COMS30026 Design Verification Assertion-based Verification

Kerstin Eder

Trustworthy Systems Laboratory

https://www.bristol.ac.uk/engineering/research/trustworthy-systems-laboratory/





What is an assertion?

- An assertion is a statement that a particular property is required to be true.
 - A property is a Boolean-valued expression, e.g. in SystemVerilog.
- Assertions can be checked either during simulation or using a formal property checker.



What is an assertion?

- An assertion is a statement that a particular property is required to be true.
 - A property is a Boolean-valued expression, e.g. in SystemVerilog.
- Assertions can be checked either during simulation or using a formal property checker.
- Assertions have been used in SW development for a long time.
 - assert.h in standard library of C

#include <assert.h>

- C preprocessor macro assert()
- Used to detect NULL pointers, out-of-range data, ensure loop invariants, pre- and post-conditions, etc.

Assertions in C code

```
1 #include <stdio.h>
 2 #include <assert.h>
 3
 4 int mysquare(int n) {
     int s = 0;
 5
 6
     int i = 0;
     int k = 0; /* assertion variable to count the number of times in the loop */
 7
 8
9
     assert (n >= 0); // Pre-condition to catch invalid input
10
11
     assert (s == k*n && i==k); // Invariant to catch mistaken variable initialisation, e.g. i != 0 or s != 0
12
13
     while (i < n) {
14
       s = s + n;
15
       i = i + 1;
16
       k = k + 1;
       assert ((s == k*n) & (i==k); // Invariant to catch errors in the loop computation
17
18
19
     }
20
21
     assert (k == n); // Post-condition to catch a mistaken final state of the loop
22
23
     assert (s == k*n \&\& i==k); // Invariant to catch errors in the loop computation
24
25
     assert (s == n * n); // Check desired post-condition
26
27
     return s;
28 }
29
30
31 int main() {
32
    int n = -4;
33
     int square = 0;
34
35
     printf("n = %d n", n);
36
    square = mysquare(n);
37
     printf("n^2 = \%d\n", square);
38
39
     return 0;
```

40 }

Assertions in C code

```
1 #include <stdio.h>
 2 #include <assert.h>
 4 int mysquare(int n) {
 5
    int s = 0;
 6
    int i = 0;
7
     int k = 0; /* assertion variable to count the number of times in the loop */
 8
9
     assert (n \geq 0); // Pre-condition to catch invalid input
10
11
     assert (s == k*n && i==k); // Invariant to catch mistaken variable initialisation, e.g. i != 0 or s != 0
12
13
     while (i < n) {
14
       s = s + n;
15
       i = i + 1;
16
       k = k + 1;
       assert ((s == k*n) & (i==k); // Invariant to catch errors in the loop computation
17
18
19
    3
20
21
     assert (k == n); // Post-condition to catch a mistaken final state of the loop
22
23
     assert (s == k*n \&\& i==k); // Invariant to catch errors in the loop computation
24
25
     assert (s == n * n); // Check desired post-condition
26
27
     return s;
28 }
                                        [cskie@it000908:SLIDES$ gcc mysquare.c -o mysquare
29
                                        [cskie@it000908:SLIDES$ ./mysquare
30
31 int main() {
                                         n = 4
32
    int n = -4;
                                        n^2 = 16
    int square = 0;
33
                                        [cskie@it000908:SLIDES$ gcc mysquare.c -o mysquare
34
                                        cskie@it000908:SLIDES$ ./mysquare
35
     printf("n = \%d n", n);
                                         n = -4
     square = mysquare(n);
36
                                        Assertion failed: (n >= 0), function mysquare, file mysquare.c, \mathcal{V} ne 9
     printf("n^2 = %d\n", square);
37
                                        Abort trap: 6
38
                                         cskie@it000908:SLIDESs
39
     return 0;
40 }
```

Assertions in C code

```
1 #include <stdio.h>
 2 #include <assert.h>
 4 int mysquare(int n) {
 5
    int s = 0;
 6
    int i = 0;
7
     int k = 0; /* assertion variable to count the number of times in the loop */
 8
9
     assert (n \geq 0); // Pre-condition to catch invalid input
10
11
     assert (s == k*n && i==k); // Invariant to catch mistaken variable initialisation, e.g. i != 0 or s != 0
12
13
     while (i < n) {
14
       s = s + n;
15
       i = i + 1;
16
       k = k + 1;
       assert ((s == k*n) & (i==k); // Invariant to catch errors in the loop computation
17
18
19
    3
20
21
     assert (k == n); // Post-condition to catch a mistaken final state of the loop
22
23
     assert (s == k*n \&\& i==k); // Invariant to catch errors in the loop computation
24
25
     assert (s == n * n); // Check desired post-condition
26
27
     return s;
28 }
                                        [cskie@it000908:SLIDES$ gcc mysquare.c -o mysquare
29
                                        [cskie@it000908:SLIDES$ ./mysquare
30
31 int main() {
                                         n = 4
32
    int n = -4;
                                        n^2 = 16
    int square = 0;
33
                                        [cskie@it000908:SLIDES$ gcc mysquare.c -o mysquare
34
                                        cskie@it000908:SLIDES$ ./mysquare
35
     printf("n = \%d n", n);
                                         n = -4
     square = mysquare(n);
36
                                        Assertion failed: (n >= 0), function mysquare, file mysquare.c, \mathcal{V} ne 9
     printf("n^2 = %d\n", square);
37
                                        Abort trap: 6
38
                                         cskie@it000908:SLIDESs
39
     return 0;
40 }
```

HW Assertions

- Combinatorial (i.e. "zero-time") conditions
 - ensure functional correctness
 - must be valid at all times
 - "The buffer never overflows."
 - "The register always holds a single-digit value."
 - "The state machine encoding is one hot."

HW Assertions

Combinatorial (i.e. "zero-time") conditions

- ensure functional correctness
- must be valid at all times
 - "The buffer never overflows."
 - "The register always holds a single-digit value."
 - "The state machine encoding is one hot."

