Issue #266 - Negative vote from Entity #6

Cliff-notes

I have begun the process of examining and proposing fixes for normative notes. The process will take some time to complete.

I have copied all of the appropriate text that needs to be examined and have proposed fixes for all of the sections through 8.13.

All of the sections after 8.13 still need to be completed. The proposals at this time are just a duplicate of the original text, which will make it easier for me to modify to form proposals.

I want to make sure I am proposing modifications according to the wishes of the subcommittees before completing this task.

Regards - Cliff Cummings

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 declarations 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 normal union of normal structures, which are C-compatible and have members in ascending address order.

4.13
WAS: 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: 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: NOTE: Queues and dynamic arrays have the same assignment and argument passing semantics.
No change required (??) - this appears to be informational and I believe the information could be gathered independent of this note. This note just helps the reader recognize an interesting fact about queues and dynamic arrays.

6.6 (multiple occurrences)
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.

The text preceding the following note/paragraph is about declared for-loop variables and class methods and how they are by default automatic. Swapping the order of the sentences in the paragraph seems to help.

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

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: ???

I do not even understand the note in this paragraph and the accompanying examples did not clarify the point of confusion. Does someone else want to understand this note enough to make a proposal??

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: 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).

I believe this is just an informative note. 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)

WAS: Note that the function prototype does not need to match the actual function declaration exactly.
PROPOSED: Note that the function prototype does not need to match the actual function declaration exactly.

WAS: Note that the assignment operator from a float to a float cannot be overloaded here because it is already legal.
PROPOSED: Note that the assignment operator from a float to a float cannot be overloaded here because it is already legal.

8.19
WAS: 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.
PROPOSED: 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.

9.4
WAS: Note: by specifying **unique** or **priority**, it is not necessary to code a **default** case to trap unexpected case values.
PROPOSED: Note: by specifying **unique** or **priority**, it is not necessary to code a **default** case to trap unexpected case values.

9.4.1
WAS: Note that static type checking ensures that *V* and the pattern have the same type.
PROPOSED: Note that static type checking ensures that *V* and the pattern have the same type.

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, but note 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: Note that SystemVerilog does not include the C **goto** statement.

9.10 (multiple occurrences)

WAS: Note that such an expression is evaluated when a changes, and not when enable changes.

PROPOSED: Note that such an expression is evaluated when a changes, and not when enable changes.

WAS: Also note that **iff** has precedence over **or**. This can be made clearer by the use of parentheses

PROPOSED: Also note that **iff** has precedence over **or**. This can be made clearer by the use of parentheses

---

11.4.2

WAS: Note that in the example, no change other than addition of the **ref** keyword is needed.

PROPOSED: Note that in the example, no change other than addition of the **ref** keyword is needed.

11.4.3

WAS: Note that default values are only allowed with the ANSI style declaration.

PROPOSED: Note that default values are only allowed with the ANSI style declaration.

11.5 (multiple occurrences)

WAS: Note that all these SystemVerilog functions must have identical argument types, as defined in the next paragraph.

PROPOSED: Note that all these 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: Note that **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.
PROPOSED: 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.

WAS: 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: 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.

WAS: 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: 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.

12.6
WAS: Note that the assignment to status is not:
PROPOSED: Note that the assignment to status is not:

12.7
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.

PROPOSED: 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.

12.8
WAS: Note that static class properties can be used without creating an object of that type.
PROPOSED: Note that static class properties can be used without creating an object of that type.

12.10
WAS: Note that in writing methods, members can be qualified with **this** to refer to the current instance, but it is usually unnecessary.
PROPOSED: Note that in writing methods, members can be qualified with **this** to refer to the current instance, but it is usually unnecessary.

12.11
WAS: Note, **new** was executed only once, so only one object has been created.
PROPOSED: Note, **new** was executed only once, so only one object has been created.

12.17
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.
PROPOSED: 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.

12.24 (multiple occurrences)
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** …).
PROPOSED: 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** …).

WAS: Note that the class keyword in the statement typedef class C2; is not necessary, and is used only for documentation purposes.
PROPOSED: Note that the class keyword in the statement typedef class C2; is not necessary, and is used only for documentation purposes.

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

13.4.11 (multiple occurrences)
WAS: 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: Note that the two constraints above are not completely equivalent; C2 is bidirectional (length can constrain v and vice-versa), whereas C1 is not.

WAS: 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.
PROPOSED: 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.

13.5.3
WAS: **13.5.3 Randomization methods notes**
PROPOSED: **13.5.3 Randomization methods notes**

13.11

WAS: 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: 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:

14.3.7
WAS: Note that calling peek() can cause one message to unblock more than one process.
PROPOSED: Note that calling peek() can cause one message to unblock more than one process.

