

32. SystemVerilog VPI Object Model

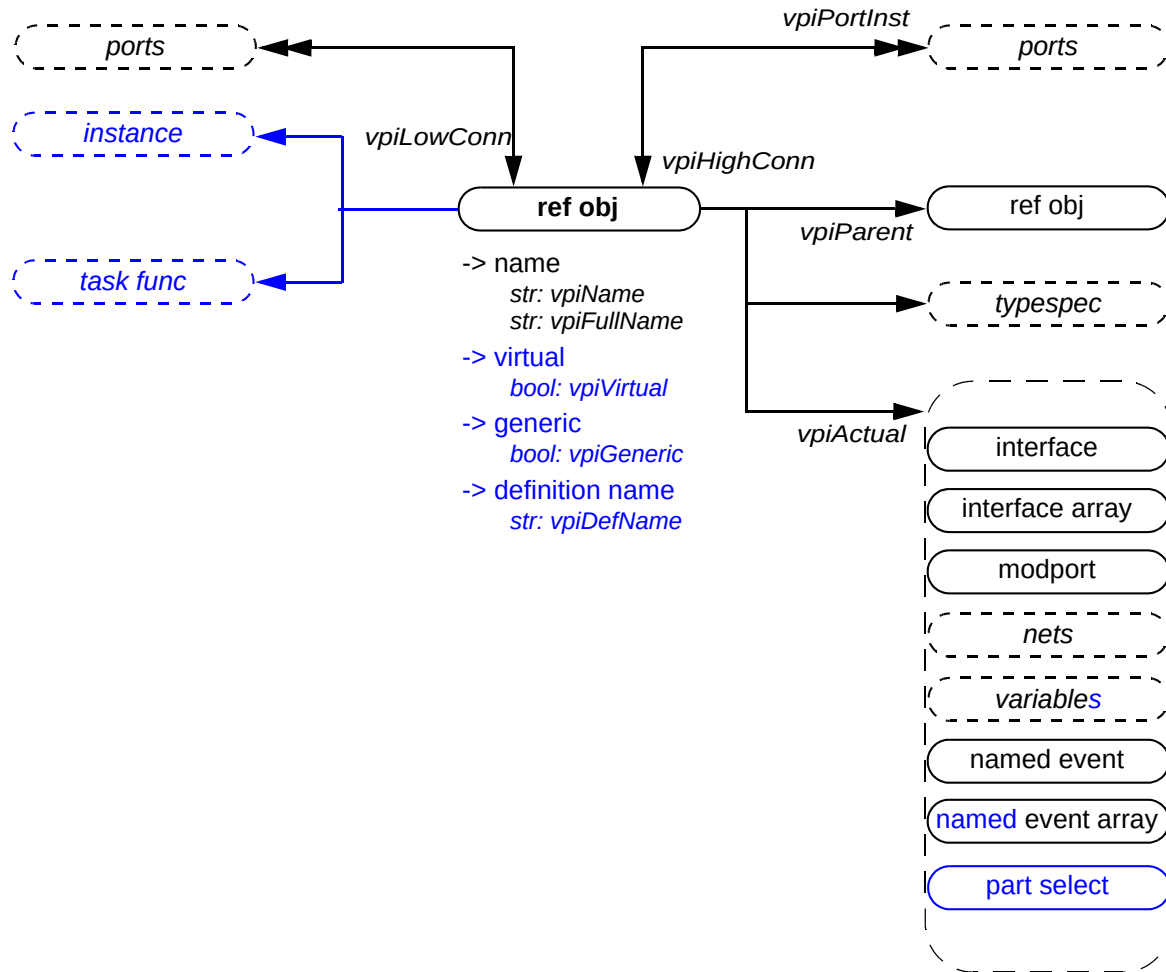
Note this proposal contains fixes for errata 489:

It adds a note for describing what the ref obj is.

I fixed the first example and provides a second example for illustrating virtual interface ref obj.

