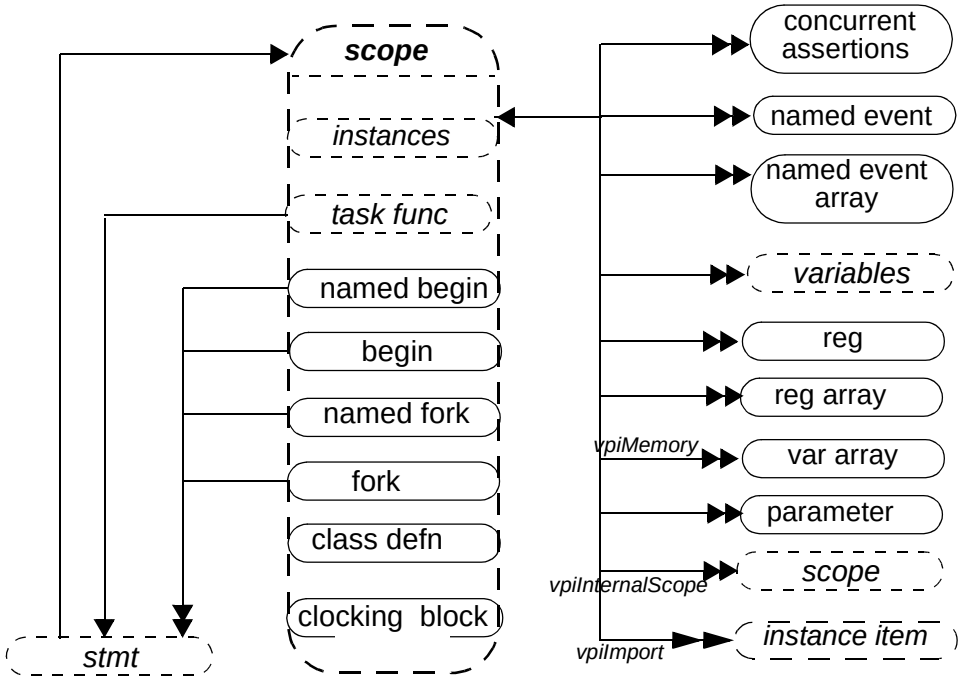
30.13 Instance Arrays (supercedes IEEE 1364-2001 26.6.2)



NOTE

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

30.14 Scope (supercedes IEEE 1364-2001 26.6.3)



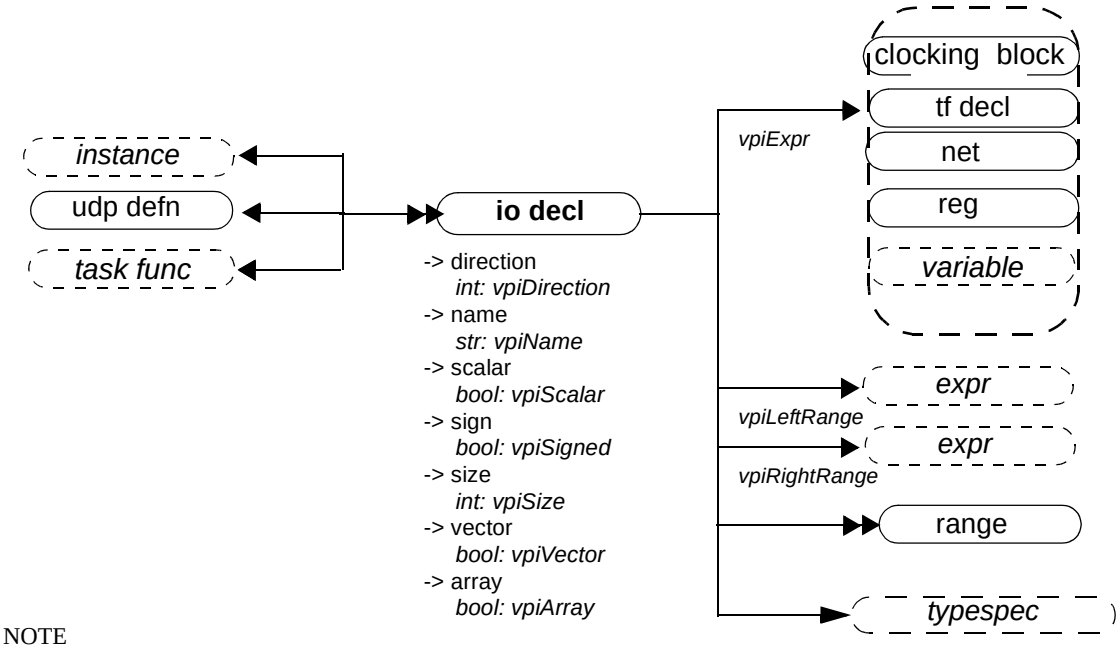
-> name
str: vpiName
str: vpiFullName

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.

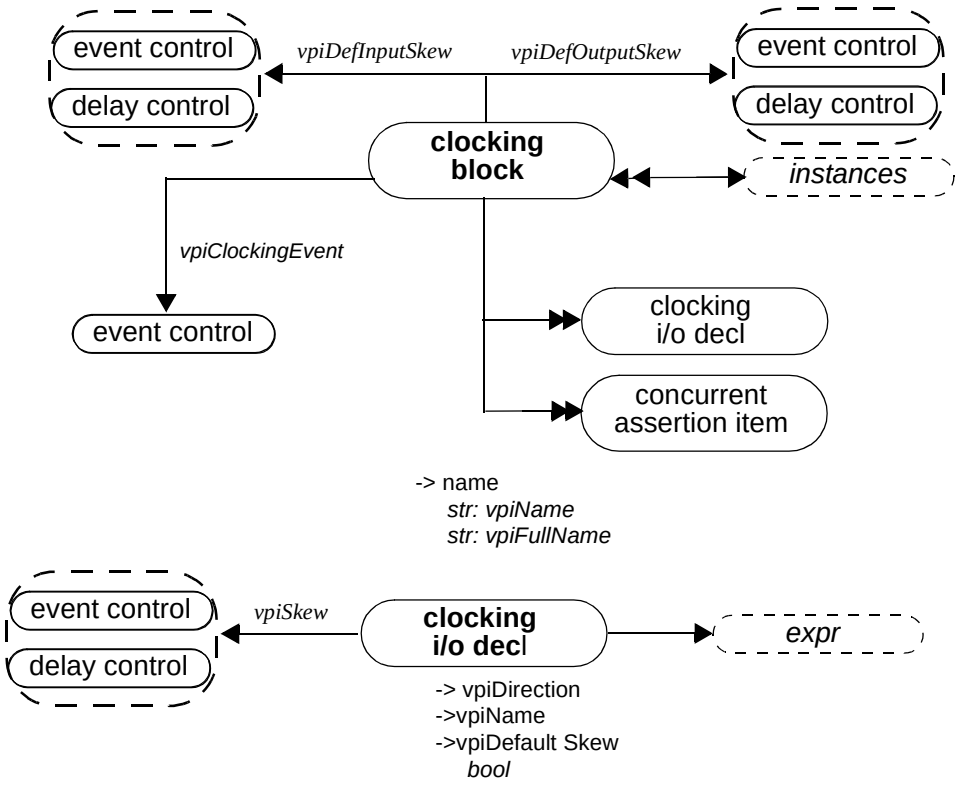
30.15 IO Declaration (supercedes IEEE 1364-2001 26.6.4)



