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])

await_stable(target, stable_req, stable_req_from, timeout, timeout_from, alert_level, msg, [scope], [msg_id], [msg_id_panel])

Logging and verbosity control

set_log_file_name(file_name, [msg_id])

log(msg_id, msg, [scope], [msg_id_panel], [log_destination], [log_file_name], [open_mode])

log_text_block(msg_id, text_block, formatting, [msg_header], [scope], [msg_id_panel], [other optional parameters])

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

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

is_log_msg_enabled (msg_id, [msg_id_panel])

Alert handling

set_alert_file_name(file_name, [msg_id])

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, [msg])

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)

v_natural := **pos_of_leftmost**(character, string, [return-value if not found])

v_natural := **pos_of_rightmost**(character, string, [return-value if not found])

v_string := **remove_initial_chars**(string, number of chars(natural))

v_string := **get_procedure_name_from_instance_name**(string)

v_string := **get_process_name_from_instance_name**(string)

v_string := **get_entity_name_from_instance_name**(string)

v_string := **replace**(string, target_character, exchange_character)

replace(inout line, target_character, exchange_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)

randomize(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

normalize_and_check (value, target, mode, value_name, target_name, msg)

wait_until_given_time_after_rising_edge(clk, wait_time)

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
[v_bool :=] check_value()	val(bool), alert_level, msg, [scope], [msg_id], [msg_id_panel] val(bool), exp(bool), alert_level, msg, [scope], [msg_id], [msg_id_panel] val(sl), exp(sl), alert_level, msg, [scope], [msg_id], [msg_id_panel] val(slv), exp(slv), alert_level, msg, [scope], [radix], [format], [msg_id], [msg_id_panel] val(u), exp(u), alert_level, msg, [scope], [radix], [format], [msg_id], [msg_id_panel] val(s), exp(s), alert_level, msg, [scope], [radix], [format], [msg_id], [msg_id_panel] val(int), exp(int), alert_level, msg, [scope], [msg_id], [msg_id_panel] 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. The result of the check is returned as a boolean if the method is called as a function. If *val* is of type *slv*, *unsigned* or *signed*, there are additional optional arguments: - *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.) - *format* may be AS_IS or SKIP_LEADING_0. Controls how the vector is formatted in the log. **Defaults**: *scope*<=C_TB_SCOPE_DEFAULT, *radix*<=HEX_BIN_IF_INVALID, *format*<=SKIP_LEADING_0, *msg_id*<=ID_POS_ACK, *msg_id_panel*<=shared_msg_id_panel
[v_bool :=] check_value_in_range()	val(u), min_value(u), max_value(u), alert_level, msg, [scope], [msg_id], [msg_id_panel] val(s), min_value(s), max_value(s), alert_level, msg, [scope], [msg_id], [msg_id_panel] val(int), min_value(int), max_value(int), alert_level, msg, [scope], [msg_id], [msg_id_panel] val(time), min_value(time), max_value(time), alert_level, msg, [scope], [msg_id], [msg_id_panel] 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. The result of the check is returned as a boolean if the method is called as a function. **Defaults**: *scope*<=C_TB_SCOPE_DEFAULT, *msg_id*<=ID_POS_ACK, *msg_id_panel*<=shared_msg_id_panel
check_stable()	target(bool), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel] target(sl), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel] target(slv), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel] target(u), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel] target(s), stable_req(time), alert_level, msg, [scope], [msg_id], [msg_id_panel] 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. If not, an alert is asserted. **Defaults**: *scope*<=C_TB_SCOPE_DEFAULT, *msg_id*<=ID_POS_ACK, *msg_id_panel*<=shared_msg_id_panel
await_change()	target(bool), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel] target(sl), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel] target(slv), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel] target(u), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel] target(s), min_time, max_time, alert_level, msg, [scope], [msg_id], [msg_id_panel] 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*. An alert is asserted if the signal does not change between *min_time* and *max_time*. Note that if the value changes at exactly *max_time*, the timeout gets precedence. **Defaults**: *scope*<=C_TB_SCOPE_DEFAULT, *msg_id*<=ID_POS_ACK, *msg_id_panel*<=shared_msg_id_panel
await_value()	target(bool), exp(bool), min_time, max_time, alert_level, msg, [scope], (etc.) target(sl), exp(sl), min_time, max_time, alert_level, msg, [scope], (etc.) target(slv), exp(slv), min_time, max_time, alert_level, msg, [scope], (etc.) target(u), exp(u), min_time, max_time, alert_level, msg, [scope], (etc.) target(s), exp(s), min_time, max_time, alert_level, msg, [scope], (etc.) target(int), exp(int), min_time, max_time, alert_level, msg, [scope], (etc.)	Waits until the *target* signal equals the *exp* signal, or times out after *max_time*. An alert is asserted if the signal does not equal the expected value between *min_time* and *max_time*. Note that if the value changes to the expected value at exactly *max_time*, the timeout gets precedence. **Defaults**: *scope*<=C_TB_SCOPE_DEFAULT, *msg_id*<=ID_POS_ACK, *msg_id_panel*<=shared_msg_id_panel

