Scoreboard - Quick Reference

UVVM Verification IP

Basic methods
config([instance], sb_config, [msg])
config(sb_config_array, [msg])
enable([instance], [msg])
disable([instance], [msg])
add_expected([instance], expected_element, [msg], [source_element])
add_expected([instance], expected_element, tag_usage, tag, [msg], [source_element])
check_received([instance], received_element, [msg])
check_received([instance], received_element, tag_usage, tag, [msg])
flush([instance], [msg])
reset([instance], [msg])
v_empty := **is_empty**([instance])
v_entered_count := **get_entered_count**([instance])
v_pending_count := **get_pending_count**([instance])
v_match_count := **get_match_count**([instance])
v_mismatch_count := **get_mismatch_count**([instance])
v_drop_count := **get_drop_count**([instance])
v_initial_garbage_count := **get_initial_garbage_count**([instance])
v_delete_count := **get_delete_count**([instance])
set_scope(scope)
v_scope := **get_scope**(void)
enable_log_msg([instance], msg_id)
disable_log_msg([instance], msg_id)
report_counters([instance])

Advanced methods
insert_expected([instance], identifier_option, identifier, expected_element, [msg], [source_element])
insert_expected([instance], identifier_option, identifier, expected_element, tag_usage, tag, [msg], [source_element])
delete_expected([instance], expected_element, [msg])
delete_expected([instance], expected_element, tag_usage, tag, [msg])
delete_expected([instance], tag_usage, tag, [msg])
delete_expected([instance], identifier_option, identifier_min, identifier_max, [msg])
delete_expected([instance], identifier_option, identifier, range_option, [msg])
v_entry_num := **find_expected_entry_num**([instance], expected_element)
v_entry_num := **find_expected_entry_num**([instance], expected_element, tag_usage, tag)
v_entry_num := **find_expected_entry_num**([instance], tag_usage, tag)
v_position := **find_expected_position**([instance], expected_element)
v_position := **find_expected_position**([instance], expected_element, tag_usage, tag)
v_position := **find_expected_position**([instance], tag_usage, tag)
v_expected_element := **peek_expected**([instance])
v_expected_element := **peek_expected**([instance], identifier_option, identifier)
v_source_element := **peek_source**([instance])
v_source_element := **peek_source**([instance], identifier_option, identifier)
v_tag := **peek_tag**([instance])
v_tag := **peek_tag**([instance], identifier_option, identifier)
v_expected_element := **fetch_expected**([instance], [msg])
v_expected_element := **fetch_expected**([instance], identifier_option, identifier, [msg])
v_source_element := **fetch_source**([instance], [msg])
v_source_element := **fetch_source**([instance], identifier_option, identifier, [msg])
v_tag := **fetch_tag**([instance], [msg])
v_tag := **fetch_tag**([instance], identifier_option, identifier, [msg])
v_exists := **exists**([instance], expected_element, [tag_usage], [tag])
v_exists := **exists**([instance], tag_usage, tag)

Scoreboard – Generics

Generic element	Type	DEFAULT
element_type	t_element	std_logic_vector
element_match	function	=
to_string_element	function	to_string
sb_config_default	t_sb_config	C_SB_CONFIG_DEFAULT
GC_QUEUE_COUNT_MAX	natural	1000
GC_QUEUE_COUNT_MAX_THRESHOLD	natural	950

Note 1: VOID is used as parameter in methods with no functional parameter. E.g: is_empty(VOID).

Note 2: optional parameters are in brackets, when there are multiple optional parameters all preceding parameters must also be listed. The exception for this is the instance parameter.