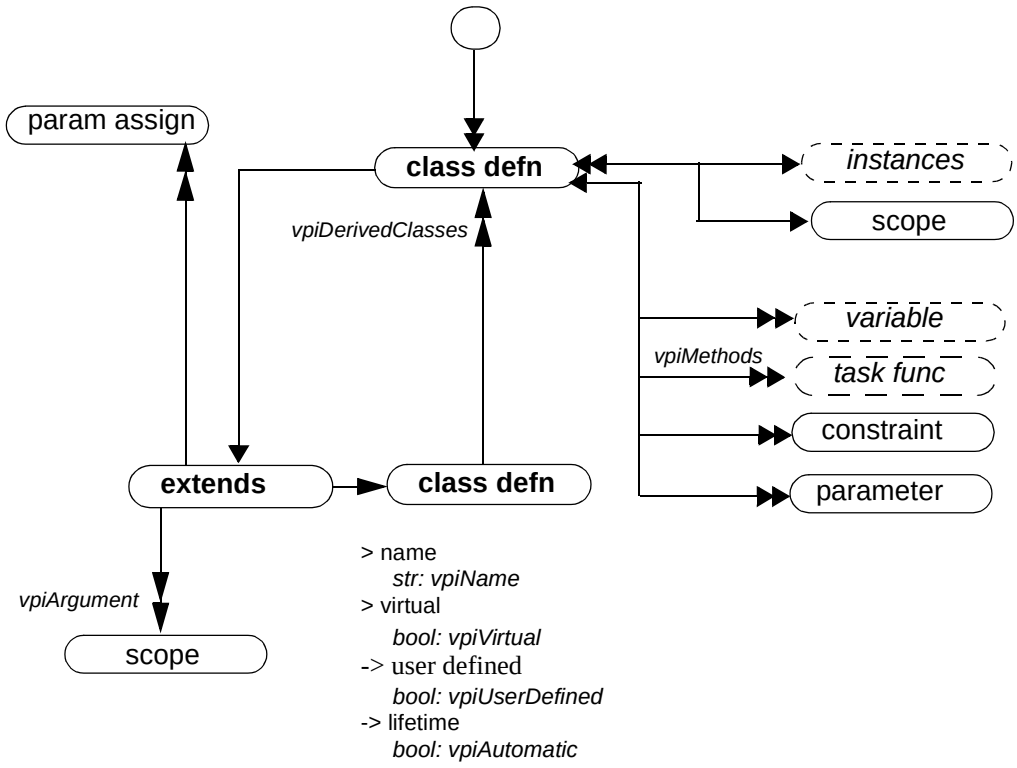
NOTE

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

30.16 clocking block



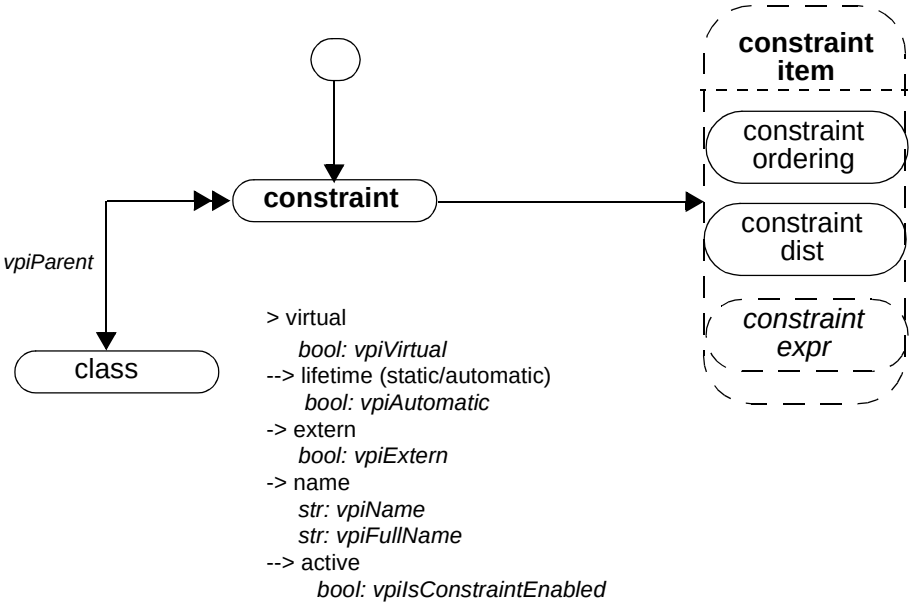
30.17 Class Object Definition

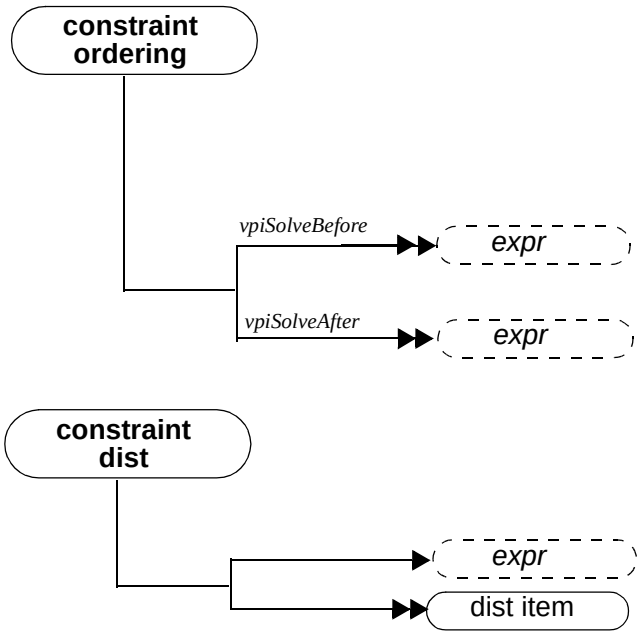


NOTE

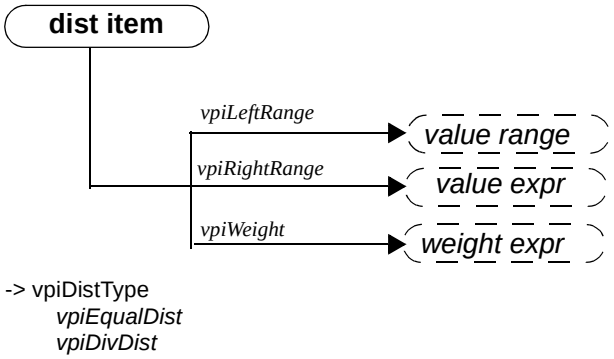
1. **ClassDefn** handle is a new concept. It does not correspond to any **vpiUserDefined** (class object) in the design. Rather it represents the actual type definition of a class.
2. Should not call **vpi_get_value**/**vpi_put_value** on the non-static variables obtained from the class definition handle.
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 extend relation to obtain the base class.
5. The vpiDerivedClasses iterator returns all the classes derived from the given class.

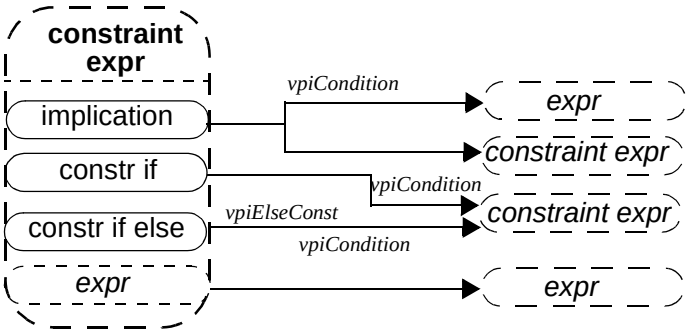
30.18 Constraint



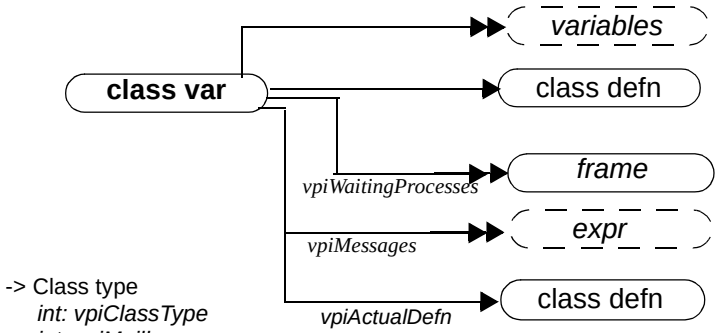


30.19 Dist Item





30.20 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

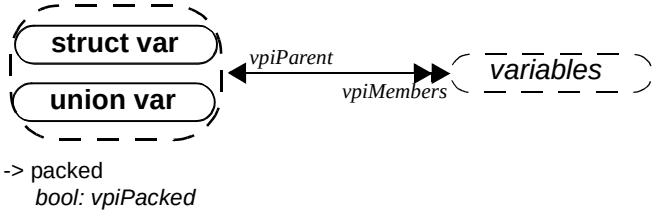
NOTES

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. **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.

30.21 Structure/Union

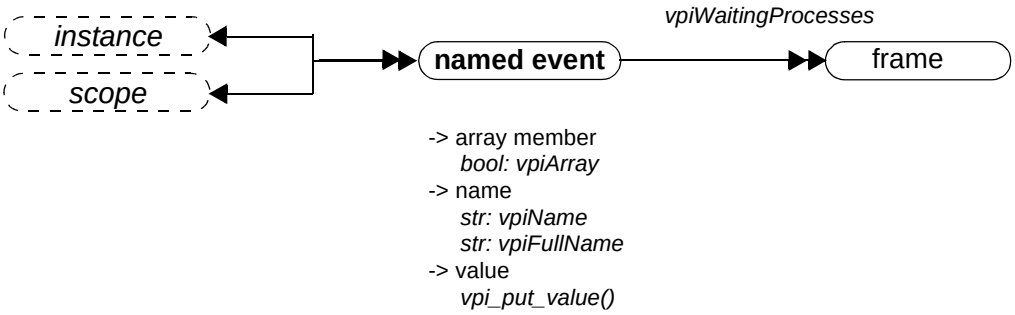


-> packed
bool: vpiPacked

NOTES

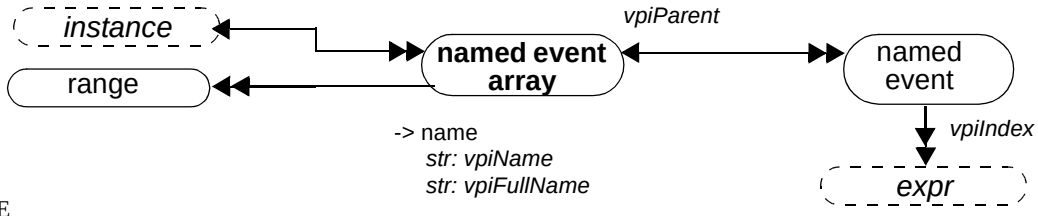
vpi_get_value/vpi_put_value cannot be used to access values of entire unpacked structures and unpacked unions.

