Specification Coverage - Quick Reference

UVVM Support VIP

The Specification Coverage feature (aka Requirements Coverage) is an efficient method for verifying the requirement specification.

Note: The first page of this QuickRef is just for syntax reference. For an introduction to the Specification Coverage concept, please see page 2.

1 UVVM VHDL Methods – see Table 2 for details.

Procedure	Parameters	Examples
initialize_req_cov()	testcase (string), req_list_file (string), partial_cov_file (string) testcase (string), partial_cov_file (string) ***(→ Note: No spec coverage)***	initialize_req_cov("t_base_func", "uart_req_list.csv", "base_func_cov.csv"); initialize_req_cov("t_base_func", "base_func_cov.csv");
register_req_cov()	requirement (string) [,testcase (string)] [,FAIL (t_test_status) [, msg [, scope]]]	register_req_cov("UART_REQ_4");
finalize_req_cov()	VOID	finalize_req_cov(VOID);

2 Script Usage – run_spec_cov.py

Call the run_spec_cov.py from a terminal, using e.g.: ***python run_spec_cov.py <see Table 6 for script arguments>***

3 File Formats – Basic usage

File	**Requirement list file ('req_list'), CSV**	**Partial coverage file ('part_cov'), CSV**
Info	*Input to testcase and run_spec_cov.py*	*Output from testcase, and input to run_spec_cov.py*
Layout	*"Requirement label", "Description" [, "Testcase"]*	*"Requirement", "Testcase", PASS | FAIL* *Preceded by header and succeeded by footer as shown below*
Example	FPGA_REQ_1, Baudrate 9k6, T_UART_1 FPGA_REQ_2, Baudrate 19k2, T_UART_1 etc.	*NOTE: <note>* *TESTCASE_NAME: <name>* *DELIMITER: <single delimiter character>* FPGA_REQ_1, T_UART_1, PASS FPGA_REQ_2, T_UART_2, FAIL etc. *SUMMARY, <testcase-name>, PASS|FAIL*

File	**Optional: Requirement map file ('req_map') , CSV**	**Optional: Partial coverage list file ('part_cov_list'), TXT**	**Optional: Script config file, TXT**
Info	*Optional input to run_spec_cov.py*	*Optional input to run_spec_cov.py*	*Optional input to run_spec_cov.py*
Layout	*Alt a) "Requirement label", "mapped req label" [, "mapped req label"]* *Alt b) "Requirement label", "sub-req label", "sub-req label", …*	*"path to a Partial coverage file"* *"path to another Partial coverage file"* *etc.*	*--argument value OR -a value* *etc.*
Example	FPGA_REQ_1, FPGA_REQ_1.a, FPGA_REQ_1.b	pc_base_func.csv ../my_sim_dir/pc_reset.csv cov_corner_cases.csv	--requirement_list path/requirement_list.csv -p partial_coverage_files.txt -s spec_cov.csv --strictness 1

NOTE: The CSV separator may be set to any separator character. Default is ',' but may be changed in VHDL (see ch 9.2). Delimiter is written to partial coverage file.

UVVM™

4 Definitions used in this document

4.1 Testcase

- A scenario or sequence of actions - controlled by the test sequencer.
- May test one or multiple features/requirements.
- Typically testing of related functionality, or a logical sequence of events, or an efficient sequence of events
- Important: The minimum sequence of events possible to run in a single simulation execution. Thus if there is an option to run one of multiple test sequences (A or B or C), a set of test sequences (A and B) or all sequences (A+B+C), then all of A, B and C are defined as individual testcases.

4.2 Specification coverage

- A summary of how all the requirements in a complete Requirement Specification have been covered by the test suite (the complete verification environment)

4.3 Partial coverage

- In this VIP a summary of how some (or all) requirements for a DUT have been covered by one specific testcase. There may be one or more testcases and partial coverage summaries for a DUT depending on complexity and approach.
- The accumulation (or merger) of all partial coverage summaries will yield the Specification coverage.
  I.e. Testcase partial specification coverage → 'Partial Coverage', and Test suite overall specification coverage → 'Specification coverage'

