The intention of this proposal is to make LRM description of the sampled value functions consistent with their formal semantics and to allow sampled value functions in assertions with arbitrary clocks. P1800 LRM requires that the specified clocking event for a sampled value function in an assertion must be identical to that of the assertion. For example,

```
assert property (@clk done |=> (out == $past(q, 2,enable, clk1)));
```

In the above example, `clk1` must be identical to `clk`. This proposal allows `clk1` to be different than `clk`.

16.8.3 Sampled value functions

REPLACE

When these functions are used in an assertion, the clocking event argument of the functions, if specified, shall be identical to the clocking event of the expression in the assertion. In the case of multiclock assertions, the appropriate clocking event for the expression where the function is used is applied to the function.

WITH

~~When these functions are used in an assertion, the clocking event argument of the functions, if specified, shall be identical to the clocking event of the expression in the assertion. In the case of multiclock assertions, the appropriate clocking event for the expression where the function is used is applied to the function.~~

ADD

Note to editor: Add to the end of the subclause.

When the sampled value functions are used in an assertion, the clocking event argument of the functions may be different from the clocking event of the expression in the assertion, as determined by the clock resolution (see 16.15 Note to editor: Subclause "Clock resolution").

Consider the following assertions:

```
bit clk, fclk, req, gnt, en;
...
a1: assert property (@(posedge clk) en && $rose(req) |=> gnt);
a2: assert property
  (@(posedge clk) en && $rose(req, @(posedge fclk)) |=> gnt);
```

Both assertions `a1` and `a2` read: whenever `en` is high and `req` rises, at the next cycle `gnt` must be asserted. In both assertions, the rise of `req` occurs if and only if the sampled value of `req` at the current **`posedge`** of `clk` is `1'b1` and the sampled value of `req` at a particular prior point is distinct from `1'b1`. The assertions differ in the specification of the prior point. In `a1` the prior point is the preceding posedge of `clk`, while in `a2` the prior point is the most recent prior posedge of `fclk`.

Annex F

Formal semantics of concurrent assertions

F.2 Abstract syntax

F.2.3 Derived forms

ADD

F.2.3.6 Derived sampled value functions

— `$sampled(`e`)` $\equiv$ e.

— `$rose(`e`,`c`)` $\equiv$ `$past(`b`,1,1,`c`)` `!==` `1` `&&` b `===` `1`, where b is the LSB of e.

— `$fell(`e`,`c`)` $\equiv$ `$past(`b`,1,1,`c`)` `!==` `0` `&&` b `===` `0`, where b is the LSB of e.

— `$stable(`e`,`c`)` $\equiv$ `$past(`e`,1,1,`c`)` `===` e.

— `$changed(`e`,`c`)` $\equiv$ `$past(`e`,1,1,`c`)` `!==` e.

F.4 Extended expressions

F.4.1 Extended booleans

REPLACE

w denotes a nonempty finite or infinite word over Σ, j denotes an integer so that $0 \leq j < |w|$, and $T(V)$ denotes an instance of a clocked or unclocked sequence that is passed the local variables V as actual arguments.

— $w^j, L_0, L_1 \models T(V)$`.ended` iff there exist $0 \leq i \leq j$ and L so that both $w^{i,j}, \{\}, L \models T(V)$ and $L_1 = L_0|_D \cup L_V$, where $D = dom(L_0) - (dom(L) \cap V)$.

— $w^j, L_0, L_1 \models$ `@(c)` $T(V)$`.matched` iff there exist $0 \leq i < j$ so that $w, i, L_0, L_1 \models T(V)$`.ended` and $w^{i+1,j}, \{\}, \{\} \models$ `(!c [*0:$] ##1 c)`.

— $w^j \models$ `@(c) $stable(e)` iff there exists $0 \leq i < j$ so that $w^{i,j}, \{\}, L \models$ `(c ##1 c [->1])` and $e[w^i] = e[w^j]$.