30.22 Named Events (supercedes IEEE 1364-2001 26.6.11)



NOTE

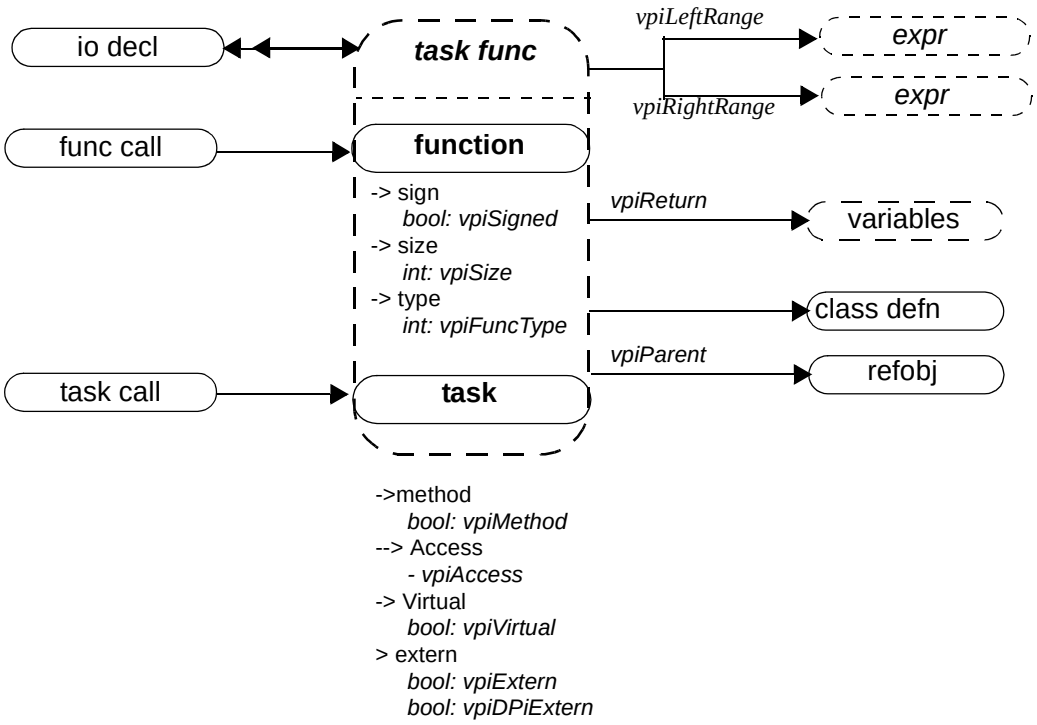
The new iterator (**vpiWaitingProcess**) returns all waiting processes, identified by their frame, for that namedEvent.



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.

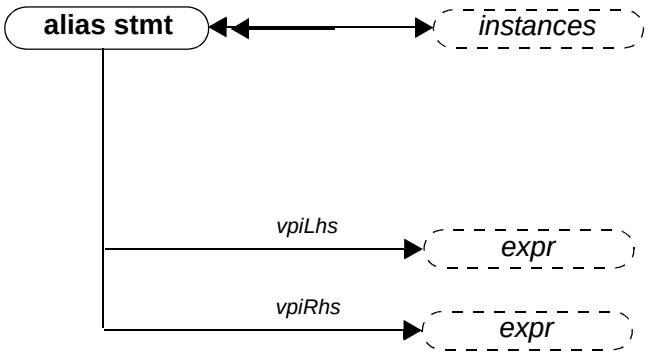
30.23 Task, Function Declaration (supercedes IEEE 1364-2001 26.6.18)



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.

30.24 Alias Statement



Examples

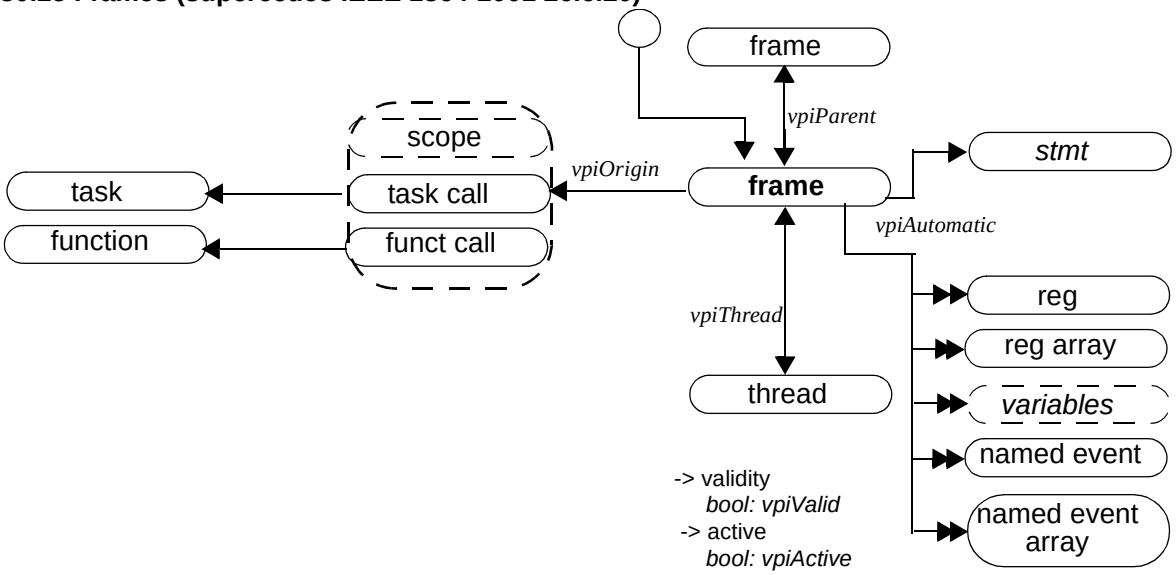
```
alias a=b=c=d
```

Results in 3 aliases:

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

d is Rhs for all.

30.25 Frames (supercedes IEEE 1364-2001 26.6.20)

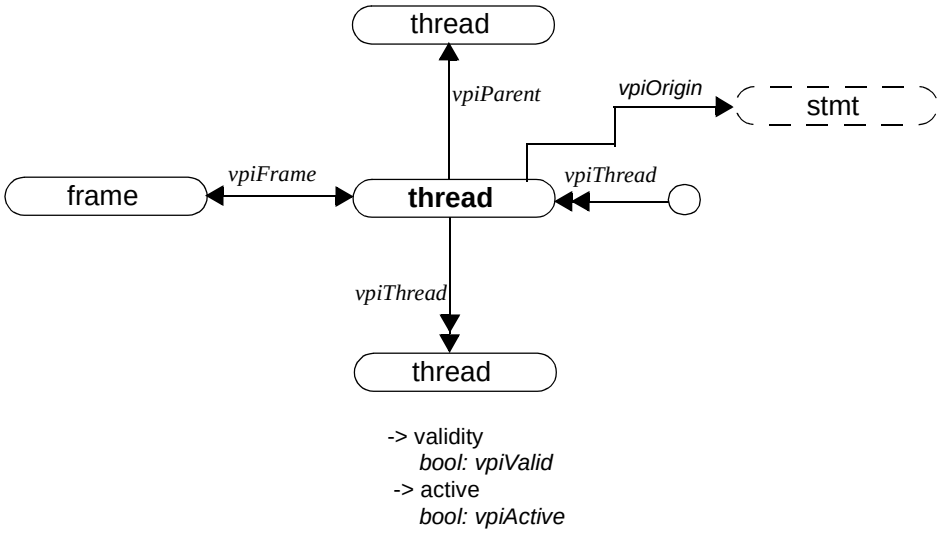


NOTES

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.

