# Bitvis Utility Library – Quick Reference

## Checks and awaits

[v_bool :=] **check_value**(value, [exp], alert_level, msg, [scope], [radix], [format], [msg_id],[msg_id_panel])

[v_bool :=] **check_value_in_range**(value, min_value, max_value, alert_level, msg, [scope], [msg_id], [msg_id_panel])

**check_stable**(target, stable_req, alert_level, msg, [scope], [msg_id], [msg_id_panel])

**await_change**(target, min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel])

**await_value**(target, exp, min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel])

## Logging and verbosity control

**set_log_file_name**(file_name)

**log**(msg_id, msg, [scope], [msg_id_panel])

**enable_log_msg**(msg_id, [msg_id_panel], [msg])

**disable_log_msg** (msg_id, [msg_id_panel], [msg]),

## Alert handling

**set_alert_file_name**(file_name)

**alert**(alert_level, msg, scope)

**[tb_]note**(msg, [scope])

**[tb_]warning**(msg, [scope])

**manual_check**(msg, [scope])

**[tb_]error**(msg, [scope])

**[tb_]failure**(msg, [scope])

**set_alert_stop_limit**(alert_level, limit)

v_int := **get_alert_stop_limit**(alert_level)

**set_alert_attention**(alert_level, attention)

v_attention := **get_alert_attention**(alert_level)

**increment_expected_alerts**(alert_level, number)

## Reporting

**report_global_ctrl**(VOID)

**report_msg_id_panel**(VOID)

**report_alert_counters**(VOID)

## String handling

v_string := **to_string**(val, [width, justified, format] [radix, format, prefix])

v_string := **justify**(val, [width], [justified], [format])

v_string := **fill_string**(val, width)

v_string := **to_upper**(val)

v_character **:= ascii_to_char**(ascii_pos, [ascii_allow])

v_int **:= char_to_ascii**(character)

## Randomization

v_slv := **random**(length)

v_sl := **random**(VOID)

v_int := **random**(min_value, max_value)

v_real := **random**(min_value, max_value)

v_time := **random**(min_value, max_value)

**random**([min_value, max_val], v_seed1, v_seed2, v_target)

**randomise**(seed1, seed2);

## Signal generators

**clock_generator** (clock, [clock_ena], clk_period, [clock_name])

**gen_pulse** (target, pulse_duration, [mode]) *or* (target, clock_signal, num_periods)

## BFM Common Package

**normalise** (value, target, mode, value_name, target_name, msg)

**wait_until_given_time_after_rising_edge**(clk, wait_time)

## Some Common Msg IDs

| | |
|---|---|
| ID_LOG_HDR | ID_PACKET_INITIATE |
| ID_SEQUENCER | ID_PACKET_COMPLETE |
| ID_SEQUENCER_SUB | ID_PACKET_HDR |
| ID_POS_ACK | ID_PACKET_DATA |
| ID_BFM | ID_LOG_MSG_CTRL |
| ID_BFM_WAIT | ALL_MESSAGES |

# 1 Method descriptions

*Note 1: Arguments common for most methods (green text) are described in chapter 1.9*

*Note 2: All methods are defined in bitvis_util.methods_pkg, unless otherwise noted.*

*Legend: bool=boolean, sl=std_logic, slv=std_logic_vector, u=unsigned, s=signed, int=integer*
**IEEE=Method is included in the ieee_proposed library for VHDL93/2002 and is native for VHDL2008 (Method is listed here for completeness.)*

