

## 14. Coverage constructs

This clause describes how to define, extend, and use coverage constructs. See also [6.2](#).

### 14.1 Defining coverage groups: `cover`

<b>Purpose</b>	Define a coverage group
<b>Category</b>	Struct member
<b>Syntax</b>	<b>cover</b> <i>event-type</i> [ <b>using</b> <i>coverage-group-option</i> , ...] <b>is</b> { <i>coverage-item-definition</i> ; ...} <b>cover</b> <i>event_type</i> <b>is empty</b>
<b>Parameters</b>	<i>event-type</i> The name of the group. This shall be the name of an event type defined previously in the struct. The event shall not have been defined in a subtype. However, if the event and coverage group are defined in the same file, the event definition does not have to appear before the coverage group definition. The event is the sampling event for the coverage group. Coverage data for the group is collected every time the event is emitted. The full name of the coverage group is <i>struct-exp.event-name</i> . The full name shall be specified for the coverage methods.
	<i>coverage-group-option</i> The coverage group options listed in <a href="#">Table 31</a> can be set via the <b>using</b> keyword. Each coverage group can have its own set of options. The options can appear in any order after the <b>using</b> keyword.
	<i>coverage-item-definition</i> The definition of a coverage item (see <a href="#">14.2</a> ).

This defines a coverage group. A *coverage group* is a struct member that contains a list of data items for which data is collected over time. Once data coverage items have been defined in a coverage group, they can be used to define special coverage group items called **transition** and **cross** items (see [14.4](#) and [14.3](#), respectively). The **is** keyword can also be used to define a new coverage group (see [14.5](#) for information on using **is also** to extend an existing coverage group).

The sampling event of a coverage group cannot be defined in a **when** subtype of a struct. However, a coverage group that uses an event defined in a parent struct can be defined in a **when** subtype. If they are extended by adding a **per\_instance** item, the instances refer only to the **when** subtype. If a **per\_instance** item is instead defined in a base type and additional items are added under the **when** construct, then the cover group instances refer to the base type and the cover item values refer to the **when** subtype (see [14.2.1](#)).

The **empty** keyword can be used to define an empty coverage group that will be extended later by using a **cover is also** struct member with the same name (see [14.5](#)). See also [Clause 10](#) and [Clause 6](#).

Table 31—Coverage group options

Option	Description
<b>no_collect</b>	This coverage group is not displayed in coverage reports and is not saved in the coverage files.
<b>count_only</b>	This option reduces memory consumption because the data collected for this coverage group is reduced. Interactive, post-processing cross coverage of items is not allowed in this case. The coverage configuration option <b>count_only</b> (see 28.8) sets this option for all coverage groups.
<b>text=string</b>	A text description for this coverage group. This can only be a quoted string (" "), not a variable or expression. The text is shown at the beginning of the information for the group in the coverage report.
<b>when=bool-exp</b>	The coverage group is sampled only when <i>bool-exp</i> is TRUE. The <i>bool-exp</i> is evaluated in the context of the parent struct, i.e., the scope of entities is taken from that context. Concretely, names of fields shall be taken as fields of the parent structs and methods called shall be executed in the context of said struct. The pseudo-variable <b>me</b> refers to that struct.
<b>global</b>	A <i>global coverage group</i> is a group whose sampling event is expected to be emitted only once. If more than one sampling event is emitted, a DUT error shall be issued. If items from a global group are used in interactive cross coverage, no timing relationships exist between the items.
<b>radix=DEC   HEX   BIN</b>	Buckets for items of type <b>int</b> or <b>uint</b> are given the item value ranges as names. This option specifies in which radix the bucket names are displayed. The global <b>print</b> radix option does not affect the bucket name radix. Legal values are <b>DEC</b> (decimal), <b>HEX</b> (hexadecimal), and <b>BIN</b> (binary). The value shall be in upper case letters. If the <b>radix</b> is not used, <b>int</b> or <b>uint</b> bucket names are displayed in decimal.
<b>weight=uint</b>	This option specifies the grading weight of the current group relative to other groups. It is a non-negative integer with a default of 1.

NOTE—Unless coverage mode is turned on first (see 28.8), no coverage results are collected, even if cover groups and cover items are defined.

Syntax example:

```
cover inst_driven is {
    item opcode;
    item op1;
    cross opcode, op1
}
```

14.2 Defining basic coverage items: *item*

<b>Purpose</b>	Define a coverage item	1
<b>Category</b>	Coverage group item	5
<b>Syntax</b>	<b>item</b> <i>item-name</i> [: <i>type=exp</i> ] [ <b>using</b> <i>coverage-item-option</i> , ...]	10
<b>Parameters</b>	<i>item-name</i> The name assigned to the coverage item. If the optional <i>type=exp</i> is not used, the value of the field named <i>item-name</i> is used as the coverage sample value. The field can be a scalar, not larger than 32 bits, or a string. If <i>type=exp</i> is specified, the value of <i>exp</i> is used as the coverage sample value.	15
	<i>type</i> The type of the item. The type expression shall evaluate to a scalar not larger than 32 bits or to a string.	20
	<i>exp</i> The expression is evaluated at the time the whole coverage group is sampled. If the <b>using when</b> option is used, expression evaluation is further restricted by the <b>when</b> expression evaluating to TRUE. The value of <i>exp</i> is recorded for the item.	25
	<i>coverage-item-option</i> The coverage group options listed in Table 32 can be set via the <b>using</b> keyword. The options can appear in any order after the <b>using</b> keyword.	