await_stable()	target(bool), stable_req(time), stable_req_from(t_from_point_in_time), timeout (time), timeout_from(t_from_point_in_time), alert_level, msg, [scope],(etc.)
	target(sl), stable_req(time), stable_req_from(t_from_point_in_time), timeout (time), timeout_from(t_from_point_in_time), alert_level, msg, [scope],(etc.)
	target(slv), stable_req(time), stable_req_from(t_from_point_in_time), timeout (time), timeout_from(t_from_point_in_time), alert_level, msg, [scope],(etc.)
	target(u), stable_req(time), stable_req_from(t_from_point_in_time), timeout (time), timeout_from(t_from_point_in_time), alert_level, msg, [scope],(etc.)
	target(s), stable_req(time), stable_req_from(t_from_point_in_time), timeout (time), timeout_from(t_from_point_in_time), alert_level, msg, [scope],(etc.)
	target(int), stable_req(time), stable_req_from(t_from_point_in_time), timeout (time), timeout_from(t_from_point_in_time), alert_level, msg, [scope],(etc.)
		Wait until the target signal has been stable for at least 'stable_req'.
		Report an error if this does not occurr within the time specified by 'timeout'.
		Note : 'Stable' refers to that the signal has not had an event (i.e. not changed value).
		Description of special arguments:
		`stable_req_from :`
		`- FROM_NOW : Target must be stable 'stable_req' from now`
		`- FROM_LAST_EVENT : Target must be stable 'stable_req' from the last event of target.`
		`timeout_from :`
		`- FROM_NOW : The timeout argument is given in time from now`
		`- FROM_LAST_EVENT : The timeout argument is given in time the last event of target.`
		Defaults: *scope*<=C_TB_SCOPE_DEFAULT, *msg_id*<=ID_POS_ACK, *msg_id_panel*<=shared_msg_id_panel