## 1.1 Checks and awaits

| Name | Parameters | Description |
|---|---|---|
| check_value() | val(bool), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>val(sl), exp(sl), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>val(slv), exp(slv), alert_level, msg, [scope], [radix], [format], [msg_id], [msg_id_panel]<br>val(u), exp(u), alert_level, msg, [scope], [radix], [format], [msg_id], [msg_id_panel]<br>val(s), exp(s), alert_level, msg, [scope], [radix], [format], [msg_id], [msg_id_panel]<br>val(int), exp(int), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>val(time), exp(time), alert_level, msg, [scope], [msg_id], [msg_id_panel] | Checks if *val* equals *exp*, and alerts with severity *alert_level* if the values do not match.<br>The result of the check is returned as a boolean if the method is called as a function.<br>If *val* is of type *slv*, *unsigned* or *signed*, there are additional optional arguments:<br>- *radix* : for the vector representation in the log: BIN, HEX, DEC or HEX_BIN_IF_INVALID. (HEX_BIN_IF_INVALID means hexadecimal, unless there are the vector contains any U, X, Z or W, - in which case it is also logged in binary radix.)<br>- *format* may be AS_IS or SKIP_LEADING_0. Controls how the vector is formatted in the log. |
| check_value_in_range() | val(u), min_value(u), max_value(u), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>val(s), min_value(s), max_value(s), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>val(int), min_value(int), max_value(int), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>val(time), min_value(time), max_value(time), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>val(real), min_value(real), max_value(real), alert_level, msg, [scope], [msg_id], [msg_id_panel] | Checks if *min_value* ≤ *val* ≤ *max_value*, and alerts with severity *alert_level* if *val* is outside the range.<br>The result of the check is returned as a boolean if the method is called as a function. |
| check_stable() | target(bool), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(sl), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(slv), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(u), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(s), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(int), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel] | Checks if the *target* signal has been stable in *stable_req* time.<br>If not, an alert is asserted. |
| await_change() | target(bool), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(sl), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(slv), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(u), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(s), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel]<br>target(int), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel] | Waits until the *target* signal changes, or times out after *max_time*.<br>An alert is asserted if the signal does not change between *min_time* and *max_time*.<br>Note that if the value changes at exactly *max_time*, the timeout gets precedence. |
| await_value() | target(bool), exp(bool), min_time, max_time, alert_level, msg, [scope], (etc.)<br>target(sl), exp(sl), min_time, max_time, alert_level, msg, [scope], (etc.) | Waits until the *target* signal equals the *exp* signal, or times out after *max_time*.<br>An alert is asserted if the signal does not equal the expected value between *min_time* and |

| | | |
|---|---|---|
| | target(slv), exp(slv), min_time, max_time, alert_level, msg, [scope], (etc.)<br>target(u), exp(u), min_time, max_time, alert_level, msg, [scope], (etc.)<br>target(s), exp(s), min_time, max_time, alert_level, msg, [scope], (etc.)<br>target(int), exp(int), min_time, max_time, alert_level, msg, [scope], (etc.) | *max_time*.<br><br>Note that if the value changes to the expected value at exactly *max_time*, the timeout gets precedence. |

## 1.2 Logging and verbosity control

| Name | Parameters | Description |
|---|---|---|
| set_log_file_name() | file_name(string) | Sets the log file name. NOTE: Must be set prior the first report, log and alert message. |
| log() | msg_id(t_msg_id), msg(string) , [scope](string), [msg_id_panel(t_msg_id_panel)] | If the *msg_id* is enabled in *msg_id_panel*, log the *msg* to STDOUT and the log file. |
| enable_log_msg () | msg_id(t_msg_id), [msg_id_panel(t_msg_id_panel)], [msg] | Enables logging for the given *msg_id*. (See ID-list on front page for special purpose IDs) |
| disable_log_msg() | msg_id(t_msg_id), [msg_id_panel(t_msg_id_panel)], [msg] | Disables logging for the given *msg_id*. (See ID-list on front page for special purpose IDs) |

## 1.3 Alerts

| Name | Parameters | Description |
|---|---|---|
| set_alert_file_name() | file_name(string) | Sets the alert file name . NOTE: Must be set prior the first report, log and alert message. |
| alert() | alert_level(t_alert_level), msg(string) , [scope](string) | - Asserts an alert with severity given by *alert_level*.<br>- Increment the counters for the given *alert_level.*<br>- If the stop_limit for the given *alert_level* is reached, stop the simulation. |
| note() error()<br>tb_note() tb_error()<br>warning() failure()<br>tb_warning() tb_failure()<br>manual_check() | msg(string), [scope](string) | Overloads for alert().<br>Nothe that: warning(msg, [scope]) = alert(warning, msg, [scope]). |
| increment_expected_alerts() | alert_level (t_alert_level), number (natural)) | Increments the expected alert counter for the given alert_level. |