VHDL 2008 only

Scoreboard – Functional parameters

Name	Type	Example(s)	Description
sb_config	t_sb_config		Configuration for the scoreboard.
sb_config_array	t_sb_config_arry		Array of t_sb_config.
scope	String	"UART"	The scope for the scoreboard instance.
expected_element	t_element	v_uart_data	The element that shall be pushed to the queue.
received_element	t_element	v_uart_data	The element that shall be checked in scoreboard.
source_element	t_source_element	v_source_data	The element that is the raw-input to the DUT, before DUT-processing. For debugging and logging only.
void	t_void	VOID	Unused, empty input parameter.
instance	integer	1	The instance number of the scoreboard. The scoreboard can have several instances, each instance acts as an individual scoreboard with its own queue and statistics. Can also be ALL_INSTANCES and ALL_ENABLED_INSTANCES. Note that instance index 0 is allowed, but will have to be specified in all method calls, and that instance index 1 is default.
tag_usage	t_tag_usage	TAG	Enumerated type used before every tag is entered. In the function syntax-descriptions this argument is given as 'TAG' as this is the only option.
tag	string	"byte1", "uart_ch3", "7"	Tag of the scoreboard-element.
identifier_option	t_identifier_option	POSITION, ENTRY_NUM	Option for type of identifier used. Can only be POSITION or ENTRY_NUM.
identifier identifier_min identifier_max	positive	1	Positive integer value. Describes the position or entry number, depending on identifier_option.
range_option	t_range_option	AND_HIGHER	Enumerated type used to describe the range. Possible options: AND_HIGHER, AND_LOWER, SINGLE.
alert_level	t_alert_level	ERROR	The alert-level.

Scoreboard – Configuration

Scoreboard configuration record ´**t_sb_config**´

Record element	Type	C_SB_CONFIG_DEFAULT	Description
mismatch_alert_level	t_alert_level	ERROR	The severity level of alert if mismatch between expected and received.
allow_lossy	boolean	FALSE	If TRUE all entries are searched until a matching entry is found, all entries before the match are dropped. If no matching entry is found a mismatch is registered.
allow_out_of_order	boolean	FALSE	If TRUE all entries are searched until match. If no matching entry is found a mismatch is registered.
overdue_check_alert_level	t_alert_level	WARNING	The severity level of alert if the time between entry and check is more than limit.
overdue_check_time_limit	time	0 ns	The time limit of which the entries should be in scoreboard. 0 ns indicates no time limit.
ignore_initial_garbage	boolean	FALSE	If TRUE all mismatches before first match are ignored and increment the initial drop count. Typically used if garbage-data is expected at start-up.

The configuration cannot be set with true for both allow_lossy and allow_out_of_order.
This will raise a TB_ERROR when running the config method.

Scoreboard simple usage

In the predefined_sb.vhd file there is a slv and an integer predefined scoreboard, and setting up a scoreboard is fast and has only a few mandatory steps:

- declare the scoreboard package and the generics
- define the scoreboard as a shared variable
- set the configuration, scope and enable the scoreboard
- start using the scoreboard

Scoreboard declaration:

```
use work.slv_sb_pkg.all;
shared variable slv_sb : t_generic_sb;
```

In sequencer:

```
library bitvis_vip_scoreboard;
use bitvis_vip_scoreboard.generic_sb_support_pkg.all;
...
slv_sb.config(C_SB_CONFIG_DEFAULT); -- initialize scoreboard
slv_sb.enable(VOID); -- enable scoreboard
slv_sb.set_scope("SLV SB"); -- set name of scoreboard
...
slv_sb.add_expected(v_expected, "Adding expected");
...
slv_sb.check_received(v_output, "Checking DUT output");
...
check_value(slv_sb.is_empty(VOID), ERROR, "Check that scoreboard is empty");
...
slv_sb.report_counters(VOID);
```

For more advanced examples see VHDL example code file.

Scoreboard details

All Scoreboard functions and procedures commonly referred to as methods are defined in the UVVM Scoreboard package, generic_sb_pkg.vhd

The scoreboard can be used with a single instance or with multiple instances. An instance is a separate queue with its own statistics. When multiple instances are used the methods are called with the instance parameter. When only using a single scoreboard instance, the instance parameter may be omitted if that instance index is 1. Multiple instances are typically used when multiple scoreboards with the same data types are needed. Note that scoreboard instance numbering is in the range of 0 to N, with 1 as default.
All parameters in brackets are optional. No optional parameter can be used without using the preceding optional parameter, with the exception of the instance parameter.

Each entry in the scoreboard instance consists of an expected element, source element and tag. Each entry also has an entry number, which is specific for each entry. The entry number is used as an identifier for a specific entry in some of the advanced methods.