30.26 Threads

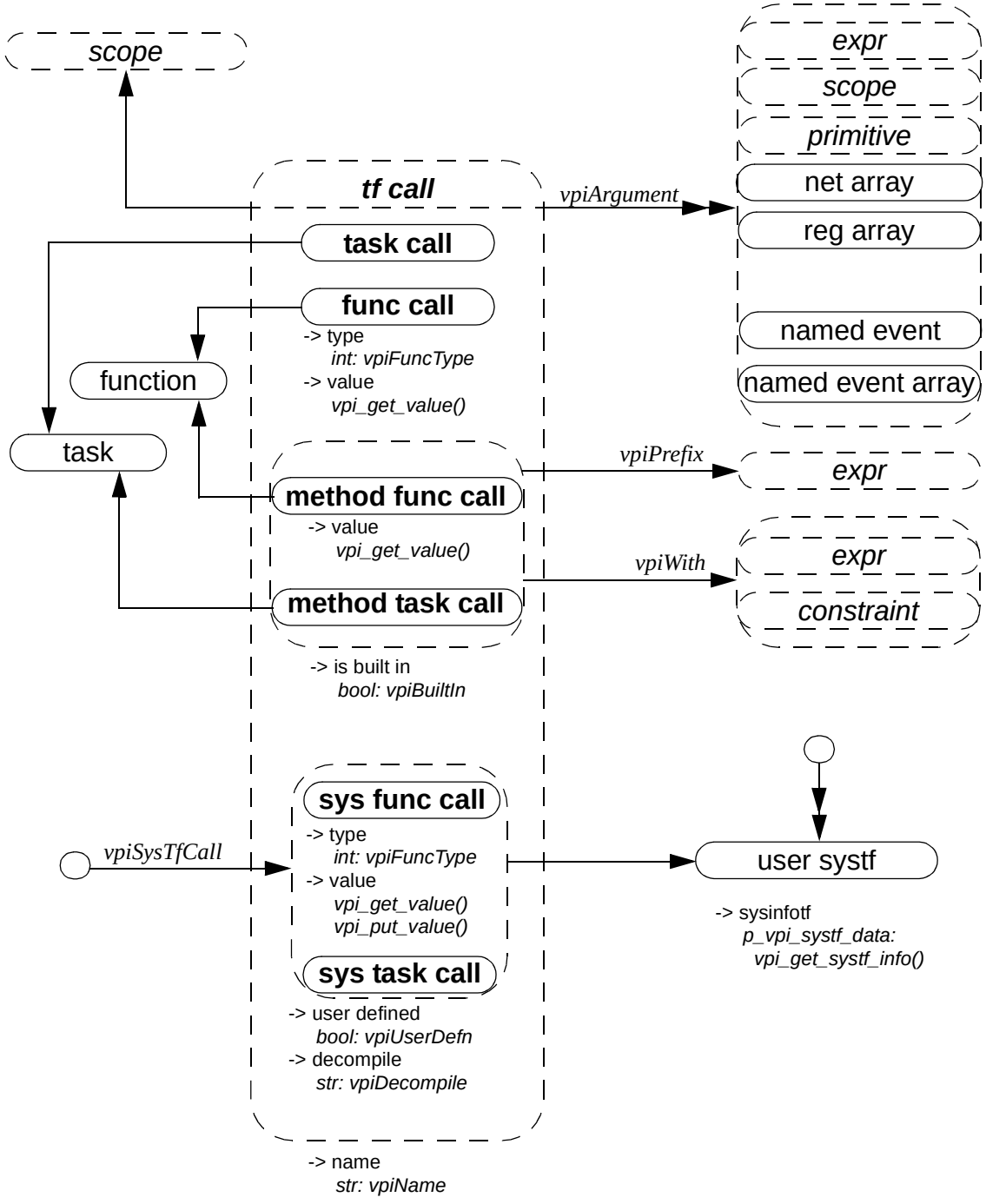


NOTES

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

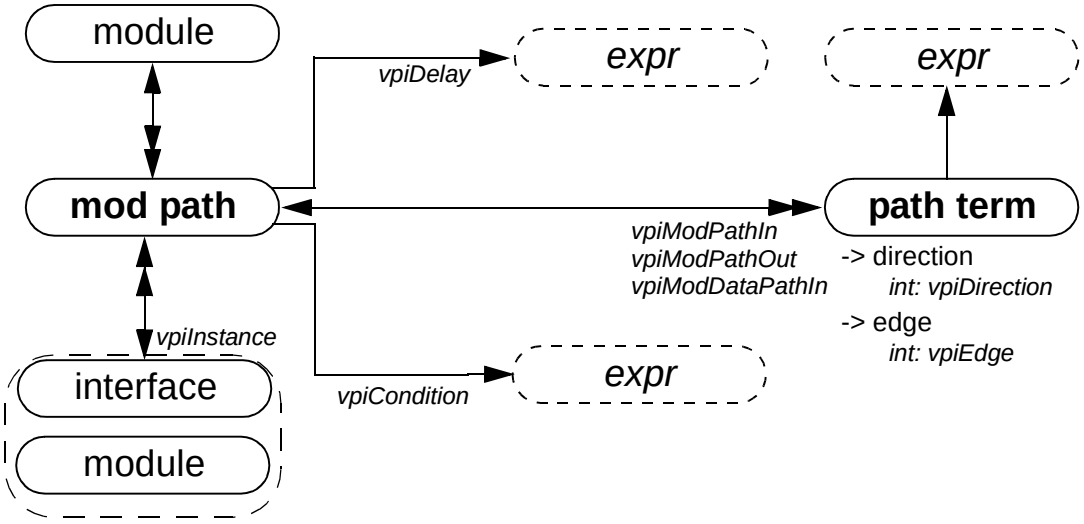
30.27 tf call (supercedes IEEE 1364-2001 26.6.19)



NOTE:

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

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



mod path properties:

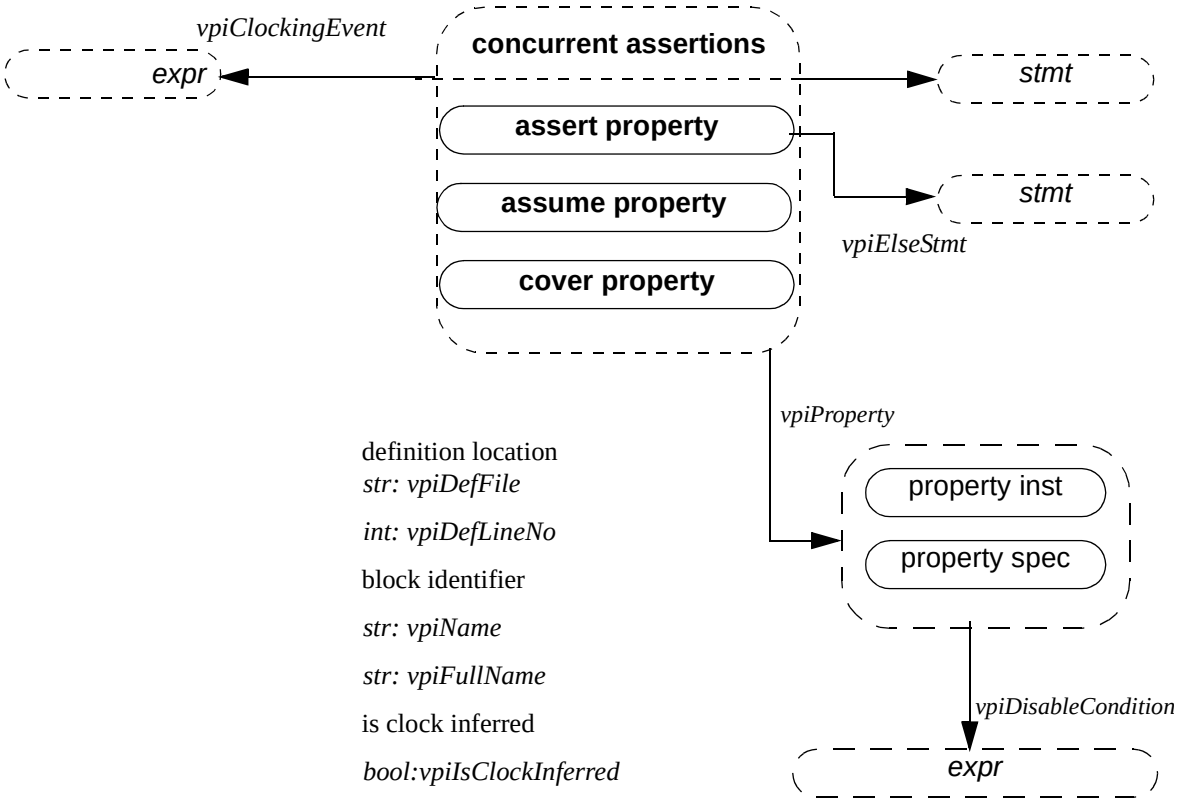
-> delay
vpi_get_delays()
vpi_put_delays()