Temporal conditions

- to verify sequential functional behaviour over a period of time
 - "The grant signal must be asserted for a single clock cycle."
 - "A request must always be followed by a grant or an abort within 5 clock cycles."
- Temporal assertion languages facilitate specification of temporal properties.
 - System Verilog Assertions (SVA)
 - Property Specification Language (PSL)



The Open Verification Library

- Revolution through Foster & Bening's OVL for Verilog in early 2000
 - Clever way of encoding a re-usable assertion library originally in Verilog. ⁽²⁾
 - 33 assertion checkers
 - OVL language support for: Verilog, VHDL, PSL, SVA

e e assert_never_logic.v								
🗅 🗁 📃 🗶 🔚 🥱 🐰 🖆 🛍								
1 // Accellera Standard V2.8.1 Open Verification Library (OVL).								
2 // Accellera Copyright (c) 2005-2014. All rights reserved.								
3								
4 //								
5 // ASSERTION 6 //								
7 `ifdef OVL_ASSERT_ON								
8								
9 // 2-STATE								
10 //								
<pre>11 wire fire_2state_1;</pre>								
12 always @(posedge clk) begin								
<pre>13 if (`OVL_RESET_SIGNAL == 1'b0) begin</pre>								
14 // OVL does not fire during reset								
15 end								
16 else begin 17 if (fire_2state 1) begin								
<pre>17 if (fire_2state_1) begin 18 ovl_error_t(`OVL_FIRE_2STATE,"Test expression is not FALSE");</pre>								
19 end								
20 end								
21 end								
22								
<pre>23 assign fire_2state_1 = (test_expr == 1'b1);</pre>								



The Open Verification Library

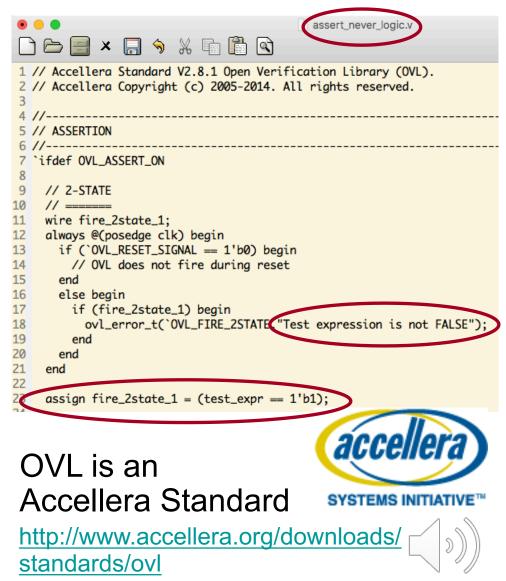
- Revolution through Foster & Bening's OVL for Verilog in early 2000
 - Clever way of encoding a re-usable assertion library originally in Verilog. ⁽²⁾
 - 33 assertion checkers
 - OVL language support for: Verilog, VHDL, PSL, SVA

	assert_never_logic.v
1/	/ Accellera Standard V2.8.1 Open Verification Library (OVL).
	/ Accellera Copyright (c) 2005-2014. All rights reserved.
3	
-	/
	/ ASSERTION
	/
	, ifdef OVL_ASSERT_ON
8	
	// 2-STATE
-	//
	wire fire_2state_1;
12	always @(posedge clk) begin
13	
14	if (`OVL_RESET_SIGNAL == 1'b0) begin
14	// OVL does not fire during reset
	end
16 17	else begin
	if (fire_2state_1) begin
18	<pre>ovl_error_t(`OVL_FIRE_2STATE("Test expression is not FALSE");</pre>
19	end
20	end
21	end
22	
4	assign fire_2state_1 = (test_expr == 1'b1);



The Open Verification Library

- Revolution through Foster & Bening's OVL for Verilog in early 2000
 - Clever way of encoding a re-usable assertion library originally in Verilog. ⁽²⁾
 - 33 assertion checkers
 - OVL language support for: Verilog, VHDL, PSL, SVA
- Assertions have now become very popular for Verification, giving rise to
 Assertion-Based
 Verification (and also Assertion-Based Design).



SAFETY & LIVENESS



Safety Properties

- Safety: Something bad does not happen
 - The FIFO does not overflow.
 - The system does not allow more than one process at a time to modify the shared memory.
 - Requests are answered within 5 clock cycles.



Safety Properties

- Safety: Something bad does not happen
 - The FIFO does not overflow.
 - The system does not allow more than one process at a time to modify the shared memory.
 - Requests are answered within 5 clock cycles.
- More formally: A safety property is a property for which any path violating the property has a finite prefix such that every extension of the prefix violates the property.

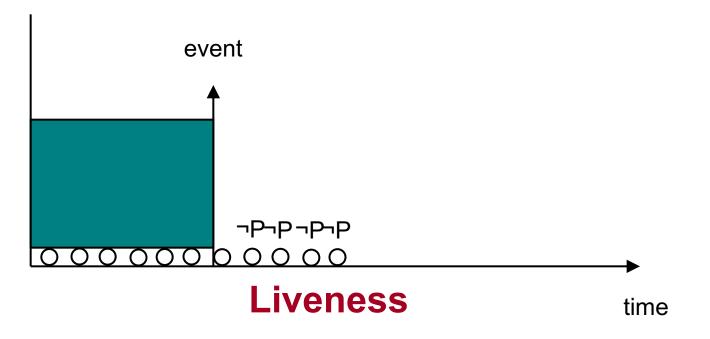
Safety properties can be falsified by a finite simulation run.



Liveness Properties

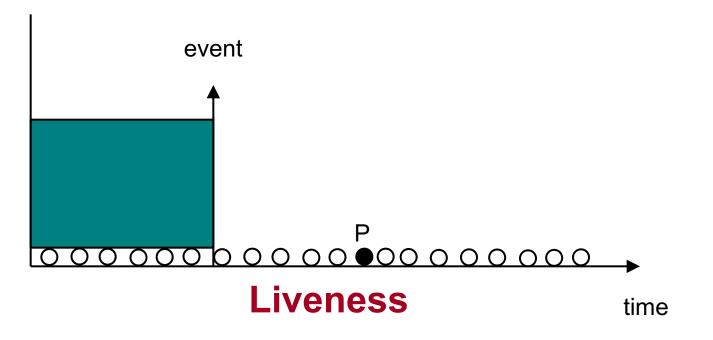
- Liveness: Something good eventually happens
 - The decoding algorithm eventually terminates.
 - Every request is eventually acknowledged.
- More formally: A liveness property is a property for which any finite path can be extended to a path satisfying the property. [Foster etal.: Assertion-Based Design. 2nd Edition, Kluwer, 2010.]

Liveness



Assertion P must eventually be valid after the event occurs.

Liveness



Assertion P must eventually be valid after the event occurs.



