IEEE P1800 First Recirculation Resolved Comments

Issue#	Sub-Group	Entity	Vote	E-No.	Clause/ Subclause	Paragraph / Figure / Table	Type of Comment (Technical / Editorial)	COMMENTS (Justification)	Proposed change	Priority	Disposition	Severity	Mantis Item #	MUST FIX	WG Res.
241		Entity-4	Disapprove	13	18.3	Paragraph	Technical	The second paragraph of clause 18.3 states "The values of variables used in assertions are sampled in the Preponed region of a time slot and the assertions are evaluated during the Observe region. This is explained in Clause 15, Scheduling Semantics". There is no mention of sampling based on clocking block if the signal used in the assertion is a clocking block signal. As per clause 15.3 "The sampling time of sampled data for property expressions is controlled in the clocking block". It is not very clear from these clauses how the sampling should be done for clocking block variables which are used in the concurrent assertion expression. Do we still continue and sample clocking block variables in the Preponed region irrespective of how the clocking block is sampling? Or are these variables are sampled based on the clocking block declaration ?	The second paragraph of clause 18.3, replace: The values of variables used in assertions are sampled in the Preponed region a time slot and the assertions are evaluated during the Observe region. This is explained in Clause 15, Scheduling Semantics By The values of variables used in assertions are sampled in the Preponed region a time slot and the assertions are evaluated during the Observe region. If a variable used in an assertion is a clocking block variable, it will be sampled twice: first based on the clocking block sampling time, and then in the Preponed region for the assertion sampling. This is explained in Clause 15, Scheduling Semantics. Add a new clause 18.17 to explain effect of clocking blocks on assertions. 18.17 Clocking blocks and Concurrent Assertions If a variable used in a concurrent assertion is a clocking block variable, it will be sampled twice: first based on the clocking block sampling time, and then in the Preponed region for the assertion sampling. Examples: module A; logic a, clk; clocking cb_with_input @(posedge clk); input a; property p1	High			626		Approved proposal in 626.pdf
236		Entity-4	Disapprove	8	11.4.3	Paragraph	Technical	The LRM requires that the terms of the default expression to be resolved in the context of the **caller**. This causes serious problems in the presence of hierarchically referenced tasks and functions when supporting separate compilation.	The cleanest semantic model is to strike the "caller scope" requirement. The textual change would be to change the paragraph: The default_value is an expression. The expression is evaluated in the scope of the caller each time the subroutine is called. The elements of the expression must be visible at the scope of subroutine and, if used, at the scope of the caller. If the default_value is not used, the expression is not evaluated and need not be	High			642		Approved proposal in 642.htm
229		Entity-4	Disapprove	1	4.11, 13.3	Paragraph	Technical	Unification of classes and structures Classes were added to SystemVerilog from a language that did not have all of the current SystemVerilog types, and those types have been left out of important functionality like randomization. It would be very helpful to use some of the same data types in the design in your testbench.	Replace the paragraph in clause 4.11 Clause 3.8 discusses assigning initial values to a structure. With the following paragraphs: Elements of a structure data type may be assigned individual default element values by using an initial assign-ment with the declaration of each element. The assigned expression must be a constant expression.	High			644		Approved proposal in 644.htm
230		Entity-4	Disapprove	2	6.6	Paragraph	Technical	The end of Clause 6.6 states "Note that automatic or dynamic variables cannot be written with nonblocking or continuous assignments. Automatic variables and dynamic constructs—objects handles, dynamic arrays, associative arrays, strings, and event variables—shall be limited to the procedural context." First of all, there is no definition of what the procedural context is. Uses of procedural context in other places in the LRM suggest that it is within a named or unnamed procedural block, function or task. That intent is certainly true for automatic variables, but the LRM clearly	Change the sentance: Note that automatic or dynamic variables cannot be written with nonblocking or continuous assignments. Automatic variables and dynamic constructs—objects handles, dynamic arrays, associative arrays, strings, and event variables—shall be limited to the procedural context. to: Note that automatic or elements of dynamic variables cannot be written with nonblocking or continuous assignments. Automatic variables and references to elements of dynamic constructs—class objects and dynamically-sized variables—shall be limited to the procedural context.	High			646		Approved proposal in 646.htm
231		Entity-4	Disapprove	3	6.9.2	Paragraph	Editorial	Rule 5 missed packed unions Packed arrays, packed structures, and built-in integral types are equivalent if	Change: Packed arrays, packed structures, and built-in integral types are equivalent if to: Packed arrays, packed structures, packed unions, and built-in integral types are equivalent if	Low	Resolved		620		Approved proposal in 620.htm

232		Entity-4	Disapprove	4	6.6/10.6/11.2	Paragraph	Technical	There are conflicting assumptions in the LRM regarding the lifetime of automatics in the context of tasks with fork-join_any/join_none. In 11.2 the "stack" semantics of automatics imply that automatics are gone one a task returns. However a fork-join_any/join_none thread can outlive the task. This can be a deeper issue if a task with automatics passes an automatic to a ref formal of a task with such a fork-join_any/join_none. In 6.6 it says: The lifetime of a fork…join, fork…join_any, or fork…join_none block shall encompass the execution of all processes spawned by the block. The lifetime of a scope enclosing any fork…join block	Add the following at the end of 11.2: A fork ... join_any or fork ... join_none block shall not refer to any ref parameter or to an automatic variable that is not local to the new thread. If the task completes prior to threads in a fork ... join_any or fork ... join_none any assignments to inout or out parameters that occur after the task has completed shall not be copied out. Alternatively, require that the references to ref or automatics in each thread occur before any possible suspension of the thread (delay, other task call, event control, etc).	High			652		Approved proposal in 652.htm
233		Entity-4	Disapprove	5	4.16 and 8.17.1	Paragraph	Technical	The LRM in section 3.16 and 7.18.1 mentions that a class is a bitstream type, and has a small example of class handle being used in a pack/unpack operation. There is no informative or normative text describing this functionality. For example, how does inheritance effect the bit ordering? What happens when the class handle is null? What happens when the class contains other class handles, like a list or a tree structure.	Change the sentence in 4.16 — Unpacked arrays, structures, or classes of the above types to — Unpacked arrays or structures of the above types Remove the text in 8.17.1	High			317		Approved proposal in 317.pdf
234		Entity-4	Disapprove	6	10.2.1 and 8.11	Paragraph	Technical	The stated sensitivities for always_comb are not correct with respect to original intent. Strictly speaking, 8.11 (Static Prefixes) does not permit hierarchically named objects as a static prefix since the base case (1) is only an identifier and the other cases permit only field and array selects. Although it was never intended to perm cross-module hierarchical references to objects or functions, it was expected that hierarchical references into interfaces and interface ports would be permitted. Note that package references are similarly ignored with the current wording.	Add 2 new clauses in 8.11 (labelled 2&3 and renumber remaining) which read: 2) A hierarchical reference to an object is a static prefix. 3) A package reference to net or variable is a static prefix. Add to 10.2.1: Hierarchical function calls and function calls from packages are analyzed as normal functions. References to class objects and method calls of class objects do not add anything to the sensitivity list of an always_comb.	High	Resolved		630		Approved proposal: Add 2 new clauses in 8.11 (labelled 2&3 and renumber remaining) which read: 2) A hierarchical reference to an object is a static prefix. 3) A package reference to net or variable is a static prefix. Add to 10.2.1: Hierarchical function calls and function calls from packages are analyzed as normal functions. References to class objects and method calls of class objects do not add anything to the sensitivity list of an always_comb.
235		Entity-4	Disapprove	7	11.4.2	Paragraph	Technical	A static task that passes arguments by reference causes illegal references if the actual arguments are automatic variables, and the formal argument arguments are hierarchically referenced. In general, hierarchical references t formal ref arguments are meaningless.	Add after: Only variables, not nets, can be passed by reference. the following sentance: Static tasks or functions shall not pass arugments by reference.	High			666		Approved Proposal in 666.htm
237		Entity-4	Disapprove	9	12.10	Paragraph	Technical	Clause 12.10 mentions that *this* is an implicit arugment to the new method, and that it may be used to qualify properties and methods. In order to create data structures like link lists or BDDs, it is usefull to use the bareword *this* as an explicit reference to the current instance.	Fix the BNF to permit this.	Low			616		Approved proposal in 616.htm

238		Entity-4	Disapprove	10	16.13	Paragraph	Technical	The clocking block section does not clearly say that there is a race condition between clocking block sampling and any always processes in the same module that are sensitive to the same clock. If such a process reads variables from the clocking block, the clocking block variables may or may not have been sampled when the always process executes. Note that this is not an issue if the clocking block is declared in a program block and the always process is declared in a module. For Example: module A; logic a, b, clk; clocking cb @(posedge clk); input b; endclocking always @(posedge clk) a = cb.b;	Clause 16.13, add the following: When a clocking block is declared in the same module as an always construct which is sensitive to the same clock, a race condition is present. Example: module A; logic a, b, clk; clocking cb @(posedge clk); input b; endclocking always @(posedge clk) a = cb.b; endmodule In the above case, the always construct may or may not see the current sampled value of the clocking block variable cb.b depending on how sampled values are updated. Commentary: It would also be possible to change the LRM such that a race-fre	High			671		Approved Proposal in 671.htm
239		Entity-4	Disapprove	11	17.2	Paragraph	Technical	There is no mention of final blocks in a program because the person that added the program block was unaware of its existance. This was an obvious omission. Without this, if the user writes their testbench entirely within a program, they have to add a module just to unclude a final block	Change A program block can contain one or more initial blocks. It cannot contain always blocks, UDPs, modules, interfaces, or other programs. to: A program block can contain one or more initial or final blocks. It cannot contain always blocks, UDPs, modules, interfaces, or other programs. Change the BNF to permit this.	Low			617		Approved proposal in 617.htm
242		Entity-4	Disapprove	12	19.2	Paragraph	Technical	Packages must not contain any processes, therefore wire delcarations with implicit continuous assignments are not allowed.	Add a new paragraph after the first paragraph in 19.2: Packages must not contain any processes, therefore wire delcarations with implicit continuous assignments are not allowed.	Low	Resolved		621		Approved proposal in 621.htm
243		Entity-4	Disapprove	13	19.3	Paragraph	Technical	There is ambiguity in the definition of compilation units and what tools should do with compilation units. Some people have interpreted the text of section 19.3 has having contradictory statements about the definition of compilation units. First, it is stated that "the exact mechanism for defining which files constitute a compilation unit is tool specific." But six paragraphs later it is stated that "the default is that each file is a separate compilation unit". The intention of the authors of this text (David Smith and myself) was that the command line invocation syntax and switches needed to define compilation units (i.e. "the exact mechanism" be left to the tool's discretion. But the general behavior, and default, is not to be left to the tool's discretion. One reason concerns pollution of the global namespace with symbols, types, variables, etc. If there is no limit on the scope-of-effect of symbols defined in the compilation unit space of a file, those symbols can inadvertently bleed over into the compilation unit spaces of subsequent files on a command l	In clause 19.3, modify the following sentences: WAS: "The exact mechanism for defining which files constitute a compilation un is tool specific. Two extreme cases are: 1) All files make a single compilation unit (in which case the declarations in the compilation-unit scope are accessible anywhere within the design) 2) Each file is a separate compilation unit (in which case the declarations in each compilation-unit scope are accessible only within its corresponding file)" SHOULD BE: "The exact mechanism for defining which files constitute a compilation unit is tool specific. However, the default compliant implementations shall have the same default behavior, and tools shall provide use models that allow any of the following three cases: 1) All files in the design make a single compilation unit (in which case the declarations in the compilation-unit scope are accessible anywhere within the design) 2) All files on a given compilation command line make a single compilation unit (in which case the declarations within those files are accessible anywhere else within the constructs defined wi WAS: "The default is that each file is a separate compilation unit". SHOULD BE: "The default definition of a compilation unit is defined in case 3) a each file is a separate compilation unit." DELETE: "A tool must also provide a mechanism (such as a command line swi specifies that all of the files compiled together are a single compilation unit." (Sc this is now enumerated as item 2) in the list above)	High	Resolved		614		Approved proposal in 614.htm
244		Entity-4	Disapprove	14	20.2	Paragraph	Technical	Currently, the actuals of interface ports are not restricted in terms of hierarchical references to interface instantiations. This can cause problems similar to other circular elaboration dependencies with generates that IEEE-1364 very carefully avoids.	Add at the end of 20.2: If the actual of an interface port connection is a hierarchical reference to an interface or a modport of a hierarchically referenced interface, the hierarchical reference shall refer to an interface instance and shall not resolve through an arrayed instance or a generate block.	High	Resolved		632		Approved propsal in 632.htm
245		Entity-4	Disapprove	15	20.2	Paragraph	Technical	Non-arrayed module instances can depend on an arrayed interface instantiation. Such a module should not be permitted to do an upwards defparam in the same manner as other array instances are not permitted to do such a defparam. In terms of the elaboration flow, any module with a reference to an arrayed interface should itself be treated as an arrayed instance.	Add at the end of 20.2: Any instantiation whose port actuals refer to an arrayed interface shall itself be treated as an arrayed instantiation when considering the elaboration flow [as described in the 1364-2005 draft].	High	Resolved		624		Approved proposal in 624.htm

246		Entity-4	Disapprove	16	20.2	Paragraph	Technical	Section 20 isn't at all clear regarding composibility of modports and the visibility of types and parameters subsequent to composition. For example, given a modport such as: modport mp (child_interface_1.mp1, child_interface_2.mp2); Assume that "child_interface_1" and "child_interface_2" have a type "T" as does the interface containing the modport. Which, if any, type "T" is visible via mp? If mp1 and mp2 both make some name "W" visible, is that legal? Without a general mechanism for renaming, it seems that these should be illegal compositions.	Add 20.4.6 "Visibility through modport declarations": A modport makes types and parameters from its interface visible when referencing the modport. A modport_hierarchical_ports_declaration makes all names visible from its referenced modport directly visible as names in the containing modport. It shall be an error if any name in a modport has more than one definition, even if the name is implicitly defined by the inclusion of a type or parameter name from an interface.	High	Resolved		629		Approved proposal in 629.htm
247		Entity-4	Disapprove	17	20.4	Paragraph	Technical	p. 302, 3rd paragraph states "No hierarchical references shall be permitted within a modport." However, the preceding example shows a modport: modport master2 (ch1.master, ch2.master); This looks hierarchical to me. In addition, the syntax 20-1 shows: modport_declaration ::= modport modport_item modport_item ::= modport_identifier(modport_ports_declaration) modport_ports_declaration ::= modport_hierarchical_ports_declaration modport_hierarchical_ports_declaration ::= interface_instance_identifier.modport_identifier	Change "No hierachical references shall be permitted within a modport." to "No hierarchical references shall be permitted within a modport other than modport hierachical port declarations."	High	Resolved		629		Approved proposal in 629.htm
248		Entity-4	Disapprove	18	20.4	Paragraph	Technical	P. 302: there is no example of how to use the interface i1 in the example. Consider the following: interface i3; typedef int T; wire a, b, c, d; modport master (input a, b, output c, d); modport slave (output a, b, input c, d); endinterface interface i3b; typedef real T; wire a, b, c, d; modport master (input a, b, output c, d); modport slave (output a, b, input c, d); endinterface interface i1; i3 ch1(); i3b ch2(); modport master2 (ch1.master, ch2.master); endinterface It isn't clear what modport "master2" exports. Following the example in 20.9, the type "T" should be directly visible when using "master2". Since there are two types "T" this conflicts. This also impacts the other elements visible in the modports. This should be modified to explicitly rename items.	The example on p.302 should be extended to show: module m1(interface.master2 i); // using the master2 // modport always @(posedge i.ch1.a) // Need to distinguish between // ch1.a and ch2.a ... ie. referring to i.a would be ambiguous since the master2 modport includes two references to a signal 'a'. An example should also be included that shows the following to be legal: interface i3; wire a, b, c, d; modport master (input a, b, output c, d); modport slave (output a, b, input c, d); endinterface interface i1; i3 ch1(), ch2(); modport master2 (ch1.master, input .A(ch2.a), .B(ch2.b), output .C(ch2.c), .D(ch2.d)); The hierarchical modport shall be first to avoid ambiguity about port direction.	High	Resolved		629		Approved proposal in 629.htm

249		Entity-4	Disapprove	19	20.6.2, 20.6.3	Paragraph	Editorial	The first modport in the example in 20.6.2 has incorrect syntax for the import statement. If the "task" keyword is present, then the full prototype is required. The same problem exists for the "export" example in 20.6.3	20.6.2: import slaveRead, slaveWrite); 20.6.3: export Read, Write);	Low	Resolved		622		Approved proposal in 622.htm
250		Entity-4	Disapprove	20	21.2	Paragraph	Technical	Can the type_option.strobe field be set or reset at any point in simulation? Does it apply to implicit sampling only? The latter point "makes sense", as explicit sampling with the sample() built-in method presumably should be done exactly as the user codes it. The set/reset of the strobe option is more problematic: I think it very likely that an implementation could be more efficient if the value of the strobe option is determined at construction time and not changed thereafter. This would make the strobe option analogous in its specification to the per_instance option.	At end of paragraph, top page of 323: The strobe option applies to implicit sampling only, when a clocking event is specified with the covergroup declaration. The strobe option, like the per_instance option, can be set only in t covergroup definition. (The second sentence could be re-iterated in or moved to Section 21.6, first paragraph page 337.)	High			636		Approved proposal in 636.htm
251		Entity-4	Disapprove	21	21.2	Paragraph	Technical	Reading the section on packages, I discover tha covergroups can appear in them. Makes sense yet Section 21 itself does not mention this, while it mentions four other places a covergroup can appear.	Top of page 322, change: A covergroup can be defined in a module, ... To A covergroup can be defined in a package, module, ...	Low			643		Approved proposal in 643.htm
252		Entity-4	Disapprove	22	21.4.1	Paragraph	Technical	The LRM does not specify whether this coverpoint transition bin should increment once or twice per sample period, in the case where th sequence b => c is completed: bins binname = { a => b => c , b => c };	Add: A transition bin is incremented at most once per sample period.	High			637		Approved proposal in 637.htm
253		Entity-4	Disapprove	23	21.4.4	Paragraph	Technical	A simple clarification is requested.	Add "and cross coverage" after "are excluded from coverage" in text after the example.	Low			649		Approved proposal in 649.htm
254		Entity-4	Disapprove	24	21.4.5	Paragraph	Technical	A simple clarification is requested.	Add "and cross coverage" after "are excluded from coverage" in text after the example.	Low			649		Approved proposal in 649.htm
255		Entity-4	Disapprove	25	21.5	Paragraph	Technical	Which bins are created when cross bins are limited by cross_auto_bin_max? On the theory that crosses are usually sparse and cross coverage is difficult to predict, it would be reasonable to limit cross bins to the first N unique bins encountered, when cross_auto_bin_max == N. This would be analogous to limiting a sparse array after its number of elements reaches N. Otherwise, some subset of N bins among an arbitrary number of dimensions (coverpoints) must be	Middle of page 333 add: When cross_auto_bin_max is set, cross bins are limited to the first cross_auto_bin_max number of unique bins incremented during simulation.	High			641		Approved proposal in 641.htm
256		Entity-4	Disapprove	26	22.2	Paragraph	Technical	P-1800 permits "parameter" to mean "localparam" within a generate block, package, or compilation unit scope. This is confusing to users for 2 reasons: 1) "defparam" to such a parameter is not permitted. 2) syntactically, you are not permitted to have "locaparam type T = int;" but you are	Change: "Unlike nonlocal parameters, local parameters can be declared in a generate block, in a package, or in a compilation unit scope. In these contexts, the parameter keyword can be used as a synonym for the localparam keyword." To: "Unlike nonlocal parameters, local parameters can be declared in a generate	Low	Resolved		674 (625.htm)		Proposed change rejected due to other ballot issue (223) that has resolved this issue. Approved changes are in 625.htm

258		Entity-4	Disapprove	27	28.4.5	Paragraph	Technical	The proposal for Mantis item #0000274 missed a bullet item in Section 28 regarding allowing packed bit vectors as function return types. That is no longer allowed as per #0000274.	In 28.4.5, DELETE the specified bullet item in the text. This is also captured in more readable HTML in the proposal for Mantis item #385 (http://www.eda.org/svdb/file_download.php?file_id=645&type=bug). 28.4.5 Function result An imported function declaration must explicitly specify a data type or void for the type of the function's return result. Function result types are restricted to small values. The following SystemVerilog data types are allowed for imported function results: — void, byte, shortint, int, longint, real, shortreal, chandle, and string DELETE THE FOLLOWING BULLET ITEM: — packed bit arrays up to 32 bits and all types that are eventually equivalent to packed bit arrays up to 32 bits — scalar values of type bit and logic The same restrictions apply for the result types of exported functions.	High			385		Approved proposal in 385.htm
259		Entity-4	Disapprove	28	E 6.3	Paragraph	Technical	Clarify enumeration type support (1) accept 4-state 'integer' and 'time'; specify they are treated as packed 4-state arrays in DPI canonical format. (2) state that enumeration base type governs: (2a) passing by value or reference for input direction mode; (2b) legality of using vars of the type as fn return types. (Mantis item 407)	Add this clarifying material to E.6.3 item on enum types. In E.6.3 Replace — Enumeration types are represented as the types associated with them. Enumerated names are not available on the C side of the interface. WITH	High			407		Approved proposal in 407.htm
261		Entity-4	Disapprove	29	E 7.7	Paragraph	Technical	List of 'small values' omits bit and logic scalars; they are defined in svdpi.h as C unsigned char and are listed as legal fn return types in 28.4.5, E.7.9 (the latter are restricted to 'small values'). (Mantis item 406 at http://www.eda.org/svdb/file_download.php?file_id=647&type=bug)	Add bit and logic scalars to E.7.7's list of small values. In E.7.7 Replace input arguments of imported functions implemented in C shall always have a const qualifier. input arguments, with the exception of open arrays, are passed by value or by reference, depending on the size. 'Small' values of formal input arguments are passed by value. The following data types are considered small: — byte, shortint, int, longint, real, shortreal — chandle, string input arguments of other types are passed by reference. WITH input arguments of imported functions implemented in C shall always have a const qualifier. input arguments, with the exception of open arrays, are passed by value or by reference, depending on the size. 'Small' values of formal input arguments are passed by value. The following data types are considered small: — byte, shortint, int, longint, real, shortreal — scalar bit and logic — chandle, string input arguments of other types are passed by reference.	Low			406		Approved proposal in 406.htm
266		Entity-6	Disapprove	4	General			In many places throughout the LRM, the usage of "Note" and "Notes" is not correct. In IEEE standards, notes are informative, and not a required part of the standard. It is clear that in many places in the LRM, there are notes that are both intended to be normative, and are required for proper implementation of the standard. I believe that IEEE standards also require that notes be in a separate paragraph, and use a smaller font. In many places in the LRM, there are sentances (often in the middle of a paragraph) that begin with "Note..." but are not in a smaller font. It is unclear as to whether these notes are meant to be an informative notes, or an important normative fact that users and/or implementors need to observe. All usages of the word "Note" "note", "NOTE", "Notes" "notes" and "NOTES" should be reviewed and brought into IEEE standards compliance.	Clauses that should be reviewed as to whether the terms "note" or "notes" are correctly used as only informative text include: 3.8, 4.3.3, 4.7.2, 4.7.3, 4.9 (multiple occurences), 4.11, 4.13, 4.14, 4.15, 5.2, 5.3, 5.6.2, 5.8, 5.14.1 (multiple occurences), 6.6 (multiple occurences), 6.7, 6.9, 6.9.1, 6.9.2, 8.3, 8.12, 8.13.2, 8.16 (multiple occurences), 8.19, 9.4, 9.4.1, 9.4.1.1, 9.6, 9.10 (multiple occurences), 11.4.2, 11.4.3, 11.5 (multiple occurences), 12.6, 12.7, 12.8, 12.10, 12.11, 12.17, 12.24 (multiple occurences), 13.4.3, 13.4.11 (multiple occurences), 13.5.3, 13.11, 14.3.7, 15.2, 16.13, 17.4, 18.2, 18.3, 18.6, 18.7.3, 18.8, 18.11.2, 18.11.3, 18.13.1, 18.13.2, 18.15 (multiple occurences), 19.6, 19.7, 19.12.2, 20.2.1, 20.3, 20.4 (multiple occurences), 20.6.4, 20.8.1, 20.10.1, 22.2, 24.17, 25.2, 28.3, 28.4.1.1, 28.4.1.4, 28.4.3 (multiple occurences), 28.4.4 (multiple occurences), 28.4.6, 28.6 (multiple occurences), 28.8 (multiple occurences), 29.3.1, 29.3.2.1 (multiple occurences), 29.4.2, 29.5.1, 29.5.2, 30.2.2.1 (multiple occurences), 30.2.2.2, 30.2.2.5, 30.4.3 (multiple occurences), 30.4.4, 31.8.3, 31.8.4, 31.10, 32.2 through 32.49				266		Approved proposal in 266.pdf

266		Entity-6	Disapprove	4	General			In many places throughout the LRM, the usage of "Note" and "Notes" is not correct. In IEEE standards, notes are informative, and not a required part of the standard. It is clear that in many places in the LRM, there are notes that are both intended to be normative, and are required for proper implementation of the standard. I believe that IEEE standards also require that notes be in a separate paragraph, and use a smaller font. In many places in the LRM, there are sentances (often in the middle of a paragraph) that begin with "Note..." but are not in a smaller font. It is unclear as to whether these notes are meant to be an informative notes, or an important normative fact that users and/or implementors need to observe. All usages of the word "Note" "note", "NOTE", "Notes" "notes" and "NOTES" should be reviewed and brought into IEEE standards compliance.	Clauses that should be reviewed as to whether the terms "note" or "notes" are correctly used as only informative text include: 3.8, 4.3.3, 4.7.2, 4.7.3, 4.9 (multiple occurences), 4.11, 4.13, 4.14, 4.15, 5.2, 5.3, 5.6.2, 5.8, 5.14.1 (multiple occurences), 6.6 (multiple occurences), 6.7, 6.9, 6.9.1, 6.9.2, 8.3, 8.12, 8.13.2, 8.16 (multiple occurences), 8.19, 9.4, 9.4.1, 9.4.1.1, 9.6, 9.10 (multiple occurences), 11.4.2, 11.4.3, 11.5 (multiple occurences), 12.6, 12.7, 12.8, 12.10, 12.11, 12.17, 12.24 (multiple occurences), 13.4.3, 13.4.11 (multiple occurences), 13.5.3, 13.11, 14.3.7, 15.2, 16.13, 17.4, 18.2, 18.3, 18.6, 18.7.3, 18.8, 18.11.2, 18.11.3, 18.13.1, 18.13.2, 18.15 (multiple occurences), 19.6, 19.7, 19.12.2, 20.2.1, 20.3, 20.4 (multiple occurences), 20.6.4, 20.8.1, 20.10.1, 22.2, 24.17, 25.2, 28.3, 28.4.1.1, 28.4.1.4, 28.4.3 (multiple occurences), 28.4.4 (multiple occurences), 28.4.6, 28.6 (multiple occurences), 28.8 (multiple occurences), 29.3.1, 29.3.2.1 (multiple occurences), 29.4.2, 29.5.1, 29.5.2, 30.2.2.1 (multiple occurences), 30.2.2.2, 30.2.2.5, 30.4.3 (multiple occurences), 30.4.4, 31.8.3, 31.8.4, 31.10, 32.2 through 32.4		Resolved		695		Approved proposal in 695.pdf
260		Entity-4	Disapprove	5	E 6.6	Paragraph	Editorial	Erroneous cross-reference needs replacing, refer to new number for sec. dealing with packe type equivalence. Cross-ref will otherwise baffle readers. (Mantis item 386 at http://www.eda.org/svdb/bug_view_page.php?bug_id=0000386)	change cross refs to "(see E.6.5 and 6.9.2)" In E.6.6 Replace 1) If a packed part of an array has more than one dimension, it is linearized as specified by the equivalence of packed types (see E.6.5 and 6.9.3). WITH 1) If a packed part of an array has more than one dimension, it is linearized as specified by the equivalence of packed types (see E.6.5 and 6.9.2).	Low					Minor editing issue implemented by the editor.
268		Entity-6	Disapprove	6	Annex C, Annex D, Annex G			Annex C, D, and G begin with text that is not in a subclause.	I believe that IEEE document rules require that if there are subclauses, then all text must be contained in subclauses to prevent ambiguities with cross referencing.						Minor editing issue implemented by the editor.
203		Entity-2	Disapprove	2			Technical	Packed unions shall not be restricted to equal length items. In case of generic width of parameters, or generic typed parameters, this cannot be guaranteed upfront. Instead, automatic filling up of different length items, shall be defined. Filling up is not unusual to the Verilog language. Using unpacked unions would work, but only for simulation. They lack a clearly defined semantic for synthesis.			Resolved		675		No action taken because the negative ballot issue was not accompanied with a proposed resolution in sufficient detail in a legible form so that the specific wording of the changes that will cause the negative voter to change his or her vote to "approve" can readily be determined.
204		Entity-2	Disapprove	3			Technical	Bind shall also be defined for classes. Bind is actually defined for programs, modules, and interfaces only. Because classes are the basic building blocks for testbenches, such as modules are for design, it shall be possible to bind one class to another. Applications for this enhancement are binding various constraints or coverage items to contol a testbench, built out of classes. A similar mechanism is available and heavily used in the langauge E.					737		While it has merit, the committee believes it is either not feasible to implement at this time or not in the scope or goals of the P1800 project as it was defined & agreed upon by the P1800 WG
205		Entity-2	Disapprove	4			Technical	Add support of unconstraint arrays also to tasks and functions (as available in VHDL). Currently unconstraint arrays, and access to array attributes is restricted to modules. Using dynamic arrays instead, is possible, but restricted to simulation because dynamic arrays lack hardware semantic.			Resolved		676		The committee read and considered this feedback. While it has merit, the committee believes it is either not feasible to implement at this time or not in the scope or goals of the P1800 project as it was defined & agreed upon by the P1800 WG
206		Entity-2	Disapprove	5			Technical	Add support of unconstraint arrays also to tasks and functions (as available in VHDL). Currently unconstraint arrays, and access to array attributes is restricted to modules. Using dynamic arrays instead, is possible, but restricted to simulation because dynamic arrays lack hardware semantic.							The committee read and considered this feedback. While it has merit, the committee believes it is either not feasible to implement at this time or not in the scope or goals of the P1800 project as it was defined & agreed upon by the P1800 WG

207		Entity-2	Disapprove	6			Technical	Variables shall be allowed for slicing vectors. This feature is supported by VHDL and synthesis tools working with VHDL. It is heavily used in generic designs and very useful here. The constant offset slice, as supported by SystemVerilog is more save, however not sufficient in all cases. Variables slicing vecors could be achieved using for loops to split operations on a slice into operations on array elements. This however causes a lot of coding overhead.			Resolved		678		The committee read and considered this feedback. While it has merit, the committee believes it is either not feasible to implement at this time or not in the scope or goals of the P1800 project as it was defined & agreed upon by the P1800 WG
257		Entity-4	Disapprove	#REF!	24	Paragraph	Technical	The LRM does not specify how the file IO tasks work with new SV data types	Add a new section that shows how $display and $fget work with C data types	Low	Resolved		697		The committee read and considered this feedback. While it has merit, the committee believes it is either not feasible to implement at this time or not in the scope or goals of the P1800 project as it was defined & agreed upon by the P1800 WG
263		Entity-6	Disapprove	1	General			As a general comment, Sutherland HDL feels that the P1800 standard, which is a set of extenstions to the P1364 standard, should not be stand-alone reference manual. These extensions should be integrated into the P1364 standard, making one unified Verilog standard, rather than two Verilog standards. However, we recognize that a majority of the P1800 working group members are not in agreement with this. Sutherland HDL will not hold back its approval of the P1800 standard due to this general comment. We will change our vote to APPROVE if the following six concerns are adequately addressed (assuming that the resolutions to other ballot comments are also satisfactory).							The committee and the entity were in agreement that no action is required for this item.
264		Entity-6	Disapprove	2	9.4			The unique and priority decision modifiers state that a violoation results in a warning. These need to be errors, not warnings. I have spoken with design and verification engineers at several companies about this, and all feel strongly that these need to be non-fatal error messages. Many companies grep log files for errors, but are like to ignore warnings.			Resolved		677		While it has merit, the committee believes it is either not feasible to implement at this time or not in the scope or goals of the P1800 project as it was defined & agreed upon by the P1800 WG
265		Entity-6	Disapprove	3	General			Throughout the LRM, there are references to errors and warnings that implementations can or shall issue. There is no definition as to the meaning of "error" and "warning". Does an "error" terminate compilation, elaboration or run-time execution? Should an error message include the term "error" and a warning message the term "warning"? It appears that "error" and "warning" are often used either inconsistently, or interchangabley. The standard needs to define the meaning of these terms, and ensure that all references to the terms are used clearly and consistently.	I suggest that there be three definitions: - Fatal error: Shall result in the generation of an error message informing the user that a specific syntatic or sematic condition has been detected. The message reporting the fatal error shall contain the word "Error", "error" or "ERROR". A fatal error shall result in the termination of the current process of compilation, elaboration or run-time execution. An implementation may report other fatal errors, non-fatal errors or warnings before terminating. - Non-fatal error: Shall result in the generation of an error message informing the user that a specific syntatic or sematic condition has been detected. The message reporting the non-fatal error shall contain the word "Error", "error" or "ERROR". A non-fatal error shall not terminate the current process of compilation, elaboration or run-time execution. - Warning: Shall result in the generation of an warning message informing the user that a specific condition has been detected and a specific Verilog semantic rule applied. The message reporting the warning shall contain the word "Warning", "warning" or "WARNING". Clauses that are ambiguou		Resolved		698		The committee read and considered this feedback. While it has merit, the committee believes it is either not feasible to implement at this time or not in the scope or goals of the P1800 project as it was defined & agreed upon by the P1800 WG

IEEE P1800 First Recirculation Resolved Comments

STU1								The rules in Clause 6.4 for in-line variable initialization, as part of a variable declaration, needs to be reviewed. First, the rules conflict with the P1364 standard for in-line varialble initialization. The two standards need to be in agreement on this. Second, there has been a debate within the P1800 subcommittees as to whether the initialization rules described in 6.4 allow simulation to correctly initialize combinatorial logic described by always_comb, always @(), continuous assignments, combinational UDPs, and inter-module port connections. suggested_remedy = Shalom Brestnicker has summarized the problem and proposed a solution as follows: There are cases where a net or variable is intended to be a strict combinational function of other nets/variables/constants. In some cases, it does not work correctly as currently defined because the result is evaluated only due to events/changes on the inputs, and the inputs in these cases are constants or have an initial value which does not create an event, and/or the construct is made sensitive to input eve event occurs.			Resolved		667 & 682		Some of the fundamental concerns raised by this issue have been addressed in the proposal in 682.htm. The rest are more appropriately addressed by the 1364. This committee has forwarded a specific recommendation to 1364 addressing a subset of the issue. The following change was approved for insertion in the P1364 LRM:In the section that was 5.6.1 in draft 5, CHANGE A continuous assignment statement (Clause 6) corresponds to a process, sensitive to the source elements in the expression. When the value of the expression changes, it causes an active update event to be added to the event queue, using current values to determine the target. TO: A continuous assignment statement (Clause 6) corresponds to a process, sensitive to the source elements in the expression. When the value of the expression changes, it causes an active update event to be added to the event queue, using current values to determine the target. A continuous assignment
STU2								Clause 18 on assertions is ambiguous as to whether identifiers must be declared prior to being referenced in a property or sequence block. Some existing implementations require the declaration come first, and some do not. It is my understanding that Verilog allows identifiers to be referenced prior to declaration in specify blocks, tasks, and functions. If this is correct, then it would be reasonable for users to expect that property and sequence blocks be consistent with that rule. Allowing reference before declaration might also help with binding externally declared properties and sequences to a design block. In either case, the P1800 LRM should explicitly state the rule of order for declaration versus reference in a property or sequence block. The current ambiguity has lead to code that is not portable across multiple tools. suggested_remedy =							The committee read and considered this feedback. While it has merit, the committee believes it is either not feasible to implement at this time or not in the scope or goals of the P1800 project as it was defined & agreed upon by the P1800 WG
95		Entity-1	Approve	95	12.25		Technical	section 12.25 bullet 2)		High		minor	512		Approved proposal in 512.htm
101		Entity-1	Approve	101	5.14.2.4, 5.14.2.5		Technical	undefined queue_type in prototypes pop_front and pop_back		High		minor	523		Approved proposal in 523.htm
1		Entity-1	Approve	1	11.4.5		Technical	SV-BC Issue 5: 1038 Optional argument list for subroutine calls - see bug note	See mantis item	High	Transfer to SV-EC	feature	93	YES	Approved proposal in 93.htm
5		Entity-1	Approve	5	13.13.1		Technical	what is the root thread? this term is not defined but used	See mantis item	High		feature	319	YES	Approved proposal in 319.htm
6		Entity-1	Approve	6	4.11		Technical	unions seem to allow some problematic members	See mantis item	High	Resolved	minor	336	YES	Approved proposal in 336.htm
7		Entity-1	Approve	7	19.2		Technical	The semantics of an anonymous program is not defined anywhere	See mantis item	High		feature	344	YES	Approved proposal in 344.pdf
8		Entity-1	Approve	8	13.13		Technical	what is the seed of top level threads in the case of hierarchical seeding	See mantis item	High		feature	370	YES	Approved proposal in 370.htm
9		Entity-1	Approve	9	32.13		Technical	vpiMember rather than vpiMembers in object model for nets	See mantis item	High		minor	382	YES	Approved proposal in 382.pdf
10		Entity-1	Approve	10	6.6, 11.4.2		Technical	Operations allowed on ref arguments not specified	See mantis item	High		major	409	YES	Approved proposal in 409.pdf
11		Entity-1	Approve	11	8.5, 8.19		Technical	Inconsistent wildcard treatment	See mantis item	High		major	410	YES	Approved proposal in 410.htm
12		Entity-1	Approve	12	11.4.3		Technical	Problem with default arguments	See mantis item	High		major	411 See 410	YES	Approved proposal in 410.htm
15		Entity-1	Approve	15	29.3.2.1		Technical	29.3.2.1 violates a principle design feature of VPI	See mantis item	High		major	425	YES	Approved proposal in 425.pdf
17		Entity-1	Approve	17	32.8		Technical	32.8 backwards compatibility broken here	See mantis item	High		major	452	YES	Approved proposal in 452.pdf
18		Entity-1	Approve	18	32.28		Technical	32.28 Should document backwards compatibility issue	See mantis item	High		minor	483	YES	Approved proposals in 485a.pdf and 485b.pdf

IEEE P1800 First Recirculation Resolved Comments

													485 B		
20		Entity-1	Approve	20	18, 24.5		Technical	true/false definitions?	The definition was made precise, also ref. to Clause 24.10 where true / false is defined	High	closed	major	509	YES	Approved proposal in 509.htm
21		Entity-1	Approve	21			Technical	inconsistency of port connection rules	See mantis item	High	Resolved	major	513	YES	Approved proposal in 513.pdf
22		Entity-1	Approve	22	14.3		Technical	mailbox type mismatch	See mantis item	High		major	514	YES	Approved proposal in 514.pdf
25		Entity-1	Approve	25	32.14, 32.22		Technical	If VPI references to elements of dynamic objects do not prevent reclaiming the associated memory ("12.26 Memory Management"), the VPI user needs a way to determine whether or not the associated object still exists and/or is valid. This applies both to VPI handles for elements of dynamic arrays and to handles for non-static members of classes.	Provide a property to determine whether a handle is still "live".	High		major	526	YES	Approved proposal in 526.pdf
27		Entity-1	Approve	27	32.22		Technical	From "Annex I: sv_vpi_user.h", it appears that the property name "vpiAccessType" in the diagram and Note 6 should be replaced by "vpiVisibility". However, it is not the class var itself that should have this property, but rather any variable, parameter, or task func that can be declared within a class definition.	Remove vpiAccessType and Note 6 from this section. Add vpiVisibility to other diagrams as appropriate.	High		major	533	YES	Approved proposal in 533.pdf
29		Entity-1	Approve	29	19.11.3, 19.11.4		Technical	Do package references get connected in implicit name and * port connections	See mantis item	High	Resolved	minor	548	YES	Approved proposal in 548.htm
35		Entity-1	Approve	35	19.8		Technical	19.8 Default rules for port kinds do not specify variable case	See mantis item	High	Resolved	minor	578	YES	Approved proposal in 578.pdf
37		Entity-1	Approve	37	32.25		Technical	"32.25 Module path, path term" does not reflect all the changes made inP1364		High		feature	377		Approved proposal in 377.pdf
40		Entity-1	Approve	40	29.1.1		Editorial	29.1.1 should be removed		High		major	422		Approved proposal in 422.pdf
41		Entity-1	Approve	41	29.2		Editorial	29.2 This section is no longer needed		High		major	423		Approved proposal in 423.pdf
42		Entity-1	Approve	42	29.3.2.1		Technical	29.3.2.1 Return value of vpi_get_assertion_info() is backwards		High		major	426		Approved proposal in 425.pdf
43		Entity-1	Approve	43	29.3.2.2		Technical	29.3.2.2 should be removed		High		major	427		Approved proposal in 427.pdf
46		Entity-1	Approve	46	29.5		Technical	29.5 polutes the VPI name space		High		major	431		Approved proposal in 431.pdf
47		Entity-1	Approve	47	29.5.2		Technical	29.5.2 Don't understand note at the end		High		minor	432		Approved proposal in 432.pdf
48		Entity-1	Approve	48	30		Editorial	30 - incorrect title		High		major	433		Approved proposal in 433.htm
50		Entity-1	Approve	50	30.4.4		Technical	30.4.4 Last paragraph does not make sense		High		major	437		Approved proposal in 437.pdf
52		Entity-1	Approve	52	Annex I		Editorial	Annex I - implied properties are really return values		High		trivial	439		Approved proposal in 439.pdf
53		Entity-1	Approve	53	Annex I		Technical	Annext I - vpiUserDefinedClass		High		trivial	440		Approved proposal in 440.pdf
55		Entity-1	Approve	55			Technical	Several Superseded sections missing information from P1364		High		major	445		Approved proposal in 445.pdf and in P1364: 445b.pdf
													445b		
56		Entity-1	Approve	56	32.5		Technical	32.5 incorrect note		High		trivial	446		Approved proposal: Change "vpiIterate(vpiTaskFunc)" to "vpi_iterate()".
57		Entity-1	Approve	57	32.5		Technical	32.5 missing types for properties		High		trivial	447		Approved proposal in 447.pdf
58		Entity-1	Approve	58	32.8		Technical	32.8 First note is not necessary and is incorrect		High		trivial	448		Approved proposal in 452.pdf
59		Entity-1	Approve	59	32.11		Technical	32.11 vpiPortType is missing from the include file		High		trivial	449		Approved proposal in 449.htm
60		Entity-1	Approve	60			Technical	Several sections missing access by index properties		High		major	453		Approved proposal in 453.pdf
61		Entity-1	Approve	61	32.13		Technical	32.13 poorly drawn diagram - regression		High		minor	456		Approved proposal: Move the vpiIndex tags to be more clear. Change the 'instance' class reference to 'expr'. Change the 'scope' class reference to 'expr'. Add a double arrow from 'module' to 'array net'.
62		Entity-1	Approve	62	32.13		Technical	32.13 LHS of bottom diagram incorrectly drawn		High		minor	457		Approved proposal in 457.pdf
63		Entity-1	Approve	63	32.13		Technical	32.13 need parens at end of routine properties		High		trivial	460		Approved proposal in 460.pdf
66		Entity-1	Approve	66	32.20		Technical	32.20 'nets' should be 'scope'		High		trivial	466		Approved proposal: Replace the dotted enclosure labeled 'nets' which is part of the parameter diagram with 'scope'.
67		Entity-1	Approve	67	32.21		Technical	32.21 extend or extends?		High		trivial	467		Approved proposal in 467.pdf
68		Entity-1	Approve	68	32.21		Technical	32.21 First note needs some work		High		minor	468		Approved proposal in 468.pdf
69		Entity-1	Approve	69	32.21		Technical	32.21 Note 2 is a partial sentence		High		minor	469		Approved proposal in 469.pdf
70		Entity-1	Approve	70	32.22		Technical	32.22 remove notes 5 and 6		High		minor	474		Approved proposal in 474.pdf
71		Entity-1	Approve	71	32.23		Technical	32.23 property return values should be specified in include file		High		minor	475		Approved proposal in 475.pdf
72		Entity-1	Approve	72	32.25		Technical	32.25 missing newer changes from P1364		High		minor	477		Approved proposal in 477.pdf
73		Entity-1	Approve	73	32.26		Technical	32.26 'refobj' should be 'ref obj'.		High		trivial	479		Approved proposal: Change the object reference 'refobj' with 'ref obj'
75		Entity-1	Approve	75	32.27		Technical	32.27 Don't need vpiBuiltIn property		High		minor	482		Approved proposal in 482.pdf
77		Entity-1	Approve	77	32.34		Technical	32.34 Order of operands should be in 32.39		High		minor	487		Approved proposal in 686.pdf
78		Entity-1	Approve	78	32.38		Technical	32.38 vpiMemory should be removed		High		minor	488		Approved proposal: Remove the two references to memory word.

79		Entity-1	Approve	79	32.38		Technical	32.38 Need better definition of a ref obj object.		High		major	489		Approved proposal in 489.pdf
80		Entity-1	Approve	80	32.39		Technical	32.39 Indexed part select is missing		High		minor	490		Approved proposal in 490.pdf
81		Entity-1	Approve	81	32.40		Technical	32.40 Missing null stmt definition		High		trivial	491		Approved proposal: Add the missing the null stmt definition from was in the P1364 specification.
83		Entity-1	Approve	83	32.40		Technical	32.40 diffs here vs the include file		High		minor	494		Approved proposal in 494.htm
84		Entity-1	Approve	84	32.40, 32.41, 32.42		Technical	32.40, 32.41, and 32.42 all supersede P1364 26.6.27		High		minor	495		Approved proposal in 495.htm
85		Entity-1	Approve	85	32.43		Technical	32.43 No explanation for a regular assignment		High		minor	496		Approved proposal in 496.pdf
86		Entity-1	Approve	86	32.45		Technical	32.45 typespec reference should be a expr reference - regression		High		minor	497		Approved proposal: Replace typespec (on right of vpiCondition) with expr (also italics).
87		Entity-1	Approve	87	32.47		Technical	32.47 Implies all expressions have a vpiName property		High		minor	500		Approved proposal: Move expr to bottom of pattern. Add vpiName as property to each of the rest of the things in pattern.
88		Entity-1	Approve	88	32.51		Technical	32.51 vpiExpr should apply to entire disables class		High		minor	502		Approved proposal: Make the 1-to-1 transition labeled vpiExpr apply to the entire disables class.
89		Entity-1	Approve	89	32.52		Technical	32.52 'return stmt' should be 'return'		High		trivial	503		Approved proposal: Change 'return stmt' to 'return'.
90		Entity-1	Approve	90	32.53		Technical	32.53 Missing info from P1364 26.6.42		High		trivial	504		Approved proposal: Add from the P1364 section 26.6.42, a 1-to-1 transition from the attribute to the unnamed class on the right hand side of the diagram. It is labeled vpiParent.
91		Entity-1	Approve	91	22.2		Technical	package parameters		High	Resolved	minor	506		Approved proposal in 506.htm
92		Entity-1	Approve	92	24.5		Technical	$isunbounded definition	Same as Mantis 509	High		minor	508		Approved proposal in 508.htm
93		Entity-1	Approve	93	8		Technical	what are the operators allowed on unpacked structs, unions and classes?		High	Resolved	major	510		Approved propsoal in 510.pdf
102		Entity-1	Approve	102	32.13, 32.14		Technical	32.13 & 32.14 vpiArrayMember and vpiMember		High		minor	524		Approved proposal in 382.pdf
103		Entity-1	Approve	103	32.21		Technical	Since a type declaration can appear as part of a class definition, the VPI object model should permit iterating over vpiTypedef from a class defn to return the included typespecs. That iteration should appear either here or in "32.9 Scope".	Add the iteration	High		major	528		Approved proposal in 528.pdf
104		Entity-1	Approve	104	E.8.4, E.10.1, E.11.1, E.11.8, E.11.9		Editorial	This is a DPI-C example and should use the DPI-C label.	Replace "DPI" by "DPI-C" in the example.	High		trivial	530		Approved proposal: Update examples to use DPI-C label in the following sections: E.8.4 E.10.1 E.11.1 E.11.8 E.11.9
105		Entity-1	Approve	105	Annex I		Technical	Note 4 of 31.41 refers to a constant type of vpiUnboundedConst, but Annex I does not declare it.	Add declaration of vpiUnboundedConst to Annex I.	High		minor	531		Approved proposal in 531.htm
106		Entity-1	Approve	106	A.2.1.3		Technical	The BNF for data_declaration is not consistent with that in "6.2 Data declaration syntax".	Pick one or the other	High		minor	532		Minor editing issue implemented by the editor.
107		Entity-1	Approve	107	32.26		Technical	The properties vpiInterfaceTask and vpiInterfaceFunction that Note 2 refers to do not appear either in the diagram or in "Annex I: sv_vpi_user.h".	Either remove Note 2 and renumber the remaining notes, or define the vpiInterfaceFunction and vpiInterfaceTask properites on the diagram.	High		minor	534		Approved proposal in 534.pdf
108		Entity-1	Approve	108	E.6.2		Technical	If packed struct and union types cannot be used as DPI types, "E.6.2 Data representation" should say so explicitly, rather than simply saying that "Packed types are represented using the canonical format defined in E.6.7." A first read of this statement implies that all packed types are allowed. However, the canonical format of E.6.7 does not include packed structures or unions, and no examples in Annex E show packed structures used as arguments.	Either add packed types as legal DPI-C argument values, or clarify the statement to exclude them explicitly.	High		major	536		Approved proposal in 536.htm

IEEE P1800 First Recirculation Resolved Comments

109		Entity-1	Approve	109	12.2		Technical	The syntax expansions for class_item and class_property imply that any kind of data_declaration can appear as a class_item. However, data_declaration includes not only a "list_of_variable_decl_assignments" but also "type_declaration", "package_import_declaration", and "virtual_interface_declaration". "type_declaration" is already listed separately as a class_item. Can the other two also actually appear in a class definition?	Simplify and clarify syntax.	High		minor	537		Approved proposal in 537.htm
110		Entity-1	Approve	110	32.47		Technical	32.47 two tagged pattern objects in pattern clas		High		minor	547		Approved proposal in 547.pdf
118		Entity-1	Approve	118	31.11		Editorial	31.11 vpi_get_time() is a void routine - backwards compatibility		High		trivial	591		Approved proposal: Change the description of the routine vpi_get_time() to be a void routine.
121		Entity-1	Approve	121	32.30		Technical	32.30 Problems with data model for clocking blocks		High		major	606		Approved proposal in 606.htm
124		Entity-1	Approve	124	32.49		Technical	No VPI access defined for enhanced for loops		Low		minor	257		Approved proposal in 257.pdf
125		Entity-1	Approve	125			Technical	Waiting process as task or function handle		Low		minor	294		Approved proposal in 528.pdf
127		Entity-1	Approve	127			Technical	SystemVerilog VPI sections do not deal with generates		Low		major	369		Approved proposal in 369.pdf
130		Entity-1	Approve	130	30.1.3		Technical	30.1.3 Last sentence is not clear		Low		major	434		Approved proposal in 434.pdf
131		Entity-1	Approve	131	30.2.2.1		Technical	30.2.2.1 Phrase " is ambiguous		Low		major	435		Approved proposal in 435.pdf
133		Entity-1	Approve	133	32.2		Technical	32.2 The note needs to go in section 32.7		Low		trivial	443		Approved proposal: The note should be moved to 32.7, note 4, incrementing the number of subsequent notes.
134		Entity-1	Approve	134	32.3, 32.6		Technical	32.3 & 32.6 Superfluous note should be removed		Low		trivial	444		Approved proposal: remove the two notes
137		Entity-1	Approve	137	32.11		Technical	32.11 incorrect format for the 2nd note		Low		feature	454		Approved proposal in 454.pdf
138		Entity-1	Approve	138	32.12.1		Technical	32.12.1 poorly written paragraphs		Low		minor	455		Approved proposal in 489.pdf
139		Entity-1	Approve	139	32.13		Technical	32.13 Question on the range relation		Low		minor	458		Approved proposal in 458.pdf
141		Entity-1	Approve	141	32.30		Technical	32.30 indentation for properties is incorrect		Low		trivial	461		Approved proposal: corrected the indentation
142		Entity-1	Approve	142	32.14		Technical	32.14 vpiParent label is poorly placed		Low		trivial	462		Approved proposal: draw the line above the label, then down. This would allow the label to be moved further to the left, and would not allow for the possibility that it applies to the transition above it.
143		Entity-1	Approve	143	32.19		Technical	32.19 Please clarify vpiWaitingProcesses iteration		Low		major	465		Approved proposal in 465.pdf
144		Entity-1	Approve	144	32.21		Technical	32.21 Properties should be with defn not ref		Low		trivial	470		Approved proposal in 470.pdf
145		Entity-1	Approve	145	32.21		Technical	32.21 typo in note 6		Low		trivial	471		Approved proposal: Change Note 6 from: ... whenever a one class ... to ... whenever one class ...
146		Entity-1	Approve	146	32.22		Technical	32.22 Note 1 needs clarification		Low		minor	472		Approved proposal in 528.pdf
147		Entity-1	Approve	147	32.22		Technical	32.22 Note 3 needs work		Low		trivial	473		Approved proposal in 473.pdf
149		Entity-1	Approve	149	32.26		Technical	32.26 property type is missing	Is a duplicate of 602	Low		trivial	478		Approved proposal in 478.pdf
150		Entity-1	Approve	150	32.26		Technical	32.26 Note 1 needs clarification		Low		minor	481		Approved proposal in 481.pdf
151		Entity-1	Approve	151	32.29		Technical	32.29 Clarify difference between a thread and a frame		Low		minor	485		Approved proposal in 485a.htm and 485b.pdf
													485b		
152		Entity-1	Approve	152	6.9		Technical	definition of the scope of a type in section 6.9		Low	Resolved	minor	493		Approved proposal in 493.htm
153		Entity-1	Approve	153	32.50		Technical	32.50 minor clarification needed here		Low		trivial	498		Approved proposal: draw the instance class reference to the right or above the alias stmt object definition.
154		Entity-1	Approve	154	6.9.2		Technical	rule 7 section 6.9.2 could be written in a simple way		Low	Resolved	tweak	499		Approved proposal in 499.htm
155		Entity-1	Approve	155	22.1		Editorial	section 22.1 typo		Low	Resolved	trivial	501		Approved proposal in 501.htm
156		Entity-1	Approve	156	28.2		Technical	Verilog compatibility and interoperability		Low	Resolved	tweak	505		Approved proposal in 505.pdf
161		Entity-1	Approve	161			Technical	specparams allowed in program blocks		Low		tweak	550		Approved proposal in 550.htm
163		Entity-1	Approve	163			Technical	Definition of program variable		Low		tweak	554		Approved proposal in 554.htm
183		Entity-1	Approve	183	31.8.6		Editorial	31.8.6 font issue		Low		minor	585		Approved proposal in 585.pdf
184		Entity-1	Approve	184	31.8.7		Technical	31.8.7 sentence need clarification		Low		minor	586		Approved proposal in 586.pdf
188		Entity-1	Approve	188	15.4		Technical	16.4 Invalid example		Low		minor	595		Approved proposal in 595.htm
189		Entity-1	Approve	189	16.12		Technical	16.12 Input sampling description makes no sense		Low		minor	596		Approved proposal in 596.htm
191		Entity-1	Approve	191	32.22		Technical	Clarify meaning of vpiClassDefn		Low		minor	598		Approved proposal in 473.pdf
192		Entity-1	Approve	192	32.21		Technical	Clarify vpiActualDefn		Low		minor	599		Approved proposal in 473.pdf
193		Entity-1	Approve	193	32.22		Technical	Confusing statement about vpiActualDefn		Low		minor	600		Approved proposal in 473.pdf
194		Entity-1	Approve	194	32.22		Editorial	class defn rather than "Class Defn"		Low		trivial	601		Approved proposal in 473.pdf

195		Entity-1	Approve	195	32.26		Technical	vpiVisibility rather than "vpiAcess" for task and function declaration	Made 478 a duplicate of this one	Low		feature	602		Approved proposal in 602.pdf
197		Entity-1	Approve	197	Annex E		Technical	Appendix E: routines are not always declared in instantiable scopes		Low		minor	604		Approved proposal in 604.htm
198		Entity-1	Approve	198	Annex I		Technical	32.2 & 32.3 & 32.6 vpiDefaultClocking is not in include file		Low		minor	605		Approved proposal in 605.pdf
199		Entity-1	Approve	199	4.3.3		Technical	4.3.3 Accessing values of clocking variables		Low		major	607		Approved proposal in 607.htm
202		Entity-2	Disapprove	1			Technical	"bind" statement needs clarification according to the place the bound module, interface, program is bound to. As an example: There is a program p with a type pt. Another program jap is bound to that program p, and uses the declared type pt. This is only valid, if the type declaration is analysed first, i.e. the jap program is "inserted" at the end of program p. Similarly, it must be specified, how a set of bind statements is analysed.			Resolved		627		Approved proposal in 627.htm Regarding additional information provided by entity, on 4/25/2005 the SV-BC unanimously approved the following response: "Bind only creates an instance, it does not create a nested module. Clause 18.15 describes bind as creating a module, interface or program instance and makes no mention c nesting modules. Refer to the following thread of the SV-BC email reflector for alternative modeling styles: http://www.eda.org/sv-bc/hm/2983.html"
													Email		
209		Entity-3	Approve	2	18.7.3	Paragraph	Technical	The example is illegal:always @(posedge clk) assert (done |=> (out == $past(q, 2,enable)) ; since: immediate_assert_statement ::= assert (expression) action_block and |=> is not an expression, it is a property expression!!		High			663		Approved proposal in 663.htm
213		Entity-3	Approve	6	18.13.1 assert statement	Paragraph	Editorial	missing semi coloumn after pass_stat property abc(a,b,c); disable iff (a==2) @(posedge clk) not (b ##1 c); endproperty env_prop: assert property (abc(rst,in1,in2)) pass_stat else fail_stat;	env_prop: assert property (abc(rst,in1,in2)) pass_stat; else fail_stat;	Low			664		Approved proposal in 664.htm
214		Entity-3	Approve	7	18.13.2 assume statement	Paragraph	Editorial	a1:assume property @(posedge clk) req dist {0:=40, 1:=60} ; property proto @(posedge clk) req |-> req[*1:$] ##0 ack; endproperty This is equivalent to: a1_assertion:assert property req inside {0, 1} ; property proto_assertion @(posedge clk) req endproperty missing semi coloumn after proto and proto_assertion missing brackets around req dist {0:=40, 1:=60} and req inside {0, 1}	a1:assume property @(posedge clk) (req dist {0:=40, 1:=60}) ; property proto; @(posedge clk) req |-> req[*1:$] ##0 ack; endproperty This is equivalent to: a1_assertion:assert property (req inside {0, 1}) ; property proto_assertion; @(posedge clk) req \|-> req[*1:$] ##0 ack; endproperty	Low			665		Approved proposal in 665.htm

