cādence™

Incisive® Enterprise

Specman Elite Testbench®

The shr_chkcov Package

Version 1.0

Coverage of Checks and Expects

1 Introduction

shr_chkcov is a shareware that enables coverage collection of the ***e*** checks and expects in Specman 6.x. It is a new version of an existing package that worked on top of Specman 4.3.x, but, due to changes in the ***e*** infrastructure, was broken in Specman 5.x.

The shareware's capabilities provide a short-term solution, and a first step on the way to a complete assertion coverage solution that will be implemented once integration of the assertion coverage concept with our ***v***Planning approach is enhanced.

shr_chkcov allows you to collect and view coverage for each of the checks in the environment that use the built-in Specman checking constructs. This enables you to determine whether any of the checks have not been evaluated in the set of simulation runs that were used to create the currently loaded coverage database.

The user can specify that coverage is not to be collected or analyzed for all checks, or a subset of the checks, in the environment.

2 Coverage of checks and expects

There are two types of assertion checks in ***e***:

1. Temporal checks defined with the **expect** construct define the correct behavior over multiple cycles. The temporal expression defined with the **expect** construct is always expected to succeed. Temporal checks are struct members and are continuously active during a run.

   Example:

   ```
   expect TE1=>TE2
   ```

   defines a temporal rule that TE2 must succeed when evaluated starting one cycle after TE1.

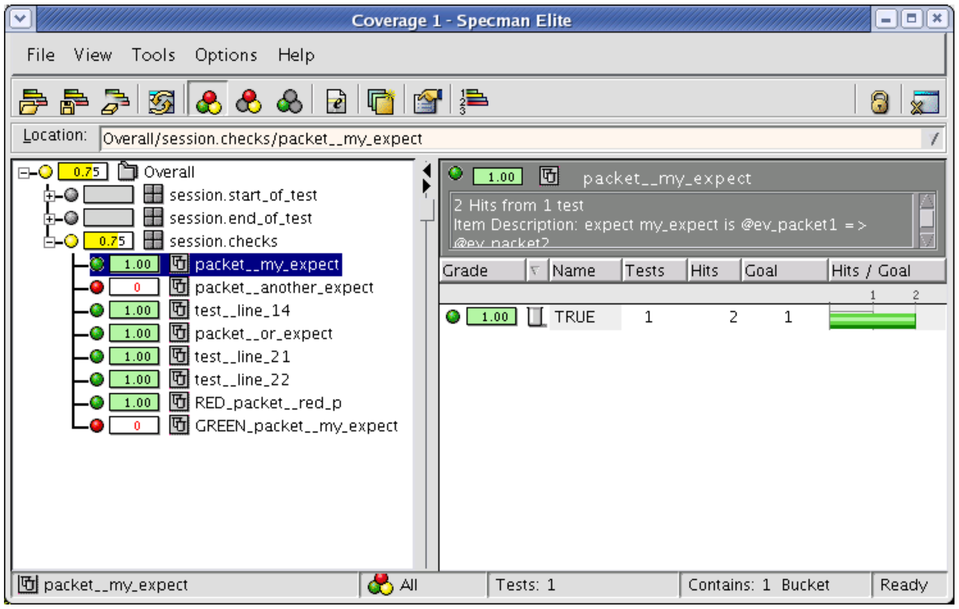
2. Data checks defined with the **check that** construct define the expected values at a particular point in time. Data checks are actions, and must be included in a method or a Time Consuming Method (TCM). They are activated only when the **check that** action is executed when the method is called.

   Example:

   ```
   check that 'data_out' == 'data_in';
   ```

Using the shr_chkcov shareware, a new cover group will be added under “session”, named “checks” (similar to the session.events cover group). This group will include all of the checks and expects in the design, each one of them represented as an item. Each item name is module-name__lineNumber (for example, test__line_20). Named expects will have their name used as the item name, in the following format: structName__checkName (for example, packet__my_expect). If a named check is defined in a when sub-type, the subtype name is used before the struct name (for example, GREEN_packet__my_expect).

Each item has only one bucket, named “TRUE”, indicating whether the check/expect was evaluated as described below, at least once, in one of the simulation runs in the coverage results database.



Location: Overall/session.checks/packet__my_expect

0.75	Overall
	session.start_of_test
	session.end_of_test
0.75	session.checks
1.00	packet__my_expect
0	packet__another_expect
1.00	test__line_14
1.00	packet__or_expect
1.00	test__line_21
1.00	test__line_22
1.00	RED_packet__red_p
0	GREEN_packet__my_expect

1.00 packet__my_expect

2 Hits from 1 test
Item Description: expect my_expect is @ev_packet1 =>
@ev_packet2

Grade	Name	Tests	Hits	Goal	Hits / Goal
1.00	TRUE	1	2	1	🟩

packet__my_expect | All | Tests: 1 | Contains: 1 Bucket | Ready

The semantics for the coverage is as follows:

- A check is covered if the **check that** action was executed.
- An expect is covered if the left hand side was evaluated to TRUE, and the '=>' was triggered. For example, the expect statement:

```
expect TE1=>TE2
```

will be considered as covered if TE1 was evaluated to TRUE.

The coverage is collected per-type only. The shr_chkcov shareware does not distinguish between different instances.

