

TYPE	NAME	PARAMETERS	PORTS	DESCRIPTION
Single-Cycle	ovl_always	#(severity_level, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	test_expr must always hold
Two Cycles	ovl_always_on_edge	#(severity_level, edge_type, property_type, msg, coverage_level)	(clock, reset, enable, sampling_event, test_expr, fire)	test_expr is true immediately following the specified edge (edge_type: 0=no-edge, 1=pos, 2=neg, 3=any)
Event-bound	ovl_arbiter	#(severity_level, width, priority, width_min_cks, max_cks, arbitration_rule, latency_check, single_and_check, priority_level, msg, coverage_level)	(clock, reset, enable, reqs, grnts, priorities, fire)	provides grants in response to requests, as per specified arbitration scheme and within a specified time window
Single-Cycle	ovl_bits	#(severity_level, width, asserted, min, max, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	checks number of asserted (or deasserted) bits is within a specified range
n-Cycles	ovl_change	#(severity_level, width, num_cks, action_on_new_start, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, fire)	test_expr must change within num_cks of start_event (action_on_new_start: 0=ignore, 1=restart, 2=error)
Single-Cycle	ovl_code_distance	#(severity_level, width, min, max, property_type, msg, coverage_level)	(clock, reset, enable, test_expr1, test_expr2, fire)	checks hamming distance between two expressions
n-Cycles	ovl_cycle_sequence	#(severity_level, num_cks, necessary_condition, property_type, msg, coverage_level)	(clock, reset, enable, event_sequence, fire)	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-unconditional)
Two Cycles	ovl_decrement	#(severity_level, width, value, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	if test_expr changes, it must decrement by the value parameter (modulo 2 ^{width})
Two Cycles	ovl_delta	#(severity_level, width, min, max, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	if test_expr changes, the delta must be >=min and <=max
Single-Cycle	ovl_even_parity	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	test_expr must have an even parity, i.e. an even number of bits asserted
Event-bound	ovl_fifo	#(severity_level, width, depth, pass_thru, registered, enq_latency, deq_latency, preload_count, high_water_mark, value_check, property_type, msg, coverage_level)	(clock, reset, enable, enq, deq, full, empty, enq_data, deq_data, preload, fire)	checks data integrity of a FIFO and ensures that the FIFO does not overflow or underflow
Two Cycles	ovl_fifo_index	#(severity_level, depth, push_width, pop_width, property_type, msg, coverage_level, simultaneous_push_pop)	(clock, reset, enable, push, pop, fire)	FIFO pointers should never overflow or underflow
n-Cycles	ovl_frame	#(severity_level, min_cks, max_cks, action_on_new_start, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, fire)	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	ovl_handshake	#(severity_level, min_ack_cycle, max_ack_cycle, req_drop, deassert_count, max_ack_length, property_type, msg, coverage_level)	(clock, reset, enable, req, ack, fire)	req and ack must follow the specified handshaking protocol
n-Cycles	ovl_hold_value	#(severity_level, width, min, max, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, value, fire)	once test_expr matches value, test_expr doesn't change value until a specified event
Single-Cycle	ovl_implication	#(severity_level, property_type, msg, coverage_level)	(clock, reset, enable, a_precedent_expr, consequent_expr, fire)	if antecedent_expr holds then consequent_expr must hold in the same cycle
Two Cycles	ovl_increment	#(severity_level, width, value, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	if test_expr changes, it must increment by the value parameter (modulo 2 ^{width})
Event-bound	ovl_memory_async	#(severity_level, data_width, addr_width, mem_size, addr_check, init_check, one_read_check, one_write_check, value_check, property_type, msg, coverage_level)	(reset, enable, start_addr, end_addr, ren, raddr, rdata, wen, waddr, wdata, fire)	ensures the integrity of accesses to an asynchronous memory
Event-bound	ovl_memory_sync	#(severity_level, data_width, addr_width, mem_size, pass_thru, addr_check, init_check, conflict_check, one_read_check, one_write_check, value_check, property_type, msg, coverage_level)	(r_clock, w_clock, reset, enable, start_addr, end_addr, ren, raddr, rdata, wen, waddr, wdata, fire)	ensures the integrity of accesses to a synchronous memory
n-Cycles	ovl_multiport_fifo	#(severity_level, width, depth, enq_count, deq_count, pass_thru, registered, enq_latency, deq_latency, preload_count, high_water_mark, full_check, empty_check, value_check, property_type, msg, coverage_level)	(clock, reset, enable, enq, deq, enq_data, deq_data, full, empty, preload, fire)	ensures data integrity of a FIFO with multiple enqueue and deque ports, and checks underflow and overflow
Single-Cycle	ovl_mutex	#(severity_level, width, invert_mode, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	ensures that the bits of an expression are mutually exclusive
Single-Cycle	ovl_never	#(severity_level, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	test_expr must never hold
Single-Cycle	ovl_never_unknown	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, qualifier, test_expr, fire)	test_expr must never be an unknown value, just boolean 0 or 1
Combinatorial	ovl_never_unknown_async	#(severity_level, width, property_type, msg, coverage_level)	(reset, enable, test_expr, fire)	test_expr must never go to an unknown value asynchronously, it must remain boolean 0 or 1
n-Cycles	ovl_next	#(severity_level, num_cks, check_overslapping, check_missing_start, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, fire)	test_expr must hold num_cks cycles after start_event holds
Event-bound	ovl_next_state	#(severity_level, width, next_count, min_hold, max_hold, disallow, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, curr_state, next_state, fire)	ensures expression transitions only to specified values