Table 32—Coverage item options

Option	Description	30
<b>per_instance</b>	Coverage data is collected and graded for all the other items in a separate listing for each bucket of this item. See also <a href="#">14.2.1</a> and <a href="#">14.2.2</a> .	
<b>no_collect</b>	This coverage item is not displayed in coverage reports and is not saved in the coverage files. However, it can be referenced by cross and transition items.	35
<b>text=string</b>	A text description for this coverage item. This can only be a quoted string (" "), not a variable or expression. In an ASCII coverage report, the text is shown along with the item name.	
<b>when=bool-exp</b>	The item is sampled only when <i>bool-exp</i> is TRUE. The <i>bool-exp</i> is evaluated in the context of the parent struct, i.e., the scope of entities is taken from that context. Concretely, names of fields shall be taken as fields of the parent structs and methods called shall be executed in the context of said struct. The pseudo-variable <b>me</b> refers to that struct. The sampling is done at runtime.	40
<b>at_least=num</b>	The minimum number of samples for each bucket of the item. Anything less than <i>num</i> is considered a hole. This option cannot be used for an <i>ungradeable item</i> , an item whose number of buckets exceeds the configuration option <b>max_int_buckets</b> . This shall be a non-negative number; the default is 1.	45

Table 32—Coverage item options (Continued)

Option	Description
<p><b>ranges</b> =  <b>{range(parameters)</b>  <b>[: range(parameters) ...]}</b></p>	<p>This option creates buckets for this item's values or ranges of values. It cannot be used for string items.</p> <p><b>range()</b> can have one, two, three, or four parameters that specify how the values are separated into buckets. The first parameter, <i>range</i>, is required. The other three are optional. The syntax for range options is:</p> <p style="text-align: center;"><b>range</b>(<i>range</i>: range, [<i>name</i>: string, <i>every-count</i>: int, <i>at-least-num</i>: int])</p> <p>The parameters are:</p> <ul style="list-style-type: none"> <li>— <i>range</i>  The range for the bucket. It shall be a literal range, such as [1 . . 5], of the proper type. Even a single value needs to be specified in brackets, e.g., [7]. If overlapping ranges are specified, the values of the overlapping region go into the first of the overlapping buckets. The specified range for a bucket is the bucket name.</li> <li>— <i>name</i>  A name for the bucket. If this parameter is used, the <i>every-count</i> parameter shall be set to UNDEF.</li> <li>— <i>every-count</i>  The size of the buckets to create within the range. If this parameter is used, the <i>name</i> parameter shall be set to an empty string ("").</li> <li>— <i>at-least-num</i>  A number that specifies the minimum number of samples required for a bucket. If the item occurs fewer times than this, a hole is marked. This parameter overrides the global <b>at_least</b> option and the per-item <b>at_least</b> option. The value of <i>at-least-num</i> can be set to zero (0), meaning "do not show holes for this range."</li> </ul>
<p><b>ignore</b>=<i>item-bool-exp</i></p>	<p>Defines values that are to be completely ignored. They do not appear in the statistics at all. The expression is a Boolean expression that can contain a coverage item name and constants.</p> <p>The Boolean expression is evaluated in a global context, not in instances of the struct, i.e., the expression shall be valid at all times, even before generation. Therefore, only constants and the item itself can be used in the expression. For example, if <i>i</i> is a coverage item and <i>j</i> is a reference to a struct field, the expression <i>i</i> &gt; 5 is a valid expression, but not <i>i</i> &gt; me.j.</p> <p>If the <b>ignore</b> expression is TRUE when the data is sampled, the sampled value is ignored (not added to the bucket count).</p> <p>To achieve the first effect (ignore specific samples) without hiding buckets containing holes (and to have the grade reflect all generated values), use the <b>when</b> option instead.</p>
<p><b>illegal</b>=<i>item-bool-exp</i></p>	<p>Defines values that are illegal. An illegal value shall cause a DUT error. If the <b>check_illegal_immediately</b> coverage configuration option is FALSE, the DUT error occurs during the <b>check_test</b> phase of the test. If that configuration option is TRUE, the DUT error occurs immediately (on-the-fly).</p> <p>The Boolean expression is evaluated in a global context, not in instances of the struct, i.e., the expression shall be valid at all times, even before generation. Therefore, only constants and the item itself can be used in the expression. For example, if <i>i</i> is a coverage item and <i>j</i> is a reference to a struct field, the expression <i>i</i> &gt; 5 is a valid expression, but not <i>i</i> &gt; me.j.</p> <p>If the coverage grades need to reflect all bucket contents, use the <b>when</b> option instead to specify the circumstances under which a given value is counted.</p>

Table 32—Coverage item options (Continued)

Option	Description
<b>radix=DEC   HEX   BIN</b>	<p>For items of type <b>int</b> or <b>uint</b>, this specifies the radix used in coverage reports for implicit buckets. If the <b>ranges</b> option is not used to create explicit buckets for an item, a bucket is created for every value of the item that occurs in the test. Each different value sampled gets its own bucket, with the value as the name of the bucket. These are called <i>implicit buckets</i>. The global <b>print</b> radix option does not affect the bucket name radix.</p> <p>Legal values are <b>DEC</b> (decimal), <b>HEX</b> (hexadecimal), and <b>BIN</b> (binary). The value shall be in upper case letters.</p> <p>If the <b>radix</b> is not used, <b>int</b> or <b>uint</b> bucket names are displayed in decimal. If no radix is specified for an item, but a radix is specified for the item's group, the group's radix applies to the item.</p>
<b>weight=uint</b>	<p>Specifies the weight of the current item relative to other items in the same coverage group. It is a non-negative integer with a default of 1.</p>

This defines a new basic coverage item with an optional type. Options specify how coverage data is collected and reported for the item. The item can be an existing field name or a new name. If a new name is used for a coverage item, the item's type and the expression that defines it shall also be specified.