A message parameter can be used for print to transcript for all methods that affect the data or the functionality of the scoreboard.

1 Basic methods

Method	Description

config()	**config ([instance], sb_config, [msg])**
	config (sb_config_array, [msg])
	
	This method updates the scoreboard instance with the configuration input. The method with sb_config parameter can be called with an instance parameter, and the sb_config is then applied to the specified instance. If there is no instance parameter the config is applied to instance 1.
	If the parameter is an array, the config at index *n* is applied to instance *n*.
	
	msg_id: ID_CONFIG
	
	Example:
	`uart_sb.config(uart_sb_config);`
	`uart_sb.config(2, uart_sb_config, "Initialize config on instance 2");`
	`uart_sb.config(uart_sb_config_array, "Initialize config on instances 1-5"); --`

enable()	**enable ([instance], [msg])**
	
	Enables the specified instance and must be called before any other method is called, except config(). This method can be called with ALL_INSTANCES as parameter. If there is no instance parameter instance 1 is enabled.
	
	msg_id: ID_CONFIG
	
	Example:
	`uart_sb.enable(VOID);`
	`uart_sb.enable(2); -- enable instance 2`
	`uart_sb.enable(ALL_INSTANCES); -- enable all instances`

disable()	**disable ([instance], [msg])**
	
	Disables the specified instance. This method can be called with ALL_INSTANCES as parameter. If there is no instance parameter instance 1 is disabled.
	
	msg_id: ID_CONFIG
	
	Example:
	`uart_sb.disable(VOID);`
	`uart_sb.disable(2); -- disable instance 2`
	`uart_sb.disable(ALL_INSTANCES); -- disable all instances`

add_expected()

add_expected ([instance], expected_element, [msg], [source_element]) ***NOTE: tag_usage can only be TAG**
add_expected ([instance], expected_element, tag_usage*, tag, [msg], [source_element])

Inserts expected element at the end of scoreboard.

msg_id: ID_DATA

Example:

```
uart_sb.add_expected(x"AA");
uart_sb.add_expected(1, x"AA", TAG, "byte1", "Insert byte 1", x"A");
uart_sb.add_expected(1, x"AA", "Insert byte 1", x"A");
```

check_received()

check_received ([instance], received_element, [msg]) ***NOTE: tag_usage can only be TAG**
check_received ([instance], received_element, tag_usage*, tag, [msg])

Checks received data against oldest element in the scoreboard. If the element is found, it is removed from the scoreboard and the matched-counter is incremented. If the element is not found, an alert is triggered and the mismatch-counter is incremented. If out-of-order is allowed the scoreboard is searched from front to back. The mismatch-counter is incremented if no match is found. If lossy is allowed the scoreboard is searched from front to back. If a match occurs, the match counter is incremented and the preceding entries dropped. If no match occurs, the mismatch counter is incremented.

msg_id: ID_DATA

Example:

```
uart_sb.check_received(v_received_byte);
uart_sb.check_received(2, v_received_byte, "Check byte1"); -- Check received data against scoreboard elements in inst. 2
uart_sb.check_received(3, v_received_byte, TAG, "byte1", "Check byte 1"); -- Check received data against scoreboard
                                                                          -- elements with the TAG "byte1" in instance 3
```

flush()

flush ([instance], [msg])

Deletes all entries in the specified instance. Can also be called with parameter ALL_INSTANCES.

msg_id: ID_DATA

Example:

```
uart_sb.flush(VOID);
uart_sb.flush(2);
uart_sb.flush(ALL_INSTANCES);
```

reset()

reset ([instance], [msg])

Deletes all entries and resets the statistics in the specified instance. Config is not affected. Can also be called with parameter ALL_INSTANCES.

msg_id: ID_CONFIG

Example:
```
uart_sb.reset(VOID); -- NO
uart_sb.reset(2);
uart_sb.reset(ALL_INSTANCES);
```

is_empty()

is_empty ([instance])

Returns true if the scoreboard is empty. Returns true if instance is empty when instance parameter is specified.

Example:
```
if uart_sb.is_empty(1) then ...
```

get_entered_count()

get_entered_count ([instance])

Returns total number of entries that have been added or inserted into the scoreboard.