223		Entity-3	Approve	16	22.2	Paragraph	Technical	Make "localparam type" legal. Right now, you can declare a "parameter type" but a localparam as per the BNF cannot be a type. This restriction doesn't really make any sense. Example of requested capability: begin localparam type out_type = $typeof(out); ... There's actually a conflict of the wording in section 6.3 and the BNF in Section 22.2. Section 6.3 indicates the existence of localparams but the BNF in section 22.2 specifically does not allow for localparams to be assigned data types. localparams should be acceptable data types parameters	Enable localparam type like parameter type. Change Section 22.2: local_parameter_declaration ::= localparam data_type_or_implicit list_of_param_assignments | localparam type list_of_type_assignments ; // from A.2.1.1 A.2.1.1 Module parameter declarations local_parameter_declaration ::= localparam data_type_or_implicit list_of_param_assignments | localparam type list_of_type_assignments;	High	Resolved		625		Approved proposal in 625.htm
224		Entity-3	Approve	17	24.2	Paragraph	Technical	A $typeof should be useable anywhere a datatype is useable. $typeof is currently limited to the following places: to assign or override a type parameter, or in a comparison with another $typeof, evaluated during elaboration. As a result, today you can only use $typeof by copying its value into a parameter type such as: parameter type out_type = $typeof(out); and then you can do things like out = out_type'(out_packed); Examples of requested capability: out = $typeof(foo)'(out_packed) and typedef $typeof(foo) my_type Additional example, today we are force to do: parameter type foo = $typeof(bar); foo barbar; assign barbar = bar; This is inconvenient. Why not simply allow: $typeof(bar) barbar;	Change: The $typeof system function returns a type derived from its argument. The data type returned by the $typeof system function may be used to assign or override a type parameter, or in a comparison with another $typeof, evaluated during elaboration. To: The $typeof system function returns a type derived from its argument. The data type returned by the $typeof system function may be used in any context where a data type identifier is allowed, including variable or net declarations, to assign or override a type parameter, or in a comparison with another $typeof, evaluated during elaboration or	High	Resolved		619		Approved proposal in 619.htm
226		Entity-3	Approve	19	8.13	Paragraph	Technical	Assignment patterns are not allowed on module outputs. This is very inconvient and counter to user expectations who consider assignment patterns to be extensions of concatenations	Enable assignment patterns on outputs using a cast to the type of the output port. This should satisfy the requirements of modular compilation. It is breaking the rule that casting is not allowed on the LHS of assignments but an exception should be permitted in this instance		Resolved		623		Approved proposal in 623.htm
227		Entity-3	Approve	20	9.4	Paragraph	Technical	Given the benefits of wild equality and wild inequality operators (Section 8.5), these benefits should be extended to case statements. Designers would like to specify don't cares in the case item expressions and would like to exclude the don't cares from the case expression	Add the selection statement: wildcase. It can be modified by unique/priority. X and Z in the case expression are not treated as don't cares.		Resolved		324		Approved proposal in 324.htm
269		Entity-7	Approve	1	14.4	Paragraph	Technical	What are the dimensions of the result of a width cast? Does the range of 5'(vec) have the same right bound as vec, or is the right bound 0, as in the result of a part-select?	In 4.4, REPLACE When changing the size, the signing passes through unchanged. When changing the signing, the size passes through unchanged. WITH When changing the size, the signing shall pass through unchanged and the result type shall be a one-dimensional packed array with a right bound of zero. When changing the signing, the type of the expression to be cast shall pass through unchanged, except for the signing.	High	Resolved		628		Approved proposal in 628.htm

270		Entity-7	Approve	2	12.11.1, 3.16	Figure	Editorial	The example in 12.11.1 (bottom of page 152) shows a std::randomize/with call as: success = std::randomize(a, b, c) with { a < b ; a + b < length }; I believe this is a syntax error, since in the constraint block each constraint has to be delimited by a ';'. The proper syntax should be success = std::randomize(a, b, c) with { a < b ; a + b < length ;}; ^^^ I'm sure this isn't the only inconsistency. I believe that syntax in A.1.9 states that the ending ';' is required before the closing '}', so it wouldn't be a bad idea to go through the entire example set and make sure they're clean. There's another example of this syntax error on p. 31 of 3.16	In 4.16, REPLACE with { p.length > 1 && p.payload.size == p.length } WITH with { p.length > 1 && p.payload.size == p.length ; } In 13.11.1, REPLACE < length } WITH < length ; } AND REPLACE > length } WITH > length ; }	High			638		Approved proposal in 638.htm
271		Entity-7	Approve	3	21.6.1	Paragraph	Technical	The default coverage goal is specified as 90% in Table 21.3, Section 21.6.1. It default goal should be 100%, since users can always set a specific goal.	21.6.1 Covergroup Type Options , Table 21.3 The default coverage goal is specified as 90. Replace 90 by 100.	High			639		Approved proposal in 639.htm
272		Entity-7	Approve	4	9.9	Paragraph	Technical	The "kill" and "suspend" methods of the built-in "process" class are incorrectly defined as tasks. These two methods are non-blocking, therefore they should be defined as void functions.	Change Section 9.9 FROM: class process; enum state { FINISHED, RUNNING, WAITING, SUSPENDED, KILLED }; static function process self(); function state status(); task kill(); task await(); task suspend(); task resume(); endclass TO: class process; enum state { FINISHED, RUNNING, WAITING, SUSPENDED, KILLED }; static function process self(); function state status(); function void kill(); task await(); function void suspend(); task resume(); endclass	High			640		Approved proposal in 640.htm

273		Entity-7	Approve	5	24.7	Paragraph	Technical	As noted in svdb erratum 304, we need $unpacked_dimensions() in addition to $dimensions().	In 5.5, REPLACE $size, and $dimensions WITH $size, $dimensions, and $unpacked_dimensions In 24.7, REPLACE If the first argument to an array dimension function WITH If the first argument to an array query function In 24-5, replace | $dimensions (array_identifier) | $dimensions (data_type) WITH | array_dimensions_function (array_identifier) | array_dimensions_function (data_type) AND after the array_query_function production ADD array_dimensions_function ::= $dimensions	High	Resolved		304		Approved proposal in 304.htm
274		Entity-7	Approve	6	8.13.2	Paragraph	Technical	In 8.13.2, the following sentence is no longer true, and is out-of-date with changes to the detailed rules near the end of the section. "In fact, it descends the nesting to a built-in type or a packed array of them." The termination condition of the recursive descent has been modified. Descent continues into packed arrays unless the type of the packed array matches the self-determined type of the default value expression.	In 8.13.2, DELETE In fact, it descends the nesting to a built-in type or a packed array of them.	High	Resolved		634		Approved proposal in 634.htm
275		Entity-7	Approve	7	8.13	Paragraph	Technical	In 8.13, rewrite the the following to remove 'input' A port connection to an input port of a module, interface, or program The passing of a value to a subroutine input port	In 8.13, DELETE the word 'input' in BOTH A port connection to an input port of a module, interface, or program AND The passing of a value to a subroutine input port	High	Resolved		692		Approved proposal in 623.htm

277		Entity-7	Approve	9	24.2	Paragraph	Technical	Section 24.2 is unclear on where $typeof() can be used. I think you should be able to use it anywhere. And, according to the LRM "In all contexts, $typeof together with its argument can be used in any place an elaboration constant is required." (It's not clear from this that it can be used at run-time.) Yet I've heard it said that the LRM "calls out" the specific places where $typeof can be used, namely, "The $typeof system function returns a type derived from its argument. The data type returned by the $typeof system function may be used to assign or override a type parameter, or in a comparison with another $typeof, evaluated during elaboration." (Again, not clear if it can be used at run-time.)	In 24.2, REPLACE The $typeof system function returns a type derived from its argument. The data type WITH The $typeof system function returns a type derived from its argument. For example, the data type AND in the first sentence of the second paragraph change $typeof into Courier AND REPLACE In all contexts, $typeof together with its argument can be used in any place an elaboration constant is required. WITH In all contexts, $typeof together with its argument can be used in any place an elaboration constant can be used.	High			619		Approved proposal in 619.htm
279		Entity-7	Approve	11	Appendix A	Paragraph	Editorial	In A.1.5 and Syntax 18-17, in bind_target_instance_list, the comma should be red. In A.2.2.1 and Syntax 4-1, in net_type, the keywords 'trireg' and 'uwire' should be red. In Syntax 6-1 and Syntax 19-4, in the footnotes, and in footnote 32 of Annex A, the keywords should be red. In footnote 29 of Annex A, the '=' and '<=' should be red. In footnote 28 of Annex A and Syntax 22-1, '$typeof' should probably be red.	In A.1.5 and Syntax 18-17, in bind_target_instance_list, the comma should be red. In A.2.2.1 and Syntax 4-1, in net_type, the keywords 'trireg' and 'uwire' should be red. In Syntax 6-1 and Syntax 19-4, in the footnotes, and in footnote 32 of Annex A, the keywords should be red. In footnote 29 of Annex A, the '=' and '<=' should be red. In footnote 28 of Annex A and Syntax 22-1, '$typeof' should probably be red.	High					Minor editing issue implemented by the editor.
281		Entity-7	Approve	13	14.3.4, 14.3.6, 14.3.8	Paragraph	Technical	Function try_get on semaphore returns specific value of 1 when the number of keys are available. The return number should be any positive number. Similarly, - Mailbox function try_put should return any positive number instead of 1 when the mailbox is not empty - Mailbox function try_get should return any positive number instead of 1 when the mailbox has a matching entry and any negative number when there is no matching entry - Mailbox function try_peek should return any positive number instead of 1 when the mailbox has a matching entry and any negative number when there is no matching entry	Section 14.2.4 try_get() , Section 14.3.4 try_put() , Section 14.3.6 try_get() and Section 14.3.8 try_peek() Replace "1" by "a positive integer" Replace "-1" by "a negative integer"	High			683		Approved proposal in 683.htm
282		Entity-7	Approve	14	Syntax 6-1	Figure	Editorial	In Syntax 6-1, data_declaration needs to be updated to excerpt the current definition in A.2.1.3.	In Syntax 6-1, data_declaration needs to be updated to excerpt the current definition in A.2.1.3.	High					Minor editing issue implemented by the editor.

IEEE P1800 First Recirculation Resolved Comments

283		Entity-7	Approve	15	Appendix A and various syntax boxes	Paragraph	Editorial	SV3.1a eliminated the restriction that dynamic arrays, queues and associative arrays be limited to one dimension, but the BNF was never updated to reflect that.	Section A.1.4, A.2.1.3, A.2.3, A.2.4, A.2.5, A.2.7, A.10, Syntax 4-1, Syntax 4-2, Syntax 5-2, Syntax 11-1 Variable dimension In A.2.5 and Syntax 5-2, REPLACE variable_dimension12 ::= { sized_or_unsized_dimension } | associative_dimension | queue_dimension WITH variable_dimension12 ::= unsized_dimension {sized_or_unsized_dimension } | unpacked_dimension | associative_dimension | queue_dimension In A.2.5, REPLACE unsized_dimension11 ::= [ ]	High			615		Approved proposal in 615.htm
287		Entity-8	Approve	3	6.7	Paragraph	Editorial	In Paragraph 6 (Page 63) of Section 6.7 the LRM says: "This makes procedural or continuous assignments to a variable connected to the output port of an instance illegal." We suggest add an extra word additional to make it clear	This makes additional procedural or continuous asignments to a variable connected to the output port of an instance illegal	Low	Resolved		689		Approved proposal in 689.htm
38		Entity-1	Approve	38	27.2 & 27.3		Editorial	Reprhase first sentence		High		Minor	419		Minor editing issue implemented by the editor.
39		Entity-1	Approve	39	27.3		Editorial	Awkward first sentence in last paragraph		High		Minor	420		Minor editing issue implemented by the editor.
44		Entity-1	Approve	44	29.4.2		Editorial	29.4.2 needs better whitespace		High		trivial	428		Minor editing issue implemented by the editor.
82		Entity-1	Approve	82	6.9.1		Technical	rule 7 of definition of matching types		High		minor	492		Minor editing issue implemented by the editor.
157		Entity-1	Approve	157	22.2		Editorial	typo in section 22.2		Low		trivial	507		Minor editing issue implemented by the editor.
158		Entity-1	Approve	158	32.13		Editorial	Grammatical error	In Note 25, replace "object which typespec" by "object for which the typespec".	Low		trivial	525		Minor editing issue implemented by the editor.
159		Entity-1	Approve	159	32.14		Editorial	Grammatical error	In Note 20, replace "object that typespec" by "object for which the typespec".	Low		trivial	527		Minor editing issue implemented by the editor.
160		Entity-1	Approve	160	32.26		Editorial	The font sizes in the notes are inconsistent.	Make the font sizes in the notes consistent.	Low		trivial	535		Minor editing issue implemented by the editor.
173		Entity-1	Approve	173	31.6		Editorial	31.6 Why is there a double line between vpi_iterate() and vpi_handle()?		Low		trivial	566		Minor editing issue implemented by the editor.
175		Entity-1	Approve	175	31.8.1		Editorial	31.8.1 typos		Low		trivial	569		Minor editing issue implemented by the editor.
179		Entity-1	Approve	179	31.8.3.2		Editorial	31.8.3.2 Typo		Low		trivial	574		Minor editing issue implemented by the editor.
181		Entity-1	Approve	181	31.8.4.2		Editorial	31.8.4.2 grammar		Low		trivial	582		Minor editing issue implemented by the editor.
182		Entity-1	Approve	182	31.8.4.2		Technical	31.8.4.2 terminology first defined in a footnote?		Low		minor	583		Minor editing issue implemented by the editor.
196		Entity-1	Approve	196	Annex E		Editorial	callee's declared the formals?		Low		trivial	603		Minor editing issue implemented by the editor.
208		Entity-3	Approve	1	18.2	Paragraph	Editorial	missing semi coloumn in example: always @(posedge clk) if (state == REQ) assert (req1 || req2) else begin t = $time; #5 $error("assert failed at time %0t",t); end	always @(posedge clk) if (state == REQ) assert (req1 || req2); else begin t = $time; #5 $error("assert failed at time %0t",t); end	Low					Minor editing issue implemented by the editor.

276		Entity-7	Approve	8	All	Paragraph	Editorial	Throughout, the apostrophes on '{...} are apparently instead single quotation marks. They should be straight, not tilted. For an illustrative example, see the second example on page 79 initial #10 s1 = '{default:'1, s:""};'	Use straight apostrophes	High					Minor editing issue implemented by the editor.
SCC14	John Scott	john.t.scott@standards.ieee.org	1	General				There is essentially nothing in this standard to attract the attention of SCC14. However, it would be nice if an informative note could be attached to Subclause 3.5, where time units are first introduced, reading something like this: "While s, ms, ns, ps and fs are the usual SI unit symbols for second, millisecond, nanosecond, picosecond and femtosecond, 'us' represents the SI unit symbol for microsecond, properly {mu}s." [Where {mu} is the Greek letter.]							There is no formal definition of the time units that exist within the P1800 standard. Instead, the formal definition of time units referred to in the P1800 standard come from the P1364 standard. As a result, we believe it is not appropriate to address the definition of these units within P1800. We have added language addressing this issue within the P1364 standard.
Editorial Coord.	Michelle Turner	m.d.turner@ieee.org	1	General			**Boldface-red** characters are used to denote reserved keywords, operators, and punctuation marks as a required part of the syntax. For example: module => ; — A vertical bar (|) separates alternative items, unless it appears in boldface-red, in which case it stands for itself. For example: unary_oper	**I copied the text from draft. This must be changed throughout the draft. Upon publication text and figures are printed in black and white.**							
			2	General			If figures and tables were derived or obtained from sources other than the Working Group itself, please obtain and supply permission from the appropriate sources							The IEEE does allow color when it enhances readability of the standard. We believe it is true in this case.

			3	General			At the time of RevCom submittal, please remember to supply the names and email/address of all working group members.								No LRM action required.
		Non-voting entity 1	1				I am concerned about one issue in the definition of P1800, where I think the current definition is somewhat broken, and discuss how it can be fixed. This is the issue of the behavior at time 0 of initial, always, and always_comb constructs, variable initializations, combinational UDPs, continuous	Proposed Prinicples of Solution: 1. The solution needs to cover: always_comb, combinational always @*, combinational always @(), non-collapsed port connections, continuous assignments, combinational UDPs. 2. It is proposed to treat all of the above as identically as possible for simplicity and consistency. 3. Since the value being assigned can be a simple constant (or parameter) where surely no event is generated, the constructs must unconditionally evaluate at time 0 even without an event. An exception is the simple always, for which we can only activate it, but not unconditionally execute its entire body. 4. The constructs must evaluate at time-0 AFTER any assignments which do not generate events. Variable initializers currently behave that way. However, regular always constructs have a problem with inputs whose initial assignments do not create events. 5. It is accepted that variable initializers execute before initial constructs. No one has argued against that.						This issue is a duplicate of STU1. Please see the WG response for that item.

Clause: 18.3

Description

The second paragraph of clause 18.3 states "The values of variables used in assertions are sampled in the Preponed region of a time slot and the assertions are evaluated during the Observe region. This is explained in Clause 15, Scheduling Semantics".

There is no mention of sampling based on clocking block if the signal used in the assertion is a clocking block signal. As per clause 15.3 "The sampling time of sampled data for property expressions is controlled in the clocking block". It is not very clear from these clauses how the sampling should be done for clocking block variables which are used in the concurrent assertion expression. Do we still continue and sample clocking block variables in the Preponed region irrespective of how the clocking block is sampling? Or are these variables are sampled based on the clocking block declaration ?

Suggested Resolution

The second paragraph of clause 18.3, replace: ~~The values of variables used in assertions are sampled in the Preponed region of a time slot and the assertions are evaluated during the Observe region. This is explained in Clause 15, Scheduling Semantics~~

By

The values of variables used in assertions are sampled in the Preponed region of a time slot and the assertions are evaluated during the Observe region. If a variable used in an assertion is a clocking block input variable, the variable must be sampled by the clocking block with #1step sampling. Any other type of sampling for the clocking block variable shall result in an error. The assertion using the clocking block variable shall not do its own sampling on the variable, but rather use the sampled value produced by the clocking block. This is explained in Clause 15, Scheduling Semantics.

Clause 15.3, remove ~~The sampling time of sampled data for property expressions is controlled in the clocking block~~

Add a new clause 18.17 to explain effect of clocking blocks on assertions.

18.17 Clocking blocks and Concurrent Assertions

If a variable used in a concurrent assertion is a clocking block variable it will be sampled only in the clocking block.

Examples:

```
module A;
 logic a, clk;

clocking cb_with_input @(posedge clk);
  input a;

  property p1;
    a;
  endproperty
```

```
endclocking

clocking cb_without_input @(posedge clk);
  property p1;
    a;
  endproperty

endclocking


property p1;

  @(posedge clk) a;

endproperty

property p2;

  @(posedge clk) cb_with_input.a;

endproperty


a1: assert property (p1);

a2: assert property (cb_with_input.p1);

a3: assert property (p2);

a4: assert property (cb_without_input.p1);

endmodule
```

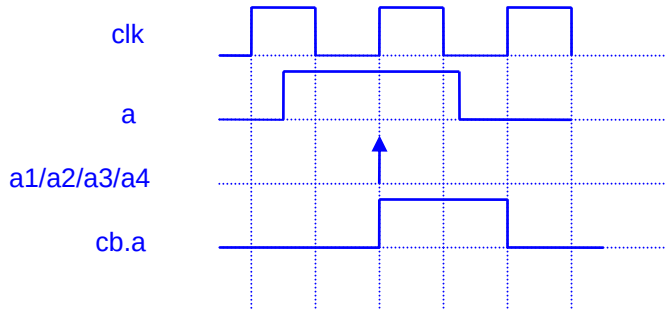


Figure 18-7 - Clocking blocks and Concurrent Assertion

Figure 18-17 explains the behavior of all the assertions. In the above example, a1, a2, a3 and a4 are equivalent.

Section **11.4.3**
Default argument values

In 11.4.3 the fourth paragraph, REPLACE

> The *default_value* is an expression. The expression is evaluated in the scope of the caller each time the subroutine is called. The elements of the expression must be visible at the scope of subroutine and, if used, at the scope of the caller. If the default_value is not used, the expression is not evaluated and need not be visible at the scope of the caller.

WITH

> The *default_value* is an expression. The expression is evaluated in the scope containing the subroutine declaration each time a call using the default is made. If the default_value is not used, the expression is not evaluated.

Replace the paragraph in clause 4.11

~~Clause 3.8 discusses assigning initial values to a structure.~~

With the following paragraphs:

Members of a structure data type may be assigned individual default member values by using an initial assignment with the declaration of each member. The assigned expression must be a constant expression.

An example of initializing member of a structure type is:

```
typedef struct {
    int addr = 1 + const;
    int crc;
    byte data [4] = '{4'{1}};
} packet1;
```

The structure can then be instantiated.

```
packet1 p1; //initialization defined by the typedef.
            //p1.crc will use the default value for an int
```

If an explicit initial value expression is used with the declaration of a variable, the initial assignment expression within the structure data type shall be ignored. Clause 3.8 discusses assigning initial values to a structure. For example:

```
packet1 pi = '{1,2,'{2,3,4,5}}; //suppresses the typedef initialization
```

Members of unpacked structures containing a union as well as members of packed structures shall not be assigned individual default member values.

The initial assignment expression within a data type shall be ignored when using a data type to declare a net (See 6.5).

Add a bullet to the end of clause 13.3:

– An unpacked structure can be declared **rand** in which case all of that structure's random members are solved concurrently using one of the rules listed in this section. Unpacked structures shall not be declared **randc**. A member of a unpacked structure can be made random by having a **rand** or **randc** modifier in the declaration of its type. Members of unpacked structures containing a union as well as members of packed structures shall not be allowed to have a random modifier.

For example:

```
class packet;
typedef struct {
    randc int addr = 1 + const;
```

```
    int crc;
    rand byte data [] = {1,2,3,4};
} header;
rand header h1;
endclass
packet p1=new;
```

Change the BNF in A.2.2.1:

struct_union_member27 ::=
{ attribute_instance } data_type_or_void list_of_member_identifiers **;**

To:

struct_union_member[27] ::=
{ attribute_instance } [random_qualifier] data_type_or_void list_of_variable_decl_assignments **;**

Remove the production in A.2.3:

list_of_member_identifiers ::=
member_identifier variable_dimension { **,** member_identifier variable_dimension }

Change A.1.8 from:

property_qualifier7 ::=
rand
| **randc**
| class_item_qualifier

To:

property_qualifier[7] ::=
random_qualifier
| class_item_qualifier

random_qualifier[7] ::=
rand
| **randc**

Update the syntax box in 4.1 & 4.4

Replace the sentence in clause 6.6

~~Note automatic or dynamic variables cannot be written with nonblocking or continuous assignments. Automatic variables and dynamic constructs—objects handles, dynamic arrays, associative arrays, strings, and event variables—shall be limited to the procedural context.~~

to:

Automatic variables and members or elements of dynamic variables—class properties and dynamically-sized variables—shall not be written with nonblocking, continuous or procedural continuous assignments. References to automatic variables and elements or members of dynamic variables shall be limited to procedural blocks.

Section **6.9.2**
Equivalence

In 6.9.2, in rule 5, REPLACE

Packed arrays, packed structures, and built-in integral types are equivalent

WITH

Packed arrays, packed structures, packed unions, and built-in integral types are equivalent

Change in section 6.6

The lifetime of a scope enclosing any **fork…join** block includes the lifetime of the **fork…join** block.

To:

The lifetime of a scope enclosing any **fork** block includes the lifetime of the **fork** block.

Proposed change in 10.6:

Change:

Automatic variables declared in the scope of the fork...join block shall be initialized to the initialization value whenever execution enters their scope, and before any processes are spawned.

To:

Variables declared in the *block_item_declaration* of a **`fork...join`**/**`join_any`**/**`join_none`** block shall be initialized to their initialization value expression whenever execution enters their scope, and before any processes are spawned. Within a **`fork...join_any`** or **`fork...join_none`** block, it shall be illegal to refer to formal arguments passed by reference other than in the initialization value expressions of variables declared in a *block_item_declaration* of the fork.

Add to the end of section 8.17

When applied to a class handle (i.e., an object) the streaming operator shall stream the contents of the object, and not the handle itself. Class items are streamed in declaration order; extended class items shall follow the items of their super-class. Embedded class handles are streamed as other aggregate types: they are recursively traversed in depth-first order until reaching integral types. A null class handle shall be ignored (not streamed), and a warning may be issued. Null handles are skipped by both the pack and unpack operators; therefore, the unpack operation shall not create class objects. If a particular object hierarchy is to be reconstructed from a stream, the object hierarchy into which the stream is unpacked must be created before the streaming operator is applied. The result of streaming an object hierarchy that contains cycles shall be undefined, and an error may be issued.

Change in section 11.4.2

To indicate argument passing by reference, the argument declaration is preceded by the **ref** keyword.

To:

To indicate argument passing by reference, the argument declaration is preceded by the **ref** keyword. It shall be illegal to use argument passing by reference for subroutines with a lifetime of **static**.

Section **A.8.4**
'this' as primary

In A.8.4, in primary, ADD

with the keyword 'this' in red.

Section **16.3 Input and output skews**

In 16.3 Add the note in blue after the following paragraph

An input skew of 1step indicates that the signal is to be sampled at the end of the previous time step. That is, the value sampled is always the signal's last value immediately before the corresponding clock edge.

Note: A clocking block does not eliminate potential races when an event control outside of a program block is sensitive to the same clock as the clocking block, and a statement after the event control attempts to read a member of the clocking block. The race is between reading the old sampled value and the new sampled value.

.

Section **17.2, A.1.7, Syntax 17-1 final block in programs**

In A.1.7 and Syntax 17-1 in non_port_program_item, REPLACE

| { attribute_instance } initial_construct

WITH

| { attribute_instance } initial_construct

| { attribute_instance } final_construct

In 17.2, REPLACE

A program block can contain one or more **initial** blocks.

WITH

A program block can contain one or more **initial** or **final** blocks.

Section **19.2**
Implicit continuous assignments in packages

In 19.2, REPLACE

> Variable declaration assignments within the package shall occur before any **initial**, **always**, **always_comb**, **always_latch** or **always_ff** blocks are started, in the same way as variables declared in a compilation unit or module.

WITH

Variable declaration assignments within the package shall occur before any **initial**, **always**, **always_comb**, **always_latch** or **always_ff** blocks are started, in the same way as variables declared in a compilation unit or module.

Packages must not contain any processes, therefore wire declarations with implicit continuous assignments are not allowed.

Clause 19.3, Clarifications on definition of compilation unit

PROPOSAL

In clause 19.3, MODIFY the text as follows:

DELETE:

~~The exact mechanism for defining which files constitute a compilation unit is tool specific. Two extreme cases are:~~

~~1) All files make a single compilation unit (in which case the declarations in the compilation-unit scope are accessible anywhere within the design)~~

~~2) Each file is a separate compilation unit (in which case the declarations in each compilation-unit scope are accessible only within its corresponding file)~~

REPLACE WITH:

The exact mechanism for defining which files constitute a compilation unit is tool specific. However, compliant implementations shall have the same default behavior, and tools shall provide use models that allow both of the following cases:

1) All files on a given compilation command line make a single compilation unit (in which case the declarations within those files are accessible anywhere else within the constructs defined within those files).

2) Each file is a separate compilation unit (in which case the declarations in each compilation-unit scope are accessible only within its corresponding file)

DELETE:

~~The default is that each file is a separate compilation unit.~~

REPLACE WITH:

The default definition of a compilation unit is defined in case 2) above, in which each file is a separate compilation unit.

DELETE:

~~A tool must also provide a mechanism (such as a command line switch) that specifies that all of the files compiled together are a single compilation unit.~~ (Since this is now enumerated as item 2) in the list above)

Section **20.2**
Hierarchical reference in interface port connection

At the end of 20.2, ADD

If the actual of an interface port connection is a hierarchical reference to an interface or a modport of a hierarchically referenced interface, the hierarchical reference shall refer to an interface instance and shall not resolve through an arrayed instance or a generate block.

In section 20.2

Add:

Interfaces can be declared and instantiated in modules (either flat or hierarchical) but modules can neither be declared nor instantiated in interfaces.

Verilog does not permit a **defparam** statement within an array of instances to modify a parameter outside the hierarchy of the instance defining the **defparam** (Verilog 12.2.1). In SystemVerilog, a **defparam** within an instance whose port actuals refer to an arrayed interface shall be subject to the same restriction: a **defparam** shall not modify a parameter outside the hierarchy of such an instance.

The simplest use of an interface is to bundle wires, as is illustrated in the examples below.

Section **29.4, A.2.9, Syntax 20-1, Syntax 20-2**
Modports

In A.2.9, Syntax 20-1, and Syntax 20-2, REPLACE

```
modport_ports_declaration ::=
        { attribute_instance } modport_simple_port_declaration
|       { attribute_instance } modport_hierarchical_ports_declaration
|       { attribute_instance } modport_tf_ports_declaration
|       { attribute_instance } modport_clocking_declaration
```

WITH

```
modport_ports_declaration ::=
        { attribute_instance } modport_simple_port_declaration
|       { attribute_instance } modport_hierarchical_ports_declaration
|       { attribute_instance } modport_tf_ports_declaration
|       { attribute_instance } modport_clocking_declaration
```

In A.2.9 and Syntax 20-1, DELETE

```
modport_hierarchical_ports_declaration ::=
        interface_instance_identifier [ [ constant_expression ] ] . modport_identifier
```

In 20.4, DELETE

In a hierarchically nested interface, the directions in a **modport** declaration can themselves be **modport** plus name.

```
interface i1;
    interface i3;
        wire a, b, c, d;
        modport master (input a, b, output c, d);
        modport slave (output a, b, input c, d);
    endinterface
    i3 ch1(), ch2();
    modport master2 (ch1.master, ch2.master);
endinterface
```

AND DELETE

No hierarchical references shall be permitted within a modport.

Section

AND DELETE

```
illegal_i ch1(), ch2();
modport master2 (ch1.master, ch2.master);
```

Section **20.6.2, 20.6.3**
Task prototypes

In 20.6.2, REPLACE

```
modport slave (input req, addr, mode, start, clk,
                   output gnt, rdy,
                   ref data,
                 import task slaveRead,
                        task slaveWrite);
```

WITH

```
modport slave (input req, addr, mode, start, clk,
                   output gnt, rdy,
                   ref data,
                 import ~~task~~ slaveRead,
                        ~~task~~ slaveWrite);
```

In 20.6.3, REPLACE

```
modport slave (input req, addr, mode, start, clk,
                   output gnt, rdy,
                   ref data,
                 export task Read,
                        task Write);
```

WITH

```
modport slave (input req, addr, mode, start, clk,
                   output gnt, rdy,
                   ref data,
                 export ~~task~~ Read,
                        ~~task~~ Write);
```

Section

Section **21.2**
Defining the coverage model: covergroup

In 21.2 the paragraph at the top of page 232 ends with the sentence:

> …The strobe option (see 21.6.1) can be used to specify that coverage points are sampled in the postponed region, thereby filtering multiple clocking events so that only one sample per time slot is taken.

APPEND the following to the end of the paragraph:

> …The **`strobe`** option only applies to the scheduling of samples triggered by a clocking event It shall have no effect on procedural calls to the built-in `sample()` method.

Section **21.6.1**
Covergroup type options

In 21.6.1 the paragraph before Syntax 21-6 REPLACE:

> Type options can be set procedurally at any time during simulation. The syntax is:

WITH:

> The **`strobe`** type option can only be set in the **`covergroup`** definition. Other type options can be set procedurally at any time during simulation. The syntax is:

Section **21.2**
Covergroups in packages

In 21.2, REPLACE

A **covergroup** can be defined in a module, program, interface, or class.

WITH

A **covergroup** can be defined in a package, module, program, interface, or class.

Section **21.4.1**
Specifying bins for transitions

At the end of this section ADD the paragraph:

> A transition bin is incremented at most once per sample. In the preceding example, on the 10th sample the transition bin **b2** is incremented only once (1 is added to the bin count).

Section **21.5**
Defining Cross Coverage

(page 333 after last example)
In the one line paragraph of in page 333, in the middle of the page,

REPLACE

No cross coverage bins shall be created for coverpoint bins that are specified as **`default`** bins.

WITH

No cross coverage bins shall be created for coverpoint bins that are specified as **`default`**, **`ignored or illegal`** bins.

Section **Table 21-1, Table 21-2, 21.9, 21.10.2 cross_auto_bin_max**

In Table 21-1, DELETE the following row

~~**cross_auto_bin_max=***number* unbounded Maximum number of automatically created cross product bins for a **cross**.~~

In Table 21-2, DELETE the following row

~~cross_auto_bin_max Yes (default for crosses) No Yes~~

In 21.9, DELETE both occurrences of

~~**int** cross_auto_bin_max ;~~

In 21.10.2, in the definition of C_i, REPLACE

B_p

WITH

$B_{~~p~~c}$

In 21.10.2, REPLACE

B_p is the number of possible auto-cross bins.

WITH

$~~B_p~~ B_c$ is the number of ~~possible~~ auto-cross bins.

In 21.10.2, DELETE

~~The term B_p represents the automatically created cross bins, as modified by the **cross_auto_bin_max** option.~~

Section

Section **A.2.1.1, Syntax 22-1**
Local parameter declaration

In A.2.1.1 and Syntax 22-1, REPLACE

local_parameter_declaration ::=
localparam data_type_or_implicit list_of_param_assignments **;**

WITH

local_parameter_declaration ::=
localparam data_type_or_implicit list_of_param_assignments **;**
| **localparam type** list_of_type_assignments **;**

Section 23.15.1, Section 23.16.3
Clarifications on $readmem/$writemem interaction with index types for associative arrays

In 28.4.5, MODIFY the text as follows:

28.4.5 Function result

An imported function declaration must explicitly specify a data type or void for the type of the function's return result. Function result types are restricted to small values. The following SystemVerilog data types are allowed for imported function results:

— **void**, **byte**, **shortint**, **int**, **longint**, **real**, **shortreal**, **chandle**, and **string**

~~— packed bit arrays up to 32 bits and all types that are eventually equivalent to packed bit arrays up to 32 bits~~

— scalar values of type **bit** and **logic**

The same restrictions apply for the result types of exported functions.

Section **E.6.3, Text**

In E.6.3 Replace

— Enumeration types are represented as the types associated with them. Enumerated names are not available on the C side of the interface.

WITH

— Enumeration types are represented ~~as the types associated with them~~ by C base types that correspond to the enumeration types' SystemVerilog base types (see Table E-1). The base type determines whether an enumeration type is considered a small value (see 28.4.5). DPI supports all the SystemVerilog enumeration base types (see 4.10 and A.2.2.1). **integer** and **time** base types are represented as 4-state packed arrays in canonical form. Enumerated names are not available on the C side of the interface.

Section **E.7.7, Text**

In E.7.7 Replace

input arguments of imported functions implemented in C shall always have a const qualifier.

input arguments, with the exception of open arrays, are passed by value or by reference, depending on the size. ‘Small’ values of formal input arguments are passed by value. The following data types are considered *small*:

— **byte**, **shortint**, **int**, **longint**, **real**, **shortreal**

— **chandle**, **string**

input arguments of other types are passed by reference.

WITH

input arguments of imported functions implemented in C shall always have a const qualifier.

input arguments, with the exception of open arrays, are passed by value or by reference, depending on the size. ‘Small’ values of formal input arguments are passed by value. The following data types are considered *small*:

— **byte**, **shortint**, **int**, **longint**, **real**, **shortreal**

— scalar **bit** and **logic**

— **chandle**, **string**

input arguments of other types are passed by reference.

Annex I
sv_vpi_user.h

p. 545 REPLACE

```
#define vpiSequenceDecl 661
#define vpiSequenceSpec 662
#define vpiActualArgExpr 663
#define vpiSequenceInst 664
```

WITH

```
#define vpiSequenceDecl 661
#define vpiSequenceSpec 662
#define vpiActualArgExpr 663
#define vpiSequenceInst 664
```

Subject: Issue #266 - Version 6 - May 4, 2005
(Version 6 is attached - Page numbers added)

Includes a few friendly amendments from various reviewers
(Arturo Salz - personal email sent 5/3/2005 - friendly amendments - see: 11.4.2, 11.5, 12.7, 13.11, 16.13)
(John Havlicek - email sent 5/4/2005 - friendly amendment - see: 18.15)
(Eduard Cerny - email to Karen Peiper sent 5/3/2005 - friendly amendment - see: 18.11.2, 18.15 (18.15 is the same comment as John Havlicek))

We have passed Issue #266 in the following committees:
SV-BC
SV-EC - (officially passed in meeting of 5/4/2005)
SV-AC - (per Surrendra Dudani (5/3/2005) & Bassam Tabbara (5/4/2005))

The SV-CC committee met today and per the sv-cc minutes of 5/4/2005, all proposed changes appear to be correct. It looks like the SV-CC committee will take an official vote on this proposal on Friday, 5/6/2005 (but so far all looks good)

The Champions Committee will have one last opportunity to make additional changes before passing the proposed resolution on to the SystemVerilog main committee.

NOTE - Thank you all for taking time to review and propose updates to this proposal. (Of course this is a non-normative expression of gratitude!) ;-)

Regards - Cliff

I have been reviewing a negative vote, Issue #266, from entity #6 on behalf of all of the SystemVerilog subcommittees. This issue is related to the informative use of note throughout the SystemVerilog LRM (negative-ballot comments shown below).

Regards - Cliff

Issue #266 Comments

In many places throughout the LRM, the usage of "Note" and "Notes" is not correct. In IEEE standards, notes are informative and not a required part of the standard. It is clear that in many places in the LRM there are notes that are both intended to be normative and are required for proper implementation of the standard.

I believe that IEEE standards also require that notes be in a separate paragraph and use a smaller font. In many places in the LRM, there are sentences (often in the middle of a paragraph) that begin with "Note..." but are not in a smaller font. It is unclear as to whether these notes are meant to be an informative notes, or an important normative fact that users and/or implementers need to observe. All usage's of the word "Note" "note," "NOTE," "Notes," "notes" and "NOTES" should be reviewed and brought into IEEE standards compliance.

Clauses that should be reviewed as to whether the terms "note" or "notes" are correctly used as only informative text include:

3.8

WAS: Note that the C-like alternative '{1, 1.0, 2, 2.0} is not allowed.

PROPOSED: The C-like alternative '{1, 1.0, 2, 2.0} for the preceding example is not allowed.

4.3.3

WAS: Note that the signed keyword is part of Verilog.

PROPOSED: The signed keyword in the preceding example is part of Verilog.

4.7.2

WAS: Note: str.putc(j, x) is semantically equivalent to str[j] = x.

PROPOSED: The putc method assignment str.putc(j, x) is semantically equivalent to str[j] = x.

4.7.3

WAS: Note: x = str.getc(j) is semantically equivalent to x = str[j].

PROPOSED: The getc method assignment x = str.getc(j) is semantically equivalent to x = str[j].

4.9 (multiple occurrences)
A type can be used before it is defined, provided it is first identified as a type by an empty **typedef**:

```
typedef foo;
foo f = 1;
typedef int foo;
```

WAS: Note that this does not apply to enumeration values, which must be defined before they are used.
PROPOSED: An empty typedef shall not be allowed with enumeration values. Enumeration values must be defined before they are used.

Sometimes a user-defined type needs to be declared before the contents of the type has been defined. This is of use with user-defined types derived from **enum**, **struct**, **union**, and **class**. For an example, see 12.24. Support for this is provided by the following forms for **typedef**: ...

WAS: Note that, while this is useful for coupled definitions of classes as shown in 12.24, it cannot be used for coupled definitions of structures, since structures are statically declared and there is no support for pointers to structures.
PROPOSED: While an empty user-defined type declaration is useful for coupled definitions of classes as shown in 12.24, it cannot be used for coupled definitions of structures, since structures are statically declared and there is no support for pointers to structures.

4.11
The text preceding this note is about packed unions (?? should "normal" union and "normal" structures both be changed to "unpacked"??)

WAS: Note that writing one member and reading another is independent of the byte ordering of the machine, unlike a normal union of normal structures, which are C-compatible and have members in ascending address order.
PROPOSED: With packed unions, writing one member and reading another is independent of the byte ordering of the machine, unlike a unpacked union of unpacked structures, which are C-compatible and have members in ascending address order.

4.13 - *first paragraph clarification and Note fixed to be normative text*

4.13 Singular and aggregate types

WAS: Data types are categorized as either *singular* or *aggregate*. A singular type shall be any data type except an unpacked structure, union, or array (see Clause 5 on arrays). An aggregate type shall be any unpacked structure, union, or array data type. A singular variable or expression represents a single value, symbols, or handle. Aggregate expressions and variables represent a set or collection of singular values. Integral types are always singular even though they can be sliced into multiple singular values.

...

Note that although a class is a type, there are no variables or expressions of class type directly, only class object handles that are singular. So classes need not be categorized in this manner ~~(see Clause 12 on classes).~~

PROPOSED: Data types are categorized as either *singular* or *aggregate*. A singular type shall be any data type except an unpacked structure, unpacked union, or unpacked array (see Clause 5 on arrays). An aggregate type shall be any unpacked structure, unpacked union, or unpacked array data type. A singular variable or expression represents a single value, symbols, or handle. Aggregate expressions and variables represent a set or collection of singular values. Integral types are always singular even though they can be sliced into multiple singular values.

...

Although a class, as described in Clause 12, is a type, there are no variables or expressions of class type directly, only class object handles that are singular. So classes need not be categorized in this manner.

4.14

WAS: Note that the **bit** data type loses X values. If these are to be preserved, the **logic** type should be used instead.

No change required - this is just an informative reminder of the behavior of the bit type, which is adequately defined elsewhere in the standard.

4.15

WAS: Note: $cast is similar to the dynamic_cast function available in C++, but, $cast allows users to check if the operation will succeed, whereas dynamic_cast always raises a C++ exception.

No change required - this is just an informative comparison to the dynamic casting capability of C++.

5.2

The preceding text in this paragraph is:

When assigning to an unpacked array, the source and target must be arrays with the same number of unpacked dimensions, and the length of each dimension must be the same. Assignment to an unpacked array is done by assigning each element of the source unpacked array to the corresponding element of the target unpacked array. The leftmost element of the source array corresponds to the leftmost element of the target array. ...

WAS: Note that an element of an unpacked array can be a packed array.

PROPOSED: Each element of an unpacked array that is assigned to the corresponding element of another unpacked array can itself be a packed array.

5.3

WAS: Note that the dimensions declared following the type and before the name ([3:0][7:0] in the preceding declaration) vary more rapidly than the dimensions following the name ([1:10] in the preceding declaration).

PROPOSED: In a multidimensional declaration, the dimensions declared following the type and before the name ([3:0][7:0] in the preceding declaration) vary more rapidly than the dimensions following the name ([1:10] in the preceding declaration).

5.6.2

WAS: ~~Note:~~ The size method is equivalent to $size(addr, 1).

PROPOSED: The size dynamic array method is equivalent to $size(addr, 1) array query system function (see Clause 24.7).

5.8
WAS: ~~Note that unsized dimensions can occur in dynamic arrays and in formal arguments of import DPI functions.~~ If one dimension of a formal is unsized, then any size of the corresponding dimension of an actual is accepted.

PROPOSED: If one dimension of a formal is unsized (unsized dimensions can occur in dynamic arrays and in formal arguments of import DPI functions) then any size of the corresponding dimension of an actual is accepted.

5.14.1 (multiple occurrences)
WAS: — An invalid index (i.e., a 4-state expression with X's or Z's, or a value that lies outside 0...$+1) shall cause a write operation to be ignored and a run-time warning to be issued. Note that writing to Q[$+1] is legal.
PROPOSED: — An invalid index (i.e., a 4-state expression with X's or Z's, or a value that lies outside 0...$+1) shall cause a write operation to be ignored and a run-time warning to be issued; however, writing to Q[$+1] is legal.

WAS:
5.14.1 Queue Operators
(1st paragraph)
Queues support the same operations that can be performed on unpacked arrays, and using the same operators and rules except as defined below:
(last paragraph - add this to the beginning of the first paragraph in the section)
~~NOTE:~~ Queues and dynamic arrays have the same assignment and argument passing semantics.

PROPOSED:
5.14.1 Queue Operators
(new, modified 1st paragraph - includes the NOTE from the end of the section)
Queues and dynamic arrays have the same assignment and argument passing semantics. Also, queues support the same operations that can be performed on unpacked arrays, and using the same operators and rules except as defined below:

6.6 (multiple occurrences) - See Mantis #646
WAS: Note that in SystemVerilog, data can be declared in unnamed blocks as well as in named blocks.
PROPOSED: In SystemVerilog, data can be declared in unnamed blocks as well as in named blocks.

WAS: Note that automatic or dynamic variables cannot be written with nonblocking or continuous assignments. Automatic variables and dynamic constructs—objects handles, dynamic arrays, associative arrays, strings, and event variables—shall be limited to the procedural context.

PROPOSED: Fixed by Mantis #646.

6.7
WAS: Note that a SystemVerilog variable cannot have an implicit continuous assignment as part of its declaration, the way a net can. An assignment as part of the logic declaration is a variable initialization, not a continuous assignment. For example:
PROPOSED: Unlike SystemVerilog nets, a SystemVerilog variable cannot have an implicit continuous assignment as part of its declaration. An assignment as part of the logic declaration is a variable initialization, not a continuous assignment. For example:

6.9
WAS: Note that there is no category for identical types defined here because there is no construct in the SystemVerilog language that requires it. For example, as defined below, **int** can be interchanged with **bit signed** [31:0] wherever it is syntactically legal to do so. Users can define their own level of type identity by using the $typename system function (see 24.3), or through use of the PLI.
PROPOSED: SystemVerilog does not require a category for identical types to be defined here because there is no construct in the SystemVerilog language that requires it. For example, as defined below, **int** can be interchanged with **bit signed** [31:0] wherever it is syntactically legal to do so. Users can define their own level of type identity by using the $typename system function (see 24.3), or through use of the PLI.

6.9.1
WAS: Two array types match if they have the same number of unpacked dimensions and their slowest-varying dimensions have matching types and the same left and right range bounds. Note that the type of the slowest varying dimension of a multidimensional array type is itself an array type.
PROPOSED: Two array types match if they have the same number of unpacked dimensions and their slowest-varying dimensions have matching types and the same left and right range bounds. The type of the slowest varying dimension of a multidimensional array type is itself an array type.

6.9.2
WAS: Note that if any bit of a packed structure or union is 4-state, the entire structure or union is considered 4-state.
PROPOSED: *(This is an informational note. No change to the note - except use a smaller font for the note and separate into a separate paragraph)*
Note that if any bit of a packed structure or union is 4-state, the entire structure or union is considered 4-state (see Clause 4.11).
(Adding the supporting section number would be useful)

8.3
The preceding text in this paragraph is:
In SystemVerilog, an expression can include a blocking assignment, provided it does not have a timing control.

WAS: Note that such an assignment must be enclosed in parentheses to avoid common mistakes such as using a=b for a==b, or a|=b for a!=b.
PROPOSED: These blocking assignments must be enclosed in parentheses to avoid common mistakes such as using a=b for a==b, or a|=b for a!=b.

8.12
WAS: Note that unlike bit concatenation, the result of a string concatenation or replication is not truncated.
PROPOSED: Unlike bit concatenation, the result of a string concatenation or replication is not truncated.

8.13.2

WAS: Note that the **default** keyword applies to members in nested structures or elements in unpacked arrays in structures.

PROPOSED: Use of the **default** keyword applies to members in nested structures or elements in unpacked arrays in structures.

8.16 (multiple occurrences)

For example, suppose there is a structure type float:

```
  typedef struct {
    bit sign;
    bit [3:0] exponent;
    bit [10:0] mantissa;
  } float;
...
bind + function float fcopyf(float);     // unary +
bind + function float fcopyi(int);       // unary +
bind + function float fcopyr(real);      // unary +
bind + function float fcopyr(shortreal); // unary +
...
```

WAS: Note that the function prototype does not need to match the actual function declaration exactly. If it does not, then the normal implicit casting rules apply when calling the function. For example the fcopyi function can be defined with an **int** argument:

PROPOSED: A function prototype does not need to match the actual function declaration exactly. If it does not, then the normal implicit casting rules apply when calling the function. For example the fcopyi function can be defined with an **int** argument:

(Pertinent text)

```
bind = function float fcopyi(int); // cast int to float
bind = function float fcopyr(real); // cast real to float
bind = function float fcopyr(shortreal); // cast shortreal to float
```

WAS: The operators that can be overloaded are the arithmetic operators, the relational operators and assignment. Note that the assignment operator from a float to a float cannot be overloaded here because it is already legal. Similarly, equality and inequality between floats cannot be overloaded.

PROPOSED: The operators that can be overloaded are the arithmetic operators, the relational operators and assignment. The assignment operator from a float to a float cannot be overloaded above because it is already legal in the three preceding bind statements. Similarly, equality and inequality between floats cannot be overloaded.

8.19
PROPOSED: Fixed by Mantis #410

9.4
WAS: Note: by specifying **unique** or **priority**, it is not necessary to code a **default** case to trap unexpected case values. For Example:
PROPOSED: *(keep this as a note - smaller font and make this a separate note-paragraph)*
Note: by specifying **unique** or **priority**, it is not necessary to code a **default** case to trap unexpected case values.
Consider the following example:

9.4.1
WAS: In pattern-matching, the value *V* of an expression is always matched against a pattern. Note that static type checking ensures that *V* and the pattern have the same type. The result of a pattern match is:
PROPOSED: In pattern-matching, the value *V* of an expression is always matched against a pattern and static type checking ensures that *V* and the pattern have the same type. The result of a pattern match is:

9.4.1.1
WAS: Example (same as previous example, but note that the first inner **case** statement involves only structures and constants but no tagged unions):
PROPOSED: Example (same as previous example, except that the first inner **case** statement involves only structures and constants but no tagged unions):

9.6
WAS: Note that SystemVerilog does not include the C **goto** statement.
PROPOSED: *(keep this as a note - smaller font and keep it as a separate note-paragraph ... unless everybody would like to add a goto statement ☺)*
Note that SystemVerilog does not include the C **goto** statement.

9.10 (multiple occurrences)

```
module latch (output logic [31:0] y, input [31:0] a, input enable);
  always @(a iff enable == 1)
    y <= a; //latch is in transparent mode
endmodule
```

WAS: The event expression only triggers if the expression after the **iff** is true, in this case when enable is equal to 1. Note that such an expression is evaluated when a changes, and not when enable changes. Also note that **iff** has precedence over **or**. This can be made clearer by the use of parentheses.

PROPOSED: The event expression only triggers if the expression after the **iff** is true, in this case when enable is equal to 1. This type of expression is evaluated when a changes, and not when enable changes. Also in similar event expressions of this type, **iff** has precedence over **or**. This can be made clearer by the use of parentheses.

11.4.2

(From 11.4.1)

For example, calling the function bellow copies 1000 bytes each time the call is made.

```
  function int crc( byte packet [1000:1] );
    for( int j= 1; j <= 1000; j++ ) begin
      crc ^= packet[j];
    end
  endfunction
```

…

(From 11.4.2 - Pass by reference)

For example, the example above can be written as:

```
  function int crc( ref byte packet [1000:1] );
    for( int j= 1; j <= 1000; j++ ) begin
      crc ^= packet[j];
    end
  endfunction
```

WAS: Note that in the example, no change other than addition of the **ref** keyword is needed. The compiler knows that packet is now addressed via a reference, but users do not need to make these references explicit either in the callee or at the point of the call. That is, the call to either version of the crc function remains the same:

PROPOSED: As shown in the preceding example , no change other than addition of the **ref** keyword is needed. The compiler knows that packet is now addressed via a reference, but users do not need to make these references explicit either in the callee or at the point of the call. That is, the call to either version of the crc function remains the same:

11.4.3

WAS: The *default_value* is an expression. The expression is evaluated in the scope of the caller each time the subroutine is called. The elements of the expression must be visible at the scope of subroutine and, if used, at the scope of the caller. If the default_value is not used, the expression is not evaluated and need not be visible at the scope of the caller. Note that default values are only allowed with the ANSI style declaration.

PROPOSED: The *default_value* is an expression. The expression is evaluated in the scope of the caller each time the subroutine is called. The elements of the expression must be visible at the scope of subroutine and, if used, at the scope of the caller. If the default_value is not used, the expression is not evaluated and need not be visible at the scope of the caller. The use of default values shall only be allowed with the ANSI style declarations.

11.5 (multiple occurrences)

WAS: Several SystemVerilog functions can be mapped to the same foreign function by supplying the same *c_identifier* for several *fnames*. Note that all these SystemVerilog functions must have identical argument types, as defined in the next paragraph.

PROPOSED: Several SystemVerilog functions can be mapped to the same foreign function by supplying the same *c_identifier* for several *fnames*. The corresponding SystemVerilog functions must have identical argument types, as defined in the next paragraph.

WAS: Note that **import** "DPI" functions declared this way can be invoked by hierarchical reference the same as any normal SystemVerilog function.

PROPOSED: All **import** "DPI" functions declared this way can be invoked by hierarchical reference the same as any normal SystemVerilog function.

WAS: Import context functions can have side effects and can use other SystemVerilog interfaces (including but not limited to VPI). However, ~~note that~~ declaring an import context function does not automatically make any other simulator interface available. For VPI access (or any other interface access) to be possible, the appropriate implementation-defined mechanism must still be used to enable these interface(s). Note also that DPI calls do not automatically create or provide any handles or any special environment that might be needed by those other interfaces. ...

PROPOSED: Import context functions can have side effects and can use other SystemVerilog interfaces (including but not limited to VPI). However, declaring an import context function does not automatically make any other simulator interface available. For VPI access (or any other interface access) to be possible, the appropriate implementation-defined mechanism must still be used to enable these interface(s). Also, SystemVerilog DPI calls do not automatically create or provide any handles or any special environment that might be needed by those other interfaces. ...

WAS: To access functions defined in any other scope the foreign code shall have to change DPI context appropriately. Attempting to invoke an exported SystemVerilog function from a scope in which it is not directly visible shall result in a runtime error. How such errors are handled shall be implementation dependent. If an imported function needs to invoke an exported function that is not visible from the current scope, it needs to change, via svSetScope, the current scope to a scope that does have visibility to the exported function. This is conceptually equivalent to making a hierarchically qualified function call in SystemVerilog. The current SystemVerilog context shall be preserved across a call to an exported function, even if current context has been modified by an application. Note that context is not defined for non-context imports and attempting to use any functionality depending on context from non-context imports can lead to unpredictable behavior.
PROPOSED: To access functions defined in any other scope the foreign code shall have to change DPI context appropriately. Attempting to invoke an exported SystemVerilog function from a scope in which it is not directly visible shall result in a runtime error. How such errors are handled shall be implementation dependent. If an imported function needs to invoke an exported function that is not visible from the current scope, it needs to change, via svSetScope, the current scope to a scope that does have visibility to the exported function. This is conceptually equivalent to making a hierarchically qualified function call in SystemVerilog. The current SystemVerilog context shall be preserved across a call to an exported function, even if current context has been modified by an application. For non-context imports the context is not defined and attempting to use any functionality depending on context from non-context imports can lead to unpredictable behavior.

12.6

12.6 Object methods

An object's methods can be accessed using the same syntax used to access class properties:

```
Packet p = new;
status = p.current_status();
```

WAS: Note that the assignment to status is not:

```
status = current_status(p);
```

PROPOSED: The above assignment to status cannot be written as:

```
status = current_status(p);
```

12.7

SystemVerilog provides a mechanism for initializing an instance at the time the object is created. When an object is created, for example

```
Packet p = new;
```

The system executes the **new** function associated with the class:

```
class Packet;
  integer command;
  function new();
    command = IDLE;
  endfunction
endclass
```

WAS: Note that **new** is now being used in two very different contexts with very different semantics. The variable declaration creates an object of class Packet. In the course of creating this instance, the **new** function is invoked, in which any specialized initialization required can be done. The **new** function is also called the *class constructor*.

PROPOSED: As shown above, **new** is now being used in two very different contexts with very different semantics. The variable declaration creates an object of class Packet. In the course of creating this instance, the **new** function is invoked, in which any specialized initialization required can be done. The **new** function is also called the *class constructor*.

12.8

```
class Packet ;
  static integer fileId = $fopen( "data", "r" );
```

Now, fileID shall be created and initialized once. Thereafter, every Packet object can access the file descriptor in the usual way:

```
Packet p;
c = $fgetc( p.fileID );
```

WAS: Note that static class properties can be used without creating an object of that type.
PROPOSED: The static class properties can be used without creating an object of that type.

12.10

The x is now both a property of the class and an argument to the function **new**. In the function **new**, an unqualified reference to x shall be resolved by looking at the innermost scope, in this case the subroutine argument declaration. To access the instance class property, it is qualified with the **this** keyword, to refer to the current instance. ...

WAS: Note that in writing methods, members can be qualified with **this** to refer to the current instance, but it is usually unnecessary.
PROPOSED: *(keep this as a note - smaller font and keep it as a separate note-paragraph)*

12.11

Declaring a class variable only creates the name by which the object is known. Thus:

```
Packet p1;
```

creates a variable, p1, that can hold the handle of an object of class Packet, but the initial value of p1 is **null**. The object does not exist, and p1 does not contain an actual handle, until an instance of type Packet is created:

```
p1 = new;
```

Thus, if another variable is declared and assigned the old handle, p1, to the new one, as in:

```
Packet p2;
p2 = p1;
```

then there is still only one object, which can be referred to with either the name p1 or p2.

...

WAS: Note, **new** was executed only once, so only one object has been created.

PROPOSED: In this example, **new** was executed only once, so only one object has been created.

12.17

A **protected** class property or method has all of the characteristics of a **local** member, except that it can be inherited; it is visible to subclasses.

WAS: Note that within the class, a local method or class property of the class can be referenced, even if it is in a different instance. For example:

PROPOSED: Within a class, a local method or class property of the same class can be referenced, even if it is in a different instance of the same class. For example:

```
class Packet;
  local integer i;
  function integer compare (Packet other);
    compare = (this.i == other.i);
  endfunction
endclass
```

A strict interpretation of encapsulation might say that other.i should not be visible inside of this packet, since it is a local class property being referenced from outside its instance. Within the same class, however, these references are allowed. In this case, this.i shall be compared to other.i and the result of the logical comparison returned.

12.24 (multiple occurrences)

Sometimes a class variable needs to be declared before the class itself has been declared. For example, if two classes each need a handle to the other. When, in the course of processing the declaration for the first class, the compiler encounters the reference to the second class, that reference is undefined and the compiler flags it as an error.

This is resolved using **typedef** to provide a forward declaration for the second class:

```
typedef class C2; // C2 is declared to be of type class
class C1;
  C2 c;
endclass
class C2;
  C1 c;
endclass
```

WAS: In this example, C2 is declared to be of type **class**, a fact that is re-enforced later in the source code. Note that the **class** construct always creates a type, and does not require a **typedef** declaration for that purpose (as in **typedef class** …). This is consistent with common C++ use.
PROPOSED: In this example, C2 is declared to be of type **class**, a fact that is re-enforced later in the source code. In SystemVerilog, the **class** construct always creates a type, and does not require a **typedef** declaration for that purpose (as in **typedef class** …). This is consistent with common C++ use.

WAS: Note that the class keyword in the statement typedef class C2; is not necessary, and is used only for documentation purposes. The statement typedef C2; is equivalent and shall work the same way.
PROPOSED: In the preceding example, the class keyword in the statement typedef class C2; is not necessary, and is used only for documentation purposes. The statement typedef C2; is equivalent and shall work the same way.

13.4.3

WAS: It is important to note that the **inside** operator is bidirectional, thus, the second example above is equivalent to a == b || a == c.
PROPOSED: In SystemVerilog, the **inside** operator is bidirectional, thus, the second example above is equivalent to a == b || a == c.

13.4.11 (multiple occurrences)
WAS: Unlike the count_ones function, more complex properties, which require temporary state or unbounded loops, may be impossible to convert into a single expression. The ability to call functions, thus, enhances the expressive power of the constraint language and reduces the likelihood of errors. Note that the two constraints above are not completely equivalent; C2 is bidirectional (length can constrain v and vice-versa), whereas C1 is not.
PROPOSED: Unlike the count_ones function, more complex properties, which require temporary state or unbounded loops, may be impossible to convert into a single expression. The ability to call functions, thus, enhances the expressive power of the constraint language and reduces the likelihood of errors. The two constraints, C1 and C2, from above are not completely equivalent; C2 is bidirectional (length can constrain v and vice-versa), whereas C1 is not.

