Review for System-Verilog 3.1 Draft 4

Annex A and Annex B

Flowing are my review notes for Annexes A and B of the System-Verilog 3.1 draft 4. my notes refer to changes added to the standard by ALL the System-Verilog sub-committees.

The comments related to the Keywords in Annex B are labeled as DJ-AB-<id> these are easy issues that could be added to the final publication. The comments related to the BNF in Annex A are labeled as DJ-AA-<id> some of these issues are more complex.

Contact Information

Dan Jacobi
Intel Corp.
E-mail : Dan.Jacobi@intel.com
Phone : (972)-4-8655855

Annex A – BNF

DJ-AA-1. No BNF for the data-type **chandle**. This data-type is described under section 3.7. DWS: Modified A.2.6 and A.2.7 to be chandle – LRM-168

DJ-AA-2. No BNF for **final** blocks. These blocks are described under section 8.6. DWS: LRM-171 opened

DJ-AA-3. No BNF for the keyword **local**. This keyword is described under section 11.16. DWS: Done in LRM-172.

DJ-AA-4. No BNF for the keyword **super**. This keyword is described under section 11.13. DWS: LRM-170 opened.

DJ-AA-5. No BNF for the keyword **this**. This keyword is described under section 11.9. DWS: LRM-173 opened.

DJ-AA-6. The following production:

```
constraint_item ::=
        constraint_expression ;
        | constraint_expression => constraint_item_or_block
        | if ( constraint_expression ) constraint_item_or_block [ else
        constraint_item_or_block ]
```

Causes a problem known as “The dangling if-else ambiguity”
The ‘else’ in the following constraint item can be matched to both ‘if’s”

```
if (mode != large)
if (mode == small)
len < 10;
else // Does the else apply to if (mode != large) or if (mode == small)
len > 100;
```

The following langadge should be added to section 12.4.6 (this is similar to the language describing the if-else statements in the IEEE 1364-2001).

Because the else part of an if-else style constraint declarations is optional, there can be confusion when an else is omitted from a nested if sequence. This is resolved by always associating the else with the closest previous if that lacks an else. In the example below, the else goes with the inner if, as shown by indentation:

```
if (mode != large)
        if (mode == small)
                len < 10;
        else // else applies to preceding if
                len > 100;
```

DWS: Handled in LRM-174

DJ-AA-7. Remove the second editors note from A.2.6 – the signing can be added before the function's data type (After the function/automatic keyword).

DWS: Done in LRM-53

DJ-AA-8. Under A.2.6 REPLACE

named_function_proto::= function_data_type function_identifier **(**
list_of_function_proto_formals **)**

WITH

named_function_proto::= [signing] function_data_type function_identifier **(**
list_of_function_proto_formals **)**

And remove the third editors note regarding the signed function prototype.

DWS: Done in LRM-54

DJ-AA-9. Remove the Editors note from A.8.3 the current BNF reflects the changes accepted by the SV-BC.

DWS: Done in LRM-57

DJ-AA-10. Under A.9.3 Remove the production:

real_identifier ::= identifier

And remove the following editors note. The real_identifier is a "left over" from the IEEE 1364 standard.

DWS: Done in LRM-59

DJ-AA-11. Under A.9.3 Remove the production:

state_identifier ::= identifier

And remove the following editors note.

DWS: Done in LRM-60

DJ-AA-12. Do not remove the production:

text_macro_identifier ::= identifier

Remove the following editor's note. The text_macro_identifier token is referenced from with-in the IEEE 1364-2001 (section 19.3.1) even though it is not referenced from the 1364 BNF. Removing this production will cause miss-consistency with the 1364 standards.

DWS: Done in LRM-61

DJ-AA-13. BNF for more than one initial statement and more than one-step assignment with in a for loop is missing. Currently the BNF for the following for loop statement is missing:

for (a=1,b=2 ; a<10 & b<200 ; a=a+1,b=b*2) …

DWS: Create LRM-175

DJ-AA-14. Under A.6.3 the following production is problematic

action_block ::= [statement] [**else** statement] **;**

The following assertion statement can be interpreted in more than one way:

assert (cond) **if** (cond2) a = 1; **else** a = 2;;

One-way to parse this statement - If the assertion succeeds (cond == true) evaluate the if-else conditional statement.

