### 16.8.3 Global clocking past and future sampled value functions

Note to the editor: Please shift the subsequent clauses accordingly. The references in the text are provided relative to Draft4.

This subclause describes the system functions available for accessing the nearest past and future values of an expression as sampled by the global clock. They may be used only if global clocking is defined (see Editor please insert reference to global clocking). These functions include the capability to access the sampled value at the global clock tick that immediately precedes or follows the timestep at which the function is called. Sampled value is explained in 16.4. The following functions are provided:

Global clocking past sampled value functions:

```
$past_gclk(expression)
$rose_gclk(expression)
$fell_gclk(expression)
$stable_gclk(expression)
$changed_gclk(expression)
```

Global clocking future sampled value functions:

```
$future_gclk(expression)
$rising_gclk(expression)
$falling_gclk(expression)
$steady_gclk(expression)
$changing_gclk(expression)
```

The behavior of the global clocking past sampled value functions can be defined using the sampled value functions as follows:

```
$past_gclk(v)     ≡  $past(v,,,  @$global_clock)
$rose_gclk(v)     ≡  $rose(v,    @$global_clock)
$fell_gclk(v)     ≡  $fell(v,    @$global_clock)
$stable_gclk(v)   ≡  $stable(v,  @$global_clock)
$changed_gclk(v)  ≡  $changed(v, @$global_clock)
```

The global clocking future sampled value functions are similar except that they use the subsequent value of the expression.

`$future_gclk(v)` is the sampled value of v at the next global clocking tick.
The other functions are defined as follows:

— `$rising_gclk(expression)` returns true if the sampled value of the LSB of the expression is changing to 1 at the next global clocking tick. Otherwise, it returns false.

— `$falling_gclk(expression)` returns true if the sampled value of the LSB of the expression is changing to 0 at the next global clocking tick. Otherwise, it returns false.

— `$steady_gclk(expression)` returns true if the sampled value of the expressions does not change at the next global clock tick. Otherwise it returns false.

— `$changing_gclk(expression)` is the complement of `$steady_gclk`, i.e., `!$steady_gclk(expression)`.

The global clocking sampled value functions may be invoked only in *property_expr* or *in sequence_expr*; this implies that they shall not be used in assertion action blocks. The global clocking past sampled value

functions are a special case of the sampled value functions, and therefore the regular restrictions imposed on the sampled value function arguments apply (see 16.8.4). Additional restrictions are imposed on the usage of the global clocking future sampled value functions: they shall not be nested and they shall not be used in assertions containing sequence match items (see 16.9, 16.10). The following example illustrates the illegal usage of the global clocking future sampled value functions:

Example.

```
// Illegal: global clocking future sampled value functions
// shall not be nested
a1: assert property (@clk $future_gclk(a || $rising_gclk(b));

sequence s;
   bit v;
   (a, v = a) ##1 (b == v)[->1];
endsequence : s

// Illegal: a global clocking future sampled value function shall not
// be used in an assertion containing sequence match items
a2: assert property (@clk s |=> $future_gclk(c));
```

Even though global clocking future sampled value functions depend on future values of their arguments, the interval of simulation timesteps for an evaluation attempt of an assertion containing global clocking future sampled value functions is defined as though the future sampled values were known in advance. The end of the evaluation attempt is defined to be the last tick of the assertion clock and is not delayed any additional timesteps up to the next global clocking tick.

The behavior of **disable iff** and other asynchronous assertion related controls such as `$assertkill` (see 19.10 and 19.11) is with respect to the interval of the evaluation attempt defined above. If, for example, `$assertkill` is executed in a timestep strictly after the last tick of the assertion clock for the evaluation attempt, then it shall not affect that attempt, even if `$assertkill` is executed no later than the next global clocking tick.

Execution of the action block of an assertion containing global clocking future sampled value functions shall be delayed until the global clocking tick that follows the last tick of the assertion clock for the attempt. If the evaluation attempt fails and `$error` is called by default (see 16.14.1), then `$error` shall be called at the global clocking tick that follows the last tick of the assertion clock.

A tool specific message that reports the starting or ending timestep of an evaluation attempt of an assertion containing global clocking future sampled functions shall be consistent with the definition above of the interval of simulation timesteps for the evaluation attempt. The message may also report the timestep in which it is written, which may be that of the global clocking tick that follows the last tick of the assertion clock.

Example 1:

The Table 16-25 (Note to the editor: Please adjust numbering as needed) shows the values returned by the global clocking future sampled value functions for `sig` at different time moments.

The following assertion states that the signal may change only on falling clock:

```
a1: assert property (@$global_clock $changing_gclk(sig)
                                     |-> $falling_gclk(clk))
   else $error(”sig is not stable”);
```

In the Figure 16-4 (Note to the editor: Please adjust numbering as needed) we see that this property is violated at time 80. The vertical arrows indicate the ticks of the global clock. The error message `$error(”sig is not stable”)` is executed at time 90.