If a value for an item falls outside all of the buckets for the item, that value does not count toward the item's grade. The **ranges** option determines the number and size of buckets into which values for the item are placed. If **ranges** is not specified, the default number of buckets is 16 [set by the **max\_int\_buckets** coverage configuration option (see 28.8)]. If buckets are not created for all possible values of the item, the values for which buckets do not exist are ungradeable. Those values are given goals of 0 and do not affect the grade for the item.

For example, a randomly generated item of type **uint** has  $2^{32}$  possible values. If no ranges are specified for a **uint** item, then buckets are created by default for only the first 16 possible values (0 through 15). Since the odds that a **uint** value will be less than 16 are very small, it is almost certain that none of the values will fall into one of the 0 to 15 buckets, which are the only buckets for which a grade is calculated. This means that the item does not receive a grade or contribute to the grade for the group.

NOTE—Unless coverage mode is turned on first (see 28.8), no coverage results are collected, even if cover groups and cover items are defined.

Syntax example:

```
cover inst_driven is {
    item op1;
    item op2;
    item op2_big : bool = (op2 >= 64);
    item hdl_sig : int = 'top.sig_1'
}
```

#### 14.2.1 Coverage per instance

The coverage per instance feature (the **per\_instance** option) enables the collection of coverage information for separate instances of structs or units, and the verification of the coverage data and grade associated with each particular instance.

When the **per\_instance** option is set in a cover item definition, that item becomes a “**per\_instance** item.” Each bucket of that item gets its own coverage grade and is shown separately in the coverage report. An instance is created for every valid bucket of the **per\_instance** item. Any instance that is not sampled is

1 marked as a hole. For example, if a struct has a field named `packet_type` and the value of the  
 5 `packet_type` field can be either Ethernet or ATM, then making that field a **per\_instance** item results  
 in a grade and a coverage report listing for Ethernet instances and a separate grade and coverage report  
 listing for ATM instances.

Along with the **per\_instance** item data, coverage data is also collected for the original **per\_type** item as if it  
 were not a **per\_instance** item. This coverage data for the **per\_type** item is the accumulated information for  
 all the instances, using the coverage options defined for the item.

10 Grading is calculated for each instance separately. The grade of the cover group is the weighted grades of all  
 the **per\_instance** items. The **per\_type** item receives the same grade it would get if there were no  
**per\_instance** items.

15 An instance item name is the name of the **per\_type** item followed by `==` and the name of the instance  
 bucket. For example, the instance item names for the case above are:

```

  packet_type == Ethernet
  packet_type == ATM
  
```

20 The following considerations also apply:

- For integer instances, the *decimal radix* is used regardless of what the radix is for the cover group.
- More than one **per\_instance** item can be defined in the same cover group. In this case, the total number  
 25 of instances is the sum of all valid buckets for all the **per\_instance** items +1 (the **per\_type**  
 bucket).
- If a **per\_instance** item definition is changed in an extension, then the coverage data for the original  
**per\_type** item might not accurately reflect nor agree with the coverage data collected per instance.
- A **per\_instance** item cannot be defined under a specific instance.
- Items with the same name can be defined under two different instances, as long as they have the  
 30 same definition (type and expression).
- If a **per\_instance** item is participating in a cross item or a transition item, then the cross or transition  
 item is not added to the instances created by the **per\_instance** item.
- To cancel per instance coverage collection in an extension, use the **also per\_instance = FALSE**  
 35 option.

#### 14.2.2 per\_instance item errors

40 [Table 33](#) lists errors that can occur when coverage per instance is used.

1

**Table 33—Coverage per instance errors**

5

Error	Description
Using a non-gradeable item as a <b>per_instance</b> item	When the user defines a <b>per_instance</b> item option for a non-gradeable item, a runtime error shall be issued.
Using a cross or transition item as a <b>per_instance</b> item	When the user defines a <b>per_instance</b> item for a cross item or a transition item, a loadtime error shall be issued.
Trying to extend an invalid instance	When the user tries to extend (using <b>cover ... is also</b> ) a group instance that does not exist, a loadtime error shall be issued.
Recursively split instances	When the user defines a <b>per_instance</b> item option for an instance group extension, a loadtime error shall be issued.
Trying to extend specific instances without using <b>is also</b>	When the user tries to extend a specific group instance using <b>is</b> instead of <b>is also</b> .
Trying to define multiple items with the same name but different definitions under different instances	When the user tries to define an item with the same name under two different instances.
Specifying an invalid instance name	When the user specifies an invalid instance name (possibly by using wild cards). No matching instance is found and the command is ignored.

10

15

20

25

**14.3 Defining cross coverage items: cross**

30

<b>Purpose</b>	Define a cross coverage item
<b>Category</b>	Coverage group item
<b>Syntax</b>	<b>cross</b> <i>item-name-1, item-name-2, ...</i> [ <b>using</b> <i>coverage-item-option, ...</i> ]
<b>Parameters</b>	<i>item-name-1, item-name-2</i> Each item name shall be one of the following: <ul style="list-style-type: none"> <li>— The name of an item defined previously in the current coverage group.</li> <li>— The name of a transition item defined previously in the current coverage group</li> <li>— The name of a cross item defined previously in the current coverage group.</li> </ul>
	<i>coverage-item-option</i> An option for the cross item (see <a href="#">Table 34</a> ).

