VhdlCohen Publishing & Consulting
http://www.vhdlcohen.com/
VhdlCohen@aol.com

PSL/SUGAR QUICK REFERENCE CARD VERILOG

From book: ***Using PSL/SUGAR with Verilog and VHDL, Guide to Property Specification Language for Assertion-Based Verification*****, Ben Cohen, 2003, isbn 0-9705394-4-4**

Rev A 5/30/03

Property Format

property <name> = [operator] [enabling_condition(s)]
[implication_operator(s)] (fulfilling_condition)
[until | until_ | abort discharging_condition]
[@(clock_expression)];

// Example with embedded PSL

// **psl property** REQ_ACK_IN_4_CYCLES =
// **always**({req && !ack} |=>{[*0:3]; ack} **abort** !reset_n) @(posedge clk);
// **never** (push && fifo_full && !pop);

Name : Identifier displayed when property fails
Operator: **always** | **never**
Enabling_condition: Boolean. Zero=false, non-zero=true
Implication_operator: Relationship between expressions
Fulfilling condition Tested behavior. Checked every verification cycle. Boolean expression or sequence. Last expression prior to any discharging condition.
Discharging condition Stop checking of the behavior.
clock expression When to sample the assertion. Generally a clock edge, but can be any Boolean expression. Default clock can be specified.

Implication_operators

->	RHS started when LHS is true. RHS is Boolean.
-> next	RHS evaluated in next cycle after LHS true. RHS is Boolean.
|->	RHS started in last cycle of the LHS true. LHS is a sequence {}
|=>	RHS started following cycle LHS condition true. LHS is a sequence {}. Shorthand for |-> {true;
eventually!	RHS is true in some future cycle, and must be true before the end of simulation. RHS is Boolean or a sequence

Discharging conditions

until(_) Fulfilling condition must hold until the expression is true.
abort Cancels checking of an assertion.

Default clock

The *clock* **default clock** = (posedge clk);
property NeverRdWrBothActive = **never** (read && write);

SERE Operators

; Temporal concatenation
{*a; b; c*} *a* at cycle t, *b* at t+1, *c* at t+2 *c* is true.

[*] Consecutive repetition

[*n]	Repeats for n cycles
[*]	Repeats for zero or any number of cycles
[+]	Repeats for one or more cycles
[*n:m]	Repeats for min of n to max of m cycles
[*n:inf]	Repeats for a minimum of n cycles
[*0: m]	Either skipped or repeats max of m cycles

[=] Non-consecutive repetition
[->] GOTO repetition

Examples

always ({go; req} |=> {!req && ack; data_transfer});

always ({a} |=> {b[*2]; c}); // if *a*, then *b; b; c* sequence
always ({a} |=> {[*2]; c}); // if *a*, then *-; - ; c* sequence
always ({a} |=> {b[*1:3]; c}); if *a* then either of sequences:
b; c | *b; b; c* | *b; b; b; c*
always ({a} |=> {b[*0:3]; c}); if *a* then either of sequences:
c | *b; c* | *b; b; c* | *b; b; b; c*
always ({a} |=> {b[+]; c}); // if *a* then either of sequence
b; c | *b; b; c* | *b; b;…..; b; c*
always ({a} |=> {[*]; b; c}); // Cannot fail. For functional coverage
// if *a* then assertion completes when *b*; *c* sequence occurs.
always ({a} |=> {b[=2]; c}); **//** If *a* then two occurrence of b before *c*;
always ({a} |=> {b[->2]; c}); **//** If *a* then two occurrence of b any cycle before *c*, but last *b* must occur in cycle before *c*;

Eventually!