-> path type
int: vpiPathType

-> polarity
int: vpiPolarity
int: vpiDataPolarity

-> hasIfNone

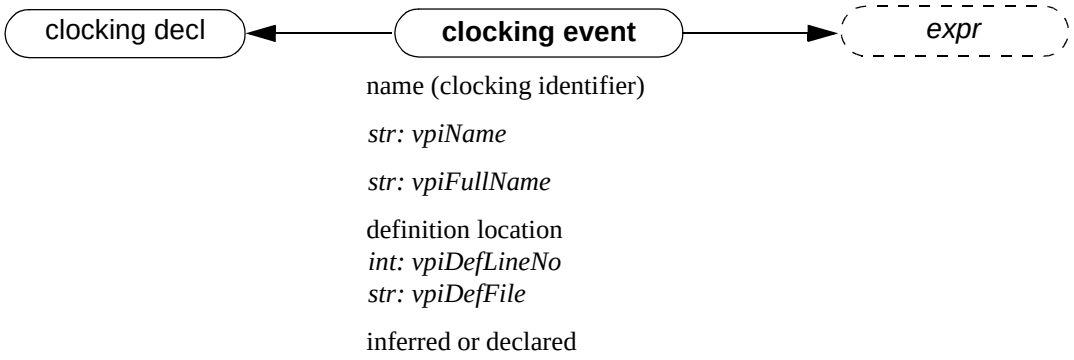
30.29 Concurrent Assertions

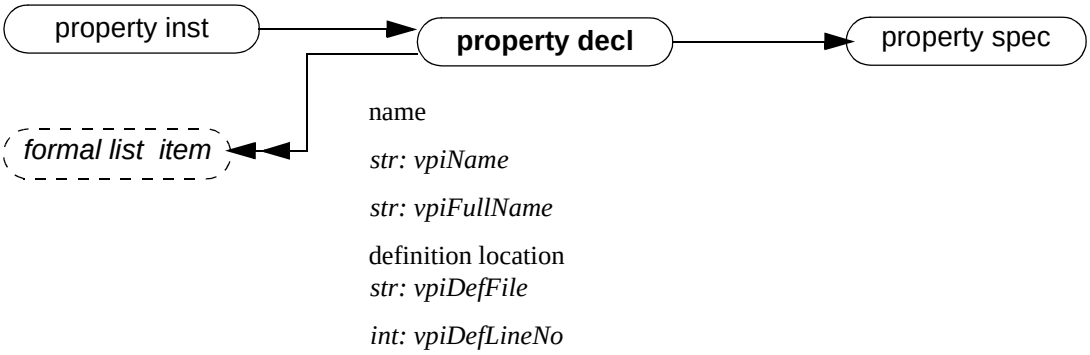


NOTE

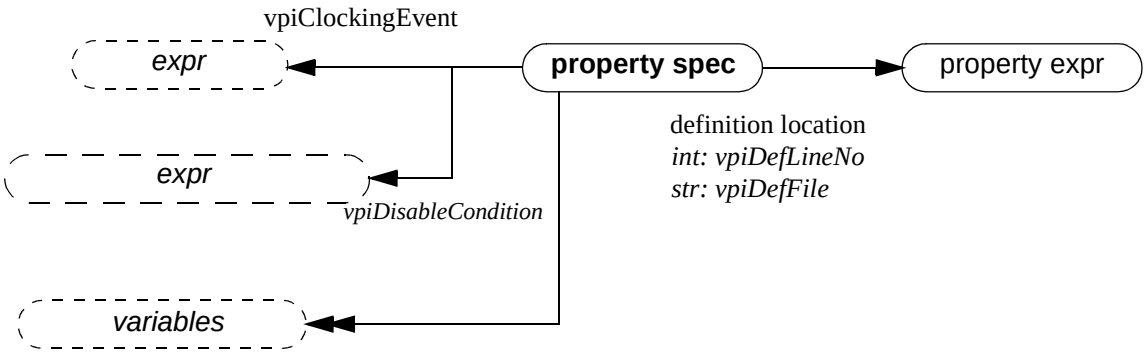
Clocking event is always the actual clocking event on which the assertion is being evaluated, regardless of whether this is explicit or implicit (inferred)

30.30 Clocking Event, Property Decl





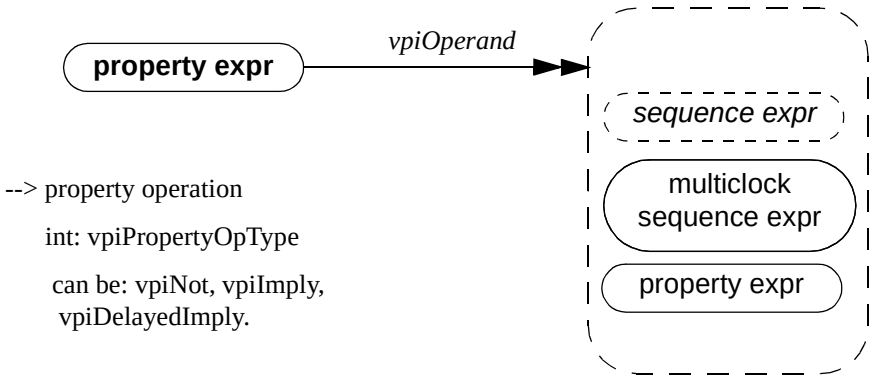
30.31 Property Specification



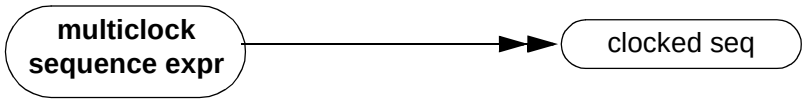
NOTE

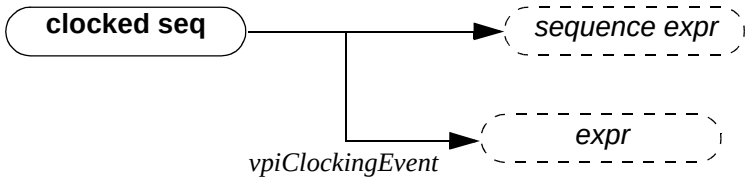
Variables are declarations of property variables. You cannot get the value of these variables.

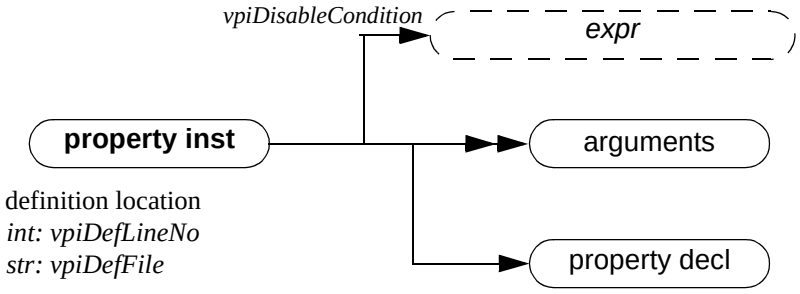
Note that the sequence bubble will be as already drawn in this diagram, but only one of them.



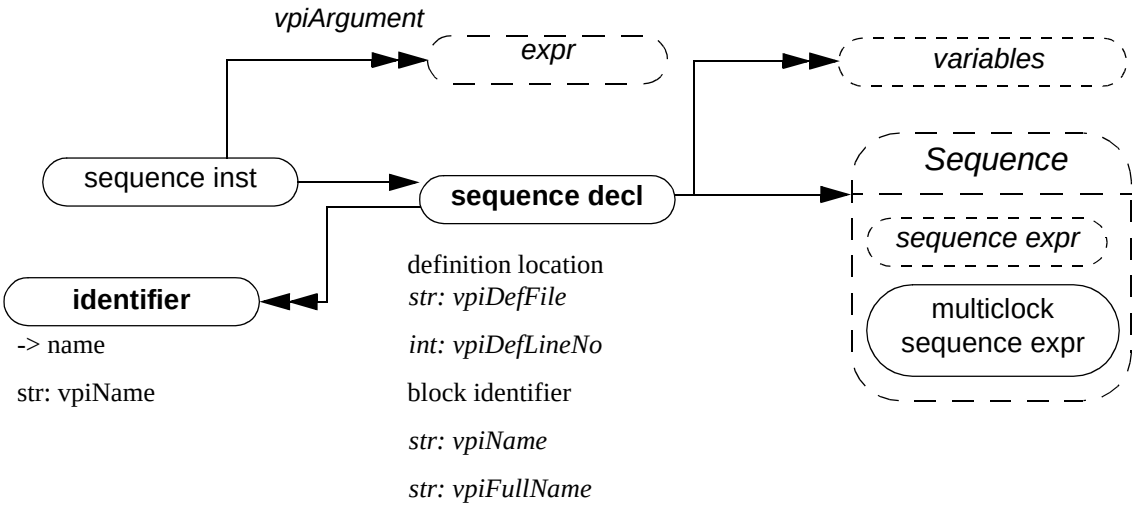
30.32 Multiclock Sequence Expression







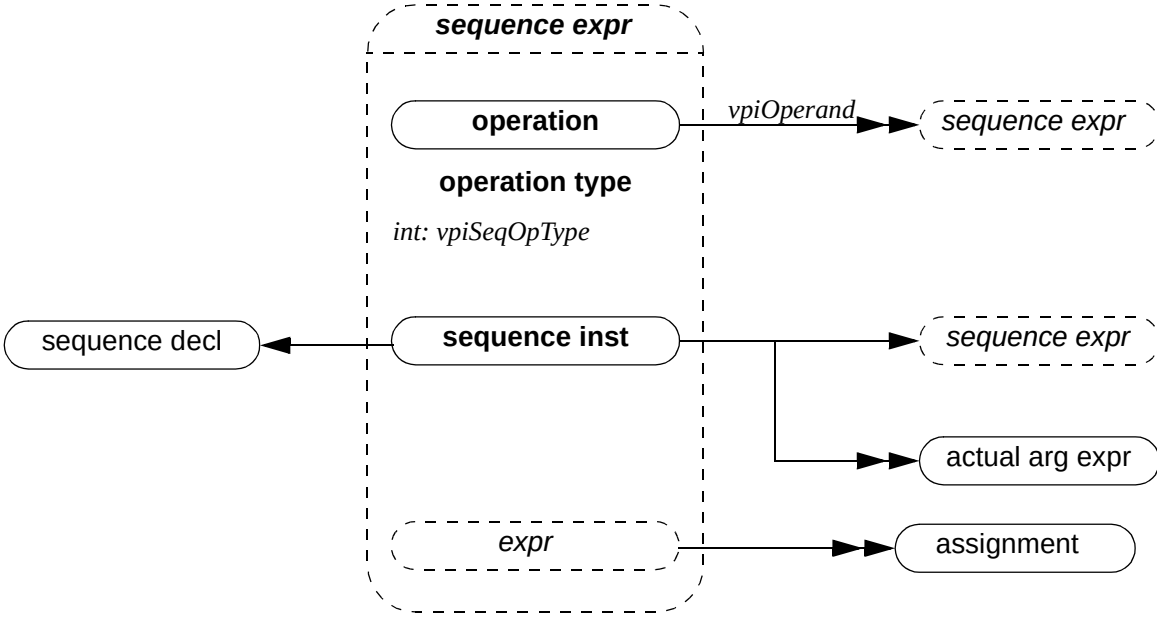
30.33 Sequence Declaration



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.