Von André Karwath aka Aka -Eigenes Werk, CC BY-SA 2.5, https://commons.wikimedia.org/ w/index.php?curid=103762





Von André Karwath aka Aka -Eigenes Werk, CC BY-SA 2.5, https://commons.wikimedia.org/ w/index.php?curid=103762







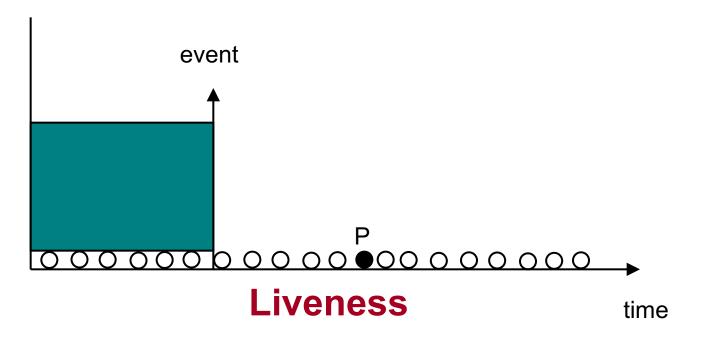


- Liveness: Something good eventually happens
 - The decoding algorithm eventually terminates.
 - Every request is eventually acknowledged.
- More formally: A liveness property is a property for which any finite path can be extended to a path satisfying the property. [Foster etal.: Assertion-Based Design. 2nd Edition, Kluwer, 2010.]

In theory, liveness properties can only be falsified by an infinite simulation run.

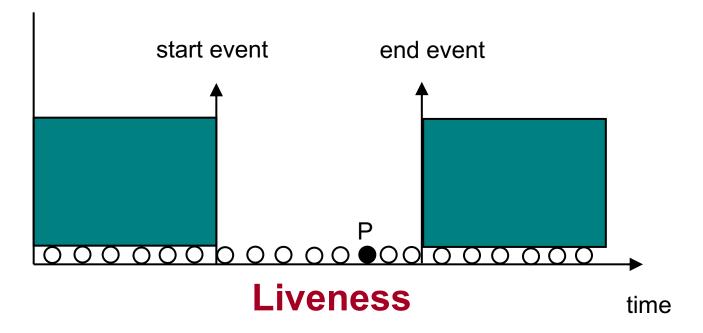
- Practically, we often assume that the "graceful end-oftest" represents infinite time.
 - If the good thing did not happen after this period, we assume that it will never happen, and thus the property is falsified.

Bounded Liveness



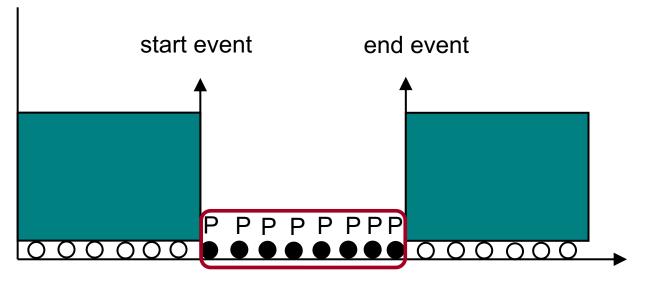
Assertion P must eventually be valid after the event occurs

Bounded Liveness



 Assertion P must eventually be valid after the start event trigger occurs and before the end event trigger occurs.

Invariant



Invariant

time

Invariant Assertion Window:

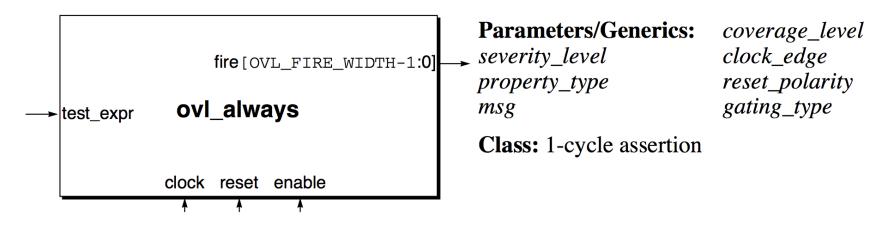
Assertion P is checked and expected to hold after the **start event** occurs and continues to be checked and is expected to hold until the **end event.**

EXAMPLE OVL CHECKERS



ovl_always

Checks that the value of an expression is TRUE.



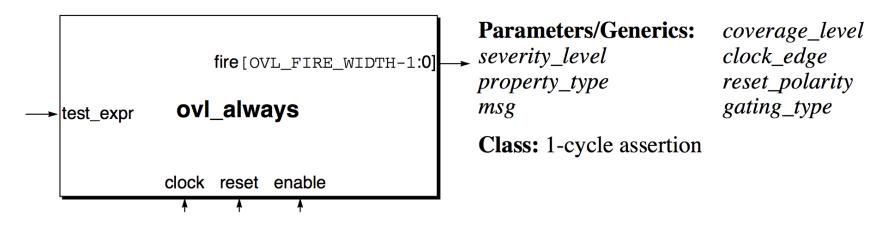
Syntax

ovl_always
 [#(severity_level, property_type, msg, coverage_level, clock_edge,
 reset_polarity, gating_type)]
 instance_name (clock, reset, enable, test_expr, fire);



ovl_always

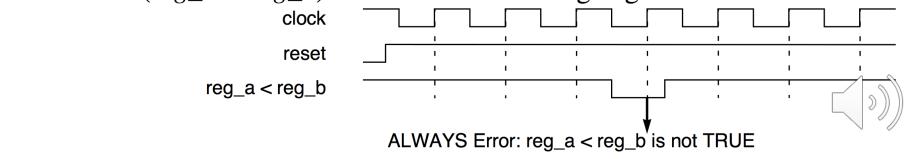
Checks that the value of an expression is TRUE.



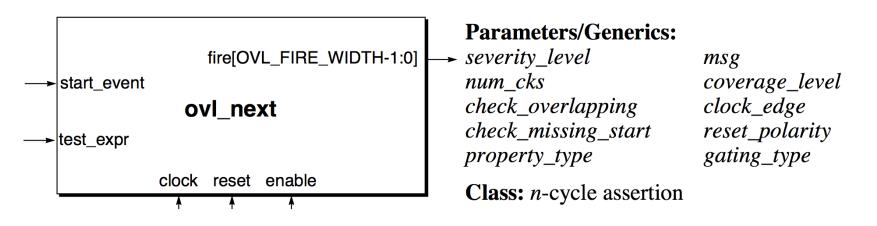
Syntax

ovl_always
 [#(severity_level, property_type, msg, coverage_level, clock_edge,
 reset_polarity, gating_type)]
 instance_name (clock, reset, enable, test_expr, fire);

Checks that $(reg_a < reg_b)$ is TRUE at each rising edge of *clock*.