... For example:

```
class B;
  rand int x, y;
  constraint C { x <= F(y); }
  constraint D { y inside { 2, 4, 8 } ; }
endclass
```

WAS: Forces y to be solved before x. Thus, constraint D is solved separately before constraint C, which uses the values of y and F(y) as state variables. Note that the behavior for variable ordering implied by function arguments differs from the behavior for ordering specified using the "**solve**...**before**..." constraint; function argument variable ordering subdivides the solution space thereby changing it. Since constraints on higher-priority variables are solved without considering lower-priority constraints at all this subdivision can cause the overall constraints to fail. Within each prioritized set of constraints, cyclical (**randc**) variables are solved first.
PROPOSED: Forces y to be solved before x. Thus, constraint D is solved separately before constraint C, which uses the values of y and F(y) as state variables. In SystemVerilog, the behavior for variable ordering implied by function arguments differs from the behavior for ordering specified using the "**solve**...**before**..." constraint; function argument variable ordering subdivides the solution space thereby changing it. Since constraints on higher-priority variables are solved without considering lower-priority constraints at all this subdivision can cause the overall constraints to fail. Within each prioritized set of constraints, cyclical (**randc**) variables are solved first.

13.5.3

WAS: **13.5.3 Randomization methods notes**

PROPOSED: **13.5.3 Behavior of Randomization methods**

13.11

For example:

```
module stim;
  bit [15:0] addr;
  bit [31:0] data;
  function bit gen_stim();
    bit success, rd_wr;
    // call std::randomize
    success = randomize( addr, data, rd_wr );
    return rd_wr ;
  endfunction
  ...
endmodule
```

WAS: The function gen_stim calls std::randomize() with three variables as arguments: addr, data, and rd_wr. Thus, std::randomize() assigns new random variables to those variables that are visible in the scope of the gen_stim function. Note that addr and data have module scope, whereas rd_wr has scope local to the function. The above example can also be written using a class:

PROPOSED: The function gen_stim calls std::randomize() with three variables as arguments: addr, data, and rd_wr. Thus, std::randomize() assigns new random variables to those variables that are visible in the scope of the gen_stim function. In the preceding example, addr and data have module scope, whereas rd_wr has scope local to the function. The preceding example can also be written using a class:

14.3.7

WAS: Note that calling peek() can cause one message to unblock more than one process. As long as a message remains in the mailbox queue, any process blocked in either a peek() or get() operation shall become unblocked.

PROPOSED: Calling the peek() method can also cause one message to unblock more than one process. As long as a message remains in the mailbox queue, any process blocked in either a peek() or get() operation shall become unblocked.

15.2

WAS: The SystemVerilog language is defined in terms of a discrete event execution model. The discrete event simulation is described in more detail in this clause to provide a context to describe the meaning and valid interpretation of SystemVerilog constructs. These resulting definitions provide the standard SystemVerilog reference algorithm for simulation, which all compliant simulators shall implement. Note that there is a great deal of choice ~~in the definitions that follow,~~ and differences in some details of execution are to be expected between different simulators. In addition, SystemVerilog simulators are free to use different algorithms than those described in this clause, provided the user-visible effect is consistent with the reference algorithm.

PROPOSED: The SystemVerilog language is defined in terms of a discrete event execution model. The discrete event simulation is described in more detail in this clause to provide a context to describe the meaning and valid interpretation of SystemVerilog constructs. These resulting definitions provide the standard SystemVerilog reference algorithm for simulation, which all compliant simulators shall implement. Within the following event execution model definitions, there is a great deal of choice and differences in some details of execution are to be expected between different simulators. In addition, SystemVerilog simulators are free to use different algorithms than those described in this clause, provided the user-visible effect is consistent with the reference algorithm.

16.13
WAS: Wait for the falling edge of the specified 1-bit slice dom.sign[a]. ~~Note that the index a is evaluated at runtime.~~
@(**negedge** dom.sign[*a*]);

(separate the note, place the note after the example code, make the font smaller - This can be an informative note because there is a sentence shortly before this list of examples that states, "Slices can include dynamic indices, which are evaluated once, when the @ expression executes.")

PROPOSED: Wait for the falling edge of the specified 1-bit slice dom.sign[a].
@(**negedge** dom.sign[*a*]);
Note: the dynamic index a is evaluated at runtime.

17.4
WAS: Since the program schedules events in the Reactive region, the clocking block construct is very useful to automatically sample the steady-state values of previous time steps or clock cycles. Programs that read design values exclusively through clocking blocks with #0 input skews are insensitive to read-write races. It is important to note that simply sampling input signals (or setting non-zero skews on clocking block inputs) does not eliminate the potential for races. Proper input sampling only addresses a single clocking block. With multiple clocks, the arbitrary order in which overlapping or simultaneous clocks are processed is still a potential source for races. The program construct addresses this issue by scheduling its execution in the Reactive region, after all design events have been processed, including clocks driven by nonblocking assignments.
PROPOSED: Since the program schedules events in the Reactive region, the clocking block construct is very useful to automatically sample the steady-state values of previous time steps or clock cycles. Programs that read design values exclusively through clocking blocks with #0 input skews are insensitive to read-write races. It is important to understand that simply sampling input signals (or setting non-zero skews on clocking block inputs) does not eliminate the potential for races. Proper input sampling only addresses a single clocking block. With multiple clocks, the arbitrary order in which overlapping or simultaneous clocks are processed is still a potential source for races. The program construct addresses this issue by scheduling its execution in the Reactive region, after all design events have been processed, including clocks driven by nonblocking assignments.

18.2

WAS: Note: The assertion control system tasks are described in 24.9.

PROPOSED: *(keep this as a note - smaller font and keep it as a separate note-paragraph)*

Note: The assertion control system tasks are described in 24.9.

18.3

The sampled values are used to evaluate value change expressions or boolean subexpressions that are required to determine a match of a sequence.

WAS: Note:

— It is important to ensure that the defined clock behavior is glitch free. Otherwise, wrong values can be sampled.

— If a variable that appears in the expression for clock also appears in an expression with an assertion, the values of the two usages of the variable can be different. The current value of the variable is used in the clock expression, while the sampled value of the variable is used within the assertion.

PROPOSED: For concurrent assertions:

— It is important to ensure that the defined clock behavior is glitch free. Otherwise, wrong values can be sampled.

— If a variable that appears in the expression for clock also appears in an expression with an assertion, the values of the two usages of the variable can be different. The current value of the variable is used in the clock expression, while the sampled value of the variable is used within the assertion.

18.6
A sequence is declared with optional formal arguments. When a sequence is instantiated, actual arguments can be passed to the sequence. The sequence gets expanded with the actual arguments by replacing the formal arguments with the actual arguments. Semantic checks are performed to ensure that the expanded sequence with the actual arguments is legal.

An actual argument can replace an:
— identifier
— expression
— event control expression
— upper delay range or repetition range if the actual argument is $

WAS: ~~Note that~~ variables used in a sequence that are not formal arguments to the sequence are resolved according to the scoping rules from the scope in which the sequence is declared.
PROPOSED: Variables used in a sequence that are not formal arguments to the sequence are resolved according to the scoping rules from the scope in which the sequence is declared.

18.7.3
When intermediate optional arguments between two arguments are not needed, a comma must be placed for each omitted argument. For example,

```
$past(in1, , enable);
```

WAS: Here, a comma is specified to omit *number_of_ticks*. The default of one is used for the empty *number_of_ticks* argument. Note that a comma for the omitted *clocking_event* argument ~~is not needed,~~ as it does not fall within the specified arguments.
PROPOSED: Here, a comma is specified to omit *number_of_ticks*. The default of one is used for the empty *number_of_ticks* argument. There is no need to include a comma for the omitted *clocking_event* argument as it does not fall within the specified arguments.

18.8

In a single cycle, there can be multiple matches of a sequence instance to which ended is applied, and these matches can have different valuations of the local variables. The multiple matches are treated semantically the same way as matching both disjuncts of an **or** (see below). In other words, the thread evaluating the instance to which ended is applied will fork to account for such distinct local variable valuations.

WAS: Note that when a local variable is a formal argument of a sequence declaration, it is illegal to declare the variable, as shown below.
PROPOSED: When a local variable is a formal argument of a sequence declaration, it is illegal to declare the variable, as shown below.

```
sequence sub_seq3(lv);
  int lv; // illegal since lv is a formal argument
  (a ##1 !a, lv = data_in) ##1 !b[*0:$] ##1 b && (data_out == lv);
endsequence
```

18.11.2

The use of implication when multi-clock sequences and properties are involved is explained in 18.12.

WAS: The following example illustrates a bus operation for data transfer from a master to a target device. When the bus enters a data transfer phase, multiple data phases can occur to transfer a block of data. During the data transfer phase, a data phase completes on any rising clock edge on which irdy is asserted and either trdy or stop is asserted. Note that an asserted signal ~~here~~ implies a value of low.
PROPOSED: The following example illustrates a bus operation for data transfer from a master to a target device. When the bus enters a data transfer phase, multiple data phases can occur to transfer a block of data. During the data transfer phase, a data phase completes on any rising clock edge on which irdy is asserted and either trdy or stop is asserted. In this example, an asserted signal implies a value of low.

18.12.3
The scope of a clocking event does not flow into the reset condition of **disable iff**.

WAS: ~~Note that~~ juxtaposing two clocking events nullifies the first of them:
PROPOSED: Juxtaposing two clocking events nullifies the first of them; therefore, the following two-clocking-event statement

```
@(d) @(c) x
```

is equivalent to

```
@(c) x
```

because the flow of clock *d* is immediately overridden by clock *c*.

18.13.1
WAS: ~~Note:~~ The pass and fail statements are executed in the Reactive region. The regions of execution are explained in the scheduling semantics Clause 15.
PROPOSED: The pass and fail statements of an **assert** statement are executed in the Reactive region. The regions of execution are explained in the scheduling semantics Clause 15.

18.13.2
WAS: Note that **assume** does not provide an action block, as the actions for an assumption serve no purpose.
PROPOSED: The **assume** statement does not provide an action block, as the actions for an assumption serve no purpose.

18.15 (multiple occurrences)
Example of binding a program instance to a module:

```
  bind cpu fpu_props fpu_rules_1(a,b,c);
```

Where:
— cpu is the name of the target module.
— fpu_props is the name of the program to be instantiated.
— fpu_rules_1 is the program instance name to be created in the target scope.
— An instance named fpu_rules_1 is instantiated in every instance of module cpu.
...

WAS: — The first three ports of program fpu_props get bound to objects a, b, and c in module cpu. ~~Note that~~ these objects are viewed from module cpu's point of view. They are completely distinct from any objects named a, b, and c that are visible in the scope that contains the **bind** directive.
PROPOSED: — The first three ports of program fpu_props get bound to objects a, b, and c in module cpu (these objects are viewed from module cpu's point of view and they are completely distinct from any objects named a, b, and c that are visible in the scope that contains the **bind** directive).

WAS: It is legal for more than one **bind** statement to bind a *bind_instantiation* into the same target scope. However, it shall be an error for a *bind_instantiation* to introduce an instance name that clashes with another name in the module name space of the target scope (See 19.13). This applies to both pre-existing names as well as instance names introduced by other **bind** statements. ~~Note that~~ the latter situation will occur if the design contains more than one instance of a module containing a **bind** statement.
PROPOSED: It is legal for more than one **bind** statement to bind a *bind_instantiation* into the same target scope. However, it shall be an error for a *bind_instantiation* to introduce an instance name that clashes with another name in the module name space of the target scope (See 19.13). This applies to both pre-existing names as well as instance names introduced by other **bind** statements. The latter situation will occur if the design contains more than one instance of a module containing a **bind** statement.

19.6
WAS: This allows the same module name, e.g. and2, to occur in different parts of the design and represent different modules. ~~Note that~~ an alternative way of handling this problem is to use configurations.
PROPOSED: This allows the same module name, e.g. and2, to occur in different parts of the design and represent different modules. An alternative way of handling this problem is to use configurations.

19.7
WAS: To support separate compilation, extern declarations of a module can be used to declare the ports on a module without defining the module itself. An extern module declaration consists of the keyword **extern** followed by the module name and the list of ports for the module. Both list of ports syntax (possibly with parameters), and original Verilog style port declarations can be used. Note that the potential existence of defparams precludes the checking of the port connection information prior to elaboration time even for list of ports style declarations.
PROPOSED: *(Separate the note -use a smaller font for the note)*
To support separate compilation, extern declarations of a module can be used to declare the ports on a module without defining the module itself. An extern module declaration consists of the keyword **extern** followed by the module name and the list of ports for the module. Both list of ports syntax (possibly with parameters), and original Verilog style port declarations can be used.

Note that the potential existence of defparams precludes the checking of the port connection information prior to elaboration time even for list of ports style declarations.

19.12.2
WAS: Note that where the data types differ between the port declaration and connection, an initial value change event can be caused at time zero.
PROPOSED: If there is a data type difference between the port declaration and connection, an initial value change event can be caused at time zero.

20.2.1
WAS: This example shows a simple bus implemented without interfaces. Note that the logic type can replace wire and reg if no resolution of multiple drivers is needed.
PROPOSED: This example shows a simple bus implemented without interfaces. The logic type, as used in this example, can replace wire and reg if no resolution of multiple drivers is needed.

20.3
WAS: Note: Because the instantiated interface names do not match the interface names used in the memMod and cpuMod modules, implicit port connections cannot be used for this example.
PROPOSED: In the preceding example, the instantiated interface names do not match the interface names used in the memMod and cpuMod modules; therefore, implicit port connections cannot be used for this example.

20.4 (multiple occurrences)
WAS: The syntax of interface_name.modport_name reference_name gives a local name for a hierarchical reference. Note that this can be generalized to any interface with a given modport name by writing **interface**. modport_name reference_name.
PROPOSED: The syntax of interface_name.modport_name reference_name gives a local name for a hierarchical reference. This technique can be generalized to any interface with a given modport name by writing **interface**. modport_name reference_name.

WAS: Note that if no **modport** is specified in the module header or in the port connection, then all the nets and variables in the interface are accessible with direction **inout** or **ref**, as in the examples above.
PROPOSED: Adding modports to an interface does not require that any of the modports be used when the interface is used. If no **modport** is specified in the module header or in the port connection, then all the nets and variables in the interface are accessible with direction **inout** or **ref**, as in the examples above.

20.6.4

WAS: For a read task, only one module should actively respond to the task call, e.g. the one containing the appropriate address. The tasks in the other modules should return with no effect. Only then should the active task write to the result variables.

Note multiple export of functions is not allowed, because they must always write to the result.

PROPOSED: For a read task, only one module should actively respond to the task call, e.g. the one containing the appropriate address. The tasks in the other modules should return with no effect. Only then should the active task write to the result variables.

Unlike tasks, multiple export of functions is not allowed, because they must always write to the result.

20.8.1

20.8.1 Virtual interfaces and clocking blocks

Clocking blocks and interfaces can be combined to represent the interconnect between synchronous blocks. Moreover, because clocking blocks provide a procedural mechanism to assign values to both nets and variables, they are ideally suited to be used by virtual interfaces. For example:

```
  interface SyncBus( input bit clk );
    wire a, b, c;

    clocking sb @(posedge clk);
      input a;
      output b;
      inout c;
    endclocking
  endinterface

  typedef virtual SyncBus VI; // A virtual interface type

  task do_it( VI v ); //handles any SyncBus via clocking sb
    if( v.sb.a == 1 )
      v.sb.b <= 0;
    else
      v.sb.c <= ##1 1;
endtask
```

WAS: In the preceding example, interface SyncBus includes a clocking block, which is used by task do_it to ensure synchronous access to the interface's signals: a, b, and c. Note that changes to the storage type of the

interface signals (from net to variable and vice-versa) requires no changes to the task. The interfaces can be instantiated as shown below.
PROPOSED: In the preceding example, interface SyncBus includes a clocking block, which is used by task do_it to ensure synchronous access to the interface's signals: a, b, and c. A change to the storage type of the interface signals (from net to variable and vice-versa) requires no changes to the task. The interfaces can be instantiated as shown below.

There is no 20.10.1

21.10
WAS: It is important to note that the cumulative coverage considers the union of all significant bins, thus, it includes the contribution of all bins (including overlapping bins) of all instances.
PROPOSED: It is important to understand that the cumulative coverage considers the union of all significant bins, thus, it includes the contribution of all bins (including overlapping bins) of all instances.

22.2
WAS: Note that function $isunbounded is used for checking the validity of the actual arguments.
PROPOSED: *(This is an informational usage note. No change to the note - except use a smaller font for the note)*

24.17
WAS: Note that the diagram would be identical if one or more of the unpacked dimension declarations were reversed, as in:
reg [31:0] mem [2:0][0:4][8:5]
PROPOSED: The above diagram would be identical if one or more of the unpacked dimension declarations were reversed, as in:
reg [31:0] mem [2:0][0:4][8:5]

25.2

WAS: Note that the current VCD format does not indicate whether a variable has been declared as **signed** or **unsigned**.

PROPOSED: *(This is an informational note. No change to the note - except use a smaller font for the note)*

28.3

Every task or function imported to SystemVerilog must eventually resolve to a global symbol. Similarly, every task or function exported from SystemVerilog defines a global symbol. Thus the tasks and functions imported to and exported from SystemVerilog have their own global name space of linkage names, different from compilation-unit scope name space. Global names of imported and exported tasks and functions must be unique (no overloading is allowed) and shall follow C conventions for naming; specifically, such names must start with a letter or underscore, and can be followed by alphanumeric characters or underscores. Exported and imported tasks and functions, however, can be declared with local SystemVerilog names. Import and export declarations allow users to specify a global name for a function in addition to its declared name. Should a global name clash with a SystemVerilog keyword or a reserved name, it shall take the form of an escaped identifier. The leading backslash (\ ) character and the trailing white space shall be stripped off by the SystemVerilog tool to create the linkage identifier. ...

WAS: Note that after this stripping, the linkage identifier so formed must comply with the normal rules for C identifier construction.

PROPOSED: After this stripping, the linkage identifier so formed must comply with the normal rules for C identifier construction.

28.4.1.1

WAS: Note that imported tasks can consume time, similar to native SystemVerilog tasks.

PROPOSED: *(This is an informational note. No change to the note - except use a smaller font for the note)*

28.4.1.4

WAS: NOTE—In this last scenario, a block of memory is allocated and freed in the foreign code, even when the standard functions malloc and free are called directly from SystemVerilog code.

PROPOSED: *(This is an informational note. No change to the note - except use a smaller font for the note)*

28.4.3 (multiple occurrences)
WAS: Only calls of context imported tasks or functions are properly instrumented and cause conservative optimizations; therefore, only those tasks or functions can safely call all tasks or functions from other APIs, including PLI and VPI functions or exported SystemVerilog tasks or functions. For imported tasks or functions not specified as **context**, the effects of calling PLI or VPI functions or SystemVerilog tasks or functions can be unpredictable and such calls can crash if the callee requires a context that has not been properly set. However note that declaring an import context task or function does not automatically make any other simulator interface automatically available. For VPI access (or any other interface access) to be possible, the appropriate implementation defined mechanism must still be used to enable these interface(s). Note also that DPI calls do not automatically create or provide any handles or any special environment that can be needed by those other interfaces. It is the user's responsibility to create, manage or otherwise manipulate the required handles/environment(s) needed by the other interfaces.
PROPOSED: Only calls of context imported tasks or functions are properly instrumented and cause conservative optimizations; therefore, only those tasks or functions can safely call all tasks or functions from other APIs, including PLI and VPI functions or exported SystemVerilog tasks or functions. For imported tasks or functions not specified as **context**, the effects of calling PLI or VPI functions or SystemVerilog tasks or functions can be unpredictable and such calls can crash if the callee requires a context that has not been properly set; however, declaring an import context task or function does not automatically make any other simulator interface automatically available. For VPI access (or any other interface access) to be possible, the appropriate implementation defined mechanism must still be used to enable these interface(s). Realize also that DPI calls do not automatically create or provide any handles or any special environment that can be needed by those other interfaces. It is the user's responsibility to create, manage or otherwise manipulate the required handles/environment(s) needed by the other interfaces.

28.4.4 (multiple occurrences)

An import declaration specifies the task or function name, function result type, and types and directions of formal arguments. It can also provide optional default values for formal arguments. Formal argument names are optional unless argument binding by name is needed. An import declaration can also specify an optional task or function property. Imported functions can have the properties context or pure; imported tasks can have the property **context**.

WAS: Note that an import declaration is equivalent to defining a task or function of that name in the SystemVerilog scope in which the import declaration occurs, and thus multiple imports of the same task or function name into the same scope are forbidden. Note that this declaration scope is particularly important in the case of imported context tasks or functions, see 28.4.3; for non-context imported tasks or functions the declaration scope has no other implications other than defining the visibility of the task or function.

PROPOSED: *(The first "Note that" should be changed to "Since" as shown below, but the second note is an informational note and should be placed in a separate paragraph with smaller font for the note)*

Since an import declaration is equivalent to defining a task or function of that name in the SystemVerilog scope in which the import declaration occurs, and thus multiple imports of the same task or function name into the same scope are forbidden.

Note that this declaration scope is particularly important in the case of imported context tasks or functions, see 28.4.3; for non-context imported tasks or functions the declaration scope has no other implications other than defining the visibility of the task or function.

WAS: ~~Note that~~ multiple declarations of the same imported or exported task or function in different scopes can vary argument names and default values, provided the type compatibility constraints are met.

PROPOSED: It is permitted to have multiple declarations of the same imported or exported task or function in different scopes; therefore, argument names and default values can vary, provided the type compatibility constraints are met.

28.4.6
WAS: — packed one dimensional arrays of type **bit** and **logic**
~~Note however, that~~ every packed type, whatever ~~is~~ its structure, is eventually equivalent to a packed one dimensional array. Therefore practically all packed types are supported, although their internal structure (individual fields of structs, multiple dimensions of arrays) shall be transparent and irrelevant.
PROPOSED: — packed one dimensional arrays of type **bit** and **logic**
Since every packed type, whatever its structure, is eventually equivalent to a packed one dimensional array. Therefore practically all packed types are supported, although their internal structure (individual fields of structs, multiple dimensions of arrays) shall be transparent and irrelevant.

28.6 (multiple occurrences)
WAS: Note that class member functions cannot be exported, but all other SystemVerilog functions can be exported.
PROPOSED: One important restriction exists. Class member functions cannot be exported, but all other SystemVerilog functions can be exported.

WAS: *c_identifier* is optional here. It defaults to *function_identifier*. For rules describing *c_identifier*, see 28.3. ~~Note that all export functions are always context functions.~~ No two functions in the same SystemVerilog scope can be exported with the same explicit or implicit *c_identifier*. The export declaration and the definition of the corresponding SystemVerilog function can occur in any order. Only one export declaration is permitted per SystemVerilog function.
PROPOSED: *c_identifier* is optional here. It defaults to *function_identifier*. For rules describing *c_identifier*, see 28.3. No two functions in the same SystemVerilog scope can be exported with the same explicit or implicit *c_identifier*. The export declaration and the definition of the corresponding SystemVerilog function can occur in any order. Only one export declaration is permitted per SystemVerilog function and all export functions are always context functions.

28.8 (multiple occurrences)
WAS: An imported task or function is said to be in the disabled state when a **disable** statement somewhere in the design targets either it or a parent for disabling. Note that the only way for an imported task or function to enter the disabled state is immediately after the return of a call to an exported task or function. An important aspect of the protocol is that disabled import tasks and functions must programmatically acknowledge that they have been disabled. A task or function can determine that it is in the disabled state by calling the API function svIsDisabledState().
PROPOSED: An imported task or function is said to be in the disabled state when a **disable** statement somewhere in the design targets either it or a parent for disabling. An imported task or function can only enter the disabled state immediately after the return of a call to an exported task or function. An important aspect of the protocol is that disabled import tasks and functions must programmatically acknowledge that they have been disabled. A task or function can determine that it is in the disabled state by calling the API function svIsDisabledState().

WAS: Note that if an exported task itself is the target of a disable, its parent imported task is not considered to be in the disabled state when the exported task returns. In such cases the exported task shall return value 0, and calls to svIsDisabledState() shall return 0 as well.
PROPOSED: If an exported task itself is the target of a disable, its parent imported task is not considered to be in the disabled state when the exported task returns. In such cases the exported task shall return value 0, and calls to svIsDisabledState() shall return 0 as well.

29.3.1
WAS: NOTES
1—As with all VPI handles, assertion handles are handles to a specific instance of a specific assertion.
2—Unnamed assertions cannot be found by name.
PROPOSED: IMPORTANT DETAILS
1—As with all VPI handles, assertion handles are handles to a specific instance of a specific assertion.
2—Unnamed assertions cannot be found by name.

29.3.2.1 (multiple occurrences)
WAS: Assertions can occur in modules and interfaces: for assertions defined in modules, the instance field in the s_vpi_assertion_info structure shall contain the handle to the appropriate module or interface instance. Note that VPI does not currently define the information model for interfaces and therefore the interface instance handle shall be implementation dependent.
PROPOSED: *(The note is an informational note and should be placed in a separate paragraph with smaller font for the note)*
Assertions can occur in modules and interfaces: for assertions defined in modules, the instance field in the s_vpi_assertion_info structure shall contain the handle to the appropriate module or interface instance.
Note that VPI does not currently define the information model for interfaces and therefore the interface instance handle shall be implementation dependent.

WAS: NOTE: a single call returns all the information for efficiency reasons.
PROPOSED: *(This is an informational note. No change to the note - except use a smaller font for the note)*

29.4.2
WAS: NOTES
1—In a failing transition, there shall always be at least one element in the expression array.
2—Placing a step callback results in the same callback function being invoked for both success and failure steps.
3—The content of
PROPOSED: IMPORTANT DETAILS
1—In a failing transition, there shall always be at least one element in the expression array.
2—Placing a step callback results in the same callback function being invoked for both success and failure steps.
3—The content of

29.5.1
WAS: vpiAssertionSysEnd discard all attempts in progress and disables any further assertions from starting. All assertion callbacks currently installed shall be removed. Note that once this control is issued, no further assertion related actions shall be permitted.
PROPOSED: vpiAssertionSysEnd discard all attempts in progress and disables any further assertions from starting. All assertion callbacks currently installed shall be removed. Once this control is issued, no further assertion related actions shall be permitted.

29.5.2
PROPOSED: Proposal already entered as Mantis #432

30.2.2.1 (multiple occurrences)
WAS: ‘SV_COV_START
If possible, starts collecting coverage information in the specified hierarchy. No effect if coverage is already being collected. Note that coverage is automatically started at the beginning of simulation for all portions of the hierarchy enabled for coverage.
PROPOSED: ‘SV_COV_START
If possible, starts collecting coverage information in the specified hierarchy. No effect if coverage is already being collected. Coverage is automatically started at the beginning of simulation for all portions of the hierarchy enabled for coverage.

WAS: ‘SV_COV_CHECK
Checks if coverage information can be obtained from the specified hierarchy. ~~Note the possibility of having~~ coverage information does imply that coverage is being collected, as the coverage could have been stopped.

PROPOSED: *(Typo?? Did the above note mean to say that "coverage information does **NOT** imply that coverage is being collected?" If not, the Note is very confusing to me)*
‘SV_COV_CHECK
Checks if coverage information can be obtained from the specified hierarchy. The existence of coverage information does not imply that coverage is being collected, as the coverage could have been stopped.

WAS: NOTE—Definition names are represented as strings, whereas instance names are referenced by hierarchical paths. A hierarchical path need not include any . if the path refers to an instance in the current context (i.e., normal Verilog hierarchical path rules apply).
PROPOSED: *(This is an informational note. No change to the note - the note already uses a smaller font)*

30.2.2.2
WAS: NOTE—This value is proportional to the design size and structure, so it also needs to be constant through multiple independent simulations and compilations of the same design, assuming any compilation options do not modify the coverage support or design structure.

PROPOSED: *(This is an informational note. No change to the note - the note already uses a smaller font)*

30.2.2.5
WAS: NOTES
1—The coverage database format is implementation-dependent.
2—Mapping of names to actual directories/files is implementation-dependent. There is no requirement that a coverage name map to any specific set of files or directories.
PROPOSED: DETAILS
1—The coverage database format is implementation-dependent.
2—Mapping of names to actual directories/files is implementation-dependent. There is no requirement that a coverage name map to any specific set of files or directories.

30.4.3 (multiple occurrences)
WAS: Returns the number of times each coverable entity referred by the handle has been covered. Note that this is only easily interpretable when the handle points to a unique coverable item (such as an individual statement); when handle points to an item containing multiple coverable entities (such as a handle to a block statement containing a number of statements), the result is the sum of coverage counts for each of the constituent entities.
PROPOSED: Returns the number of times each coverable entity referred by the handle has been covered. The handle coverage information is only easily interpretable when the handle points to a unique coverable item (such as an individual statement); when handle points to an item containing multiple coverable entities (such as a handle to a block statement containing a number of statements), the result is the sum of coverage counts for each of the constituent entities.

WAS: Returns the number of coverable entities pointed by the handle. Note that this shall always return 1 (one) when applied to an assertion or FSM state handle.
PROPOSED: Returns the number of coverable entities pointed by the handle. The number returned shall always be return 1 (one) when applied to an assertion or FSM state handle.

30.4.4
WAS: Controls the collection of coverage on the given instance or assertion. Note that statement, toggle and FSM coverage are not individually controllable (i.e., they are controllable only at the instance level and not on a per statement/signal/FSM basis).
PROPOSED: Controls the collection of coverage on the given instance or assertion. Statement, toggle and FSM coverage are not individually controllable (i.e., they are controllable only at the instance level and not on a per statement/signal/FSM basis).

31.8.3
WAS: Note that loading the object means loading the object from a database into memory, or marking it for active use if it is already in the memory hierarchy.
PROPOSED: "Loading the object" means loading the object from a database into memory, or marking it for active use if it is already in the memory hierarchy.

31.8.4
WAS: vpiHandle trvsHndl = vpi_handle(vpiTrvsObj, object_handle);
Note that the user (or tool) application can create more than one value change traverse handle for the same object, thus providing different views of the value changes.
PROPOSED: vpiHandle trvsHndl = vpi_handle(vpiTrvsObj, object_handle);
A user (or tool) application can create more than one value change traverse handle for the same object, thus providing different views of the value changes.

31.10
WAS: The SystemVerilog tool the user application is running under is responsible for loading the appropriate extension, i.e. the reader API library in the case of the read API. The extension name is used for this purpose, following a specific policy, for example, this extension name can be the name of the library to be loaded. Once the reader API library is loaded all VPI function calls that wish to use the implementation in the library shall be performed using the returned p_vpi_extension pointer as an indirection to call the function pointers specified in s_vpi_extension or the extended vendor specific structure as described above. Note that, as stated earlier, in the case the application is using the built-in routine implementation (i.e. the ones provided by the tool (e.g. simulator) it is running under) then the de-reference through the pointer is not necessary.
PROPOSED: The SystemVerilog tool the user application is running under is responsible for loading the appropriate extension, i.e. the reader API library in the case of the read API. The extension name is used for this purpose, following a specific policy, for example, this extension name can be the name of the library to be loaded. Once the reader API library is loaded all VPI function calls that wish to use the implementation in the library shall be performed using the returned p_vpi_extension pointer as an indirection to call the function pointers specified in s_vpi_extension or the extended vendor specific structure as described above. As stated earlier, in any case that the application is using the built-in routine implementation (i.e. the ones provided by the tool (e.g. simulator) it is running under) then the de-reference through the pointer is not necessary.

32.2 through 32.49 - many of these diagrams have supporting descriptions included in a normative "NOTES" section at the bottom of each section. Globally change "NOTES" to "DETAILS" and change "NOTE" to "DETAIL" to fix the "informative" problem.

WAS: NOTES:
1) **vpiMemory** shall return array variable objects rather than **vpiMemory** objects. The IEEE P1364 standard has made a similar update to the Verilog VPI (refer to note 1 in P1364, 26.6.9)
PROPOSED: DETAILS
1) **vpiMemory** shall return array variable objects rather than **vpiMemory** objects. The IEEE P1364 standard has made a similar update to the Verilog VPI (refer to note 1 in P1364, 26.6.9)

(Wait to see if the P1364 LRM changes before making a proposal)

... (same global solution until section 32.9)

(32.9 under Notes)
WAS: 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 19.2.2 for more details.
PROPOSED: 2) The **vpiImport** iterator shall return all objects imported into the current scope via import statements. Only objects actually referenced through the import shall be returned, rather than items potentially made visible as a result of the import. Refer to 19.2.2 for more details.

... (same global solution until section 32.14)

(32.14 under Notes)
WAS: 19) Note that:
logic var == reg
var bit == reg bit
array var == reg array
PROPOSED: 19) In the above diagram:
logic var == reg
var bit == reg bit
array var == reg array

C.2
WAS: The mailbox class is described in 14.3 and its prototype is:
Note: *dynamic_singular_type* below represents a special type that enables run-time type-checking.
class mailbox
PROPOSED: The mailbox class is described in 14.3 and its prototype is:
The *dynamic_singular_type* below represents a special type that enables run-time type-checking.
class mailbox

E.5
WAS: **E.5 Semantic constraints**
Note that the constraints expressed here merely restate those expressed in 28.4.1.
PROPOSED: *(This is an informational note. No change to the note - except use a smaller font for the note)*

E.5.7
WAS: NOTE—In this last scenario, a block of memory is allocated and freed in C code, even when the standard functions malloc and free are called directly from SystemVerilog code.
PROPOSED: *(This is an informational note. No change to the note)*

E.6.1
WAS: NOTE—The actual argument can have both packed and unpacked parts of an array; either can be multidimensional.
PROPOSED: *(This is an informational note. No change to the note)*

E.6.4
WAS: Note that input mode arguments of type **byte unsigned** and **shortint unsigned** are not equivalent to **bit**[7:0] or **bit**[15:0], respectively, since the former are passed as C types **unsigned char** and **unsigned short** and the latter are both passed by reference as svBitPackedArrRef
PROPOSED: The input mode arguments of type **byte unsigned** and **shortint unsigned** are not equivalent to **bit**[7:0] or **bit**[15:0], respectively, since the former are passed as C types **unsigned char** and **unsigned short** and the latter are both passed by reference as svBitPackedArrRef

---

E.6.6

WAS: 1) If a packed part of an array has more than one dimension, it is linearized as specified by the equivalence of packed types (see E.6.5 and 6.9.3).

2) A packed array of range [L:R] is normalized as [abs(L-R):0]; its most significant bit has a normalized index abs(L-R) and its least significant bit has a normalized index 0.

3) The natural order of elements for each dimension in the layout of an unpacked array shall be used, i.e., elements with lower indices go first. For SystemVerilog range [L:R], the element with SystemVerilog index min(L,R) has the C index 0 and the element with SystemVerilog index max(L,R) has the C index abs(LR).

~~NOTE —~~The above range mapping from SystemVerilog to C applies to calls made in both directions, i.e., SystemVerilog calls to C and C-calls to SystemVerilog.

PROPOSED: 1) If a packed part of an array has more than one dimension, it is linearized as specified by the equivalence of packed types (see E.6.5 and 6.9.3).

2) A packed array of range [L:R] is normalized as [abs(L-R):0]; its most significant bit has a normalized index abs(L-R) and its least significant bit has a normalized index 0.

3) The natural order of elements for each dimension in the layout of an unpacked array shall be used, i.e., elements with lower indices go first. For SystemVerilog range [L:R], the element with SystemVerilog index min(L,R) has the C index 0 and the element with SystemVerilog index max(L,R) has the C index abs(LR).

The above range mapping from SystemVerilog to C applies to calls made in both directions, i.e., SystemVerilog calls to C and C-calls to SystemVerilog.

E.8.1

WAS: Note that all DPI export tasks and functions require that the context of their call is known.

PROPOSED: All DPI export tasks and functions require that the context of their call is known.

E.8.2

WAS: Note that context is transitive through imported and export context tasks and functions declared in the same scope. That is, if an imported task or function is running in a certain context, and if it in turn calls an exported task or function that is available in the same context, the exported task or function can be called without any use of svSetScope(). For example, consider a SystemVerilog call to a native function f(), which in turn calls a native function g(). Now replace the native function f() with an equivalent imported context C function, f'(). The system shall behave identically regardless if f() or f'() is in the call chain above g(). g() has the proper execution context in both cases.

(Modified per email from Doug Warmke - 4/18/2005 - 04:26 PM)

PROPOSED: The context property is transitive through imported and export context tasks and functions declared in the same scope. That is, if an imported task or function is running in a certain context, and if it in turn calls an exported task or function that is available in the same context, the exported task or function can be called without any use of svSetScope(). For example, consider a SystemVerilog call to a native function f(), which in turn calls a native function g(). Now replace the native function f() with an equivalent imported context C function, f'(). The system shall behave identically regardless if f() or f'() is in the call chain above g(). g() has the proper execution context in both cases.

E.8.3 (multiple occurrences)
WAS: To achieve shared data storage, a related set of context imported tasks and functions should all use the same user-Key. To achieve unique data storage, a context import task or function should use a unique key. Note that it is a requirement on the user that such a key be truly unique from all other keys that could possibly be used by C code.
...
PROPOSED: To achieve shared data storage, a related set of context imported tasks and functions should all use the same user-Key. To achieve unique data storage, a context import task or function should use a unique key and it is a requirement on the user that such a key be truly unique from all other keys that could possibly be used by C code.
...

WAS: Note that it is never possible to share user data storage across different contexts. For example, if a Verilog module m declares a context imported task or function f, and m is instantiated more than once in the SystemVerilog design, then f shall execute under different values of svScope.
PROPOSED: It is never possible to share user data storage across different contexts. For example, if a Verilog module m declares a context imported task or function f, and m is instantiated more than once in the SystemVerilog design, then f shall execute under different values of svScope.

E.8.5
WAS: There is no specific relationship defined between DPI and the existing Verilog programming interfaces (VPI and PLI). Programmers must make no assumptions about how DPI and the other interfaces interact. In particular, note that a vpiHandle is not equivalent to an svOpenArrayHandle, and the two must not be interchanged and passed between functions defined in two different interface standards.
PROPOSED: There is no specific relationship defined between DPI and the existing Verilog programming interfaces (VPI and PLI). Programmers must make no assumptions about how DPI and the other interfaces interact. For example, a vpiHandle is not equivalent to an svOpenArrayHandle, and the two must not be interchanged and passed between functions defined in two different interface standards.

E.11
WAS: Formal arguments specified as open arrays allows passing actual arguments of different sizes (i.e., different range and/or different number of elements), which facilitates writing more general C code that can handle SystemVerilog arrays of different sizes. The elements of an open array can be accessed in C by using the same range of indices and the same indexing as in SystemVerilog. Plus, inquiries about the dimensions and the original boundaries of SystemVerilog actual arguments are supported for open arrays.
NOTE—Both packed and unpacked array dimensions can be unsized.
PROPOSED: *(This is an informational note. No change to the note)*

E.11.1
WAS: If a formal argument is specified as a sized array, then it shall be passed by reference, with no overhead, and is directly accessible as a normalized array. If a formal argument is specified as an open (unsized) array, then it shall be passed by handle, with some overhead, and is mostly indirectly accessible, again with some overhead, although it retains the original argument boundaries.
NOTE—This provides some degree of flexibility and allows the programmer to control the trade-off of performance vs. convenience.
PROPOSED: *(This is an informational note. No change to the note)*

E.11.4
WAS: If the actual layout of the SystemVerilog array passed as an argument for an open unpacked array is different than the C layout, then it is not possible to access such an array as a whole; therefore, the address and size of such an array shall be undefined (zero (0), to be exact). Nonetheless, the addresses of individual elements of an array shall be always supported.
NOTE—No specific representation of an array is assumed here; hence, all functions use a generic pointer void *.
PROPOSED: If the actual layout of the SystemVerilog array passed as an argument for an open unpacked array is different than the C layout, then it is not possible to access such an array as a whole; therefore, the address and size of such an array shall be undefined (zero (0), to be exact). Nonetheless, the addresses of individual elements of an array shall be always supported.
PROPOSED: *(This is an informational note. No change to the note)*

G
WAS: — An user might want to switch between selections or provide additional code. This-use case is covered by providing a set of tool switches to define the corresponding information, although it might also use the bootstrap file approach.
NOTE—This annex defines a set of switch names to be used for a particular functionality.
PROPOSED: *(This is an informational note. No change to the note)*

G.2
WAS: The following conditions also apply.
— The compiled object code itself shall be provided in form of a shared library having the appropriate extension for the actual platform.
NOTE—Shared libraries use, for example, .so for Solaris and .sl for HP-UX; other operating systems might use different extensions. In any case, the SystemVerilog application needs to identify the appropriate extension.
PROPOSED: *(This is an informational note. No change to the note)*

H.3
WAS: The semantics of assertions and properties is defined via a relation of satisfaction by empty, finite, and infinite words over the alphabet Σ = 2**P** ∪ {⊤, ⊥}. Such a word is an empty, finite, or infinite sequence of elements of Σ. The number of elements in the sequence is called the *length* of the word, and the length of word *w* is denoted |*w*|. Note that |*w*| is either a non-negative integer or infinity.
PROPOSED: The semantics of assertions and properties is defined via a relation of satisfaction by empty, finite, and infinite words over the alphabet Σ = 2**P** ∪ {⊤, ⊥}. Such a word is an empty, finite, or infinite sequence of elements of Σ. The number of elements in the sequence is called the *length* of the word, and the length of word *w* is denoted |*w*|, where |*w*| is either a non-negative integer or infinity.

Section **12.25**
Classes vs. structs

In 12.25, REPLACE

four fundamental ways

WITH

~~four~~ three fundamental ways

In 12.25, DELETE bullet 2 and renumber bullets 3 and 4 to 2 and 3 respectively.

Section **5.14.2.2-7**
queue method prototypes

In 5.14.2.2, REPLACE

```
    function void insert(input int index, ref queue_t
queue, input element_t item);
```

WITH

```
    function void insert(input int index, ref queue_t
queue, input element_t item);
```

In 5.14.2.4, REPLACE

```
    function queue_type pop_front();
```

WITH

```
    function queue_type element_t pop_front();
```

In 5.14.2.5, REPLACE

```
    function queue_type pop_back();
```

WITH

```
    function queue_type element_t pop_back();
```

In 5.14.2.6, REPLACE

```
    function void push_front(ref queue_t queue, input
element_t item);
```

WITH

```
    function void push_front(ref queue_t queue, input
element_t item);
```

In 5.14.2.7, REPLACE

```
    function void push_back(ref queue_t queue, input
element_t item);
```

WITH

```
    function void push_back(ref queue_t queue, input
element_t item);
```

Section **11.4.5, A.8.2, 1.2**
Function calls – omission of parentheses

In 11.4.5, REPLACE

When a task or function specifies no arguments, the empty parenthesis, (), following the task/function name shall be optional. This is also true for tasks or functions that require arguments, when all arguments have defaults specified.

WITH

When a ~~task or function~~ void function or class function method specifies no arguments, the empty parenthesis, (), following the ~~task/function~~ subroutine name shall be optional. This is also true for ~~tasks or functions~~ tasks, void functions, and class methods that require arguments, when all arguments have defaults specified. It shall be illegal to omit the parenthesis in a directly recursive non-void function method call that is not hierarchically qualified.

In A.8.2, in tf_call, ADD the following footnote

It shall be illegal to omit the parentheses in a tf_call unless the subroutine is a task, void function or class method. If the subroutine is a non-void class function method, it shall be illegal to omit the parentheses if the call is directly recursive.

In 1.2, REPLACE

C like

WITH

C-like

Section

Section **4,6, 4.11**
Union members

In 4.6, REPLACE

— Chandles shall not be used:
— In any expression other than as permitted above
— As ports
— In sensitivity lists or event expressions
— In continuous assignments
— In unions
— In packed types

WITH

— Chandles shall not be used:
— In any expression other than as permitted above
— As ports
— In sensitivity lists or event expressions
— In continuous assignments
— In untagged unions
— In packed types

In 4.11, REPLACE

Thus, it is impossible to store a value of one type and (mis)interpret the bits as another type.

WITH

Thus, it is impossible to store a value of one type and (mis)interpret the bits as another type.

Dynamic types and chandles shall not be used in untagged unions, but may be used in tagged unions.

Section

Modify the text in 17.1

2. It creates a scope that encapsulates program-wide data and procedures.

Add to the end of syntax box 17-1

anonymous_program ::= **program ;** { anonymous_program_item } **endprogram**
anonymous_program_item ::=
task_declaration
| function_declaration
| class_declaration
| covergroup_declaration
| class_constructor_declaration
| **;**

Delete 17.3

~~**17.3 Multiple programs**~~
~~It is allowed to have any arbitrary number of program definitions or instances. The programs can be fully independent (without inter-program communication), or cooperative. The degree of communication can be controlled by choosing to share data using nested blocks, packages, or hierarchical references, or making the data private by declaring it inside the corresponding program block.~~

Add a new section 17.6 before Program control tasks

17.6 Program-wide space and Anonymous programs

The set of program definitions and instances define a space of program-wide data and procedures that is accessible only to programs.

Anonymous programs can be used inside packages or compilation unit scopes (See 19.2 and 19.3) to declare items that are part of the program-wide space without declaring a new scope. Items declared in an anonymous program share the same namespace as the package or compilation unit scope in which they are declared, but add the same semantic restrictions imposed on a program block.

Note: Although identifiers declared inside an anonymous program cannot be referenced outside any program block, attempting to declare another identifier with the same name outside the anonymous program block will generate an error. This occurs because the identifier shares the same namespace within the scope of the surrounding package or compilation unit.

Section **13.13.1**
Root threads

In 13.13.1, REPLACE

All RNG’s can be manually seeded. Combined with hierarchical seeding, this facility allows users to define the operation of a subsystem (hierarchy subtree) completely with a single seed at the root thread of the system.

WITH

All RNG's can be manually seeded. Combined with hierarchical seeding, this facility allows users to define the operation of a subsystem (hierarchy subtree) completely with a single seed at the root thread of the subsystem."

Section 13.13.1 Random stability properties

ADD the blue text.

Random stability encompasses the following properties:

— Initialization RNG

Each module instance, interface instance, program instance, and package has an initialization RNG. Each initialization RNG is seeded with the *default seed*. The *default seed* is an implementation dependent value. An initialization RNG shall be used in the creation of static threads and static initializers (see the following bullets).

— Thread stability

Each thread has an independent RNG for all randomization system calls invoked from that thread. When a new dynamic thread is created, its RNG is seeded with the next random value from its parent thread. This property is called *hierarchical seeding*. When a static thread is created, its RNG is seeded with the next value from the initialization RNG of the module instance, interface instance, program instance or package containing the thread declaration.

Program and thread stability is guaranteed as long as thread creation and random number generation is done in the same order as before. When adding new threads to an existing test, they can be added at the end of a code block in order to maintain random number stability of previously created work.

— Object stability

Each class instance (object) has an independent RNG for all randomization methods in the class. When an object is created using **new**, its RNG is seeded with the next random value from the thread that creates the object. When a class object is created by a static declaration initializer, there is no active thread, thus, the RNG of the created object is seeded with the next random value of the initialization RNG of the module instance, interface instance, program instance or package in which the declaration occurred.

Object stability is guaranteed as long as object and thread creation, as well as random number generation, are done in the same order as before. In order to maintain random number stability, new objects, threads and random numbers can be created after existing objects are created.

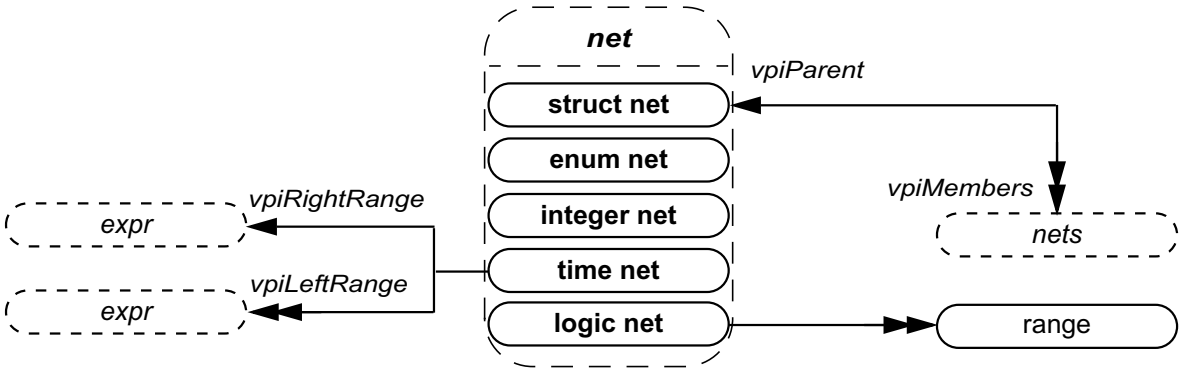
— Manual seeding

All non-initialization RNG's can be manually seeded. Combined with hierarchical seeding, this facility allows users to define the operation of a subsystem (hierarchy subtree) completely with a single seed at the root thread of the subsystem.

REPLACE:

32.13 Nets (supersedes P1364 26.6.6)

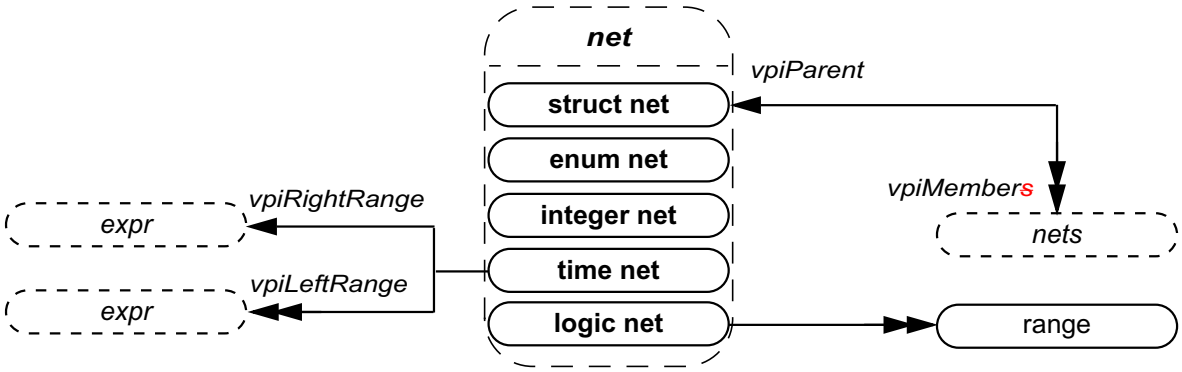
-> access by index
vpi_handle_by_index
vpi_handle_by_multi_index
-> array member
bool: vpiArray (deprecated)
bool: vpiArrayMember
-> delay
vpi_get_delays()
-> expanded
bool: vpiExpanded
-> implicitly declared
bool: vpiImplicitDecl
-> name
str: vpiName
str: vpiFullName
-> net decl assign
bool: vpiNetDeclAssign
-> net type
int: vpiNetType
int: vpiResolvedNetType
-> scalar
bool: vpiScalar
-> scalared declaration
bool: vpiExplicitScalared
-> sign
bool: vpiSigned
-> size
int: vpiSize
-> strength
int: vpiStrength0
int: vpiStrength1
int: vpiChargeStrength
-> value
vpi_get_value()
vpi_put_value()
-> vector
bool: vpiVector
-> vectored declaration
bool: vpiExplicitVectored



WITH:

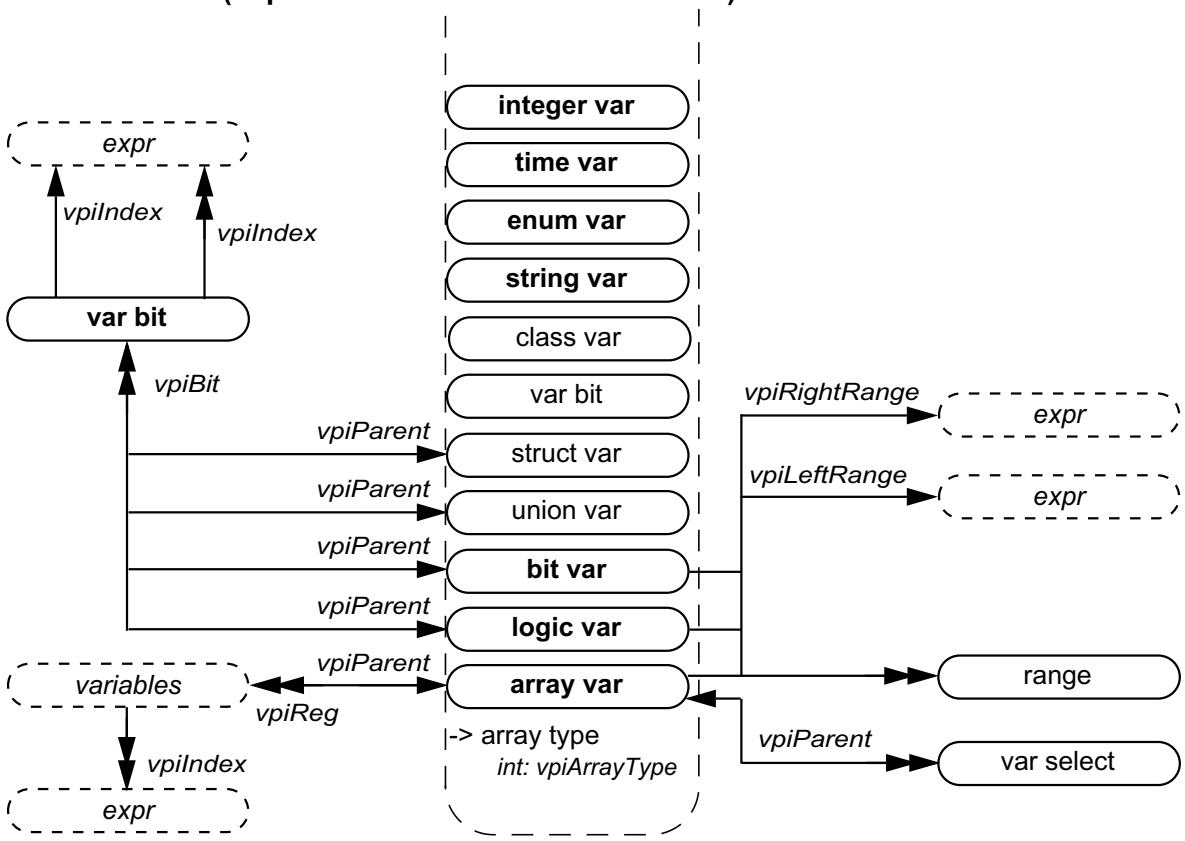
32.13 Nets (supersedes P1364 26.6.6)

-> access by index
  vpi_handle_by_index
  vpi_handle_by_multi_index
-> array member
  bool: vpiArray (deprecated)
  bool: vpiArrayMember
-> delay
  vpi_get_delays()
-> expanded
  bool: vpiExpanded
-> implicitly declared
  bool: vpiImplicitDecl
-> name
  str: vpiName
  str: vpiFullName
-> net decl assign
  bool: vpiNetDeclAssign
-> net type
  int: vpiNetType
  int: vpiResolvedNetType
-> scalar
  bool: vpiScalar
-> scalared declaration
  bool: vpiExplicitScalared
-> sign
  bool: vpiSigned
-> size
  int: vpiSize
-> strength
  int: vpiStrength0
  int: vpiStrength1
  int: vpiChargeStrength
-> value
  vpi_get_value()
  vpi_put_value()
-> vector
  bool: vpiVector
-> vectored declaration
  bool: vpiExplicitVectored
-> member
  bool: vpiStructUnionMember



REPLACE:

32.14 Variables (supersedes P1364 26.6.7 and 26.6.8)



-> array member
bool: vpiArray (deprecated)
bool: vpiArrayMember
-> name
str: vpiName
str: vpiFullName
-> sign
bool: vpiSigned
-> size
int: vpiSize

-> array type
int: vpiArrayType
-> lifetime
bool: vpiAutomatic
-> constant variable
bool: vpiConstantVariable
-> determine random availability
bool: vpiIsRandomized
-> randomization type
int: vpiRandType

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

NOTES

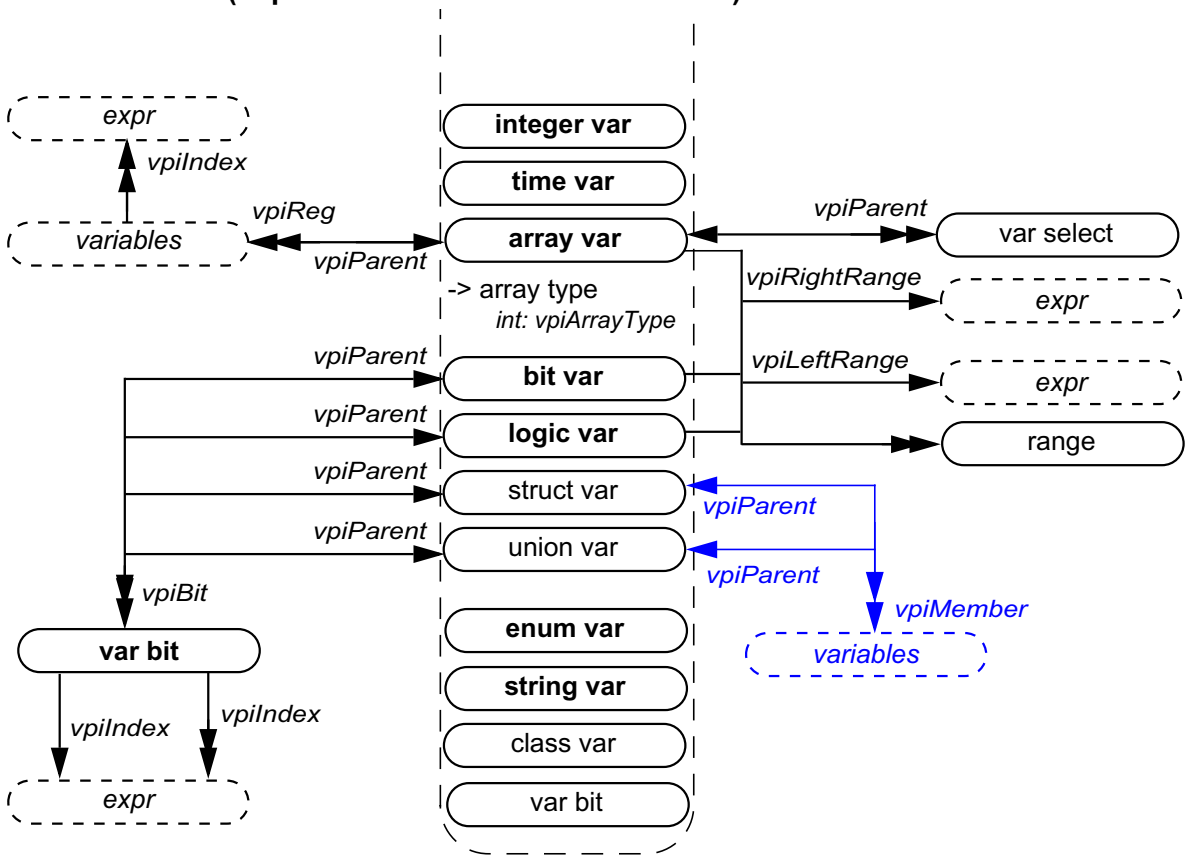
16) **vpiIsRandomized** is a property to determine whether a random variable is currently active for randomization.

17) When the **vpiMember** property is TRUE, it indicates that the variable is a member of a parent struct or union variable. See also the relations in 32.18

18) If a variable is an element of an array (the **vpiArrayMember** property is TRUE), the **vpiIndex** iterator shall return the indexing expressions that select that specific variable out of the array.

