Beyond Feasibility

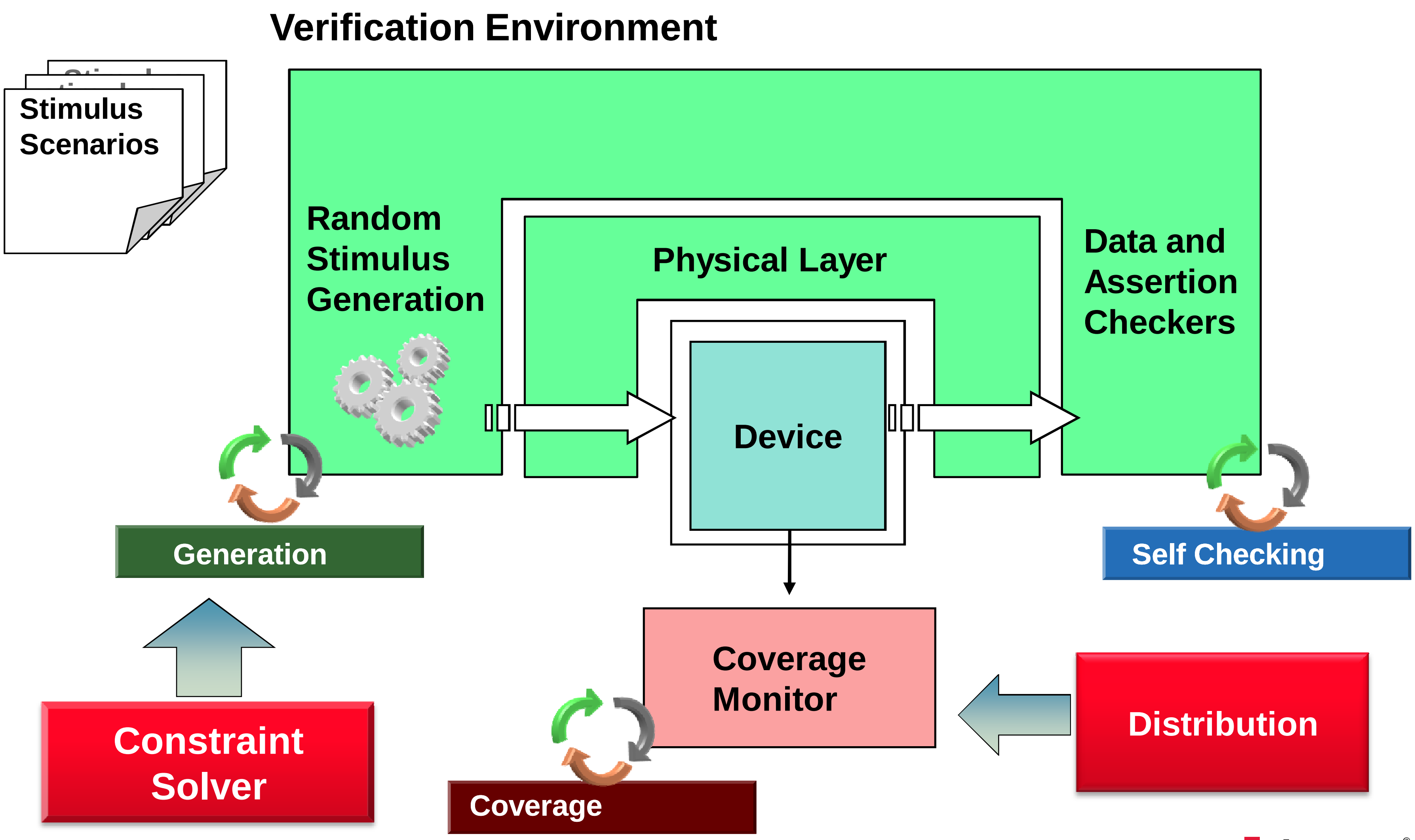
CP Usage in Constrained-Random Functional Hardware Verification