Boolean -> eventualy! Boolean
Boolean -> eventualy! {SERE}

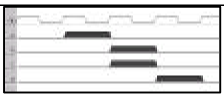
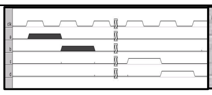
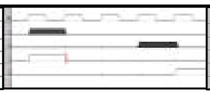
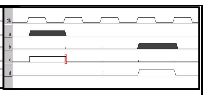
// Grant must always occur sometime after req.
// **always** ((req) -> **eventually!** (grant);
// **always** ((req) -> **eventually!** {grant; ok});

Until

Boolean **until** Boolean
Boolean -> **next** Boolean **until** Boolean
Boolean -> **eventually!** Boolean **until** Boolean
Boolean -> **next** Boolean
-> **eventually!** Boolean **until** Boolean
Boolean -> **eventually!** Boolean
-> **eventually!** Boolean **until** Boolean
({ SERE} |=> { SERE}) **until** Boolean

always (({a; b} |=> {c}) **until** (rst));
always ((state = S1) -> **next** ((state = S2) **until** (state=S3)));
always(req -> **eventually!** ack **until** data_xfr;
always ((state = S1) -> **next** (state = S2) -> **eventually!** (state=S3)
until done);
always(req -> **eventually!** ack -> **eventually!** data_xfr **until** done**);**

always (({go; req} |=> {!req && ack; data_xfr}) **until** done);

Sequence composition operators **: sequence fusion.** Two sequences overlap by one cycle **| sequence disjunction.** One of two alternative sequences hold at the current cycle **& non-length-matching sequence conjunction.** Two sequences both hold at the current cycle, regardless of whether they complete in same cycle or in different cycles. **&& length-matching sequence conjunction.** Two sequences both hold at the current cycle, and both complete in the same cycle.	    🡅Fusion {a; b} : {c; d} 🡅Sequence Disjuction {a; b} | {c; d} Non-Length-Matching {a; [*]; b} & {c[*1:5]; d}🡅 Length-Matching {a; [*]; b} && {c[*1:5]; d} 🡅

Named sequences
Define common sequences by name.
Creates more readable and reusable code
	// **psl default clock** = (posedge clk);
	// **psl sequence** AB **=** {a; b};
	// **psl sequence** CD = {c; d};
	// **psl sequence** AB_EV = {a; [*]; b};
	// **psl sequence** CD_LIMITED = {c[*1:5]; d};
	// **psl sequence** EMBEDDED = {AB; CD};
	// **psl property** TEST1 =
	// **always** ({go} |-> {AB; CD_LIMITED});
	// **psl property** FUSION_TEST =
	// **always** ({go} |=> {AB[*2] }: {CD[*2] });
	// **psl property** DISJUCTION_TEST =
	// **always** ({go} |=> {AB} | {CD});
	// **psl property** SEQUENCE_NON_MATCH_TEST =
	// **always** ({go} |=> {AB_EV} & {CD_LIMITED});

Verification Unit
Assertions in external files
	module reqack // in file reqack.v
	(…); …
	endmodule
	
	vunit v1 (reqack) {
	//in file reqack.vu
	default clock = (posedge clk);
	property REQ_ACK = **always** ((req && bus_available) -> (ack));
	property ABCD_NEXT = **always** (a -> **next** b -> **next** c -> **next** d);
	property ABCD_IF = **always** (a -> b -> c -> d);
	}

Verification directives **assert** Property **;** **assume** Property **;** **assume_guarantee** Property **;** **restrict** Sequence **;** **restrict_guarantee** Sequence **;** **cover** Sequence **;** **fairness** Boolean **;** **strong fairness** Boolean **,** Boolean **;**	 Verify a property holds. Constrain verification (e.g., input behavior) so that a property holds. Property and assumed property both hold. Initialize design to get to a specific state before checking assertions. Constrain design so sequence holds and verify restrict sequence holds. Check if a certain path was covered by the verification space Guide to verify the property only over fair paths. A path is *fair* if every fairness constraint holds along the path.
PSL in Design Process • Requirements • Synthesizable HDL • Testbench • Integration into application • Verification • Documentation	 . Clarifies properties of specifications. Great for requirements review. . Documents design implementation properties. Great for code reviews . Facilitates TB designs. Eases uncertainties in microcycle timing of DUT. . Detects errors during design integration. . Detects errors, white-box verification. Provides functional verification. . Properties and simulation with assertion metrics enhance documentation.
Guide also available at	**http://www.vhdlcohen.com/** Models and Papers