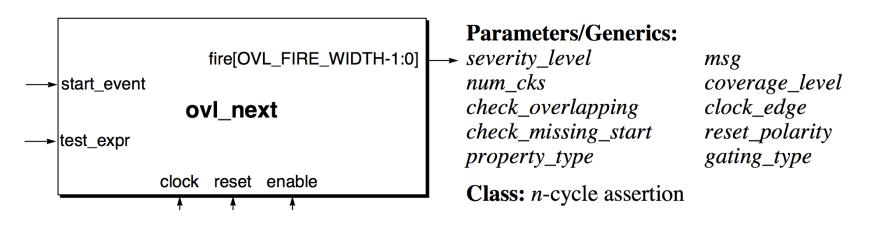


Checks that the value of an expression is TRUE a specified number of cycles after a start event.





Checks that the value of an expression is TRUE a specified number of cycles after a start event.



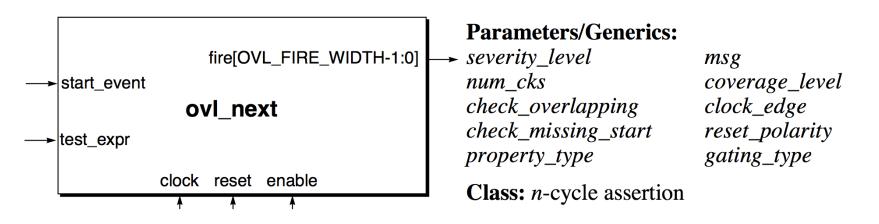
Syntax

ovl_next

Number of cycles after start_event is TRUE to wait to check that the value of test_expr is TRUE. Default: 1.



Checks that the value of an expression is TRUE a specified number of cycles after a start event.



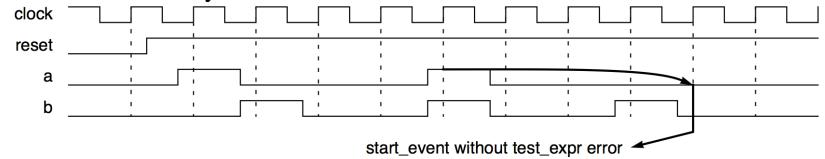
Syntax

ovl_next

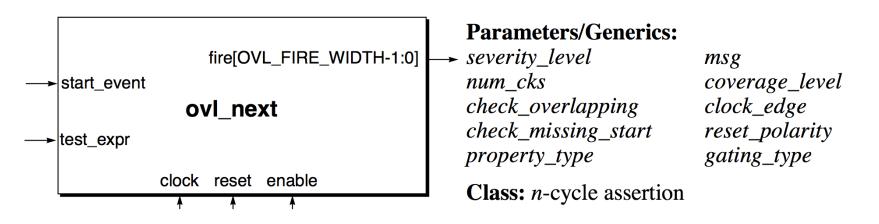
[#(severity_level, num_cks, check_overlapping, check_missing_start, property_type, msg, coverage_level, clock_edge, reset_polarity, gating_type)]

instance_name (clock, reset, enable, start_event, test_expr, fire);

Checks that b is TRUE 4 cycles after a is TRUE.



Checks that the value of an expression is TRUE a specified number of cycles after a start event.



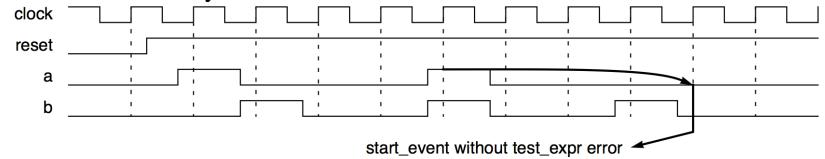
Syntax

ovl_next

[#(severity_level, num_cks, check_overlapping, check_missing_start, property_type, msg, coverage_level, clock_edge, reset_polarity, gating_type)]

instance_name (clock, reset, enable, start_event, test_expr, fire);

Checks that b is TRUE 4 cycles after a is TRUE.



