

## 25. Simulation-related constructs

This clause describes simulation-related actions, expressions, and routines. *See also:* Clause 6.

**\*\*Is this material now too thin??**

### 25.1 force

<b>Purpose</b>	Force a value on an HDL object
<b>Category</b>	Action
<b>Syntax</b>	<b>force</b> ' <i>HDL-pathname</i> ' = <i>exp</i>
<b>Parameters</b>	<i>HDL-pathname</i> The full path name of an HDL object (see 25.3), including any expressions.
	<i>exp</i> Any scalar expression or literal constant, as long as it is composed only of 1's and 0's. No x or z values are allowed. Thus, 16'hf0f1 or <code>sys.my_val + 5</code> are legal values.

This forces an HDL object to a specified value, overriding the current value and preventing the DUT from driving any value. The HDL object remains at the specified value until a subsequent **force** action from *e* or until freed by a **release** action (see 25.2). The following also apply.

- If part of a vectored object is forced, the **force** action is propagated to the rest of the object.
- To force single elements of an array of a scalar integer or enumerated type, use the predefined routine **simulator\_command()** (see 25.4).

Syntax example:

```
force '~/top/sig' = 7;
```

### 25.2 release

<b>Purpose</b>	Remove a <b>force</b> action from an HDL object
<b>Category</b>	Action
<b>Syntax</b>	<b>release</b> ' <i>HDL-pathname</i> '
<b>Parameters</b>	<i>HDL-pathname</i> The full path name of an HDL object previously specified in a <b>force</b> action.

This releases the HDL object which previously has been forced (see 25.1).

Syntax example:

```
release 'top.sig';
```

1

175 - 176

5

134 - 136

10

15

20

104

111

25

30

137

35

40

45

111

103

50

55

### 25.3 'HDL-pathname'

<b>Purpose</b>	Access HDL objects, using <i>full-path-names</i>
<b>Category</b>	Expression
<b>Syntax</b>	' <i>HDL-pathname</i> [ <i>index-exp</i>   <i>bit-range</i> ] [ <i>@(x   z   n)</i> ]'
<b>Parameters</b>	<i>HDL-pathname</i> The full path name of an HDL object, including any expressions and composite data.
	<i>index-exp</i> Accesses a single bit of a Verilog vector, a single element of a Verilog memory, or a single vector of a VHDL array of vectors.
	<i>bit-range</i> <i>bit-range</i> has the format [ <i>high-bit-num</i> : <i>low-bit-num</i> ] and is extracted from the object from the high-bit to low-bit. Slices of buses are treated exactly as they are in HDL languages. They need to be specified in the same direction as in the HDL code and reference the same bit numbers.
	<i>@x</i>   <i>@z</i> Sets or gets the x or z component of the value. When this notation is not used in accessing an HDL object, <i>e</i> translates the values of x to zero (0) and z to one (1). When reading HDL objects using <i>@x</i> (or <i>@z</i> ), <i>e</i> translates the specified value (x or z) to one (1) and all other values to zero (0). When writing HDL objects, if <i>@x</i> (or <i>@z</i> ) is specified, <i>e</i> sets every bit that has a value of 1 to x (or z). In this way, <i>@x</i> or <i>@z</i> acts much like a data mask, manipulating only those bits that match the value of x or z.
<i>@n</i> When this specifier is used for driving HDL objects, the new (simulator) value is visible immediately (now). The <i>default mode</i> is to buffer projected values and update only at the end of the tick.	

This accesses Verilog and VHDL objects from *e*.

Syntax example:

```
'~/top/sig' = 7;
print '~/top/sig';
```

112

### 25.4 simulator\_command()

<b>Purpose</b>	Issue a simulator command
<b>Category</b>	Predefined routine
<b>Syntax</b>	<b>simulator_command</b> ( <i>command</i> : string)
<b>Parameters</b>	<i>command</i> A valid simulator command, enclosed in double quotes (" "). <b>simulator_command()</b> cannot be used to pass commands that change the state of simulation, such as <i>run</i> , <i>restart</i> , <i>restore</i> , or <i>exit</i> (to the simulator).

This passes a command to the HDL simulator from *e*. The command shall not return a value. The output of the command is sent to the standard output and log file.

139

Syntax example:

```
simulator_command("force -deposit memA(31:0)");
```

## 25.5 stop\_run()

<b>Purpose</b>	Stop a simulation run cleanly
<b>Category</b>	Predefined routine
<b>Syntax</b>	stop_run()

This stops the simulator and initiates post-simulation phases. This method needs to be called by a user-defined method or TCM to stop the simulation run cleanly. The following things occur when **stop\_run()** is invoked:

- The **quit()** method of each struct under **sys** is called. Each **quit()** method emits a “quit” event for that struct instance at the end of the current tick.
- The scheduler continues running all threads until the end of the current tick.
- At the end of the current tick, the extract, check, and finalize test phases are performed.
- If a simulator is linked here, *e* terminates the simulation cleanly after the test is finalized.

Plus, these restrictions also apply.

- Executing a tick after calling **stop\_run()** shall be considered an error.
- If the simulator `exit` command is called before **stop\_run()**, the global methods for extracting, checking and finalizing the test are called.

NOTE—Use **sys.extract()** and extend that to make something happen right after stopping a run (rather than extending or modifying the **stop\_run()** method).

See also: 23.2.7 and the **run** option of 24.7.1.

Syntax example:

```
stop_run();
```