The other way to parse this statement is – If the assertion succeeds (cond == true) and if cond2 is true than assign 'a' with the value 1. If the assertion fails then assign 'a' with the value 2.

Some language needs to be added chapter 17 that deals with this case and with more complicated cases such as nested assertion and nested conditional if-else statements and the nesting of assertions in if-else statements and vise-versa for example how should the following RTL be parsed "

always

if (c1) **assert** (c2); **else assert**(c3); **else if** (c4) a =1; **else** a = 2;; **else** a=3;;;;;;;;

DWS: Create LRM-176

DJ-AA-15. Under A.2.10 the following production has a loop:

sequence_expr ::=
[cycle_delay_range] sequence_expr { cycle_delay_range sequence_expr }
…

The token sequence_expr can parse itself I would recommend the following change to the begging of the sequence_expr production

sequence_expr ::=
~~[cycle_delay_range] sequence_expr { cycle_delay_range sequence_expr }~~
cycle_delay_range sequence_expr { cycle_delay_range sequence_expr }
| sequence_expr cycle_delay_range sequence_expr
{ cycle_delay_range sequence_expr }
…

DWS: Create LRM-176

DJ-AA-16. Under A.2.10 the prentices of the sequence_expr production should be in bold.

REPLACE

sequence_expr ::=
…
| (sequence_expr) [sequence_abbrev]
…

WITH

sequence_expr ::=
…
| **(** sequence_expr **)** [sequence_abbrev]
…

DWS: Done in LRM-178

DJ-AA-17. Precedence for the operators added by the sequence_expr production under A.2.10 should be added to section. The precedence for the and, intersect, or, first_match, throughout, and within operators is not defined.

The following sequence expression :

d1 **intersect** d2 **within** d2

Can be parsed in two ways:

(d1 **intersect** d2) **within** d2
d1 **intersect** (d2 **within** d2)

DWS: Create LRM-179

DJ-AA-18. Under A.2.10 the production sequence_expr causes a problem when adding a prentices around an expression that relates from the primary production.
In the following example:
d1 **within (**d2 & d3**)**
The prentices may be parsed using the primary production (primary :: = **(** mintypmax_expression **)**) or using the sequence_expr production (sequence_expr ::= **(** sequence_expr **)** [sequence_abbrev])
DWS: Create LRM-180

DJ-AA-19. Under A.2.9 replace the name of the token const_range_expression to sequence_const_range_expression. The original name causes confusion with the constant_range_expression token.
DWS: Create LRM-181

Annex B – Keywords

DJ-AB-1. The keyword **private** does not appear in Annex B even though it appears under the BNF A.1.8.
DWS: Private is replaced with local which is in the BNF and the keywords.

DJ-AB-2. The keyword **handle** was removed from Annex B even though it appears under the BNF A.2.6.
DWS: Handle is replaced with chandle in the BNF so handle should be removed from Annex B.

DJ-AB-3. The keyword **endsequence** does not appear in Annex B even though it appears under the BNF A.2.10.
DWS: Added in LRM_166

DJ-AB-4. The keyword **endproperty** does not appear in Annex B even though it appears under the BNF A.2.10.
DWS: Added in LRM_166

DJ-AB-5. The keyword **randomize** does not appear in Annex B even though it appears under the BNF A.6.2.
DWS: randomize is NOT a language construct. It is a built-in method. As such it does not need to be in either the BNF or in the keyword list (anymore than any of the system tasks/functions do). This is true for ALL built-in methods and classes.

DJ-AB-6. Should the sequence **"DPI"** be marked as a keyword? It appears under the BNF A.2.6.
DWS: Why would you want a string to be a keyword. It is just a string. That is why it was chosen so it would not be a keyword.