OVL QUICK REFERENCE (www.eda.org/ovl) Last updated: 25th May 2007

AmeA					-		
SimuNote of the second se							
Name Nam Name Name Name						typic: 0=no-eidge, 1=pics, 2=neig, 3=any)	
Model Provide and end/or state state state state state state state. Person and state stat			Asswerity_level, width, priority_width, min_cks, max_cks, arbitration_rule,				
mm mm< mm< mm mm<	Single-Cycle	ovi_bits	priority check, sincle ont check, property type, msg, opverage level) #(seventy_level, width, asserted, min, max, property_type, msg, coverage_level)	(clock, reset, enable, test_exepr, fire)	checks number of asserted (or deasserted) bits is within a specifi	ed range	
Note: <th< td=""><td>n-Cycles</td><td>ovl_change</td><td></td><td>(clock, reset, enable, start_event, test_expr, fire)</td><td>test_expr must change within num_cks of start_event (action_on,</td><td>ne w_start: 0=igno re, 1=restart, 2=error)</td></th<>	n-Cycles	ovl_change		(clock, reset, enable, start_event, test_expr, fire)	test_expr must change within num_cks of start_event (action_on,	ne w_start: 0=igno re, 1=restart, 2=error)	
Name Name <th< td=""><td>Single-Cycle</td><td>ovl_code_distance</td><td></td><td>(clock, reset, enable, test_expr1, test_expr2, fre)</td><td>checks hamming distance between two expressions</td><td></td></th<>	Single-Cycle	ovl_code_distance		(clock, reset, enable, test_expr1, test_expr2, fre)	checks hamming distance between two expressions		
Schwart Name Joseph and Berley and Schwart Name Joseph and Schwart Name Joseph and Schwart Name Joseph and Schwart Schwart Schwart Name Joseph and Schwart	n-Oycles	ovl_cycle_sequence	#(seventy_level, num_cks, necessary_condition, property_type, msg. coverage_level)	(clock, reset, enable, event_sequence, fire)		icessary_condition: 0=trigger-on-most, 1=trigger-on-fint, 2=trigger-on-first-	
Mark Mark Mark and an and an and a start and a					If test_expr changes, it must decrement by the value parameter ()	nodulo 2'width)	
Mining Ass. ¹ Meensking and end generating an					If test_exprchanges, the data must be >=min and <=max		
Mode Mode Mode Mode Mode Mode Mode Mode			A seventy_level, width, depth, pass_thru, registered, enq_latency, deq_latency,	(clock, reset, enable, and, dag, full, empty, and_data,			
Ame Ames	Two Cydes	ovi filo index			FIFO pointers should never overflow or underflow		
Control Another Market Mark Another Mark Mark Another Mark Mark Market Mark Mark Mark Control Another Mark Market Mark <td></td> <td></td> <td>simultaneous push popi</td> <td></td> <td></td> <td></td>			simultaneous push popi				
Nome Rescherber innergischere Aurorigie der Au	n-cycles	ovi_trame		(cock, reliset, enable, start_event, test_expr, fre)		east once by max_oks cycles (action_on_new_start: u ≈ginore, 1≈restart,	
Solid Addition Perchast data from some participant data from some partipant data from	n-Cycles	ovl_handshake		(clock, reset, enable, req, ack, fire)	req and a ck must follow the specified handshalking protocol		
Nome Number Number<							
Image: Not service of the second se							
 at and M (see a set of the set	Single-Cycle	ovl_implication	#(seventy_level, pisperty_type, msg, coverage_level)	(clock, reset, enable, antecedent_expr, consequent_expr, itee)	If ante cedent_expr holds then consequent_expr must hold in the	same cyle	
Marka Rived Reservation (minimum reserved R						odulo 2*width)	
state	Event-bound	ovl_memory_async			ensure s the integrity of accesses to an asynchronous memory		
office index							
Nome Number of the second o	Event-bound	ovi_mamay_sync			ensures the integrity of accesses to an synchronous memory		
 Pickers of Marka Pickers of Marka							
matrix matrix<	n-Cycles	ovl_multiport_filo			ensures data integrity of a FIFO with multiple enqueue and deque	ports, and checks underflow and overflow	
hinder in Anor Hinder Ander				ampry, preiciaz, irrej			
 Barbon Janson Age Strateging and Strat	~ /		#(seventy_level, width, invert_mode, property_type, msg. coversige_level)	(clock, reset, enable, test_expr, fre)	ensures that the bits of an expression are mutually exclusive		
 Indiana Jaya India							
mark mark <th< td=""><td></td><td></td><td>Kseventy_level, width propenty_type, msg, coverage_level) Kseventy_level, width, property_type, msg, coverage_level)</td><td></td><td></td><td>must re main boole an 0 or 1</td></th<>			Kseventy_level, width propenty_type, msg, coverage_level) Kseventy_level, width, property_type, msg, coverage_level)			must re main boole an 0 or 1	
minute μ, μ	n-Cycles	ovi_next		(clock, reset, enable, start_event, test_exepr, fire)	test_exprimust hold num_cks cycles afterstart_event holds		
 Markan Markan Mar	Event-bound	ovi_next_state		(clock, reset, enable, test_expr, curr_state, next_state, fire)	ensures expression transitions only to specified values		
No. operation Non-price	East-band	ad as contention		(ment mobile text over driver mobiles fini)	one on that a but is down a coording to see clied contention ad-		
Normal Solution Assay station As				(reso, enable, loss_copr, envar_enables, net)			
An outborner An outborner							
Name of a constraint Notation Notation Notation Notation Notation							
Part of chi Sorte of chi So				Construction and an Coder and			
Dependent Support Numerity (with sparty, jone, no, comp, (with) Mader, make status, data paper, tool, with sparty sparts, no, comp, (with) Mader, make status, data paper, tool, with sparty sparts, no, comp, (with) Mader, make status, data paper, tool, with sparts, sparts, no, comp, (with) Mader, make status, data paper, tool, with sparts, sparts, no, comp, (with) Mader, make status, data paper, tool, with sparts, sparts, no, make status, and sparts, data with smarts, data wit	Single-Cycle	swl_ane_adid		(clock, reset, enable, test_expr, fre)	te st_expr must be one-cold i.e. exactly one bit set low (in active: 0	anou ≂also-ali-zero, 1 ≈also-ali-ones, 2≈pure-o ne-cold)	
box Sold standard Notify should with party is you may solved just if is any of the standard with the party is in the should and party solutified is black with a stand is in the should and party is in the should party is in the should party is in the should part							
Op/Column Op/Column <t< td=""><td></td><td></td><td></td><td></td><td colspan="2"></td></t<>							
markade Multiplicability Monethy lowerity, book update data space data data data data data data data dat	Sinda Outla	out repose	through and with the max month the men courses level		hat over mut he berrie and strong v		
Open solution Manual place in the second secon				(clock, reset, enable, start_event, end_event, src_expr,			
Open solution All registree All regi	n-Cycles	ovil rato actic unique	d'swerty level, min cles, mar, des method, property type, msg. coverage, level)		ensure severy request receives a corresponding acknowledge in	a see cified time window	
Open display Market Appending tools (addity, depit, depit, addity, depit, depit, pole, Marget, Melly, Marget, Mell, addity, eterning, addity, depit,	-						
Import Import<	o O relac	eud etterik	Wearachy level width doubh much Internet men Internet hide writer med-		one on the data intends of a stack and are use that the stack d	national associated as a control flast	
No. Manufacion Avendy, Jonet Web, Yuebell, Web, Yuebell, Yue, Ying, Yue, Ying, Yueb, Y		ON SEACH		(000, rese, enable, push, pop, ruil, empty, push_oalla, pop_data, fie)	ensure sithe data integrity of a stack and ensures that the stack does not overflow or und enflow		