WITH:

32.14 Variables (supersedes P1364 26.6.7 and 26.6.8)



-> array member
bool: vpiArray (deprecated)
bool: vpiArrayMember

-> name
str: vpiName
str: vpiFullName

-> sign
bool: vpiSigned

-> size
int: vpiSize

~~-> array type~~
~~*int: vpiArrayType*~~

-> lifetime
bool: vpiAutomatic

-> constant variable
bool: vpiConstantVariable

-> determine random availability
bool: vpiIsRandomized

-> randomization type
int: vpiRandType

-> member
bool: vpiStructUnionMember

->value
vpi_get_value()
vpi_put_value()

-> scalar
bool: vpiScalar

-> visibility
int: vpiVisibility

-> vector
bool: vpiVector

NOTES

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

17) When the vpiStructUnionMember property is TRUE, it indicates that the variable is a member of a parent struct or union variable. See also the relations in 32.18

18) If a variable is an element of an array (the **vpiArrayMember** property is TRUE), the **vpiIndex** iterator shall return the indexing expressions that select that specific variable out of the array.

REPLACE:

Annex I

```
/************ methods used to traverse 1 to many relationships ************/
#define vpiTypedef                725
#define vpiImport                 726
#define vpiDerivedClasses         727
#define vpiMethods                730
#define vpiSolveBefore            731
#define vpiSolveAfter             732
#define vpiWaitingProcesses       734
#define vpiMessages               735
#define vpiLoopVars               737
#define vpiConcurrentAssertions   740
#define vpiMatchItem              741
/************ methods both 1-1 and 1-many relations ***********************/
#define vpiInstance               745
/**************************************************************************/
/************************ generic object properties ***********************/
/**************************************************************************/
#define vpiTop                    600
#define vpiUnit                   602
#define vpiAccessType             604
#define vpiForkJoinAcc          1
#define vpiExternAcc            2
#define vpiDPIExternAcc         3
#define vpiDPIImportAcc         4
#define vpiArrayType              606
#define vpiStaticArray          1
#define vpiDynamicArray         2
#define vpiAssocArray           3
#define vpiQueueArray           4
#define vpiArrayMember            607
#define vpiIsRandomized           608
#define vpiRandType               610
#define vpiNotRand              1
#define vpiRand                 2
#define vpiRandC                3
#define vpiConstantVariable       612
#define vpiMember                 615
#define vpiVisibility             620
#define vpiPublicVis            1
#define vpiProtectedVis         2
#define vpiLocalVis             3
#define vpiNullConst              622
#define vpiOneStepConst           623
#define vpiAlwaysType             624
#define vpiAlwaysComb           1
#define vpiAlwaysFF             2
#define vpiAlwaysLatch          3
#define vpiDistType               625
#define vpiEqualDist            1 /* constraint equal distribution */
#define vpiDivDist              2 /* constraint divided distribution */
#define vpiPacked                 630
#define vpiTagged                 632
#define vpiRef                    633
#define vpiDefaultSkew            634
#define vpiVirtual                635
```

```
#define vpiUserDefined            636
#define vpiIsConstraintEnabled    638
#define vpiClassType              640
#define vpiMailboxClass        1
#define vpiSemaphoreClass      2
#define vpiUserDefineClass     3
#define vpiMethod                 645
#define vpiIsClockInferred        649
#define vpiQualifier              650
#define vpiNoQualifier         0
#define vpiUniqueQualifier     1
#define vpiPriorityQualifier   2
#define vpiTaggedQualifier     4
#define vpiRandQualifier       8
```

WITH:

Annex I

```
/************ methods used to traverse 1 to many relationships ************/
#define vpiTypedef                725
#define vpiImport                 726
#define vpiDerivedClasses         727
#define vpiMethods                730
#define vpiSolveBefore            731
#define vpiSolveAfter             732
#define vpiWaitingProcesses       734
#define vpiMessages               735
#define vpiLoopVars               737
#define vpiConcurrentAssertions   740
#define vpiMatchItem              741
#define vpiMember                 746
/************ methods both 1-1 and 1-many relations ***********************/
#define vpiInstance               745
/**************************************************************************/
/************************ generic object properties ***********************/
/**************************************************************************/
#define vpiTop                    600
#define vpiUnit                   602
#define vpiAccessType             604
#define vpiForkJoinAcc         1
#define vpiExternAcc           2
#define vpiDPIExternAcc        3
#define vpiDPIImportAcc        4
#define vpiArrayType              606
#define vpiStaticArray         1
#define vpiDynamicArray        2
#define vpiAssocArray          3
#define vpiQueueArray          4
#define vpiArrayMember            607
#define vpiIsRandomized           608
#define vpiRandType               610
#define vpiNotRand             1
#define vpiRand                2
#define vpiRandC               3
#define vpiConstantVariable       612
#define vpiStructUnionMember      615
#define vpiVisibility             620
#define vpiPublicVis           1
```

```
#define vpiProtectedVis         2
#define vpiLocalVis             3
#define vpiNullConst              622
#define vpiOneStepConst           623
#define vpiAlwaysType             624
#define vpiAlwaysComb           1
#define vpiAlwaysFF             2
#define vpiAlwaysLatch          3
#define vpiDistType               625
#define vpiEqualDist            1 /* constraint equal distribution */
#define vpiDivDist              2 /* constraint divided distribution */
#define vpiPacked                 630
#define vpiTagged                 632
#define vpiRef                    633
#define vpiDefaultSkew            634
#define vpiVirtual                635
#define vpiUserDefined            636
#define vpiIsConstraintEnabled    638
#define vpiClassType              640
#define vpiMailboxClass         1
#define vpiSemaphoreClass       2
#define vpiUserDefineClass      3
#define vpiMethod                 645
#define vpiIsClockInferred        649
#define vpiQualifier              650
#define vpiNoQualifier          0
#define vpiUniqueQualifier      1
#define vpiPriorityQualifier    2
#define vpiTaggedQualifier      4
#define vpiRandQualifier        8
```

Proposal for ballot issue 10, Mantis item 409:

In 11.4.2, AFTER:

Only variables, not nets, can be passed by reference.

ADD new paragraph:

Since a variable passed by reference may be an automatic variable, a **ref** argument shall not be used in any context forbidden for automatic variables.

Section **8.19**
inside operator

In 8.19, REPLACE

Integral expressions also use the equality operator, except that a z inside a value in the set is treated as a don’t care and that bit position shall not be considered. Note that unlike comparisons performed by the **casez** statement, z values in the expression on the left-hand side are not treated as a don’t-care; the don’t-care is unidirectional.

WITH

~~Integral expressions also use the equality operator, except that a z inside a value in the set is treated as a don’t care and that bit position shall not be considered. Note that unlike comparisons performed by the **casez** statement, z values in the expression on the left-hand side are not treated as a don’t-care; the don’t-care is unidirectional.~~ Integral expressions use the wildcard equality (==?) operator, so that an x or z bit in a value in the set is treated as a don't-care in that bit position (see 8.5). As with wildcard equality, an x or z in the expression on the left-hand side of the inside operator is not treated as a don't-care.

29.3.2.1 Using `vpi_get_assertion_info`

CHANGE:

29.3.2.1 Using `vpi_get_assertion_info`

Static information can be obtained directly from an assertion handle by using `vpi_get_assertion_info()`, as shown below.

```
typedef struct t_vpi_source_info {
   PLI_BYTE8 *fileName;
   PLI_INT32 startLine;
   PLI_INT32 startColumn;
   PLI_INT32 endLine;
   PLI_INT32 endColumn;
} s_vpi_source_info, *p_vpi_source_info;
typedef struct t_vpi_assertion_info {
   PLI_BYTE8 *assertName; /* name of assertion */
   vpiHandle instance; /* instance containing assertion */
   PLI_BYTE8 defname; /* name of module/interface containing assertion */
   vpiHandle clock; /* clocking expression */
   PLI_INT32 assertionType; /* vpiSequenceInst, vpiAssert, vpiAssume,
                               vpiCover, */
                            /* vpiPropertyInst, vpiImmediateAssert */
   s_vpi_source_info sourceInfo;
} s_vpi_assertion_info, *p_vpi_assertion_info;
PLI_INT32 vpi_get_assertion_info (assert_handle, p_vpi_assertion_info);
```

This call obtains all the static information associated with an assertion.

The inputs are a valid handle to an assertion and a pointer to an existing `s_vpi_assertion_info` data structure. On success, the function returns `TRUE` and the `s_vpi_assertion_info` data structure is filled in as appropriate. On failure, the function returns `FALSE` and the contents of the assertion data structure are unpredictable.

Assertions can occur in modules and interfaces: for assertions defined in modules, the instance field in the `s_vpi_assertion_info` structure shall contain the handle to the appropriate module or interface instance. Note that VPI does not currently define the information model for interfaces and therefore the interface instance handle shall be implementation dependent. The clock field of that structure contains a handle to the event expression representing the clock for the assertion, as determined by 18.14.

NOTE: a single call returns all the information for efficiency reasons.

TO:

~~**29.3.2.1 Using** `vpi_get_assertion_info`~~

~~Static information can be obtained directly from an assertion handle by using `vpi_get_assertion_info()`, as shown below.~~

```
typedef struct t_vpi_source_info {
   PLI_BYTE8 *fileName;
   PLI_INT32 startLine;
   PLI_INT32 startColumn;
   PLI_INT32 endLine;
   PLI_INT32 endColumn;
} s_vpi_source_info, *p_vpi_source_info;
typedef struct t_vpi_assertion_info {
   PLI_BYTE8 *assertName; /* name of assertion */
   vpiHandle instance; /* instance containing assertion */
```

```
   PLI_BYTE8 defname; /* name of module/interface containing assertion */
   vpiHandle clock; /* clocking expression */
   PLI_INT32 assertionType; /* vpiSequenceInst, vpiAssert, vpiAssume,
                               vpiCover, */
                            /* vpiPropertyInst, vpiImmediateAssert */
   s_vpi_source_info sourceInfo;
} s_vpi_assertion_info, *p_vpi_assertion_info;
PLI_INT32 vpi_get_assertion_info (assert_handle, p_vpi_assertion_info);
```

This call obtains all the static information associated with an assertion.

The inputs are a valid handle to an assertion and a pointer to an existing `s_vpi_assertion_info` data structure. On success, the function returns `TRUE` and the `s_vpi_assertion_info` data structure is filled in as appropriate. On failure, the function returns `FALSE` and the contents of the assertion data structure are unpredictable.

Assertions can occur in modules and interfaces: for assertions defined in modules, the instance field in the `s_vpi_assertion_info` structure shall contain the handle to the appropriate module or interface instance. Note that VPI does not currently define the information model for interfaces and therefore the interface instance handle shall be implementation dependent. The clock field of that structure contains a handle to the event expression representing the clock for the assertion, as determined by 18.14.

NOTE: a single call returns all the information for efficiency reasons.

29.3.2.2 Extending `vpi_get()` and `vpi_get_str()`

CHANGE

In addition to `vpi_get_assertion_info`, the following existing VPI functions are also extended:

TO

In addition to `vpi_get_assertion_info`,tThe following existing VPI functions are also extended:

29.4.2 Placing assertions callbacks

CHANGE

Use `vpi_register_assertion_cb()` to place an assertion callback; the prototype is:

```
/* typedef for vpi_register_assertion_cb callback function */
typedef  PLI_INT32 (vpi_assertion_callback_func)(
   PLI_INT32 reason,          /* callback reason */
   p_vpi_time cb_time,        /* callback time */
   vpiHandle assertion,       /* handle to assertion */
   p_vpi_attempt_info info,   /* attempt related information */
   PLI_BYTE8 *user_data       /* user data entered upon registration */
);

vpiHandle vpi_register_assertion_cb(
   vpiHandle assertion,       /* handle to assertion */
   PLI_INT32 reason,          /* reason for which callbacks needed */
   vpi_assertion_callback_func *cb_rtn,
   PLI_BYTE8 *user_data       /* user data to be supplied to cb */
);
typedef struct t_vpi_assertion_step_info {
   PLI_INT32 matched_expression_count;
   vpiHandle *matched_exprs;  /* array of expressions */
   p_vpi_source_info *exprs_source_info; /* array of source info */
   PLI_INT32 stateFrom, stateTo;/* identify transition */
```

```
} s_vpi_assertion_step_info, *p_vpi_assertion_step_info;
typedef struct t_vpi_attempt_info {
   union {
      vpiHandle failExpr;
      p_vpi_assertion_step_info step;
   } detail;
   s_vpi_time attemptStartTime;  /* Time attempt triggered */
} s_vpi_attempt_info, *p_vpi_attempt_info;
```

TO

Use `vpi_register_assertion_cb()` to place an assertion callback; the prototype is:

```
/* typedef for vpi_register_assertion_cb callback function */
typedef  PLI_INT32 (vpi_assertion_callback_func)(
   PLI_INT32 reason,          /* callback reason */
   p_vpi_time cb_time,        /* callback time */
   vpiHandle assertion,       /* handle to assertion */
   p_vpi_attempt_info info,   /* attempt related information */
   PLI_BYTE8 *user_data       /* user data entered upon registration */
);

vpiHandle vpi_register_assertion_cb(
   vpiHandle assertion,       /* handle to assertion */
   PLI_INT32 reason,          /* reason for which callbacks needed */
   vpi_assertion_callback_func *cb_rtn,
   PLI_BYTE8 *user_data       /* user data to be supplied to cb */
);

typedef struct t_vpi_assertion_step_info {
   PLI_INT32 matched_expression_count;
   vpiHandle *matched_exprs;  /* array of expressions */
   p_vpi_source_info *exprs_source_info; /* array of source info */
   PLI_INT32 stateFrom, stateTo;/* identify transition */
} s_vpi_assertion_step_info, *p_vpi_assertion_step_info;

typedef struct t_vpi_attempt_info {
   union {
      vpiHandle failExpr;
      p_vpi_assertion_step_info step;
   } detail;
   s_vpi_time attemptStartTime;  /* Time attempt triggered */
} s_vpi_attempt_info, *p_vpi_attempt_info;
```

31.10 Reading data from multiple databases and/or different read library providers

CHANGE

The structure shall have an entry for *every* VPI routine; the order and synopsis of these entries within the structure shall exactly match the order and synopsis of function definitions in Clause 27 of the P1364 Verilog standard. After those entries the SystemVerilog VPI routine additions for assertions `vpi_get_assertion_info()` and then `vpi_register_assertion_cb()` shall be added in that order. Then all new reader routines defined in Table shall be added in exactly the order noted in the table. If a particular extension does not support a specific VPI routine, then it shall still have an entry (with the correct prototype), and a dummy body that shall always have a return (consistent with the VPI prototype) to signify failure (i.e. `NULL` or `FALSE`). The routine call must also raise the appropriate VPI error, which can be checked by `vpi_chk_error()`, and/or automatically generate

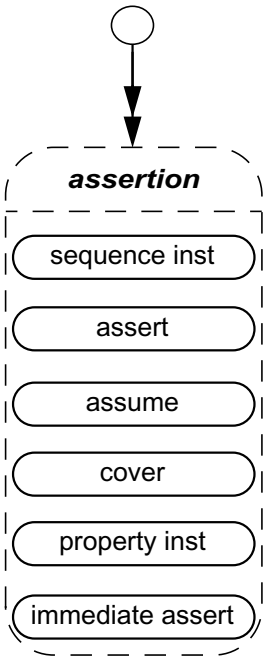
an error message in a manner consistent with the specific VPI routine.

TO

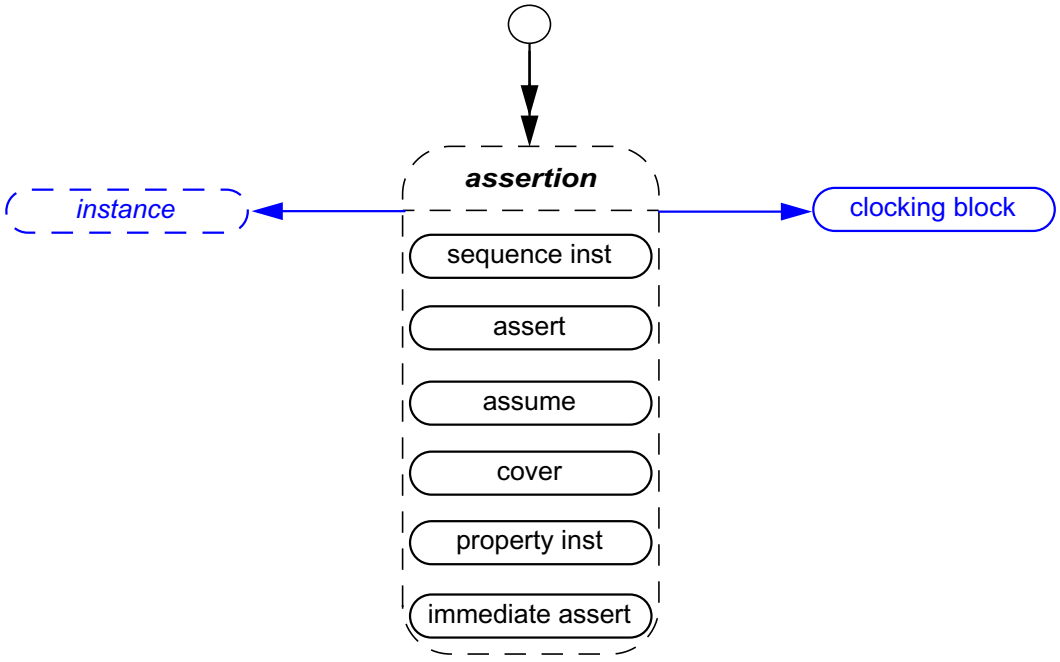
The structure shall have an entry for *every* VPI routine; the order and synopsis of these entries within the structure shall exactly match the order and synopsis of function definitions in Clause 27 of the P1364 Verilog standard. After those entries the SystemVerilog VPI routine ~~additions for assertions~~ ~~`vpi_get_assertion_info()`~~ ~~and then~~ `vpi_register_assertion_cb()` shall be added ~~in that order~~. Then all new reader routines defined in Table shall be added in exactly the order noted in the table. If a particular extension does not support a specific VPI routine, then it shall still have an entry (with the correct prototype), and a dummy body that shall always have a return (consistent with the VPI prototype) to signify failure (i.e. `NULL` or `FALSE`). The routine call must also raise the appropriate VPI error, which can be checked by `vpi_chk_error()`, and/or automatically generate an error message in a manner consistent with the specific VPI routine.

32.31 Assertion

CHANGE



TO



-> assertion type
int: vpiAssertType

-> location
str: vpiFile
int: vpiStartLine
int: vpiColumn
int: vpiEndLine
int: vpiEndColumn

-> assertion name
str: vpiName

Annex A

(normative)

sv_vpi_user.h

CHANGE

```
/* assertion related structs and types */
typedef struct t_vpi_source_info {
   PLI_BYTE8 *fileName;
   PLI_INT32 startLine;
   PLI_INT32 startColumn;
   PLI_INT32 endLine;
   PLI_INT32 endColumn;
} s_vpi_source_info, *p_vpi_source_info;

typedef struct t_vpi_assertion_info {
   PLI_BYTE8 *assertName;           /* name of assertion */
   vpiHandle instance;              /* instance containing assertion */
   PLI_BYTE8 defname;               /* name of module/interface containing
                                     * the assertion */
   vpiHandle clock;                 /* clocking expression */
   PLI_INT32 assertionType;         /* vpiSequenceInst, vpiAssert, vpiAssume,
                                     * vpiCover, vpiPropertyInst,
                                     * vpiImmediateAssert */
   s_vpi_source_info sourceInfo;
} s_vpi_assertion_info, *p_vpi_assertion_info;

typedef struct t_vpi_assertion_step_info {
   PLI_INT32 matched_expression_count;
   vpiHandle *matched_exprs;                /* array of expressions */
   p_vpi_source_info *exprs_source_info;    /* array of source info */
   PLI_INT32 stateFrom, stateTo;            /* identify transition */
} s_vpi_assertion_step_info, *p_vpi_assertion_step_info;
```

TO

```
/* assertion related structs and types */
typedef struct t_vpi_source_info {
   PLI_BYTE8 *fileName;
   PLI_INT32 startLine;
   PLI_INT32 startColumn;
   PLI_INT32 endLine;
   PLI_INT32 endColumn;
} s_vpi_source_info, *p_vpi_source_info;

typedef struct t_vpi_assertion_info {
   PLI_BYTE8 *assertName;           /* name of assertion */
   vpiHandle instance;              /* instance containing assertion */
   PLI_BYTE8 defname;               /* name of module/interface containing
                                     * the assertion */
   vpiHandle clock;                 /* clocking expression */
   PLI_INT32 assertionType;         /* vpiSequenceInst, vpiAssert, vpiAssume,
                                     * vpiCover, vpiPropertyInst,
                                     * vpiImmediateAssert */
```

```
   s_vpi_source_info sourceInfo;
} s_vpi_assertion_info, *p_vpi_assertion_info;

typedef struct t_vpi_assertion_step_info {
   PLI_INT32 matched_expression_count;
   vpiHandle *matched_exprs;                /* array of expressions */
   p_vpi_source_info *exprs_source_info;    /* array of source info */
   PLI_INT32 stateFrom, stateTo;            /* identify transition */
} s_vpi_assertion_step_info, *p_vpi_assertion_step_info;
```

CHANGE

```
/* typedef for vpi_register_assertion_cb callback function */
typedef PLI_INT32(vpi_assertion_callback_func)(
   PLI_INT32 reason,                /* callback reason */
   p_vpi_time cb_time,              /* callback time */
   vpiHandle assertion,             /* handle to assertion */
   p_vpi_attempt_info info,         /* attempt related information */
   PLI_BYTE8 *user_data             /* user data entered upon registration */
);

/* assertion specific VPI functions */
PLI_INT32 vpi_get_assertion_info(
   vpiHandle assertion,             /* handle to assertion */
   p_vpi_assertion_info info        /* assertion related information */
);

vpiHandle vpi_register_assertion_cb(
   vpiHandle assertion,             /* handle to assertion */
   PLI_INT32 reason,                /* reason for which callbacks needed */
   vpi_assertion_callback_func *cb_rtn,
   PLI_BYTE8 *user_data             /* user data to be supplied to cb */
);
```

TO

```
/* typedef for vpi_register_assertion_cb callback function */
typedef PLI_INT32(vpi_assertion_callback_func)(
   PLI_INT32 reason,                /* callback reason */
   p_vpi_time cb_time,              /* callback time */
   vpiHandle assertion,             /* handle to assertion */
   p_vpi_attempt_info info,         /* attempt related information */
   PLI_BYTE8 *user_data             /* user data entered upon registration */
);

/* assertion specific VPI functions */
PLI_INT32 vpi_get_assertion_info(
   vpiHandle assertion,             /* handle to assertion */
   p_vpi_assertion_info info        /* assertion related information */
);

vpiHandle vpi_register_assertion_cb(
   vpiHandle assertion,             /* handle to assertion */
   PLI_INT32 reason,                /* reason for which callbacks needed */
   vpi_assertion_callback_func *cb_rtn,
   PLI_BYTE8 *user_data             /* user data to be supplied to cb */
);
```

Clarification on thread and frame

Some of the NOTES in this proposal already exist in SVDB #445.
The NOTES in this proposal supersedes the NOTES in proposal #445 both in numbering and in content.

In 32.28 MODIFY the NOTES as follows:

NOTES:

1) Frames correspond to the set of automatic variables declared in a given task or function
2) The following callbacks shall be supported on frames:
   — **cbStartOfFrame**: triggers whenever any frame gets executed.
   — **cbEndOfFrame**: triggers when a particular thread frame is deleted after all storage is deleted.
3) It shall be illegal to place value change callbacks on automatic variables.
4) It shall be illegal to put a value with a delay on automatic variables.
5) There is at most only one active frame at any time in a given thread. To get a handle to the currently active frame, use **vpi_handle(vpiFrame, NULL)**. The **frame** to **stmt** transition shall return the currently active statement within the frame.
6) Frame handles must be freed using **vpi_free_object()** once the application no longer needs the handle. If the handle is not freed it shall continue to exist, even after the frame has completed execution.
7) The Frame object model is not backwards compatible with IEEE 1364-2005.

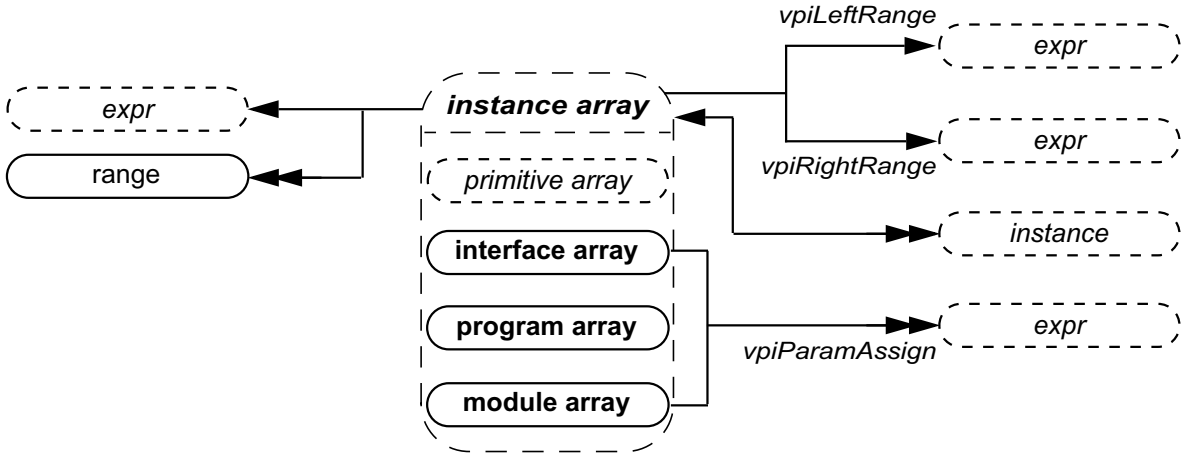
In 32.29 MODIFY the NOTES as follows:

NOTES:

— A thread is a Verilog process such as an **always** block or a branch of a **fork** construct. As a thread works its way down a call chain of tasks and/or functions, a new frame is activated as each new task or function is entered.
— 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

REPLACE

32.8 Instance arrays (supersedes P1364 26.6.2)



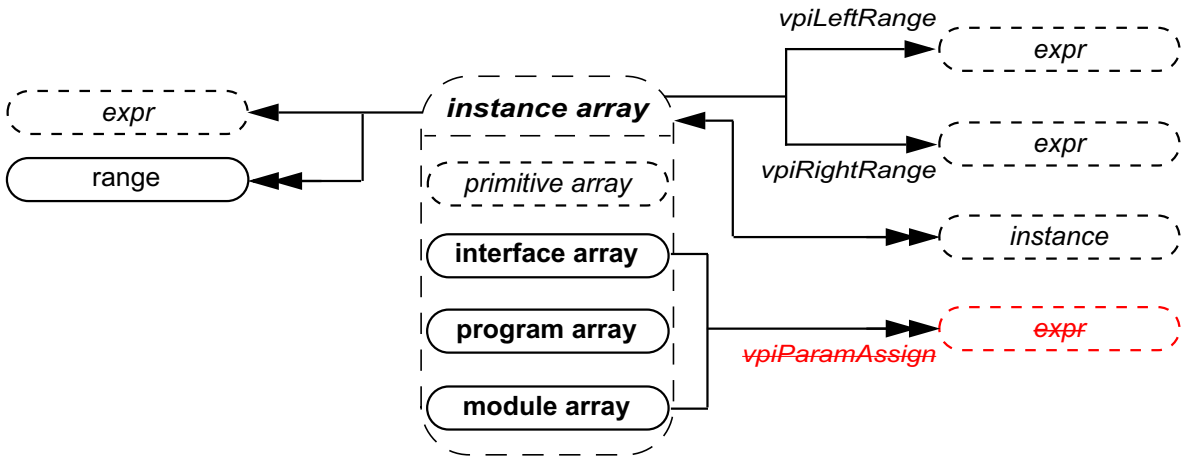
NOTES:

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

2) To obtain all the dimensions of a multidimensional array, the range iterator must be used. Using the **vpiLeftRange**/**vpiRightRange** properties only returns the last dimension of a multidimensional array.

DELETE THE FOLLOWING

32.8 Instance arrays (supersedes P1364 26.6.2)



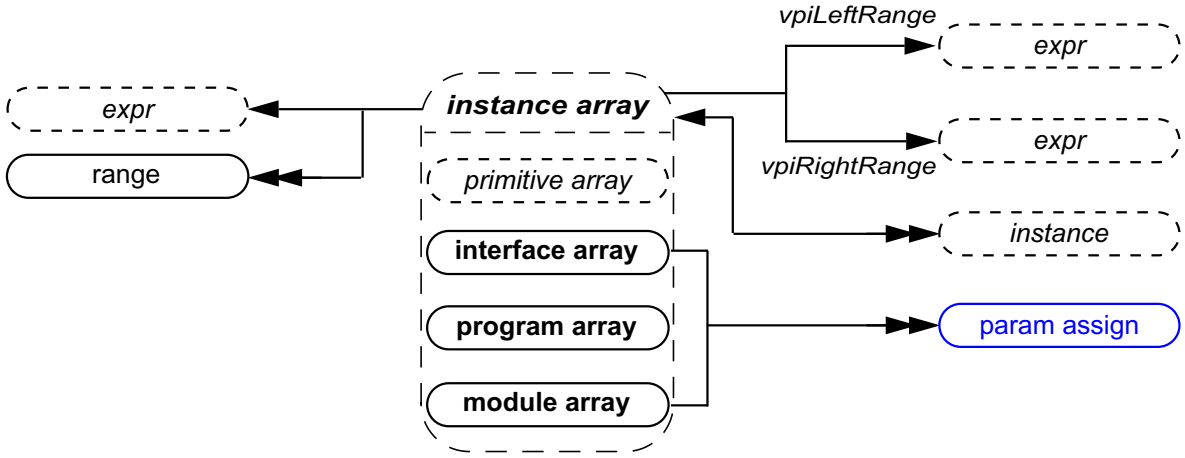
NOTES:

~~1) Param assignments can only be obtained from non-primitive instance arrays.~~

2) To obtain all the dimensions of a multidimensional array, the range iterator must be used. Using the **vpiLeftRange**/**vpiRightRange** properties only returns the last dimension of a multidimensional array.

THEN ADD THE FOLLOWING:

32.8 Instance arrays (supersedes P1364 26.6.2)

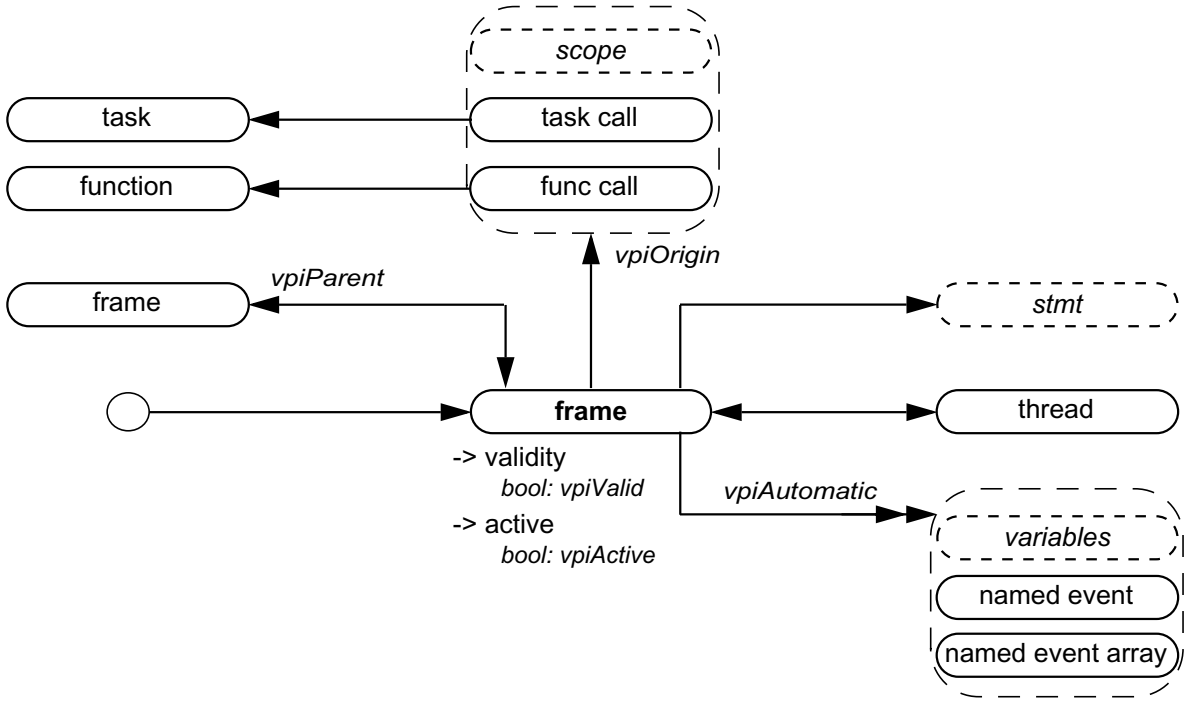


NOTES:

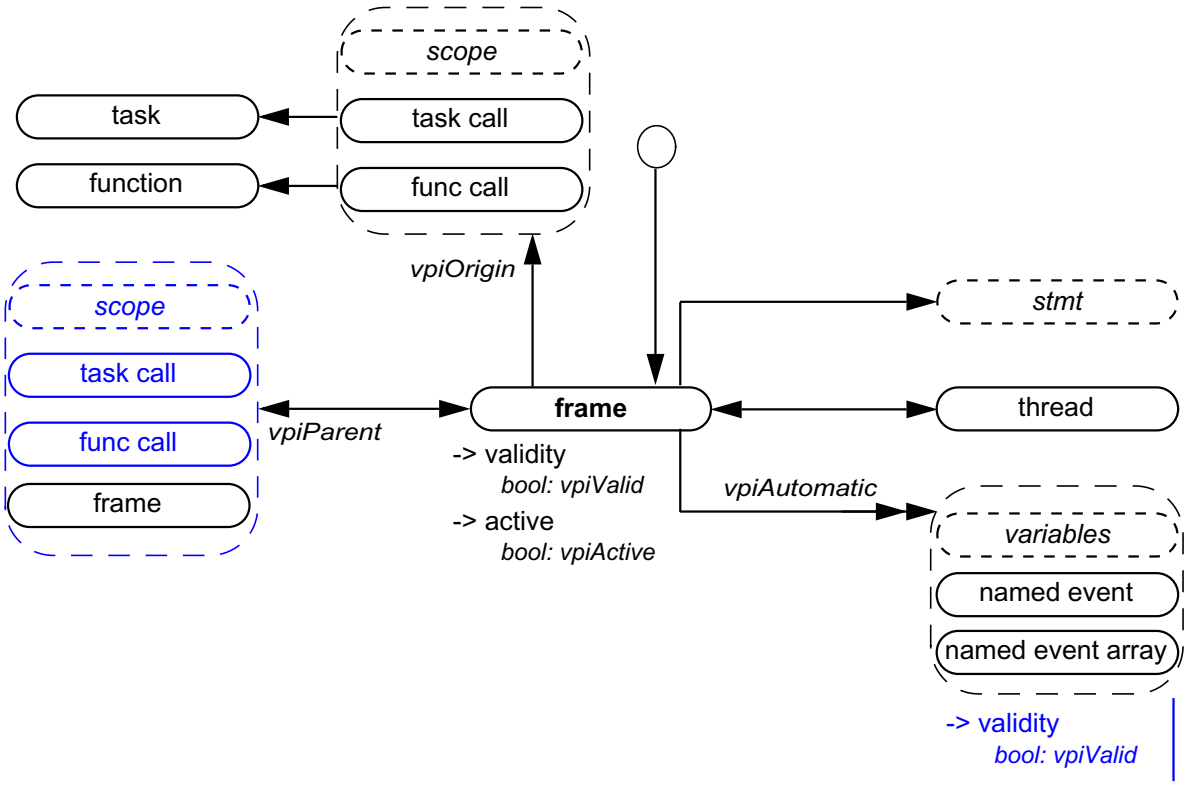
1) To obtain all the dimensions of a multidimensional array, the range iterator must be used. Using the **vpiLeftRange**/**vpiRightRange** properties only returns the last dimension of a multidimensional array.

32.28 Frames (supersedes P1364 26.6.20)

CHANGE



TO



Change the proposal for Item 485 as follows:

~~7) The Frame object model is not backwards compatible with IEEE 1364-2005.~~

24.5, Page 352

Change

Then $isunbounded shall return true.

TO

Then $isunbounded (foo) shall return true. Otherwise, it shall return false. True and false are defined in 24.10.

ERRATA 319: Port connection rules

The two sections 19.11.3 and 11.11.4 are displayed below with the changes proposed for fixing this errata.

19.11.3 Instantiation using implicit .name port connections

SystemVerilog adds the capability to implicitly instantiate ports using a .name syntax if the instance-port name and equivalent type match the connecting ~~variable~~declaration-port name and type. This enhancement eliminates the requirement to list a port name twice when the port name and ~~signal~~ declaration name are the same, while still listing all of the ports of the instantiated module for documentation purposes.

In the following alu_accum3 example, all of the ports of the instantiated alu module match the names of the ~~variables~~ declarations connected to the ports, except for the unconnected zero port, which is listed using a named port connection, showing that the port is unconnected. Implicit .name port connections are made for all name and equivalent type matching connections on the instantiated module.

In the same alu_accum3 example, the accum module has an 8-bit port called dataout that is connected to a 16-bit bus called dataout. Because the internal and external sizes of dataout do not match, the port must be connected by name, showing which bits of the 16-bit bus are connected to the 8-bit port. The datain port on the accum is connected to a bus by a different name (alu_out), so this port is also connected by name. The clk and rst_n ports are connected using implicit .name port connections. Also in the same alu_accum3 example, the xtend module has an 8-bit output port called dout and a 1- bit input port called din. Since neither of these port names match the names (or sizes) of the connecting ~~variables~~declarations, both are connected by name. The clk and rst_n ports are connected using implicit .name port connections.

```
module alu_accum3 (
output [15:0] dataout,
input [7:0] ain, bin,
input [2:0] opcode,
input clk, rst_n);
wire [7:0] alu_out;
alu alu (.alu_out, .zero(), .ain, .bin, .opcode);
accum accum (.dataout(dataout[7:0]), .datain(alu_out), .clk, .rst_n);
xtend xtend (.dout(dataout[15:8]), .din(alu_out[7]), .clk, .rst_n);
endmodule
```

A *.port_identifier* port connection is semantically equivalent to the named port connection *.port_identifier*(~~name~~port_identifier) ~~port connection~~ with the following exceptions:

— The port connection shall not create an implicit wire declaration.

— The declarations on each side of the port connection shall have equivalent data types.

— An implicit *.port_identifier* port connection between nets of two dissimilar net types shall generate an error when it is a warning in an explicit named port connection as required by the Verilog standard.

~~— The identifier referenced by *.port_identifier* shall not create an implicit wire declaration.~~

~~— It shall be illegal for a *.port_identifier* port connection to create an implicit cast. This includes truncation or padding.~~

~~— A conversion between a 2-state and 4-state type of the same bit length is a legitimate cast.~~

~~— It shall be an error if a *.port_identifier* port connection between two dissimilar net types would generate a~~

~~warning message as required by the Verilog standard.~~

19.11.4 Instantiation using implicit .* port connections

SystemVerilog adds the capability to implicitly instantiate ports using a .* syntax for all ports where the instance-port name ~~and type~~ matches the connecting ~~variable~~ port name and their data types are equivalent-~~type~~. This enhancement eliminates the requirement to list any port where the name and type of the connecting ~~variable~~ declaration match the name and equivalent type of the instance port. This implicit port connection style is used to indicate that all port names and types match the connections where emphasis is placed only on the exception ports. The implicit .* port connection syntax can greatly facilitate rapid block-level testbench generation where all of the testbench ~~variables~~ declarations are chosen to match the instantiated module port names and types.

In the following alu_accum4 example, all of the ports of the instantiated alu module match the names of the ~~variables~~ declarations connected to the ports, except for the unconnected zero port, which is listed using a named port connection, showing that the port is unconnected. The implicit .* port connection syntax connects all other ports on the instantiated module.

In the same alu_accum4 example, the accum module has an 8-bit port called dataout that is connected to a 16-bit bus called dataout. Because the internal and external sizes of dataout do not match, the port must be connected by name, showing which bits of the 16-bit bus are connected to the 8-bit port. The datain port on the accum is connected to a bus by a different name (alu_out), so this port is also connected by name. The clk and rst_n ports are connected using implicit .* port connections. Also in the same alu_accum4 example, the xtend module has an 8-bit output port called dout and a 1- bit input port called din. Since neither of these port names match the names (or sizes) of the connecting ~~variables~~ declarations, both are connected by name. The clk and rst_n ports are connected using implicit .* port connections.

```
module alu_accum4 (
output [15:0] dataout,
input [7:0] ain, bin,
input [2:0] opcode,
input clk, rst_n);
wire [7:0] alu_out;
alu alu (.*, .zero());
accum accum (.*, .dataout(dataout[7:0]), .datain(alu_out));
xtend xtend (.*, .dout(dataout[15:8]), .din(alu_out[7]));
endmodule
```

An implicit .* port connection is semantically equivalent to a default .name port connection for every port declared in the instantiated module.

A named port connection can be mixed with a .* connection to override the
port connection to a different expression or to leave the port unconnected.

When the implicit .* port connection is mixed in the same instantiation with named port connections, the implicit .* port connection token can be placed anywhere in the port list. The .* token can only appear at most once in the port list.

Modules can be instantiated into the same parent module using any combination of legal positional, named, implicit .name connected and implicit .* connected instances as shown in alu_accum5 example.

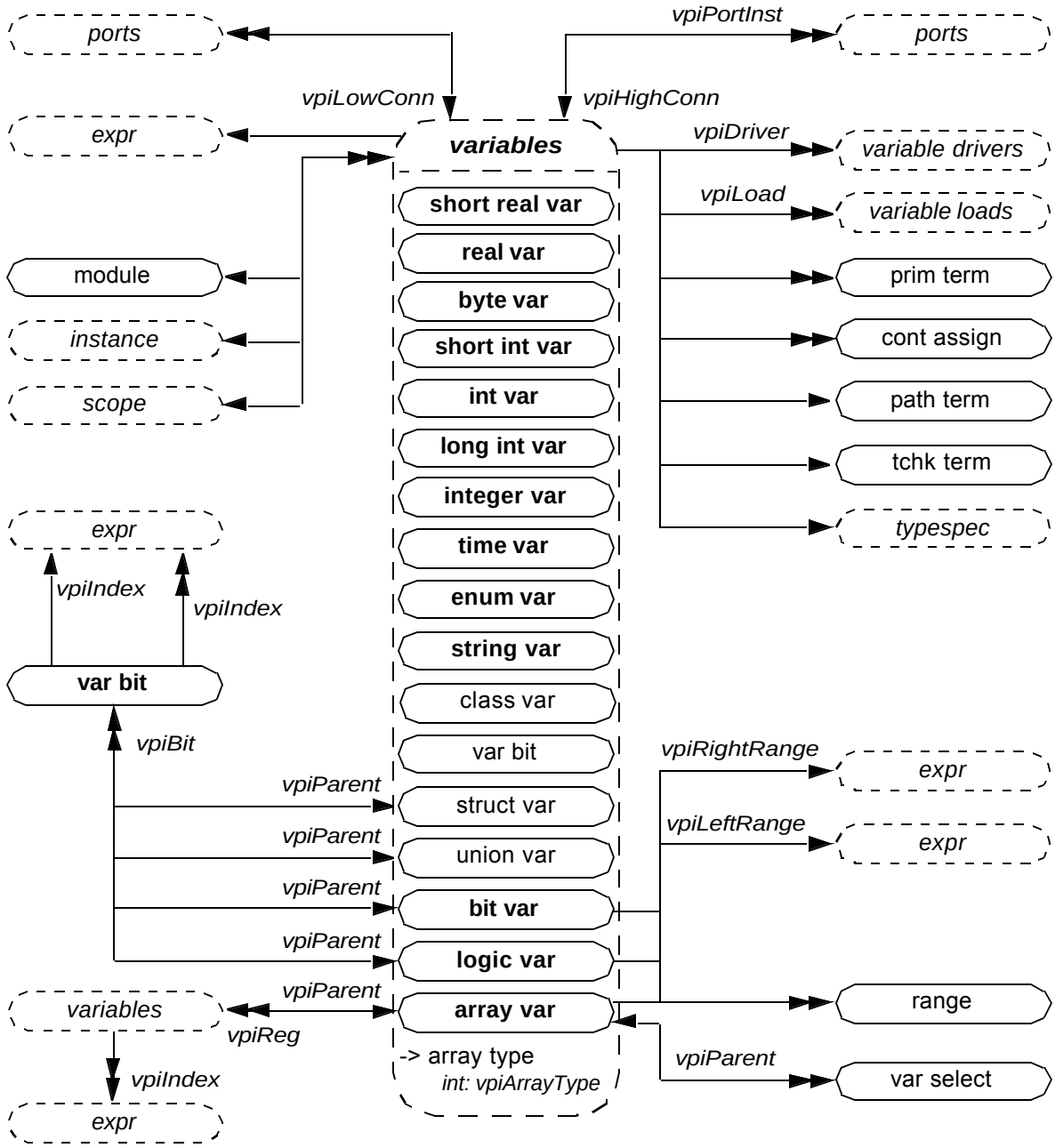
```
module alu_accum5 (
output [15:0] dataout,
```



```
input [7:0] ain, bin,
input [2:0] opcode,
input clk, rst_n);
wire [7:0] alu_out;
// mixture of named port connections and
// implicit .name port connections
alu alu (.ain(ain), .bin(bin), .alu_out, .zero(), .opcode);
// positional port connections
accum accum (dataout[7:0], alu_out, clk, rst_n);
// mixture of named port connections and implicit .* port connections
xtend xtend (.dout(dataout[15:8]), .*, .din(alu_out[7]));

endmodule
```


Section 32.14 Variables (supersedes P1364 26.6.7 and 26.6.8)

REPLACE



-> array member
bool: vpiArray (deprecated)
bool: vpiArrayMember

-> name
str: vpiName
str: vpiFullName

-> sign
bool: vpiSigned

-> size
int: vpiSize

-> array type
int: vpiArrayType

-> lifetime
bool: vpiAutomatic

-> constant variable
bool: vpiConstantVariable

-> determine random availability
bool: vpiIsRandomized

-> randomization type
int: vpiRandType

-> member
bool: vpiMember

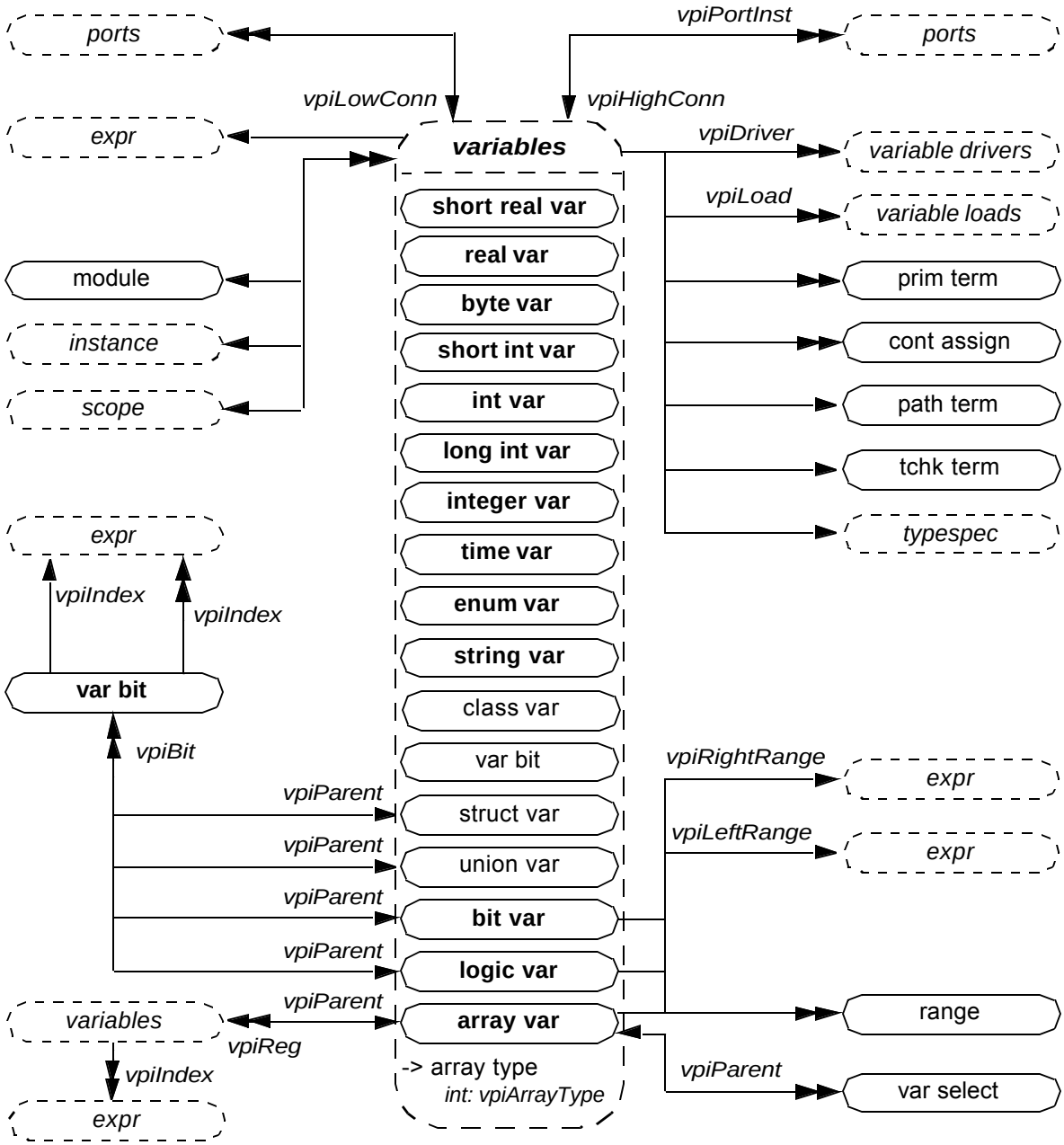
->value
vpi_get_value()
vpi_put_value()

-> scalar
bool: vpiScalar

-> visibility
int: vpiVisibility

-> vector
bool: vpiVector

WITH



-> array member
bool: vpiArray (deprecated)
bool: vpiArrayMember

-> name
str: vpiName
str: vpiFullName

-> sign
bool: vpiSigned

-> size
int: vpiSize

-> array type
int: vpiArrayType

-> lifetime
bool: vpiAutomatic

-> constant variable
bool: vpiConstantVariable

-> determine random availability
bool: vpiIsRandomized

-> randomization type
int: vpiRandType

-> member
bool: vpiMember

->value
vpi_get_value()
vpi_put_value()

-> scalar
bool: vpiScalar

-> visibility
int: vpiVisibility

-> vector
bool: vpiVector

-> valid
int: vpiValid

Note to editor: The new text should precede the original NOTES section as indicated below. (According to proposal 266, the original NOTES section will be renamed DETAILS.) The NOTE in the new text is informative rather than normative, and hence uses the modal auxiliary “will” rather than “shall”. The new NOTE clarifies the implications of the definitions of **vpiValidTrue** and **vpiValidFalse**.

REPLACE

NOTES

1)....

WITH

A value of **vpiValidTrue** for the property **vpiValid** shall indicate that the application may continue to access the properties, relationships, and value of the variable indicated by the reference handle. A value of **vpiValidFalse** shall indicate that the variable indicated by the reference handle can no longer be accessed. A variable may cease to exist, for example, if the scope terminates in which an automatic variable is declared (6.6), or because the object is reclaimed by automatic memory management (12.26).

If an implementation is unable to determine whether or not the associated variable still exists, it may cause **vpiValid** to return a value of **vpiValidUnknown**. In this case, the application may attempt to reacquire a handle to the variable by starting from a static handle or from a handle for which **vpiValid** returns **vpiValidTrue**.

NOTE -- An attempt to reacquire a handle to a variable for which **vpiValid** returns **vpiValidFalse** will always fail. An attempt to reacquire a handle to a variable for which **vpiValid** returns **vpiValidTrue** is unnecessary, but will always succeed.

It shall be an error for an application to attempt to obtain a property, relationship, or value of an invalid variable.

NOTES

1)...

Annex I
(normative)

vpi_user.h

REPLACE

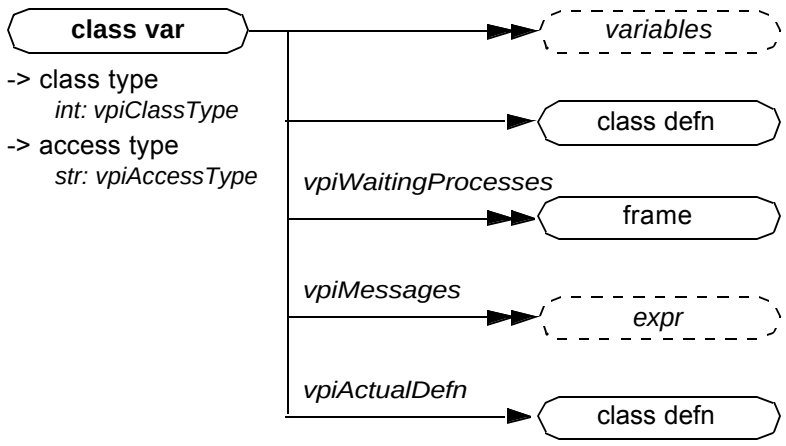
```
/************************ task/function properties ************************/
#define vpiOtherFunc         6 /* returns other types; for property vpiFuncType */
```

WITH

```
/************************ task/function properties ************************/
#define vpiOtherFunc         6 /* returns other types; for property vpiFuncType */

/************************ value for vpiValid ******************************/
#define vpiValidUnknown      2 /* Validity of variable is unknown */
```

REPLACE

32.22 Class variables



NOTES:

1) The **vpiWaitingProcesses** iterator on a mailbox or semaphore shall 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 that 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”.

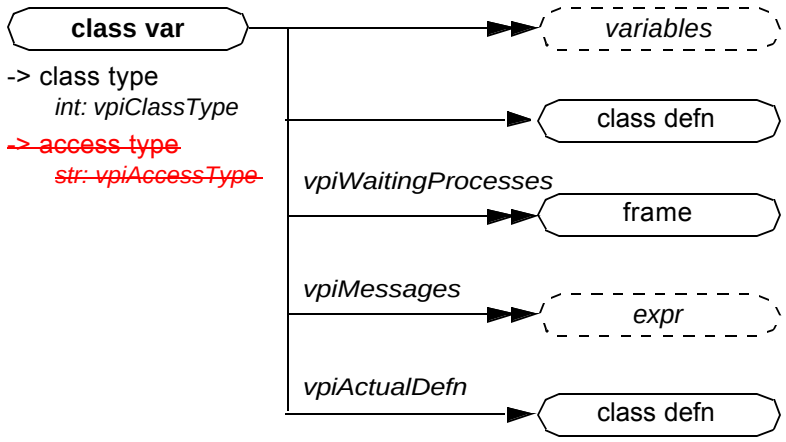
4) **vpiClassDefn/vpiActualDefn** both shall return `NULL` for built-in classes.

5) **vpiClassType** can be one of **vpiUserDefinedClass**, **vpiMailboxClass**, **vpiSemaphoreClass**.

6) **vpiAccessType** can be one of **vpiPublicAcc**, **vpiProtectedAcc**, **vpiLocalAcc**.

WITH

32.22 Class variables



NOTES:

1) The **vpiWaitingProcesses** iterator on a mailbox or semaphore shall 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 that 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.

5) **vpiClassType** can be one of **vpiUserDefinedClass**, **vpiMailboxClass**, **vpiSemaphoreClass**.

~~6) **vpiAccessType** can be one of **vpiPublicAcc, vpiProtectedAcc, vpiLocalAcc.**~~

REPLACE

32.14 Variables (supersedes P1364 26.6.7 and 26.6.8)

NOTES

1)....

2)...

23)....

WITH

1)...

2)..

23) ...

24) **vpiVisibility** denotes the visibility (**local**, **protected**, or default) of a variable that is a class member. **vpiVisibility** shall return **vpiPublicVis** for a class member that is not **local** or **protected**, or for a variable that is not a class member.

Section 32.26 Task and function declaration (supersedes p1364 26.6.18)

REPLACE

4) **vpiReturn** shall always return a var object, even for simple returns.

WITH

4) **vpiReturn** shall always return a var object, even for simple returns.

5) **vpiVisibility** denotes the visibility (**local**, **protected**, or default) of a task or function that is a class member (a method). **vpiVisibility** shall return **vpiPublicVis** for a class member that is not **local** or **protected**, or for a task or function that is not a class member.

Proposed wording from April 4 F2F:

In Section 19.11.4, page 291, CHANGE:

An implicit .* port connection is semantically equivalent to a default .name port connection for every port declared in the instantiated module. A named port connection can be mixed with a .* connection to override the port connection to a different expression or to leave the port unconnected.

TO:

An implicit **.*** port connection is semantically equivalent to a default .name port connection for every port declared in the instantiated module with the exception that **.*** does not create a sufficient reference for a wildcard import of a name from a package. A named port connection can be mixed with a **.*** connection to override a port connection to a different expression, or to leave a port unconnected, A named or implicit .name connection can be mixed with a **.*** connection to create a sufficient reference for a wildcard import of a name from a package.

In 19.8, CHANGE:

For the first port, if neither a type nor a direction is specified, then it shall be assumed to be a member of a port list, and any port direction or type declarations must be declared after the port list. This is compatible with the Verilog-1995 syntax. If the first port type but no direction is specified, then the port direction shall default to **inout**. If the first port direction but no port kind or data type is specified, then the port shall default to a net of net type **wire**. This default net type can be changed using the `‘default_nettype` compiler directive, as in Verilog.

TO:

For the first port, if neither a type nor a direction is specified, then it shall be assumed to be a member of a port list, and any port direction or type declarations must be declared after the port list. This is compatible with the Verilog-1995 syntax. If the first ~~port type~~ port kind or data type is specified but no direction is specified, then the port direction shall default to **inout**. If the first port direction is specified but no port kind or data type is specified, then the port shall default to a net of net type **wire**. This default net type can be changed using the `‘default_nettype` compiler directive, as in Verilog.