Example:
```
v_entered_count := uart_sb.get_entered_count(1);
```

get_pending_count()

get_pending_count ([instance])

Returns the number of remaining entries in scoreboard.

Example:
```
v_pending_count := uart_sb.get_pending_count(1);
```

get_match_count()

get_match_count ([instance])

Returns the number of checks with match in scoreboard.

Example:
```
v_match_count := uart_sb.get_match_count(1);
```

get_mismatch_count()

get_mismatch_count ([instance])

Returns the number of checks without match in scoreboard.

Example:
```
v_mismatch_count := uart_sb.get_mismatch_count(1);
```

get_drop_count()

get_drop_count ([instance])

Returns the number of dropped items during lossy mode. Initial drop count is not included.

Example:
```
v_drop_count := uart_sb.get_drop_count(1);
```

get_initial_garbage_count()

get_initial_garbage_count ([instance])

Returns the number of dropped items before first match when ignore_initial_mismatch in config is TRUE.

Example:
```
v_initial_garbage_count := uart_sb.get_initial_garbage_count(1);
```

get_delete_count()

get_delete_count ([instance])

Returns the number of explicitly deleted items by delete and flush methods.

Example:
```
v_delete_count := uart_sb.get_delete_count(1);
```

set_scope()

set_scope (scope)

Sets the scope of the scoreboard.

msg_id: ID_CONFIG

Example:
```
uart_sb.set_scope("UART");
```

get_scope()

get_scope (void)

Returns the scope of the scoreboard instance.

Example:
```
v_scope := uart_sb.get_scope(VOID);
v_scope := uart_sb.get_scope(2);
```

enable_log_msg()

enable_log_msg ([instance], msg_id)

Enables the message id for the specified instance.

Example:

```
uart_sb.enable_log_msg(ID_CONFIG);
uart_sb.enable_log_msg(2, ID_DATA);
uart_sb.enable_log_msg(ALL_INSTANCES, ID_CONFIG);
```

disable_log_msg()

disable_log_msg ([instance], msg_id)

Disables the message id for the specified instance.

Example:

```
uart_sb.disable_log_msg(ID_CONFIG);
uart_sb.disable_log_msg(2, ID_DATA);
uart_sb.disable_log_msg(ALL_INSTANCES, ID_CONFIG);
```

report_counters()

report_counters ([instance])

Prints the scoreboard results. If no instance parameter is specified all enabled instances is reported.

Example:

```
uart_sb.report_counters(VOID);
uart_sb.report_counters(4);
uart_sb.report_counters(ALL_ENABLED_INSTANCES);
```

2 Advanced methods

Method	Description

insert_expected()	**insert_expected ([instance], identifier_option, identifier, expected_element, [msg], [source_element])** ***NOTE: tag_usage can only be TAG**
	insert_expected ([instance], identifier_option, identifier, expected_element, tag_usage*, tag, [msg], [source_element])
	
	Inserts expected element into the scoreboard in the specified position or after the entry specified by entry number, depending on the parameter identifier is POSITION or ENTRY_NUM respectively.
	
	msg_id: ID_DATA
	
	Example:
	`uart_sb.insert_expected(POSITION, 5, x"AA"); -- insert element with value x"AA" at position 5`
	`uart_sb.insert_expected(2, ENTRY_NUM, 23, x"AA", TAG, "byte3", "Insert byte 4", x"A"); -- insert element with value`
	`-- x"AA", tag "byte3" and source element x"A" after element with entry number 23`

delete_expected()	**delete_expected ([instance], element, [msg])** ***NOTE: tag_usage can only be TAG**
	delete_expected ([instance], element, tag_usage*, tag, [msg])
	delete_expected ([instance], tag_usage*, tag, [msg])
	delete_expected ([instance], identifier_option, identifier_min, identifier_max, [msg])
	delete_expected ([instance], identifier_option , identifier, range_option, [msg])
	
	Deletes the foremost matching entry from the scoreboard queue. When both tag and element is parameters both have to match. If identifier parameter is used, the specified entry is deleted. Identifier can be used with range_option set as SINGLE, AND_HIGHER and AND_LOWER. A range between identifier_min and identifier_max can also be defined, entries between and including these values are deleted. A TB_ERROR alert is reported if identifier is not found.
	