35

40

45

50

55

Table 34—Cross coverage item options

Option	Description
<b>name</b> = <i>label</i>	Specifies a name for a cross coverage item. No white spaces are allowed in the label. The default is <code>cross__item-name-1__item-name-2</code> .
<b>text</b> = <i>string</i>	A text description for this coverage item. This can only be a quoted string (" "), not a variable or expression. The text is shown along with the item name at the top of the coverage information for the item.
<b>when</b> = <i>bool-exp</i>	The item is sampled only when <i>bool-exp</i> is TRUE. The <i>bool-exp</i> is evaluated in the context of the parent struct, i.e., the scope of entities is taken from that context. Concretely, names of fields shall be taken as fields of the parent structs, and methods called shall be executed in the context of said struct. The pseudo-variable <b>me</b> refers to that struct.
<b>no_collect</b>	This cross coverage item is not displayed in coverage reports and is not saved in the coverage files. However, it can be referenced by other cross and transition items.
<b>at_least</b> = <i>num</i>	The minimum number of samples for each bucket of the item. Anything less than <i>num</i> is considered a hole. This option cannot be used with string items or for unconstrained integer items (items that do not have specified ranges). This shall be a non-negative number; the default is 1.
<b>ignore</b> = <i>item-bool-exp</i>	Defines values that are to be completely ignored. They do not appear in the statistics at all. The expression is a Boolean expression that can contain a coverage item name and constants. The Boolean expression is evaluated in a global context, not in instances of the struct, i.e., the expression shall be valid at all times, even before generation. Therefore, only constants and the item itself can be used in the expression. In a cross, this means any of the participating items. In a transition, it means the item or <b>prev__item</b> . For example, if <i>i</i> is a coverage item and <i>j</i> is a reference to a struct field, the expression <code>i &gt; 5</code> is a valid expression, but not <code>i &gt; me.j</code> . If the <b>ignore</b> expression is TRUE when the data is sampled, the sampled value is ignored (not added to the bucket count). To achieve the first effect (ignore specific samples) without hiding buckets containing holes (and to have the grade reflect all generated values), use the <b>when</b> option instead.
<b>illegal</b> = <i>item-bool-exp</i>	Defines values that are illegal. An illegal value shall cause a DUT error. If the <b>check_illegal_immediately</b> coverage configuration option is FALSE, the DUT error occurs during the <b>check_test</b> phase of the test. If that configuration option is TRUE, the DUT error occurs immediately (on-the-fly). The Boolean expression is evaluated in a global context, not in instances of the struct, i.e., the expression shall be valid at all times, even before generation. Therefore, only constants and the item itself can be used in the expression. In a cross, this means any of the participating items. For example, if <i>i</i> is a coverage item and <i>j</i> is a reference to a struct field, the expression <code>i &gt; 5</code> is a valid expression, but not <code>i &gt; me.j</code> . If the coverage grades need to reflect all bucket contents, use the <b>when</b> option instead to specify the circumstances under which a given value is counted.
<b>weight</b> = <i>uint</i>	Specifies the weight of the current cross item relative to other items in the same coverage group. It is a non-negative integer with a default of 1.

This defines cross coverage between items in the same coverage group. It creates a new item with a name specified using a **name** option or by using the default name of `cross__item-name-1__item-name-2` (with two underscores separating the parts of the name). This shows every combination of values of the first and second items, and every combination of the third item and the first item, the third item and the second item,

and so on. Any combination of basic coverage items, cross items, and transitions defined in the same coverage group can be crossed. 1

The **using when**, **using ignore**, and **using illegal** options of the constituent items restrict the sampled values for a cross item. For example, in the `inst_driven` coverage group below, if `item opcode` includes the option `using ignore = (opcode != ADD)`, the cross coverage item `cross__opcode__op1` would exclude all buckets with the `opcode` value `ADD`. 5

Syntax example: 10

```
cover inst_driven is {
    item opcode;
    item op1;
    cross opcode, op1
}
```

 15

#### 14.4 Defining transition coverage items: transition 20

<b>Purpose</b>	Define a coverage transition item	20
<b>Category</b>	Coverage group item	
<b>Syntax</b>	<b>transition</b> <i>item-name</i> [ <b>using</b> <i>coverage-item-option</i> , ...]	25
<b>Parameters</b>	<i>item-name</i> A coverage item defined previously in the current coverage group.	
	<i>coverage-item-option</i> An option for the transition item (see <a href="#">Table 35</a> ).	30

This defines coverage for changes from one value to another of a coverage item. If no name is specified for the transition item with the **name** option, it gets a default name of `transition__item-name` (with two underscores between `transition` and `item-name`). If `item-name` had  $n$  samples during the test, then the transition item has  $n-1$  samples, where each sample has the format `previous-value, value`. 35

Syntax example:

```
cover state_change is {
    item st : cpu_state = 'top.cpu.main_cur_state';
    transition st
}
```