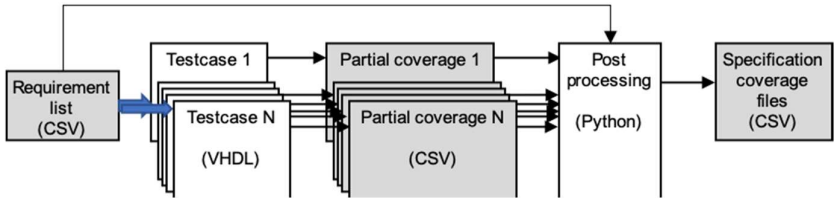


Figure 1: Workflow: Specification coverage vs Partial Coverage. Details in Figure 2:

5 Specification Coverage Concept

An important step of design verification is to check that all requirements have been met. Requirements can be defined very differently depending on application, project management, quality requirements, etc. In some projects, requirements barely exist, and the functionality is based on a brief description. However, in projects where safety and reliability are key the requirement handling is an essential part of the project management flow. In some development standards the requirements and the corresponding testcases that verify the requirements need to be defined, reviewed and accepted by a third-party assessor before even starting the verification flow.
This UVVM Verification IP is intended for projects where requirements are essential in the workflow but may also be used in a very simple way for projects with lower requirements. Examples of requirements can be seen in Table 1. It is in general a good idea to label the various requirements, and in many projects this would be mandatory. The example in Table 1 shows of course only a subset of all the requirements.

Requirement Label	Description
UART_REQ_1	The device UART interface shall accept a baud rate of 9600kbps.
UART_REQ_2	The device UART interface shall accept a baud rate of 19k2 bps.
UART_REQ_3	The device UART interface shall accept an odd parity
UART_REQ_4	The device reset shall be active low.

Table 1 Requirement examples. (Requirement labels are defined by the user)

There are lots of acceptable approaches with respect to how much functionality is verified in each testcase and how these are organised. This VIP will allow various approaches from dead simple to advanced. In order to explain the concepts, we start with the simplest case and add step-by-step on that until we have built a full advanced specification coverage system.

6 Conceptual introduction and the Simplest possible usage, with a single testcase

For any FPGA / ASIC it is always important to properly specify the design requirements and check that they have all been tested. Normally it is often just ticked off somewhere that a particular requirement is tested – often only once during the development phase, and sometimes just as a mental exercise. It is always better to use a written, repeatable and automated approach. This VIP significantly simplifies such an approach.

When feasible, the simplest structured approach would be to test all requirements in one single self-checking testcase. If so – all you want to do is the following – as illustrated in Figure 2.

1. List all DUT requirements in a requirement list CSV file. (RL).
   This could mean anything from just writing down the requirements directly, to a fully automated requirement extraction from an existing Requirement Specification.
   NOTE: For specification coverage this list is mandatory, but a simplified mode of pure test reporting without the need for a prior listing of the requirements is available (see more info in section 7.1).
2. Implement your testcase (T) with all tests required to verify the DUT. (see section 9.1)
   The test sequencer should initiate coverage using initialize_req_cov() (T1), then for each verified requirement call register_req_cov() (T2) and then finalise coverage reporting using finalize_req_cov() (T3).
3. When the testcase is executed (run), initialize_req_cov() (T1) (see section 9.1) will read the given requirement list file (RL), the new partial coverage file (PC) is created, and the testcase name is stored.
   The header of the partial coverage file – with NOTE, testcase-name and delimiter is written. The header is not shown in any of the examples, but is shown in the front page file formats.
   Then for each register_req_cov() (T2) a separate line is written into the partial coverage file with a) the given requirement label, b) the name of the testcase, and c) the result of the test.
   The result of the test will be PASS - unless marked as FAIL in the procedure call or unexpected serious alerts (>= ERROR / TB_ERROR) have occurred, in which case it will be marked as FAIL.
   Finally when finalize_req_cov() (T3) is executed, a closing check of the alert counters is made. If ok, then 'SUMMARY, <Testcase name>, PASS' is written at the end of the Partial coverage file. Otherwise FAIL rather than PASS (provided testcase does not stop on the alert). If a testcase fails before reaching finalize_req_cov(), then no SUMMARY line will be written. This is interpreted as FAIL.
   Note that a given requirement may be tested and reported several times, so that for instance UART_REQ_3 may be listed multiple times in the Partial coverage file (PC).
4. After the testcase has been executed, the overall Specification coverage (SC) can be found by executing the Python script run_spec_cov.py (section 10).
   This script traverses the requirement list (RL) and Partial coverage file (PC) and from that generates the specification coverage (SC).
   Each requirement is listed only once in the specification coverage file. If a requirement has one or more FAIL in a Partial coverage file, the result is NON_COMPLIANT for that requirement.
   For a simple scenario with a single testcase, the partial coverage (PC) file and the specification coverage file (SC) yield the same information, but the specification coverage potentially with fewer lines.