15.2
WAS: 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.
PROPOSED: 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.

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*]);
PROPOSED: 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*]);

17.4
WAS: 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.
PROPOSED: 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.

18.2

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

PROPOSED: No change necessary

18.3

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

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

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.

18.6

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: 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.

18.7.3
WAS: Note that a comma for the omitted *clocking_event* argument is not needed, as it does not fall within the specified arguments.
PROPOSED: Note that a comma for the omitted *clocking_event* argument is not needed, as it does not fall within the specified arguments.

18.8
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: Note that when a local variable is a formal argument of a sequence declaration, it is illegal to declare the variable, as shown below.

18.11.2
WAS: 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: 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.

18.12.3
WAS: The scope of a clocking event does not flow into the reset condition of **disable iff**.
Note that juxtaposing two clocking events nullifies the first of them:
@(d) @(c) x
PROPOSED: The scope of a clocking event does not flow into the reset condition of **disable iff**.
Note that juxtaposing two clocking events nullifies the first of them:
@(d) @(c) x

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: Note: The pass and fail statements 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: Note that **assume** does not provide an action block, as the actions for an assumption serve no purpose.

18.15 (multiple occurrences)
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. 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.

WAS: Note that the latter situation will occur if the design contains more than one instance of a module containing a **bind** statement.
PROPOSED: Note that 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. Note that an alternative way of handling this problem is to use configurations.

19.7

WAS: 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: 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: Note that where the data types differ 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. Note that the logic type 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: 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.

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. Note that this 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: 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.

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

20.8.1
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. 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.

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 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.

22.2

WAS: Note that function $isunbounded is used for checking the validity of the actual arguments.

No change - this is an informational usage 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: 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]

25.2

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

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

28.3

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

PROPOSED: Note that 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.

No change - this is an informational 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: 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.

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 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.

28.4.4 (multiple occurrences)

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: 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.

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: 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.

WAS: // Note the following import uses the same foreign function for
// implementation as the prior import, but has different SystemVerilog name
// and provides a default value for the argument.

No change - this is an informational example comment

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.
PROPOSED: — 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.

28.6 (multiple occurrences)
WAS: Note that class member functions cannot be exported, but all other SystemVerilog functions can be exported.
PROPOSED: Note that 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.
PROPOSED: *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.

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.
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. 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.

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
PROPOSED: 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

---

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: 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.

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

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: 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

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. Note that once this control is issued, no further assertion related actions shall be permitted.

29.5.2
WAS: NOTE—In this release, the only step control constant available is vpiAssertionClockSteps, indicating callbacks on a per assertion/clock-tick basis.
PROPOSED: NOTE—In this release, the only step control constant available is vpiAssertionClockSteps, indicating callbacks on a per assertion/clock-tick basis.

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. Note that 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: ‘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.

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: 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).

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: 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.

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: 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.

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. 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.

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. Note that this shall always 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. 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).

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: 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.

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);
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.

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. 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.

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 "IMPORTANT DETAILS" 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: IMPORTANT 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)

...

(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. 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.

...

(32.14 under Notes)
WAS: 19) Note that:

logic var == reg
var bit == reg bit
array var == reg array
PROPOSED: 19) Note that:
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:
Note: *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: **E.5 Semantic constraints**
Note that the constraints expressed here merely restate those expressed in 28.4.1.

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: 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.

E.6.1

WAS: NOTE—The actual argument can have both packed and unpacked parts of an array; either can be multidimensional.
PROPOSED: NOTE—The actual argument can have both packed and unpacked parts of an array; either can be multidimensional.

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: 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

---

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., SystemVerilogcalls 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).
NOTE—The above range mapping from SystemVerilog to C applies to calls made in both directions, i.e., SystemVerilogcalls 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: Note that 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.
PROPOSED: Note that context is transitive through imported and export context tasks and functions declared in the same scope.

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. 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.

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: 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.

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. 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.

E.11
WAS: Plus, inquiries about the dimensions and the original boundaries of SystemVerilog actual arguments are supported for open arrays.
PROPOSED: NOTE—Both packed and unpacked array dimensions can be unsized. 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.

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.
PROPOSED: NOTE—This provides some degree of flexibility and allows the programmer to control the trade-off of performance vs. convenience.

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.

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.
PROPOSED: NOTE—No specific representation of an array is assumed here; hence, all functions use a generic pointer void *.
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 *.

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: — 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.

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: 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.

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*|. Note that |*w*| is either a non-negative integer or infinity.