Open Section Appendix June Section	n-Oycles	ovi_time	#(seventy_level, num_cks, action_on_new_start, property_type, msg. coverage_level)	(clock, reset, enable, start_event, test_exepr, fire)	te st_expr must hold for num_cks cycl es after start_event (action_	on_new_start: 0≓ignore, 1≕resta <i>it</i> , 2≔erro <i>r</i>)	
Openet Appenet	Two Cycles	ovl_transition	#(sevenity_level, width, property_type, msg. co.verage_level)	(clock, reset, enable, test_expr, start_state, next_state, fire)	If test_expr changes from start_state, then it can only change to r	ext_state	
Address Severage. Mol Address of a severage. Mol Address of a severage. Mol Visited Address Address of a severage. Mol Address of a severage. Mol Address of a severage. Mol Address of a severage. Mol Address of a severage. Mol Address of a severage. Mol Addres severage. Mol Address of a severage.	n.Ovdas	ovi undvance					
main free			coverage_level)				
indig Code Aboverthy, New Terms, Terms, Stern,	n-Cycles	ovl_valid_id				ndow; that returned ID's matchissued IDs; and that the issued and outstanding	
Cycles ov/.wh1 Apwardty, load, incide, mee, des property, lyae, meg, coverage, lood) dode, nead, watable, des, exp, ond, sevent, but out out out that the that mem, des can that, des cycles out. Mandbauer out. Mandbauer Abwerdty, load, incide, mee, des property, lyae, meg, coverage, lood) dode, nead, watable, des, exp, ond, sevent, load, exp, mult hold abree meint, des ordet and ind, severt but out and multiple mem, des meint, des cycles dode, nead, watable, des, exp, ond, sevent, load, load, meint, des cycles dode, nead, watable, des, exp, ond, sevent, load, load, meint, des cycles dode, nead, watable, des, exp, ond, sevent, load, load, meint, des cycles dode, nead, watable, des, exp, ond, sevent, load, load, meint, des cycles dode, nead, watable, des, exp, ond, sevent, load, load, meint, des cycles wint board out. Mandbauer deswerty, load, meent, load			#(sevenity_level, width, num_values, property_type, msg. coverage_level)		ensures the value of an expression either matches a value in a s	pecified list or does not match any value in the list	
Wint board out, who wint board feed							
wirtband old ywr, undhange Revendy, beal, widt, propring, type, meg, coverage, beal) clock, read, audit, start, seart, teit, segr, and, event, Inci. teit, sear muttba one hot or zero is a strong one bit at host. wirteband xiz zero one hot teiteentry, beal, widt, propring, type, meg, coverage, beal) dook, reade audits, teit, segr, and, event, Inci. teit aven muttba one hot or zero is a strong one bit at host. WARNETERS USINGOVL DESIGN ASSERTIONS INPUT ASSUMPTIONS warzdruf, lavel -define+OVL_ASSERT_ON Monitors internal signals & Outputs Restricts environment 'OVL_FATAL -define+OVL_MAX_REPORT_ERROR=1	Event-bound	ovl_win_change	P(seventy_level, width, property_type, msg, coverage_level)	(dock, resot, enable, start_event, test_expr, end_event, tre)	te st_expr must change between start_event and end_event		
Index Out Index Index Avia and one hot Takework low low hot areand, low may ownage low line index index Avia Address of the section of the	Event-bound	ovl_window	#(sevenity_level, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, end_event, fre) test_expr must hold after the start_event and up to (and including)the end_event	
Instruction Instruction avise one bot Newerity lead wide, insteal one bits, test, each, fead bits are installing and bits of bits in bits APAMETERS USINGOVL DESIGN ASSERTIONS INPUT ASSUMPTIONS wareful, lavel +define+OVL_ASSERT_ON Monitors internal signals & Outputs Restricts environment 'OVL_FATAL +define+OVL_MAX_REPORT_ERROR=1	Event-bound	ovl_win_unchange	#(sevenity_level, width, property_type, msg, coverage_level)	(clock, reset, enable, start_event, test_expr, end_event,	te st_expr must not chan ge between start_event and end_event		
APAMAETERS USINGO/L DESIGN ASSERTIONS INPUT ASSUMPTIONS awardin_lard define+OVL_ASSERT_ON Monitors internal signals & Outputs Restricts environment 'OVL_FATAL define+OVL_MAX_REPORT_ERROR=1	Cinete Contra	ud ann an hat		frei:	had a ser much be seen but as more than the second s		
averity. level +define+OVL_ASSERT_ON Monitors internal signals & Outputs Restricts environment 'OVL_FATAL +define+OVL_NAX_REPORT_ERROR=1 : 'OVL_ERROR +define+OVL_INIT_MSG Examples 'OVL_WARNING +define+OVL_INIT_COUNT= *One hot FSM 'OVL_NARNING +define+OVL_INIT_COUNT= *One hot FSM 'OVL_SASSERT +libext+.vt.vilb *FIO / Stack 'OVL_ASSERT -y <ovl_dir>/std_ovl *FIO/ Stack 'OVL_SASSUME +indir+ *FSM transitions 'OVL_GNORE *X checkers (ovf_inverouvn) *Bus protocol</ovl_dir>	csingle-Cycle	2/4_2020_one_hdt	Resevency level, width, property, type, msg. colverage, level)	(coox, resid, enable, test_oppr, fre)	test, exprimust be one-hot or zero, i.e. at most one bit set high		
'OVL_FATAL +define+OVL_MAX_REPORT_ERROR=1 'OVL_ERROR +define+OVL_INIT_MSG Examples 'OVL_WARNING +define+OVL_INIT_COUNT= +define+OVL_INIT_COUNT= *define+OVL_INIT_COUNT= *Methed=unt>.ovL_init_count *One hot FSM *One hot inputs 'OVL_NRFO *Hit default case items *Range limits e.g. cache sizes *Range limits e.g. cache sizes *Mttp://WWW.ACC 'OVL_NSFO *FIFO / Stack *Stability e.g. cache sizes *Mttp://WWW.ACC 'OVL_ASSUME +libext+.vt.vibb *FIFO / Stack *Stability e.g. cache sizes 'OVL_ASSUME +lindir+ <ovl_dir>/std_ovl *Counters (overflow/increment) *No back-back regs 'OVL_SNORE *X checkers (ovf_i_never_unknown) *Bus protocol Advanceded/State</ovl_dir>	PARAMETE	RS	USING OVL	DESIGN ASSERTIONS	INPUT ASSUMPTIONS		
OVL_ERROR +define+OVL_NIT_MSG Examples OVL_WARNING +define+OVL_NIT_COUNT= *One hot FSM *One hot inputs 'OVL_INFO *Itid statu case items *Range limits e.g. cache sizes wraperty_type *Ibext+.vi+.viib *FFO / Stack *Stability e.g. cache sizes 'OVL_ASSERT -y <ovl_dir>/std_ovI *Counters (overflow increment) *Noack-boback reqs 'OVL_ASSUME +inodir+ *FSM transitions *Handshaiking sequences 'OVL_SNORE *X checkers (ovf_never_unknown) *Bus protocol</ovl_dir>	severity_lev	rel	+define+OVL_ASSERT_ON	Monitors internal signals & Outputs	Restricts environment		
'OVL_WARNING +define+OVL_INIT_COUNT= +define+OVL_INIT_COUNT= *One hot FSM *One hot inputs 'OVL_INFO *Hit default case items *Range limits e.g. cache sizes *Range limits e.g. cache sizes 'OVL_ASSERT -y <ovl_dir>(std_ovI *FIFO / Stack *Stability e.g. cache sizes 'OVL_ASSENT -y <ovl_dir>(std_ovI Counters (overflow/increment) *No back-to-back regs 'OVL_SINDRE +lindir+ *END *rsitions *Hatdabaking sequences</ovl_dir></ovl_dir>	'OVL_FAT	AL	+define+OVL_MAX_REPORT_ERROR=1				
'OVL_WARNING +define+OVL_INIT_COUNT= +define+OVL_INIT_COUNT= *One hot FSM *One hot inputs 'OVL_INFO *Hit default case items *Range limits e.g. cache sizes *Range limits e.g. cache sizes 'OVL_ASSERT +illow1+.vt+.vlib *FFO / Stack *Stability e.g. cache sizes 'OVL_ASSERT -y <ovl_dir>/std_ovI Counters (overflow/increment) *No back-to-back regs 'OVL_SINDRE +linodir+<ovl_dir>/std_ovI *ENM transitions *Handshaking sequences</ovl_dir></ovl_dir>				Examples	Examples		
'OVL_INFO * Hit default case items * Range limits e.g. cache sizes wroperty.type + libext+.v+.vlib * FIFO / Stack * Stability e.g. cache sizes 'OVL_ASSERT -y < OVL_DIR>/std_ovI * Counters (overflow/increment) * No back-to-back reqs 'OVL_ASSUME + inodir+ <ovl_dir>/std_ovI * FSM transitions * Handshaking sequences 'OVL_IGNORE * X checkers (ow_inever_unknown) * Bus protocol</ovl_dir>	_						
			The second s				
			+ Thart+ v+ vTh			http://www.oo	
						$\underline{\mathbf{n}}$	
			,				
			+incair+ <ovl_dir>/std_ov1</ovl_dir>			downloads/sta	
neg descriptive sting				* X checkers (ovl_never_unknown)	" Bus proto col		
	msg descrip	ptive string					