 40

45

50

55

Table 35—Transition coverage item options

Option	Description
<b>name</b> =string	Specifies a name for a transition coverage item. The default name is <code>transition__item-name</code> (where two underscores separate <code>transition</code> and <code>item-name</code> ).
<b>text</b> =string	A text description for this coverage item. This can only be a quoted string (" "), not a variable or expression. The text is shown along with the item name at the top of the coverage information for the item.
<b>no_collect</b>	This transition coverage item is not displayed in coverage reports and is not saved in the coverage files. However, it can be referenced by other cross and transition items.
<b>when</b> =bool-exp	The item is sampled only when <i>bool-exp</i> is TRUE. The <i>bool-exp</i> is evaluated in the context of the parent struct, i.e., the scope of entities is taken from that context. Concretely, names of fields shall be taken as fields of the parent structs and methods called shall be executed in the context of said struct. The pseudo-variable <b>me</b> refers to that struct.
<b>at_least</b> =num	The minimum number of samples for each bucket of each of the transition items. Anything less than <i>num</i> is considered a hole. This option cannot be used with string items or for unconstrained integer items (items that do not have specified ranges). This shall be a non-negative number; the default is 1.
<b>ignore</b> =item-bool-exp	Define values that are to be completely ignored. They do not appear in the statistics at all. The expression is a Boolean expression that can contain a coverage item name and constants. The previous value can be accessed as <code>prev_item-name</code> . The <b>prev</b> prefix is predefined for this purpose. The Boolean expression is evaluated in a global context, not in instances of the struct, i.e., the expression shall be valid at all times, even before generation. Therefore, only constants and the item itself can be used in the expression. In a cross, this means any of the participating items. In a transition, it means the item or <code>prev_item</code> . For example, if <i>i</i> is a coverage item and <i>j</i> is a reference to a struct field, the expression <code>i &gt; 5</code> is a valid expression, but not <code>i &gt; me.j</code> . If the <b>ignore</b> expression is TRUE when the data is sampled, the sampled value is ignored (not added to the bucket count). To achieve the first effect (ignore specific samples) without hiding buckets containing holes (and to have the grade reflect all generated values), use the <b>when</b> option instead.
<b>illegal</b> =item-bool-exp	Define values that are illegal. An illegal value shall cause a DUT error. If the <b>check_illegal_immediately</b> coverage configuration option is FALSE, the DUT error occurs during the <b>check_test</b> phase of the test. If that configuration option is TRUE, the DUT error occurs immediately (on-the-fly). The Boolean expression is evaluated in a global context, not in instances of the struct, i.e., the expression shall be valid at all times, even before generation. Therefore, only constants and the item itself can be used in the expression. In a transition, this means any of the participating items. For example, if <i>i</i> is a coverage item and <i>j</i> is a reference to a struct field, the expression <code>i &gt; 5</code> is a valid expression, but not <code>i &gt; me.j</code> . If the coverage grades need to reflect all bucket contents, use the <b>when</b> option instead to specify the circumstances under which a given value is counted.
<b>weight</b> =uint	Specifies the weight of the current transition item relative to other items in the same coverage group. It is a non-negative integer with a default of 1.

## 14.5 Extending coverage groups: `cover ... using also ... is also`

<b>Purpose</b>	Extend a coverage group	
<b>Category</b>	Struct member	
<b>Syntax</b>	<b>cover</b> <i>event-type</i> <b>using also</b> <i>cover-option</i> , ...[ <b>is also</b> { <i>coverage-item-definition</i> ; ... } ]	
<b>Parameters</b>	<i>event-type</i>	The name of the coverage group. This shall be an event defined previously in the struct. The event is the sampling event for the coverage group.
	<i>cover-option</i>	The definition for this option is shown in <a href="#">Table 32</a> .
	<i>coverage-item-definition</i>	The definition of a coverage item (see <a href="#">Table 31</a> ).

The **using also** clause changes, overrides, or extends options previously defined for the coverage group. The **is also** clause adds new items to a previously defined coverage group or it can be used to change the options for previously defined items (see [14.6](#)).

The following considerations also apply:

- If a coverage group is defined under a **when** subtype, it can only be extended under that subtype.
- If **per\_instance** coverage is being used (see [14.2.1](#)), a particular cover group instance can be extended to complement or override options set in the base type cover group. To change an item's options in a particular instance, enter the instance name in the **cover is also** construct.
- If an instance is extended, but it never gets created (due to an **ignore** or **illegal** option), a warning is issued and no information for the extension is put in the coverage data.
- If the coverage options of an instance are changed, the coverage data for the **per\_type** item might no longer reflect or agree with the **per\_instance** coverage data.
- If, in an extension of a cover group, a cover group **when** option is overridden, then the overriding condition is only considered after the condition in the base group is satisfied, i.e., sampling of the item is only performed when the logical AND of the cover group **when** options is TRUE.
- When **using also** is used to extend or change a **when**, **illegal**, or **ignore** option, a special variable named **prev** is automatically created. The **prev** variable holds the results of all previous **when**, **illegal**, or **ignore** options, so it can be used as a shorthand to assert those previous options combined with a new option value.

Syntax examples:

```
cover rclk is also {
    item rflag
};

cover rclk using also text = "RX clock";

cover rclk using also no_collect is also {
    item rvalue
}
```

## 14.6 Extending coverage items: item ... using also

<b>Purpose</b>	Change or extend the options on a cover item	
<b>Category</b>	Coverage group item	
<b>Syntax</b>	<b>item</b> <i>item-name</i> <b>using also</b> <i>coverage-item-option</i> , ...	
<b>Parameters</b>	<i>item-name</i>	The name assigned to the coverage item. If the optional <i>type=exp</i> is not used, the value of the field named <i>item-name</i> is used as the coverage sample value.
	<i>coverage-item-option</i>	The coverage item options are listed in <a href="#">Table 32</a> . The options can appear in any order after the <b>using</b> keyword.

Cover item extensibility enables extending, changing, or overriding a previously defined coverage item. To extend a coverage item, see [14.5](#).