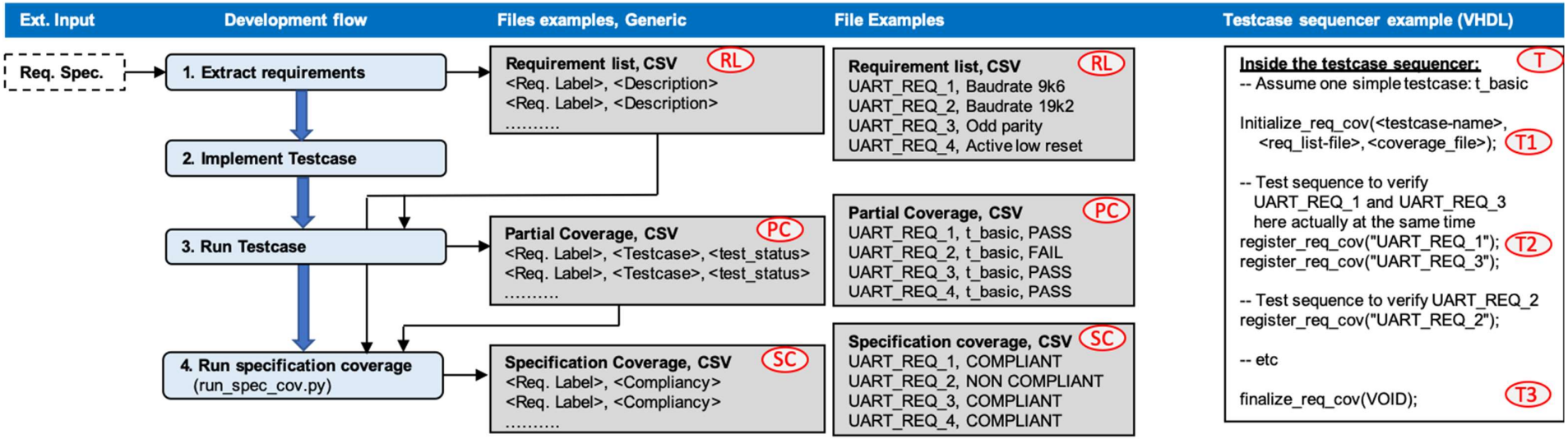


Figure 2: Simplest possible specification coverage (note that partial coverage files only show the actual requirement coverage lines)

7 Simple usage

7.1 Shortcut with no requirement list

A shortcut is supported to allow all tested requirements to be reported to the Partial coverage file – without the need for a prior listing of the requirements. This shortcut does of course not yield any specification coverage, as no specification is given, but could be useful for scenarios or early testing where only a list of executed tests is wanted. This shortcut mode is automatically applied when no requirement list is provided as an input to the initialize_req_cov() VHDL command in the test sequencer. If so, the partial coverage files will be generated, but no specification coverage file (the Python script) shall be run.

7.2 Simple usage, with multiple testcases

Many verification systems will have multiple testcases per DUT. If so, the above simplest approach is not possible.
However, if your tests are split on multiple testcases, but with no requirement as to which testcase tests what, then you can apply almost the same simple approach as the simplest case above.