CP, September 2013

Reuven Naveh & Amit Metodi
Cadence R&D, Rosh Ha'Ayin, Israel

cādence®

Functional Verification



cadence

Random testing and constraints

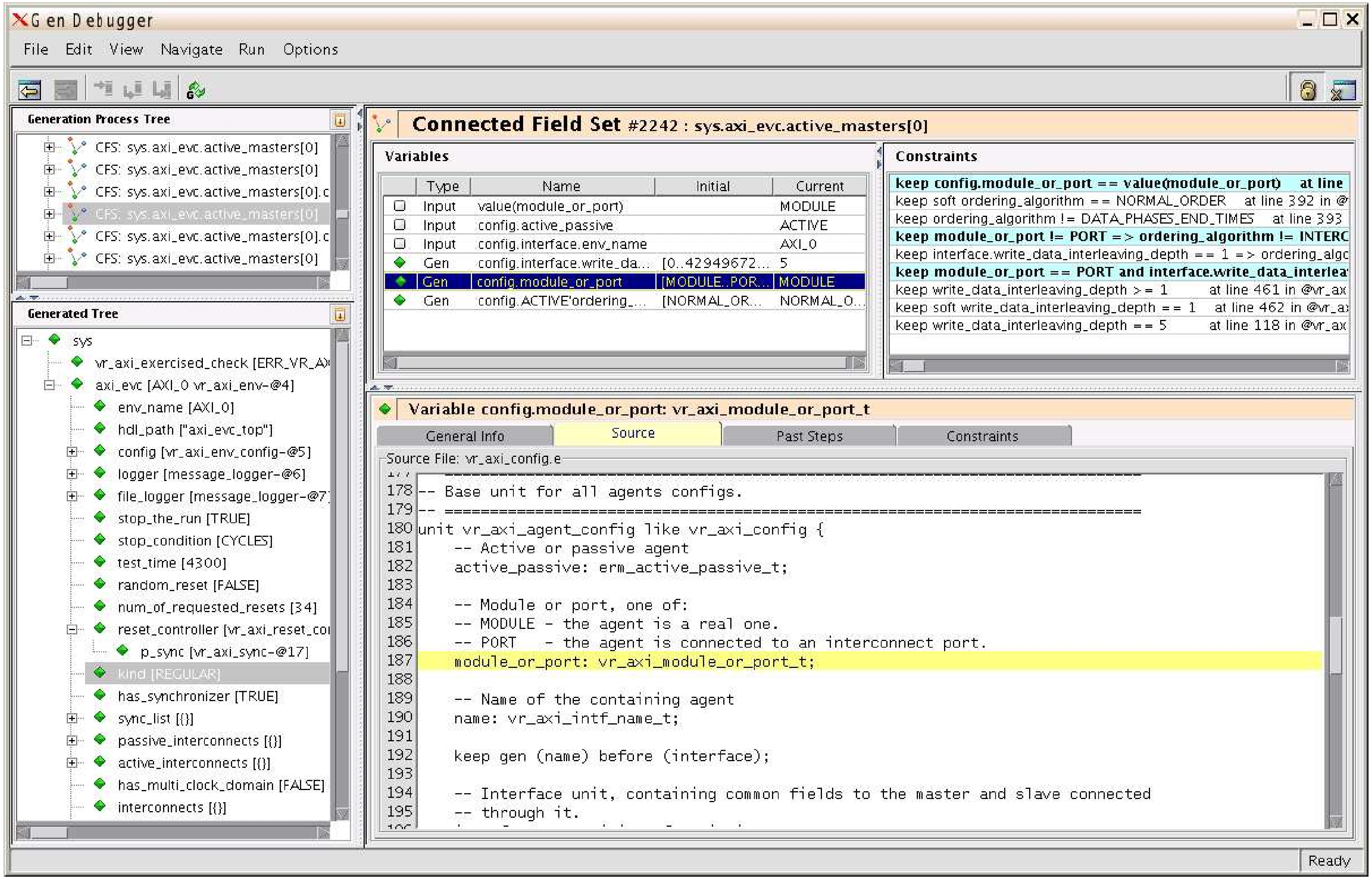
- Random stimuli is used to find bugs in the device
  - Diversity of tests is needed so all scenarios will be testes
  - Constraints are used to define the injected data's properties
- Verification environments can be huge
  - Many separate constraint problems
  - Some of the problems are solved many times
  - Some are solved only once
  - The problems are not necessarily hard to solve
- Debug is a crucial matter
  - Tests need to be reproduced
  - The verification engineers are not constraint experts