In 19.8, CHANGE:

For an **inout** port, if the port kind is omitted, then the port shall default to a net of net type **wire**. This default net type can be changed using the `‘default_nettype` compiler directive, as in Verilog.

```
// the inout port defaults to a net of net type wire
module mh2 (inout integer a);
...
endmodule
```

TO:

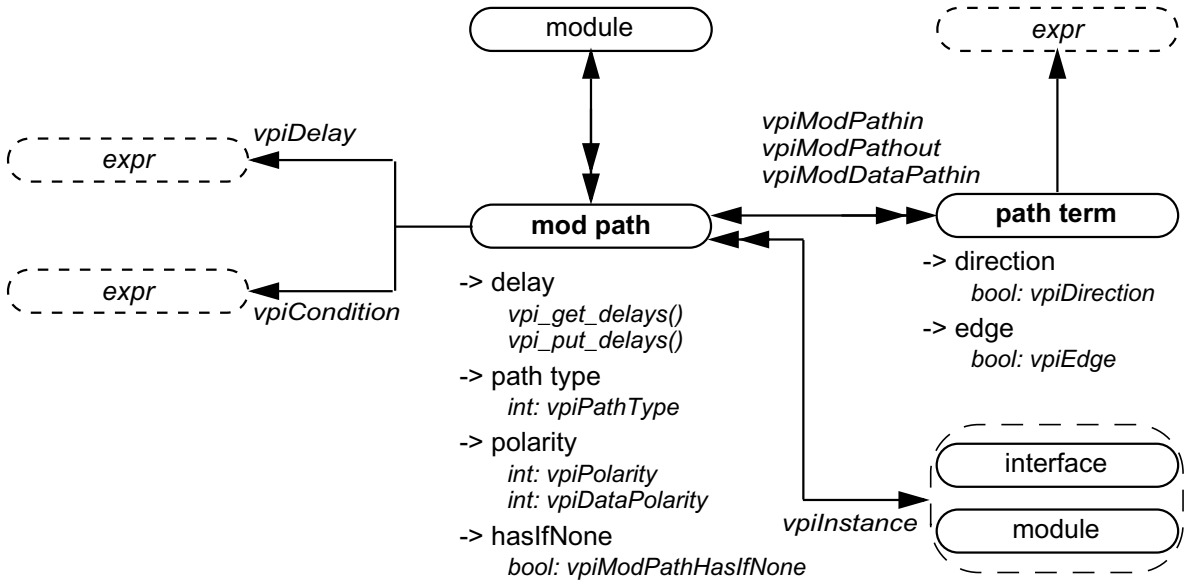
For ~~an **inout** port~~ **input** and **inout** ports, if the port kind is omitted, then the port shall default to a net of net type **wire**. This default net type can be changed using the `‘default_nettype` compiler directive, as in Verilog.

```
// the inout port defaults to a net of net type wire
module mh2 (inout integer a);
...
endmodule
```

For output ports, if the port kind is omitted, the default port kind depends on how the data type is specified. If the port is declared without the data_type syntax, then the port kind defaults to a net of the default net type. This provides backward compatibility with Verilog. If the data type is declared with the data_type syntax, the port kind defaults to variable.

REPLACE

32.25 Module path, path term (supersedes P1364 26.6.15)

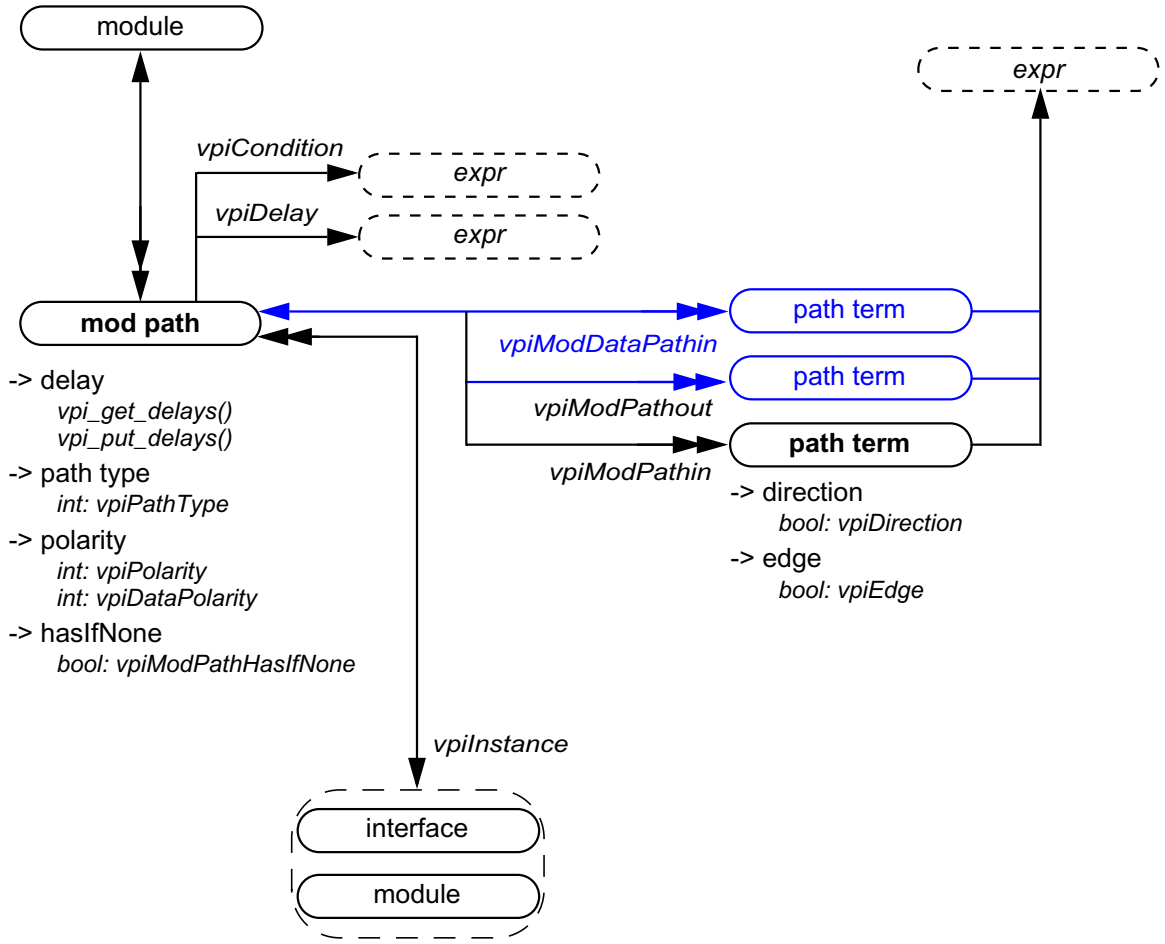


NOTE:

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

WITH

32.25 Module path, path term (supersedes P1364 26.6.15)



NOTE:

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

Section **29.3.2.2**

REMOVE section 29.3.2.2 completely

~~29.3.2.2 Extending vpi_get() and vpi_get_str()~~
~~In addition to vpi_get_assertion_info, the following existing VPI functions are also extended:~~
~~vpi_get(), vpi_get_str()~~
~~vpi_get() can be used to query the following VPI property from a handle to an assertion:~~

~~vpiType~~
~~returns one of vpiSequenceInst, vpiAssert, vpiCover, vpiAssume,~~
~~vpiPropertyInst, vpiImmediateAssert.~~

~~vpiLineNo~~
~~returns the line number where the assertion is declared.~~

~~vpi_get_str() can be used to obtain the following VPI properties from an assertion handle:~~

~~vpiFileName~~
~~returns the filename of the source file where the assertion was declared.~~

~~vpiName~~
~~returns the name of the assertion.~~

~~vpiFullName~~
~~returns the fully qualified name of the assertion.~~

Section

Clause 30

MODIFY the title of Clause 30 as follows:

~~**SystemVerilog Coverage API**~~

SystemVerilog Code Coverage Control and API

Section 29.5.2

Change:

NOTE—In this release, the only step control constant available is `vpiAssertionClockSteps`, indicating callbacks on a per assertion/clock-tick basis. The assertion clock is the event expression supplied as the clocking expression to the assertion declaration. The assertion shall “advance” whenever this event occurs and, when stepping is enabled, such events shall also cause step callbacks to occur.

To: (part of body, no longer a note)

~~NOTE—In this release, t~~The ~~only~~ fine grained step control constant ~~available is~~ `vpiAssertionClockSteps`~~,~~ indicat~~ing~~es callbacks on a per assertion/clock-tick basis. The assertion clock is the event expression supplied as the clocking expression to the assertion declaration. ~~The assertion shall “advance” whenever this event occurs and, when stepping is enabled, such events shall also cause step callbacks to occur.~~ This step callback shall occur at every clocking event, when stepping is enabled, as the assertion “advances” in evaluation.

REPLACE

30.4.4 Controlling coverage

```
vpi_control(<coverageControl>, <coverageType>, instance_handle)
vpi_control(<coverageControl>, <coverageType>, assertion_handle)
```

Controls the collection of coverage on the given instance or assertion. Note that statement, toggle and FSM coverage are not individually controllable (i.e., they are controllable only at the instance level and not on a per statement/signal/FSM basis). The semantics and behavior are as per the `$coverage_control` system function (see 30.2.2.1). *coverageControl* is one `vpiCoverageStart`, `vpiCoverageStop`, `vpiCoverageReset` or `vpiCoverageCheck`, as defined in 30.4.2. *coverageType* is any one of the VPI coverage type properties (see 30.4.2)

```
vpi_control(<coverageControl>, <coverageType>, name)
```

This saves or merges coverage into the current simulation. The semantics and behavior are specified as per the equivalent system functions `$coverage_merge` (see 30.2.2.4) and `$coverage_save` (see 30.2.2.5). *coverageControl* is one of `vpiCoverageMerge` or `vpiCoverageSave`, defined in 30.4.2.

WITH

30.4.4 Controlling coverage

Control of the collection of coverage shall be through the `vpi_control()` routine:

```
vpi_control(<coverageControl>, <coverageType>, instance_handle)
vpi_control(<coverageControl>, <coverageType>, assertion_handle)
```

~~Controls the collection of coverage on the given instance or assertion. Note that s~~Statement, toggle and FSM coverage are not individually controllable (i.e., they are controllable only at the instance level and not on a per ~~statement/signal/FSM~~statement, signal, or FSM basis). The semantics and behavior are as per the `$coverage_control` system function (see 30.2.2.1). *coverageControl* ~~is one~~ shall be `vpiCoverageStart`, `vpiCoverageStop`, `vpiCoverageReset` or `vpiCoverageCheck`, as defined in 30.4.2. *coverageType* is any one of the VPI coverage type properties (see 30.4.2)

To save coverage for the current simulation use:

```
vpi_control(<coverageControl>, <coverageType>, name)
```

```
vpi_control(vpiCoverageSave, <coverageType>, name)
```

as defined in 30.4.2. ~~This saves or merges coverage into the current simulation.~~ The semantics and behavior are specified as per the equivalent system functions~~ `$coverage_merge` (see 30.2.2.4) and~~ `$coverage_save` (see 30.2.2.5). ~~*coverageControl* is one of `vpiCoverageMerge` or `vpiCoverageSave`, defined in 30.4.2.~~

To merge coverage for the current simulation use:

```
vpi_control(vpiCoverageMerge, <coverageType>, name)
```

as defined in 30.4.2. The semantics and behavior are specified as per the equivalent system function `$coverage_merge` (see 30.2.2.4).

Section: 29.2.3 Callbacks

Change:

2) “Assertion system”

```
cbAssertionSysInitialized
cbAssertionSysStart
cbAssertionSysStop
cbAssertionSysEnd
cbAssertionSysReset
```

To:

2) “Assertion system”

```
cbAssertionSysInitialized
cbAssertionSysStartOn
cbAssertionSysStopOff
cbAssertionSysKill
cbAssertionSysEnd
cbAssertionSysReset
```

Section: 29.2.4 Control constants

Change:

This subclause lists the system control constant callbacks. Refer to 29.5 for details on how to use `vpi_control()` with these constants to obtain assertion control information, and control the assertion system.

1) Assertion

```
vpiAssertionDisable
vpiAssertionEnable
vpiAssertionReset
vpiAssertionKill
vpiAssertionEnableStep
vpiAssertionDisableStep
```

2) Assertion stepping

```
vpiAssertionClockSteps
```

3) “Assertion system”

```
vpiAssertionSysStart
vpiAssertionSysStop
vpiAssertionSysEnd
vpiAssertionSysReset
```

To:

This subclause lists the system control constants ~~callbacks~~. Refer to 29.5 for details on how to use `vpi_control()` with these constants to obtain assertion control information, and control the assertion system.

1) Assertion

```
vpiAssertionDisable
vpiAssertionEnable
vpiAssertionReset
vpiAssertionKill
vpiAssertionEnableStep
vpiAssertionDisableStep
```

2) Assertion stepping

```
vpiAssertionClockSteps
```

3) “Assertion system”

```
vpiAssertionSysStartOn
```

vpiAssertionSys~~Stop~~Off
vpiAssertionSysKill
vpiAssertionSysEnd
vpiAssertionSysReset

Section: 29.4.1 Placing assertion system callbacks

Change:

Use vpi_register_cb(), setting the cb_rtn element to the function to be invoked and the reason element of the s_cb_data structure to one of the following values, to place an assertion system callback.

cbAssertionSysInitialized
occurs after the system has initialized. No assertion-specific actions can be performed until this callback completes. The assertion system can initialize before cbStartOfSimulation does or afterwards.

cbAssertionSysStart
the assertion system has become active and starts processing assertion attempts. This always occur after cbAssertionSysInitialized. By default, the assertion system is "started" on simulation startup, but the user can delay this by using assertion system control actions.

cbAssertionSysStop
the assertion system has been temporarily suspended. While stopped no assertion attempts are processed and no assertion-related callbacks occur. The assertion system can be stopped and resumed an arbitrary number of times during a single simulation run.

cbAssertionSysEnd
occurs when all assertions have completed and no new attempts shall start. Once this callback occurs no more assertion-related callbacks shall occur and assertion-related actions shall have no further effect. This typically occurs after the end of simulation.

cbAssertionSysReset
occurs when the assertion system is reset, e.g., due to a system control action.

To:

Use vpi_register_cb(), setting the cb_rtn element to the function to be invoked and the reason element of the s_cb_data structure to one of the following values, to place an assertion system callback.

cbAssertionSysInitialized
occurs after the system has initialized. No assertion-specific actions can be performed until this callback completes. The assertion system can initialize before cbStartOfSimulation does or afterwards.

cbAssertionSys~~Start~~On
the assertion system has become active and starts processing assertion attempts. This always occur after cbAssertionSysInitialized. By default, the assertion system is "started" on simulation startup, but the user can delay this by using assertion system control actions.

cbAssertionSys~~Stop~~Off
the assertion system has been temporarily stopped. While stopped no new assertion attempts are processed and no new assertion-related callbacks occur. Assertions already executing are not affected. The assertion system can be stopped and resumed an arbitrary number of times during a single simulation run.

cbAssertionSysKill
the assertion system has been temporarily suspended. While suspended no assertion attempts are processed and no assertion-related callbacks occur. The assertion system can be suspended and resumed an arbitrary number of times during a single simulation run.

cbAssertionSysEnd
occurs when all assertions have completed and no new attempts shall start. Once this callback occurs no more assertion-related callbacks shall occur and assertion-related actions shall have no further effect. This typically occurs after the end of simulation.

cbAssertionSysReset
occurs when the assertion system is reset, e.g., due to a system control action.

Section: 29.5.1 Assertion system control

Change:

Use `vpi_control()`, with one of the following operators and no other arguments, to obtain assertion system control information.

Usage example: `vpi_control(vpiAssertionSysReset)`

`vpiAssertionSysReset`

discards all attempts in progress for all assertions and restore the entire assertion system to its initial state.Any pre-existing `vpiAssertionStepSuccess` and `vpiAssertionStepFailure` callbacks shall be removed; all other assertion callbacks shall remain.

Usage example: `vpi_control(vpiAssertionSysStop)`

`vpiAssertionSysStop`

considers all attempts in progress as unterminated and disable any further assertions from being started. This control has no effect on pre-existing assertion callbacks.

Usage example: `vpi_control(vpiAssertionSysStart)`

`vpiAssertionSysStart`

restarts the assertion system after it was stopped (e.g., due to `vpiAssertionSysStop`). Once started, attempts shall resume on all assertions. This control has no effect on prior assertion callbacks.

Usage example: `vpi_control(vpiAssertionSysEnd)`

`vpiAssertionSysEnd`

discard all attempts in progress and disables any further assertions from starting. All assertion callbacks currently installed shall be removed. Note that once this control is issued, no further assertion related actions shall be permitted.

To:

Use `vpi_control()`, with one of the following ~~operator~~constants and ~~no other arguments~~a second handle argument which is either a `vpiHandle` for a scope or a `vpiCollection` of handles for a list of scopes, to ~~obtain~~ control the assertion system ~~control information~~. A `NULL` handle signifies that the control applies to all assertions regardless of scope.

Usage example: `vpi_control(vpiAssertionSysReset, handle)`

`vpiAssertionSysReset`

discards all attempts in progress for all assertions and restore the entire assertion system to its initial state.Any pre-existing `vpiAssertionStepSuccess` and `vpiAssertionStepFailure` callbacks shall be removed; all other assertion callbacks shall remain.

Usage example: `vpi_control(vpiAssertionSys~~Stop~~Off, handle)`

`vpiAssertionSys~~Stop~~Off`

~~considers all attempts in progress as unterminated and~~ disables any further assertions from being started. Assertions already executing are not affected. This control has no effect on pre-existing assertion callbacks.

`vpiAssertionSysKill`

considers all attempts in progress as unterminated and disables any further assertions from being started. This control has no effect on pre-existing assertion callbacks.

Usage example: `vpi_control(vpiAssertionSys~~Start~~On, handle)`

`vpiAssertionSys~~Start~~On`

restarts the assertion system after it was stopped or suspended (e.g., due to `vpiAssertionSys~~Stop~~Off` or `vpiAssertionSysKill`). Once started, attempts shall resume on all assertions. This control has no effect on prior assertion callbacks.

Usage example: `vpi_control(vpiAssertionSysEnd, handle)`

`vpiAssertionSysEnd`

discards all attempts in progress and disables any further assertions from starting. All assertion callbacks currently installed shall be removed. Note that once this control is issued, no further assertion related actions shall be permitted.

Section: Annex I

Change:

```
#define cbAssertionStart 606
#define cbAssertionSuccess 607
#define cbAssertionFailure 608
#define cbAssertionStepSuccess 609
#define cbAssertionStepFailure 610
#define cbAssertionDisable 611
#define cbAssertionEnable 612
#define cbAssertionReset 613
#define cbAssertionKill 614
/* assertion "system" callback types */
#define cbAssertionSysInitialized 615
#define cbAssertionSysStart 616
#define cbAssertionSysStop 617
#define cbAssertionSysEnd 618
#define cbAssertionSysReset 619
/* assertion control constants */
#define vpiAssertionDisable 620
#define vpiAssertionEnable 621
#define vpiAssertionReset 622
#define vpiAssertionKill 623
#define vpiAssertionEnableStep 624
#define vpiAssertionDisableStep 625
#define vpiAssertionClockSteps 626
#define vpiAssertionSysStart 627
#define vpiAssertionSysStop 628
#define vpiAssertionSysEnd 629
#define vpiAssertionSysReset 630
```

To:

```
#define cbAssertionStart 606
#define cbAssertionSuccess 607
#define cbAssertionFailure 608
#define cbAssertionStepSuccess 609
#define cbAssertionStepFailure 610
#define cbAssertionDisable 611
#define cbAssertionEnable 612
#define cbAssertionReset 613
#define cbAssertionKill 614
/* assertion "system" callback types */
#define cbAssertionSysInitialized 615
#define cbAssertionSysStartOn 616
#define cbAssertionSysStopOff 617
#define cbAssertionSysKill       631
#define cbAssertionSysEnd 618
#define cbAssertionSysReset 619
/* assertion control constants */
#define vpiAssertionDisable 620
#define vpiAssertionEnable 621
#define vpiAssertionReset 622
```

```
#define vpiAssertionKill 623
#define vpiAssertionEnableStep 624
#define vpiAssertionDisableStep 625
#define vpiAssertionClockSteps 626
#define vpiAssertionSysStartOn 627
#define vpiAssertionSysStopOff 628
#define vpiAssertionSysKill      632
#define vpiAssertionSysEnd 629
#define vpiAssertionSysReset 630
```

26.6.1 Module

REPLACE

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

WITH

2) Passing a `NULL` handle to **vpi_get()** with ~~types~~ properties **vpiTimePrecision** or **vpiTimeUnit** shall return the smallest time precision of all modules in the instantiated design.

26.6.2 Instance arrays

REPLACE

-> access by index
vpi_handle_by_index()
vpi_handle_by_multi_index()
-> name
str: vpiName
str: vpiFullName
-> size
bool: vpiSize

WITH

-> access by index
vpi_handle_by_index()
vpi_handle_by_multi_index()
-> name
str: vpiName
str: vpiFullName
-> size
~~*bool*~~*int: vpiSize*

26.6.5 Ports

REPLACE

3) 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.

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

WITH

3) Properties ~~*scalar* and *vector*~~ **vpiScalar** and **vpiVector** 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.

4) Properties ~~*index* and *name*~~ **vpiIndex** and **vpiName** shall not apply for port bits.

26.6.19 Task and Function call

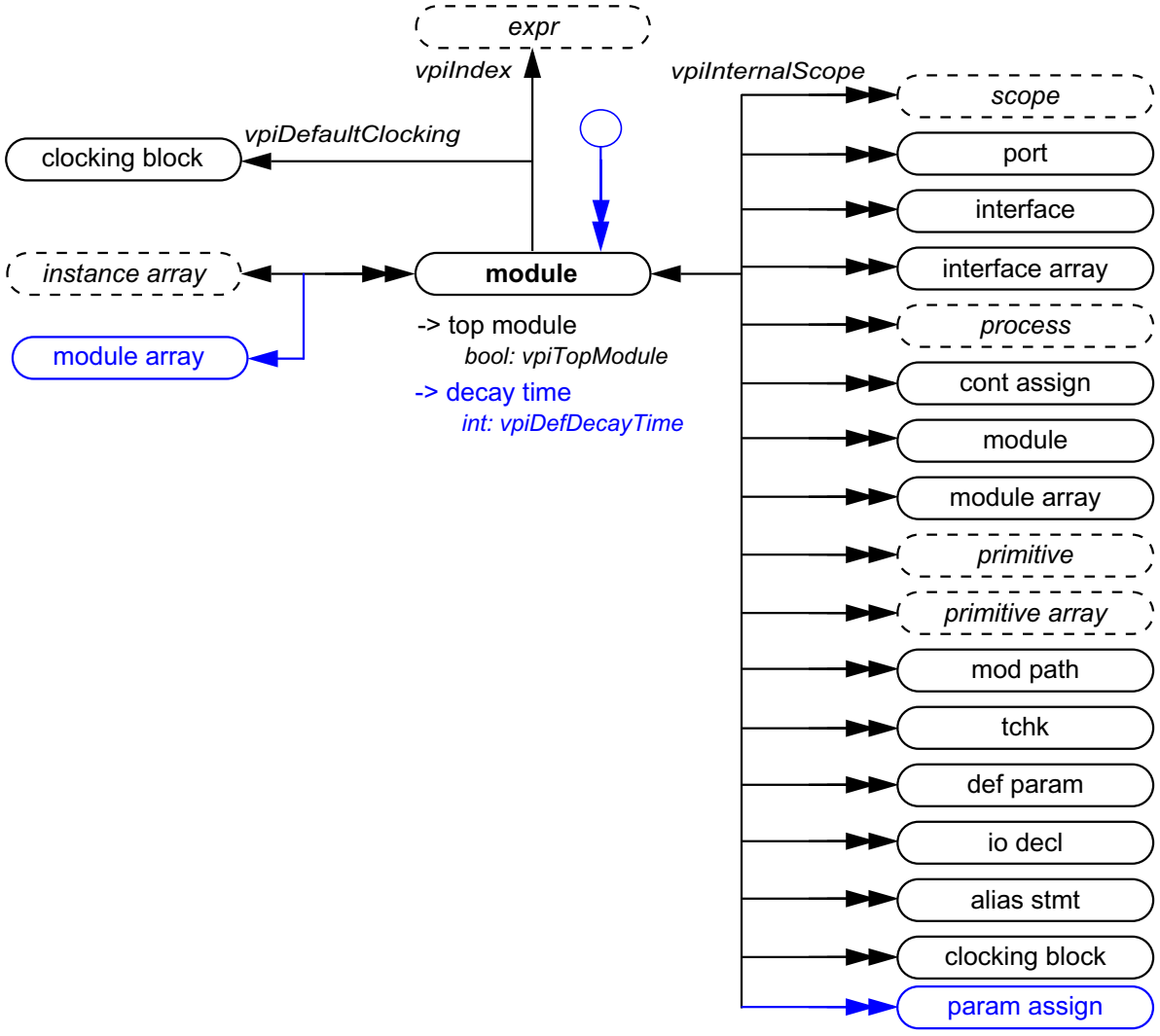
REPLACE

1) The property **vpiDecompile** will return a string with a functionally equivalent system task or function call to what was in the original HDL. The arguments will be decompiled using the same manner as any expression is decompiled. See 32.39 for a description of expression decompilation.

WITH

1) The property **vpiDecompile** ~~will~~ shall return a string with a functionally equivalent system task or function call to what was in the original HDL. The arguments ~~will~~ shall be decompiled using the same manner as any expression is decompiled. See 32.39 for a description of expression decompilation.

32.2 Module (supersedes P1364 26.6.1)

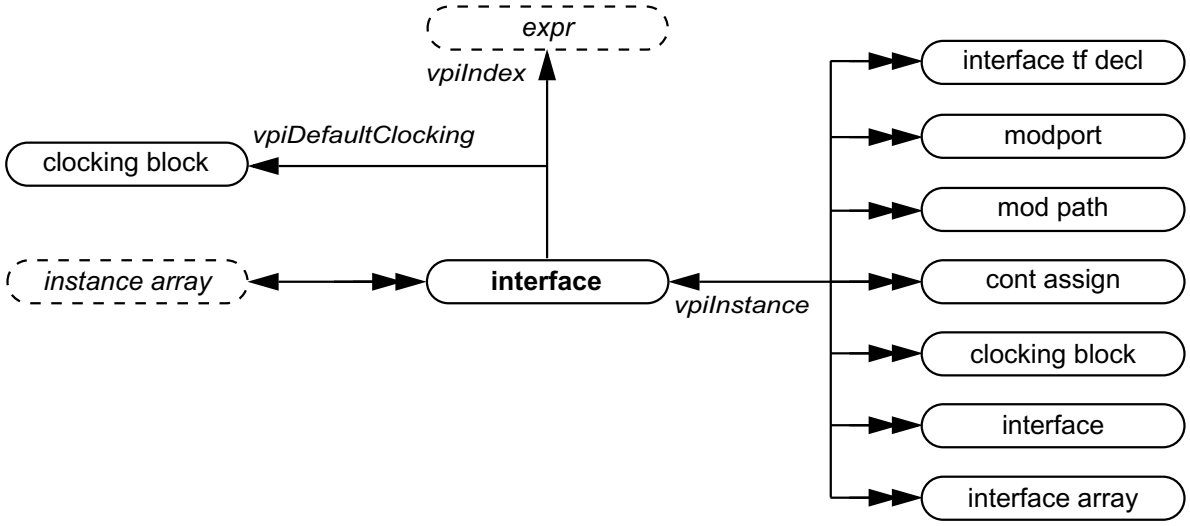


NOTES:

1) ~~**vpiMemory** shall return array variable objects rather than **vpiMemory** objects. The IEEE P1364 standard has made a similar update to the Verilog VPI (refer to note 1 in P1364, 26.6.9)~~Top-level modules shall be accessed using **vpi_iterate()** with a NULL reference object.

2) If a module is an element within a module array, the **vpiIndex** transition is used to access the index within the array. If a module is not part of a module array, this transition shall return NULL.

32.3 Interface

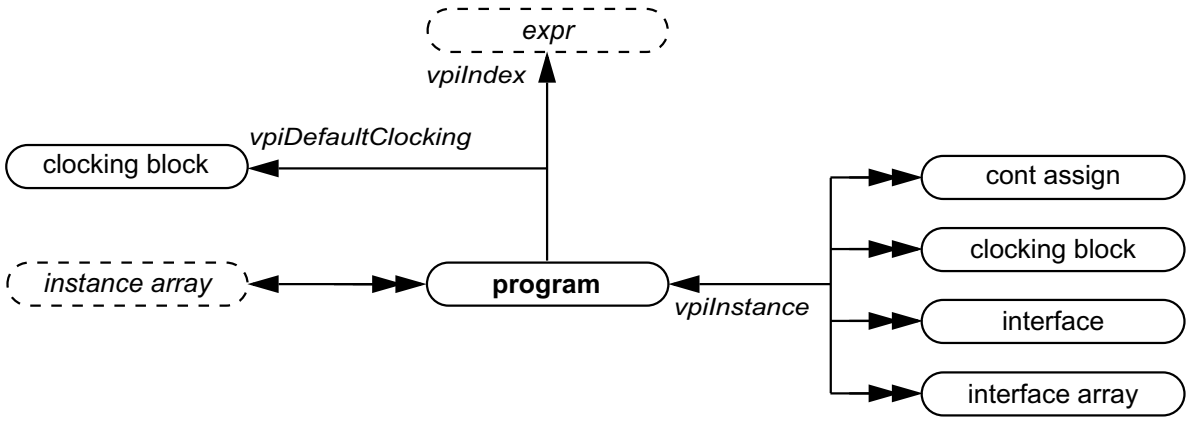


NOTE:

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

2) If an interface is an element within an instance array, the **vpiIndex** transition is used to access the index within the array. If an interface is not part of an instance array, this transition shall return NULL.

32.6 Program

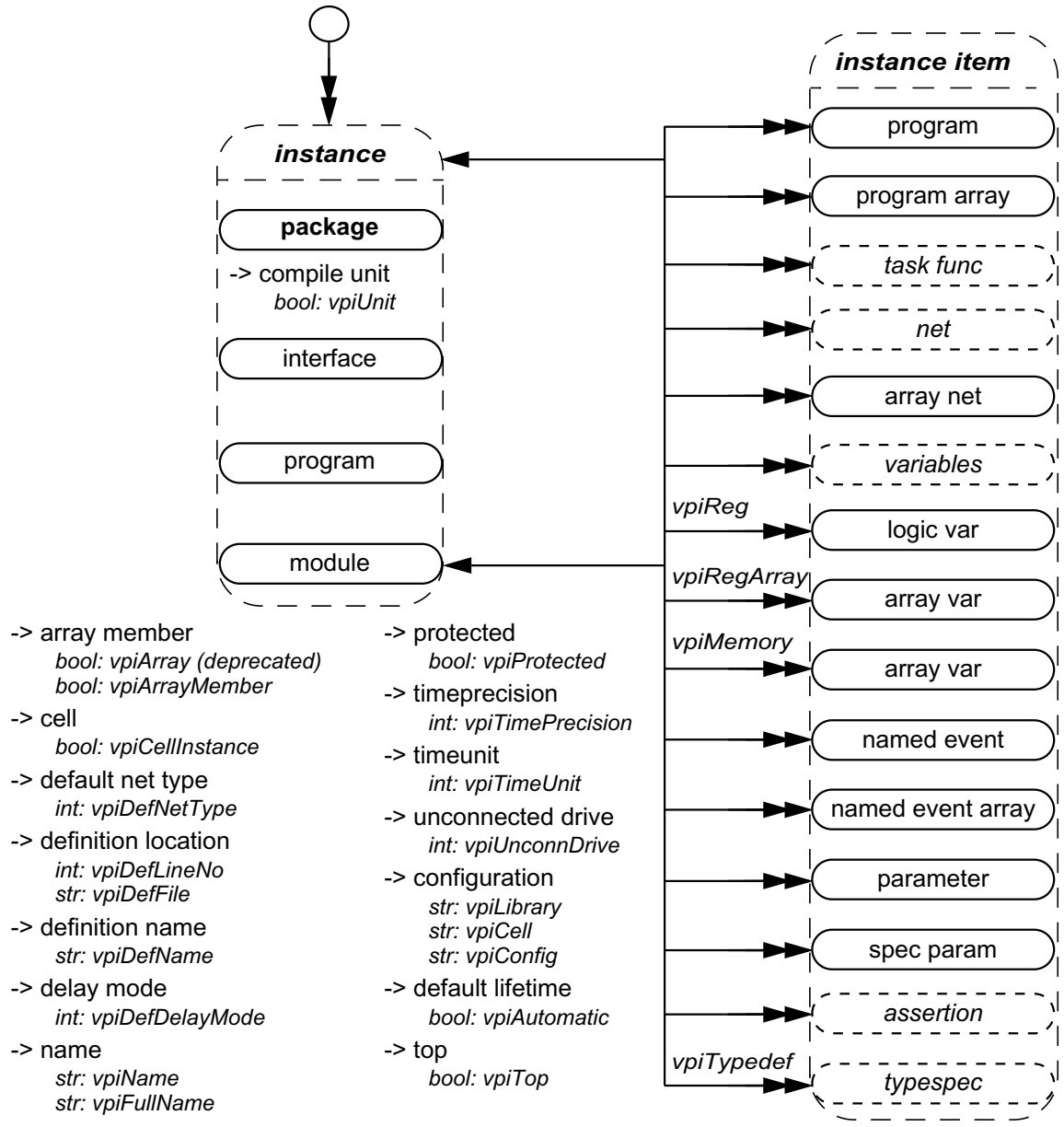


NOTE:

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

2) If a program is an element within an instance array, the **vpiIndex** transition is used to access the index within the array. If a program is not part of an instance array, this transition shall return NULL.

32.7 Instance



NOTES:

1) The **vpiTypedef** iteration shall return the user-defined typespecs that have typedefs explicitly declared in the instance.

2) **vpiModule** shall return a module if the object is inside a module instance, otherwise `NULL`.

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

4) **vpiFullName** for objects that exist within a compilation unit shall begin with '`$unit::`' and therefore may be ambiguous. **vpiFullName** for a package shall be the name of the package and should end with "`::`"; this syntax disambiguates between a module and a package of the same name. **vpiFullName** for objects that exist in a package shall begin with the name of the package followed by "`::`". The separator `::` shall only appear

between the package name and the immediately following the name component. The `.` separator shall be used in all other cases.

5) The following items shall not be accessible via `vpi_handle_by_name()`:

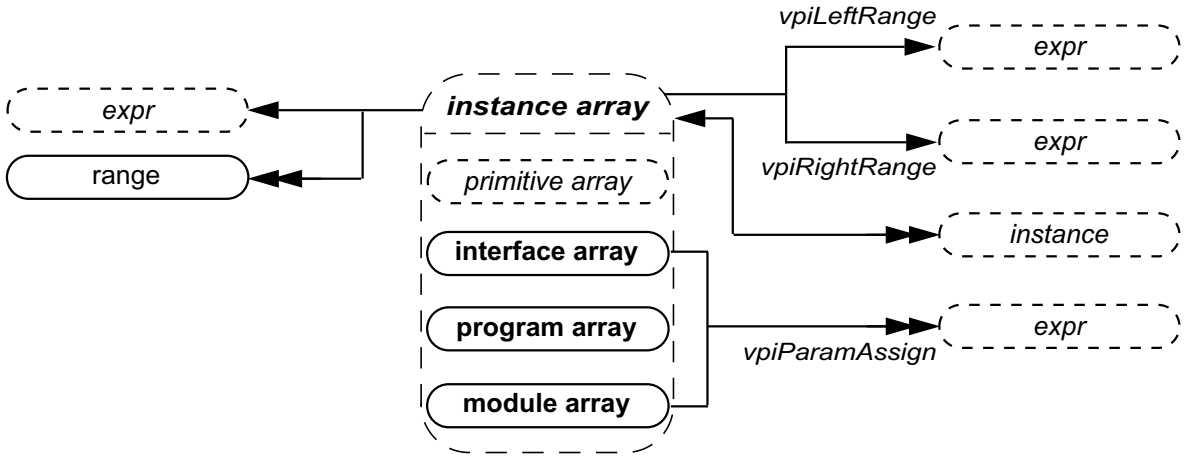
   — imported items

   — objects that exist within a compilation unit

6) Passing a `NULL` handle to **vpi_get()** with ~~types~~ properties (should be fixed in 26.6.1 of 1364 too) **vpiTimePrecision** or **vpiTimeUnit** shall return the smallest time precision of all modules in the instantiated design.

7) The properties **vpiDefLineNo** and **vpiDefFile** can be affected by the **'line** compiler directive. See **P1364 19.7** for more details on the **'line** directive.

32.8 Instance arrays (supersedes P1364 26.6.2)



-> access by index
vpi_handle_by_index()
vpi_handle_by_multi_index()

-> name
str: vpiName
str: vpiFullName

-> size
~~*bool*~~*int: vpiSize (needs to be fixed in 1364 too)*

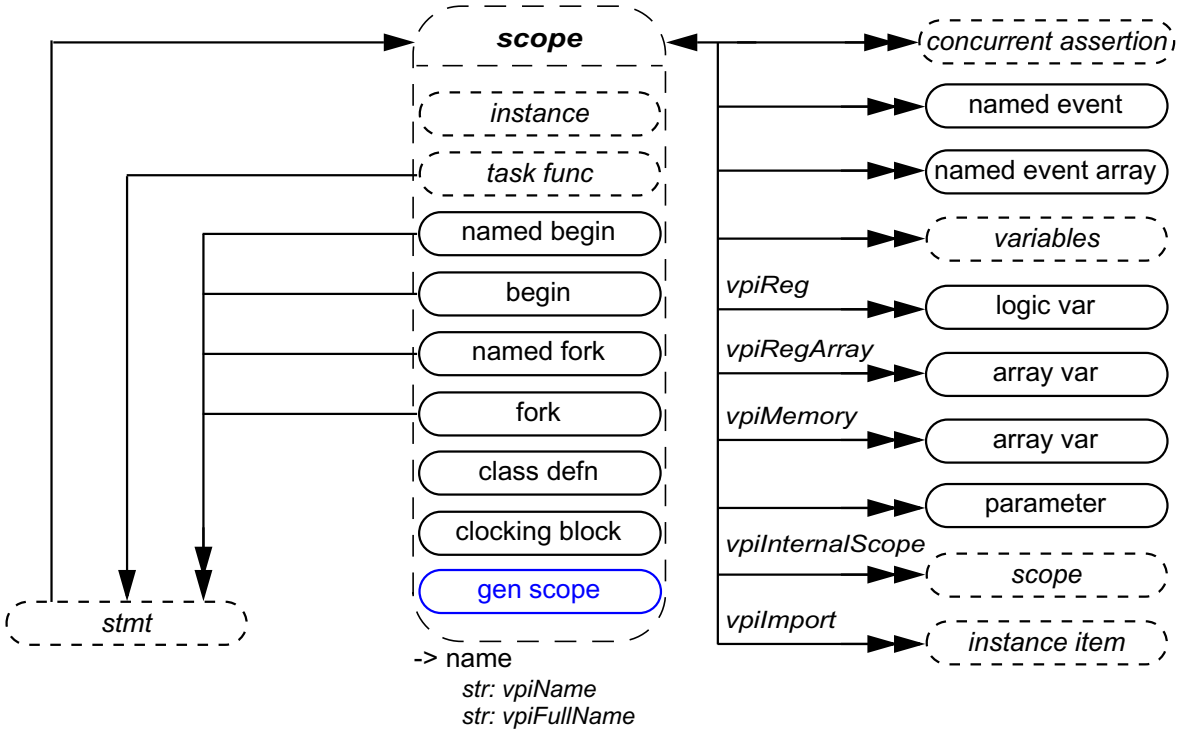
NOTES:

1) Traversing from the instance array to **expr** shall return a simple expression object of type **vpiOperation** with a **vpiOpType** of **vpiListOp**. This expression can be used to access the actual list of connections to the ~~module or primitive~~ instance array in the Verilog source code.

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

3) To obtain all the dimensions of a multidimensional array, the range iterator must be used. Using the **vpiLeftRange**/**vpiRightRange** properties only returns the last dimension of a multidimensional array.

32.9 Scope (supersedes P1364 26.6.3)



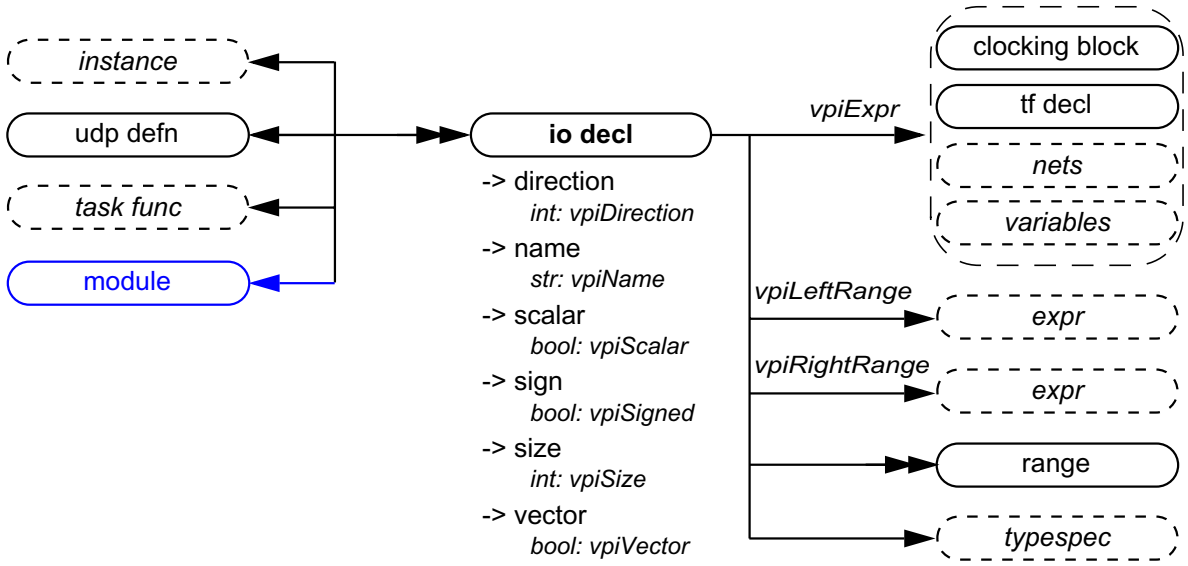
NOTES:

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 19.2.2 for more details.

3) A task func can have zero or more statements (see 11.2 and 11.3). If the number of statements is greater than 1, the **vpiStmt** relation shall return an unnamed **`begin`** that contains the statements of the task or function. If the number of statements is zero, the **vpiStmt** relation shall return `NULL`.

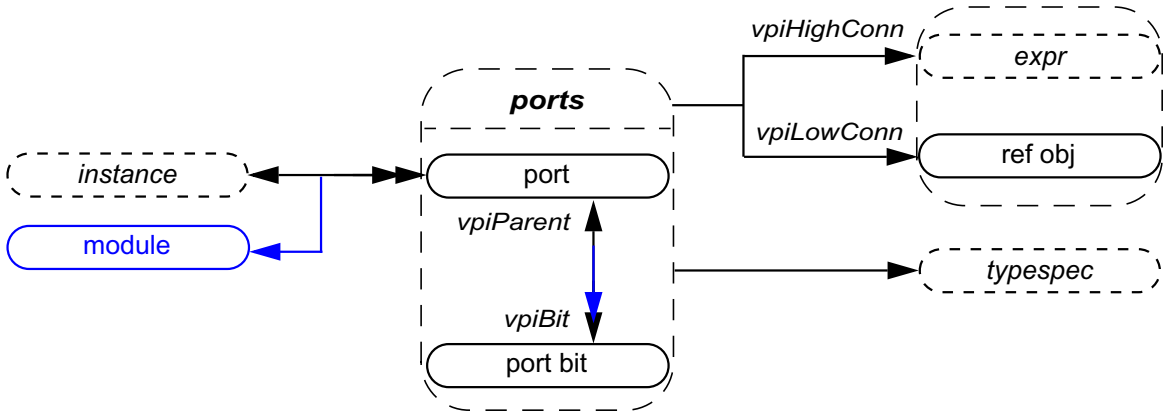
32.10 IO declaration (supersedes P1364 26.6.4)



NOTE:

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

32.11 Ports (supersedes P1364 26.6.5)



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~~ **vpiScalar** and **vpiVector** 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~~ **vpiIndex** and **vpiName** shall not apply for port bits.

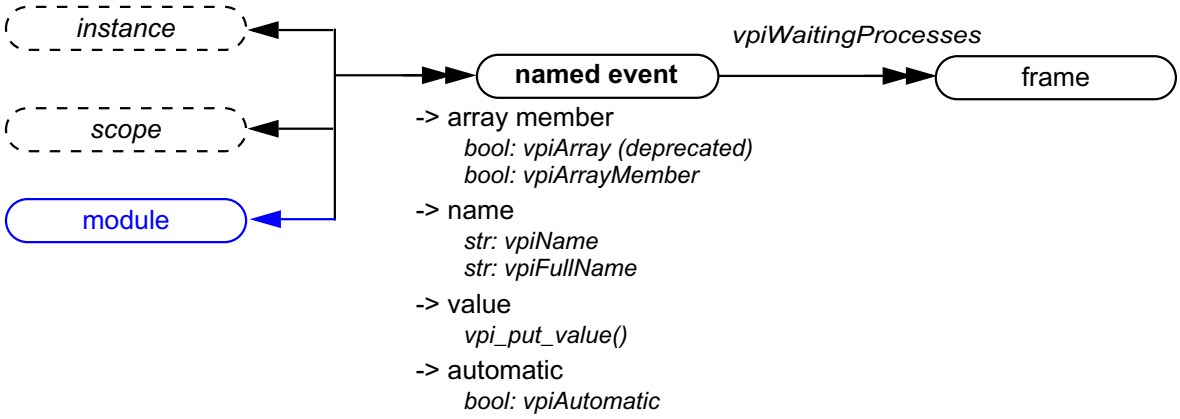
8) If a port is explicitly named, then the explicit name shall be returned. If not, and a name exists, then that name shall be returned. Otherwise, `NULL` shall be returned.

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~~. **vpiLowConn** shall return `NULL` if the module or interface or program port is a null port (e.g. "**module** M();"). **vpiHighConn** shall return `NULL` if the instance of the module or interface or program does not have a connection to the port.

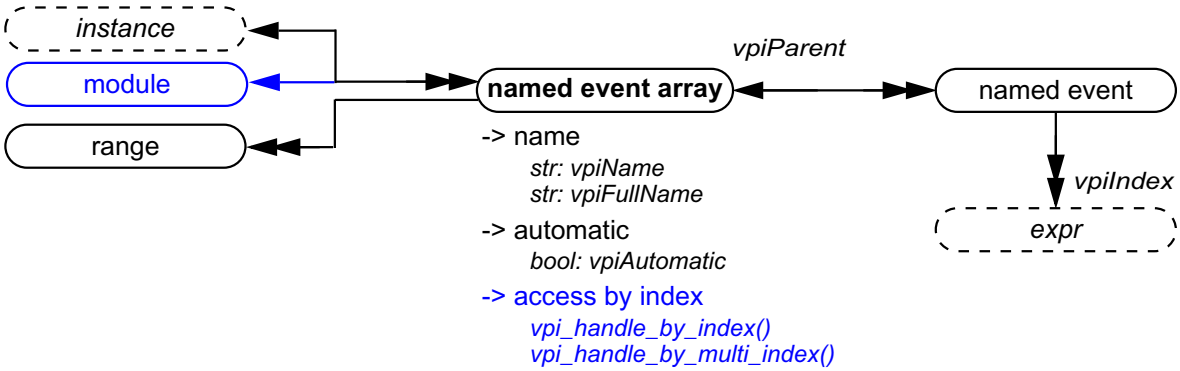
11) **vpiSize** for a null port shall return 0.

32.19 Named events (supersedes P1364 26.6.11)



NOTE:

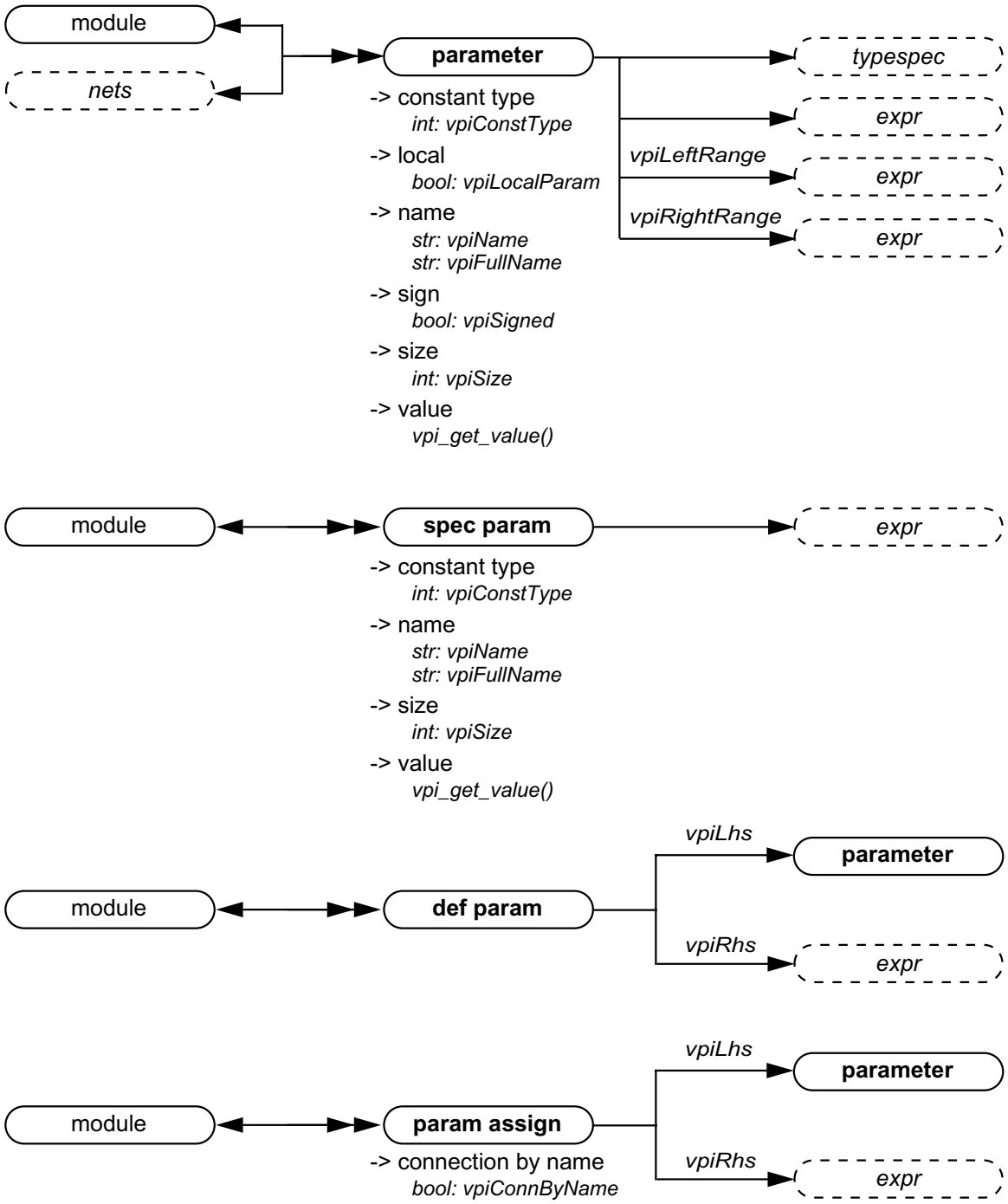
— The **vpiWaitingProcesses** iterator returns all waiting processes, identified by their frame, for that named event.



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.

32.20 Parameter (supersedes P1364 26.6.12)



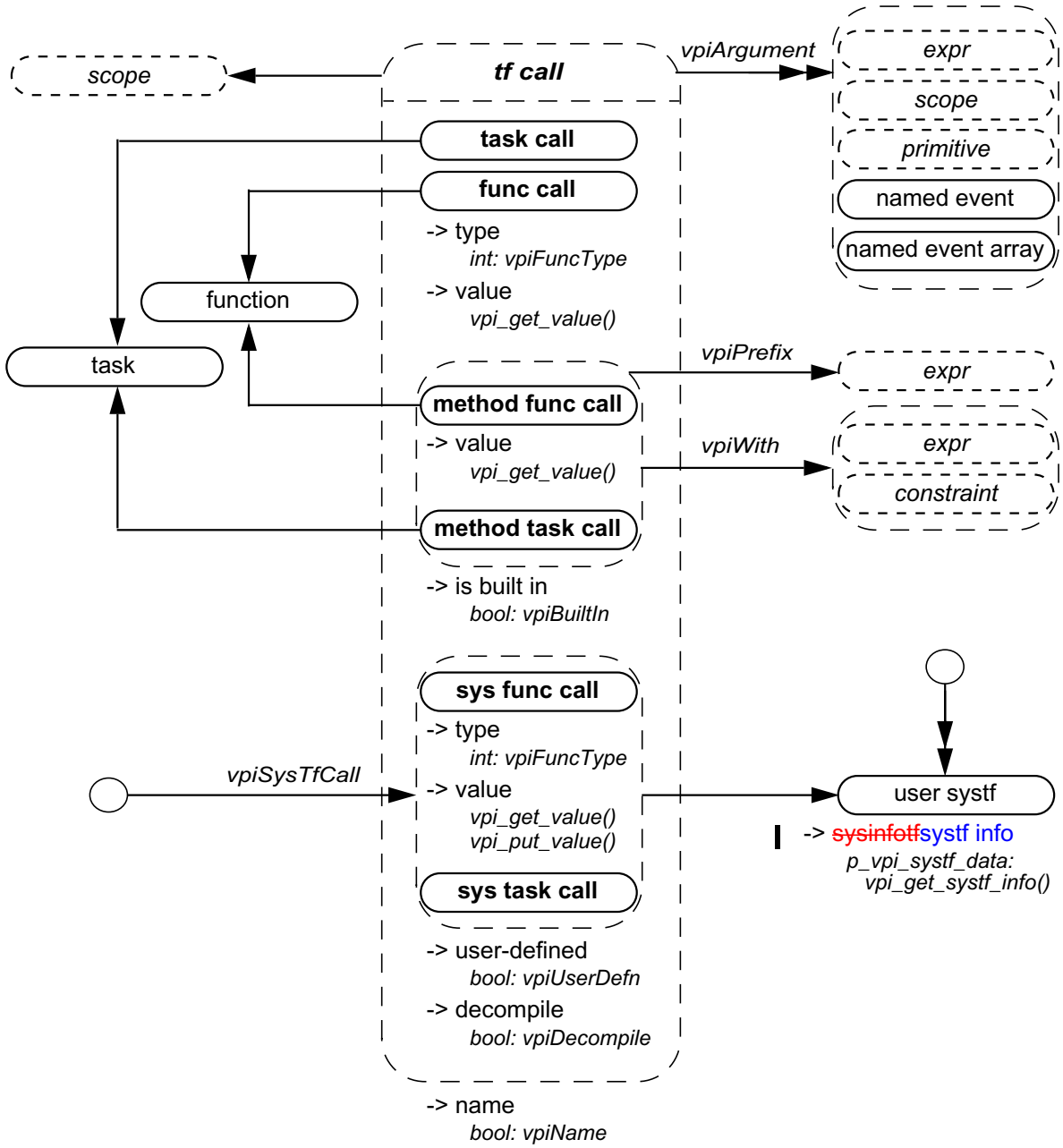
NOTES:

1) Obtaining the value from the object **parameter** shall return the final value of the parameter after all module instantiation overrides and defparams have been resolved.

2) **vpiLhs** from a param assign object shall return a handle to the overridden parameter.

3) If a parameter does not have an explicitly defined range or is a type parameter, **vpiLeftRange** and **vpiRightRange** shall return a NULL handle.

32.27 Task and function call (supersedes P1364 26.6.19)



NOTES:

1) ~~t~~The **vpiWith** relation is only available for randomize methods (see 13.6) and for array locator methods (see 5.15.1).

2) For methods (method func call, method task call), the **vpiPrefix** relation shall 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`".

3) The system task or function that invoked an application shall be accessed with **vpi_handle(vpiSysTfCall, NULL)**

4) **vpi_get_value()** shall return the current value of the system function.

5) If the **vpiUserDefn** property of a system task or function call is true, then the properties of the corresponding systf object shall be obtained via **vpi_get_systf_info()**.

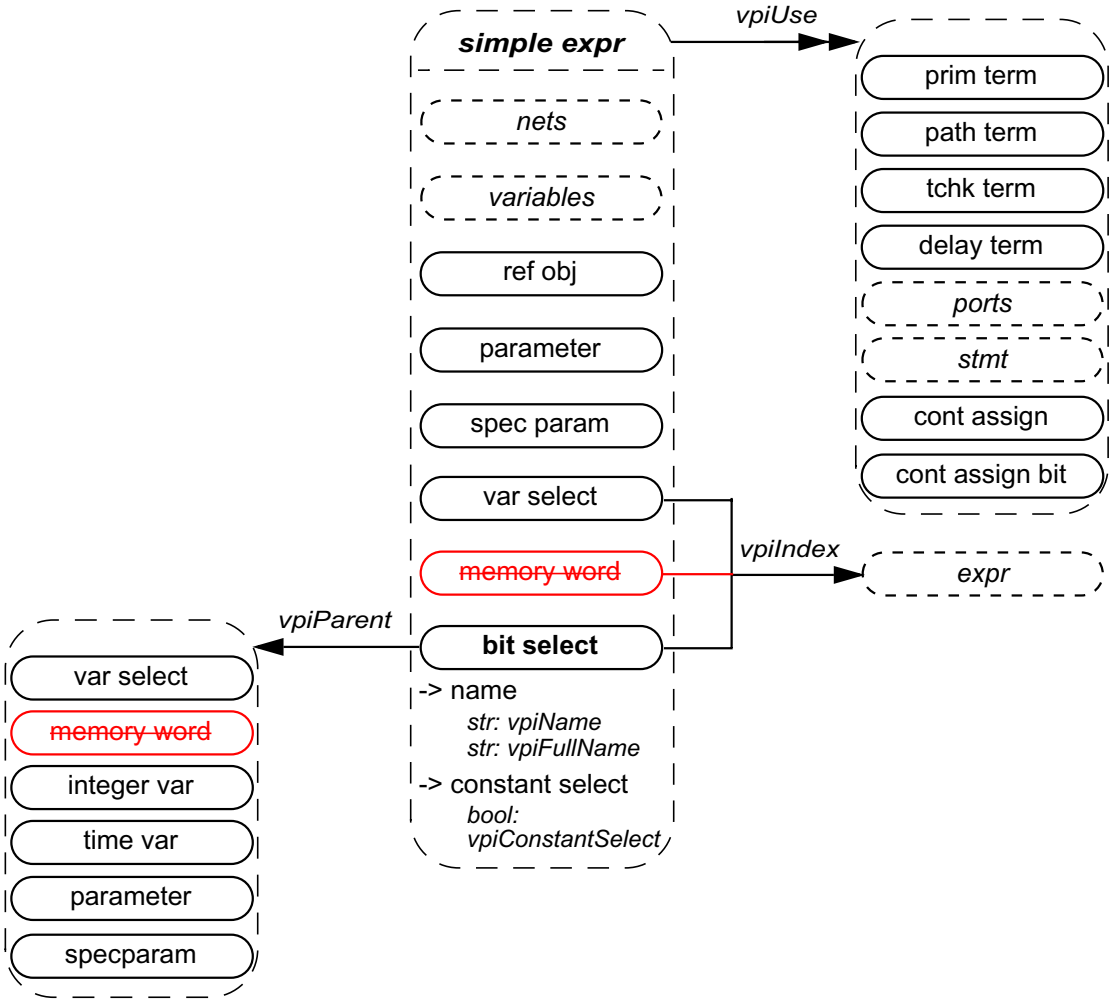
6) All user-defined system tasks or functions shall be retrieved using **vpi_iterate()**, with **vpiUserSystf** as the type argument, and a NULL reference argument.

7) Arguments to PLI tasks or functions are not evaluated until an application requests their value. Effectively, the value of any argument is not known until the application asks for it. When an argument is an HDL or system function call, the function cannot be evaluated until the application asks for its value. If the application never asks for the value of the function, it is never evaluated. If the application has a handle to an HDL or system function it may ask for its value at any time in the simulation. When this happens the function is called and evaluated at this time.

8) A null argument is an expression with a **vpiType** of **vpiOperation** and a **vpiOpType** of **vpiNullOp**.

9) The property **vpiDecompile** ~~will~~ shall return a string with a functionally equivalent system task or function call to what was in the original HDL. The arguments ~~will~~ shall be decompiled using the same manner as any expression is decompiled. See 32.39 for a description of expression decompilation.