3 shr_chkcov Usage

3.1 Using shr_chkcov with Your Environment

To use the shr_chkcov shareware, you simply need to load/import the top file. It is important that the shareware is loaded before loading the user's VE containing the checks/expects. You also can compile the shareware on top of Specman and then use this new executable when loading/compiling your environment.

For example:

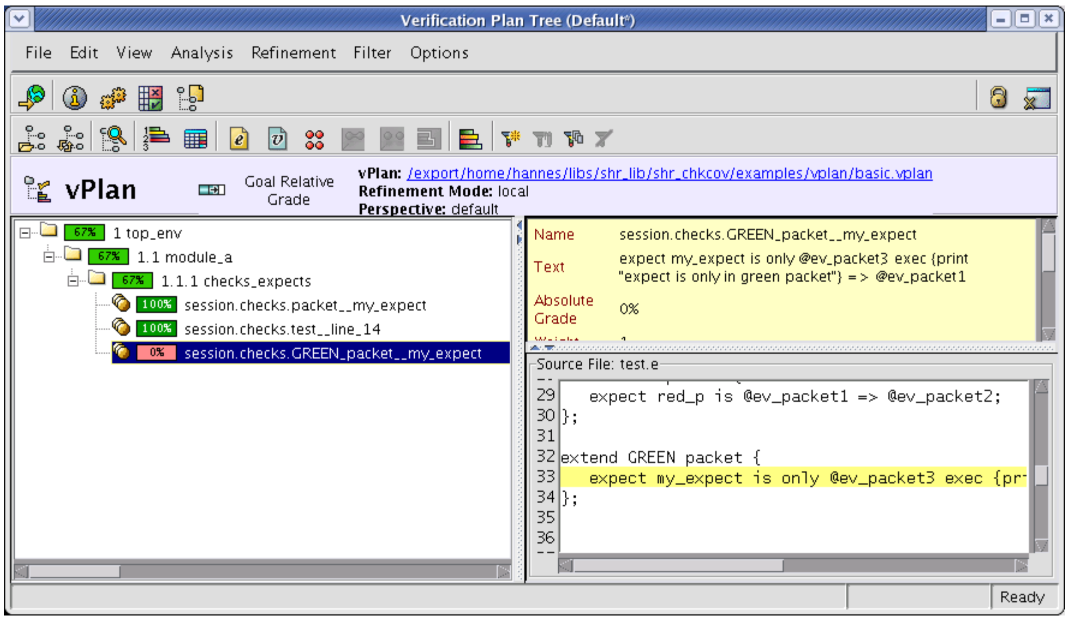
```
load shr_chkcov/e/shr_chkcov_top.e
load my_env/sve/xyz_config_top.e
```

3.2 Connecting Checks and Expects Coverage to a *v*Plan

To include the coverage points of checks and expects in your ***v***Plan, use the full path notation of the coverage database, according to the naming described above.

For example:

```
section top_env {
    section module_a {
        section checks_expects {
            coverage: session.checks.packet__my_expect;
            coverage: session.checks.test__line_14;
            coverage: session.checks.GREEN_packet__my_expect;
        }
    }
}
perspective default {
    top_section: top_env;
}
```


Verification Plan Tree (Default*)
File Edit View Analysis Refinement Filter Options
vPlan
Goal Relative Grade
Source File: test.e
Ready

vPlan: /export/home/hannes/libs/shr_lib/shr_chkcov/examples/vplan/basic.vplan
Refinement Mode: local
Perspective: default

67%	1 top_env
67%	1.1 module_a
67%	1.1.1 checks_expects
100%	session.checks.packet__my_expect
100%	session.checks.test__line_14
0%	session.checks.GREEN_packet__my_expect

Name	session.checks.GREEN_packet__my_expect
Text	expect my_expect is only @ev_packet3 exec {print "expect is only in green packet"} => @ev_packet1
Absolute Grade	0%

```
  expect red_p is @ev_packet1 => @ev_packet2;
};

extend GREEN packet {
  expect my_expect is only @ev_packet3 exec {pr
};
```

3.3 Configuring Coverage Buckets

It is possible to configure (set on/off coverage) of a specific check or expect, using the **set_cover()** method. The method supports wildcards as well.

For example:

```
extend sys {
init() is also {
    set_cover("session.checks.packet*",FALSE);
    set_cover("session.checks.packet__my_expect",TRUE);
    };
};
```

3.4 Limitations

Be aware of the following issues and limitations:

- Assertion coverage semantics is not tightly defined by its nature.
- This solution does not try to define the full semantics for assertion coverage (if such exists), but supplies only a minimal set of capabilities that were included in the previous shareware.

- Coverage of checks and expects is collected per-type only.
- Minimal support level will be supplied for the utility.
- Minimal testing has been done for the utility.
- The proposed solution might not be compatible with future implementations of an enhanced solution.
- Coverage is collected before the execution of the check, and the check will be considered as covered even if the check fails (see below). If the user changed the severity of the check to IGNORE, coverage is not the right indication for the check if it was successful or not.
- An expect is covered if the left hand side was evaluated to TRUE, and the '=>' was triggered, and we don't check that the right hand side was evaluated to TRUE as well. So for TE1=>TE2, if the simulation ends after TE1 succeeds but before TE2 succeeds, our coverage will record this as covered.