## 1.4 Reporting

| Name | Parameters | Description |
|---|---|---|
| report_global_ctrl() | VOID | Logs the values in the global_ctrl signal, which is described in chapter 1.11 |
| report_msg_id_panel() | VOID | Logs the values in the msg_id_panel, which is described in chapter 1.11 |
| report_alert_counters() | VOID | Logs the status of all alert counters, typically at the end of simulation.<br>For each alert_level, the alert counter is compared with the expected counter. |

## 1.5 String handling

*(Methods are defined in bitvis_util.string_methods and in ieee_proposed.standard_additions_c)*

| Name | Parameters | Description |
|---|---|---|
| to_string() | val(bool), width(natural), [justified(side)], [format(t_format_string)]<br>val(int), width(natural), [justified(side)], [format(t_format_string)]<br>val(sl) *IEEE<br>val(slv), radix(t_radix), [format(t_format_zeros)], [prefix(t_radix_prefix)]<br>val(u), radix(t_radix), [format(t_format_zeros)], [prefix(t_radix_prefix)]<br>val(s), radix(t_radix), [format(t_format_zeros)], [prefix(t_radix_prefix)]<br>val(real), [format(string)] *IEEE | Return a *string* with the value of the argument 'val'.<br>- type t_radix is (BIN, HEX, DEC, HEX_BIN_IF_INVALID)<br>- type t_format_string is (AS_IS, TRUNCATE, SKIP_LEADING_SPACE)<br>- type t_format_zeros is (AS_IS, SKIP_LEADING_0)<br>- type t_radix_prefix is (EXCL_RADIX, INCL_RADIX) |
| to_upper() | val(string) | Returns a *string* containing an upper case version of the argument 'val' |
| justify() | val(string), width(natural), [justified(side)], [format(t_format_string)] | Returns a *string* where 'val' is justified to the side given by 'justified' (right, left). |
| fill_string() | val(character), width(natural) | Returns a *string* filled with the character 'val'. |
| ascii_to_char() | ascii_pos(int), [ascii_allow (t_ascii_allow)] | Return the ASCII to character located at the argument 'ascii_pos'<br>- type t_ascii_allow is (ALLOW_ALL (default), ALLOW_PRINTABLE_ONLY) |
| char_to_ascii() | char (character) | Return the ASCII value (integer) of the argument 'char' |

## 1.6 Randomisation

| Name | Parameters | Description |
|---|---|---|
| random() | length(int) | Returns a random std_logic_vector of size *length*. The function uses and updates a global seed |
| random() | VOID | Returns a random std_logic. The function uses and updates a global seed |
| random() | min_value(int), max_value(int)<br>min_value(real), max_value(real)<br>min_value(time), max_value(time) | Returns a random *integer*, *real* or *time* between min_value and max_value.<br>The function uses and updates a global seed |
| random() | v_seed1(positive *variable*), v_seed2(positive *variable*), v_target(slv *variable*) | Sets v_target to a random value. The procedure uses and updates v_seed1 and v_seed2. |
| random() | min_value(int), max_value(int), v_seed1(positive *var*), v_seed2(positive *var*), v_target(int *var*)<br>min_value(real), max_value(real), v_seed1(positive *var*), v_seed2(positive *var*), v_target(real *var*)<br>min_value(time), max_value(time), v_seed1(positive *var*), v_seed2(positive *var*), v_target(time *var*) | Sets v_target to a random value between min_value and max_value.<br>The procedure uses and updates v_seed1 and v_seed2. |
| randomise() | seed1(positive), seed2(positive) | Sets the global seeds to *seed1* and *seed2*. |