— $w^j \models$ `@(c) $rose(e)` iff $b[w^j] = 1$ and (if there exists $0 \leq i < j$ so that $w^{i,j}, \{\}, L \models$ `(c ##1 c [->1])` then $b[w^j] \neq 1$) where b is the LSB of e.

— $w^j \models$ `@(c) $fell(e)` iff $b[w^j] = 0$ and (if there exists $0 \leq i < j$ so that $w^{i,j}, \{\}, L \models$ `(c ##1 c [->1])` then $b[w^j] \neq 0$) where b is the LSB of e.

WITH

w denotes a nonempty finite or infinite word over Σ, j denotes an integer so that $0 \leq j < |w|$, and $T(V)$ denotes an instance of a clocked or unclocked sequence that is passed the local variables V as actual arguments.

— $w^j, L_0, L_1 \models T(V)$`.ended` iff there exist $0 \leq i \leq j$ and L so that both $w^{i,j}, \{\}, L \models T(V)$ and $L_1 = L_0|_D \cup L_V$, where $D = dom(L_0) - (dom(L) \cap V)$.

— $w^j, L_0, L_1 \models$ `@(c)` $T(V)$`.matched` iff there exist $0 \leq i < j$ so that $w, i, L_0, L_1 \models T(V)$`.ended` and $w^{i+1,j}, \{\}, \{\} \models$ ~~`(!c [*0:$] ##1 c)`~~ `c[->1]`.

— ~~$w^j \models$ `@(c) $stable(e)` iff there exists $0 \leq i < j$ so that $w^{i,j}, \{\}, L \models$ `(c ##1 c [->1])` and $e[w^i] = e[w^j]$.~~

— ~~$w^j \models$ `@(c) $rose(e)` iff $b[w^j] = 1$ and (if there exists $0 \leq i < j$ so that $w^{i,j}, \{\}, L \models$ `(c ##1 c [->1])` then $b[w^j] \neq 1$) where b is the LSB of e.~~

— ~~$w^j \models$ `@(c) $fell(e)` iff $b[w^j] = 0$ and (if there exists $0 \leq i < j$ so that $w^{i,j}, \{\}, L \models$ `(c ##1 c [->1])` then $b[w^j] \neq 0$) where b is the LSB of e.~~

F.4.2 Past

REPLACE

w denotes a nonempty finite or infinite word over Σ, and j denotes an integer so that $0 \leq j < |w|$.

— Let $n \geq 1$. If there exist $0 \leq i < j$ so that $w^{i,j}, \{\}, \{\} \models$ `(c ##1 c [->n ])`, then `@(c)$past(e, n)`$[w^j] = e[w^j]$. Otherwise, `@(c)$past(e, n)`$[w^j]$ has the value `x`.

— `$past(e)` $\equiv$ `$past(e,1)`

WITH

w denotes a nonempty finite or infinite word over Σ, and j denotes an integer so that $0 \leq j < |w|$.

— ~~Let $n \geq 1$. If there exist $0 \leq i < j$ so that $w^{i,j}, \{\}, \{\} \models$ `(c ##1 c [->n ])`, then `@(c)$past(e, n)`$[w^j] = e[w^j]$. Otherwise, `@(c)$past(e, n)`$[w^j]$ has the value `x`.~~

— ~~`$past(e)` $\equiv$ `$past(e,1)`~~

Let $n \geq 1$. If there exists $0 \leq i < j$ so that
$w^{i,j}, \{\}, \{\} \models$ `((c && ` e_2`) ##1 (c && ` e_2`[=n-1] ##1 1)`, then `$past(`e_1`, n, `e_2`, c)`$[w^j] = e_1[w^i]$. Otherwise, `$past(`e_1`, n, `e_2`, c)`$[w^j]$ is the result of evaluating the expression e_1 using the initial values of the variables comprising the expression. The initial value of a static variable is the value assigned in its declaration, or, in the absence of such an assignment, it is the default (or uninitialized) value of the corresponding type (see 6.7, Table 6-1). The initial value of any other variable or signal is the default value of the corresponding type (see 6.7, Table 6-1).