30.34 Sequence Expression



int: vpiSeqOpType is one of:

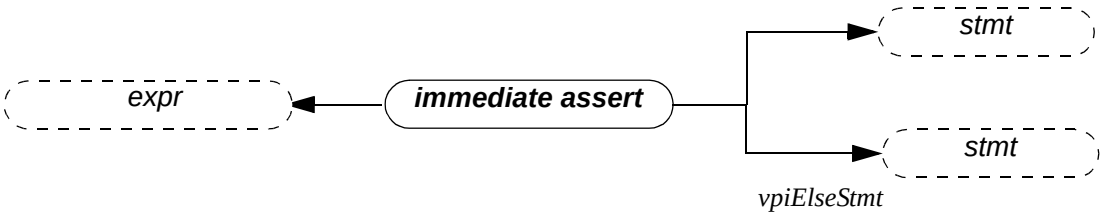
and, intersect, or,

first_match,

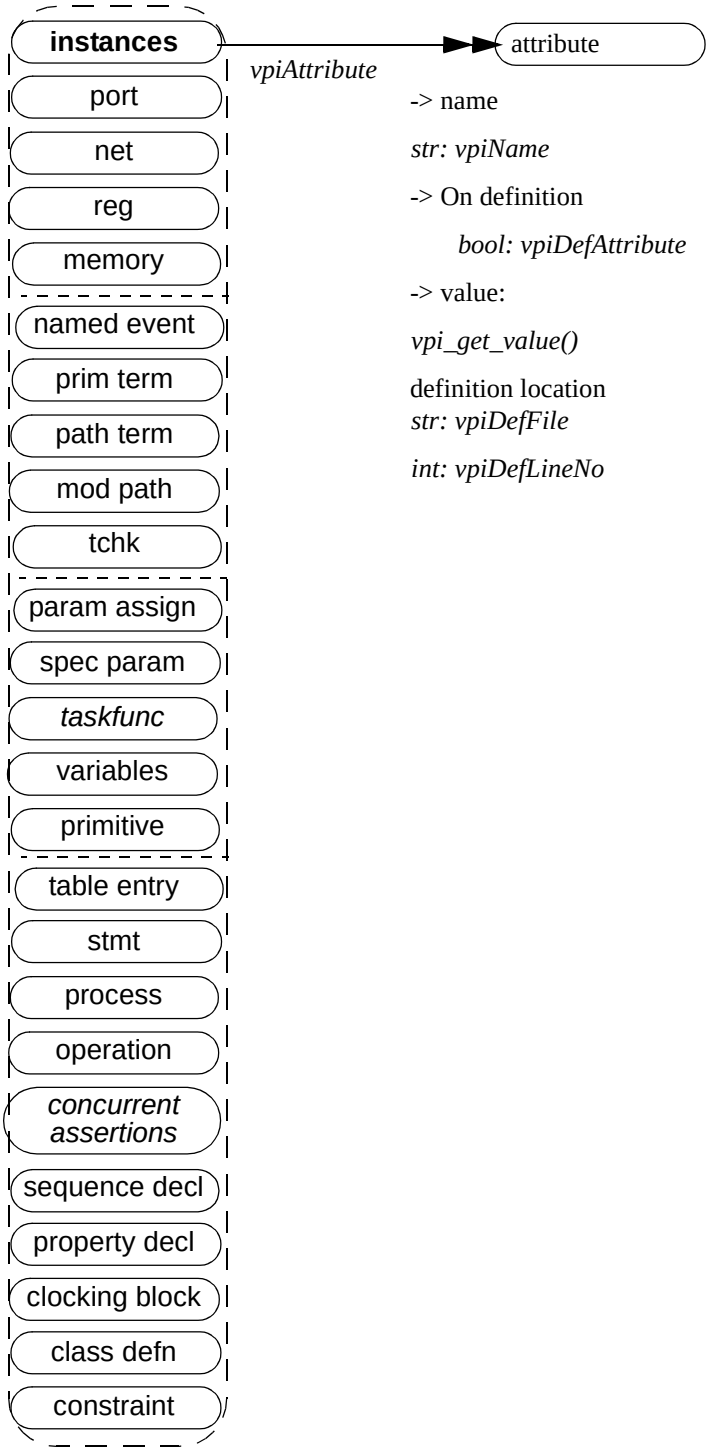
throughout, within,

##,

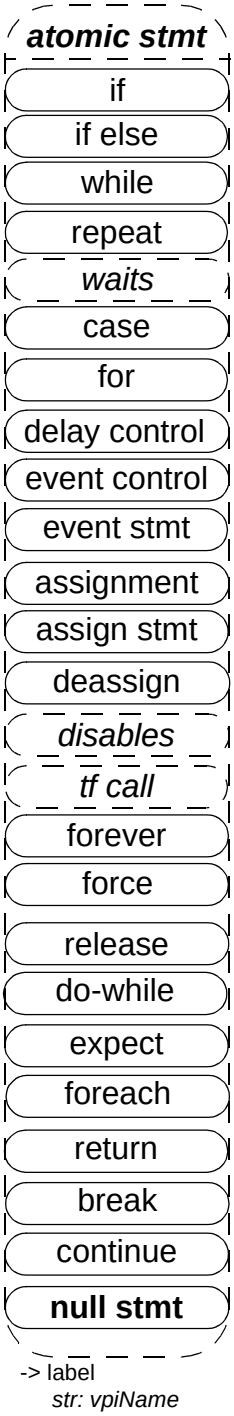
[*], [*=], [*->]



