

TYPE	NAME	PARAMETERS	PORTS	DESCRIPTION
Single-Cycle	assert_always	#{severity_level, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	test_expr must always hold
Two Cycles	assert_always_on_edge	#{severity_level, edge_type, property_type, msg, coverage_level}	(clk, reset_n, sampling_event, test_expr)	test_expr is true immediately following the specified edge (edge_type: 0=no-edge, 1=pos, 2=neg, 3=any)
n-Cycles	assert_change	#{severity_level, width, num_cks, action_on_new_start, property_type, msg, coverage_level}	(clk, reset_n, start_event, test_expr)	test_expr must change within num_cks of start_event (action_on_new_start: 0=ignore, 1=restart, 2=error)
n-Cycles	assert_cycle_sequence	#{severity_level, num_cks, necessary_condition, property_type, msg, coverage_level}	(clk, reset_n, event_sequence)	if the initial sequence holds, the final sequence must also hold (necessary_condition: 0=trigger-on-most, 1=trigger-on-first, 2=trigger-on-first-unpipelined)
Two Cycles	assert_decrement	#{severity_level, width, value, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	if test_expr changes, it must decrement by the value parameter (modulo 2 ^{width})
Two Cycles	assert_delta	#{severity_level, width, min, max, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	if test_expr changes, the delta must be >=min and <=max
Single-Cycle	assert_even_parity	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	test_expr must have an even parity, i.e. an even number of bits asserted
Two Cycles	assert_fifo_index	#{severity_level, depth, push_width, pop_width, property_type, msg, coverage_level, simultaneous_push_pop}	(clk, reset_n, push, pop)	FIFO pointers should never overflow or underflow
n-Cycles	assert_frame	#{severity_level, min_cks, max_cks, action_on_new_start, property_type, msg, coverage_level}	(clk, reset_n, start_event, test_expr)	test_expr must not hold before min_cks cycles, but must hold at least once by max_cks cycles (action_on_new_start: 0=ignore, 1=restart, 2=error)
n-Cycles	assert_handshake	#{severity_level, min_ack_cycle, max_ack_cycle, req_drop, deassert_count, max_ack_length, property_type, msg, coverage_level}	(clk, reset_n, req, ack)	req and ack must follow the specified handshaking protocol
Single-Cycle	assert_implication	#{severity_level, property_type, msg, coverage_level}	(clk, reset_n, antecedent_expr, consequent_expr)	if antecedent_expr holds then consequent_expr must hold in the same cycle
Two Cycles	assert_increment	#{severity_level, width, value, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	if test_expr changes, it must increment by the value parameter (modulo 2 ^{width})
Single-Cycle	assert_never	#{severity_level, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	test_expr must never hold
Single-Cycle	assert_never_unknown	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, qualifier, test_expr)	test_expr must never be an unknown value, just boolean 0 or 1
Combinatorial	assert_never_unknown_async	#{severity_level, width, property_type, msg, coverage_level}	(reset_n, test_expr)	test_expr must never go to an unknown value asynchronously, it must remain boolean 0 or 1
n-Cycles	assert_next	#{severity_level, num_cks, check_overlapping, check_missing_start, property_type, msg, coverage_level}	(clk, reset_n, start_event, test_expr)	test_expr must hold num_cks cycles after start_event holds
Two Cycles	assert_no_overflow	#{severity_level, width, min, max, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	if test_expr is at max, in the next cycle test_expr must be >min and <=max
Two Cycles	assert_no_transition	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, test_expr, start_state, next_state)	if test_expr==start_state, in the next cycle test_expr must not change to next_state
Two Cycles	assert_no_underflow	#{severity_level, width, min, max, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	if test_expr is at min, in the next cycle test_expr must be >=min and <max
Single-Cycle	assert_odd_parity	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	test_expr must have an odd parity, i.e. an odd number of bits asserted
Single-Cycle	assert_one_cold	#{severity_level, width, inactive, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	test_expr must be one-cold i.e. exactly one bit set low (inactive: 0=also-all-zero, 1=also-all-ones, 2=pure-one-cold)
Single-Cycle	assert_one_hot	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	test_expr must be one-hot i.e. exactly one bit set high
Combinatorial	assert_proposition	#{severity_level, property_type, msg, coverage_level}	(reset_n, test_expr)	test_expr must hold asynchronously (not just at a clock edge)
Two Cycles	assert_quiescent_state	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, state_expr, check_value, sample_event)	state_expr must equal check_value on a rising edge of sample_event (also checked on rising edge of 'OVL_END_OF_SIMULATION')
Single-Cycle	assert_range	#{severity_level, width, min, max, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	test_expr must be >=min and <=max
n-Cycles	assert_time	#{severity_level, num_cks, action_on_new_start, property_type, msg, coverage_level}	(clk, reset_n, start_event, test_expr)	test_expr must hold for num_cks cycles after start_event (action_on_new_start: 0=ignore, 1=restart, 2=error)
Two Cycles	assert_transition	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, test_expr, start_state, next_state)	if test_expr changes from start_state, then it can only change to next_state
n-Cycles	assert_unchange	#{severity_level, width, num_cks, action_on_new_start, property_type, msg, coverage_level}	(clk, reset_n, start_event, test_expr)	test_expr must not change within num_cks of start_event (action_on_new_start: 0=ignore, 1=restart, 2=error)
n-Cycles	assert_width	#{severity_level, min_cks, max_cks, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	test_expr must hold for between min_cks and max_cks cycles
Event-bound	assert_win_change	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, start_event, test_expr, end_event)	test_expr must change between start_event and end_event
Event-bound	assert_window	#{severity_level, property_type, msg, coverage_level}	(clk, reset_n, start_event, test_expr, end_event)	test_expr must hold after the start_event and up to (and including) the end_event
Event-bound	assert_win_unchange	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, start_event, test_expr, end_event)	test_expr must not change between start_event and end_event
Single-Cycle	assert_zero_one_hot	#{severity_level, width, property_type, msg, coverage_level}	(clk, reset_n, test_expr)	test_expr must be one-hot or zero, i.e. at most one bit set high

PARAMETERS

severity_level
`OVL_FATAL
`OVL_ERROR
`OVL_WARNING
`OVL_INFO
property_type
`OVL_ASSERT
`OVL_ASSUME
`OVL_IGNORE
msg descriptive string

USING OVL

+define+OVL_ASSERT_ON
+define+OVL_MAX_REPORT_ERROR=1
+define+OVL_INIT_MSG
+define+OVL_INIT_COUNT=<tbench>.ovl_init_count
+libext+.v+.vlib
-y <OVL_DIR>/std_ovl
+incdir+<OVL_DIR>/std_ovl

DESIGN ASSERTIONS

Monitors internal signals & Outputs

Examples
* One hot FSM
* Hit default case items
* FIFO / Stack
* Counters (overflow/increment)
* FSM transitions
* X checkers (assert_never_unknown)

INPUT ASSUMPTIONS

Restricts environment

Examples
* One hot inputs
* Range limits e.g. cache sizes
* Stability e.g. cache sizes
* No back-to-back reqs
* Handshaking sequences
* Bus protocol