32.38 Simple expressions (supersedes P1364 26.6.25)

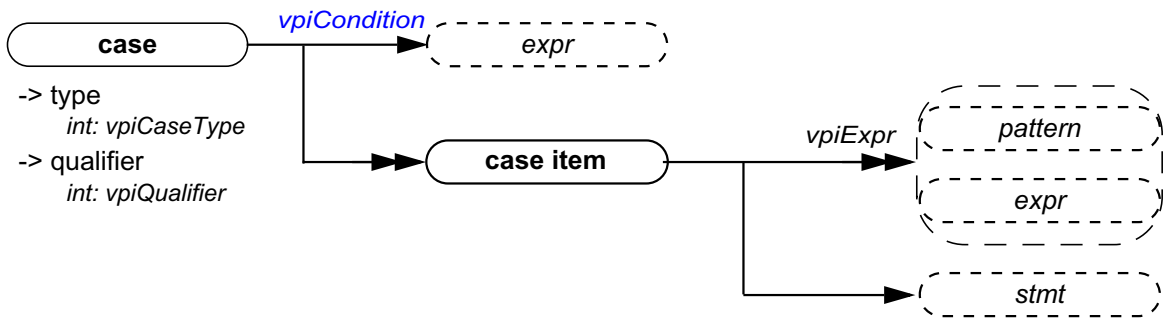


NOTES:

1) For vectors, the **vpiUse** relationship shall access any use of the vector or part-selects or bit-selects thereof.

2) For bit-selects, the **vpiUse** relationship shall access any specific use of that bit, any use of the parent vector, and any partselect that contains that bit.

32.47 Case, pattern (supersedes P1364 26.6.36)



NOTES:

1) The *case item* shall group all case conditions that branch to the same statement.

2) **vpi_iterate()** shall return `NULL` for the default case item since there is no expression with the default case.

Annex I
sv_vpi_user.h

Change:

```
#define vpiClassType 640
#define vpiMailboxClass 1
#define vpiSemaphoreClass 2
#define vpiUserDefineClass 3
```

To:

```
#define vpiClassType 640
#define vpiMailboxClass 1
#define vpiSemaphoreClass 2
#define vpiUserDefinedClass 3
```

Annex I

(normative)

REPLACE

```
#define vpiVisibility            620
#define vpiPublicVis           1
#define vpiProtectedVis        2
#define vpiLocalVis            3

#define vpiNullConst             622
#define vpiOneStepConst          623

#define vpiAlwaysType            624
#define vpiAlwaysComb          1
#define vpiAlwaysFF            2
#define vpiAlwaysLatch         3

#define vpiDistType              625
#define vpiEqualDist           1 /* constraint equal distribution */
#define vpiDivDist             2 /* constraint divided distribution */

#define vpiPacked                630
#define vpiTagged                632
#define vpiRef                   633
#define vpiDefaultSkew           634
#define vpiVirtual               635
#define vpiUserDefined           636
#define vpiIsConstraintEnabled   638

#define vpiClassType             640
#define vpiMailboxClass        1
#define vpiSemaphoreClass      2
#define vpiUserDefineClass     3

#define vpiMethod                645
#define vpiIsClockInferred       649
```

WITH

```
#define vpiVisibility            620
#define vpiPublicVis           1
#define vpiProtectedVis        2
#define vpiLocalVis            3

#define vpiNullConst             622
#define vpiOneStepConst          623
/* Return values for vpiConstType property */
#define vpiNullConst         8
#define vpiOneStepConst      9
```

```
#define vpiAlwaysType             624
#define vpiAlwaysComb          1
#define vpiAlwaysFF            2
#define vpiAlwaysLatch         3

#define vpiDistType               625
#define vpiEqualDist           1 /* constraint equal distribution */
#define vpiDivDist             2 /* constraint divided distribution */

#define vpiPacked                 630
#define vpiTagged                 632
#define vpiRef                    633

#define vpiRef                6 /* Return value for vpiDirection property */

#define vpiDefaultSkew            634
#define vpiVirtual                635
#define vpiUserDefined            636
#define vpiIsConstraintEnabled    638

#define vpiClassType              640
#define vpiMailboxClass        1
#define vpiSemaphoreClass      2
#define vpiUserDefineClass     3

#define vpiMethod                 645
#define vpiIsClockInferred        649
```

Section **29.2**

REMOVE section 29.2 completely

~~29.2 Extensions to VPI enumerations~~

~~29.2.1 Object types~~
~~This subclause lists the object type VPI calls.~~
~~vpiAssertion /* assertion */~~
~~Refer to 29.3.1 for details on how to obtain assertion handles using this object type.~~

~~29.2.2 Object properties~~
~~This subclause lists the object property VPI calls.~~
~~/* Assertion types */~~
~~vpiSequenceInst~~
~~vpiAssert~~
~~vpiAssume~~
~~vpiCover~~
~~vpiPropertyInst~~
~~vpiImmediateAssert~~
~~Refer to 29.3.1for details on how to obtain handles for a specific assertion type using this object type.~~

~~29.2.3 Callbacks~~
~~This subclause lists the system callbacks. Refer to 29.4.2 for details on placing assertion callbacks, and to 29.4.1 for details on placing assertion callbacks for the assertion system.~~
~~1) Assertion~~
~~cbAssertionStart~~
~~cbAssertionSuccess~~
~~cbAssertionFailure~~
~~cbAssertionStepSuccess~~
~~cbAssertionStepFailure~~
~~cbAssertionDisable~~
~~cbAssertionEnable~~
~~cbAssertionReset~~
~~cbAssertionKill~~
~~2) ‘‘Assertion system’’~~
~~cbAssertionSysInitialized~~
~~cbAssertionSysStart~~
~~cbAssertionSysStop~~
~~cbAssertionSysEnd~~
~~cbAssertionSysReset~~
~~29.2.4 Control constants~~
~~This subclause lists the system control constant callbacks. Refer to 29.5 for details on how to use~~
~~vpi_control() with these constants to obtain assertion control information, and control the assertion system.~~
~~1) Assertion~~
~~vpiAssertionDisable~~
~~vpiAssertionEnable~~
~~vpiAssertionReset~~

~~vpiAssertionKill~~
~~vpiAssertionEnableStep~~
~~vpiAssertionDisableStep~~

~~2) Assertion stepping~~

~~vpiAssertionClockSteps~~

~~3) ‘‘Assertion system’’~~

~~vpiAssertionSysStart~~
~~vpiAssertionSysStop~~
~~vpiAssertionSysEnd~~
~~vpiAssertionSysReset~~

Section **29.1.1 & 30.1.2**

REMOVE the two sections 29.1.1 and 30.1.2 completely

~~29.1.1 Naming conventions~~
~~All elements added by this interface shall conform to the Verilog Procedural Interface (VPI) interface naming conventions.~~

- ~~All names are prefixed by vpi.~~
- ~~All type names shall start with vpi, followed by initially capitalized words with no separators, e.g., vpiAssertCheck.~~
- ~~All callback names shall start with cb, followed by initially capitalized words with no separators, e.g., cbAssertionStart.~~
- ~~All function names shall start with vpi_, followed by all lowercase words separated by underscores (_), e.g., vpi_get_assert_info().~~

~~**30.1.2 Naming conventions**~~
~~All elements added by this interface shall conform to the Verilog Procedural Interface (VPI) interface naming conventions.~~

- ~~All names are prefixed by vpi.~~
- ~~All *type names* shall start with vpi, followed by initially capitalized words with no separators, e.g., vpiCoverageStmt.~~
- ~~All callback names shall start with cb, followed by initially capitalized words with no separators, e.g., cbAssertionStart.~~
- ~~All *function names* shall start with vpi_, followed by all lowercase words separated by underscores (_), e.g., vpi_control().~~

ERRATA 514: Section 14.3 mailbox type mismatch

14.3.5 get()

Change:

If ~~there isa~~ the type ~~mismatch between~~ of the *message* variable is not equivalent to the type of ~~and~~ the message in the mailbox, a runtime error is generated.

14.3.6 try_get()

Change:

The try_get() method tries to retrieve one message from the mailbox. If the mailbox is empty, then the method returns 0. If ~~there isa~~ the type ~~mismatch between~~ of the *message* variable is not equivalent to the type of ~~and~~ the message in the mailbox, the method returns –1. If a message is available and the message type ~~matches~~ is equivalent to the type of the *message* variable, the message is retrieved and the method returns 1.

14.3.6 peek()

Change:

If ~~there isa~~ the type ~~mismatch between~~ of the *message* variable is not equivalent to the type of ~~and~~ the message in the mailbox, a runtime error is generated.

14.3.7 try_peek()

Change:

If ~~there isa~~ the type ~~mismatch between~~ of the *message* variable is not equivalent to the type of ~~and~~ the message in the mailbox, the method returns –1. If a message is available and its type is equivalent to the type of the *message* ~~type matches~~ variable, the message is copied and the method returns 1.

14.4 Parameterized mailboxes

Change:

This is a very powerful mechanism that, unfortunately, can also result in run-time errors due to type mismatches (types not equivalent) between a message and the type of the variable used to retrieve the message. Frequently, a mailbox is used to transfer a particular message type, and, in that case, it is useful to detect type mismatches at compile time.

The only difference between a generic (dynamic) mailbox and a parameterized mailbox is that for a parameterized mailbox, the compiler ensures that the calls to put, try_put, peek, try_peek, get and try_get methods ~~are~~use argument types equivalent to~~compatible with~~ the mailbox message type, so that all type mismatches are caught by the compiler and not at run-time.

•

32.34 Property specification

Change:

NOTES:

1) ...

2) Within the context of a property expr, **vpiOpType** can be any one of vpiNotOp, vpiImplyOp, vpiDelayedImplyOp, vpiAndOp, vpiOrOp, vpiIfOp, vpiIfElseOp. Operands to these operations shall be provided in the same order as shown in the BNF.

To:

NOTES:

1) ...

2) Within the context of a property expr, **vpiOpType** can be any one of vpiNotOp, vpiOverlapImplyOp, vpiNonOverlap~~Delayed~~ImplyOp, vpiCompAndOp, vpiCompOrOp, vpiIfOp~~,~~ or vpiIfElseOp. Operands to these operations shall be provided in the same order as shown in the BNF.

32.36 Sequence expression

Change:

NOTES:

1) Within a sequence expression, **vpiOpType** can be any one of vpiAndOp, vpiIntersectOp, vpiOrOp, vpiFirstMatchOp, vpiThroughoutOp, vpiWithinOp, vpiUnaryCycleDelayOp, vpiCycleDelayOp, vpiRepeatOp, vpiConsecutiveRepeatOp or vpiGotoRepeatOp.

To:

NOTES:

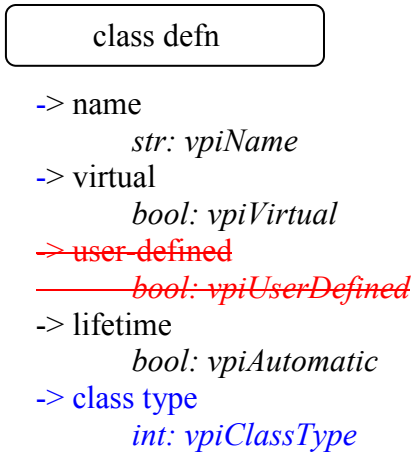
1) Within a sequence expression, **vpiOpType** can be any one of vpiCompAndOp, vpiIntersectOp, vpiCompOrOp, vpiFirstMatchOp, vpiThroughoutOp, vpiWithinOp, vpiUnaryCycleDelayOp, vpiCycleDelayOp, vpiRepeatOp, vpiConsecutiveRepeatOp or vpiGotoRepeatOp.

Annex I

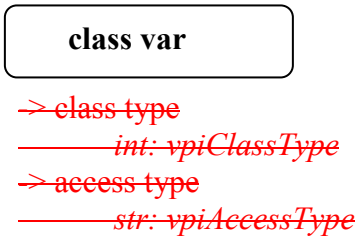
Add:

```
#define vpiAssignmentPatternOp      75      /* '{} assignment pattern */
#define vpiMultiAssignmentPatternOp 76      /* '{n{}} multi assignment pattern */

#define vpiIfOp                 77      /* if operator */
#define vpiIfElseOp             78      /* if...else operator */
#define vpiCompAndOp            79      /* Composite and operator */
#define vpiCompOrOp             80      /* Composite or operator */
```

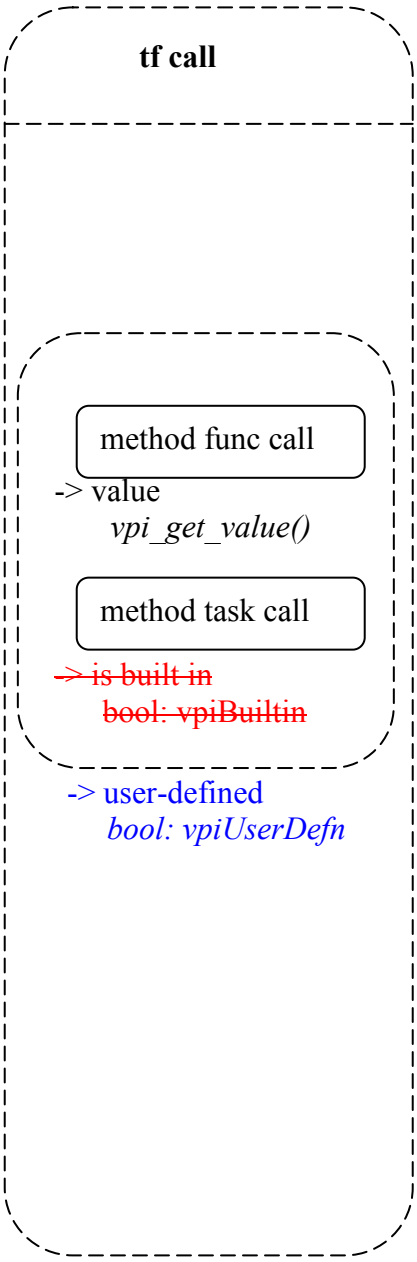
32.21 Class object definition



32.22 Class variables



32.27 Task and function call (supercedes P1364 26.6.19)

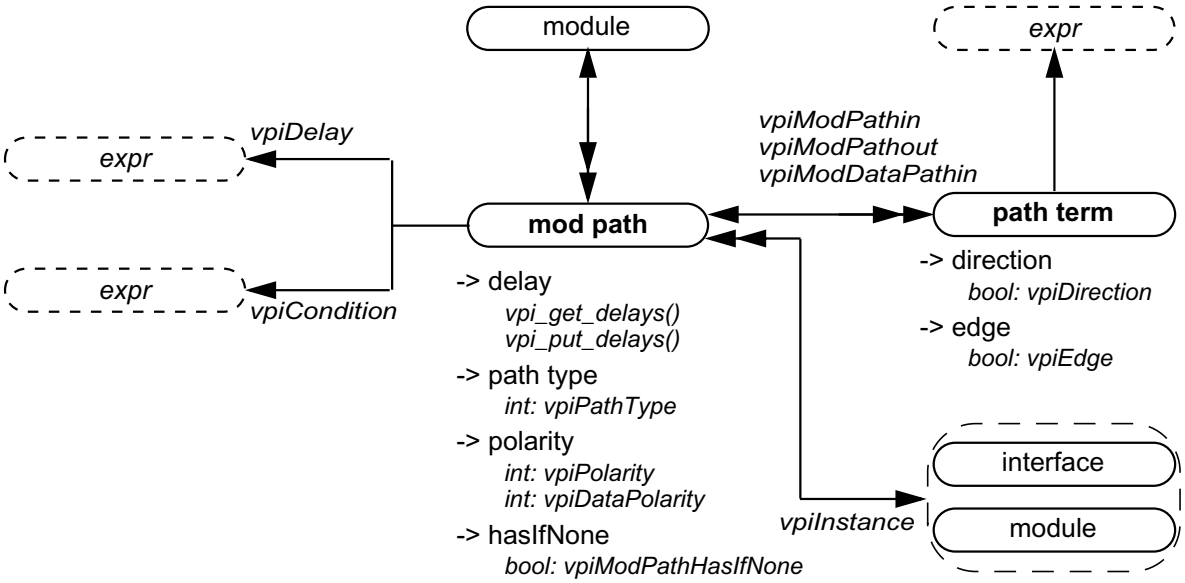


Annex I

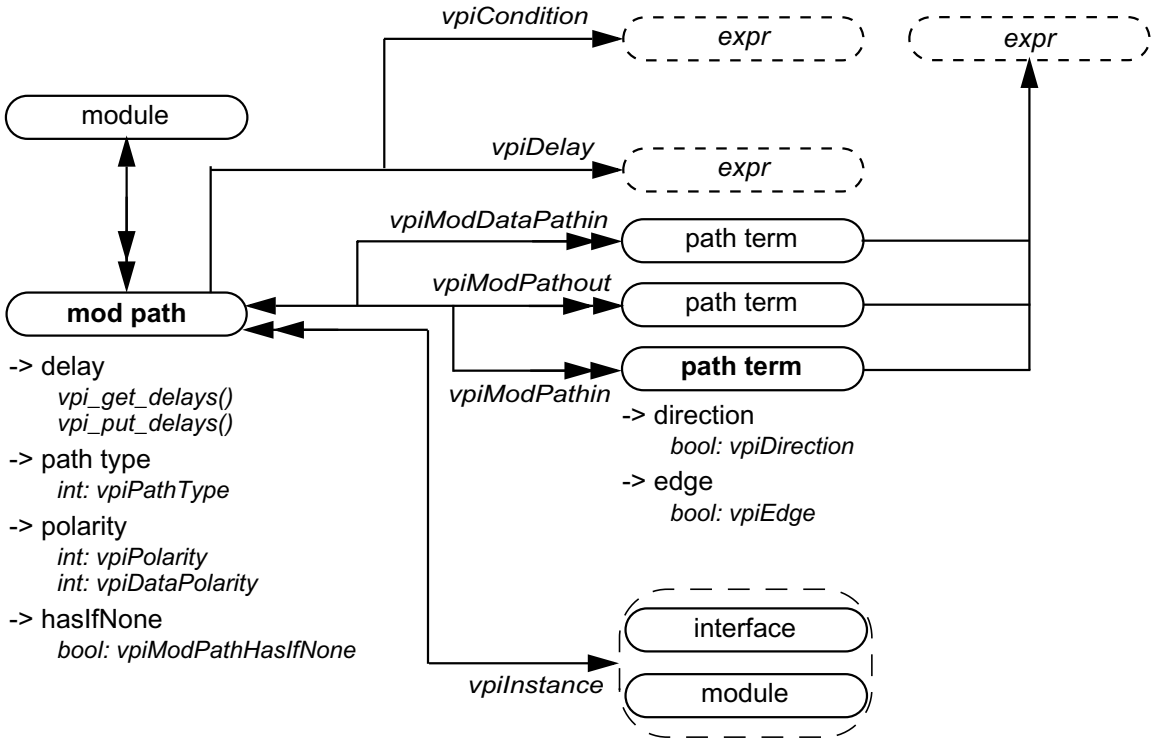
```
#define vpiPacked                630
#define vpiTagged                632
#define vpiRef                   633
#define vpiDefaultSkew           634
#define vpiVirtual               635
#define vpiUserDefined           636
#define vpiIsConstraintEnabled   638
```

32.25 Module path, path term (supersedes P1364 26.6.15)

REPLACE



WITH



32.23 Constraint, constraint ordering, distribution

NOTES:

~~-- vpiDistType can be one of the following:~~ **~~vpiEqualDist~~** ~~or~~ **~~vpiDivDist~~**

32.22 Class Variables

NOTES:

1) The **vpiWaitinProcesses** iterator on a mailbox shall 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 that 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
5) ~~**vpiClassType** can be **vpiUserDefinedClass**, **vpiMailboxClass**, **vpiSemaphoreClass**~~
6) ~~**vpiAccessType** can be one of **vpiPublicAcc**, **vpiProtectedAcc**, **vpiLocalAcc**~~

Section 32.21 Class object definition

REPLACE

NOTES

1) ....

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

WITH

1) ....

~~2) Should not call **vpi_get_value**/**vpi_put_value** on the non-static variables obtained from the class definition handle.~~

2) **vpi_get_value** and **vpi_put_value** are not allowed for non-static variable handles obtained from class defn handles.

Section 32.21 Class object definition

REPLACE

NOTES

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) The iterator to constraints returns only normal constraints and not inline constraints.

4) To get constraints inherited from base classes, it is necessary to traverse the extend relation to obtain the base class.

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

6) The relation to **vpiExtend** exists whenever a one class is derived from another class (refer to 12.12). 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 (refer to 12.16 and 12.23)

WITH

~~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.

2) The iterator to constraints returns only normal constraints and not inline constraints.

3) To get constraints inherited from base classes, it is necessary to traverse the extend relation to obtain the base class.

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

5) The relation to **vpiExtend** exists whenever a one class is derived from another class (refer to 12.12). 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 (refer to 12.16 and 12.23)

Section 32.21 Class object definition

REPLACE

NOTES

1) ....

2) ...

.....

6) The relation to **vpiExtend** exists whenever a one class is derived from another class (refer to 12.12). 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 (refer to 12.16 and 12.23)

WITH

1) ....

2) ....

6) The relation to ~~**vpiExtend**~~ **vpiExtends** exists whenever ~~a~~ one class is derived from another class (refer to 12.12). The relation from ~~extend~~ extends to ~~ClassDefn~~ class defn provides the base class. The iterators from ~~extend~~ extends to param assign and arguments provide the parameters and arguments used in constructor chaining (refer to 12.16 and 12.23)

Annex I

(normative)

sv_vpi_user.h

REPLACE

#define vpiExtend 677

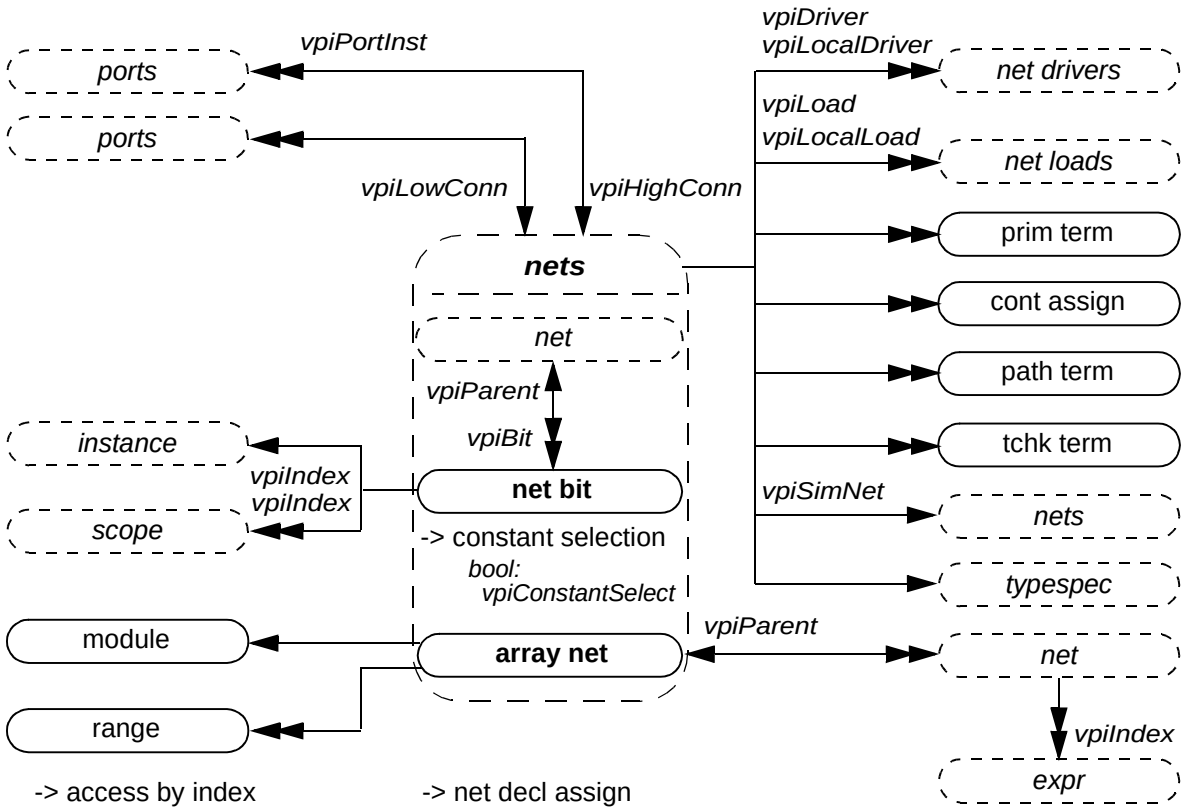
WITH

~~#define vpiExtend 677~~

#define vpiExtends 677

REPLACE:

Nets (supersedes P1364 26.6.6)



-> access by index
vpi_handle_by_index
vpi_handle_by_multi_index

-> array member
bool: vpiArray (deprecated)
bool: vpiArrayMember

-> delay
vpi_get_delays()

-> expanded
bool: vpiExpanded

-> implicitly declared
bool: vpiImplicitDecl

-> name
str: vpiName
str: vpiFullName

-> net decl assign
bool: vpiNetDeclAssign

-> net type
int: vpiNetType
int: vpiResolvedNetType

-> scalar
bool: vpiScalar

-> scalared declaration
bool: vpiExplicitScalared

-> sign
bool: vpiSigned

-> size
int: vpiSize

-> strength
int: vpiStrength0
int: vpiStrength1
int: vpiChargeStrength

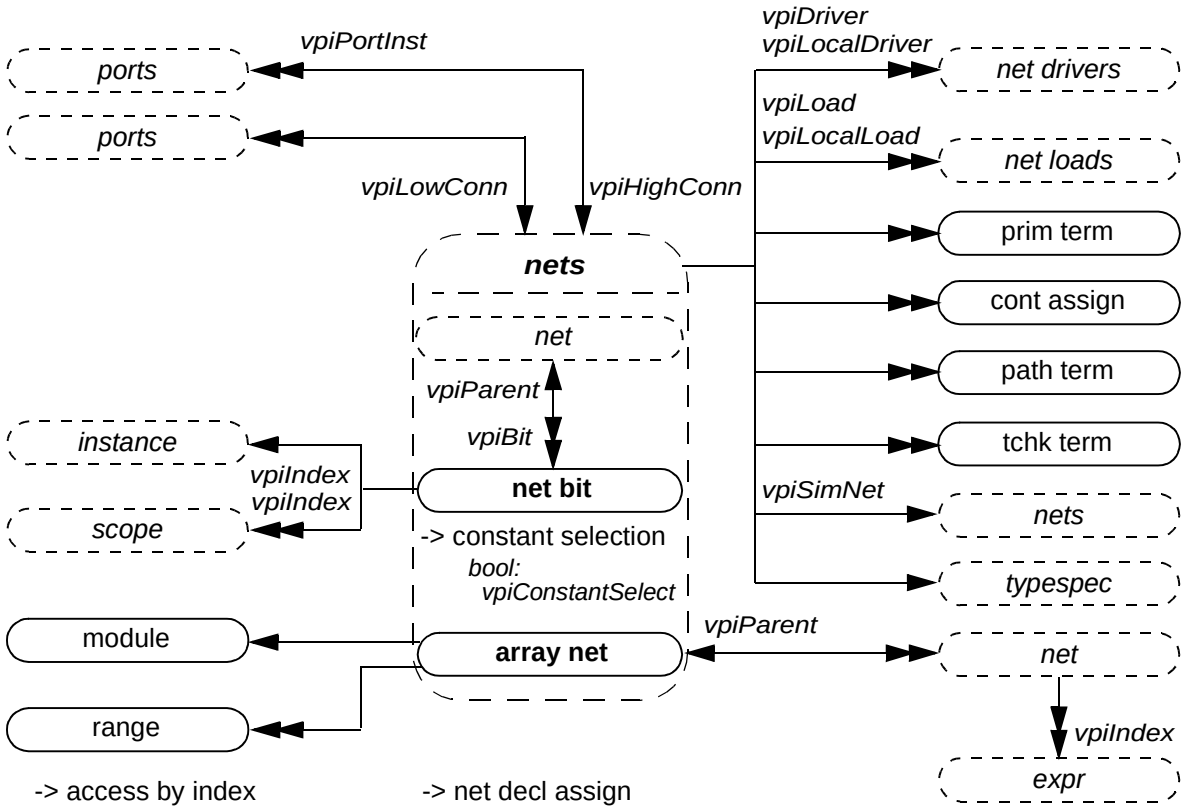
-> value
vpi_get_value()
vpi_put_value()

-> vector
bool: vpiVector

-> vectored declaration
bool: vpiExplicitVectored

WITH:

32.13 Nets (supersedes P1364 26.6.6)



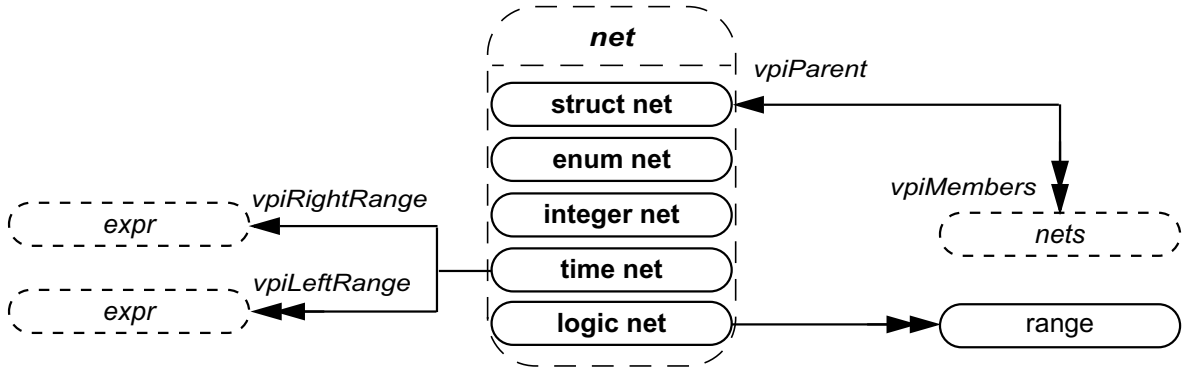
-> access by index
vpi_handle_by_index()
vpi_handle_by_multi_index()
-> array member
bool: vpiArray (deprecated)
bool: vpiArrayMember
-> delay
vpi_get_delays()
-> expanded
bool: vpiExpanded
-> implicitly declared
bool: vpiImplicitDecl
-> name
str: vpiName
str: vpiFullName

-> net decl assign
bool: vpiNetDeclAssign
-> net type
int: vpiNetType
int: vpiResolvedNetType
-> scalar
bool: vpiScalar
-> scalared declaration
bool: vpiExplicitScalared
-> sign
bool: vpiSigned
-> size
int: vpiSize

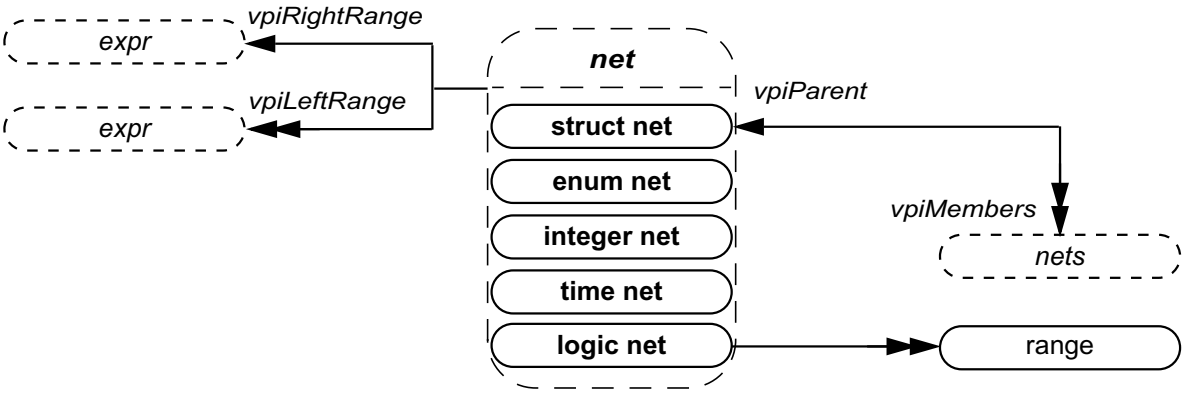
-> strength
int: vpiStrength0
int: vpiStrength1
int: vpiChargeStrength
-> value
vpi_get_value()
vpi_put_value()
-> vector
bool: vpiVector
-> vectored declaration
bool: vpiExplicitVectored

32.13 Nets (supersedes P1364 26.6.6)

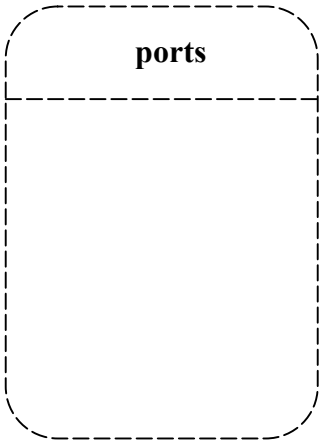
REPLACE



WITH

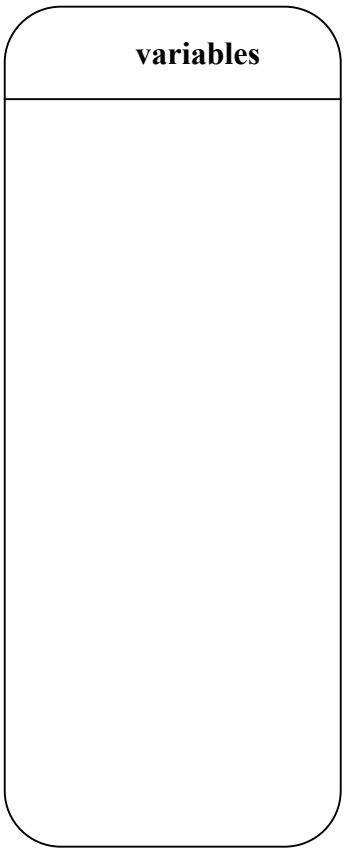


32.11 Ports (supercedes P1364 26.6.5)



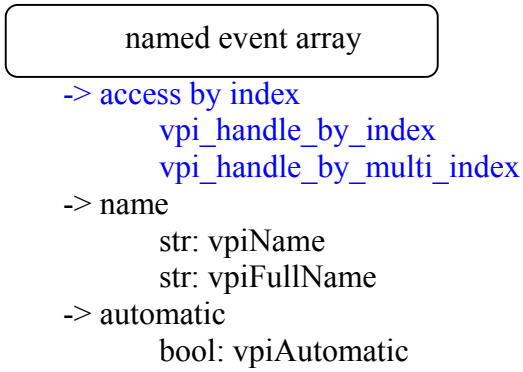
-> access by index
    vpi_handle_by_index
    vpi_handle_by_multi_index
-> connected by name
    bool: vpiConnByName
-> delay (mipd)
    vpi_get_delays()
    vpi_put_delays()
-> direction
    int: vpiDirection
-> explicitly named
    bool: vpiExplicitName
-> index
    int: vpiPortIndex

32.14 Variables (supercedes P1364 26.6.7 and 26.6.8)



-> access by index
    vpi_handle_by_index
    vpi_handle_by_multi_index
-> array member
    bool: vpiArray (deprecated)
    bool: vpiArrayMember
-> name
    str: vpiName
    str: vpiFullName
-> sign
    bool: vpiSigned
-> size
    int: vpiSize

32.19 Named events (supercedes P1364 26.6.11)



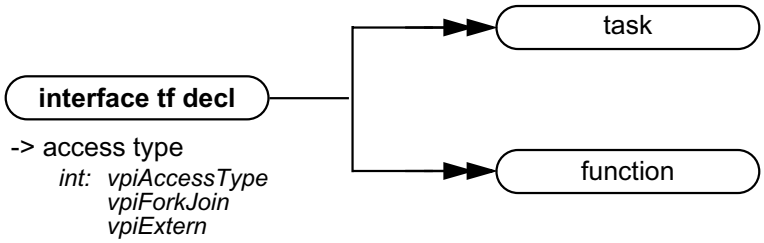
Annex I, vpiPortType

In Annex I, MODIFY the text as follows:

```
#define vpiRandType             610
#define vpiNotRand          1
#define vpiRand             2
#define vpiRandC            3
#define vpiPortType             611
#define vpiInterfacePort    1
#define vpiModportPort      2
/* vpiPort is also a port type.  It is defined in vpi_user.h */
```

32.5 Interface task/function declaration

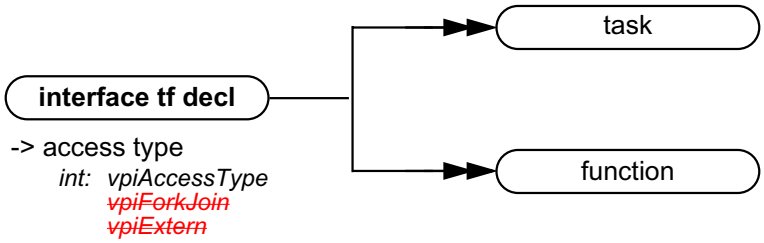
REPLACE



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.

WITH



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.

— Possible return values for the **vpiAccessType** property for an *interface tf decl* are **vpiForkJoin** and **vpiExtern**.

Section 28.4.6, Types of Formal Arguments, and Annex E.6, Various Sections

In 28.4.6, MODIFY the text as follows:

~~— packed one dimensional arrays of type bit and logic~~
~~Note however, that every packed type, whatever is its structure, is eventually equivalent to a packed one dimensional array. Therefore practically all packed types are supported, although their internal structure (individual fields of structs, multiple dimensions of arrays) shall be transparent and irrelevant.~~

— packed arrays, structs, and unions composed of types bit and logic.
Every packed type is eventually equivalent to a packed one dimensional array. On the foreign language side of the DPI interface, all packed types are perceived as packed one dimensional arrays regardless of their declaration in the SystemVerilog code.

In E.6.4, MODIFY the text as follows:

The DPI interface also supports the SystemVerilog and C unsigned integer data types that correspond to the mappings Table E-1 shows for their signed equivalents.

~~Note that i~~Input mode arguments of type byte unsigned and shortint unsigned are not equivalent to bit[7:0] or bit [15:0], respectively, since the former are passed as C types unsigned char and unsigned short and the latter are both passed by reference as ~~svBitPackedArrRef~~ svBitVecVal types. A similar lack of equivalence applies to passing such parameters by reference for output and inout modes, e.g. byte unsigned is passed as C type unsigned char* while bit [7:0] is passed by reference as ~~svBitPackedArrRef~~ svBitVecVal.

In addition to declaring DPI formal arguments of packed bit and logic arrays, it is also possible to declare formal arguments of packed struct and union types. DPI handles these types as if they were declared with equivalent one dimensional packed array syntax. See Section 6.9.2.

Refer to E.6.8 for details on unpacked aggregate types that are composed of the basic types described in this subclause.

ADD new Section E.10.3 as follows.
RENUMBER existing sections E.10.3 and E.10.4 to E.10.4 and E.10.5, respectively.
RENUMBER existing examples 5 through 10 to be examples 6 through 11, respectively.

E.10.3 Example 5 – Using packed struct and union arguments

This example shows how packed struct and union arguments correspond to one dimensional packed array arguments.

```
// SV code
module m;
```

```
   typedef bit [2:0] A;
   typedef struct packed { bit a; bit b; bit c; } S;
   typedef union packed { A a; S s; } U;
   S s;
   U u;
   A a;

   // Import function takes three arguments
   import "DPI-C" function void foo8(input A fa, input S fs, input U fu);

   initial begin
      s.a = 1'b1;
      s.b = 1'b0;
      s.c = 1'b0;
      a = 3'b100;
      u.a = 3'b100;
      foo8(a, s, u);
   end

endmodule

/* C code */
void foo8(
   const svBitVecVal* fa,
   const svBitVecVal* fs,
   const svBitVecVal* fu)
{
    printf("fa is %d, fs is %d, fu is %d\n", *fa, *fs, *fu);
}
```

The output of the printf will be "fa is 4, fs is 4, fu is 4".

In Section E.10.1, MODIFY Example 4 as follows:

```
// SV code
module m;
   parameter W = 33;
   int abv1;
   bit [29:0] abv2;
   bit [W-1:0] abv3;

   // Two ways of handling 2-state packed array arguments
   import "DPI-C" void function void foo7(input int unsigned fbv1,
                                          input int unsigned fbv2,
                                          input bit [W-1:0] fbv3);
```

```
   initial
      foo7(abv1, abv2, abv3);
endmodule

/* C code */
void function foo7(unsigned int fbv1, unsigned int fbv2,
                   const svBitVecVal* fbv3)
{
   printf("fbv1 is %d, fbv2 is %d\n", fbv1, fbv2);
   /* Use of the 2-state svdpi utilities is needed to transform fbv3 into a
   C representation */
}
```

32.26 Task and function declaration (supersedes P1364 26.6.18)

REPLACE

NOTES:

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** shall always return a var object, even for simple returns.

WITH

NOTES:

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~~2) 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~~3) **vpiReturn** shall always return a var object, even for simple returns.

Section

•

Annex I

ADD (assuming changes in SV-CC 439 have already been made)

#define vpiNullConst 8

#define vpiOneStepConst 9

#define vpiUnboundedConst 10

32. SystemVerilog VPI Object Model

Note this proposal contains fixes for the following problems:

Diagram 32.7 is missing class defn as an instance item.

Diagram 32.9 errata # 528: missing type declaration iteration in class defn and other scope classes, missing class obj as a scope

Diagram 32.17 Typespec: added paramAssign from class typespec.

Diagram 32.21 did not allow access to virtual interface eclarations,I added vpiInterfaceDecl which returns a handle of type vpiRefObj This diagram also provides a clarification to vpiWaitingProcesses iteration for the class diagram.

Diagram 32.22 adds the class obj which represents the instantiation of a class, most relationships originate now from the class obj rather than from the class var. Added vpi_get_value and vpi_put_value for a class var.

Diagram 32.23 replace class var with class obj

32.1

32.2

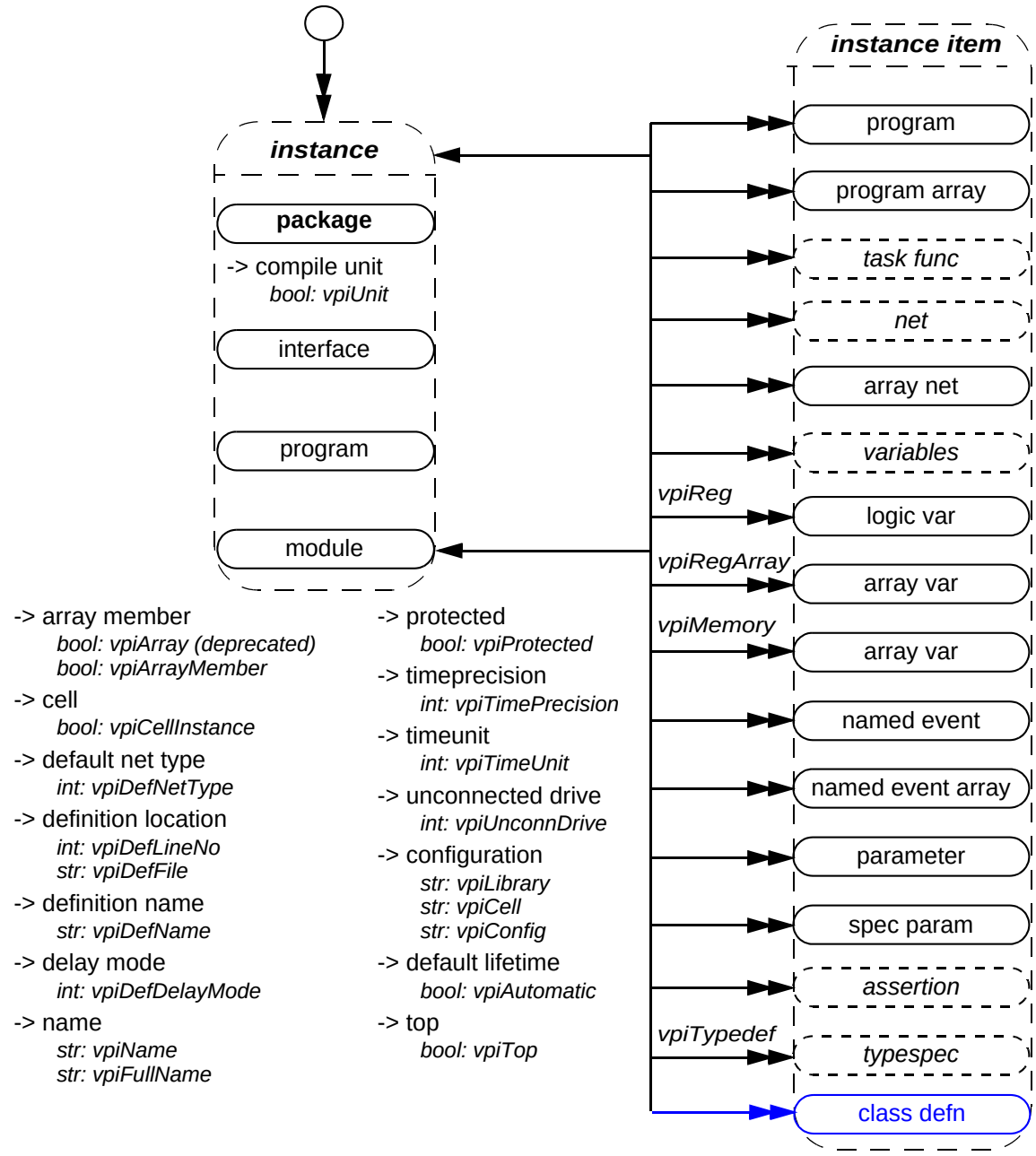
32.3

32.4

32.5

32.6 I

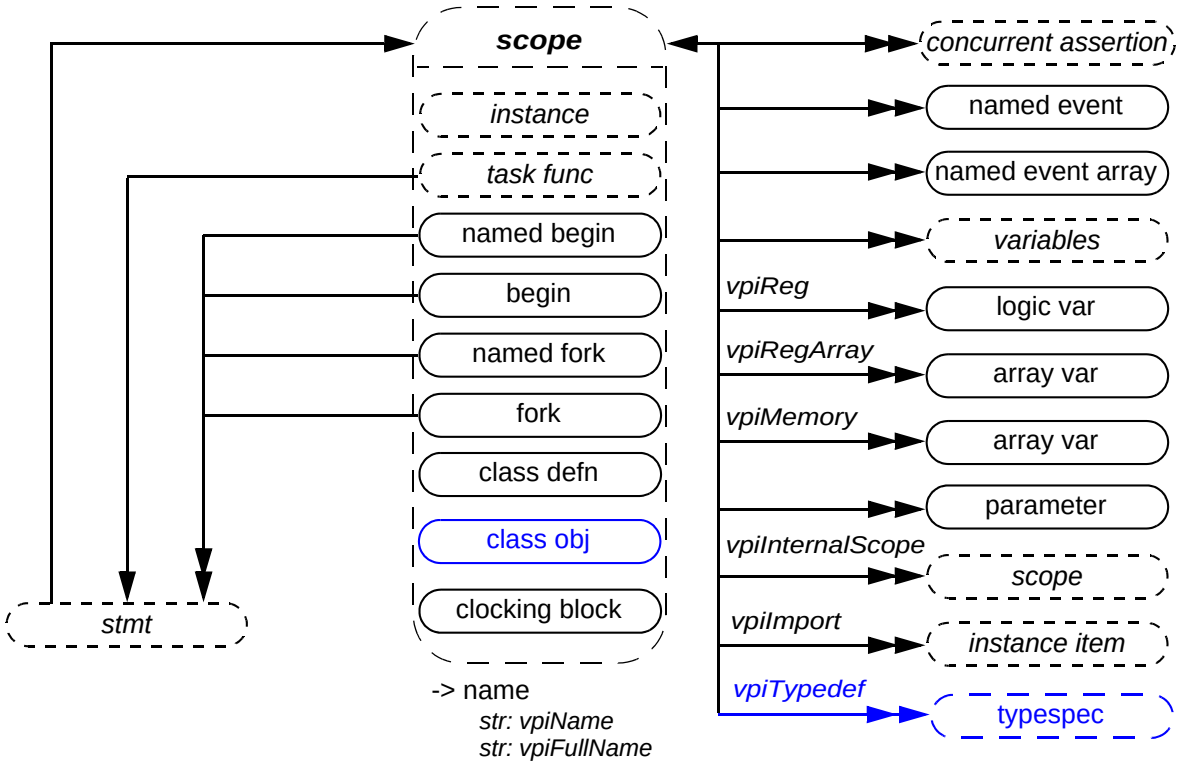
32.7 Instance



32.8

32.9 Scope (supersedes P1364 26.6.3)

Add iteration on vpiTypedef: fix errata #548



NOTES:

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 19.2.2 for more details.

3) A task func can have zero or more statements (see 11.2 and 11.3). If the number of statements is greater than 1, the **vpiStmt** relation shall return an unnamed **`begin`** that contains the statements of the task or function. If the number of statements is zero, the **vpiStmt** relation shall return `NULL`.

32.10

32.11

32.12

32.13

32.14

32.15

32.16

32.17 Typespec (added iteration on param assign from a class type spec)

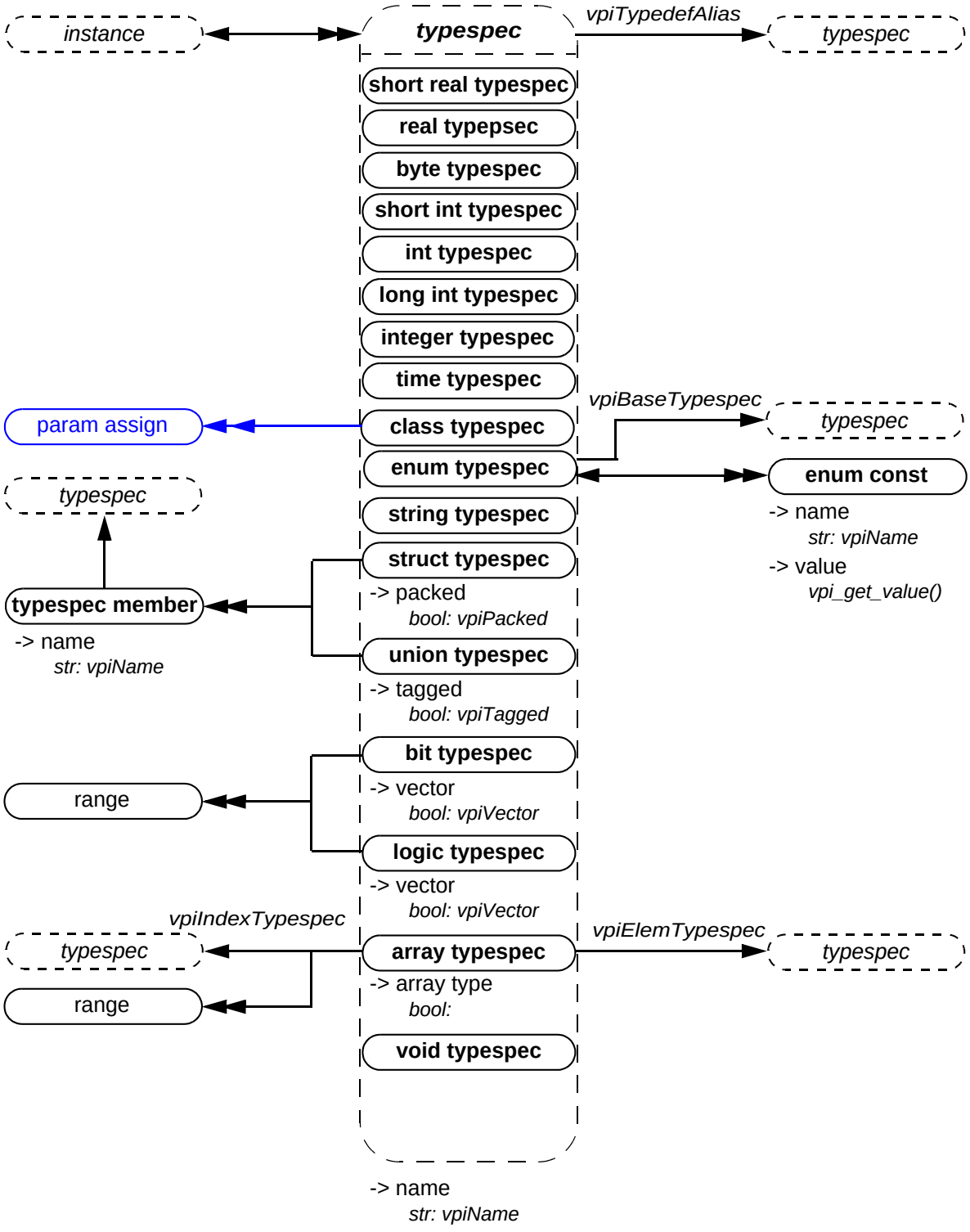
I moved the class typespec up in order to be able to draw the iteration on paramassign.

Note: this is necessary to model the following typedef or data types.

```
class vector #(int size = 1);
bit [size-1:0] a;
endclass
```

Instances of this class can then be instantiated:

```
vector #(10) vten; // object with vector of size 10
vector #(.size(2)) vtwo; // object with vector of size 2

typedef vector#(4) Vfour; // Class with vector of size 4
```



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 **vpiStructTypesec** 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**".

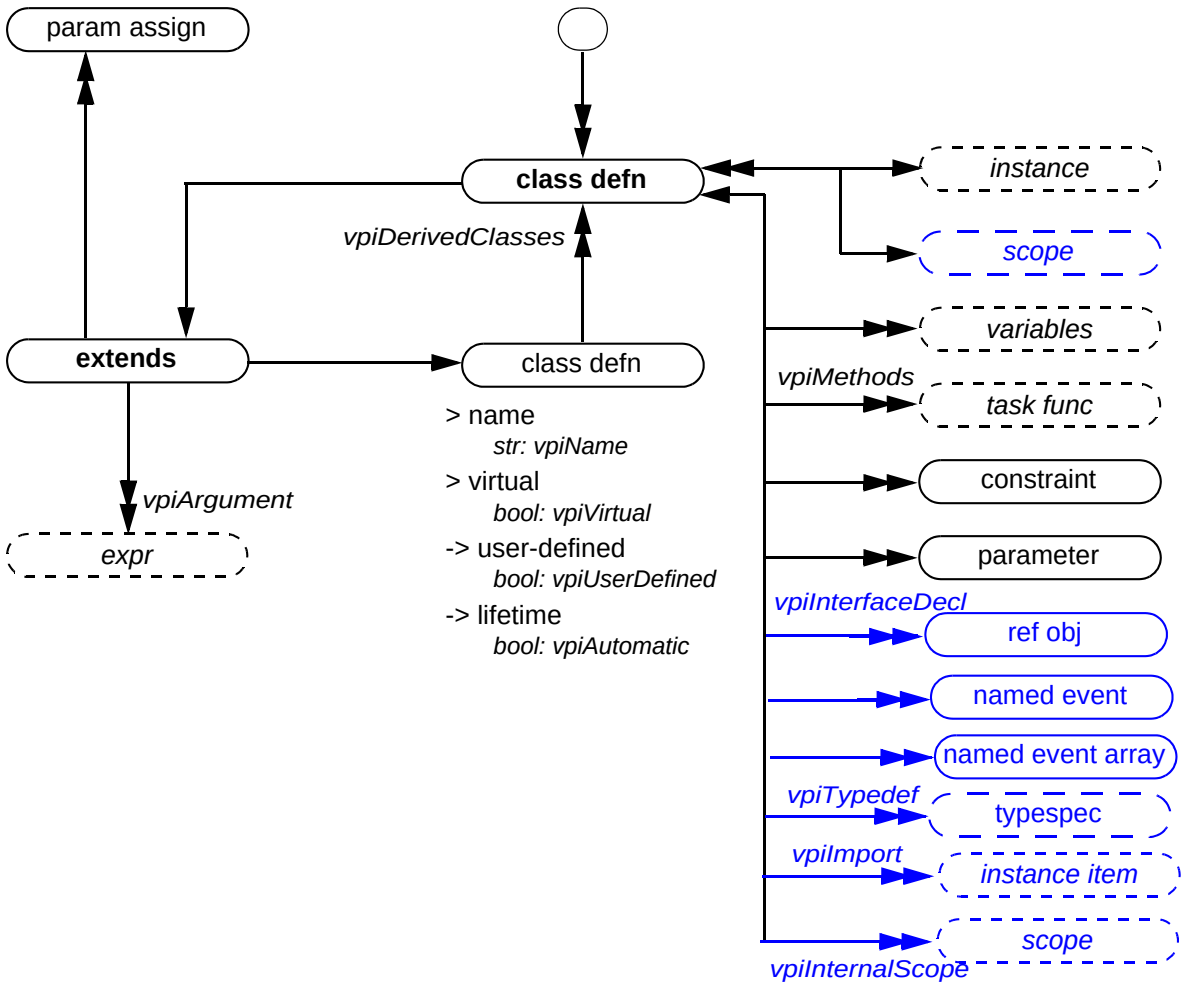
32.18

32.19

32.20

CHANGE title:

32.21 Class ~~object~~ definition



NOTES:

1) **class defn** 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) The iterator to constraints returns only normal constraints and not inline constraints.

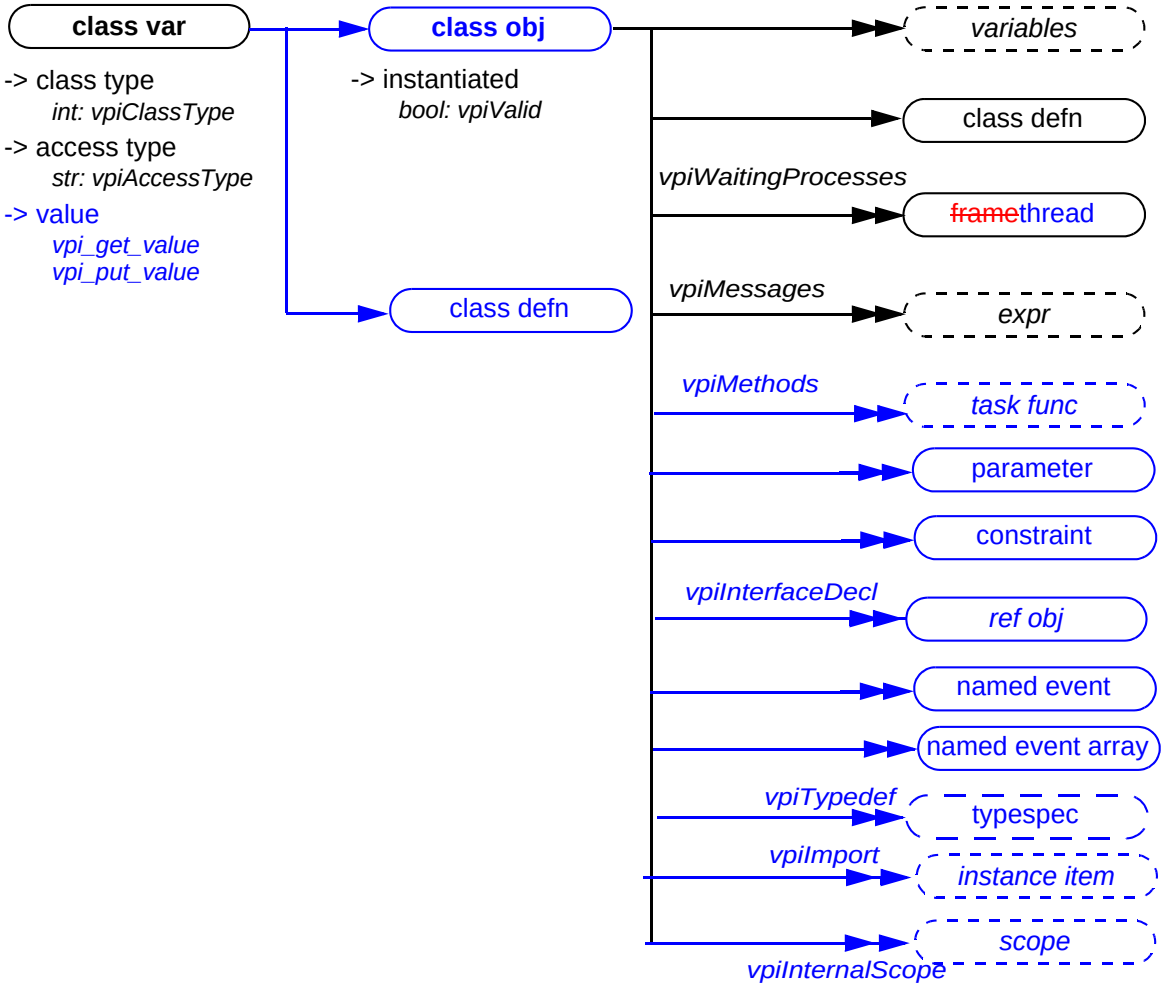
4) To get constraints inherited from base classes, it is necessary to traverse the extend relation to obtain the base class.

