6.7.3 Event triggered statements

The following two restrictions apply to statements which are evaluated as a result of an event being triggered.

- The statement can not have expressions which use analog operators. This statement can not maintain its internal state since it is only executed intermittently when the corresponding event is triggered.
- The statement can not be a contribution statement because it can generate discontinuity in analog signals.

6.7.4 Global events

Global events are generated by a simulator at various stages of simulation. The user model can not generate these events. These events are detected by using the name of the global event in an event expression with the **@** operator.

Global events are pre-defined in Verilog-AMS HDL. These events can not be redefined in a model.

The pre-defined global events are shown in Syntax 6-11.

```
global_event ::=
      initial_step [ ( analysis_list ) ]
    | final_step [ ( analysis_list ) ]

analysis_list ::=
    analysis_name { , analysis_name }

analysis_name ::=
    " analysis_identifier "
```

Syntax 6-11—Global events

initial_step and **final_step** generate global events on the first and the last point in an analysis respectively. They are useful when performing actions which should only occur at the beginning or the end of an analysis. Both global events can take an optional argument, consisting of an analysis list for the active global event.

Examples:

For example,

```
@(initial_step("ac", "dc"))   // active for dc and ac only
@(initial_step("tran")) // active for transient only
```

Table 6-1 describes the return value of `initial_step` and `final_step` for standard analysis types. Each column shows the return-on-event status. One (`1`) represents *Yes* and zero (`0`) represents *No*. A Verilog-AMS HDL simulator can use any or all of these

typical analysis types. Additional analysis names can also be used as necessary for specific implementations. (See Section 4.5.1 for further details.)

Table 6-1—Return Values for initial_step and final_step

Analysis[a]	**DCOP** OP	**TRAN** OP p1 pN	**AC** OP p1 pN	**NOISE** OP p1 pN
initial_step	1	1 0 0	1 0 0	1 0 0
initial_step("ac")	0	0 0 0	1 0 0	0 0 0
initial_step("noise")	0	0 0 0	0 0 0	1 0 0
initial_step("tran")	0	1 0 0	0 0 0	0 0 0
initial_step("dc")	1	0 0 0	0 0 0	0 0 0
initial_step(*unknown*)	0	0 0 0	0 0 0	0 0 0
final_step	1	0 0 1	0 0 1	0 0 1
final_step("ac")	0	0 0 0	0 0 1	0 0 0
final_step("noise")	0	0 0 0	0 0 0	0 0 1
final_step("tran")	0	0 0 1	0 0 0	0 0 0
final_step("dc")	1	0 0 0	0 0 0	0 0 0
final_step(*unknown*)	1	0 0 0	0 0 0	0 0 0

a. p*X* Table 6-1 designates frequency/time analysis point *X*, *X* = `1` to *N*; *OP* designates the Operating Point.

Examples:

The following example measures the bit-error rate of a signal and prints the result at the end of the simulation.

```
module bitErrorRate (in, ref) ;
   input in, ref ;
   electrical in, ref ;
   parameter real period=1, thresh=0.5 ;
   integer bits, errors ;

   analog begin
      @(initial_step) begin
         bits = 0 ;
         errors = 0 ;
      end
```

```
        @(timer(0, period)) begin
            if ((V(in) > thresh) != (V(ref) > thresh))
                errors = errors + 1 ;
            bits = bits + 1 ;
        end

        @(final_step)
          $strobe("bit error rate = %f%%", 100.0 * errors / bits ) ;
    end
endmodule
```

initial_step and **final_step** take a list of quoted strings as optional arguments. The strings are compared to the name of the analysis being run. If any string matches the name of the current analysis name, the simulator generates an event on the first point and the last point of that particular analysis, respectively.

If no analysis list is specified, the **initial_step** global event is active during the solution of the first point (or initial DC analysis) of every analysis. The **final_step** global event, without an analysis list, is only active during the solution of the last point of every analyses.

6.7.5 Monitored events

Monitored events are detected using event functions (see Syntax 6-12) with the @ operator. The triggering of a monitored event is implicit due to change in signals, simulation time, or other runtime conditions.

```
event_function ::=
      cross_function
    | timer_function
```

Syntax 6-12—Monitored events

6.7.5.1 cross function

The **cross()** function is used for generating a monitored analog event to detect threshold crossings in analog signals when the expression crosses zero (`0`) in the specified direction. In addition, **cross()** controls the timestep to accurately resolve the crossing.

The general form is

```
cross ( expr [ , dir [ , time_tol [ , expr_tol ] ] ] );
```

where *expr* is required, and *dir*, *time_tol*, and *expr_tol* are optional. All arguments are real expressions, except *dir* (which is an integer expression). If the tolerances are not defined, then the tool (e.g., the simulator) sets them. If either or both tolerances are defined, then the direction shall also be defined.

The direction indicator can only evaluate to `+1`, `-1`, or `0`. If it is set to `0` or is not specified, the event and timestep control occur on both positive and negative crossings of the signal. If *dir* is `+1` (or `-1`), the event and timestep control only occur on rising edge