# Corrections to draft5

## 1.1 Annex H, Title

## ~~SystemVerilog~~ Formal Semantics of Concurrent Assertions ~~Semantics~~

## 1.2 Page 201, Section 17.1

— Concurrent assertions are based on clock semantics and use sampled values of variables. One of the goals of SystemVerilog assertions is to provide a common semantic meaning for assertions so that they can be used to drive various design and verification tools. Many tools, such as formal verification tools, evaluate circuit descriptions using cycle-based semantics, which typically relies on a clock signal or signals to drive the evaluation of the circuit. Any timing or event behavior between clock edges is abstracted away. Concurrent assertions incorporate these clock ~~semantic~~ semantics. While this approach generally simplifies the evaluation of a circuit description, there are a number of scenarios under which this cycle-based evaluation

## 1.3 Page 204, Section 17.3

The clock expression that controls evaluation of a sequence can be more complex than just a single signal name. An expression such as (clk && gating_signal) ~~and~~ and (clk iff gating_signal) ~~could~~ can be used to represent a gated clock. Other more complex expressions are possible. However, in ~~In~~ order to ensure proper behavior of the system and conform as closely as possible to truly cycle-based semantics, the signals in a clock expression must be glitch-free and should only transition once at any simulation time.

## 1.4 Page 208, Section 17.5

A ## followed by a number or range specifies the delay from the current ~~cycle~~ clock tick to the beginning of the sequence that follows. The delay ##1 indicates that the beginning of the sequence that follows is one clock tick later than the current ~~cycle~~ clock tick. The delay ##0 indicates that the beginning of the sequence that follows is at the same clock tick as the current ~~cycle~~ clock tick.

## 1.5 Page 210, Section 17.6

~~Formal arguments can be optionally specified. A formal argument is untyped, and is used for syntactic replacement of a name or an expression in the sequence.~~

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.

## 1.6 Page 211, Section 17.6

In this example, sequences s1 and s2 are evaluated on successive posedge ~~event~~ events of clk. The sequence s3 is evaluated on successive negedge events of clk.

 

### 1.7 Page 211, Section 17.6

To use a named sequence as a sub-sequence of another sequence, simply reference its name. The evaluation of a sequence that references a named sequence is performed in the same way as if the named sequence was contained as a lexical part of the referencing sequence, with the formal arguments of the named sequence replaced by the actual ones and the remaining variables in the named sequence resolved according to the scope of the declaration of the named sequence. An example is shown below:

### 1.8 Page 221, Section 17.7.4

If te1 and te2 are sampled expressions (not sequences), the sequence (te1 and te2) matches if te1 and te2 both evaluate to ~~be~~ true.

### 1.9 Page 234, Section 17.8

The type of variable is explicitly specified. The variable can be assigned at the end point of any syntactic subsequence by placing the ~~subsequence~~ sub-sequence, comma separated from the sampling assignment, in parentheses. For example, if in

### 1.10 Page 239, Section 17.10

An ~~x~~ `X` and ~~z~~ `Z` value of a bit is not counted towards the number of ones.

### 1.11 Page 239, Section 17.10

In this example, source sequence e1 is evaluated at clock clk, while the destination sequence e2 is evaluated at clock sysclk. In e2, the end point of the instance e1(ready,proc1,proc2) is tested to occur sometime after the occurrence of inst. Notice that method ~~matched~~ `matched` only tests for the end point of e1(ready,proc1,proc2) and has no bearing on the starting point of e1(ready,proc1,proc2).

Local variables can be passed into an instance of a named sequence to which ~~matched~~ `matched` is applied. The same restrictions apply as in the case of ~~ended~~ `ended`. Values of local variables sampled in an instance of a named sequence to which ~~matched~~ `matched` is applied will flow out under the same conditions as for ~~ended~~ `ended`. See Section 17.8.

As with ~~ended~~ `ended`, a sequence instance to which ~~matched~~ `matched` is applied can have multiple matches in a single cycle of the destination sequence clock. The multiple matches are treated semantically the same way as matching both disjuncts of an or. In other words, the thread evaluating the destination sequence will fork to account for such distinct local variable valuations.

### 1.12 Page 246, Section 17.11.1

```
|->
(
   (
      (data_valid && (data_valid_tag == tag))
      |->
      (data == expected_data[i*8+:8])
   )
   and
```