1.2 Logging and verbosity control

Name	Parameters	Description
set_log_file_name()	file_name(string), [msg_id(t_msg_id)]	Sets the log file name. NOTE: Must be set prior to any other call except set_alert_file_name(). **Defaults**: *file_name*<=C_LOG_FILE_NAME, *msg_id*<=C_UTIL_SETUP
log()	msg_id(t_msg_id), msg(string), [scope(string)], [msg_id_panel(t_msg_id_panel)], [log_destination(t_log_destination)], [log_file_name(string)], [open_mode(file_open_kind)]	Writes message to log. If the *msg_id* is enabled in *msg_id_panel*, log the *msg*. Log destination defines where the message will be written to (CONSOLE_AND_LOG, CONSOLE_ONLY, LOG_ONLY). Log file name defines the log file that the text block shall be written to. open_mode indicates how the log file shall be opened (write_mode, append_mode). **Defaults**: *scope*<=C_TB_SCOPE_DEFAULT, *msg_id_panel*<=shared_msg_id_panel, *log_destination*<=CONSOLE_AND_LOG, *log_file_name*<=C_LOG_FILE_NAME, *open_mode*<=append_mode
log_text_block()	msg_id(t_msg_id), text_block(line), formatting(t_log_format), [msg_header(string)], [scope(string)], [msg_id_panel(t_msg_id_panel)], [log_if_block_empty(t_log_if_block_empty)], [log_destination(t_log_destination)], [log_file_name(string)], [open_mode(file_open_kind)]	Writes text block from VHDL line to log. Formatting either FORMATTED or UNFORMATTED. msg_header is an optional header message for the text_block. log_if_block_empty defines how an empty text block is handled (WRITE_HDR_IF_BLOCK_EMPTY/SKIP_LOG_IF_BLOCK_EMPTY/NOTIFY_IF_BLOCK_EMPTY). Log destination defines where the message will be written to (CONSOLE_AND_LOG, CONSOLE_ONLY, LOG_ONLY). Log file name defines the log file that the text block shall be written to. open_mode indicates how the log file shall be opened (write_mode, append_mode). **Defaults**: *msg_header*<="", *scope*<=C_TB_SCOPE_DEFAULT, *msg_id_panel*<=shared_msg_id_panel, log_if_block_empty<=WRITE_HDR_IF_BLOCK_EMPTY, *log_destination*<=CONSOLE_AND_LOG, *log_file_name*<=C_LOG_FILE_NAME, *open_mode*<=append_mode
enable_log_msg ()	msg_id(t_msg_id), [msg_id_panel(t_msg_id_panel)], [msg] , [scope])	Enables logging for the given *msg_id*. (See ID-list on front page for special purpose IDs) **Defaults**: *msg_id_panel*<=shared_msg_id_panel, *msg*<="", *scope*<=C_TB_SCOPE_DEFAULT
disable_log_msg()	msg_id(t_msg_id), [msg_id_panel(t_msg_id_panel)], [msg] , [scope], [quietness(t_quietness)])	Disables logging for the given *msg_id*. (See ID-list on front page for special purpose IDs). Logging of disable_log_msg() can be turned off by setting quietness=QUIET. **Defaults**: *msg_id_panel*<=shared_msg_id_panel, *msg*<="", *scope*<=C_TB_SCOPE_DEFAULT, *quietness*<=NON_QUIET
is_log_msg_enabled ()	msg_id(t_msg_id), [msg_id_panel(t_msg_id_panel)]	Returns Boolean 'true' if given message ID is enabled. Otherwise 'false' **Defaults**: *msg_id_panel*<=shared_msg_id_panel

1.2.1 General string handling features for log()

- All log messages will be given using the user defined layout in adaptations_pkg.vhd
- \n and \r may be used to force line shifts (\r: blank line apart from prefix. \n: Line shift after scope column, before message column)

1.3 Alerts

Name	Parameters	Description
set_alert_file_name()	file_name(string), [msg_id(t_msg_id)]	Sets the alert file name. NOTE: Must be set prior to any other call except set_log_file_name(). **Defaults**: *file_name*<=C_ALERT_FILE_NAME, *msg_id*<=ID_UTIL_SETUP
alert()	alert_level(t_alert_level), msg(string) , [scope](string)	- Asserts an alert with severity given by *alert_level*. - Increment the counters for the given *alert_level.* - If the stop_limit for the given *alert_level* is reached, stop the simulation. **Defaults**: *scope* <=C_TB_SCOPE_DEFAULT
note() error() tb_note() tb_error() warning() failure() tb_warning() tb_failure() manual_check()	msg(string), [scope](string)	Overloads for alert(). Note that: warning(msg, [scope]) = alert(warning, msg, [scope]). **Defaults**: *scope* <=C_TB_SCOPE_DEFAULT
increment_expected_alerts()	alert_level (t_alert_level), [number (natural)] , [msg(string)] , [scope(string)]	Increments the expected alert counter for the given alert_level. **Defaults**: *number*<=1,*msg*<="", *scope* <=C_TB_SCOPE_DEFAULT
set_alert_stop_limit()	alert_level (t_alert_level), number (natural)	Simulator will stop on hitting <number> of specified alert type (0 means never stop).
v_int := get_alert_stop_limit()	alert_level (t_alert_level)	Returns current stop limit for given alert type.
set_alert_attention()	alert_level (t_alert_level), attention (IGNORE or REGARD), [msg(string)]	Set given alert type to IGNORE or REGARD. **Defaults**: *msg* <=""
v_attention := get_alert_attention()	alert_level (t_alert_level)	Returns current attention (IGNORE or REGARD) for given alert type.

