

8. Events 1

The *e* language provides temporal constructs for specifying and verifying behavior over time. All *e* temporal language features depend on the occurrence of events, which are used to synchronize activity with a simulator and within the *e* program. 5

8.1 Causes of events 10

Table 1 describes how an event is made to occur. 10

Table 1—Event causation 15

Syntax	Cause of the event
<code>event a is (@b and @c)@d</code>	Derived from other events (see Clause 9).
<code>event a is rise('top.b')@sim</code>	Derived from behavior of a simulated device (see Clause 9).
<code>event a;</code> <code>meth_b()@c is { ... ; emit a; ... };</code>	By the emit action in procedural code (see 8.3.2). 20

8.2 Scope of events 25

Events are defined as a part of a *struct* definition. When a *struct* is instantiated, each instance has its own event instances. Each event instance has its own schedule of occurrences. There is no relation between occurrences of event instances of the same type. All references to events are to event instances. 30

The scoping rules for events are similar to other struct members, such as *fields*. 30

- If a path is provided, use the event defined in the struct instance pointed to by the path.
- If no path is provided, the event is resolved at compile time. The current struct instance is searched.
- If the event instance is not found, a compile-time error shall be issued. 35

8.3 Defining and emitting named events

This section describes the **event** and **emit** constructs. 40

45

50

55

8.3.1 event

Purpose	Define a named event	
Category	Struct member	
Syntax	event <i>event-type</i> [is [only] <i>temporal-expression</i>]	
Parameters	<i>event-type</i>	The name of the event type (any legal <i>e</i> identifier.)
	<i>temporal-expression</i>	An event or combination of events and temporal operators. <i>See also</i> : Clause 9.

Events can be attached to temporal expressions, using the option **is** [**only**] *temporal-expression* syntax, or they can be unattached. An *attached event* is emitted automatically during any tick in which the temporal expression attached to it succeeds. For a definition of the success of a temporal expression, see 9.3.

Events, like methods, can be redefined in *struct* extensions. The **is only** *temporal-expression* syntax in a *struct* extension is used to change the definition of an event, e.g., to define an event once and then attach it to several different temporal expressions under different **when** *struct* subtypes.

8.3.2 emit

Purpose	Cause a named event to occur	
Category	Action	
Syntax	emit [<i>struct-exp.</i>] <i>event-type</i>	
Parameters	<i>struct-exp</i>	An expression referring to the <i>struct</i> instance in which the event is defined.
	<i>event-type</i>	The type of event to emit.

This causes an event of the specified type to occur.

- Emitting an event causes the immediate evaluation of all temporal expressions containing that event.
- The **emit** event does not consume time. It can be used in regular methods and in TCMs.
- The simplest usage of **emit** is to synchronize two TCMs, where one TCM waits for the named event and the other TCM emits it.

Syntax example:

```
emit ready;
```

8.4 Predefined events

Predefined events are emitted at particular points in time.

8.4.1 General predefined events

Table 2 lists the general predefined events.

Table 2—Predefined events

Predefined event	Description
sys.any	Emitted on every tick.
sys.tick_start	Emitted at the start of every tick.
sys.tick_end	Emitted at the end of every tick.
session.start_of_test	Emitted once at test start.
session.end_of_test	Emitted once at test end.
struct.quit	Emitted when a struct's quit() method is called. Only exists in structs that contain events or have members that consume time (for example, time-consuming methods, or on struct members).
sys.new_time	In stand-alone operation (no simulator), this event is emitted on every sys.any event. When a simulator is being used, this event is emitted whenever a callback occurs and the attached simulator's time has changed since the previous callback.

8.4.1.1 sys.any

This event is a special event that defines the finest granularity of time. The occurrence of any event in the system causes an occurrence of the **any** event at the same tick. In stand-alone *e* program operation (that is, with no simulator attached), the **sys.any** event is the only one that occurs automatically. It typically is used as the clock for stand-alone operation.

8.4.1.2 sys.tick_start

This event is provided mainly for visualizing and debugging the program flow in the event viewer.

8.4.1.3 sys.tick_end

This event is provided mainly for visualizing and debugging the program flow in the event viewer. It also can be used to provide visibility into changes of values that are computed during the tick, such as the values of coverage items.

8.4.1.4 session.start_of_test

The first action the predefined **run()** method executes is to emit the **session.start_of_test** event. This event is typically used to anchor temporal expressions to the beginning of a test.

8.4.1.5 session.end_of_test

This event is typically used to sample data at the end of the test. This event cannot be used in temporal expressions, as it is emitted after evaluation of temporal expression has been stopped. The **on session.end_of_test** struct member is typically used to prepare the data sampled at the end of the test.

8.4.1.6 struct.quit

This only exists in *structs* that contain temporal members (events, **on**, **expect**, or TCMs). It is emitted when the struct's **quit()** method is called, to signal the end of time for the struct.

The first action executed during the check test phase is to emit the **quit** event for each *struct* that contains it. This event can be used to cause the evaluation of temporal expressions that contain the **eventually** temporal operator (and check for **eventually** temporal expressions that have not been satisfied).

8.4.1.7 **sys.new_time**

This event is emitted on every **sys.any** event in stand-alone operation (no simulator). When a simulator is being used, this event is emitted whenever a callback occurs [and](#) the attached simulator's time has changed since the previous callback.

8.4.2 **Simulation time and ticks**

Using any of the following expressions causes the DUT to be monitored for a change in that expression:

- **rise** | **fall** | **change** (*HDL expression*) **@sim**
- **wait delay** *expression*
- Verilog event

For each simulation delta cycle where a change in at least one of these monitored expressions occurs, the simulator passes control to the *e* program. If simulation time has advanced since the last time control was passed to the *e* program, a **new_time** event is issued. In any case, **tick_start** and **any** events are issued. Then, after emitting all events initiated by changes in monitored expressions in that simulation delta cycle, a **tick_end** event is issued.

Thus, the **new_time** event corresponds to a new simulation time slot and a tick corresponds to a simulation delta cycle where at least one monitored expression changes.

Multiple ticks can occur in the same simulation time slot under the following conditions:

- when a new value is driven into the DUT and that value causes a change in a monitored HDL object, as in a clock generator
- when a monitored event is derived from another monitored event, as in a clock tree
- when a zero-delay HDL subprogram is call from *e*.

For an explanation of when values are assigned, see 16.5.

NOTE—Glitches that occur in a single simulation time slot are ignored; only the first occurrence of a particular monitored event in a single simulation time slot is recognized.