## 1.7 Signal generators

| Name | Parameters | Description |
|---|---|---|
| clock_generator() | clock_signal(sl), clock_period(time)<br>clock_signal(sl), clock_ena(boolean), clock_period(time), clock_name(string) | Generates a clock signal.<br>Usage: Include the the clock_generator as a concurrent procedure from your test bench.<br>By using the variant with the *clock_ena* input, the clock can be started and stopped during simulation. Each start/stop is logged (if the msg_id ID_CLOCK_GEN is enabled). |
| gen_pulse() | target(sl), pulse_duration(time), [mode(t_mode)], msg, [scope], [msg_id], [msg_id_panel]<br>target(sl), clock_signal(sl), num_periods(int), msg, [scope], [msg_id], [msg_id_panel]<br>target(slv), pulse_value(slv), clock_signal(sl), num_periods(int), msg, [scope], [msg_id], [msg_id_panel] | Generates a pulse on the target signal for a certain amount of time or a number of clock cycles.<br>- If mode = BLOCKING (default): Procedure blocks the caller (f.ex the test sequencer) until the pulse is done.<br>- If mode = NON_BLOCKING : Procedure starts the pulse and schedules the end of the pulse, so that the caller can continue immediately. |

## 1.8 BFM Common package

*(Methods are defined in bitvis_util.bfm_common_pkg)*

| Name | Parameters | Description |
|---|---|---|
| normalise() | value(slv), target(slv), mode (t_normalisation_mode), value_name, target_name, msg,<br>value(u), target (u), mode (t_normalisation_mode), value_name, target_name, msg,<br>value(s), target (s), mode (t_normalisation_mode), value_name, target_name, msg, | Normalise 'value' to the width given by 'target'.<br>If value'length > target'length, remove leading zeros (or sign bits) from value<br>If value'length < target'length, add padding (leading zeros, or sign bits) to value<br>Mode (t_normalisation_mode) is used for sanity checks, and can be one of :<br>ALLOW_WIDER : Allow only value'length > target'length<br>ALLOW_NARROWER : Allow only value'length < target'length<br>ALLOW_WIDER_NARROWER : Allow both of the above<br>ALLOW_EXACT_ONLY : Allow only value'length = target'length |
| wait_until_given_time_after_rising_edge | clk(sl), wait_time | Wait until wait_time after rising_edge(clk)<br>If the time passed since the previous rising_edge is less than wait_time,<br>don't wait until the next rising_edge, just wait_time after the previous rising_edge. |

## 1.9 Message IDs

*A sub set of message IDs is listed in this table. All the message IDs are defined in bitvis_util.adaptations_pkg.*

| Message ID | Description |
|---|---|
| ID_LOG_HDR | For all test sequencer log headers. Special format with preceding empty line and underlined message |
| ID_SEQUENCER | For all other test sequencer messages |
| ID_SEQUENCER_SUB | For general purpose procedures defined inside TB and called from test sequencer |
| ID_POS_ACK | A general positive acknowledge for check routines (incl. awaits) |
| ID_BFM | BFM operation (e.g. message that a write operation is completed) (BFM: Bus Functional Model, basically a procedure to handle a physical interface) |
| ID_BFM_WAIT | Typically BFM is waiting for response (e.g. waiting for ready, or predefined number of wait states) |
| ID_PACKET_INITIATE | A packet has been initiated (Either about to start or just started) |
| ID_PACKET_COMPLETE | Packet completion |
| ID_PACKET_HDR | Packet header information |
| ID_PACKET_DATA | Packet data information |
| ID_LOG_MSG_CTRL | Dedicated ID for enable/disable_log_msg |
| ID_CLOCK_GEN | Used for logging when clock generators are enabled or disabled |
| ID_GEN_PULSE | Used for logging when a gen_pulse procedure starts pulsing a signal |
| ALL_MESSAGES | Not an ID. Applies to all IDs (apart from ID_NEVER) |
| ID_NEVER | Not intended for normal use. Only to be used inside procedures/functions that are using log-mechanisms, but where a log is never wanted. |

## 1.10 Common arguments in checks and awaits

Most check and await methods have two groups of arguments:

- arguments specific to this function/procedure
- **common_args**: arguments common for all functions/procedures:
  - alert_level, msg, [scope], [msg_id], [msg_id_panel]

For example: check_value(*val*, *exp*, ERROR, "Check that the val signal equals the exp signal", C_SCOPE);

The common arguments are described in the following table.