Diagram 32.12 ref obj has the vpiVirtual property which only applies to ref obj of a virtual interface declaration.

32.12 Reference objects



Details:

1. A **ref obj** represents a declared object or sub-element of that object that is a reference to an actual instantiated object. A **ref obj** exists for ports with ref direction, for an interface port, a modport port, or for formal task function ref arguments. The specific cases for a ref obj are:
 - a variable, named event, named event array that is the lowconn of a ref port,
 - any sub-element expression of the above,
 - a local declaration of an interface or modport passed through a port, or any net, variable, named event, named event array of those,

- a virtual interface declaration in a class definition,
- a ref formal argument of a task or function, or sub-element expression of it.

A **ref obj** may be obtained when walking port connections (lowConn, highConn), when traversing an expression which is a use of such **ref obj**, when accessing the virtual interface of a class, or when accessing the **io decl** of an instance or task or function.

2. The name of **ref obj** can be different at every instance level it is being declared. The **vpiActual** relationship always returns the actual instantiated object if the **ref obj** is bound to an actual object at the time of the query.
 3. The **vpiParent** relationship allows the traversal of a **ref obj** which is a sub-element of a **ref obj**. In the example below, r[0] is a **ref obj** which parent is the **ref obj** r. The **vpiActual** for the **ref obj** 'r[0]' would return the **var bit** 'a[0]' and the **vpiActual** of the **ref obj** 'r' would return the variable 'a'.
- ```

module top;
 logic [2:0] a;
 u1 m (a);
endmodule

module n (ref [2:0] r);
 initial
 r[0] = 1'b0;
endmodule

```
4. The **vpiVirtual** property shall return TRUE if the **ref obj** is a reference to a virtual interface, FALSE if the **ref obj** is a reference to an interface that is not a virtual interface. The **vpiVirtual** property shall return **vpiUndefined** for all other kinds of **ref obj**.
  5. The **vpiGeneric** property shall return TRUE if the **ref obj** is a reference to a generic interface, FALSE if the **ref obj** is a reference to an interface that is not a generic interface. The **vpiGeneric** property shall return **vpiUndefined** for all other kinds of **ref obj**.
  6. The **vpiDefName** property when applied to a refobj which is an actual of an interface or modport shall return the interface definition name or modport name.
  7. The **vpiTypeSpec** property returns NULL for a **ref obj** which **vpiActual** is a not a net, variable or part select.

### 32.12.1 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.~~

Example 1: Passing an interface or modport through a port.

```

interface simple ();
 logic req, gnt;
 modport slave (input req, output gnt);
 modport master (input gnt, output req);
endinterface

```

```

module top();

 interface simple i;

 child1 i1(i);
 child2 i2(i.master);
endmodule

```

```

/*****
for the port of i1,

```

the vpiHighConn relationship= returns a handle of type vpiRefObj. The ~~where~~ vpiRefObjTypeActual = relationship applied to the ref obj returns a handle of type vpiInterface.  
 for the port of i2 ,  
 the vpiHighConn relationship= returns a handle of type vpiRefObj. The ~~where~~ vpiFullTypeActual =relationship applied to the ref obj returns a handle of type vpiModport.  
 \*\*\*\*\*/

```
module child1(interface simple s);
 c1 c_1(s);
 c1 c_2(s.master);
endmodule
```

\*\*\*\*\*/

for the port of module child1,

the vpiLowConn relationship= returns a handle of type vpiRefObj. The ~~where~~ vpiActual relationship= applied to the ref obj returns a handle of type vpiInterface

for that refObj,

the vpiPort relationship returns the ~~is~~ port of child1.

the vpiPortInst iteration returns handles to ~~is~~ s, s.master

the vpiActual relationship returns a handle to ~~is~~ i.

for the port of instance c\_1 :

vpiHighConn returns a handle of type ~~is a~~ vpiRefObj, ~~where~~. The ~~full type~~ vpiActual relationship applied to the ref obj handle returns a handle of type ~~is~~ vpiInterface.

for the port of instance c\_2 :

vpiHighConn returns a handle of type ~~is a~~ vpiRefObj, ~~where~~. The ~~full type~~ vpiActual relationship applied to the ref obj handle returns a handle of type ~~is~~ vpiModport.