IntelliGen

- Part of Cadence Specman tool
  - Introduced in 2007
  - Works with the *e* verification language
- Suitable for large verification environments
  - Splits the environment to many separate problems
  - Uses a reusable and low cost solver
- Various kinds of constraints
  - Word-level (arithmetic) and bit-level constraints
  - Global constraints (sum, count, all-different)
  - Soft constraints
  - Distribution constraints
- Provides interactive generation debugger

cādence®

IntelliGen: Gen Debugger



cadence

IntelliGen's High Level Solving

- Propagation-based solver
- Based on the standard backtrack flow
- Variable selection
  - Based on the 'first-fail' principle
  - Involves randomization between variables with similar domains
- Value selection
  - Uniform randomization from the variable's domain
  - Variable domains represent the word-level and the bit-level

Distribution of Random Tests

- "…we want the tests to be distributed as uniformly as possible among all possible tests that conform to the CSP. **Essentially, we want to reach a significantly different solution each time we run the solver on the same CSP**" (Y. Naveh et. al., 2007)
- A good distribution means:
  - All 'interesting' test scenarios are sampled
  - The scenarios are chosen randomly
  - No repetitions of solutions in consecutive runs
  - No bias towards specific cases

Uniform Distribution

- The intuitive solution for generation of random stimuli
  - Part of the SystemVerilog IEEE standard
  - Also suggested by various academic papers
  - Not so easy to achieve:
    - BDD solvers might explode
    - Solving performance might be damaged
- Is uniform distribution really desired?
  - The desired behavior if the problem is symmetric
  - Highly undesired in asymmetric problems
- Uniformity over the set of scenarios, and not over the solution space

Uniform distribution- Example

x is a variable with domain {1..4}

y is a variable with domain {1..2^32}

x=1 -> y=2

- Uniform distribution scenario:
  - (x=1,y=2) has 1/(3*2^32+1) chance to occur
  - The value x=1 will almost never occur

Rx	UNR	Name	Overall Average Grade	Overall Covered
		(no filter)	(no filter)	(no filter)
		x	75%	3 / 4 (75%)
		y	100%	16 / 16 (100%)
		A×B cross_x_y	97.96%	48 / 49 (97.96%)

Uniform distribution- Example

x is a variable with domain {1..4}

y is a variable with domain {1..2^32}

x=1 -> y=2

- IntelliGen’s behavior (much better):
  - x is generated first (smallest domain), and its value is randomized uniformly
  - All four values of x are sampled in equal shares

Ex	UNR	Name	Overall Average Grade	Overall Covered
		(no filter)	(no filter)	(no filter)
		x	100%	4 / 4 (100%)
		y	100%	16 / 16 (100%)
		A×B cross__x__y	100%	49 / 49 (100%)

cadence®

Distribution – Biased values

- Biased values are values which are repeated more than their theoretical chances
  - May cause duplicate tests
  - Requires much longer coverage filling
- Can be caused by randomization of Boolean expressions
  - An expression (e.g. ‘x+y=100’) may be satisfied even when unneeded
  - An example of improved efficiency which harms distribution
- SMT solvers are especially prone to this, assuming:
  - Boolean expressions are translated to the SAT layer
  - Randomization starts with the SAT variables

Biased Values - Example

x,y and z are variables with domain {1..2^32}

x=1 -> y!=1

false -> z=1

- If the Boolean expressions are randomized:
  - x=1 or y=1 will occur in most tests
  - z=1 will occur in half the tests!