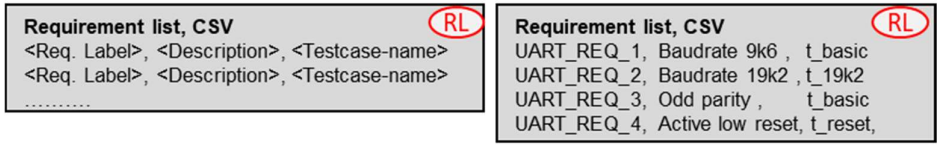
For this scenario, there will be two or more testcases, and so you will have to run all relevant testcases. There will of course still be only a single Requirement list.
For every single testcase the same set of commands will be applied – with initialize_req_cov(), one or more register_req_cov(), and finalize_req_cov(). The only thing to remember here is that initialize_req_cov() has to specify separate Partial coverage files for each testcase. Hence, after simulation you should end up with as many Partial coverage files as testcases

Note: register_req_cov() will as default use the testcase name specified with initialize_req_cov() when writing to the partial coverage file. It is however possible to specify a dedicated testcase name as a parameter to register_req_cov(). If this testcase name is different from the testcase name already specified by initialize_req_cov(), then this testcase name is used when logging that specific requirement to the partial coverage file. All following register_req_cov() with no testcase name specified will again use the name given with initialize_req_cov(). The partial coverage file may in other words include references to both itself and other testcase names. It is also possible inside a test sequencer to execute initialize_req_cov() multiple times, but only when the previous initialize_req_cov() has been terminated with finalize_req_cov(). If multiple initialize… & finalize… these should operate on different partial coverage files to avoid overwriting the previous section.

The Python script run_spec_cov.py will be run in the same way as before, but needs to be given a list of all the relevant Partial coverage files.
Then the Specification coverage is generated exactly as before.

7.3 Multiple testcases – with strict requirement vs testcase relation

For most applications where high quality and confidence is required it is mandatory to specify up front in which testcase a given requirement will be tested. In these cases, the requirement list must be extended to include the testcase in which a requirement will be tested, - as shown to the right. The example now shows more testcases than just t_basic.



Specifying a testcase name in the requirement list will however not force a requirement vs testcase check by itself. In order to check that a requirement is tested in the specified testcase the switch '–strictness 1' must be used when calling run_spec_cov.py as described in section 10.1.
If strictness is set to 1 then for example the requirement 'UART_REQ_3, Odd parity, t_basic will be:

a) marked as COMPLIANT in the specification coverage file if UART_REQ_3 is checked positive in testcase t_basic. UART_REQ_3 may additionally also be checked elsewhere.
b) Will be marked as NON-COMPLIANT in the specification coverage file if UART_REQ_3 is not checked in testcase t_basic (but for instance only in t_19k2).

Default strictness is –strictness 0, i.e. neither of the above strict checking. See section 10.1 for various strictness settings.

Testcase names and Requirement labels are not case sensitive for any comparison. For any output report the names and labels from the requirement list will be used if available. If not provided via the requirement list, then testcase name is taken from the initialize_req_cov() and the requirement label from the register_req_cov().

8 Advanced usage

8.1 Advanced usage – Only one of multiple testcases for a given requirement must pass

In the previous example all tests in all testcases had to pass for the overall specification coverage to pass.
There may however be situations where a given requirement is tested in multiple testcases and it is sufficient that only any one of these pass – given of course that none of the others have failed, but just haven't been executed. A fail in any executed testcase will always result in a summary fail.
This approach could for instance be used to qualify for a lab test release.
A requirement for this approach to work is to remove all old partial coverage files before running your test suite generating new partial coverage files. The run_spec_cov.py option –clean may be used for this.

Requirement list, CSV RL
<Req. Label>, <Description>, <Testcase-name> [, <Testcase-name>, <Testcase-name>, …]
<Req. Label>, <Description>, <Testcase-name> [, <Testcase-name>, <Testcase-name>, …]
……

Requirement list, CSV RL
UART_REQ_1, Baudrate 9k6 , t_basic
UART_REQ_2, Baudrate 19k2 , t_19k2
UART_REQ_3, Odd parity , t_basic, t_19k2
UART_REQ_4, Active low reset, t_reset, t_basic, t_19k2

The example above shows that UART_REQ_3 is covered by both testcases t_basic and t_19k2, and that UART_REQ_4 may in fact be tested by any of the three testcases.

This example also shows that testcase t_reset is not required to be executed in order for all requirements to be tested. This ***could*** be a sign that t_reset should be removed (optimized away), but given that t_reset is required for some special reason, then it could for instance be left out of from a reduced test suite to qualify for lab test only.

8.2 Advanced usage – All (or some) of multiple testcases for a given requirement must pass

This is basically the opposite of the above and is easy to achieve by just adding lines in the requirement list for all wanted combinations of requirements and testcases. The example to the right states that UART_REQ_3 must pass in both t_basic and t_19k2.