	msg_id: ID_DATA
	
	Example:
	`uart_sb.delete_expected (v_received_data);`
	`uart_sb.delete_expected (TAG, "byte1");`
	`uart_sb.delete_expected (1, ENTRY_NUM, v_entry_num, v_entry_num);`
	`uart_sb.delete_expected (1, POSITION, v_position, AND_LOWER, "Delete entry in specified position and positions lower");`

find_expected_position()
find_expected_entry_num()
	find_expected_position / find_expected_entry_num ([instance], element, [msg]) ***NOTE: tag_usage can only be TAG**
	find_expected_position / find_expected_entry_num ([instance], element, tag_usage*, tag, [msg])
	find_expected_position / find_expected_entry_num ([instance], tag_usage*, tag, [msg])
	
	Returns entry number or position to entry if found, else -1. The search parameter can be tag, element or both.
	
	msg_id: ID_DATA
	
	Example:
	`v_entry_num := uart_sb.find_expected_entry_num(x"AA");`
	`v_position  := uart_sb.find_expected_position(TAG, "byte1");`
	`v_position  := uart_sb.find_expected_position(1, x"AA", TAG, "byte1", "Find byte 1");`

peek_expected()
peek_source()
peek_tag()

peek_expected / peek_source / peek_tag ([instance])
peek_expected / peek_source / peek_tag ([instance], identifier_option, identifier)

Returns the expected_element/source_element/tag of specified entry in scoreboard without deleting entry. Returns front entry expected_element/source_element/tag in scoreboard queue if no identifier specified. If identifier is not found TB_ERROR alert is reported and undefined is returned.

Example:
```
v_expected_element := uart_sb.peek_expected(VOID);
v_source_element   := uart_sb.peek_source(1, POSITION, v_position);
v_tag              := uart_sb.peek_tag(1, ENTRY_NUM, v_entry_num);
```

fetch_expected()
fetch_source()
fetch_tag()

fetch_expected / fetch_source / fetch_tag ([instance], [msg])
fetch_expected / fetch_source / fetch_tag ([instance], identifier_option, identifier, [msg])

Returns the expected_element/source_element/tag of specified entry in scoreboard and deletes the entry. Returns the front entry expected_element/source_element/tag in scoreboard queue if no identifier specified. If identifier is not found TB_ERROR alert is reported and undefined is returned.

The method deletes the entry. If all information from an entry shall be retrieved, peek must be used for the first two entry elements.

msg_id: ID_DATA

Example:
```
v_expected_element := uart_sb.fetch_expected(VOID);
v_source_element   := uart_sb.fetch_source(1);
v_tag              := uart_sb.fetch_tag(1, ENTRY_NUM, v_entry_num, "Fetch tag");
v_expected_element := uart_sb.fetch_expected(1, POSITION, v_position, "Fetch expected");
```

exists()

exists ([instance], expected_element, [tag_usage*], [tag])
exists ([instance], tag_usage*, tag)

***NOTE: tag_usage can only be TAG**

Returns true if entry found, false if not. If element and tag parameter is entered, both must match.

Example:
```
if uart_sb.exists(1, x"AA", TAG, "byte1") then ...
if uart_sb.exists(1, TAG, "byte1") then ...
```

3 Additional documentation

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

4 Compilation

The Generic Scoreboard must be compiled with VHDL 2008.
It is dependent on the following libraries

- ***UVVM Utility Library (UVVM-Util), version 2.5.0 and up***
- ***UVVM VVC Framework, version 2.3.0 and up***

Before compiling the Generic Scoreboard, assure that uvvm_vvc_framework and uvvm_util have been compiled.