			OVL QUICK REFERENCE	(www.eda.org/ovl) Last upda	ated: 25th May 2007	
TYPE	NAME		and hadd realize			DESCRIPTION	
Single-Cycle Two Cycles Event-bound	ovil_atways ovil_atways_on_ado ovil_atbitar	n-Cycles	ovl_hold_value		() vort, test_expr, fire) vrianities , fire)	test_expr must always hold test_expr is true imme diately following the specifie d edge (edge providies grants in response to requests, as per specified arbits	
Single-Cycle	ovi_bits	Single-Cycle	ovl_implication		ira)	checks number of asserted (or deasserted) bits is within a spec	if ed range
n-Oycles Single-Oycle	ovi_change	0 0 0			test_expr2, fre)	test_expr must change within num_oks of start_event (action_o checks hamming distance between two expressions	n_new_start: u=ignore, 1=restart, 2=error)
n-Oycles	ov/_cycle_sequence				ance, fire)		(nace ssary_condition: 0=trig ger-on-most, 1=trig ger-on-linst, 2=trigg er-on-finst-
Two Cydes Two Cydes	ov/_docrement ov/_dota	Two Cycles	ovl_increment		10) 10)	If test_expr changes, it must decrement by the value parameter If test_expr changes, it is deta must be >=min and <=max	(mod ulo 2*wi dth)
Singlie Oycle Event-boun d	ov/_aven_parity ov/_tito	Event-bound	ovl_memory_async		re) II, empty, enq_data,	te st_exprimust have an even parity, i.e. an even number of bits checks data integrity of a FIFO and ensures that the FIFO does	
Two Cydes	ovi_filo_index	L vent-bound		,	ira)	RFO pointers should never overflow or underflow	
n-Oycles	ov/_frame				test_expr, fire)	te st_expr must not hold before min_cks cycle s, but must hold a	t least once by max_cks cycles (action_on_new_start: 0 ≓gnore, 1=re start,
n-Cycles	ovl_handshake				0	req and a ck must follow the specified handshaking protocol	
n-Oycles	ovi_toid_value	Event-bound			alue, fire)	once test_expr matches value, test_expr do esrit change value	
Single-Oycle	ovi_implication	Event-bound	ovl_memory_sync		jexpr, consequent_expr,	If ante cedent_expr holds then consequent_exprmust hold in th	
Two Clydies Event-bound	ovi_increment ovi_memory_async				ie) ddr, ren, raddir, rdata,	If test_expr changes, it must increment by the value parameter ensures the integrity of accesses to an asynchronous memory	(modulo 2*width)
Event-bound	ovi_mmary_sync					ansure s the integrity of accesses to an synchronous memory	
n-Oycles	ovi_multiport_file		1 10 1 60		, fina) rg_clata, cloq_clata, full,	ensures data integrity of a FEO with multiple enqueue and deq	ue norts, and charks underflew and overflew
	on ji an par Caro	n-Cycles	ovl_multiport_fifo		d'head and head out		
Single-Cycle Single-Cycle	ovi_mutex ovi_never				18) 10)	ensures that the bits of an expression are mutually exclusive test, exprimust never hold	
Single-Cycle Combinatorial	ovi_never_unknow ovi_never_unknow				at_orpr.fire)	test, exprimust never ble an unknown value, just boole an 0 or 1 test, exprimust never go to an unknown value a synchronously,	
n-Oydes	ovi_next				Nest_expr, fire)	te st_expr must hold num_oks cycles after start_ovent holds	
Event-bound Event-bound	ovl_next_state	Single-Cycle	ovl_mutex		anables, fire)) ensures e opression tran sit ons only to specified values ensures that a bas is driven according to specified contention rules	
Two Cycles Two Cycles	ovi_no_overflow	Single-Cycle	ovl_never		re) tart_state, nært_state, fire)	//ftest_expris at max. In the next cycle test_expr must be >min and <=max.) (Ftest_expr==start_state, in the next cycle test_expr must not change to next_state	
Two Cycles Single-Cycle	ovi_no_underflow	Single-Cycle	ovl_never_unknow	•	u)	if test_expris at min, in the next cycle test_exprimist be >>min test_exprimist have an odd parity, i.e. an odd number of bits a	and <max< th=""></max<>
Single-Cycle Single-Cycle Single-Cycle	ovi_ane_odd				(a) (a) (a)	te st_exprimust have an odd parky, i.e. an odd number of ols a te st_exprimust be on e-cold i.e. exactly one bit set low (in active te st_exprimust be on e-hot i.e. exactly one bit set high	santao 0 ≈also-all-zero, 1 ≈also-all-ones, 2≈pune-o ne-cold)
Combi natorial Two Cydles	ovi_proposition ovi_quiescent_stat	Combinatorial	ovl_never_unknow	n_async	chock_value,	te st, expr must hold asynchronously (not just at a clock edge)	event (also checked on rising edge of 'OVL_END_OF_SMULATION)
Single-Cycle Event-bound	ovi_range ovi_rag_loaded	n-Cycles	ovl_next		re) and_evant,src_axpr,	te st_exprimust be >⇒min and ≪max ensures that a register is loaded with source data within a spec	flad time window
n-Cycles n-Cycles	ovi_req_ack_uniqu ovi_req_requires				i) req_fdlowar,	ensure sie very request, receive sia come spionding acknowle dge i ensure sithat, every request event initiates a valid request, resno	n a spe cified time window rise event sequence that finishes within a specified time window
n-Oycles	owl_stack	Event-bound	ovl_next_state		ull, empty, push_data,	ensures the data integrity of a stack and ensures that the stack	
n-Cycles	ovi_time	L vent-bound	UVI_INEXI_SLALE		test_sopr, fire)	te st_expr must hold for num_cks cycles after start_event (action	n_on_new_start: 0 ≒ig.nore, 1≈resta#, 2≈error)
Two C yoles	ovl_transition				tart_state, next_state, fire)	If test_exprohanges from start_state, then it can only change to	
n-Oydes	ovi_unchange	Event-bound	ovl_no_contention		test_expr, fire) ed_count_raturned_flus.h	te st_expr must not change within num_cks of start_event (actio	n_on_new_start: 0=ignois, 1=restart; 2=emor) window: that returned ID's matchissued IDs: and that the issued and outstanding
Single-Cycle	ovijvalue				, fre) als, disallow, fre)	D s do not exce ed specified limits. ensures the value of an expression either matches a value in a	
n-Cycles Event-bound	ovl_width ovl_win_change	Two Cycles	and no exertions		re) test_expr, and_evant,	te st_expr must hold for between min_cks and max_cks cycles te st_expr must change between start_event and end_event	
Event-bound	ovl_window		ovl_no_overflow		test_expr, and_exent, fire)	test_expr must hold after the start_event and up to (and include	ng) the end_event
Event-bound	ovi_win_unchange	Two Cycles	ovl_no_transition		test_expr, and_avent,	test_expr must not change between stat_event and end_event	
PARAMET	zvi zero one hot				<u>(a)</u>	lest exprimist be on o hot or zero, i.e. at most one bit set high INPUT ASSUMPTIONS	
severity_le		Two Cycles	ovl_no_underflow		Outputs	Restricts environment	
'OVL_FAT 'OVL_ERF						Examples	
'OVL_WA	RNING	Single-Cycle	ovl_odd_parity			* One hot inputs	
`OVL_INF property_t		Single-Cycle	ovl_one_cold			* Range limits e.g. cache sizes * Stability e.g. cache sizes	http://www.acco
OVL_ASS		www.www.www.		and the second	J ament)	* No back-to-back reqs	http://www.acce
OVL_ASS		+incdir+ <ovl_0< th=""><th>DIR>/std_ovl</th><th>* FSM transitions</th><th></th><th>* Handshaking sequences</th><th>downloads/star</th></ovl_0<>	DIR>/std_ovl	* FSM transitions		* Handshaking sequences	downloads/star
'OVL_IGN	IORE			* X checkers (ovl_never_	unknown)	* Bus proto col	