Requirement list, CSV RL
UART_REQ_1, Baudrate 9k6 , t_basic
UART_REQ_2, Baudrate 19k2 , t_19k2
UART_REQ_3, Odd parity , t_basic
UART_REQ_3, Odd parity , t_19k2
UART_REQ_4, Active low reset, t_reset, t_basic, t_19k2

8.3 Advanced usage – Requirement mapping

Requirement mapping just maps one or more requirements to another requirement.
This is intended for two different use cases:

8.3.1 Mapping of project requirements to IP or legacy requirements

Assuming you already have a UART IP, that has been properly verified, and the provided UART testbench already has full UVVM specification coverage support, with a requirement list file and testcases generating Partial coverage files.
Assuming you then have a project with its own UART requirements, that hopefully more or less matches that of the IP, but the requirement labels and combinations may be different.
Then you don't want to modify the provided UART testbench in order just to get the "right" requirement labels.
A much better approach is to map the project requirements to the UART IP requirements.

From the UART IP

Requirement list, CSV (RL)
UART_REQ_1, Baudrate 9k6 , t_basic
UART_REQ_2, Baudrate 19k2 , t_19k2
UART_REQ_3, Odd parity , t_basic
UART_REQ_4, Active low reset, t_reset,

From the PROJECT requirements

Requirement list, CSV (RL)
UART_REQ_A, Baudrate 9k6 and 19k2
UART_REQ_B, Odd parity

Now assume the already shown UART requirement list is that of the IP, and that we have similar project UART requirements, we may have a scenario as shown above.
Here we can see that UART_REQ_B of the project matches that of UART_REQ_3 of the UART IP, and that UART_REQ_A of the project is actually not covered by any single UART IP requirement, but in fact must include both UART_REQ_1 and UART_REQ_2.
The map would for this case be as shown to the right.

Requirement map file, CSV (RM)
UART_REQ_A, UART_REQ_1, UART_REQ_2
UART_REQ_B, UART_REQ_3

Making the requirement map file is of course a manual job, which could be simple or complex depending on how much the two requirement lists differ in structure.

The requirement map file is only used as an input to the run_spec_cov.py Python script, as is also the Partial coverage file from the UART IP verification using the IP (or legacy) testbench.
The Python script will check that

- For project requirement UART_REQ_A, both UART_REQ_1 and UART_REQ_2 have passed
- For project requirement UART_REQ_B, UART_REQ_3 has passed
- Etc…

The report from run_spec_cov.py will show compliancy for the project requirement (e.g.UART_REQ_A), but also for the "sub-requirement(s)" (e.g. UART_REQ_1 and UART_REQ_2).

8.3.2 Mapping of requirements to multiple sub-requirements

Often the original requirements are too complex (or compound), so that it is difficult to tick off a requirement as checked until a whole lot of different things are tested. An example of that could be a UART requirement like the one shown to the right – with a single UART_REQ_GENERAL requirement.
If you want to split this into more specific requirements you have several options, with some potential options listed below and illustrated to the right.

a) Rename the requirements
b) Extend the names
c) Extend the names using a record like notation

All of these and more are of course possible, but the problem is that they don't show the relation to the original requirement.

Requirement list, CSV
UART_REQ_GENERAL, Baudrates 9k6 and 19k2, odd and even parity

a) Potential new Requirement list, CSV
UART_REQ_BR_A, Baudrate 9k6
UART_REQ_BR_B, Baudrate 19k2
UART_REQ_ODD, Odd parity
UART_REQ_EVEN, Even parity

b) Potential new Requirement list, CSV
UART_REQ_GENERAL_BR_A, Baudrate 9k6
UART_REQ_ GENERAL_BR_B, Baudrate 19k2
UART_REQ_ GENERAL_ODD, Odd parity
UART_REQ_ GENERAL_EVEN, Even parity

c) Potential new Requirement list, CSV
UART_REQ_GENERAL.BR_A, Baudrate 9k6
UART_REQ_ GENERAL.BR_B, Baudrate 19k2
UART_REQ_ GENERAL.ODD, Odd parity
UART_REQ_ GENERAL.EVEN, Even parity

Showing this relationship is quite simple in UVVM, by using exactly the same mechanism as for the IP or legacy scenario in the previous example.
Thus all you have to do is to make your own requirement list and then map the original requirement to a set of requirements in your new list.

