

Using OneSpin Quantify

Darren Galpin, Infineon UK



OneSpin Quantify – what is it?

- A tool for gauging the quality of property/assertion sets using all available assertions/properties.
- Generates a structural coverage metric for the code under test.
- Use to identify holes in the formal environment which completeness checks might have missed, or might have been missed due to constraints.
- Uses SVA, ITL, PSL and VHDL assertions together.

How to run

- Prove all properties/assertions once.
- Type "quantify" on the command line.
- Wait.....
- Analyse results.

Quantify running

Assertion Checks | Lint Browser

Name	Proof Status	Case Split Status	Witness Status	Validity	Unres.	Source
! [dropdown]	! <any statu: [dropdown]	! <any sl [dropdown]	! <any st [dropdown]	! <any va [dropdown]	!	[dropdown]
Assertions						
Constraints						
Macros						
Properties						
during_reset	hold	open	pass (1)	up_to_date		pfi_formal.vli:8
basic_demand_access_no_prefetch	hold	open	pass (1)	up_to_date		pfi_formal.vli:27
pipelined_demand_access_no_prefetch	hold	open	pass (1)	up_to_date		pfi_formal.vli:99
demand_access_then_prefetch_demand_to_same_while_prefetch_ongoing	hold	open	pass (1)	up_to_date		pfi_formal.vli:168
demand_access_then_prefetch_demand_to_same_after_prefetch_completed_size1	hold	open	pass (1)	up_to_date		pfi_formal.vli:274
demand_access_then_prefetch_demand_to_same_after_prefetch_completed	hold	open	pass (1)	up_to_date		pfi_formal.vli:341
basic_demand_access_ecc_c_fail	hold	open	pass (1)	up_to_date		pfi_formal.vli:497
basic_d [dropdown]						al.vli:600
basic_d [dropdown]						al.vli:704
pipeline						al.vli:818
pipeline						al.vli:920
pipeline						al.vli:1184
pipeline						al.vli:1357
pipeline						al.vli:1485
pipeline						al.vli:1623
demand						al.vli:1683
unsupp						al.vli:1850
demand						al.vli:1877
demand						al.vli:1997
demand						al.vli:2069
tag_val						al.vli:2199
pmbi_d						al.vli:2212
prefetch						al.vli:2314
prefetch						al.vli:2520
prefetch						al.vli:2621
never_e						al.vli:2737
error_n						al.vli:2753
error_te						al.vli:2789
error_n						al.vli:2826
basic_le						al.vli:2861
basic_e						al.vli:2930
disable						al.vli:2997
basic_le						al.vli:3089
demand						al.vli:3155

Shell | Messages | Progress

	%	#Statements	%	#Branches
-I- done	70.50%	644	70.77%	260

-I- Exclusion Summary	%	#Statements	%	#Branches

-I- excluded by user	0.00%	0	0.00%	0
-I- excluded redundant code	0.62%	4	1.52%	4
-I- excluded verification code	0.00%	0	0.00%	0
-I- quantify targets	99.38%	644	98.48%	260

-I- total code	100.00%	648	100.00%	264

-I- Result Summary	%	#Statements	%	#Branches

-I- covered	68.48%	441	70.77%	184
-I- reached	27.80%	179	25.38%	66
-I- unknown	1.71%	11	3.85%	10
-I- unobserved	1.09%	7	0.00%	0
-I- uncovered	0.93%	6	0.00%	0
-I- constrained	0.00%	0	0.00%	0
-I- dead	0.00%	0	0.00%	0



-I- done	70.50%	644	70.77%	260

What the summary means

- Only runs on executable design code – ignores code it thinks is redundant, or is included in embedded design assertions.
- Covered – The line of code was executed, and the effect of the line was observed. Similar but not identical to a Certitude fault insert and detection.
- Reached – Line of code was executed, but observation of effect not known. Equivalent of simulation statement coverage.
- Unknown – Haven't executed line, don't know if it can be observed.
- Unobserved – Line of code was executed, and proven not to be observable.
- Uncovered – Haven't executed line, proven not to be observable.
- Constrained – Can't hit due to constraints.

On finishing run.....

- Once run has been completed, generate one of two reports.
- 1) HTML.....

Structural Coverage Overview					
Status		Statements		Branches	
1	covered	10	52.63%	7	58.33%
R	reached	0	0.00%	0	0.00%
U	unknown	0	0.00%	0	0.00%
OR	unobserved	2	10.53%	1	8.33%
0	uncovered	5	26.32%	2	16.67%
OC	constrained	1	5.26%	1	8.33%
OD	dead	1	5.26%	1	8.33%
Sum	quantify targets	19		12	

57	<code>else if (error_i == 1'b1)</code>	OC
58	<code>ncnt = '1;</code>	OC
59	<code>else if (abort_i == 1'b1)</code>	OD
60	<code>ncnt = '1;</code>	OD
61	<code>else</code>	
62	<code>ncnt = cnt + 1;</code>	1
63	<code>end</code>	
64	<code>aborted: begin</code>	0

Reporting

- 2) Can generate a UCIS compliant XML report file.
- XML file can then be read into another tool such as AsureSign, merged with simulation coverage data to give an overview of whole coverage.
- XML contains pass/fail for all assertions/properties in design and test environment, plus the “structural coverage” information.
- Can thus use formal results to prove bits missed by simulation within the same analysis environment.

Pros and Cons

■ Pros

- Leverages all assertions and properties within environment, whether in RTL or in property environment.
- Easy understood results which are directly analogous to structural coverage. Doesn't need user to consider all inputs/outputs or restructure properties as you may have to for a completeness check.
- Easily shows holes in property coverage – easy to identify missing chunks, or chunks missed due to constraints.
- When merging the coverage in AsureSign, can demonstrate that lines not hit by simulation have been hit formally – coverage is comparable between the two.

■ Cons

- Long property times snowball the quantify run time. Have a property set of around 40 properties, one of which takes 8 hours to run, the others all less than 10 mins. Takes 4 days to run Quantify. Others do run much quicker....
- As with structural coverage, only shows you've tested what is there, not if any function is missing.



ENERGY EFFICIENCY COMMUNICATIONS SECURITY

Innovative semiconductor solutions for energy efficiency, communications and security.

