

## 10. Temporal struct members 1

In addition to the **event** struct member (see 8.3.1), there are two struct members used for temporal coding: **on** and **expect | assume**. 5

### 10.1 on 10

<b>Purpose</b>	Specify a block of actions that execute on an event	10
<b>Category</b>	Struct member	
<b>Syntax</b>	<b>on</b> <i>event-type</i> { <i>action</i> ; ...}	15
<b>Parameters</b>	<i>event-type</i> The name of an event that invokes the action block.	
	<i>action</i> ; ...      A block of non-time-consuming actions.	

This defines a struct member which executes a block of actions immediately whenever a specified event occurs. An **on** struct member is similar to a regular method, except the action block for an **on** struct member is executed before TCMs waiting for the same event. The actions are executed in the order in which they appear in the action block. 20

To extend an **on** struct member, repeat its declaration and use a different action block. This has the same effect as using **is also** to extend a method. 25

The **on** struct member is implemented as a method, named **on\_event-type()**. To invoke the action block without the occurrence of the event, call the **on\_event-type()** method. This method can be extended like any other method, by using **is**, **is also**, **is only**, or **is first**. 30

The following restrictions also apply.

- The named event shall be local to the struct in which the **on** is defined.
- The **on** action block shall not contain any time-consuming actions (TCMs). 35

*See also:* 2.2.3 and 15.1.3.

Syntax example: 40

```
on xmit_ready {transmit();};
```

45

50

55

## 10.2 expect | assume

<b>Purpose</b>	Define a temporal rule
<b>Category</b>	Struct member
<b>Syntax</b>	<b>expect</b>   <b>assume</b> [ <i>rule-name</i> <b>is</b> [ <b>only</b> ]] <i>temporal-expression</i> [ <b>else dut_error</b> ( <i>string-exp</i> )] <b>expect</b>   <b>assume</b> <i>rule-name</i>
<b>Parameters</b>	<i>rule-name</i> A name that uniquely identifies the rule from other rules or events within the struct. It can be used to override the temporal rule later on in the code or change from <b>expect</b> to <b>assume</b> or vice-versa.
	<i>temporal-expression</i> A temporal expression that is always expected to succeed. Typically involves a temporal yield (=>) operation.
	<i>string-exp</i> A string or a method that returns a string. If the temporal expression fails, the string is printed, or the method is executed and its result is printed.

Both the **expect** and **assume** struct members are used for defining temporal properties.

- When a simulation-based tool executes the *e* program, it evaluates the rule expressed by the temporal expression. If the temporal expression fails (see 9.3) at some point in time, the tool reports an error as specified with the **dut\_error** clause (if no **dut\_error** clause is specified, the tool prints the rule name). The notion of failure of the temporal expression implies a new evaluation starts on every state following a state in which the sampling event occurs (see 9.3.3). Simulation-based tools typically treat **expect** and **assume** in exactly the same manner.
- When a formal verification tool analyses the *e* program, **expect** struct members are interpreted as rules the tool needs to verify, whereas **assume** struct members are interpreted as constraints on legal behavior. This means the tool looks for program execution paths where the temporal expression bound to an **expect** fails (see 9.3) [and](#) none of the temporal expressions bound to the **assumes** fail.

Once a rule has been defined, it can be modified using the **is only** syntax and can be changed from an **expect** to an **assume** or vice-versa. To perform multiple verification runs, vary the rules slightly or use the same set of rules in different **expect/assume** combinations.

Syntax example:

```
expect @a => {[1..5];@b} @clk;
```

*Example*

This example defines an **expect**, *bus\_cycle\_length*, which requires the length of the bus cycle to be no longer than 1000 cycles.

```
struct bus_e {
    event bus_clk is change('top.b_clk') @sim;
    event transmit_start is rise('top.trans') @bus_clk;
    event transmit_end is rise('top.transmit_done') @bus_clk;
    event bus_cycle_length;
    expect bus_cycle_length is
        @transmit_start => {[0..999];@transmit_end} @bus_clk
        else dut_error("Bus cycle did not end in 1000 cycles");
};
```