Requirement map file, CSV
UART_REQ_GENERAL, UART_REQ_GENERAL.BR_A, UART_REQ_GENERAL.BR_B, UART_REQ_GENERAL.ODD, UART_REQ_GENERAL.EVEN

- Or even simpler just name the subrequirements A,B,C,….

Requirement map file, CSV
UART_REQ_GENERAL, UART_REQ_GENERAL.A, UART_REQ_GENERAL.B, UART_REQ_GENERAL.C, UART_REQ_GENERAL.D

As for the IP scenario this approach allows the report to show both the original requirement and its new sub-requirements.

9 VHDL Package

A vital part of the specification coverage concept is the VHDL testbench methods. These methods are described in Table 2. The methods are located inside the *spec_cov_pkg.vhd* file in the *src/* directory of this VIP.

9.1 VHDL Methods Details

Procedure	Parameters and examples	Description

initialize_req_cov()	testcase (string), req_list_file (string), partial_cov_file (string)
	or
	testcase (string), partial_cov_file (string)
	
	Examples
	initialize_req_cov("T_UART_9k6", "c:/my_folder/requirements.csv", "./cov_9k6.csv");
	initialize_req_cov("T_UART_9k6", "requirements.csv", "cov_9k6.csv");
		Starts the requirement coverage process in a testcase.
		
		The requirement list file is optional, but without it the specification coverage is of course not possible and run_spec_cov.py shall not be executed.
		
		The partial_coverage_file is created – and the header is written with NOTE, Testcase-name and Delimiter on the first three lines. If file already exists, it will be overwritten.

register_req_cov()	requirement(string) [, testcase(string)] [, test_status(t_test_status)]
	or
	requirement(string) [, testcase(string)], test_status(t_test_status), msg [, scope]
	
	Examples
	-- Will pass if no unexpected alert occurred
	register_req_cov("UART_REQ_1", "T_UART_9k6");
	
	-- Will fail since passed argument is set to false
	register_req_cov("UART_REQ_1", "T_UART_9k6", FAIL);
	
	-- In order to include msg and scope test_status must be included
	register_req_cov("UART_REQ_1", "T_UART_9k6", NA, "my_msg");
	or e.g.
	register_req_cov("UART_REQ_1", NA, "my_msg", "my_scope");
		Evaluates and logs the specified requirement. The procedure checks the global alert mismatch status, and if an alert mismatch is present on ERROR, FAILURE, TB_ERROR or TB_FAILURE the requirement will be marked as FAIL. If there are no such alert mismatches, the requirement will be marked as PASS, unless the test_status is explicitly set to FAIL.
		The result is written to both the transcript (and log) and the partial coverage file (specified in the initialize_req_cov() command).
		The *register_req_cov()* will look up the specified requirement and testcase in the requirement list specified in *initialize_req_cov()*, and use the description from this entry as a minimum log message. The procedure will also issue a TB_WARNING if the specified requirement was not found.
		
		- *requirement:* String with the requirement label. Must as default match a requirement label in the given requirement list.
		- *testcase*: Optional: String with the testcase name. Default: Testcase name from initialize_req_cov()
		- test_status*:* Optional: Enter FAIL to explicitly fail the requirement. Default: NA. Not applicable means test status will be determined by an alert mismatch as described above.
		- msg Optional message. Only possible after preceding test_status (use NA unless FAIL).
		- scope Optional scope. Only possible after preceding msg.

finalize_req_cov()	VOID(t_void) **Example** finalize_req_cov(VOID);	Ends the requirement coverage process in a test. If alert status is OK – appends a line to partial coverage file: 'SUMMARY, <Testcase name>, PASS' If alert status is not OK – appends a line to partial coverage file: 'SUMMARY, <Testcase name>, FAIL' This line is used later by the run_spec_cov.py script. If simulation never reached this command, e.g. if failed, then no summary line is written – indicating FAIL

Table 2 VHDL Methods

9.2 Specification Coverage configuration record: shared_spec_cov_config

This record is located in the local adaptation package 'bitvis_vip_spec_cov/src/local_adaptations_pkg.vhd'
The configuration record is applied as a shared_variable 'shared_spec_cov_config' to allow different configuration for different DUTs.
Any test sequencer may then set the complete record as required – or even just parts of it like shared_spec_cov_config.csv_delimiter := ';';