1.4 Reporting

Name	Parameters	Description
report_global_ctrl()	VOID	Logs the values in the global_ctrl signal, which is described in chapter 0
report_msg_id_panel()	VOID	Logs the values in the msg_id_panel, which is described in chapter 0
report_alert_counters()	VOID	Logs the status of all alert counters, typically at the end of simulation. 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
v_string := to_string()	val(bool), width(natural), [justified(side)], [format(t_format_string)] val(int), width(natural), [justified(side)], [format(t_format_string)] val(sl) *IEEE val(slv), radix(t_radix), [format(t_format_zeros)], [prefix(t_radix_prefix)] val(u), radix(t_radix), [format(t_format_zeros)], [prefix(t_radix_prefix)] val(s), radix(t_radix), [format(t_format_zeros)], [prefix(t_radix_prefix)] val(real), [format(string)] *IEEE val(string) -- Removes non printable ascii characters	Return a *string* with the value of the argument 'val'. - type t_radix is (BIN, HEX, DEC, HEX_BIN_IF_INVALID) - type t_format_string is (AS_IS, TRUNCATE, SKIP_LEADING_SPACE) - type t_format_zeros is (AS_IS, SKIP_LEADING_0) - type t_radix_prefix is (EXCL_RADIX, INCL_RADIX) **Defaults**: *justified*<=RIGHT, *format*<=AS_IS, *prefix*<=EXCL_RADIX
v_string := to_upper()	val(string)	Returns a *string* containing an upper case version of the argument 'val'
v_string := 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). **Defaults**: *width*<=0, *justified*<=RIGHT, *format*<=AS_IS
v_string := fill_string()	val(character), width(natural)	Returns a *string* filled with the character 'val'.
v_character := ascii_to_char()	ascii_pos(int), [ascii_allow (t_ascii_allow)]	Return the ASCII to character located at the argument 'ascii_pos' - type t_ascii_allow is (ALLOW_ALL, ALLOW_PRINTABLE_ONLY) **Defaults**: *ascii_allow*<=ALLOW_ALL
v_int := char_to_ascii()	char (character)	Return the ASCII value (integer) of the argument 'char'
v_natural := pos_of_leftmost()	character, string, [return-value if not found (natural)])	Returns position of left most 'character' in 'string', alternatively return-value if not found **Defaults:** *result_if_not_found*<=1
v_natural := pos_of_rightmost()	character, string, [return-value if not found (natural)])	Returns position of right most 'character' in 'string', alternatively return-value if not found **Defaults:** *result_if_not_found*<=1
v_string := remove_initial_chars()	string, number of chars(natural)	Return string less the N (number of chars) first characters
v_string := get_procedure_name_from_instance_name()	String	Returns procedure, process or entity name from the given instance name as *string*. The instance name must be <object>'instance_name, where object is a signal, variable or constant defined in the procedure, process and entity/process respectively e.g. get_entity_name_from_instance_name(my_process_variable'instance-name)
v_string := get_process_name_from_instance_name()	String	
v_string := get_entity_name_from_instance_name()	String	
v_string := replace()	val(string), target_char(character), exchange_char(character)	String function returns a *string* where the target character has been replaced by the exchange character.
replace()	variable text_line(inout line), target_char(character), exchange_char(character)	Similar to function version of replace(). Line procedure replaces the input with a line where the target character has been replaced by the exchange character.

Note: See section 1.2.1 for general string handling features for the log() procedure

1.6 Randomization

Name	Parameters	Description
v_slv := random()	length(int)	Returns a random std_logic_vector of size *length*. The function uses and updates a global seed.
v_sl := random()	VOID	Returns a random std_logic. The function uses and updates a global seed
{v_int,v_real,v_time} := random()	min_value(int), max_value(int) min_value(real), max_value(real) min_value(time), max_value(time)	Returns a random *integer*, *real* or *time* between min_value and max_value. 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*) min_value(real), max_value(real), v_seed1(positive *var*), v_seed2(positive *var*), v_target(real *var*) 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. The procedure uses and updates v_seed1 and v_seed2.
randomize()	seed1(positive), seed2(positive) , [msg(string)] , [scope(string)])	Sets the global seeds to *seed1* and *seed2*.