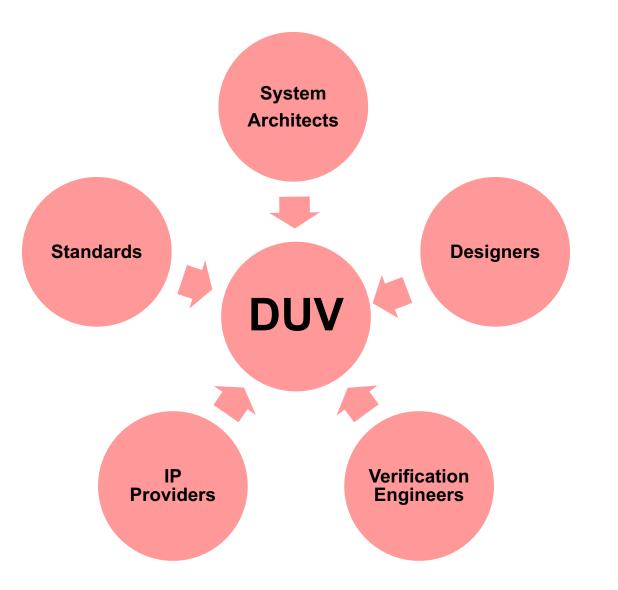
ellera.org

msg. descriptive string

WHERE DO ASSERTIONS COME FROM?



Who writes the assertions?





Implementation Assertions

- Also called "design" assertions.
 - Specified by the designer.
- Encode designer's assumptions.
 - Interface assertions:
 - Catch different interpretations between individual designers.
 - Conditions of design misuse or design faults:
 - detect buffer over/under flow
 - detect buffer read & write at the same time when only one is allowed
- Implementation assertions can detect discrepancies between design assumptions and implementation.
- But implementation assertions won't detect discrepancies between functional intent and design! (Remember: Verification Independence!)

Specification Assertions

- Also called "intent" assertions
 - Often high-level properties.
- Specified by architects, verification engineers, IP providers, standards.
- Encode expectations of the design based on understanding of functional intent.
- Provide a "functional error detection" mechanism.
- Supplement error detection performed by selfchecking testbenches.
 - Instead of using (implementing) a monitor and checker, in many cases writing a block-level assertion can be much simpler.

End of Part I

COMS30026 Design Verification Assertion-based Verification

Kerstin Eder

Trustworthy Systems Laboratory

https://www.bristol.ac.uk/engineering/research/trustworthy-systems-laboratory/