30.35 Instances

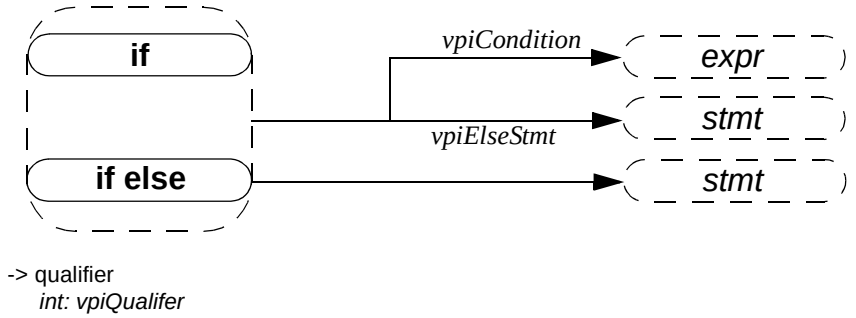


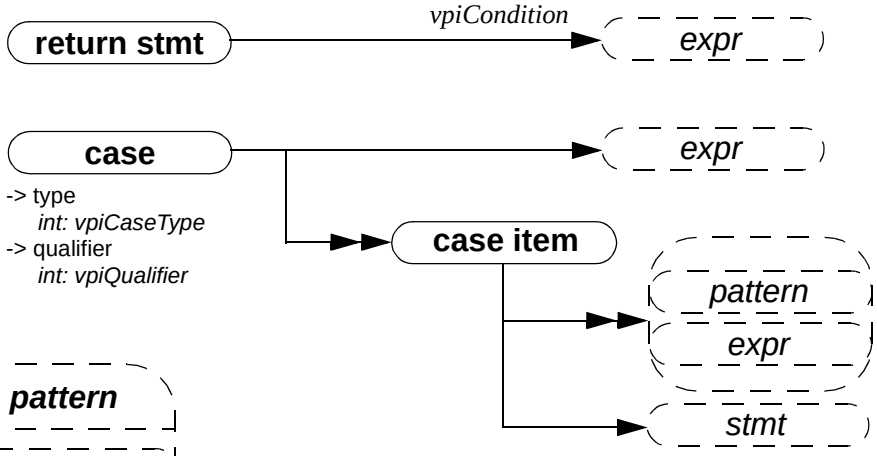
30.36 Atomic Statement

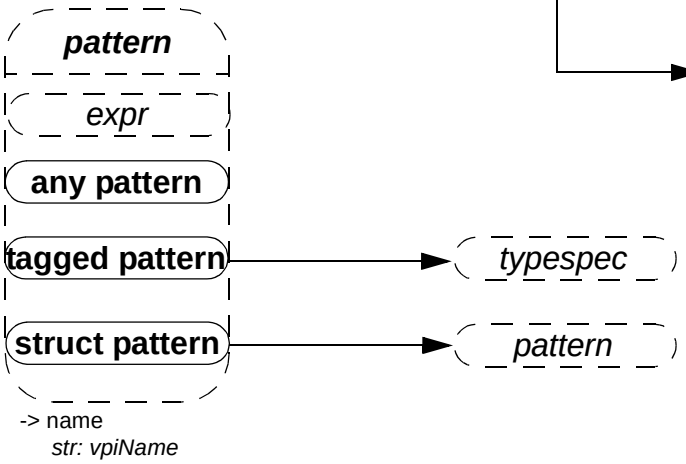


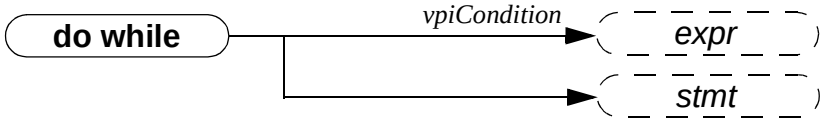
The vpiName property provides the statement label if one was given, otherwise the name is NULL.

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

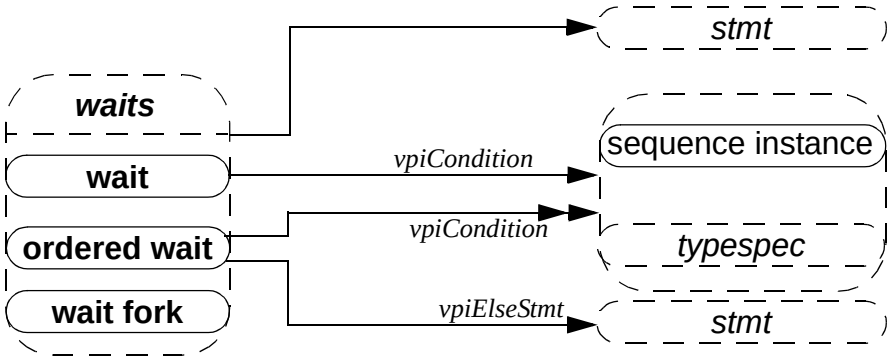


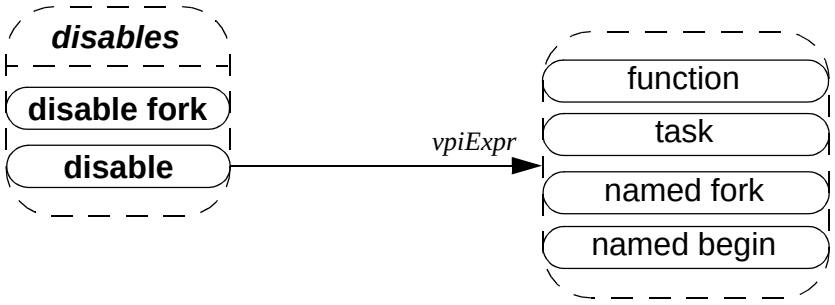


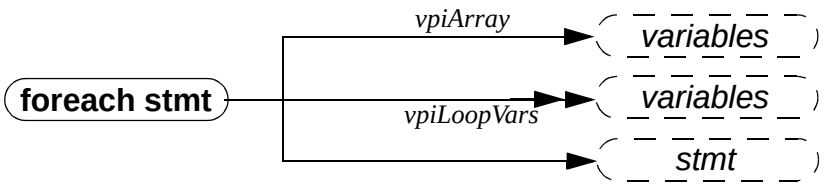




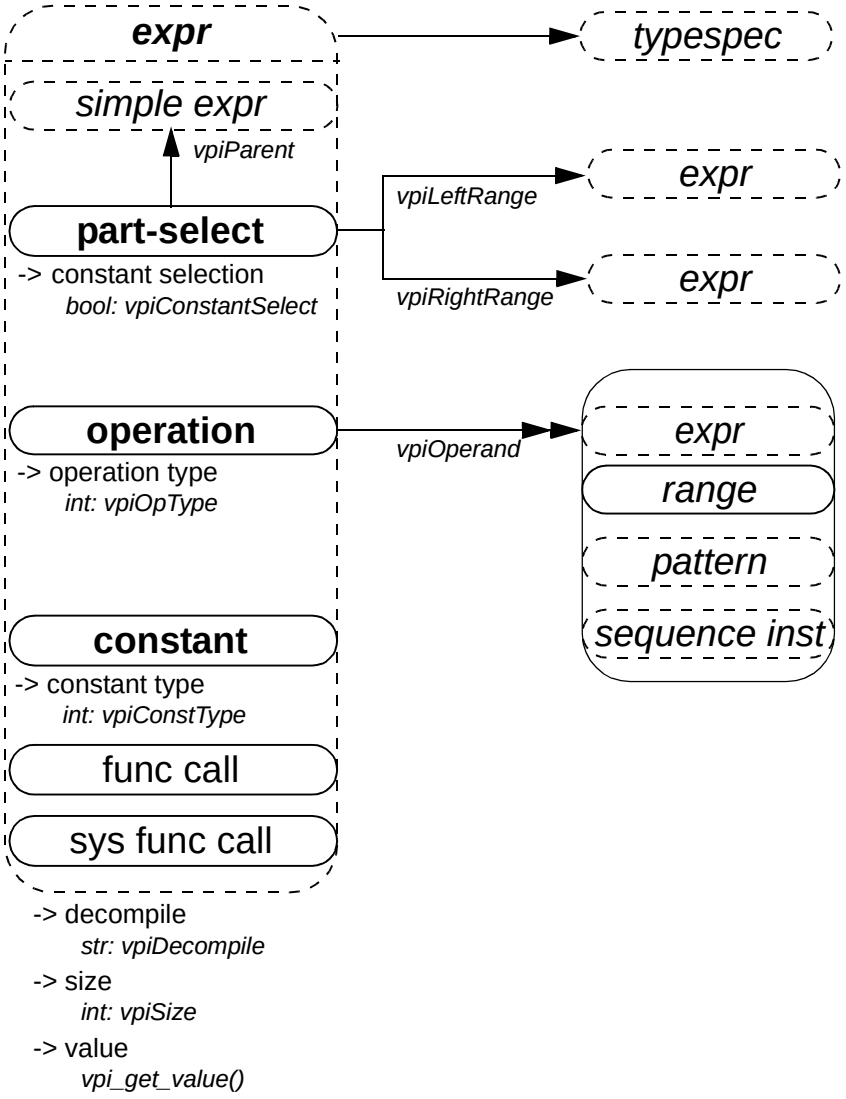
30.38 waits, disables, expect, foreach (supercedes IEEE 1364 26.6.38)







30.39 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.

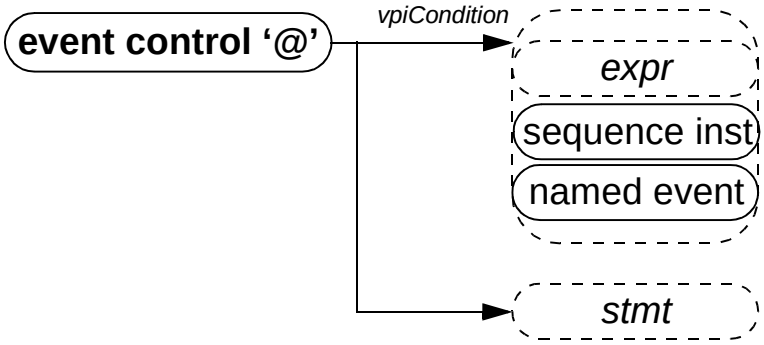
2) The property **vpiDecompile** will return a string with a functionally equivalent expression to the original expression within the HDL. Parenthesis shall be added only to preserve precedence. Each operand and operator shall be separated by a single space character. No additional white space shall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

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.