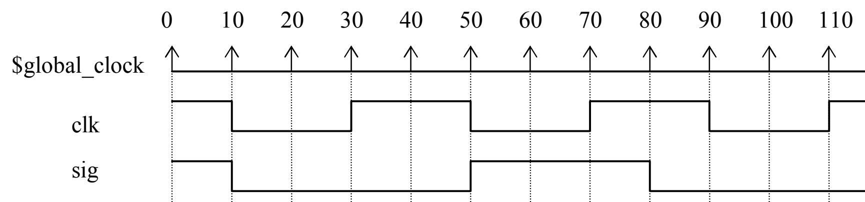


**Figure 16-4 Future value change (Note to the editor: Please adjust numbering as needed)**

**Table 16-25: Global clocking future sampled value functions**

| Time | $sampled(sig) | $future_gclk(sig) | $rising_gclk(sig) | $falling_gclk(sig) | $changing_gclk(sig) | $steady_gclk(sig) |
|---|---|---|---|---|---|---|
| 10 | 1’b1 | 1’b0 | 1’b0 | 1’b1 | 1’b1 | 1’b0 |
| 30 | 1’b0 | 1’b0 | 1’b0 | 1’b0 | 1’b0 | 1’b1 |
| 40 | 1’b0 | 1’b0 | 1’b0 | 1’b0 | 1’b0 | 1’b1 |
| 50 | 1’b0 | 1’b1 | 1’b1 | 1’b0 | 1’b1 | 1’b0 |
| 80 | 1'b1 | 1'b0 | 1’b0 | 1’b1 | 1’b1 | 1’b0 |

Example 2:

The following assumption states that a signal `sig` must remain stable between two falling edges of a clock `clk` as sampled by global clocking. This differs from the property in Example 1 in the case where the first falling edge of `clk` has not yet occurred. In Example 1, `sig` is not allowed to change in that case, but in this example `sig` can toggle freely while we wait for `clk` to begin.

```
a2: assume property(@$global_clock
        $falling_gclk(clk) ##1 (!$falling_gclk(clk)[*1:$]) |->
                                        $steady_gclk(sig));
```

Example 3.

Assume that the signal rst is high between times 82 and 84, and is low at all other time moments. Then the following assertion

```
a3: assert property (@$global_clock disable iff (rst) $changing_gclk(sig)
                                        |-> $falling_gclk(clk))
    else $error(”sig is not stable”);
```

fails at time 80 (see Figure 16-14 Note to the editor: please, adjust the reference) since `rst` is inactive at time 80. The interval of the failing evaluation attempt starts and ends at time 80. Although `rst` is active prior to the execution of the action block at time 90, the attempt is not disabled.

## 19.10 Assertion control system tasks

ADD before “When invoked with no arguments, the system task shall apply to all assertions.”

The details related to the behavior of `$assertkill` and `$assertoff` for assertions referring to global clocking future sampled value functions are explained in 16.8.3.

## 19.11 Assertion action control system tasks

ADD before “When invoked with no arguments, the system task shall apply to all assertions.”

The details related to the behavior of `$assertpassoff`, `$assertfailoff`, and `$assertvacuousoff` for assertions referring to global clocking sampled future value functions are explained in 16.8.3.

## 19.12 Assertion system functions

Note to the editor - add the following text at the end of this sub-clause:
The following functions allow to access the sampled value of an expression at the immediate past and future ticks of the global clock and to detect changes in the sampled value from the past (resp. current) tick of the global clock to its current (resp. next) tick.

Global clocking past sampled value functions:

```
$past_gclk(expression)
$rose_gclk(expression)
$fell_gclk(expression)
$stable_gclk(expression)
$changed_gclk(expression)
```

Global clocking future sampled value functions:

```
$future_gclk(expression)
$rising_gclk(expression)
$falling_gclk(expression)
$steady_gclk(expression)
$changing_gclk(expression)
```

These functions are discussed in 16.8.3.

### 38.4.2 Placing assertions callbacks

REPLACE

d) In contrast to *cb_time*, the content of *attemptStartTime* is always the start time of the actual attempt of an assertion. It can be used as a unique identifier that distinguishes the attempts of any given assertion.

WITH

d) In contrast to *cb_time*, the content of *attemptStartTime* is always the start time of the actual attempt of an assertion. It can be used as a unique identifier that distinguishes the attempts of any given assertion.

e) See 38.4.2.1 for callbacks for assertions containing global clocking future sampled value functions.

#### 38.4.2.1 Placing callbacks for assertions with global clocking future sampled value functions

Callback execution for assertions referring to global clocking future sampled value functions (see 16.8.3) has the following peculiarities:

— the callback is executed at the nearest tick of the global clock strictly following the callback event

— *cb_time* contains the time of the callback event.

For example:

```
a1: assert property(@(posedge clk) $falling_gclk(a) |=> b);
a2: assert property(@(posedge clk) a |=> $falling_gclk(b));
```

Suppose that `clk` rises at simulation time 10, 30, 50, …, the global clock changes at simulation time 2, 4, 6, …, the sampled value of `a` is `1'b1` at time 10, and it is `1'b0` at time 12, and `$assertkill` is issued at time 11. For both assertions `a1` and `a2` the callback executes at time 12, and not at time 11 when `$assertkill` directive was issued. *cb_time* has the time value of 11 – the time when `$assertkill` was actually issued, and *attemptStartTime* has the time value of 10.