JASPER design automation

PSL Reference Guide, VHDL Flavor

Properties and Verification Units	Description	Example

vunit <name>[(<ent>(<arch>))] {
[inherit <verification unit>;]
[<directive>;]
}
	Container for defining a complete verification job. Can contain all verification directives.	vunit my_vunit(fifo(rtl)) {
		restrict {reset; not reset[*]};
		assert never oe and we;
		}

vmode <name>[(<ent>(<arch>))] {
…
}
	Container for defining verification environment contraints. Can not contain the assert directives.	vmode my_vmode(fifo(rtl)) {
		assume never rd and wr;
		}

vprop <name>[(<ent>(<arch>))] {
…
}
	Container for defining verification objectives. Can only contain the assert directive.	vprop my_vprop(fifo(rtl)) {
		assert never {rd;rd;wr};
		}

Declarations	Description	Example
property <name>[(<parameter list>)] is <property>;	A declared property can be instantiated wherever properties are allowed.	property my_p is always ack -> busy; assert p1;

sequence <name>[(<parameter list>)] is <sequence>;	A delared sequence can be instantiated wherever sequences are allowed.	sequence my_s is {oe[*5]};
		assert s1;

endpoint <name>[(<paramter list>)] is <sequence>;	Endpoints can be used as booleans that are true in a specific cycle if the sequence completes in that cycle.	endpoint my_e is {sample[=8]; done};
		assert always ack -> (busy until e1);

Basic logic operators	Description	Example
not <property>	Logical negation.	not reset
<property> and <property>	Logical and.	rd and wr
<property> or <property>	Logical or.	rd or wr
<property> -> <property>	Logical implication.	req -> next ack
<property> <-> <property>	Logical equivalence.	(req and not busy) <-> next ack

Temporal Operators	Description	Example
always <property or sequence>	The property or sequence should always hold.	always oe -> not we
never <property or sequence>	The property or sequence should never hold.	never or and we
next <property>	Property holds one clock cycle in the future.	always req -> next ack
next[n] (<property>)	Property holds on the n'th cycle into the future.	always req -> next[2](ack)
next_a[<range>] (<property>)	Property holds at all clock cycles of a range of future clock cycles.	always next_a[2 to inf](rd -> not wr)
next_e[<range>] (<property>)	Property holds at least once in the range of future clock cycles.	always next_e[*2 to 5](rd -> wr)
next_event (<bool>) [n] (<property>)	Property holds at n'th occurrence of Boolean expression.	always next_event(rd)[1](req)
next_event_a (<bool>) [<range>] (<property>)	Property holds at all cycles in the range of future clock cycle.	always next_event_a(req)[2 to inf](next ack)
next_event_e (<bool>) [<range>] (<property>)	Property holds at least once in the range of Boolean occurrences.	always next_event_e(req)[1 to 4](ack)
<property> until <property>	Must hold until a certain event.	always (full_fifo until wr)
<property> before <property>	Must hold before a certain event.	always ack -> (req before ack)
<property> abort <bool>	Terminate at a certain even.	always (grant -> (busy until done) abort reset)
<sqeuence> |=> <sqeuence>	Suffix implication, precondition is followed by another sequence in the next clock cycle.	{rd;rd} |=> {{not rd} : {wr}}
<sequence> |-> <sequence>	Overlapping suffix implication, precondition is followed but another sequence starts in the last clock cycle.	{rd;rd;not rd} |-> {wr}
whilenot (<bool>) <sequence>	Sequence should hold until Boolean occurs.	whilenot(reset){wr;wr;rd}

Clockhandling	Description	Example
default clock is <clock>;	Define one default clock in a vunit to be used by all properties and sequences.	default clock is (clk'event and clk='0');
@clock	Clock a property or sequence with a certain clock signal. Usage will override any default clock.	property my_fifo is (req -> next ack) @(clk'event and clk='1');

JASPER design automation

Sequences and SERE's	Description	Example
<SERE>; <SERE>	Concatenation of sequences.	{rd; rd; wr}
<sequence> : <sequence>	Two sequences overlap by one clock cycle.	{rd; rd; wr} : {req; ack}
<sequence> | <sequence>	One of two sequences hold at a specific clock cycle.	{rd; rd} | {rd; wr}
<sequence> & <sequence>	Two sequences start at the same clock cycle, they do not need to be of the same length.	{rd; rd; wr} & {req; ack}
<sequence> && <sequence>	Two sequences start at the same clock cycle and they need to be of the same length.	{rd; rd; wr} && {!int[*]}
<SERE>[*n]	Repetition in n consecutive clock cycles.	{rd[*5]}
<SERE>[*]	Repetition for 0 or any number of clock cycles.	{rd[*]; rd; wr}
<SERE>[+]	Repetition for 1 or more clock cycles.	{rd[+]; wr}
<SERE>[*n:m]	Repetition for n to m number of clock cycles.	{rd[*2:5]} |=> {wr}
<SERE>[=n]	Repetition for n non-consecutive clock cycles.	{rd[=3]} |=> {wr}

Built-in functions	Description	Example
rose(<bool>)	Boolean was false at previous clock cycle and true at current.	rose(xmittiing) -> (busy until done_xmitting)
fell(<bool>)	Boolean was true at previous clock cycle and false at current.	never fell(rcving) && !done_rcving;
prev(<expression>)	A function, returns the value of <expression> in the previous clock cycle.	always ((rd = '0') -> next (prev(data_out) = data_out));
prev(<expression>, n)	A functions, returns the value of <expression> in the n'th previous clock cycle.	{(!rd)[*3]; rd} |=> {prev(data_in, 5) = data_out};

Inheritance	Description	Example
inherit <verification unit>;	Inherits verification directives from other verification units.	inherit my_vunit;

Safelogic extensions	Description	Example
initially <sequence>;	Initilizes verification to start at certain state[1].	initially {reset == '0'};
clock_generator <clock> is <pattern>;	Defines how a specific clock should be generated.	clock_generator clk_1 = "0011";

Verification Directives	Description	Example
assert <property>;	Verify that a property holds.	assert always req -> (ack || retry);
assume <property>;	Assume that the property holds during verification.	assume never rd && wr;
assume_guarantee <property>;	Treated as assume If the vunit that the directive is defined in, binds to the top level, and as assert otherwise.	assume_guarantee never busy && rd;
restrict <sequence>;	Constrain verification according to a specific sequence[2].	restrict {reset; !reset[*]};
restrict_guarantee <sequence>;	Treated as restrict if the vunit that the directive is defined in, binds to the top level, and as assert otherwise.	restrict_guarantee {!wr[*]; rd; [*]};
cover <sequence>;	Check if the sequence was fulfilled during verification.	cover {state == BUSY; [*]; state == IDLE};

Parameter types	Description
const	Represents a constant integer expression.
boolean	Any boolean-layer expression.
property	Any property.
sequence	Any sequence;

Precedence	Operator	Description
High	<boolean>	HDL operators.
↓	@	Clocking operator.
	; [*] [=] [->]	SERE construction operators.
	: | & &&	Sequence implication operators.
	|-> |=>	Foundation Language implication operators.
	always never next* within* whilenot* G F X [W]	Foundation Language occurrence operators.
Low	abort until* before *	Termination operators.

[1] An initially directive must be applied to a sequence with a statically computable length.
[2] A restrict directive matches the infinite verification trace. Sequences used in restrict directives should always have infinite lengths.