item	type	x	y	z
0.	packet	3711715037	2646077124	1
1.	packet	2583696458	2269431876	1
2.	packet	1	1550073392	1
3.	packet	1394952020	850029955	1719951043
4.	packet	1	947226110	1
5.	packet	1	1424182069	325353279
6.	packet	610733067	2570607476	1
7.	packet	1741043782	1	1
8.	packet	1653442553	1	2184797630
9.	packet	1	1310499754	3990453812

cādence®

Biased Values - Example

x,y and z are variables with domain {1..2^32}

x=1 -> y!=1

false -> z=1

- IntelliGen's behavior:
  - x, y and z are generated freely
  - The cases x=1, y=1, z=1 (almost) never occur

item	type	x	y	z
0.	packet	4141845465	2529035483	1719951042
1.	packet	614412132	2155380232	325353278
2.	packet	543210106	2032447663	2184797629
3.	packet	3171158384	3142928999	3990453812
4.	packet	3556484519	1152172248	3318226916
5.	packet	3854000865	4087924829	3801316077
6.	packet	2173679970	1093209980	3288114751
7.	packet	3279981198	1195564507	387709006
8.	packet	4158125639	1230279420	3825377226
9.	packet	1142898690	3840206046	2780807071

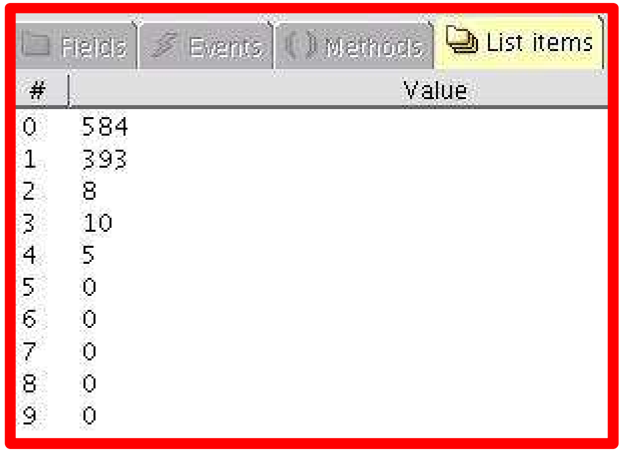
The Global Sum Propagator

- Demonstrates failure of the default CP randomization
- Required specific randomization process

L.sum() = 1000

(L[0] + L[1] + … = 1000)

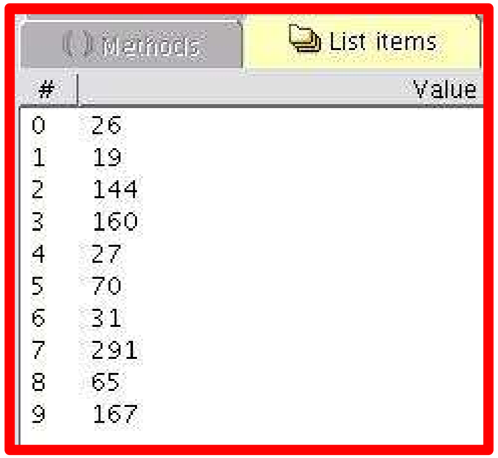
Older IntelliGen



Fields | Events | Methods | List items

#	Value
0	584
1	393
2	8
3	10
4	5
5	0
6	0
7	0
8	0
9	0

Newer IntelliGen



Methods | List items

#	Value
0	26
1	19
2	144
3	160
4	27
5	70
6	31
7	291
8	65
9	167

cadence

Reproducibility

- Reproducibility is a crucial matter for functional verification
- The same test may run several times and in different modes
  - Inside a regression (batch mode)
  - During a debug session (interactive mode, breakpoints, etc.)
  - As a validation of a bug fix (code might have changed)
- Cases which are prone to produce irreproducible tests
  - Non-deterministic algorithms (e.g. parallel search/propagation)
  - Learning solvers: test behavior may be influenced by exact timing of garbage collection
  - Additional of irrelevant code
- More information in the paper…

Summary

- Efficient finding of a feasible solution is not enough
  - Solvers which can handle huge verification environment
  - Handle both single solving and multiple solving of the same problem
  - Randomization which distribute well over the solution space
  - Reproducibility of the solving process
  - Ability for the verification engineer to debug the solving
- Why CP?
  - Light and reusable solvers
  - Flexibility in generation of diverse tests – not just uniform distribution
  - Adapted better to Reproducibility
  - Solving can be explained easier to the users

cadence®