1.7 Signal generators

Name	Parameters	Description
clock_generator()	clock_signal(sl), clock_period(time) clock_signal(sl), clock_ena(boolean), clock_period(time), clock_name(string)	Generates a clock signal. Usage: Include the the clock_generator as a concurrent procedure from your test bench. 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] target(sl), clock_signal(sl), num_periods(int), msg, [scope], [msg_id], [msg_id_panel] 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. - If mode = BLOCKING: Procedure blocks the caller (f.ex the test sequencer) until the pulse is done. - If mode = NON_BLOCKING : Procedure starts the pulse and schedules the end of the pulse, so that the caller can continue immediately. **Defaults**: *scope*<=C_TB_SCOPE_DEFAULT, *msg_id*<=ID_GEN_PULSE, *msg_id_panel*<=shared_msg_id_panel

1.8 BFM Common package

(Methods are defined in bitvis_util.bfm_common_pkg)

Name	Parameters	Description
normalize_and_check()	value(slv), target(slv), mode (t_normalization_mode), value_name, target_name, msg value(u), target (u), mode (t_normalization_mode), value_name, target_name, msg value(s), target (s), mode (t_normalization_mode), value_name, target_name, msg	Normalize 'value' to the width given by 'target'. If value'length > target'length, remove leading zeros (or sign bits) from value If value'length < target'length, add padding (leading zeros, or sign bits) to value Mode (t_normalization_mode) is used for sanity checks, and can be one of : ALLOW_WIDER : Allow only value'length > target'length ALLOW_NARROWER : Allow only value'length < target'length ALLOW_WIDER_NARROWER : Allow both of the above 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) If the time passed since the previous rising_edge is less than wait_time, 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 (also applies to ID_LOG_HDR_LARGE and ID_LOG_HDR_XL).
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_BFM_POLL	Used inside a BFM when polling until reading a given value, i.e., to show all reads until expected value found.
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)

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. 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. 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.
The layout constants and global signals are described in the following tables.

Constant	Description
C_ALERT_FILE_NAME	Name of the alert file.
C_LOG_FILE_NAME	Name of the log file.
C_SHOW_BITVIS_UTILITY_LIBRARY_INFO	General information about the Bitvis Utility Library will be shown when this is enabled.
C_SHOW_BITVIS_UTILITY_LIBRARY_RELEASE_INFO	Release information will be shown when this is enabled.
C_LOG_PREFIX	The prefix to all log messages. "Bitvis: " by default.
C_LOG_PREFIX_WIDTH	Number of characters to be used for the log prefix.
C_LOG_MSG_ID_WIDTH	Number of characters to be used for the message ID.
C_LOG_TIME_WIDTH	Number of characters to be used for the log time. Three characters are used for time unit, .e.g., ' ns'.
C_LOG_TIME_BASE	The unit in which time is shown in the log. Either ns or ps.
C_LOG_TIME_DECIMALS	Number of decimals to show for the time.
C_LOG_SCOPE_WIDTH	Number of characters to be used to show log scope.
C_LOG_LINE_WIDTH	Number of characters allowed in each line in the log.
C_LOG_INFO_WIDTH	Number of characters of information allowed in each line in the log. By default this is set to C_LOG_LINE_WIDTH – C_LOG_PREFIX_WIDTH.
C_LOG_HDR_FOR_WAVEVIEW_WIDTH	Number of characters for a string in the waveview indicating last log header.
C_USE_BACKSLASH_N_AS_LF	If true '\n' will be interpreted as line feed.
C_SINGLE_LINE_ALERT	If true prints alerts on a single line. Default false.
C_SINGLE_LINE_LOG	If true prints logs messages on a single line. Default false.
C_TB_SCOPE_DEFAULT	The default scope in the test sequencer.
C_LOG_TIME_TRUNC_WARNING	Yields a single TB_WARNING if time stamp truncated. Otherwise none.
C_DEFAULT_MSG_ID_PANEL	Sets the default message IDs that shall be shown in the log.
C_MSG_ID_INDENT	Sets the indentation for each message ID.
C_DEFAULT_ALERT_ATTENTION	Sets the default alert attention.
C_DEFAULT_STOP_LIMIT	Sets the default alert stop limit.

Global signal	Signal type	Description
global_show_log_id	boolean	If true the message IDs will be shown in the log.
global_show_log_scope	boolean	If true the message scope will be shown in the log.
global_show_msg_for_uvvm_cmd	boolean	If true messages for Bitvis UVVM commands will be shown if applicable.

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 minimize 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)