If a coverage item is originally defined under a **when** subtype, it can only be extended in the same subtype of the base type.

When **using also** is used to extend or change a **when**, **illegal**, or **ignore** option, a special variable named **prev** is automatically created. The **prev** variable holds the results of all previous **when**, **illegal**, or **ignore** options, so it can be used as a shorthand to assert those previous options combined with a new option value.

Once an item is extended, it shall be referenced using its full name. If an item with that name does not exist, an error shall be issued.

Syntax example:

```
item len using also radix = HEX
```

## 14.7 Coverage API

This subclause contains descriptions of the *e* coverage API.

### 14.7.1 user\_cover\_struct

The coverage API is accessed through a set of methods defined in the struct **user\_cover\_struct**. Once an instance of this struct is declared, the **scan\_cover()** method may be invoked which, in turn, invokes accessory methods, such as **start\_group()** and **start\_instance()**.

### 14.7.2 package\_name

<b>Purpose</b>	Hold the name of the package in which the struct defining the items was declared
<b>Category</b>	Field
<b>Syntax</b>	<b>package_name</b> : string

The **package\_name** method holds the name of the package in which the output type was declared.

Syntax example:

```
package_name : "Turbo Mode Package"
```

### 14.7.3 Coverage API methods

This subclause contains descriptions of the *e* coverage predefined API methods.

#### 14.7.3.1 scan\_cover()

<b>Purpose</b>	Activate the coverage API and specify items to cover
<b>Category</b>	Method
<b>Syntax</b>	<b>scan_cover</b> ( <i>item-names</i> :string): int
<b>Parameters</b>	<i>item-names</i> The names of the coverage items to be scanned by <b>scan_cover()</b> . This is a string of the form <i>struct-name.group-name.item-name</i> (for example, "inst.start.opcode"). Wild cards are allowed.

The **scan\_cover()** method initiates the coverage data-scanning process. It goes through all the items in all the groups specified in the *item-names* parameter in the order that groups and items have been defined. This method cannot be extended.

For each group, **scan\_cover()** calls **start\_group()**. For each instance in the group, **scan\_cover()** calls **start\_instance()**. For each item in the current instance, **scan\_cover()** calls **start\_item()**. Then for each bucket of the item, **scan\_cover()** calls **scan\_bucket()**. After all of the buckets of the item have been processed, **scan\_cover()** calls **end\_item()**. After all items of the instance have been processed, **scan\_cover()** calls **end\_instance()**. After all instances in the group have been processed, **scan\_cover()** calls **end\_group()**.

Before each call to any of the preceding methods, the relevant fields in the **user\_cover\_struct** are updated to reflect the current item (and also the current bucket for **scan\_bucket()**).

The **scan\_cover()** method returns the number of coverage items actually scanned.

NOTE—The methods called by **scan\_cover()**: **start\_group()**, **start\_instance()**, **start\_item()**, **scan\_bucket()**, **end\_item()**, **end\_instance()**, and **end\_group()** are initially empty and meant to be extended.

Syntax example:

```
num_items = cover_info.scan_cover("cpu.inst_driven.*")
```

#### 14.7.3.2 start\_group()

<b>Purpose</b>	Process coverage group information according to user preferences
<b>Category</b>	Method
<b>Syntax</b>	<b>start_group()</b>

When the **scan\_cover()** method initiates the coverage data-scanning process for a group, it updates the group-related fields within the containing **user\_cover\_struct** and then calls the **start\_group()** method. The

**start\_group()** method is called for every group to be processed by **scan\_cover()**. For every instance within a group, **scan\_cover()** calls the **start\_instance()** method.

The **start\_group()** method is originally empty. It is meant to be extended to process group data according to user preferences.

NOTE—**start\_group()**, **start\_instance()**, and **scan\_cover()** are all methods of the **user\_cover\_struct**.

Syntax example:

```
start_group() is {
    if group_text != NULL then {
        out("Description: ", group_text)
    }
}
```

#### 14.7.3.3 start\_instance()

<b>Purpose</b>	Process coverage instance information according to user preferences
<b>Category</b>	Method
<b>Syntax</b>	<b>start_instance()</b>

When the **scan\_cover()** method initiates the coverage data-scanning process for an instance, it updates the instance-related fields within the containing **user\_cover\_struct** and then calls the **start\_instance()** method. The **start\_instance()** method is called for every instance to be processed by **scan\_cover()**.

The **start\_instance()** method is originally empty. It is meant to be extended to process instance data according to user preferences.

NOTE—**start\_instance()** and **scan\_cover()** are methods of the **user\_cover\_struct**.

Syntax example:

```
start_instance() is {
    if instance_text != NULL then {
        out("Description: ", instance_text)
    }
}
```

#### 14.7.3.4 start\_item()

<b>Purpose</b>	Process coverage item information according to user preferences
<b>Category</b>	Method
<b>Syntax</b>	<b>start_item()</b>

When the **scan\_cover()** method initiates the coverage data-scanning process for an item, it updates the item-related fields within the containing **user\_cover\_struct** and then calls the **start\_item()** method. The **start\_item()** method is called for every item to be processed by **scan\_cover()**.

The **start\_item()** method is originally empty. It is meant to be extended to process item data according to user preferences. 1

NOTE—**start\_item()** and **scan\_cover()** are methods of the **user\_cover\_struct**. 5

Syntax example:

```
start_item() is {
    if item_text != NULL then {
        out("Description: ", item_text)
    }
}
```

#### 14.7.3.5 scan\_bucket()

<b>Purpose</b>	Process coverage item information according to user preferences
<b>Category</b>	Method
<b>Syntax</b>	<b>scan_bucket()</b>

When the **scan\_cover()** method processes coverage data, then for every bucket of the item, it updates the bucket-related fields within the containing **user\_cover\_struct** and calls **scan\_bucket()**. 20 25

The **scan\_bucket()** method is originally empty. It is meant to be extended to process bucket data according to user preferences.