\*\*\*\*\*/

Example 2: virtual interface declaration in a class definition

```
interface SBus; // A Simple bus interface
```

```
 logic req, grant;
```

```
 logic [7:0] addr, data;
```

```
endinterface
```

```
class SBusTransactor; // SBus transactor class
```

```
 virtual SBus bus; // virtual interface of type SBus
```

```
 function new(virtual SBus s);
```

```
 bus = s; // initialize the virtual interface
```

```
endfunction
```

```
 task request(); // request the bus
```

```
 bus.req <= 1'b1;
```

```
 endtask
```

```
 task wait_for_bus(); // wait for the bus to be granted
```

```
 @(posedge bus.grant);
```

```
 endtask
```

```
endclass
```

```
module devA(Sbus s) ... endmodule // devices that use SBus
```

```
module devB(Sbus s) ... endmodule
```

```
module top;
```

```
 SBus s[1:4] (); // instantiate 4 interfaces
```

```
 devA a1(s[1]); // instantiate 4 devices
```

```
 devB b1(s[2]);
```

```
 devA a2(s[3]);
```

```
 devB b2(s[4]);
```

```
initial begin
```

```

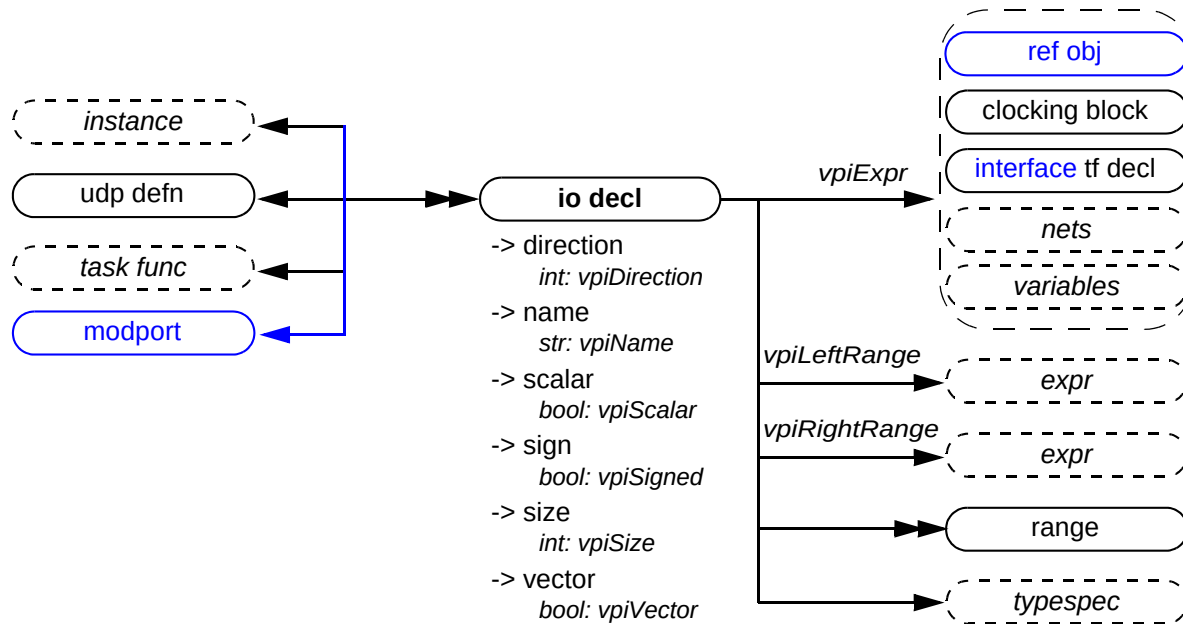
SbusTransactor t[1:4]; // create 4 bus-transactors and bind
t[1] = new(s[1]);
t[2] = new(s[2]);
t[3] = new(s[3]);
t[4] = new(s[4]);
end
endmodule