Record elements	Type	Default	Description
missing_req_label_severity	t_alert_level	TB_WARNING	Alert level used when the register_req_cov() procedure does not find the specified requirement label in the requirement list, given that a requirement list is given in the initialize_req_cov() command.
csv_delimiter	character	','	Character used as delimiter in the CSV files. This will also be written into all partial coverage files. run_spec_cov.py will find the delimiter there.
max_requirements	natural	1000	Maximum number of requirements in the req_map file used in initialize_req_cov(). Increase this number if the number of requirements exceeds 1000.
max_testcases_per_req	natural	20	Maximum number of testcases allowed per requirement. This is applicable when one requirement is verified by one or more testcases.
csv_max_line_length	positive	256	Maximum length of each line in any CSV file. (i.e. max number of characters for all values and separators in total)

Table 3 Specification coverage adaptation

The specification coverage implementation uses three new message IDs, as described in the table below. All message Ids are located in uvvm_util adaptations package. The specification coverage implementation uses the shared message id panel for all logging.

Message Id	Description
ID_FILE_OPEN_CLOSE	Id used for any file open and close operation
ID_FILE_PARSER	Id used for CSV parser messages.
ID_SPEC_COV	Id used for all messages that are not directly related to CSV parsing.

Table 4 Message ID usage

9.3 Additional Documentation

Additional documentation about UVVM and its features can be found under "/uvvm_vvc_framework/doc/".

9.3.1 Compilation

This VHDL package may only be compiled with VHDL 2008. It is dependent on the following libraries

- ***UVVM Utility Library (UVVM-Util), version 2.11.0 and up***
- ***UVVM VVC Framework, version 2.7.0 and up***

Before compiling the Specification Coverage component, make sure that uvvm_vvc_framework and uvvm_util have been compiled.
See UVVM Essential Mechanisms located in uvvm_vvc_framework/doc for information about compile scripts.

Compile order for the Specification vs Verification Matrix Component:

Compile to library	File	Comment
bitvis_vip_spec_cov	local_adaptations_pkg.vhd	Local file for user adaptations
bitvis_vip_spec_cov	csv_file_reader_pkg.vhd	Package for reading and parsing of CSV input files
bitvis_vip_spec_cov	spec_coverage_pkg.vhd	Specification Coverage component implementation

Table 5 Compile order

9.3.2 Simulator compatibility and setup

See UVVM/README.md for a list of supported simulators.
For required simulator setup see UVVM-Util Quick reference.

10 Post-processing Script

The final step of the Specification Coverage usage is to run a post-processing script to evaluate all the simulation results. This script is called *run_spec_cov.py*. The script requires Python 3.x.
The script can be called with the arguments listed in Table 6 from the command line.
The CSV delimiter is fetched by the Python script from the partial coverage file headers.

Note: All files may be referenced with absolute paths or relative to working directory.

Argument	Short form	Example	Description
--requirement_list	-r	--requirement_list path/requirements.csv	Points to the requirement list. This argument is mandatory. (For the case without a requirement list – this script may not be executed and wouldn't make sense anyway.)
--partial_cov	-p	--partial_cov my_testcase_cov.csv --partial_cov my_coverage_files.txt	Points to the partial coverage file generated by the VHDL simulation as input to the post processing. May also point to a file list including references to multiple partial coverage files. The format of such a file would be just each file on a separate line – potentially prefixed by a relative or absolute path.
--requirement_map_list	-m	--requirement_map_list path/subrequirements.csv	Optional: Points to the requirement map file, described in section 8.3. If this argument is omitted, the script assumes that no sub-requirements exist.
--spec_cov	-s	--spec_cov uart_spec_cov.csv	Name (and optional path) of the specification_coverage file name, which is used to generate the following 3 files: • <specification_coverage_file_name>.req_vs_single_tc.csv • <specification_coverage_file_name>.tc_vs_reqs.csv • <specification_coverage_file_name>.req_vs_tcs.csv Note that the filename extension, i.e. .csv, will have to be part of the specified specification_coverage file name.
--clean		--clean	Will clean any/all given partial coverage files. No short form defined here - to avoid unwanted clean
--strictness		--strictness 1	Default strictness is 0 (when not applied). 1 is stricter and 2 is much stricter (see sections 7.3 and 10.1). No short form defined here – to avoid wrong strictness
--config		--config path/config_file.txt	Optional configuration file where all the arguments can be placed. This argument will override all other arguments. The configuration file does not need to have the .txt extension. All arguments shall be added on a new line. Example configuration file contents: *--requirement_list my_path/requirements.csv* *--partial_cov my_testcase_cov_files.txt* *--spec_cov my_spec_cov.csv*
--help	-h	--help	Display the script argument options.