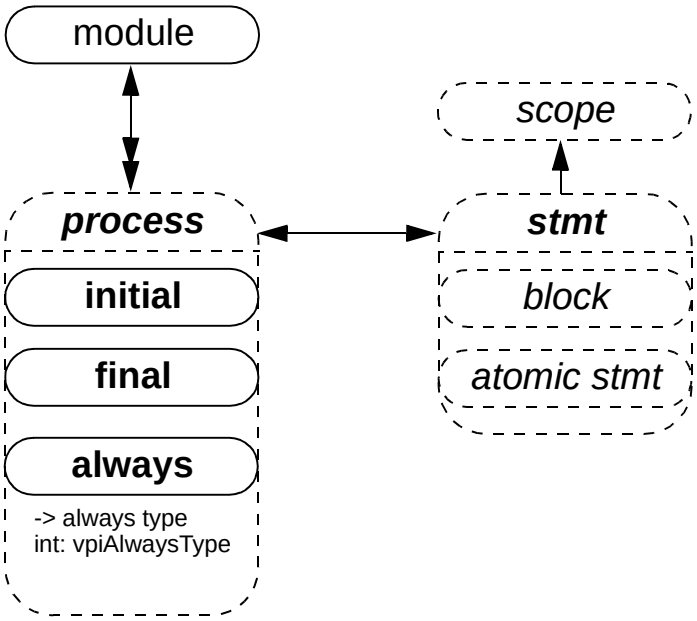
30.40 Event control (supercedes IEEE 1364-2001 26.6.30)



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

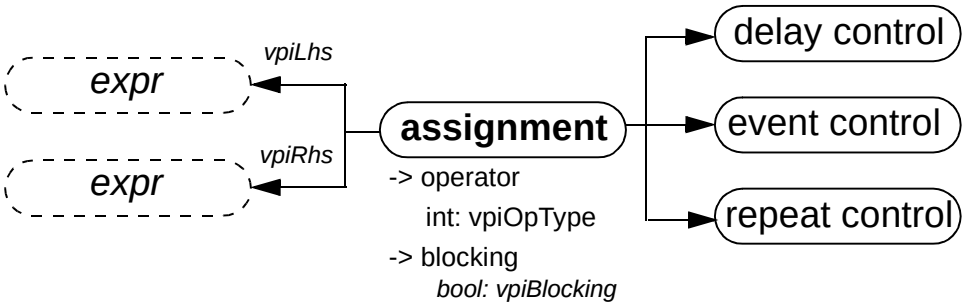
30.41 Event stmt (supercedes IEEE 1364-2001 26.6.27)

30.42 Process (supercedes IEEE 1364-2001 26.6.27)



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

30.43 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.

Annex K: sv_vpi_user.h

(normative)

sv_vpi_user.h

```
/*****************************************************************************
 * sv_vpi_user.h
 *
 * Accellera SystemVerilog VPI extensions.
 *
 * This file contains the constant definitions, structure definitions, and
 * routine declarations used by the Verilog PLI procedural interface VPI
 * access routines.
 *
 *****************************************************************************/

#ifndef SV_VPI_USER_H
#define SV_VPI_USER_h

#include <vpi_user.h>

#ifdef __cplusplus
extern "C" {
#endif

/******************************** OBJECT TYPES *******************************/
#define vpiPackage
#define vpiInterface
#define vpiProgram
#define vpiInterfaceArray
#define vpiProgramArray
#define vpiTypespec
#define vpiModport
#define vpiInterfaceTfDecl
#define vpiRefObj
#define vpiVarBitVar        vpiRegBit
#define vpiLongIntVar
#define vpiShortIntVar
#define vpiIntVar
#define vpiShortRealVar
#define vpiByteVar
#define vpiClassVar
#define vpiStringVar
#define vpiEnumVar
#define vpiStructVar
#define vpiUnionVar
#define vpiBitVar
#define vpiLogicVar         vpiRegVar
#define vpiArrayVar         vpiRegArray
#define vpiLongIntTypespec
#define vpiShortRealTypespec
#define vpiByteTypespec
#define vpiShortIntTypespec
#define vpiIntTypespec
```

```
#define vpiClassTypespec
#define vpiStringTypespec
#define vpiVarBitTypespec
#define vpiEnumTypespec
#define vpiEnumConst
#define vpiIntegerTypespec
#define vpiTimeTypespec
#define vpiRealTypespec
#define vpiStructTypespec
#define vpiUnionTypespec
#define vpiBitTypespec
#define vpiLogicTypespec
#define vpiArrayTypespec
#define vpiVoidTypespec
#define vpiMemberTypespec
#define vpiClockingBlock
#define vpiClockingIODecl
#define vpiClassDefn
#define vpiConstraint
#define vpiConstraintOrdering
#define vpiConstraintDist
#define vpiDistItem
#define vpiAliasStmt
#define vpiThread
#define vpiMethodFuncCall
#define vpiMethodTaskCall
#define vpiAssertProperty
#define vpiAssumeProperty
#define vpiCoverProperty
#define vpiDisableCondition
#define vpiClockingEvent
#define vpiPropertyDecl
#define vpiPropertySpec
#define vpiPropertyExpr
#define vpiMulticlockSequenceExpr
#define vpiClockedSeq
#define vpiPropertyInst
#define vpiSequenceDecl
#define vpiSequenceSpec
#define vpiActualArgExpr
#define vpiSequenceInst
#define vpiImmediateAssert
#define vpiReturn
#define vpiAnyPattern
#define vpiTaggedPattern
#define vpiStructPattern
#define vpiDoWhile
#define vpiOrderedWait
#define vpiWaitFork
#define vpiDisableFork
#define vpiExpectStmt
#define vpiForeachStmt
#define vpiFinal
```

```
/********************************* METHODS *********************************/
/************* methods used to traverse 1 to 1 relationships **************/
#define vpiInterfaceConn
#define vpiTypedefAlias
#define vpiBaseTypespec
#define vpiElemTypespec
#define vpiDefInputSkew
#define vpiDefOutputSkew
#define vpiSkew
#define vpiBaseClass
#define vpiActualDefn
#define vpiLhs
#define vpiRhs
#define vpiOrigin
#define vpiPrefix
#define vpiWith
#define vpiSuccessStmt
#define vpiFailStmt
#define vpiProperty


/************* methods used to traverse 1 to many relationships *************/
#define vpiTypedef
#define vpiDefaultClocking
#define vpiInstance
#define vpiImport
#define vpiDerivedClasses
#define vpiMethods
#define vpiSolveBefore
#define vpiSolveAfter
#define vpiWeight
#define vpiWaitingProcesses
#define vpiMessages
#define vpiMembers
#define vpiLoopVars


/************************* generic object properties ************************/
#define vpiTop
#define vpiUnit
#define vpiAccessType
#define vpiForkJoin
#define vpiExtern
#define vpiDPIExtern
#define vpiDPIImport
#define vpiArrayType
#define vpiDynamicArray
#define vpiQueueArray
#define vpiStaticArray
#define vpiIsRandomized
#define vpiRandType
```

```
#define vpiVpiRand
#define vpiRandC
#define vpiNotRand
#define vpiConstantVar
#define vpiMember
#define vpiVisibility
#define vpiPublic
#define vpiProtected
#define vpiPrivate
#define vpiPacked
#define vpiTagged
#define vpiRef
#define vpiDefaultSkew
#define vpiVirtual
#define vpiUserDefined
#define vpiIsConstraintEnabled
#define vpiClassType
#define vpiMailbox
#define vpiSemaphore
#define vpiAssociativeArray
#define vpiIndexTypespec
#define vpiMethod
#define vpiValid
#define vpiActive
#define vpiIsClockInferred
#define vpiUniqueQualifier
#define vpiPriorityQualifier
#define vpiTaggedQualifier
#define vpiNullConst
#define vpiOneStepConst
#define VpiAlwaysType
#define vpiAlwaysComb
#define vpiAlwaysFF
#define vpiAlwaysLatch



/********************************* Operators *********************************/
#define vpiEqualDist                /* constraint equal distribution operator */
#define vpiDivDist                  /* constraint divided distribution operator */

#define vpiImplyOp                  /* -> implication operator */
#define vpiNonOverlapImplyOp            /* |=> non-overlapped implication */
#define vpiOverlapImplyOp             /* |-> overlapped implication operator */
#define vpiCycleDelayOp              /* cycle delay (##) operator */
#define vpiIntersectOp              /* intersection operator */
#define vpiFirstMatchOp               /* first_match operator */
#define vpiThroughoutOp              /* throught operator */
#define vpiWithinOp                 /* within operator */
#define vpiRepeatOp                 /* [*=] non-consecutive repetition */
#define vpiConsecutiveRepeatOp            /* [*] consecutive repetition */
#define vpiGotoRepeatOp              /* [*->] goto repetition */

#define vpiPostIncOp                /* ++ post-increment */
```

```
#define vpiPreIncOp                 /* ++ pre-increment */
#define vpiPostDecOp                /* -- post-decrement */
#define vpiPreDecOp                 /* -- pre-decrement */

#define vpiMatchOp                  /* match() operator */
#define vpiCastOp                   /* type‘() operator */
#define vpiIffOp                    /* iff operator */
#define vpiWildEqOp                 /* =?= operator */
#define vpiWildNeqOp                /* !?= operator */


/************************* STRUCTURE DEFINITIONS *************************/

/*****************************  structure *****************************/

/***************************** CALLBACK REASONS *****************************/
#define cbStartOfThread             /* callback on thread creation */
#define cbEndOfThread               /* callback on thread termination */
#define cbEnterThread               /* callback on re-entering thread */
#define cbStartOfFrame              /* callback on frame creation */
#define cbEndOfFrame                /* callback on frame exit */
#define cbTypeChange                /* callback on variable type/size change */

/************************* FUNCTION DECLARATIONS *************************/

#ifdef __cplusplus
}
#endif

#endif
```