Compile order for the Generic Scoreboard:

Compile to library	File	Comment
bitvis_vip_scoreboard	generic_sb_support_pkg.vhd	Config declaration
bitvis_vip_scoreboard	generic_sb_pkg.vhd	Generic Scoreboard
bitvis_vip_scoreboard	predefined_sb.vhd	Predefined packages with SLV and integer

5 Simulator compatibility and setup

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

Appendix

A verification scoreboard typically includes some way to store expected outputs generated by e.g. the sequencer or a reference model, compare the expected outputs to received outputs, and to keep track of pass and failure rates identified in the comparison process. The advantage of using a scoreboard is that the checking/monitor side remains simple and has no need to know what the test/stimulus generation side is doing. Figure 1 shows the dataflow through the scoreboard and its main functions.

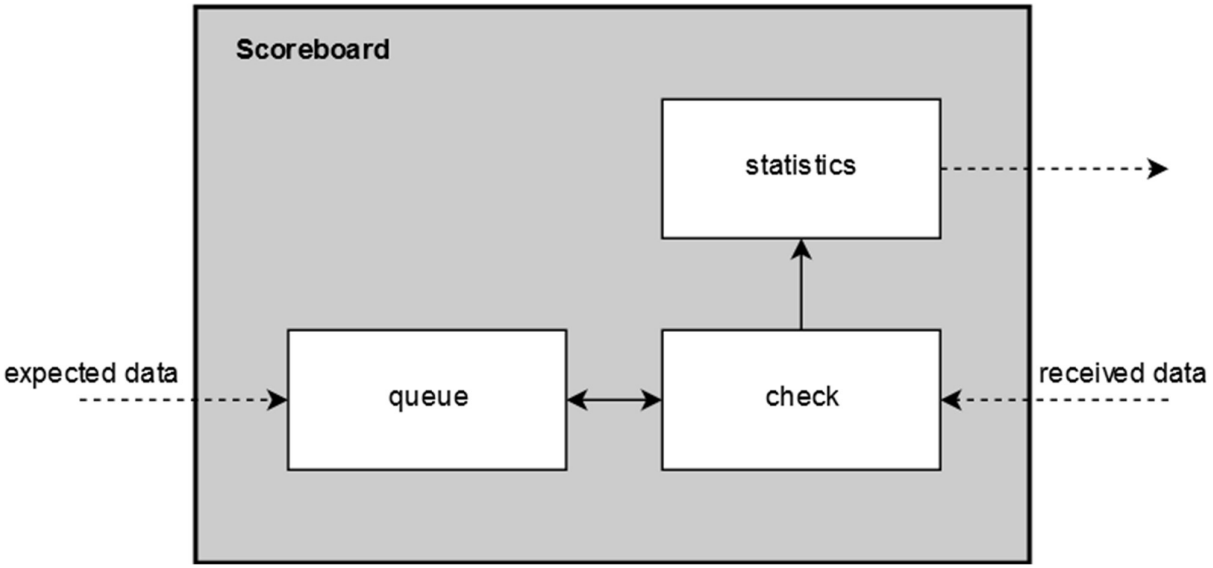

Figure 1: Basic scoreboard

The Scoreboard operate as a queue that holds generic elements. It is implemented as a protected type in a generic package, and a package declaration has to be made for each element type. The scoreboard offers advanced functionality for checking received data against expected data.

Out of order
Data can be checked out of order by enabling the allow_out_of_order parameter in config.

Lossy
Lossy protocols, where data can be dropped during transfer, is supported by enabling allow_lossy parameter in config.

Out of order and lossy cannot both be enabled at the same time; the scoreboard can't know if something is loss or out of order. In the case of both out of order and lossy, enable out of order. The number of remaining entries in the scoreboard at the end of the simulation is the number of dropped data elements.

Use-cases

Scenarios where a scoreboard with out-of-order function is applicable:

- DUT with multiple execution paths:
  Data can take multiple paths inside DUT which affects the execution time. Modulating packet-order may be complex and all that is needed is a check that all data is getting through independent of order.

- Many-to-one interfaces:
  Data from multiple asynchronous interfaces into the DUT is being retransmitted in one interface out of the DUT.

Scenarios where a scoreboard with lossy function is applicable:

- Interfaces with packet-loss:
  Packet-loss is accepted and handled at a higher abstraction level. E.g. TCP.

- Non-monitored error-injection:
  Error is injected at a level that is not monitored into the DUT and the DUT stops all packets with wrong parity/CRC. Which packets that are lost is unpredictable.