Table 6 Script Arguments

10.1 Strictness for requirement vs testcase relation

Strictness does not apply to VHDL testcases – only to the post processing Python script.
For all strictness levels, all requirements must be compliant for the complete specification to be compliant.
Default strictness is –strictness 0.

10.1.1 Strictness 0

This is the least strictness possible. Focus is only on the requirements, - with no concern at all as to in which testcase the various requirements have been tested.
Any requirement is compliant if executed as PASSed in any passing testcase, and not failing anywhere. This is independent of whether one or more testcases are specified for a given requirement in the Requirement list.

10.1.2 Strictness 1

For this strictness level a requirement is only compliant if executed in the testcase(s) specified in the Requirement list. The list shown here is used as example for the cases below:

- If no testcase is specified for a given requirement, this requirement may be checked anywhere. Compliant if PASSed.
- If a requirement line is specified with a single testcase, this requirement (e.g. UART_REQ_1) is only compliant if executed by that testcase.
- If a requirement line is specified with multiple testcases (e.g. UART_REQ_4), this requirement is compliant if PASS in one or more of these testcases. There can be no FAIL in other testcases.
- If a requirement line is specified multiple times (like UART_REQ_3), every single line is mandatory. Hence UART_REQ_3 must PASS in both testcases (t_basic and t_19k2)

Requirement list, CSV
UART_REQ_1, Baudrate 9k6 , t_basic
UART_REQ_2, Baudrate 19k2 , t_19k2
UART_REQ_3, Odd parity , t_basic
UART_REQ_3, Odd parity , t_19k2
UART_REQ_4, Active low reset, t_reset, t_basic, t_19k2

- A requirement may be checked in **any** testcase in addition to any specified testcase(s).
- If a requirement status from any testcase is FAIL, that requirement is NON COMPLIANT, - even if PASS in other testcases.

10.1.3 Strictness 2

With this strictness a requirement will be NON_COMPLIANT if it is tested in a testcase that in the Requirement list was not specified as testcase for that requirement. In that case it will be NON_COMPLIANT even if it has passed in one or more testcases specified for that requirement.
E.g. if UART_REQ_1 is tested in testcase t_basic, but **also** in t_reset.

10.2 Output of post-processing script

The output of the post-processing lists all the requirements and sub-requirements marked as either compliant or non-compliant in three different files, each with a different format. The post-processing script will also print a transcript to file, where it is indicated whether or not the script succeeded. The formats of the output files are the following:

File	**<spec_cov_file>.req_vs_single_tc.csv**	**<spec_cov_file>.tc_vs_reqs.csv**	**<spec_cov_file>.req_vs_tcs**
Layout	*"Requirement", "Testcase", "Compliance"*	*"Testcase", "Requirements", "Result"*	*"Requirement", "Testcases", "Compliance"*
Example	FPGA_REQ_1, T_UART_1, COMPLIANT FPGA_REQ_2, T_UART_1, COMPLIANT FPGA_REQ_2, T_UART_2, COMPLIANT FPGA_REQ_3, , NOT VERIFIED FPGA_REQ_4, , NOT VERIFIED	T_UART_1, FPGA_REQ_1 FPGA_REQ_2, PASS T_UART_2, FPGA_REQ_2, PASS T_UART_5, , NOT RUN	FPGA_REQ_1, T_UART_1, COMPLIANT FPGA_REQ_2, T_UART_1 T_UART_2, COMPLIANT FPGA_REQ_3, , NOT VERIFIED FPGA_REQ_4, , NOT VERIFIED

11 Example demos

There are two example demos provided under the demo directory, one with the most basic usage and one with a more complete functionality.