Event-bound	ovl_no_contention	#(severity_level, width, num_drivers, min_quiet, max_quiet, property_type, msg, coverage_level)	(reset, enable, test_expr, driver_enables, fire)	ensures that a bus is driven according to specified contention rules
Two Cycles	ovl_no_overflow	#(severity_level, width, min, max, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	if test_expr is at max, in the next cycle test_expr must be >min and <=max
Two Cycles	ovl_no_transition	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, start_state, next_state, fire)	if test_expr==start_state, in the next cycle test_expr must not change to next_state
Two Cycles	ovl_no_underflow	#(severity_level, width, min, max, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	if test_expr is at min, in the next cycle test_expr must be >=min and <max
Single-Cycle	ovl_odd_parity	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	test_expr must have an odd parity, i.e. an odd number of bits asserted
Single-Cycle	ovl_one_cold	#(severity_level, width, inactive, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	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	ovl_one_hot	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	test_expr must be one-hot i.e. exactly one bit set high
Combinatorial	ovl_proposition	#(severity_level, property_type, msg, coverage_level)	(reset, n, enable, test_expr, fire)	test_expr must hold asynchronously (not just at a clock edge)
Two Cycles	ovl_quiescent_state	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, state_expr, check_value, sample_event, fire)	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	ovl_range	#(severity_level, width, min, max, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	test_expr must be >=min and <=max
Event-bound	ovl_reg_loaded	#(severity_level, width, start_count, end_count, property_type, msg, coverage_level)	(clock, reset, enable, start_event, end_event, src_expr, dest_expr, fire)	ensures that a register is loaded with source data within a specified time window
n-Cycles	ovl_req_ack_unique	#(severity_level, min_cks, max_cks, method, property_type, msg, coverage_level)	(clock, reset, enable, req, ack, fire)	ensures every request receives a corresponding acknowledge in a specified time window
n-Cycles	ovl_req_requires	#(severity_level, min_cks, max_cks, property_type, msg, coverage_level)	(clock, reset, enable, req_trigger, req_follower, resp_loader, resp_trigger, fire)	ensures that every request event initiates a valid request-response event sequence that finishes within a specified time window
n-Cycles	ovl_stack	#(severity_level, width, depth, push_latency, pop_latency, high_water_mark, property_type, msg, coverage_level)	(clock, reset, enable, push, pop, full, empty, push_data, pop_data, fire)	ensures the data integrity of a stack and ensures that the stack does not overflow or underflow
n-Cycles	ovl_time	#(severity_level, num_cks, action_on_new_start, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, fire)	test_expr must hold for num_cks cycles after start_event (action_on_new_start: 0=ignore, 1=restart, 2=error)
Two Cycles	ovl_transition	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, start_state, next_state, fire)	if test_expr changes from start_state, then it can only change to next_state
n-Cycles	ovl_unchange	#(severity_level, width, num_cks, action_on_new_start, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, fire)	test_expr must not change within num_cks of start_event (action_on_new_start: 0=ignore, 1=restart, 2=error)
n-Cycles	ovl_valid_id	#(severity_level, width, min_cks, max_cks, max_instances, max_ids, max_instances_per_id, instance_count, width, property_type, msg, coverage_level)	(clock, reset, enable, issued, issued_count, returned, flush, issued_id, returned_id, flush, id, fire)	Ensures that each issued ID is returned within a specified time window; that returned IDs match issued IDs; and that the issued and outstanding IDs do not exceed specified limits.
Single-Cycle	ovl_value	#(severity_level, width, num_values, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, vals, disallow, fire)	ensures the value of an expression either matches a value in a specified list or does not match any value in the list
n-Cycles	ovl_width	#(severity_level, min_cks, max_cks, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	test_expr must hold for between min_cks and max_cks cycles
Event-bound	ovl_win_change	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, end_event, fire)	test_expr must change between start_event and end_event
Event-bound	ovl_window	#(severity_level, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, end_event, fire)	test_expr must hold after the start_event and up to (and including) the end_event
Event-bound	ovl_win_unchange	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, end_event, fire)	test_expr must not change between start_event and end_event
Single-Cycle	ovl_zero_one_hot	#(severity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, test_expr, fire)	test_expr must be one-hot or zero, i.e. at most one bit set high

PARAMETERS	USING OVL	DESIGN ASSERTIONS	INPUT ASSUMPTIONS
severity_level	+define+OVL_ASSERT_ON	<i>Monitors internal signals & Outputs</i>	<i>Restricts environment</i>
`OVL_FATAL	+define+OVL_MAX_REPORT_ERROR=1		
`OVL_ERROR	+define+OVL_INIT_MSG	<i>Examples</i>	<i>Examples</i>
`OVL_WARNING	+define+OVL_INIT_COUNT=<tbench>_ovl_init_count	* One hot FSM	* One hot inputs
`OVL_INFO		* Hit default case items	* Range limits e.g. cache sizes
property_type	+libext+.v+.vlib	* FIFO / Stack	* Stability e.g. cache sizes
`OVL_ASSERT	-y <OVL_DIR>/std_ovl	* Counters (overflow/increment)	* No back-to-back reqs
`OVL_ASSUME	+incdir+<OVL_DIR>/std_ovl	* FSM transitions	* Handshaking sequences
`OVL_IGNORE		* X checkers (ovl_never_unknown)	* Bus protocol
msg descriptive string			