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

6) The relation to **vpiExtends** exists whenever one class is derived from another class (refer to 12.12). The relation from extends to class defn provides the base class. The iterators from extends to param assign and arguments provide the parameters and arguments used in constructor chaining (refer to 12.16 and 12.23)

7) The vpiInterfaceDecl iteration returns the virtual interface declarations in the class definition.

32.22 Class variables and class objects



NOTES:

1) The **vpiWaitingProcesses** iterator on a mailbox or semaphore shall ~~show~~ return the ~~processes~~ threads waiting on the class object or object resource.

   — ~~W~~A waiting process is ~~either~~ a static or dynamic process ~~frame,~~ represented by its suspended thread ~~or task/function handle~~. A process may be waiting to retrieve a message from a mailbox or waiting for a semaphore resource key.

2) **vpiMessage**s iteration~~or~~ shall return all the messages in a mailbox.

3) This note supercedes the change made for 473. Please change note 3 to the following: For a **class var**, **vpiClassDefn** returns the **class defn** with which the **class var** was declared in the SystemVerilog source text. If the **class var** has the value of null, the **vpiClassObj** relationship applied to the **class var s**hall return a null handle. **vpiClassDefn** when applied to a **class ob**j handle returns the **class defn** with which the **class obj** was created. The difference between the two usages of **vpiClassDefn** 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 **vpiClassDefn** of the **class obj** associated with variable p is “LinkedPacket”.

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

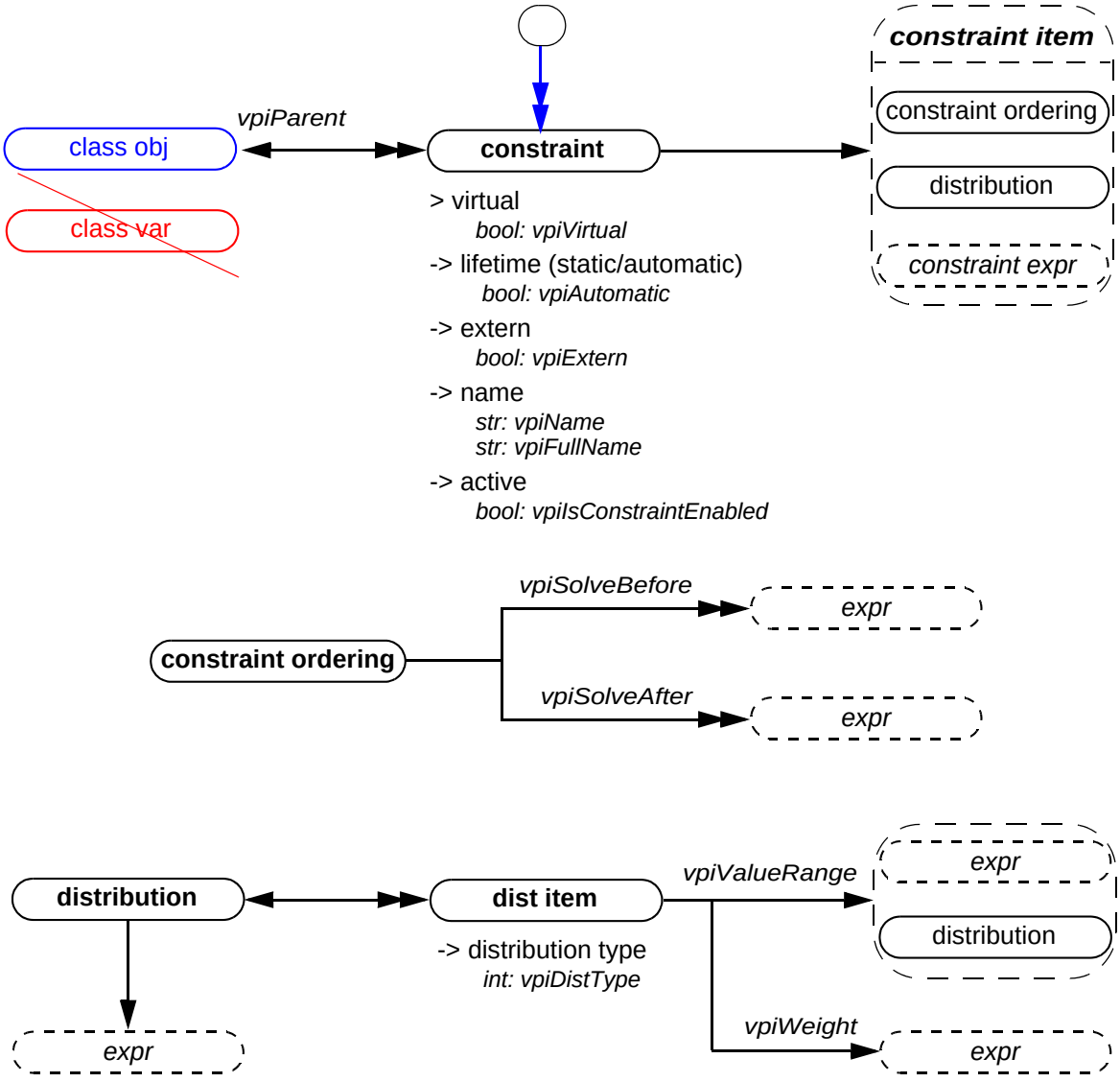
5) **vpiClassType** can be one of **vpiUserDefinedClass**, **vpiMailboxClass**, **vpiSemaphoreClass**.

6) **vpiAccessType** can be one of **vpiPublicAcc**, **vpiProtectedAcc**, **vpiLocalAcc**.

7) The **vpiInterfaceDecl** iteration returns the virtual interfaces of the class object.

8) **vpi_get_value()** when applied to a **class var** reference handle shall provide the value of the handle to the class object, or 0 if the **class var** is null.

32.23 Constraint, constraint ordering, distribution



NOTE:

— **vpiDistType** can be one of the following: **vpiEqualDist** or **vpiDivDist**.

32. Annex I

remove vpiActualDefn

~~#define vpiActualDefn 710~~

Stu please add a number for vpiClassObj (1 to 1 relationship)

#define vpiClassObj

ERRATA 510: What are the operators allowed on unpacked structs, unions and classes?

Add ta ble in section 8.2 after table *Syntax 8-1—Operator syntax (excerpt from Annex A)*

operator	name	datatypes
=	assignment operator	any
+= -= /= *=	C like assignment operators	integral, real, shortreal
%= &= |= ^= <<= >>= <<<= >>>=	bit wise assignment operators	integral
? :	conditional expression	any
+ -	unary arithmetic operators	integral, real, shortreal
!	unary logical operator	integral, real, shortreal
~ & ~& | ~| ^ ~^ ^~	unary logical operators	integral
+ - * / **	arithmetic binary operators	integral, real, shortreal

operator	name	datatypes
% & | ^ ^~ ~^ >> << >>> <<<	binary logical operators	integral
! && ||	other binary logical operators	integral, real, shortreal
< <= > >=	relational operators	integral, real, shortreal
=== !==	case equality operators	any except real and shortreal
== !=	logical equality operators	any
==? !=?	wild card equality operators	integral
++ --	increment, decrement operators	integral, real, shortreal
inside	set membership operator	singular for the left operand
{} {{}}	concatenation, replication operators	integral
dist	distribution operator	integral

Operator and datatypes

Section **22.2**
$isunbounded()

In 22.2, REPLACE

To support whether a constant is **$**, a system function is provided to test whether a constant is a **$**. The syntax of the system function is

WITH

~~To support whether a constant is **$**, a~~ A system function is provided to test whether a constant is a **$**. The syntax of the system function is

Section **22.2**
Parameters

In 22.2, REPLACE

SystemVerilog adds the ability for a parameter to also specify a data type, allowing modules or instances to have data whose type is set for each instance.

WITH

Unlike nonlocal parameters, local parameters can be declared in a generate block, in a package, or in a compilation unit scope. In these contexts, the **parameter** keyword can be used as a synonym for the **localparam** keyword.

SystemVerilog adds the ability for a parameter to also specify a data type, allowing modules or instances to have data whose type is set for each instance.

In 22.2, REPLACE

It is an error to override a type parameter with a **defparam** statement.

Unlike nonlocal parameters, local parameters can be declared in a generate block, in a package, or in a compilation unit scope. In these contexts, the **parameter** keyword can be used as a synonym for the **localparam** keyword.

WITH

It is an error to override a type parameter with a **defparam** statement.

~~Unlike nonlocal parameters, local parameters can be declared in a generate block, in a package, or in a compilation unit scope. In these contexts, the **parameter** keyword can be used as a synonym for the **localparam** keyword.~~

Section

NOTE:

- **vpiOpType** shall return **vpiAssignmentOp** for normal non-blocking '=' assignments, and the operator combined with the assignment for the operators described in 8.3.

  For example, the assignment
  A[i] += 2;
  shall return **vpiAddOp** for the **vpiOpType** property.

CHANGE TO:

NOTE:

- **vpiOpType** shall return **vpiAssignmentOp** for normal assignments (both blocking '=' , non-blocking '<='), or the vpiOpType of the operator combined with the assignment for the operators described in 8.3.

  For example, the assignment
  A += 2;
  shall return **vpiAddOp** for the **vpiOpType** property.

•

Section 32.40

REPLACE

32.40 Atomic statement (supersedes P1364 26.6.27)

WITH

32.40 Atomic statement (supersedes atomic stmt in P1364 26.6.27)

•

Section 32.40

REPLACE (text in object reference)

expect

WITH

expect stmt

REPLACE (text in object reference)

foreach

WITH

foreach stmt

Annex I

ADD

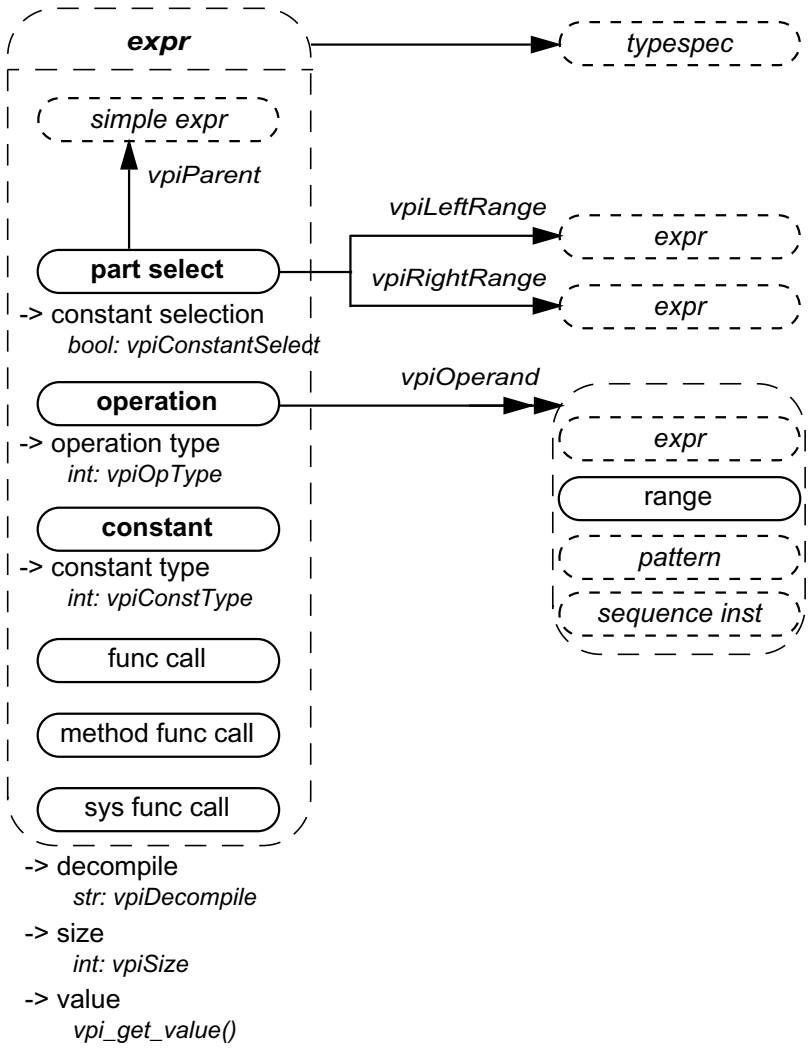
#define vpiStructNet 683

#define vpiBreak 684

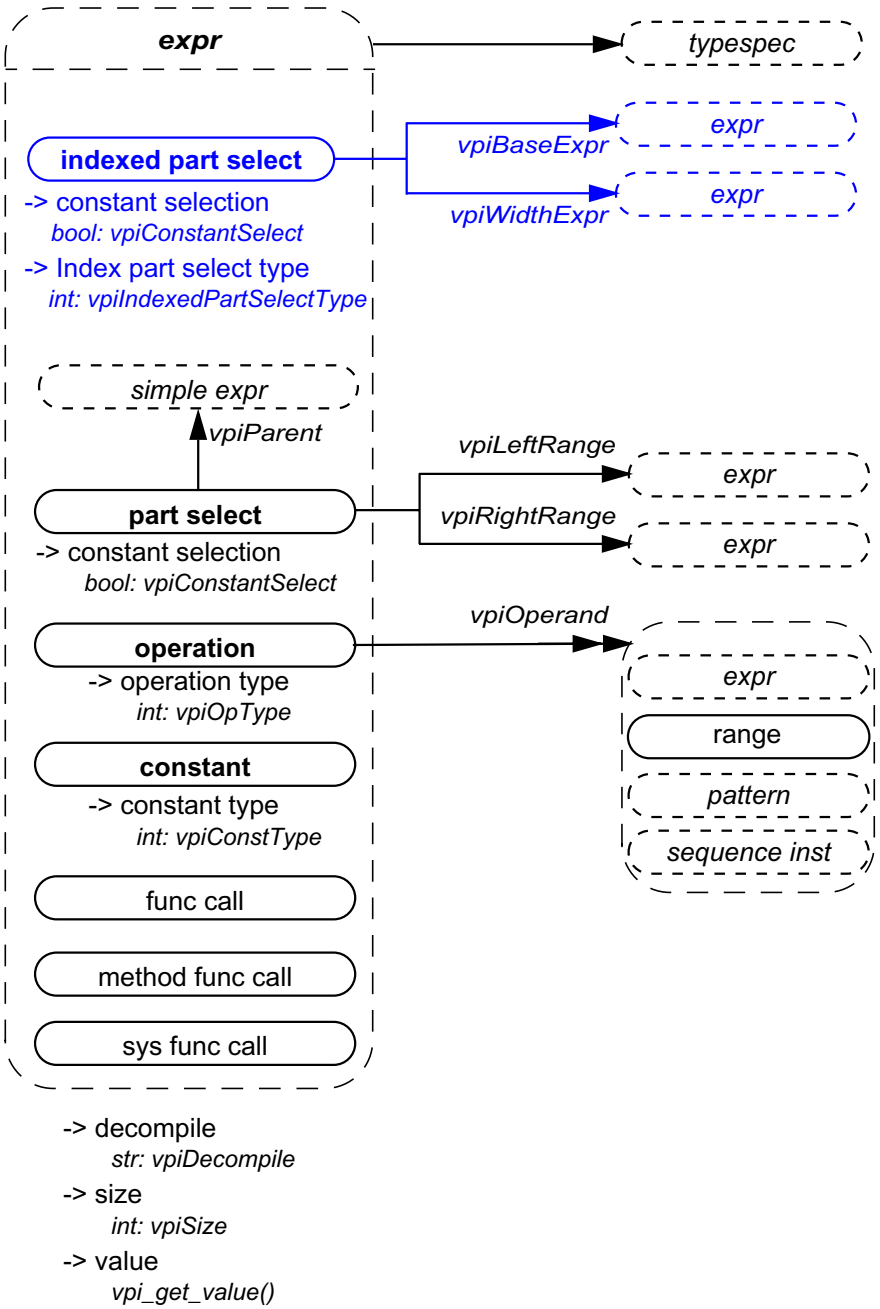
#define vpiContinue 685

32.13 Expressions (supersedes P1364 26.6.26)

REPLACE



WITH



32. SystemVerilog VPI Object Model

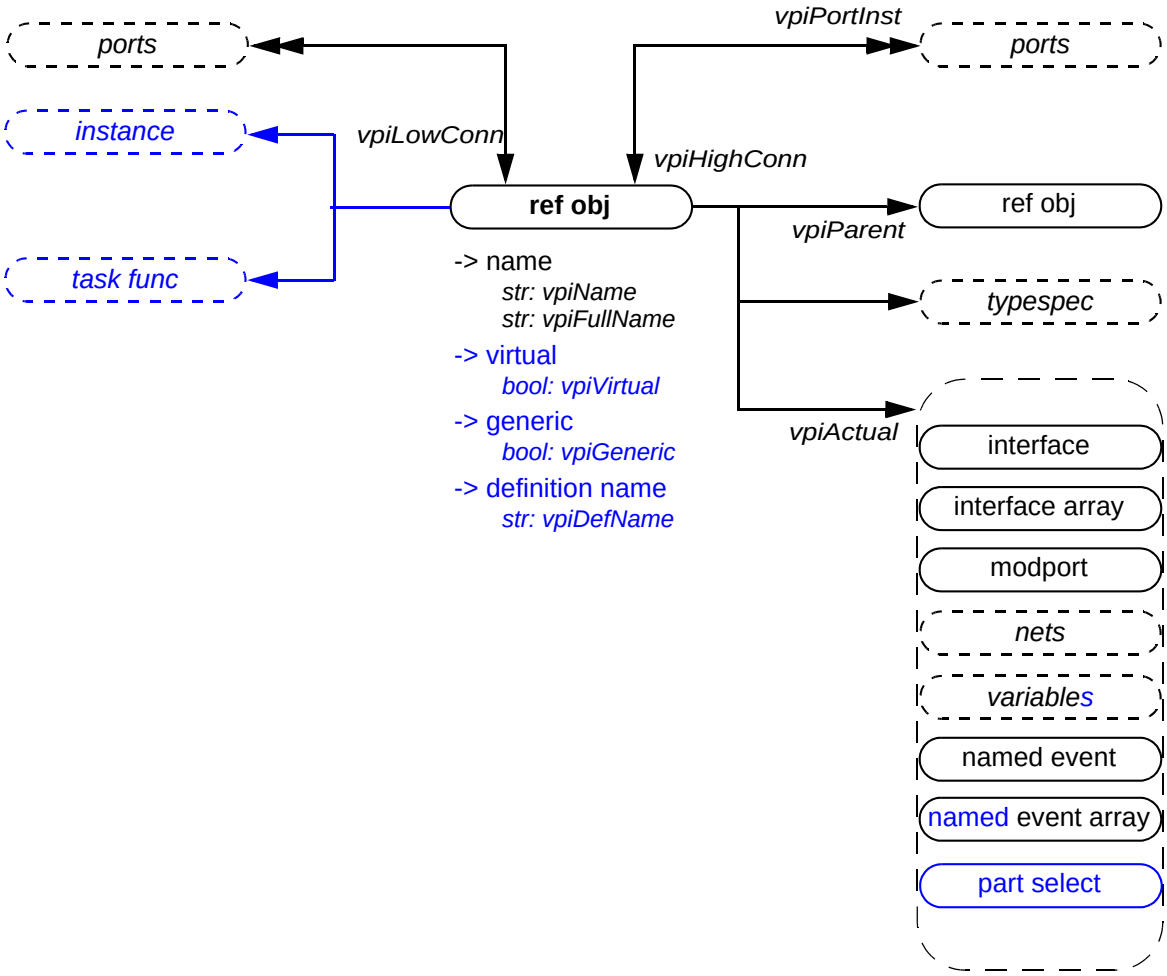
Note this proposal contains fixes for errata 489:

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.

32.12.1Examples

~~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~~ vpi~~RefObjType~~Actual = 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~~ vpi~~fullType~~Actual =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 reurns 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.
*************************************/

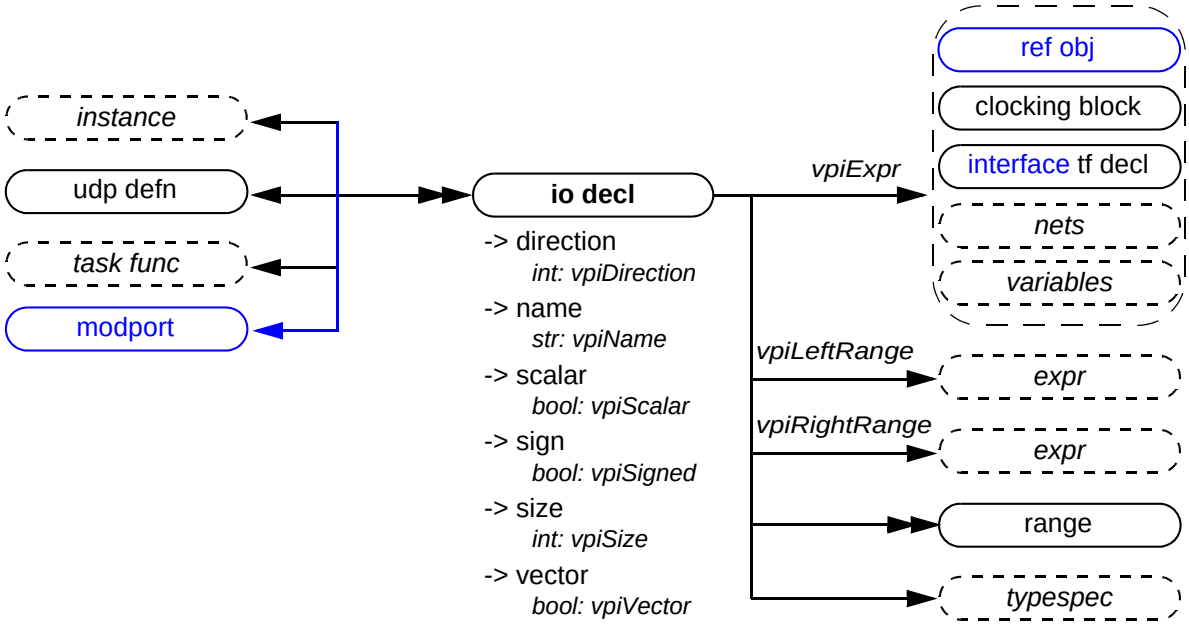
```
Example 2: virtual interface declaration in a class definition

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.

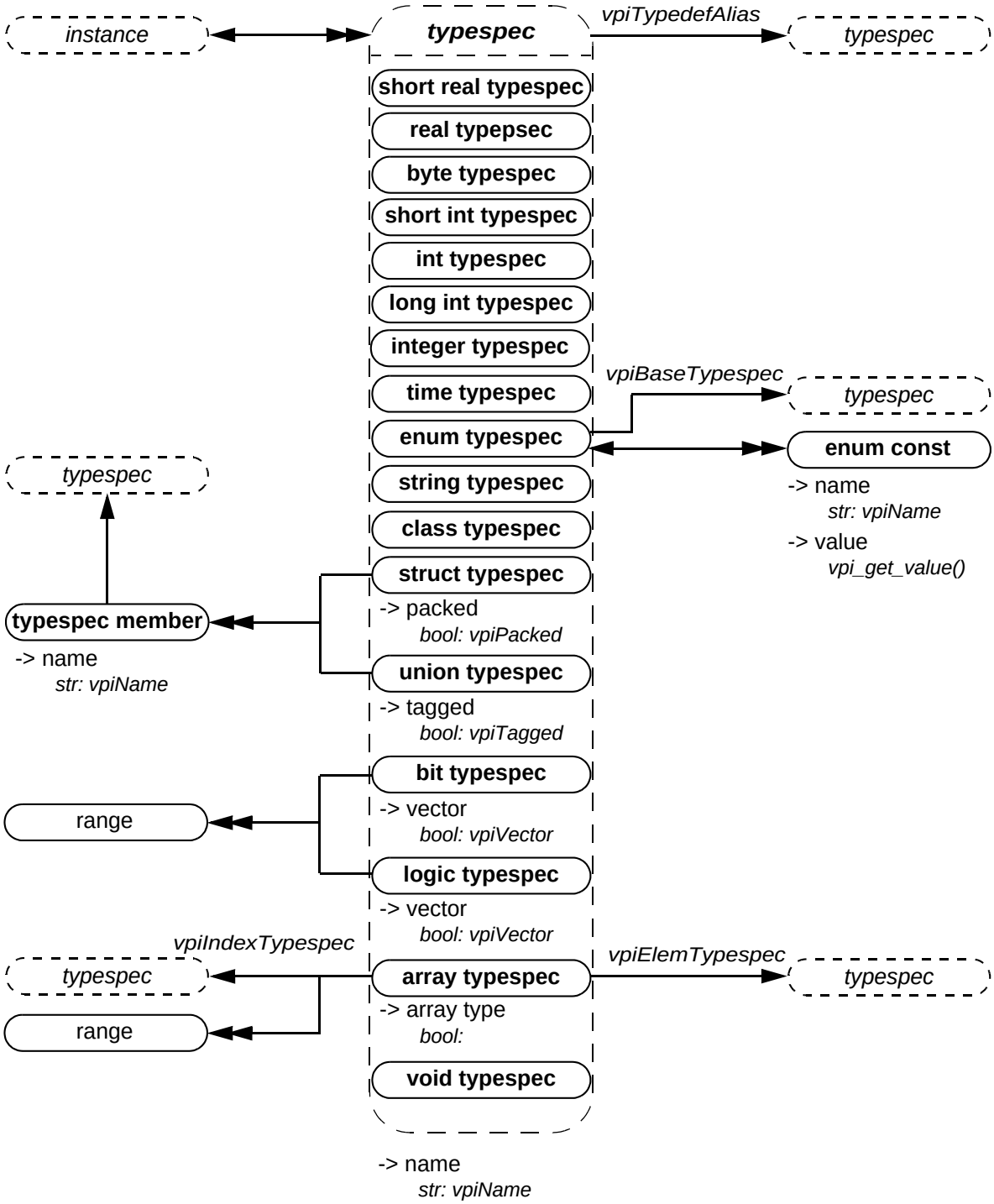
— 2) A **ref obj** type handle may be returned for the **vpiExpr** of an **io decl** if it is passed by reference or if the **io decl** is an interface or a modport.

— 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**".

Section 32.22 Class variables

REPLACE

NOTES

1) ....

2) ...

3) **vpiClassDefn** returns the ClassDefn that 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”.

WITH

1) ....

2) ....

3) ~~**vpiClassDefn** returns the ClassDefn that 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:~~

vpiClassDefn returns the class defn with which the class var was declared in the SystemVerilog source text. **vpiActualDefn** shall return the class defn of the object to which the class var is currently pointing. The **vpiActualDefn** handle could either be the base class (obtained with **vpiClassDefn**), a subclass derived from the base class, or NULL if the class var is uninitialized or assigned to NULL. The difference between **vpiClassDefn** and **vpiActualDefn** can be seen in the following example:

```
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”.

Section **16.12**
Input sampling

In 16.12, REPLACE

Samples happen immediately (the calling process does not block). When a signal appears in an expression, it is replaced by the signal’s sampled value, that is, the value that was sampled at the last sampling point.

WITH

~~Samples happen immediately (the calling process does not block).~~ When a signal appears in an expression, it is replaced by the signal’s sampled value, that is, the value that was sampled at the last sampling point.

Section **16.4, A.6.11, Syntax 16-1 clocking_decl_assign (BNF)**

In 16.4, APPEND the following paragraph to the end of the subclause

In a clocking block any expression assigned to a signal in its declaration shall be an expression that would be legal in a port connection to a port of any of the directions specified in the declaration. For example, it would be illegal to assign an inout signal an expression in its declaration that would be illegal in a port connection to an inout port.

In A.6.11 and Syntax 16-1, REPLACE

clocking_decl_assign ::= signal_identifier [= hierarchical_identifier]

WITH

clocking_decl_assign ::= signal_identifier [= ~~hierarchical_identifier~~ expression]

Section 31.8.7

Change:

A move to next (previous) VC means move to the next (previous) earliest VC among the objects in the collection; any traverse handle that does not have any VC is ignored; on return its new handle points to the same place as its old.

To:

A move to next (previous) VC means move to the next (previous) earliest VC among the handles in the collection~~; any traverse~~. Any handle that does not have a~~ny~~ VC at that time is ~~ignored; on return its new handle points to the same place as its old~~ unchanged.

Section 31.8.6

Change:

Object based traversal can be performed by creating a traverse handle for the object and then moving it back and forth to the next or previous Value Change (VC) or by performing jumps in time. A traverse object handle for any object in the design can be obtained by calling `vpi_handle()` with a `vpiTrvsObj` type, and an object `vpi-Handle`. This is the method described in 31.8.4, and used in all the code examples thus far.

To:

Object based traversal can be performed by creating a traverse handle for the object and then moving it back and forth to the next or previous Value Change (VC) or by performing jumps in time. A traverse object handle for any object in the design can be obtained by calling `vpi_handle()` with a `vpiTrvsObj` type, and ~~an object `vpi-Handle`~~ the object's handle. This is the method described in 31.8.4, and used in all the code examples thus far.

Section **17.2**
program variables

In the last paragraph of 17.2, REPLACE

Program variables can only be assigned using blocking assignments.

WITH

Variables declared within the scope of a program are called program variables. Program variables can only be assigned using blocking assignments.

Section **17.2, A.1.7, Syntax 17-1 specparams outside of specify blocks**

In A.1.6, in non_port_interface_item, DELETE

~~| { attribute_instance } specparam_declaration~~

In A.1.7 and Syntax 17-1, in non_port_program_item, DELETE

~~| { attribute_instance } specparam_declaration~~

In A.1.10 and Syntax 19-1, in package_item, DELETE

~~| specparam_declaration~~

In A.8.4, in constant_primary, REPLACE

| ps_specparam_identifier [constant_range_expression]

WITH

| ~~ps_~~specparam_identifier [constant_range_expression]

AND DELETE the surrounding blank lines.

In A.9.3, DELETE

~~ps_specparam_identifier ::= [package_scope] specparam_identifier~~

Proposal for ballot comment 156, Mantis item 505:

In 22.2, CHANGE:

A module, interface, program or class can have parameters, which are set during elaboration and are constant during simulation. They are defined with data types and default values. With System-Verilog, if no data type is supplied, parameters default to type `logic` of arbitrary size for Verilog compatibility and interoperability.

TO:

A module, interface, program or class can have parameters, which are set during elaboration and are constant during simulation. They are defined with data types and default values. ~~With System-Verilog, if no data type is supplied, parameters default to type `logic` of arbitrary size for Verilog compatibility and interoperability.~~ For compatibility with Verilog, if no data type is supplied, the type is determined when the value is determined.

and CHANGE:

In an assignment to, or override of, a parameter without an explicit type declaration, the type of the right-hand expression shall be unsized, real or integral.

TO:

In an assignment to, or override of, a parameter without an explicit type declaration, the type of the right-hand expression shall be ~~unsized,~~ real or integral. If the expression is real, the parameter is real. If the expression is integral, the parameter is a `logic` vector of the same size with range [size-1:0].

Section **22.1**
Parameter declaration

In 22.1,

Verilog provides three constructs for defining compile time constants: the **parameter**, **localparam** and **specparam** statements.

WITH

Verilog provides three constructs for defining ~~compile~~ elaboration-time constants: the **parameter**, **localparam** and **specparam** ~~statements~~ declarations.

Section **6.9.1, 6.9.2**
Type compatibility

MODIFY Clause 6.9.1 as follows:

1) A simple bit vector type that does not have a predefined width and one that does have a predefined width match if both are 2-state or both are 4-state, both are signed or both are unsigned, both have the same width, and the range of the simple bit vector type without a predefined width is [width-1:0].~~right range bound of the simple bit vector type without a predefined width is zero.~~

```
typedef bit signed [7:0] BYTE; // matches the byte type
typedef bit signed [0:7] ETYB; // does not match the byte type
```

7) ~~Explicitly adding signed or unsigned modifiers to a type that does not change its default signing, does not create a non-matching type.~~

Explicitly adding signed or unsigned modifiers to a type that does not change its default signing, creates a type which matches the type without the explicit signing specification.

MODIFY Clause 6.9.2 as follows:

Two data types shall be defined as equivalent data types using the following inductive definition. If the two data types are not defined equivalent using the following definition, then they shall be defined to be non-equivalent.

1) If two types match, they are equivalent. ~~Any built-in type is equivalent to every other occurrence of itself, in every scope.~~

~~2) A simple typedef or type parameter override that renames a built-in or user-defined type is equivalent to that built-in or user-defined type within the scope of the type identifier.~~

```
typedef bit node; // 'bit' and 'node' are equivalent types
typedef type1 type2; // 'type1' and 'type2' are equivalent types
```

2) ~~3)~~ An anonymous enum, unpacked struct, or unpacked union type is equivalent to itself among data objects declared within the same declaration statement and no other data types.

```
struct {int A; int B;} AB1, AB2; // AB1, AB2 have equivalent types
struct {int A; int B;} AB3; // AB3 is not type equivalent to AB1
```

~~4) A typedef for an enum, unpacked struct, or unpacked union, or a class is equivalent to itself and to data objects that are declared using that data type within the scope of the data type identifier.~~

```
typedef struct {int A; int B;} AB_t;
AB_t AB1; AB_t AB2; // AB1 and AB2 have equivalent types

typedef struct {int A; int B;} otherAB_t;
```

---

otherAB_t AB3; // AB3 is not type equivalent to AB1 or AB2

3) 5) Packed arrays, packed structures, and built-in integral types are equivalent if they contain the same number of total bits, are either all 2-state or all 4-state, and are either all signed or all unsigned. Note that if any bit of a packed structure or union is 4-state, the entire structure or union is considered 4-state.

```
typedef bit signed [7:0] BYTE; // equivalent to the byte type
typedef struct packed signed {bit[3:0] a, b;} uint8;
// equivalent to the byte type
```

4) 6) Unpacked array types are equivalent by having equivalent element types and identical shape. Shape is defined as the number of dimensions and the number of elements in each dimension, not the actual range of the dimension.

```
bit [9:0] A [0:5];
bit [1:10] B [6];
typedef bit [10:1] uint10;
uint10 C [6:1]; // A, B and C have equivalent types
typedef int anint [0:0]; // anint is not type equivalent to int
```

7) Explicitly adding signed or unsigned modifiers to a type that does not change its default signing, does not create a non-equivalent type. Otherwise, the signing must match to have equivalence

typedef bit unsigned ubit; // type equivalent to bit

8) A typedef for an enum, unpacked struct, or unpacked union, or a class type declared in a package is always equivalent to itself, regardless of the scope where the type is imported.

MODIFY Clause 5.2 as follows:

SystemVerilog accepts a single positive number, as an alternative to a range, to specify the size of an unpacked array, like
C. That is, [size] becomes the same as [0:size-1]. For example:

int Array[8][32]; is the same as: int Array[0:7][0:31];

Section **6.9.1**
Scope of a type

In 6.9, at the end of the subclause, ADD the following paragraph

The scope of a data type identifier shall include the hierarchical instance scope. This means that each instance with a user-defined type declared inside the instance creates a unique type. To have type matching or equivalence among multiple instances of the same module, interface or program, a class, enum, unpacked structure or unpacked union type must be declared at a higher level in the compilation unit scope than the declaration of the module, interface or program, or imported from a package. For type matching, this is true even for packed structure and packed union types.

In 6.9.1, REPLACE

8) A typedef for an enum, struct, union, or class type declared in a package always matches itself, regardless of the scope into which the type is imported. The scope of a type identifier includes the hierarchical instance scope. This means that each instance with a user-defined type declared inside the instance creates a unique type. To have type matching among multiple instances of the same module, interface, or program, a type must be declared at higher level in the compilation unit scope than the declaration of the module, interface or program, or imported from a package.

WITH

8) A typedef for an enum, struct, union, or class type declared in a package always matches itself, regardless of the scope into which the type is imported. ~~The scope of a type identifier includes the hierarchical instance scope. This means that each instance with a user-defined type declared inside the instance creates a unique type. To have type matching among multiple instances of the same module, interface, or program, a type must be declared at higher level in the compilation unit scope than the declaration of the module, interface or program, or imported from a package.~~

Section

In 6.9.2, REPLACE

8) A typedef for an enum, unpacked struct, or unpacked union, or a class type declared in a package is always equivalent to itself, regardless of the scope where the type is imported.

The scope of a type identifier includes the hierarchical instance scope. This means that each instance with user-defined types declared inside the instance creates a unique type. To have type equivalence among multiple instances of the same module, interface, or program, a type must be declared at higher level in the compilation unit scope than the declaration of the module, interface or program, or imported from a package.

WITH

8) A typedef for an enum, unpacked struct, or unpacked union, or a class type declared in a package is always equivalent to itself, regardless of the scope where the type is imported.

The scope of a type identifier includes the hierarchical instance scope. This means that each instance with user-defined types declared inside the instance creates a unique type. To have type equivalence among multiple instances of the same module, interface, or program, a type must be declared at higher level in the compilation unit scope than the declaration of the module, interface or program, or imported from a package.

32.26 Task and function declaration (supersedes P1364 26.6.18)

CHANGE the first note in this section from

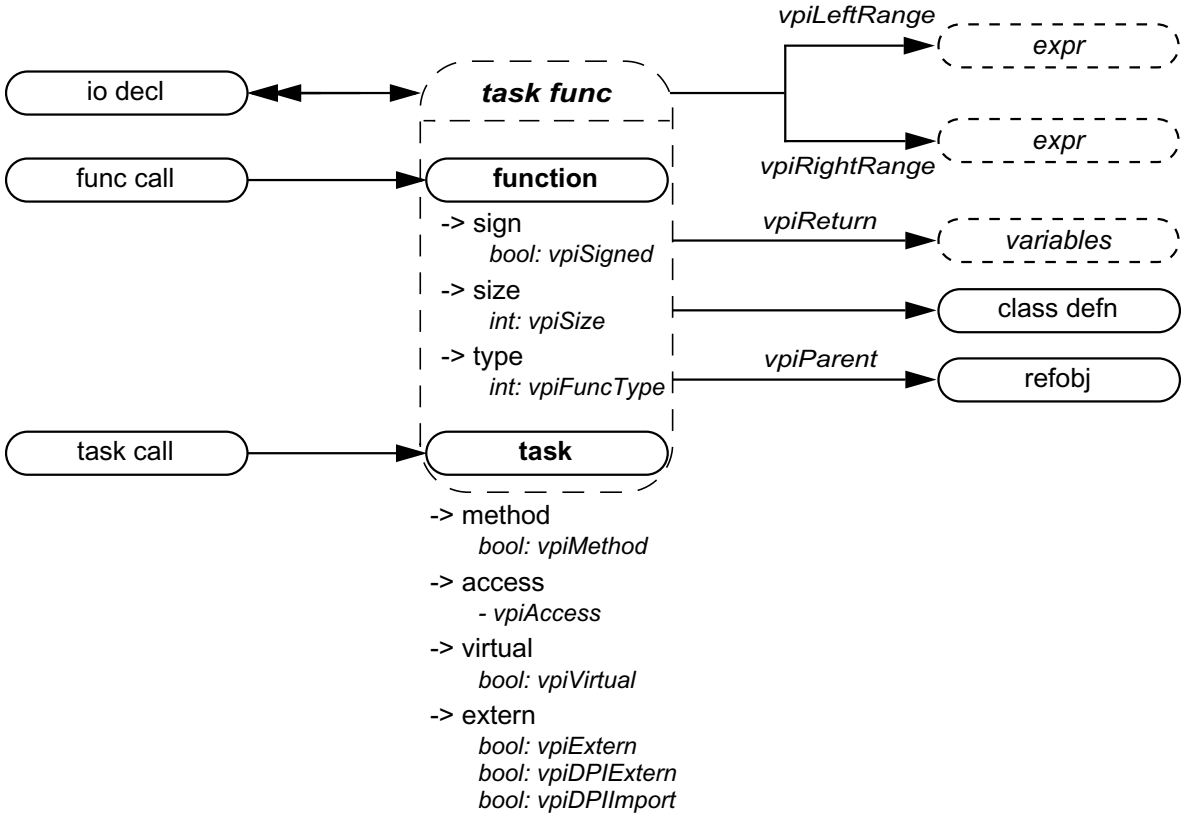
1) A Verilog HDL function shall contain an object with the same name, size, and type as the function.

TO

1) A Verilog HDL function shall contain an object with the same name, size, and type as the function. This object shall be used to capture the return value for this function.

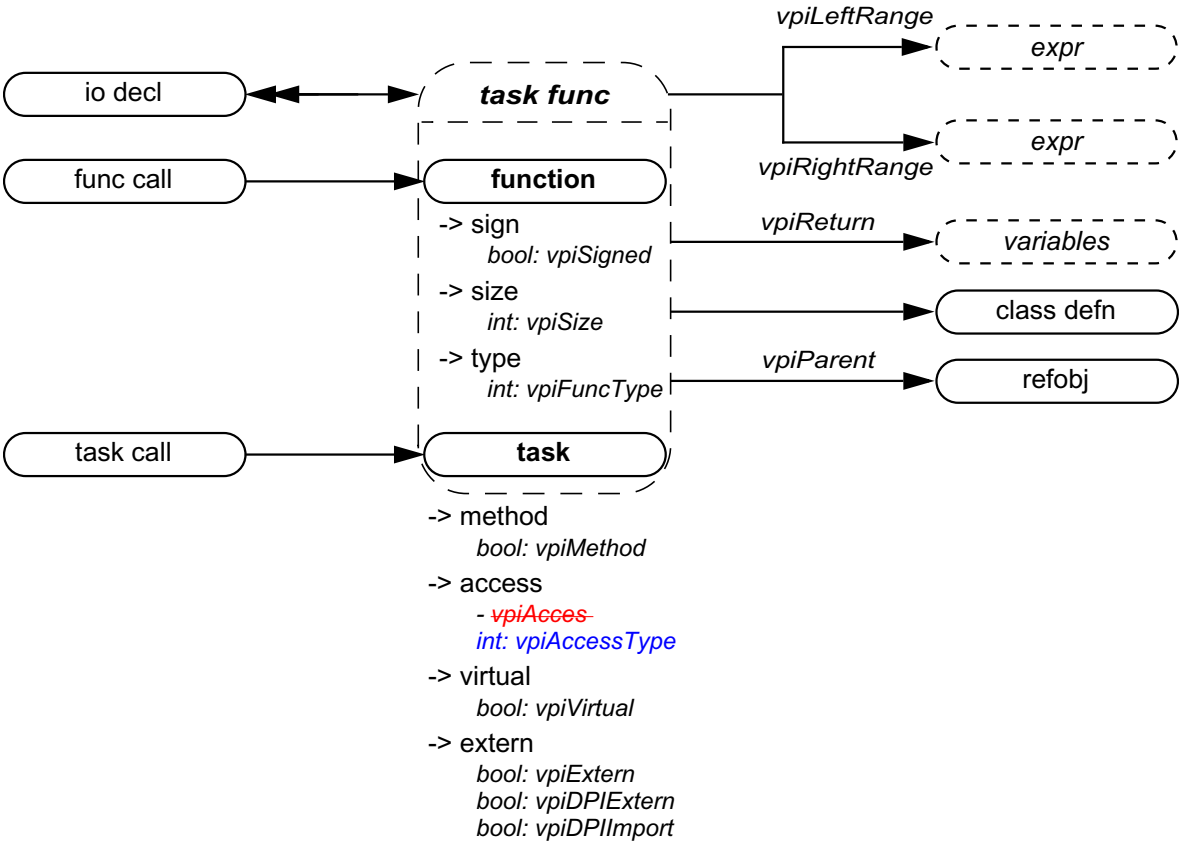
Section 32.26 Task and function declaration (supersedes p1364 26.6.18)

REPLACE



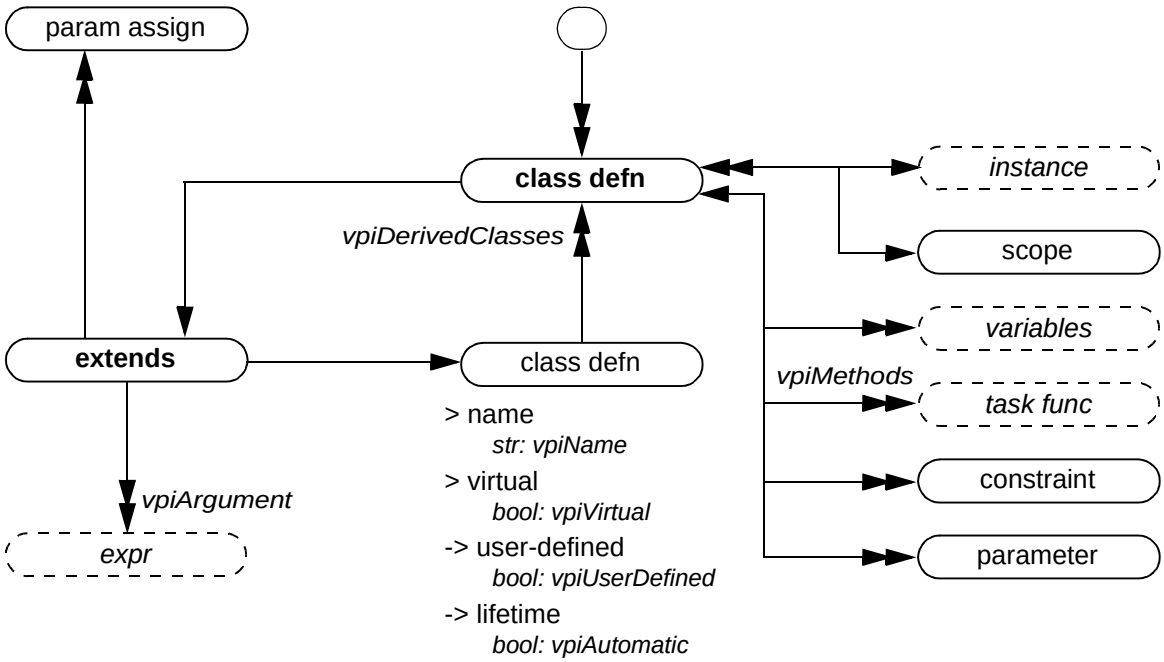
WITH

.



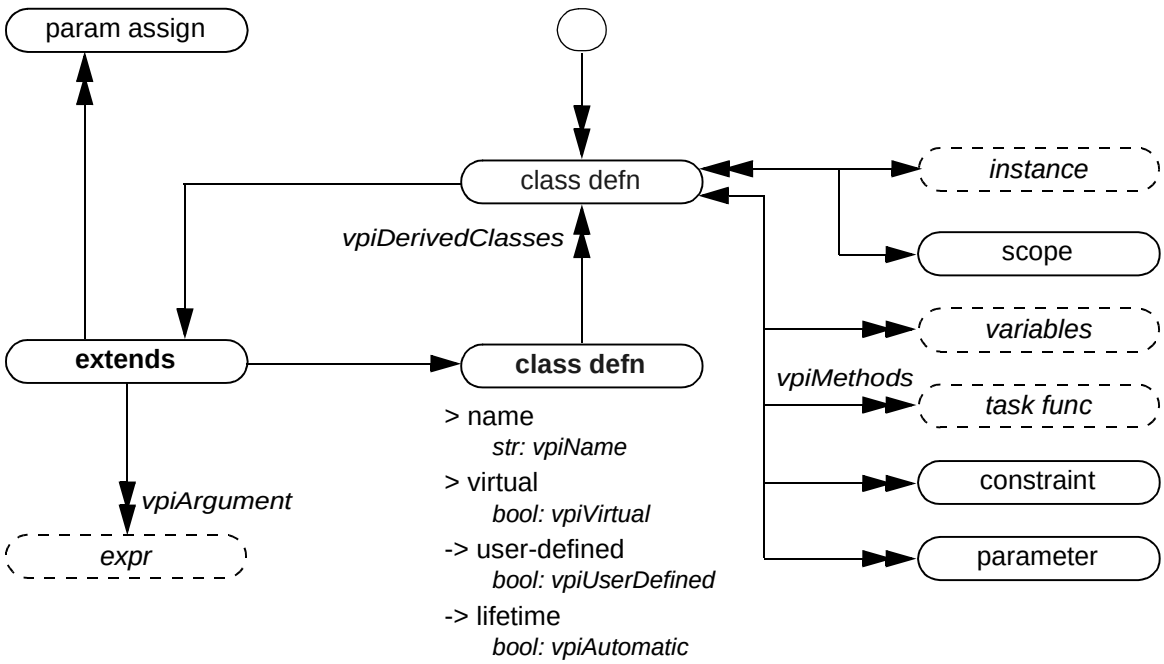
REPLACE:

32.13 Class object definition



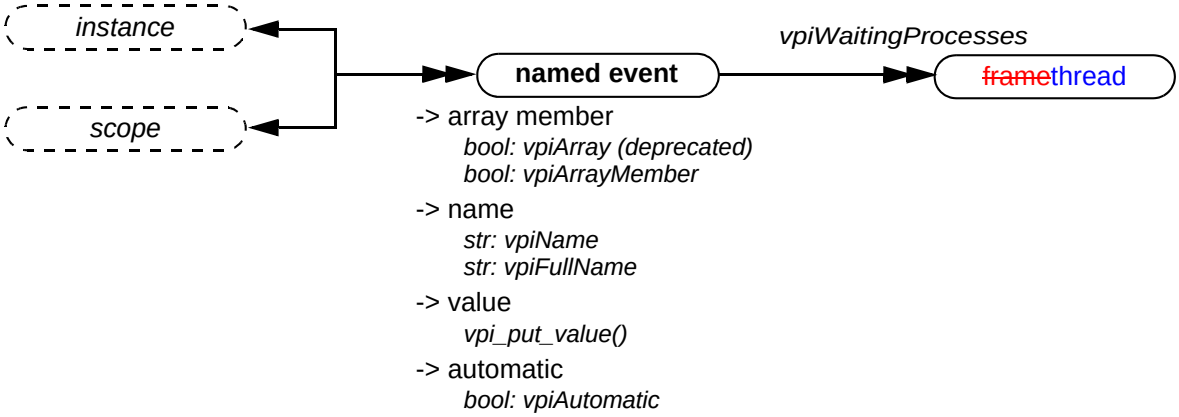
WITH:

32.14 Class object definition



Errata 465

32.19Named events (supersedes P1364 26.6.11)



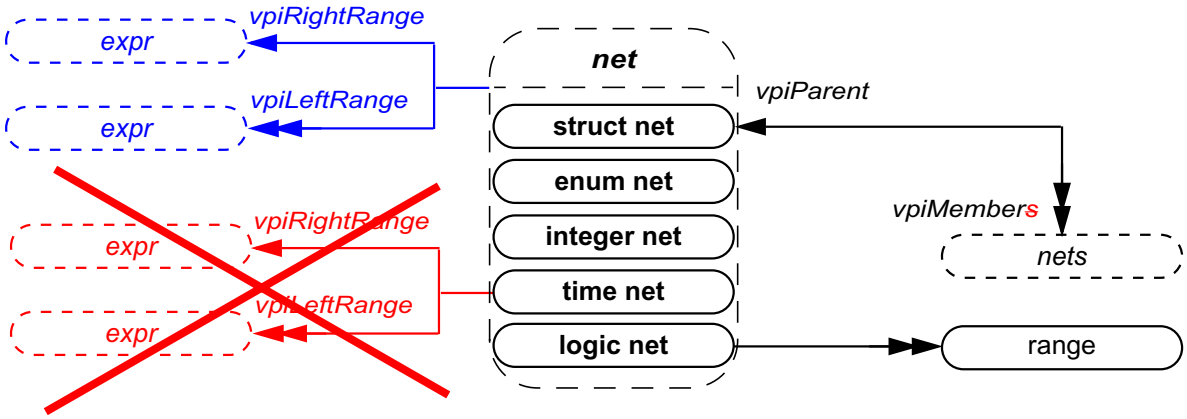
NOTE:

— The **vpiWaitingProcesses** iterator returns all waiting processes, static or dynamic, identified by their ~~frame~~ thread, for that named event.

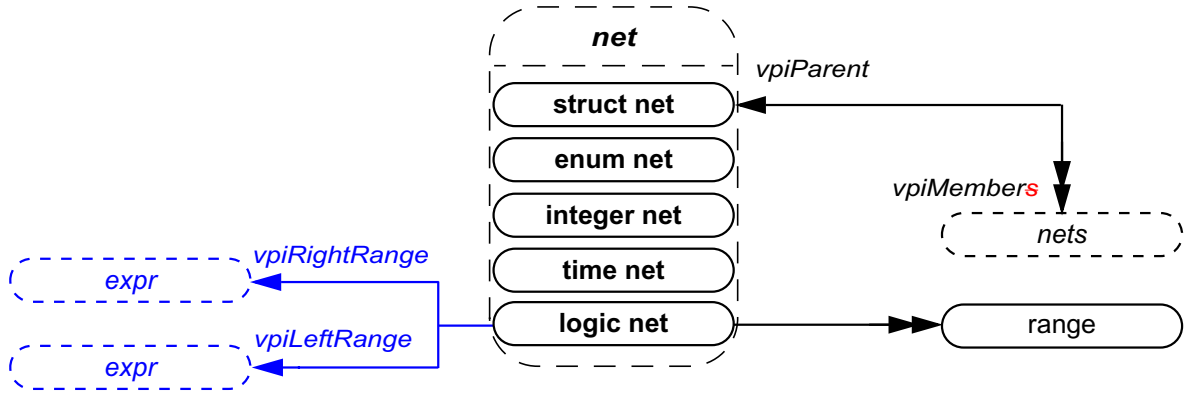
32.13 Nets (supersedes P1364 26.6.6)

NOTE: These changes are with respect to Draft 5:

REPLACE



WITH



32.11 Ports (supersedes P1364 26.6.5)

REPLACE

2) **vpi_get_delays, vpi_put_delays** delays shall not be applicable for **vpiInterfacePort.**

WITH

2) **vpi_get_delays(), vpi_put_delays()** delays shall not be applicable for **vpiInterfacePort.**

REPLACE

The table below describes the possible return values for each of the coverage control options:

Table 30-1: Coverage control return values

	`'SV_COV_OK`	`'SV_COV_ERROR`	`'SV_COV_NOCOV`	`'SV_COV_PARTIAL`
`'SV_COV_START`	success	bad args	no coverage	partial coverage
`'SV_COV_STOP`	success	bad args	-	-
`'SV_COV_RESET`	success	bad args	-	-
`'SV_COV_CHECK`	full coverage	bad args	no coverage	partial coverage

Starting, stopping, or resetting coverage multiple times in succession for the same instance(s) has no further effect if coverage has already been started, stopped, or reset for that/those instance(s).

The hierarchy(ies) being controlled or queried are specified as follows.

`'SV_MODULE_COV, "unique module def name"`
provides coverage of all instances of the given module (the unique module name is a string), excluding any child instances in the instances of the given module. The module definition name can use special notation to describe nested module definitions.

WITH

The table below describes the possible return values for each of the coverage control options:

Table 30-1: Coverage control return values

	`'SV_COV_OK`	`'SV_COV_ERROR`	`'SV_COV_NOCOV`	`'SV_COV_PARTIAL`
`'SV_COV_START`	success	bad args	no coverage	partial coverage
`'SV_COV_STOP`	success	bad args	-	-
`'SV_COV_RESET`	success	bad args	-	-
`'SV_COV_CHECK`	full coverage	bad args	no coverage	partial coverage

~~Starting, stopping, or resetting coverage multiple times in succession for the same instance(s) has no further effect if coverage has already been started, stopped, or reset for that/those instance(s).~~

Starting coverage on an instance that has already had coverage started via a prior call to $coverage_control() shall have no effect. Similarly, repeated calls to stop or reset coverage shall have no effect.

The hierarchy(ies) being controlled or queried are specified as follows.

`'SV_MODULE_COV, "unique module def name"`
provides coverage of all instances of the given module (the unique module name is a string), excluding any child instances in the instances of the given module. The module definition name can use special notation to describe nested module definitions.

Section 30.1.3

Change:

The granularity of statement coverage can be per-statement or per-statement block (however defined).

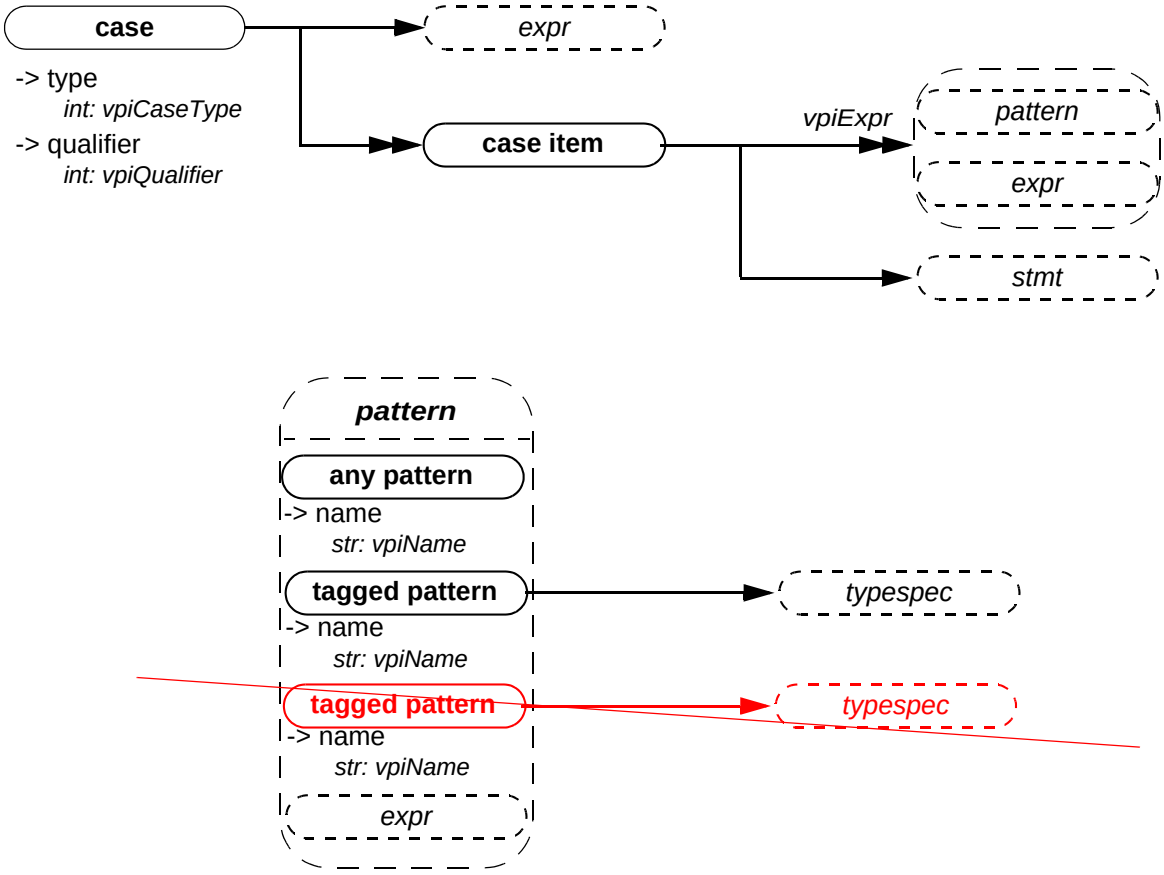
To:

The granularity of statement coverage can be per-statement or per-statement block ~~(however defined)~~ depending on the query, see 30.4.3 for details.