```

A **ref obj** is returned for the left hand side expression of the statement “bus = s” in the constructor of the class definition SBustransactor. The **vpiName** of that **ref obj** is “bus” and its **vpiDefName** is the name of the interface “SBus”. The **vpiActual** relationship returns the interface instance associated with that particular call to new after the assignment has executed. For example if it was “new ( s[1] )” **vpiActual** would return the interface s[1]. If **vpiActual** is queried before the assignment is executed, the method may return NULL if the virtual “bus” interface is uninitialized. The right hand side expression also returns a ref obj which **vpiActual** is the interface instance passed to the call to new.

IO declaration diagram: add ref obj, correct interface tf decl instead of tf decl  
 Question in note 3). Committee needs to decide.

### 32.10 IO declaration (supersedes P1364 26.6.4)

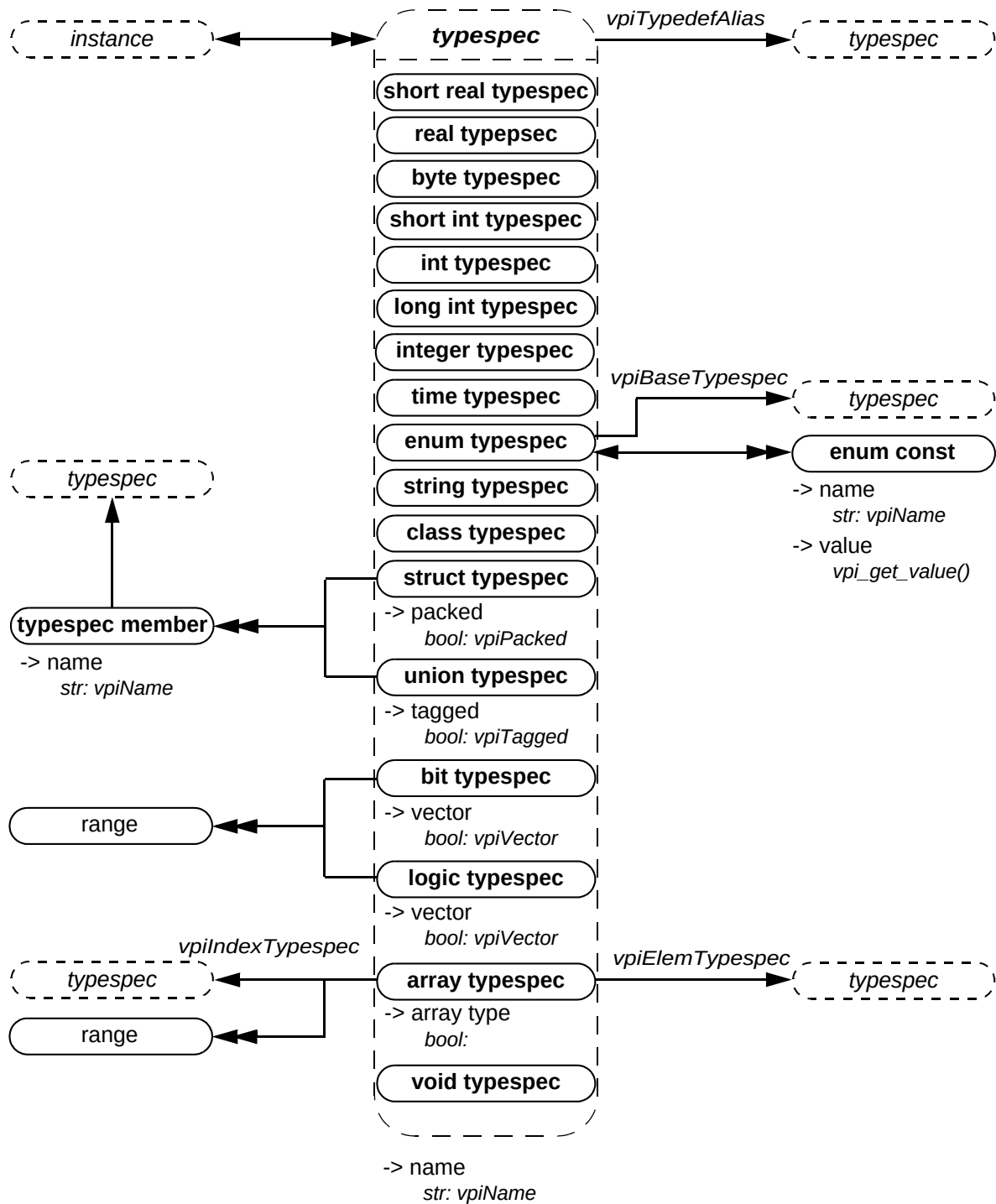


#### NOTES:

- 1) **vpiDirection** returns **vpiRef** for pass by **ref** ports or arguments.
- 2) A **ref obj** type handle may be returned for the **vpiExpr** of an **io decl** if it is passed by reference or if the **io decl** is an interface or a modport.
- 3) If the **vpiExpr** of an **io decl** is a **ref obj**, and if the **vpiActual** of the **ref obj** is an interface or modport declaration, then the **vpiDirection** of the **io decl** shall be undefined.

The typespec diagram :  
correct typo in note 3)

## 32.17 Typespec



NOTES:

- 1) If a typespec denotes a type that has a user-defined typedef, the **vpiName** property shall return the name of that type; otherwise, the **vpiName** property shall return NULL. Consequently the **vpiName** property returns NULL for any SystemVerilog built-in type. If the typespec denotes a type with a typedef that creates an alias of another typedef, then the **vpiTypedefAlias** of the typespec shall return a non NULL handle, which represents the handle to the aliased typedef. For example:

```
typedef enum bit [0:2] {red, yellow, blue} primary_colors;
typedef primary_colors colors;
```

If “h1” is a handle to the typespec colors, its **vpiType** shall return **vpiEnumTypespec**, the **vpiName** property shall return “colors”, **vpiTypedefAlias** shall return a handle “h2” to the typespec “primary\_colors” of **vpiType vpiEnumTypespec**. The **vpiName** property for “h2” shall return “primary\_colors” and its **vpiTypedefAlias** shall return NULL.

- 2) **vpiIndexTypespec** relation is present only on associative arrays and returns the type that is used as the key into the associative array.
- 3) If the value of the property **vpiType** of a typespec is **vpiStructTypespec** or **vpiUnionTypespec**, then it is possible to iterate over **vpiTypespecMember** to obtain the structure of the user-defined type. For each typespec member the typespec relation indicates the type of the member.
- 4) The property **vpiName** of a typespec member returns the name of the corresponding member, rather than the name (if any) of the associated typespec.
- 5) The name of a typedef may be the empty string if the typespec denotes typedef field defined inline rather than via a typedef declaration. For example:

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

The typespec representing the typedef C is a struct typespec; it has a single typespec member named B. The typespec relation for B returns another struct typespec that has no name and has a single typespec member named “a”. The typespec relation for “a” returns an int typespec.

- 6) If a type is defined as an alias of another type, it inherits the **vpiType** of this other type. For example:

```
typedef time my_time;
my_time t;
```

The **vpiTypespec** of the variable named “t” shall return a handle h1 to the typespec “my\_time” whose **vpiType** shall be a **vpiTimeTypespec**. The **vpiTypedefAlias** applied to handle h1 shall return a typespec handle h2 to the predefined type “time”.