NOTE—**scan\_bucket()** and **scan\_cover()** are methods of the **user\_cover\_struct**. 30

Syntax example:

```
scan_bucket() is {
    out(count, " ", percent, "% ", bucket_name)
}
```

#### 14.7.3.6 end\_item()

<b>Purpose</b>	Report end of item coverage information according to user preferences
<b>Category</b>	Method
<b>Syntax</b>	<b>end_item()</b>

When the **scan\_cover()** method completes the processing of coverage data for an item, it calls the **end\_item()** method to report the end of item information according to user preferences. When all items in the current group have been processed, **scan\_cover()** calls the **start\_instance()** method for the next instance. 40 50

The **end\_item()** method is originally empty. It is meant to be extended so as to process item data according to user preferences.

NOTE—**end\_item()**, **start\_instance()**, and **scan\_cover()** are all methods of the **user\_cover\_struct**. 55

1 Syntax example:

```

    end_item() is {
        out("finished item ", item_name, "\n")
5    }

```

#### 14.7.3.7 end\_instance()

<b>Purpose</b>	Process end of instance coverage information according to user preferences
<b>Category</b>	Method
<b>Syntax</b>	<b>end_instance()</b>

10 When the **scan\_cover()** method completes the processing of coverage data for an instance, it calls the **end\_instance()** method to report the end of instance information according to user preferences. When all instances in the current group have been processed, **scan\_cover()** calls the **start\_group()** method for the next group.

20 The **end\_instance()** method is originally empty. It is meant to be extended so as to process instance data according to user preferences.

25 NOTE—**end\_instance()**, **start\_group()**, and **scan\_cover()** are all methods of the **user\_cover\_struct**.

Syntax example:

```

    end_instance() is {
        out("finished instance ", instance_name, "\n")
30    }

```

#### 14.7.3.8 end\_group()

<b>Purpose</b>	Report end of group coverage information according to user preferences
<b>Category</b>	Method
<b>Syntax</b>	<b>end_group()</b>

35 When the **scan\_cover()** method completes the processing of coverage data for a group, it calls the **end\_group()** method to report the end of group information according to user preferences.

45 The **end\_group()** method is originally empty. It is meant to be extended so as to process item data according to user preferences.

50 NOTE—**end\_group()** and **scan\_cover()** are both methods of the **user\_cover\_struct**.

Syntax example:

```

    end_group() is {
        out("finished group", group_name, "\n")
55    }

```

## 14.8 Coverage methods for the covers struct

The **covers** struct is a predefined struct containing methods used for coverage and coverage grading. With the exception of the **write\_cover\_file()** method, all of the following methods are methods of the **covers** struct. See also [16.2.4](#).

### 14.8.1 include\_tests()

<b>Purpose</b>	Specify which to display test run coverage information
<b>Category</b>	Predefined method
<b>Syntax</b>	<b>covers.include_tests</b> ( <i>full-run-name</i> : string, <i>include-run</i> : bool)
<b>Parameters</b>	<i>full-run-name</i> The name of the test to include or exclude.
	<i>include-run</i> Set to TRUE to include the specified run, FALSE to exclude it.

This method specifies which test runs to use in showing coverage information. If `.ecov` files are being read to load coverage information, only call this method after the `.ecov` files have been read.

Syntax example:

```
covers.include_tests("tests_A:run_A_10", TRUE)
```

### 14.8.2 set\_weight()

<b>Purpose</b>	Specify the coverage grading weight of a group or item
<b>Category</b>	Predefined method
<b>Syntax</b>	<b>covers.set_weight</b> ( <i>entity-name</i> : string, <i>value</i> : int, <i>multiply-value</i> : bool)
<b>Parameters</b>	<i>entity-name</i> The group or item for which to set the weight. This can include wild cards.
	<i>value</i> The integer weight value to set.
	<i>multiply-value</i> When this is FALSE, it changes the weights of all matching groups or items to <i>value</i> . When this is TRUE, the weights of all matching groups or items are multiplied by <i>value</i> .

Coverage grading uses weights to emphasize the affect of particular groups or items relative to others. The weights can be specified in the coverage group or item definitions. This method sets the weights procedurally. It overrides the weights set in the group or item definitions. Weights can be set explicitly or multiplied by a given value.

If `.ecov` files are being read to load coverage information, only call this method after the `.ecov` files have been read.

Syntax example:

```
covers.set_weight("inst.done", 4, FALSE)
```

### 14.8.3 set\_at\_least()

<b>Purpose</b>	Set the minimum number of samples needed to fill a bucket	
<b>Category</b>	Predefined method	
<b>Syntax</b>	<b>covers.set_at_least</b> ( <i>entity-name</i> : string, <i>value</i> : int, <i>multiply-value</i> : bool)	
<b>Parameters</b>	<i>entity-name</i>	The group or item for which to set the “at-least” number. This can include wild cards.
	<i>value</i>	The at-least integer value to set.
	<i>multiply-value</i>	When this is FALSE, it changes the at-least number of all matching items to <i>value</i> . When this is TRUE, it multiplies the at-least number by <i>value</i> .