Errata 547

32.47Case, pattern (supersedes P1364 26.6.36)

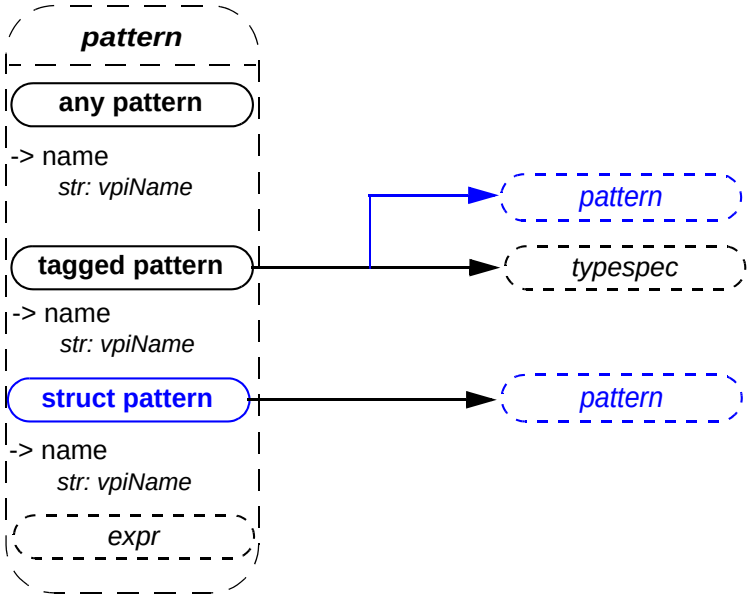
Delete the second tagged pattern and second typespec



Add structPattern and a arrow to pattern.

Add an arrow to pattern from a tagged pattern.

The new diagram should look like:



Section **A.1.8, Syntax 12-1**
Class items

A class declaration can declare a virtual interface or import from a package.

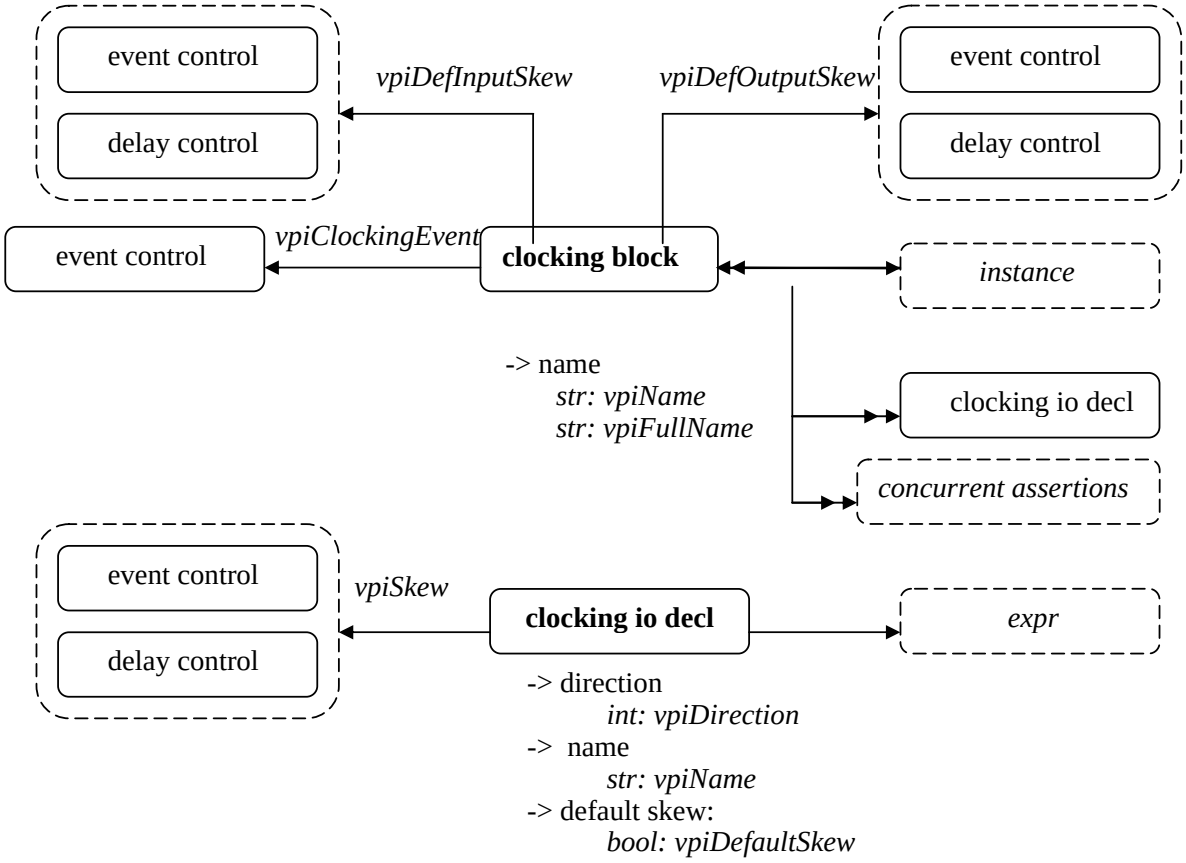
In A.1.8 and Syntax 12-1, in class_item, DELETE

~~| { attribute_instance } type_declaration~~

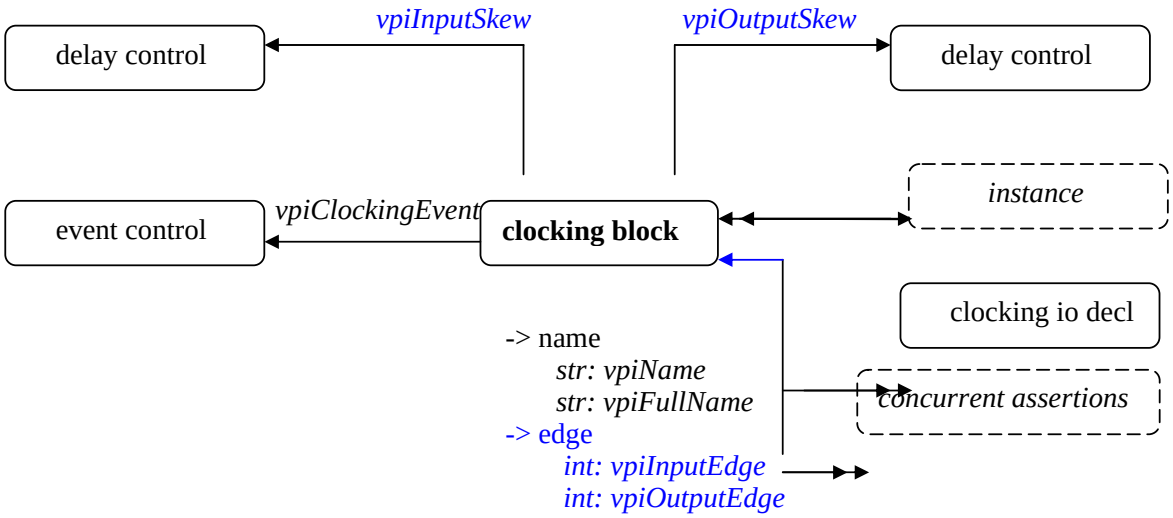
BECAUSE it is redundant.

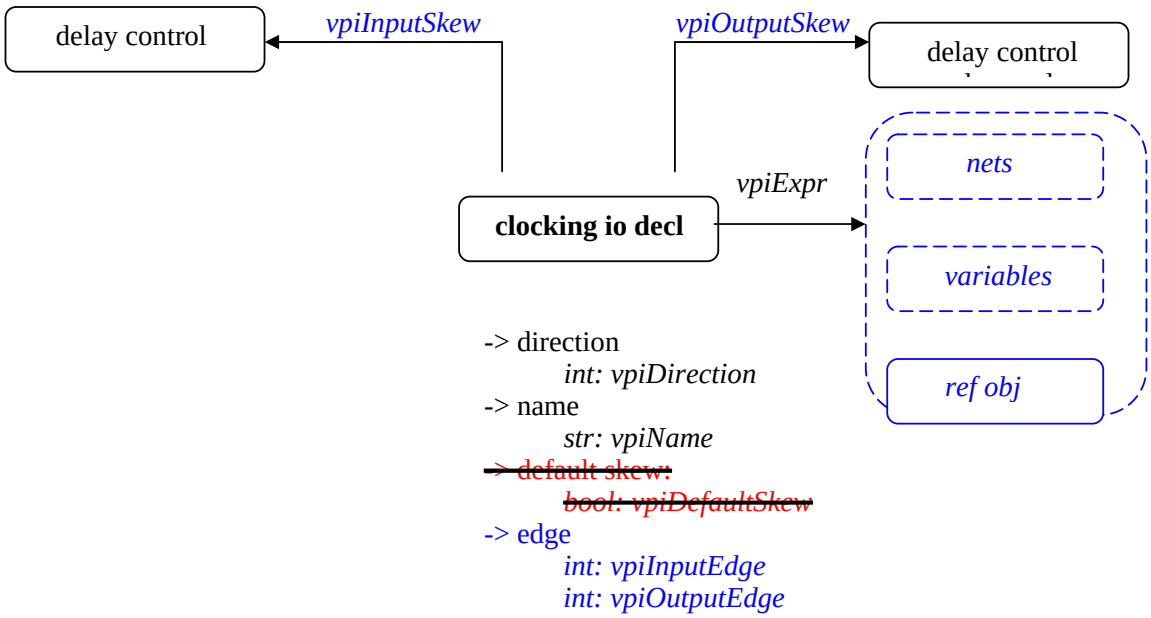
32.30 Clocking block

REPLACE



WITH
-



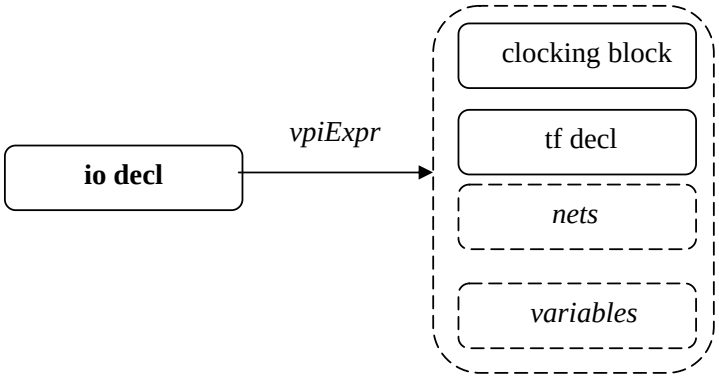


NOTES:

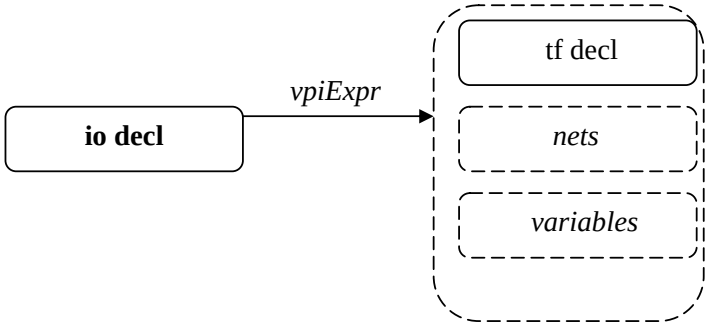
1) The methods, **vpiInputSkew** and **vpiOutputSkew** and properties, **vpiInputEdge** and **vpiOutputEdge** on the clocking block apply to the default constructs. The same methods and properties on the clocking io decl apply to the clocking io decl itself.
2) vpiExpr shall return the expression or ref obj referenced by the clocking io decl. Consider input enable = top.mem1.enable. Here “enable” is represented by a clocking io decl and vpiExpr relation returns a handle to “top.mem1.enable”.

32.10 IO declaration (supersedes P1364 26.6.4)

REPLACE the unnamed object return by vpiExpr relation



WITH



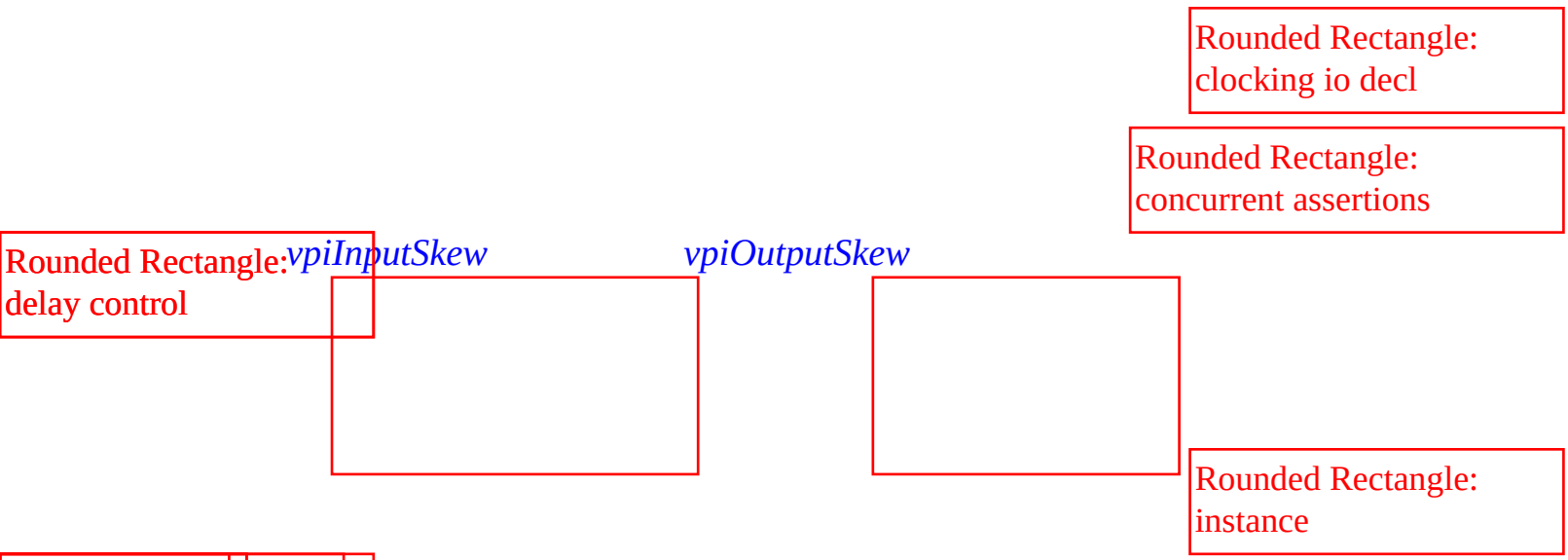
Annex I

```
#define vpiIndexTypespec                702
#define vpiBaseTypespec                 703
#define vpiElemTypespec                 704

#define vpiDefInputSkew                 706
#define vpiDefOutputSkew                707
#define vpiClockingSkew                 708
……….
……….
#define vpiPacked                       630
#define vpiTagged                       632
#define vpiRef                          633
#define vpiDefaultSkew                  634
#define vpiVirtual                      635
……….
……….

#define vpiQualifier                    650
#define vpiNoQualifier          0
#define vpiUniqueQualifier              1
#define vpiPriorityQualifier            2
#define vpiTaggedQualifier              3
#define vpiRandQualifier                4

#define vpiInputEdge                    651 /* returns vpiNoEdge, vpiPosedge or vpiNegedge */
#define vpiOutputEdge           652 /* returns vpiNoEdge, vpiPosedge or vpiNegedge */
```

vpiSkew

-> direction
int: vpiDirection
-> name
str: vpiName
-> default skew:
bool: vpiDefaultSkew

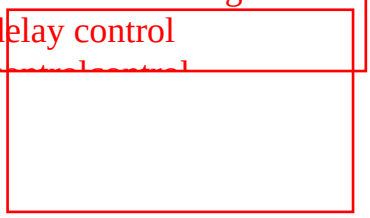
WITH



-> name
str: vpiName
str: vpiFullName
-> edge
int: vpiInputEdge
int: vpiOutputEdge

Rounded Rectangle: delay control

vpiInputSkew *vpiOutputSkew*



Rounded Rectangle: clocking io decl

vpiExpr

Rounded Rectangle: variables

Rounded Rectangle: ref obj

-> direction
int: vpiDirection
-> name
str: vpiName
-> default skew:
bool: vpiDefaultSkew
-> edge
int: vpiInputEdge
int: vpiOutputEdge

NOTES:

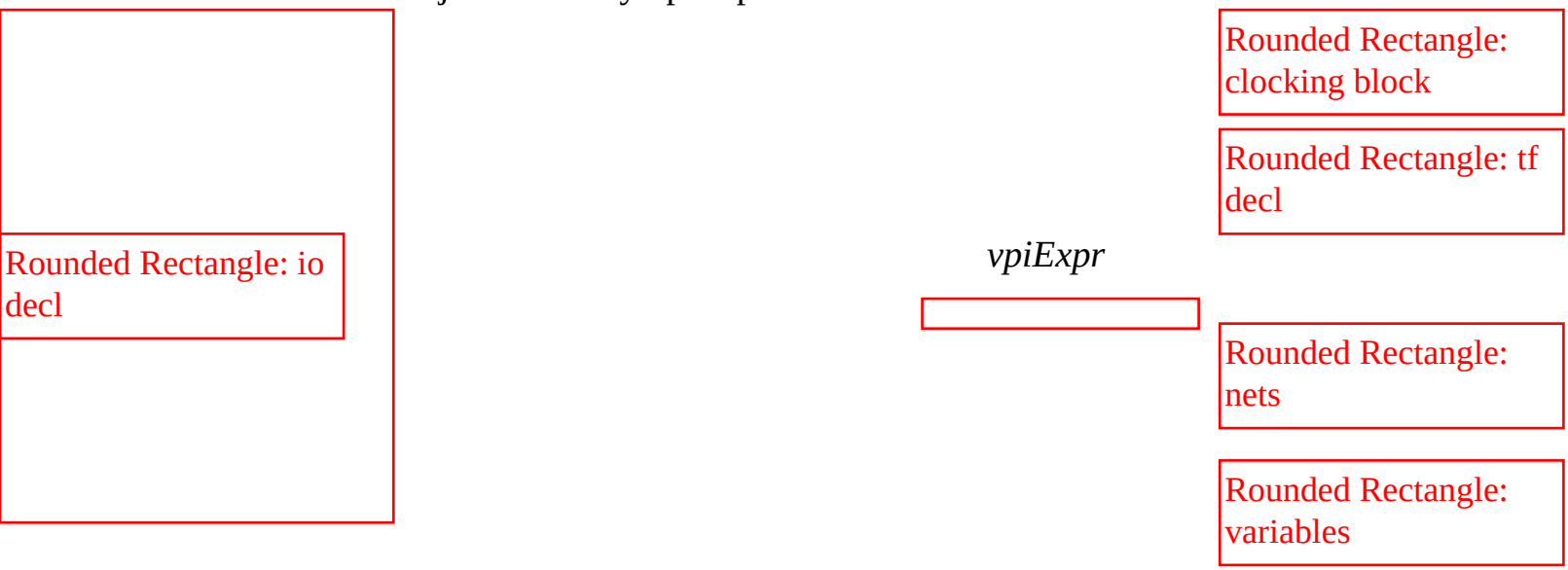
1) The methods, **vpiInputSkew** and **vpiOutputSkew** and properties, **vpiInputEdge** and **vpiOutputEdge** on the clocking block apply to the default constructs. The same methods and properties on the clocking io decl apply to the clocking io decl itself.

2) vpiExpr shall return the expression or ref obj referenced by the clocking io decl. Consider input enable = top.mem1.enable. Here “enable” is represented by a clocking io decl and vpiExpr relation returns a handle to “top.mem1.enable”.

IEEE P1800/D4, February 16,2005

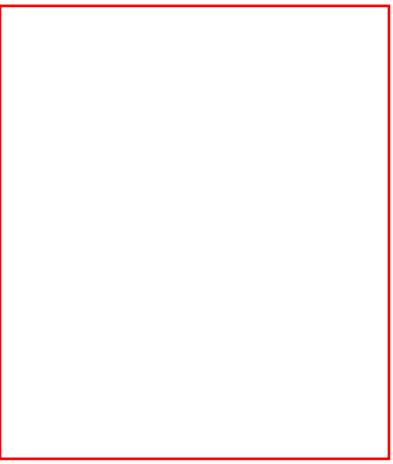
32.10 IO declaration (supersedes P1364 26.6.4)

REPLACE the unnamed object return by vpiExpr relation



WITH

Rounded Rectangle: variables



vpiExpr

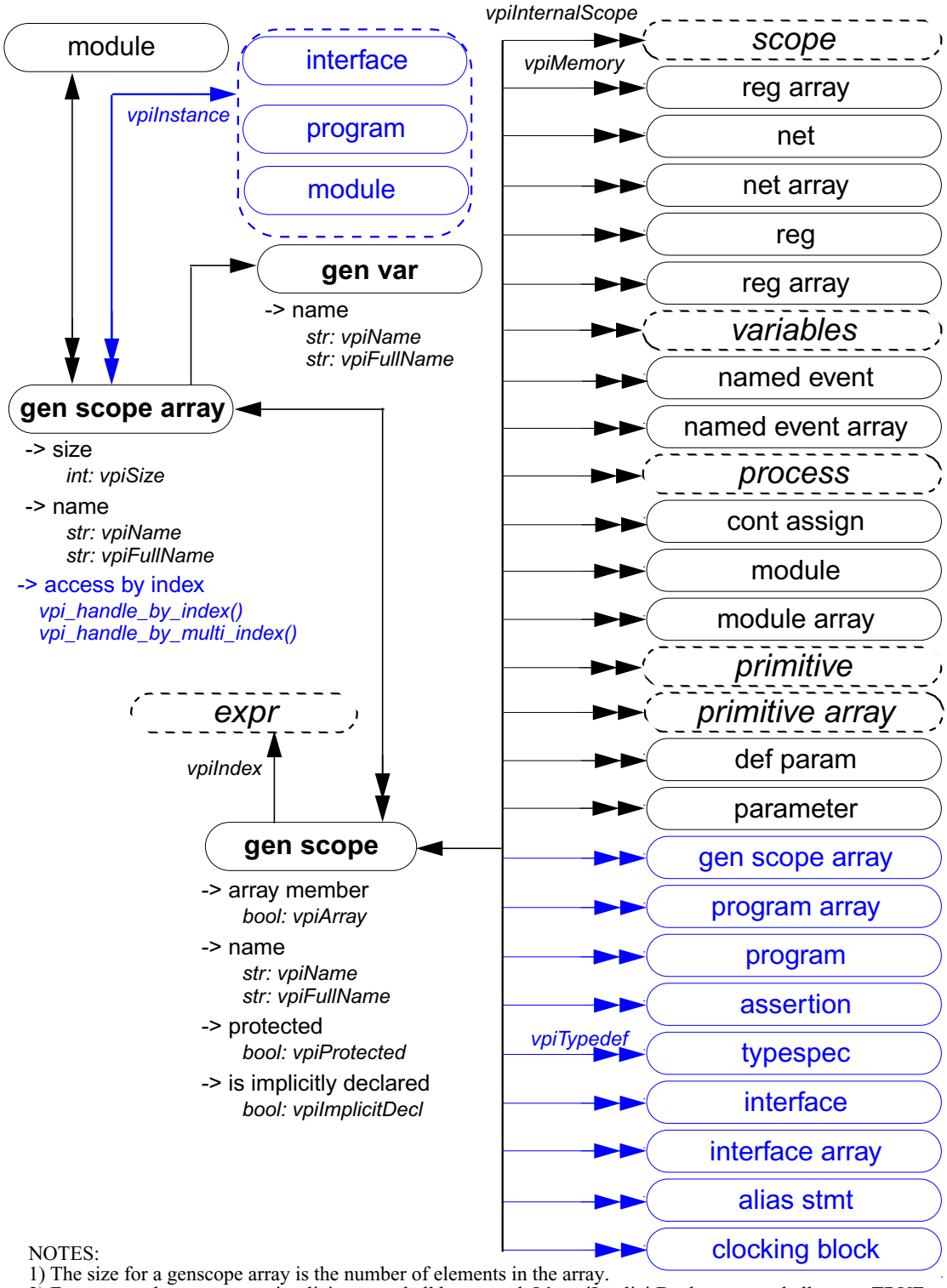
Annex I

```
#define vpiIndexTypespec          702
#define vpiBaseTypespec           703
#define vpiElemTypespec           704

#define vpiDefInputSkew           706
#define vpiDefOutputSkew          707
#define vpiClockingSkew           708
……….
……….
#define vpiPacked                 630
#define vpiTagged                 632
#define vpiRef                    633
#define vpiDefaultSkew            634
#define vpiVirtual                635
……….
……….

#define vpiQualifier              650
#define vpiNoQualifier            0
#define vpiUniqueQualifier        1
```

```
#define vpiPriorityQualifier        2
#define vpiTaggedQualifier          3
#define vpiRandQualifier            4

#define vpiInputEdge                651 /* returns vpiNoEdge, vpiPosedge or vpiNegedge */
#define vpiOutputEdge               652 /* returns vpiNoEdge, vpiPosedge or vpiNegedge */
```

Add the following section after 32.53:

32.54 Generates (supersedes P1364 26.6.44)



NOTES:
1) The size for a genscope array is the number of elements in the array.
2) For unnamed generates, an implicit scope shall be created. It's vpiImplicitDecl property shall return TRUE.
3) References to genvars within the genscope shall be treated as local parameters
4) Parameters within the genscope must be local parameters

18.7.3, Page 224 (Added property keyword and a parenthesis)

Change

```
assert (done |=> (out == $past(q, 2,enable)) ;
```

TO

```
assert property (done |=> (out == $past(q, 2,enable)) );
```

[sv-bc] Fwd: AW: Ballot feedback feedback

From: Karen Pieper <Karen.Pieper_at_.....>
Date: Fri Apr 22 2005 - 10:04:54 PDT

Feedback from the entity describing the bind issue:

>One example is shown attached. Here, a type
>is declared in a module (here i_t3). In this module, a new type,
call "new_type"
>is declared.
>
>Another module (i_x_t3) is bound to that module and uses the
declared type
>"new_type". An applicaiton example would be to sample that value for
a later
>check with an assertion.
>
>As far as I remember, SystemVerilog requires, that a type must be
declared before
>its use.
>
>Now to the example: Assume, the bound module i_x_t3 is handeled like
a
>nested module at the end of module i_t3, then the new_type is
declared
>before, and is known to i_x_t3.
>
>Assume however, that the i_x_t3 is handeled like a nested module
inserted at
>the beginning of the module, then type new_type is not declared at
that
>place. An error should be reported in that case.
>
>To make this clear, I recommend, to define in the LRM, at which
place a bound
>module is inserted, and how the compile order of the bind statement
influences
>the place, the bound modules (in case there are more than one) are
inserted.

```
>
>
>
>//=============================================================
>
>bind i_t3 i_x_t3 i_x_t3_inst(tmp);
>
>//=============================================================
>
>module i_t3 (output logic y, input bit a);
>
>typedef new_type logic[1];
>
>new_type tmp;
>
><mailto:always@(y>always@(y)
>
>tmp[0] = a;
>
>y<=(!tmp[0]);
>
>end
>
>endmodule
>
>//=============================================================
>
>module i_x_t3 (input new_type x);
>
>new_type y;
>
><mailto:always@(x>always@(x)
>
>begin
>
>  y = x;
>
>   assert y = !x;
>
>end
>
>endmodule
>
```

>
>
>

My letter requesting feedback:

> In the SV-BC, we are discussing one of your items:
>"bind" statement needs clarification according to the place the bound module, interface, program is bound to. As an example: There is a program p with a type pt. Another program jap is bound to that program p, and uses the declared type pt. This is only valid, if the type declaration is analysed first, i.e. the jap program is "inserted" at the end of program p. Similarly, it must be specified, how a set of bind statements is analysed.
>In the SV-BC, we believe that the ordering of bind statements is immaterial to understanding a design. Can you provide an example that demonstrates that this isn't so?
>
>The committee has added language that defines that the bind statements are analyzed at the end of the module. The proposal can be found as added blue text in the attached html file on:
>
>http://www.eda.org/svdb/bug_view_page.php?bug_id=0000627
>The file: http://www.eda.org/svdb/file_download.php?file_id=789&type=bug
>
>We'd appreciate your insights if you see additional issues.
>
>Thanks,
>
>Karen

Received on Fri Apr 22 10:05:11 2005

Clause 18.15, the *bind* statement

PROPOSAL

In clause18.15, ADD the following blue text in the indicated position:

In this example, interface range is instantiated in the module cr_unit. Effectively, every instance of module cr_unit shall contain the interface instance r1.

The *bind_instantiation* portion of the bind statement allows the complete range of SystemVerilog instantiation syntax. This means that both parameter and port associations may appear in the *bind_instantiation*. All actual ports and parameters in the *bind_instantiation* refer to objects from the viewpoint of the *bind_target* instance.

When an instance is bound into a target scope, the effect will be as if the instance was present at the very end of the target scope. That means all declarations present in the target scope are visible to the bound instance.

If multiple **bind** statements are present in a given scope, the order of those statements is not important. An implementation is free to elaborate **bind** statements in any order it chooses.

The following is an example of a module containing a **bind** statement with complex instantiation syntax. All identifiers in the **bind** instantiation are referenced from the *bind_target*'s point of view in the overall design hierarchy.

Section **16.2-3**
clocking blocks

In 16.2, REPLACE

Bidirectional signals (**`inout`**) are sampled as well as driven.

WITH

Bidirectional signals (**`inout`**) are sampled as well as driven. An **`output`** signal cannot be read, and an **`input`** signal cannot be driven.

In 16.3, REPLACE

Inputs with explicit #0 skew are sampled at the same time as their corresponding clocking event, but to avoid races, they are sampled in the Observed region. Likewise, clocking block outputs with no skew (or explicit #0 skew) are driven at the same time as their specified clocking event, as nonblocking assignments (in the NBA region).

WITH

Inputs with explicit #0 skew ~~are~~ shall be sampled at the same time as their corresponding clocking event, but to avoid races, they are sampled in the Observed region. Likewise, clocking block outputs with no skew (or explicit #0 skew) ~~are~~ shall be driven at the same time as their specified clocking event, as nonblocking assignments (in the NBA region).

Annex I
sv_vpi_user.h

Change:

```
#define vpiDefInputSkew 706
#define vpiDefOutputSkew 707
#define vpiClockingSkew 708

#define vpiActualDefn 710
```

To:

```
#define vpiDefInputSkew 706
#define vpiDefOutputSkew 707
#define vpiClockingSkew 708
#define vpiDefaultClocking 709

#define vpiActualDefn 710
```

Section 28.4.3 Context tasks and functions (paragraph 2)

Replace

All DPI exported tasks or functions require that the context of their call is known. This occurs since SystemVerilog task or function declarations always occur in instantiable scopes, hence allowing a multiplicity of unique task or function instances.

WITH

~~All DPI exported tasks or functions require that the context of their call is known. This occurs since SystemVerilog task or function declarations always occur in instantiable scopes, hence allowing a multiplicity of unique task or function instances~~. . The SystemVerilog context of DPI export tasks and functions must be known when they are called, including when they are called by imports. When an import invokes the svSetScope utility prior to calling the export, it sets the context explicitly. Otherwise, the context will be the context of the instantiated scope where the import declaration is located. Since imports with diverse instantiated scopes can export the same task or function, multiple instances of such an export can exist after elaboration. Prior to any invocations of svSetScope, these export instances would have different contexts, which would reflect their imported caller's instantiated scope.

Section **E.5.5 Context and non-context tasks and functions (par.3)**

Replace

All DPI export tasks and functions require that the context of their call is known. This occurs since SystemVerilog task and function declarations always occur in instantiable scopes, hence allowing a multiplicity of unique task and function instances in the simulator's elaborated database. Thus, there is no such thing as a non-context export task or function.

WITH

~~All DPI export tasks and functions require that the context of their call is known. This occurs since SystemVerilog task and function declarations always occur in instantiable scopes, hence allowing a multiplicity of unique task and function instances in the simulator's elaborated database. Thus, there is no such thing as a non-context export task or function~~. The SystemVerilog context of DPI export tasks and functions must be known when they are called, including when they are called by imports. When an import invokes the svSetScope utility prior to calling the export, it sets the context explicitly. Otherwise, the context will be the context of the instantiated scope where the import declaration is located. Since imports with diverse instantiated scopes can export the same task or function, multiple instances of such an export can exist after elaboration. Prior to any invocations of svSetScope, these export instances would have different contexts, which would reflect their imported caller's instantiated scope.

Section 28.4.3 Context tasks and functions (paragraph 2)

Section

Replace

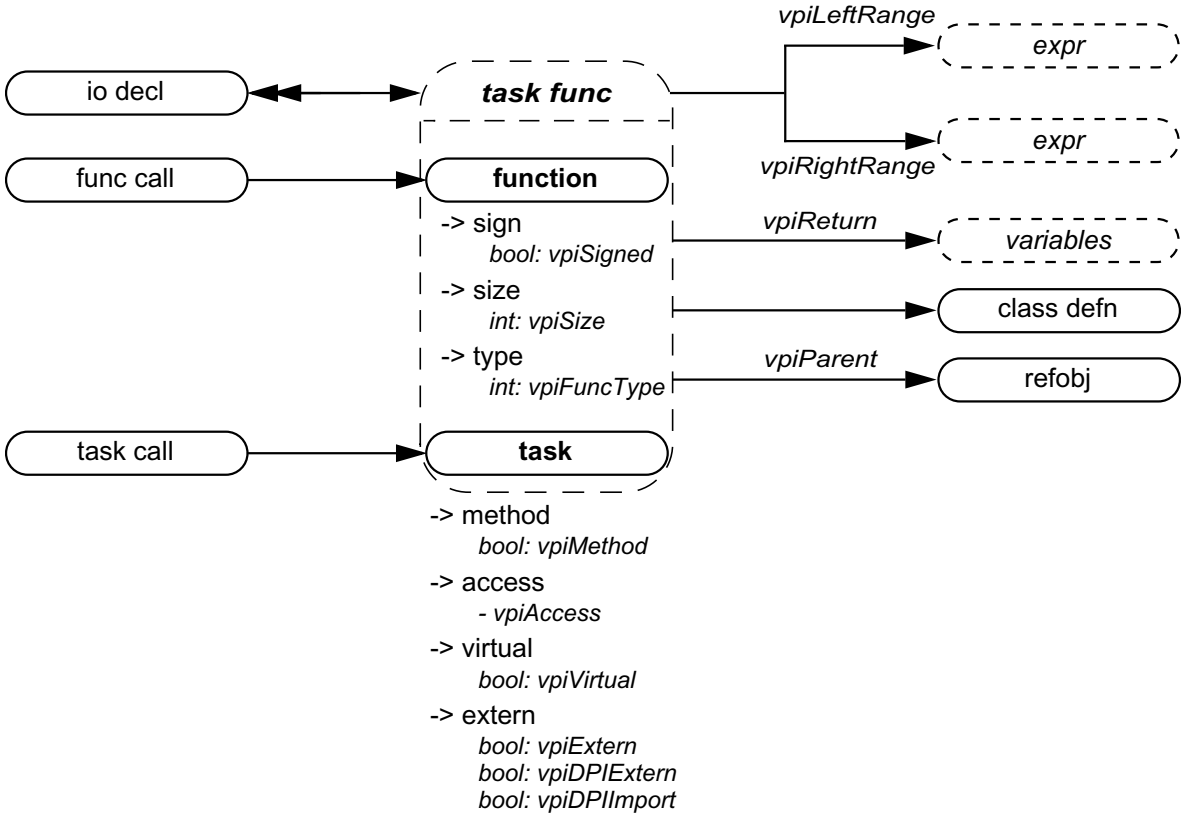
Note that all DPI export tasks and functions require that the context of their call is known. This occurs since SystemVerilog task and function declarations always occur in instantiable scopes, hence giving rise to a multiplicity of associated task or function instances in the simulator’s database. Thus, there is no such thing as a non-context export tasks or function. All export task and function calls must have their execution scope specified in advance by use of a context-setting API function.

WITH

Note that all DPI export tasks and functions require that the context of their call is known. This occurs since SystemVerilog task and function declarations always occur in instantiable scopes, hence giving rise to a multiplicity of associated task or function instances in the simulator’s database. Thus, there is no such thing as a non-context export tasks or function. All export task and function calls must have their execution scope specified in advance by use of a context-setting API function.. The SystemVerilog context of DPI export tasks and functions must be known when they are called, including when they are called by imports. When an import invokes the svSetScope utility prior to calling the export, it sets the context explicitly. Otherwise, the context will be the context of the instantiated scope where the import declaration is located. Since imports with diverse instantiated scopes can export the same task or function, multiple instances of such an export can exist after elaboration. Prior to any invocations of svSetScope, these export instances would have different contexts, which would reflect their imported caller’s instantiated scope.

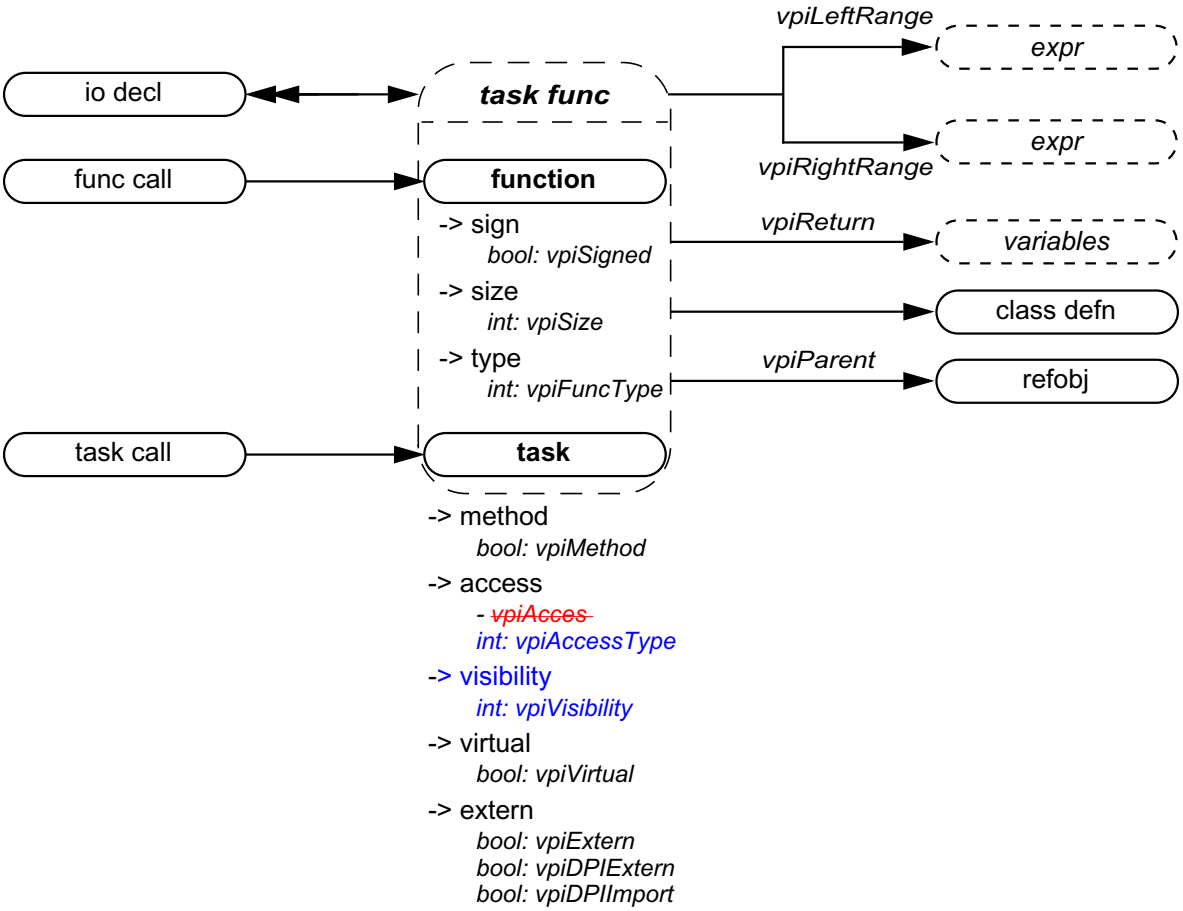
Section 32.26 Task and function declaration (supersedes p1364 26.6.18)

REPLACE



WITH

.This one takes care of Bug ID 478 and 602.



Section **4.4**
Width casts

In 4.4, REPLACE

When changing the size, the signing passes through unchanged. When changing the signing, the size passes through unchanged.

WITH

When changing the size, the signing shall pass through unchanged and the result type shall be a one-dimensional packed array with a right bound of zero. When changing the signing, the type of the expression to be cast shall pass through unchanged, except for the signing.

Section **9.4, A.6.7, Syntax 9-3**
Wild case

At the end of 9.4, APPEND the following paragraph

The keyword **inside** can be used after the parenthesized expression to indicate a set membership (see 8.19) **case**... **inside** statement. In a **case**...**inside** statement, the case expression shall be compared with each case item expression (*open_range_list*) using the set membership **inside** operator. The **inside** operator uses asymmetric wild card matching (see 8.4), and accordingly, the case expression shall be the left operand, and each case item expression shall be the right operand. The case expression given in parentheses and each case item expression in braces shall be evaluated in the order specified by a normal **case**, **unique case**, or **priority case** statement. A case item shall be matched when the **inside** operation compares the case expression to the case item expression and returns 1'b1, and no match when the operation returns 1'b0 or 1'bx. If all comparisons do not match and the default item is given, the default item statement shall be executed

For example:

```
logic [2:0] status;
always @(posedge clock)
        priority case (status) inside
        1, 3 : task1; // matches b001 and b011
        3'b0?0, [4:7]: task2; // matches b000 b010 b0x0 b0z0 b100 b101 b110 b111
        endcase // priority case fails all other values including b00x b01x bxxx
```

In A.6.7 and Syntax 9-3, REPLACE

```
            case_statement ::=
        [ unique_priority ] case_keyword ( expression ) case_item { case_item } endcase
|           [ unique_priority ] case_keyword ( expression ) matches case_pattern_item { case_pattern_item } endcase
```

WITH

```
            case_statement ::=
                [ unique_priority ] case_keyword ( expression )  case_item { case_item } endcase
              |             [ unique_priority ] case_keyword ( expression ) matches case_pattern_item
{ case_pattern_item } endcase
              |             [ unique_priority ] case ( expression ) inside case_inside_item { case_inside_item }
endcase
```

In A.6.7 and Syntax 9-3, ADD

```
            case_inside_item ::=
                        open_range_list : statement_or_null
            |           default [ : ] statement_or_null
```

Section

Section **3.7, 3.8, 8.13, 8.13.1, 8.13.2, 32.39, A.6.8, A.8.5, Index Assignment patterns as lvalues**

In 8.13, REPLACE

An assignment pattern is built from braces, keys, and expressions and is prefixed with an apostrophe. For example:

```
var int A[N] = '{default:1};
var integer i = '{31:1, 23:1, 15:1, 8:1, default:0};

typedef struct {real r, th;} C;
var C x = '{th:PI/2.0, r:1.0};
```

A positional notation without keys can also be used. For example:

```
var int B[4] = '{a, b, c, d};
var C y = '{1.0, PI/2.0};
```

WITH

An *assignment pattern* specifies a correspondence between a collection of expressions and the fields and elements in a data object or data value. An assignment pattern has no self-determined data type, but can be used as one of the sides in an assignment-like context (see below) when the other side has a self-determined data type. An assignment pattern is built from braces, keys, and expressions and is prefixed with an apostrophe. For example:

```
var int A[N] = '{default:1};
var integer i = '{31:1, 23:1, 15:1, 8:1, default:0};

typedef struct {real r, th;} C;
var C x = '{th:PI/2.0, r:1.0};
var real y, z;
```

A positional notation without keys can also be used. For example:

```
var int B[4] = '{a, b, c, d};
var C y = '{1.0, PI/2.0};
'{a, b, c, d} = B;
```

When an assignment pattern is used as the left-hand side of an assignment-like context, the positional notation shall be required and each member expression shall have a bit-stream data type that is assignment compatible with and has the same number of bits as the data type of the corresponding element on the right-hand side.

In 8.13, REPLACE

Neither a static cast nor a default correspondence between an expression in an assignment pattern and a field or element in a data object shall be considered an assignment-like context.

An assignment pattern can be used to construct a structure or array value by prefixing the pattern with the name of a data type to form an assignment pattern expression that yields the value that a variable of the data type would hold if it were initialized using the assignment pattern.

```
      typedef logic [1:0] [3:0] T;
shortint'({T'{1,2}, T'{3,4}}) // yields 16'sh1234
```

WITH

Neither a static cast nor a default correspondence between an expression in an assignment pattern and a field or element in a data object shall be considered an assignment-like context. No other contexts shall be considered assignment-like contexts. In particular, none of the following shall be considered assignment-like contexts:

-- A static cast
-- A default correspondence between an expression in an assignment pattern and a field or element in a data object
-- A port expression in a module, interface or program declaration
-- The passing of a value to a subroutine ref port
-- A port connection to an inout or ref port of a module, interface or program

An assignment pattern can be used to construct a structure or array value by prefixing the pattern with the name of a data type to form an assignment pattern expression that yields the value that a variable of the data type would hold if it were initialized using the assignment pattern. An assignment pattern can be used to construct or deconstruct a structure or array by prefixing the pattern with the name of a data type to form an *assignment pattern expression*. Unlike an assignment pattern, an assignment pattern expression has a self-determined data type and is not restricted to being one of the sides in an assignment-like context. When an assignment pattern expression is used in a right-hand side expression, it shall yield the value that a variable of the data type would hold if it were initialized using the assignment pattern.

```
typedef logic [1:0] [3:0] T;
shortint'({T'{1,2}, T'{3,4}}) // yields 16'sh1234
```

When an assignment pattern expression is used in a left-hand side expression, the positional notation shall be required and each member expression shall have a bit-stream data type that is assignment compatible with and has the same number of bits as the corresponding element in the data type of the assignment pattern expression. If the right-hand side expression has a self-determined data type, then it shall be assignment compatible with and have the same number of bits as the data type of the assignment pattern expression.

```
typedef byte U[3];
var U A = '{1, 2, 3};
var byte a, b, c;
U'{a, b, c} = A;
U'{c, a, b} = '{a+1, b+1, c+1};
```

An assignment pattern expression shall not be used in a port expression in a module, interface or program declaration.

Section

In A.6.8, ADD

assignment_pattern_net_lvalue ::=

'{ net_lvalue {, net_value } }

assignment_pattern_variable_lvalue ::=

'{ variable_lvalue {, variable_value } }

In A.8.5, in net_lvalue, ADD

| [assignment_pattern_expression_type] assignment_pattern_net_lvalue

In A.8.5, in variable_lvalue, ADD

| [assignment_pattern_expression_type] assignment_pattern_variable_lvalue

In 8.13, in the definition of assignment-like context, REPLACE

A port connection to an input port of a module, interface, or program

WITH

A port connection to an input or output port of a module, interface, or program

AND REPLACE

The passing of a value to a subroutine input port

WITH

The passing of a value to a subroutine input, output or inout port

AND REPLACE

A non-default correspondence between an expression in an assignment pattern and a field or element in a

data object

WITH

A non-default correspondence between an expression in an assignment pattern and a field or element in a data object or data value

In 8.13, REPLACE

nor a default correspondence between an expression in an assignment pattern and a field or element in a data object

WITH

nor a default correspondence between an expression in an assignment pattern and a field or element in a data object or data value

In 8.13.1, REPLACE

Verilog uses concatenation braces to construct simple bit vectors. SystemVerilog adds a similar syntax to support the construction of arrays.

WITH

Verilog uses concatenation braces to construct and deconstruct simple bit vectors. SystemVerilog adds a similar syntax to support the construction and deconstruction of arrays.

In 8.13.2, REPLACE

A structure value can be constructed with a structure assignment pattern built from member expressions using braces and commas, with the members in declaration order.

WITH

A structure value can be constructed and deconstructed with a structure assignment pattern built from member expressions using braces and commas, with the members in declaration order.

In 3.7, REPLACE

Array literals are array assignment patterns with constant member expressions (see 8.13.1). An array literal must have a type, which may be either explicitly indicated with a prefix or implicitly indicated by an assignment-like context.

WITH

Array literals are array assignment patterns or pattern expressions with constant member expressions (see 8.13.1). An array literal must have a type, which may be either explicitly indicated with a prefix or implicitly indicated by an assignment-like context.

In 3.8, REPLACE

Structure literals are structure assignment patterns with constant member expressions (see 8.13.2). A structure literal must have a type, which may be either explicitly indicated with a prefix or implicitly indicated by an assignment-like context (see 4.14).

WITH

Structure literals are structure assignment patterns or pattern expressions with constant member expressions (see 8.13.2). A structure literal must have a type, which may be either explicitly indicated with a prefix or implicitly indicated by an assignment-like context (see 4.14).

In 32.39, in bullet 5, REPLACE

when the curly braces of the assignment pattern are prefixed by a data type name.

WITH

when the curly braces of the assignment pattern are prefixed by a data type name to form an assignment pattern expression.

In 32.39, in bullet 6, REPLACE

Nesting of assignment pattern expressions shall be preserved.

WITH

Nesting of assignment pattern expressions patterns shall be preserved.

In 32.39, in bullet 6, REPLACE

The assignment pattern that initializes the struct variable ABC uses member, type and default keys. The **vpiOperand** traversal would represent this assignment pattern expression as:

WITH

The assignment pattern that initializes the struct variable ABC uses member, type and default keys. The **vpiOperand** traversal would represent this assignment pattern expression as:

In 32.39, in bullet 6, REPLACE

or some other equivalent positional assignment pattern expression.

WITH

or some other equivalent positional assignment pattern expression.

In 32.39, in bullet 6, REPLACE

The assignment pattern which initializes the array variable varr uses index and default keys. The **vpiOperand** traversal would represent this assignment pattern expression

WITH

The assignment pattern which initializes the array variable varr uses index and default keys. The **vpiOperand** traversal would represent this assignment pattern expression

In 32.39, in bullet 7, REPLACE

For the assignment pattern expression '{2{y}},

WITH

For the assignment pattern expression '{2{y}},

In the Index, ADD references to "assignment pattern" and "assignment pattern expression" that point to 8.13.

Section

Section **1.5, 6.10, 11.3.3, 24.2, 24.3, A.2.2.1, A.2.4, Syntax 4-1, Syntax 19-4, Syntax 22-1**
Type references

In A.2.2.1 and Syntax 4-1, ADD

type_reference ::=
type (expression[28] **)**
| **type (** data_type **)**

and in Footnote A.10.28, REPLACE

The expression that is used as the argument to the $typeof system function shall contain no hierarchical references.

WITH

~~The~~ An expression that is used as the argument ~~to the $typeof system function~~ in a type_reference ~~shall contain no hierarchical references~~ shall not contain any hierarchical references or references to elements of dynamic objects.

In A.2.2.1 and Syntax 4-1, in data_type, ADD

| type_reference

and ADD the following footnote

When a type_reference is used in a net declaration, it shall be preceded by a net type keyword, and when it is used in a variable declaration, it shall be preceded by the **`var`** keyword.

In A.2.4 and Syntax 22-1, REPLACE

type_assignment ::=
type_identifier = data_type
| type_identifier = **$typeof (** expression[28] **)**

Section

| type_identifier = **$typeof (** data_type **)**

WITH

type_assignment ::=
type_identifier **=** data_type
| type_identifier = **$typeof (** expression[28] **)**
| type_identifier = **$typeof (** data_type **)**

and, in Syntax 22-1, ADD an excerpt of

type_reference

In A.8.4, in constant_primary, ADD

| type_reference

and ADD the following footnote,

It shall be legal to use a type_reference constant_primary as the casting_type in a static cast. It shall be illegal for a type_reference constant_primary to be used with any operators except the equality/inequality and case equality/inequality operators.

In 1.5, REPLACE

Clause 24—System Tasks and System Functions: This clause describes several extensions to Verilog system tasks and system functions, including a $typeof function, $typename function, $bits size function, range function, array querying functions, assertion control tasks, random number functions, program control tasks, coverage functions, and enhancements to Verilog system tasks and

system functions.

WITH

Clause 24—System Tasks and System Functions: This clause describes several extensions to Verilog system tasks and system functions, including a $typeof function, $typename function, $bits size function, range function, array querying functions, assertion control tasks, random number functions, program control tasks, coverage functions, and enhancements to Verilog system tasks and system functions.

In 6.10, ADD

The **type** operator provides a way to refer to the data type of an expression. A type reference can be used like a type name or local type parameter, for example, in casts, data object declarations, and type parameter assignments and overrides. It can also be used in equality/inequality and case equality/inequality comparisons with other type references, and such comparisons are considered to be constant expressions. When a type reference is used in a net declaration, it shall be preceded by a net type keyword, and when it is used in a variable declaration, it shall be preceded by the **var** keyword.

```
var type(a+b) c, d;

c = type(i+3)'(v[15:0]);
```

The **type** operator applied to an expression shall represent the self-determined result type of that expression. The expression shall not be evaluated and shall not contain any hierarchical references or references to elements of dynamic objects.

The **type** operator can also be applied to a data type.

```
localparam type T = type(bit[12:0]);
```

When a type reference is used in an equality/inequality or case equality/inequality comparison, it shall only be compared with another type reference. Two type references shall be considered equal in such comparisons if and only if the types they refer to match (see 6.9.1).

```
bit [12:0] A_bus, B_bus;
parameter type bus_t = type(A_bus);
generate
   case (type(bus_t))
      type(bit[12:0]): addfixed_int #(bus_t) (A_bus,
      B_bus);
      type(real): add_float #($typeof(A_bus))
      (A_bus,B_bus);
   endcase
endgenerate
```

In 11.3.3, REPLACE

A constant function may call any system function that may be called in a constant expression. This includes $bits, the array query functions and $typeof.

WITH

A constant function may call any system function that may be called in a constant expression. This includes $bits and, the array query functions and $typeof.

DELETE 24.2

In 24.3, REPLACE

This process is similar to the way that type equality is computed, except that array ranges and built-in equivalent types are not normalized in the generated string. Thus $typename can be used in string comparisons for stricter type-checking of arrays than $typeof.

WITH

This process is similar to the way that type equality matching (see 6.9.1) is computed, except that array ranges and built-in equivalent types are not normalized simple bit vectors types with predefined widths are distinguished from those with user-defined widths in the generated string. Thus $typename can be used in string comparisons for stricter type-checking type comparison of arrays than with type references $typeof.

In 24.3, REPLACE

the expression shall not contain any hierarchical identifiers or references to elements of dynamic objects

WITH

the expression shall not contain any hierarchical identifiers references or references to elements of dynamic objects

In Syntax 19-4, update footnote 32 to be consistent with A.10.32 – specifically, REPLACE

When a net_port_type contains a data_type, it shall only be legal to omit the explicit net_type_or_implicit when declaring an **inout** port.

WITH

When a net_port_type contains a data_type, it shall only be legal to omit the

explicit net_type net_type_or_implicit when declaring an **inout** port.

Section **10.9**
Fine-grain process control

In 10.9, REPLACE

```
      task kill();
task await();
task suspend();
```

WITH

```
      task function void kill();
task await();
task function void suspend();
```

Section **Table 21-3**
Default coverage goal

In Table 21-3, in the column "Default", REPLACE

90

WITH

~~90~~100

Section **4.16, 13.11.1**
Constraint syntax

In 4.16, REPLACE

```
with { p.length > 1 && p.payload.size == p.length }
```

WITH

```
with { p.length > 1 && p.payload.size == p.length ; }
```

In 13.11.1, REPLACE

```
< length }
```

WITH

```
< length ; }
```

AND REPLACE

```
> length }
```

WITH

```
> length ; }
```

Section **8.13.2**
Wire declarations in packages

In 8.13.2, DELETE

~~In fact, it descends the nesting to a built-in type or a packed array of them.~~

Section **5.5, 24.7, Syntax 24-5 $unpacked_dimensions**

In 5.5, REPLACE

$size, and $dimensions

WITH

$size, ~~and~~ $dimensions, and $unpacked_dimensions

In 24.7, REPLACE

If the first argument to an array dimension function

WITH

If the first argument to an array ~~dimension~~ query function

In Syntax 24-5, replace

```
|       $dimensions ( array_identifier )
|       $dimensions ( data_type )
```

WITH

```
|       ~~$dimensions~~ array_dimensions_function ( array_identifier )
|       ~~$dimensions~~ array_dimensions_function ( data_type )
```

AND after the array_query_function production ADD

```
array_dimensions_function ::=
        $dimensions
  |     $unpacked_dimensions
```

In 24.7, after "-- 0 for any other type" ADD

```
-- $unpacked_dimensions shall return:
  -- The total number of unpacked dimensions for an array (static or dynamic)
  -- 0 for any other type
```

Section

Section **14.2.4, 14.3.4, 14.3.6, 14.3.8**
Return value on try_* functions

In 14.2.4, REPLACE

the method returns 1 and execution continues

WITH

the method returns ~~1~~ a positive integer and execution continues

In 14.3.4, REPLACE

and the function returns 1

WITH

and the function returns ~~1~~ a positive integer

In 14.3.6, REPLACE

the method returns –1

WITH

the method returns ~~–1~~ a negative integer

In 14.3.6, REPLACE

the method returns 1

WITH

–

the method returns 1 a positive integer

In 14.3.8, REPLACE

the method returns –1

WITH

the method returns –1 a negative integer

In 14.3.8, REPLACE

the method returns 1

WITH

the method returns 1 a positive integer

Section **6.7**
Assignments

In 6.7, REPLACE

A continuous assignment is implied when a variable is connected to an input port declaration. This makes assignments to a variable declared as an input port illegal. A continuous assignment is implied when a variable is connected to the output port of an instance. This makes procedural or continuous assignments to a variable connected to the output port of an instance illegal.

WITH

A continuous assignment is implied when a variable is connected to an input port declaration. This makes assignments to a variable declared as an input port illegal. A continuous assignment is implied when a variable is connected to the output port of an instance. This makes additional procedural or continuous assignments to a variable connected to the output port of an instance illegal.

Section **A.1.4, A.2.1.3, A.2.3, A.2.4, A.2.5, A.2.7, A.10, Syntax 4-1, Syntax 4-2, Syntax 5-2, Syntax 11-1**
Variable dimension

In A.2.5 and Syntax 5-2, REPLACE

variable_dimension[12] ::=

{ sized_or_unsized_dimension }

| associative_dimension

| queue_dimension

WITH

variable_dimension[~~12~~] ::=

unsized_dimension ~~{sized_or_unsized_dimension }~~

| unpacked_dimension

| associative_dimension

| queue_dimension

In A.2.5, REPLACE

unsized_dimension[11] ::= **[ ]**

WITH

unsized_dimension[~~11~~] ::= **[ ]**

In A.2.5, DELETE

~~sized_or_unsized_dimension ::= unpacked_dimension | unsized_dimension~~

Replace each occurrence of

variable_dimension

Section

WITH

{ variable_dimension }

IN

A.1.4

ansi_port_declaration

A.2.1.3 and Syntax 4-2

type_declaration

A.2.3 (twice in each production)

list_of_member_identifiers

list_of_tf_variable_identifiers

list_of_variable_identifiers

list_of_variable_port_identifiers

A.2.4 and Syntax 4-1

variable_decl_assignment

A.2.7 and Syntax 11-1

tf_port_item

In A.10, footnote 11, REPLACE

unsized_dimension is permitted only in declarations of import DPI functions, see dpi_function_proto.

WITH

In packed_dimension, unsized_dimension is permitted only in declarations of import DPI functions, see dpi_function_proto.

In A.10, DELETE footnote 12.