| Argument | Type | Example | Description |
|---|---|---|---|
| alert_level | t_alert_level; | ERROR | Set the severity for the alert that may be asserted by the method. |
| msg | string; | “Check that bus is stable” | A custom message to be appended in the log/alert. |
| scope | string; | "TB Sequencer" | A string describing the scope from which the log/alert originates. |
| msg_id | t_msg_id | ID_BFM | Optional message ID, defined in the adaptations package.<br>Default value for check routines = ID_POS_ACK; |
| msg_id_panel | t_msg_id_panel | local_msg_id_panel | Optional msg_id_panel, controlling verbosity within a specified scope.<br>Defaults to a common ID panel defined in the adaptations package. |

### 1.11 Adaptation package

The `adaptations_pkg.vhd` is intended for local modifications to library behaviour and log layout.
This way only one file needs to merged when a new versions of the library is released.
This package may of course also be used to set up a company or project specific behaviour and layout.

## 2 Additional Documentation

There are two other main documents for the Bitvis Utility Library (available from our Downloads page)

- 'Making a simple, structured and efficient VHDL testbench – Step-by-step'
- 'Bitvis Utility Library – Concepts and Usage'

There is also a webinar available on 'Making a simple, structured and efficient VHDL testbench – Step-by-step' (via Aldec). Link on our downloads page.

# 3 Compilation

Bitvis Utility Library may be compiled with VHDL 2008, VHDL 2002 or VHDL 93.
The 2008 and 2002 version use protected types that allow safe update of shared variables.
To minimise the difference (and thus reduce maintenance overhead) ordinary shared variables have been used wherever this is acceptable from a functionality point of view (i.e. the tools may yield warnings). The 2008 or 2002 version is recommended as alert counters here are guaranteed to be correct, whereas there is a very small probability that two alerts may be counted as one in the 93 version.

The 2002 and 93 versions use the ieee_proposed library, which allows 2008-functionality to be used in simulators not supporting 2008.

Compile order for Bitvis Utility Library:

| Compile to library | File | Comment |
|---|---|---|
| ieee_proposed | x_ieee_proposed/src/standard_additions_c.vhdl | 93/2002 version only |
| ieee_proposed | x_ieee_proposed/src/standard_textio_additions_c.vhdl | 93/2002 version only |
| ieee_proposed | x_ieee_proposed/src/std_logic_1164_additions.vhdl | 93/2002 version only |
| ieee_proposed | x_ieee_proposed/src/numeric_std_additions.vhdl | 93/2002 version only |
| bitvis_util | bitvis_util/src*/types_pkg.vhd | |
| bitvis_util | bitvis_util/src*/adaptations_pkg.vhd | |
| bitvis_util | bitvis_util/src*/string_methods_pkg.vhd | |
| bitvis_util | bitvis_util/src*/protected_types_pkg.vhd | 2002/2008 version only |
| bitvis_util | bitvis_util/src*/vhdl_version_layer_pkg.vhd | |
| bitvis_util | bitvis_util/src*/license_open_pkg.vhd | |
| bitvis_util | bitvis_util/src*/methods_pkg.vhd | |

Modelsim users can compile the libraries by sourcing the following files:
`script/compile_dep.do (compiles ieee_proposed. Only needed for the vhdl93 and vhdl2002 version of Bitvis Util)`
`script/compile_src<version>.do` , where <version> = 93, 2002 or 2008

Note that the compile script compiles the Utility Library with the following Modelsim directives for the vcom command:

| Directive | Description |
|---|---|
| -suppress 1346,1236 | Suppress warnings about the use of shared variables (93 version) or protected types (2002/2008 version). These can be ignored. |

The bitvis_util project is opened by opening `sim/bitvis_util.mpf` in Modelsim.

# 4 Simulator compatibility and setup

Bitvis Utility Library has been compiled and tested with Modelsim and Active HDL.
The VHDL 93 version should also support other simulators (like Xilinx ISim and Vivado Simulator).
Required setup:

- Textio buffering should be removed or reduced. (Modelsim.ini: Set UnbufferedOutput to 1)
- Simulator transcript (and log file viewer) should be set to a fixed width font type for proper alignment (e.g. Courier New 8)
- Simulator must be set up to break the simulation on failure (or lower severity)