Overlapping operators in multiclock environment

Motivation

Currently, the operators `##0`, `|->` and **`if`** … **`else`**, may only be specified when the same clocking event is used on all the operands. This proposal aims to relax this restriction to allow these operators to be used in multiclock environment provided that global clocking is defined.

Changing clock `clk1` to clock `clk2` with zero delay has the following meaning: wait for the tick of `clk1` and then wait for the nearest tick `@clk2`, i.e., if the tick of `clk2` happens at the same time as the tick of `clk1`, there is no waiting after `clk1` tick, if the ticks of `clk1` and `clk2` do not happen simultaneously, the next tick of `clk2` after `clk1` is awaited.

Allowing clock change with zero delay simplifies multiclock property definition in case of general LTL operators (see Mantis 1932). Currently 1932 contains the limitation that the operand properties shall begin on the same clock as the clock of the underlined property. E.g.,

```
@(posedge clk0) (@(posedge clk0)s1 until @(posedge clk0) s2)
```

is legal, but

```
@(posedge clk0) (@(posedge clk1)s1 until @(posedge clk0) s2)
```

is illegal because the `(@(posedge clk1)s1` operand property begins on a different clock from the clock.

The current proposal allows lifting this limitation.

16.13.1 Multiclocked sequences

REPLACE

Multiclocked sequences are built by concatenating singly clocked subsequences using the single-delay concatenation operator `##1`. This operator is nonoverlapping and synchronizes between the clocks of the two sequences. The single delay indicated by `##1` is understood to be from the end point of the first sequence, which occurs at a tick of the first clock, to the nearest strictly subsequent tick of the second clock, where the second sequence begins.

For example, consider

```
@(posedge clk0) sig0 ##1 @(posedge clk1) sig1
```

A match of this sequence starts with a match of `sig0` at **`posedge`** `clk0`. Then `##1` moves the time to the nearest strictly subsequent **`posedge`** `clk1`, and the match of the sequence ends at that point with a match of `sig1`. If `clk0` and `clk1` are not identical, then the clocking event for the sequence changes after `##1`. If `clk0` and `clk1` are identical, then the clocking event does not change after `##1`, and the above sequence is equivalent to the singly clocked sequence

```
@(posedge clk0) sig0 ##1 sig1
```

When concatenating differently clocked sequences, the maximal singly clocked subsequences are required to admit only nonempty matches.

WITH

In the absence of global clocking, multiclocked ~~Multiclocked~~ sequences are built by concatenating singly clocked subsequences using the single-delay concatenation operator `##1`. This operator is nonoverlapping and synchronizes between the clocks of the two sequences. The single delay indicated by `##1` is understood to be from the end point of the first sequence, which occurs at a tick of the first clock, to the nearest strictly subsequent tick of the second clock, where the second sequence begins.

For example, consider

```
@(posedge clk0) sig0 ##1 @(posedge clk1) sig1
```

A match of this sequence starts with a match of `sig0` at **`posedge`** `clk0`. Then `##1` moves the time to the nearest strictly subsequent **`posedge`** `clk1`, and the match of the sequence ends at that point with a match of `sig1`. If `clk0` and `clk1` are not identical, then the clocking event for the sequence changes after `##1`. If `clk0` and `clk1` are identical, then the clocking event does not change after `##1`, and the above sequence is equivalent to the singly clocked sequence

```
@(posedge clk0) sig0 ##1 sig1
```

In the absence of global clocking, when ~~When~~ concatenating differently clocked sequences, the maximal singly clocked subsequences are required
to admit only nonempty matches.

INSERT just before 16.13.2

When global clocking is defined, multi-clocked sequences can be built by concatenating singly clocked subsequences using the same concatenation operator as in single-clocked sequences. The operator is overlapping and thus allows using `##0`. If the clocking events of the two operands do not occur in the same simulation time step, then `##0` behave the same way as the multiclock `##1`.

For example, consider again

```
@(posedge clk0) sig0 ##0 @(posedge clk1) sig1
```

but suppose that global clocking has been defined. If `sig0` matches and the tick of posedge `clk1` occurs in the same time step as the match of `sig0`, then `sig1` evaluates in that time step. If the tick of **`posedge`** `clk1` does not occur in the same time step, then `sig1` evaluates at the nearest future tick of **`posedge`** `clk1` in which case it behaves as if the multiclock `##1` were used to concatenate the two sequences. Only `##0` and `##1` are allowed on clocking event change in a multiclock sequence when global clocking is defined.

16.13.2 Multiclocked properties

REPLACE

Because synchronization between distinct clocks always requires strict advance of time, the two property building operators that require special care with multiple clocks are the overlapping implication `|->` and **`if`**/**`if`**...**`else`**.

WITH

~~Because~~ In the absence of global clocking, Because synchronization between distinct clocks always requires strict advance of time, the two property

building operators that require special care with multiple clocks are the overlapping implication `|->` and **`if`**/**`if`**...**`else`**.

INSERT just before 16.13.3

When global clocking is defined, multi-clocked properties can be formulated using `|->` and **`if`**/**`if...else`**. Their behavior is as follows:

If the clocking event at which the antecedent of the overlapping implication matched occurs in the same time step as the leading clocking event of the consequent then the consequent starts evaluating in that time step. Otherwise the implication behaves the same way as the non-overlapping implication `|=>`, that is, the consequent starts evaluating on the nearest future clocking event of its leading clock.

The clocking event of the condition of **`if`**/**`if`**...**`else`** can be different from the clocking events of the **`if`** and **`else`** clauses. The behavior of

```
@clk1 if (b) @clk2 p1 else @clk3 p3
```

where `b` is an expression and `p1` and `p2` are properties is equivalent to

```
(@clk1 b |-> @clk2 p1) and (@clk1 !b |-> @clk3 p3)
```