The minimum number of samples required to fill a bucket can be set in the coverage group or item definitions. This method can be used to set the number procedurally. It overrides the numbers set in the group or item definitions. If the *entity-name* is a coverage group name, all items in the group are affected. If the *entity-name* matches items within a coverage group, only those items are affected.

If .ecov files are being read to load coverage information, only call this method after the .ecov files have been read.

Syntax example:

```
covers.set_at_least("inst.done", 4, FALSE)
```

### 14.8.4 set\_cover()

<b>Purpose</b>	Turn coverage data collection and display on or off for specified items or events	
<b>Category</b>	Predefined method	
<b>Syntax</b>	<b>covers.set_cover</b> (( <i>item</i>   <i>event</i> ): string, <i>collect-coverage</i> : exp)	
<b>Parameters</b>	<i>item</i>	A string, enclosed in double quotes (" "), specifying the coverage item to turn on or off. This can include wild cards.
	<i>event</i>	A string, enclosed in double quotes (" "), specifying the event to turn on or off. This can include wild cards. Enter the name of the event using the following syntax: <b>session.events.struct_type__event_name</b> where <i>struct_type</i> and <i>event_name</i> are separated by two underscores. Wild cards can also be used here. If only one name is specified, it is treated as a struct type and the method shall affect all events in that struct type.
	<i>collect-coverage</i>	Set to TRUE to turn on coverage for the item or FALSE to turn coverage off.

By default, coverage data is collected for all defined coverage items and groups, and for all user-defined events. This method selectively turns data collection on or off for specified items, groups, or events. Although this method can be used to filter samples during periods in which they are not valid, for performance reasons, use **when** subtypes instead.

Additionally, if the test ends while coverage collection is turned off by `set_cover()` for one or more coverage groups, then `set_cover()` needs to be called again to re-enable sampling before the `.ecov` file is written, in order to include the previously collected samples for those groups in the `.ecov` file.

Syntax example:

```
covers.set_cover("packet.*", FALSE)
```

#### 14.8.5 `get_contributing_runs()`

<b>Purpose</b>	Return a list of the test runs that contributed samples to a bucket
<b>Category</b>	Predefined method
<b>Syntax</b>	<code>covers.get_contributing_runs(item-name: string, bucket-name: string)</code> : list of string
<b>Parameters</b>	<i>item-name</i> A string, enclosed in double quotes (" "), specifying the coverage item that contains <i>bucket-name</i> .
	<i>bucket-name</i> A string, enclosed in double quotes (" "), specifying the bucket for which contributing test run names are to be listed.

This method returns a list of strings that are the full run names of the test runs that placed samples in a specified bucket. For a cross item, the *bucket-name* can be a bucket of any level, with the bucket set names separated by slashes, e.g., `ADD/REG1/[0xC0..0xCF]`.

Syntax example:

```
bkl = covers.get_contributing_runs("inst.done.len", "[0..4]")
```

#### 14.8.6 `get_unique_buckets()`

<b>Purpose</b>	Return a list of the names of unique buckets from specific tests.
<b>Category</b>	Method
<b>Syntax</b>	<code>covers.get_unique_buckets(file-name: string)</code> : list of string
<b>Parameters</b>	<i>file-name</i> A string, enclosed in double quotes, (" ") specifying which coverage database files, containing unique buckets, to display. Wild cards cannot be used in the file name.

A *unique bucket* is a bucket that is covered by only one test. This method reports, for each specified test, the full names of its unique buckets, if there are any.

Syntax example:

```
print covers.get_unique_buckets("test_rx")
```

**14.8.7 write\_cover\_file()**

<b>Purpose</b>	Write the coverage results during a test
<b>Category</b>	Predefined method
<b>Syntax</b>	<b>write_cover_file()</b>

This method writes the coverage results `.ecov` file during a test run. It can only be invoked during a test, not before the run starts nor after it ends.

The coverage file written by this method does not contain the `session.end_of_test` or `session.events` coverage groups.

Syntax example:

```
write_cover_file()
```

**14.8.8 get\_overall\_grade()**

<b>Purpose</b>	Return the normalized overall coverage grade
<b>Category</b>	Predefined method
<b>Syntax</b>	<b>covers.get_overall_grade(): int</b>

This method returns an integer that represents the overall coverage grade for the current coverage results. Since `e` does not handle floating point types, the value is a normalized value between 1 and 100M. To obtain a value equivalent to the overall grade, divide the returned value by 100M.

Syntax example:

```
grade = covers.get_overall_grade()
```

**14.8.9 get\_ecov\_name()**

<b>Purpose</b>	Return the name of the <code>.ecov</code> file
<b>Category</b>	Predefined method
<b>Syntax</b>	<b>covers.get_ecov_name(): string</b>

This method returns the name of the `.ecov` file in which the current coverage results are to be stored.

Syntax example:

```
ecov_file = covers.get_ecov_name()
```

**14.8.10 get\_test\_name()**

1

<b>Purpose</b>	Return the name of the current test	
<b>Category</b>	Predefined method	5
<b>Syntax</b>	<b>covers.get_test_name()</b> : string	10

This method returns the identifier of the current test run.

Syntax example:

```
ecov_file = covers.get_test_name()
```

15

**14.8.11 get\_seed()**

<b>Purpose</b>	Return the value of the seed for the current test	20
<b>Category</b>	Predefined method	
<b>Syntax</b>	<b>covers.get_seed()</b> : int	25

This method returns the current test seed.

Syntax example:

```
seed_val= covers.get_seed()
```

30

35

40

45

50

55

1

5

10

15

20

25

30

35

40

45

50

55