An Efficient Methodology to Find Bugs Using ABV

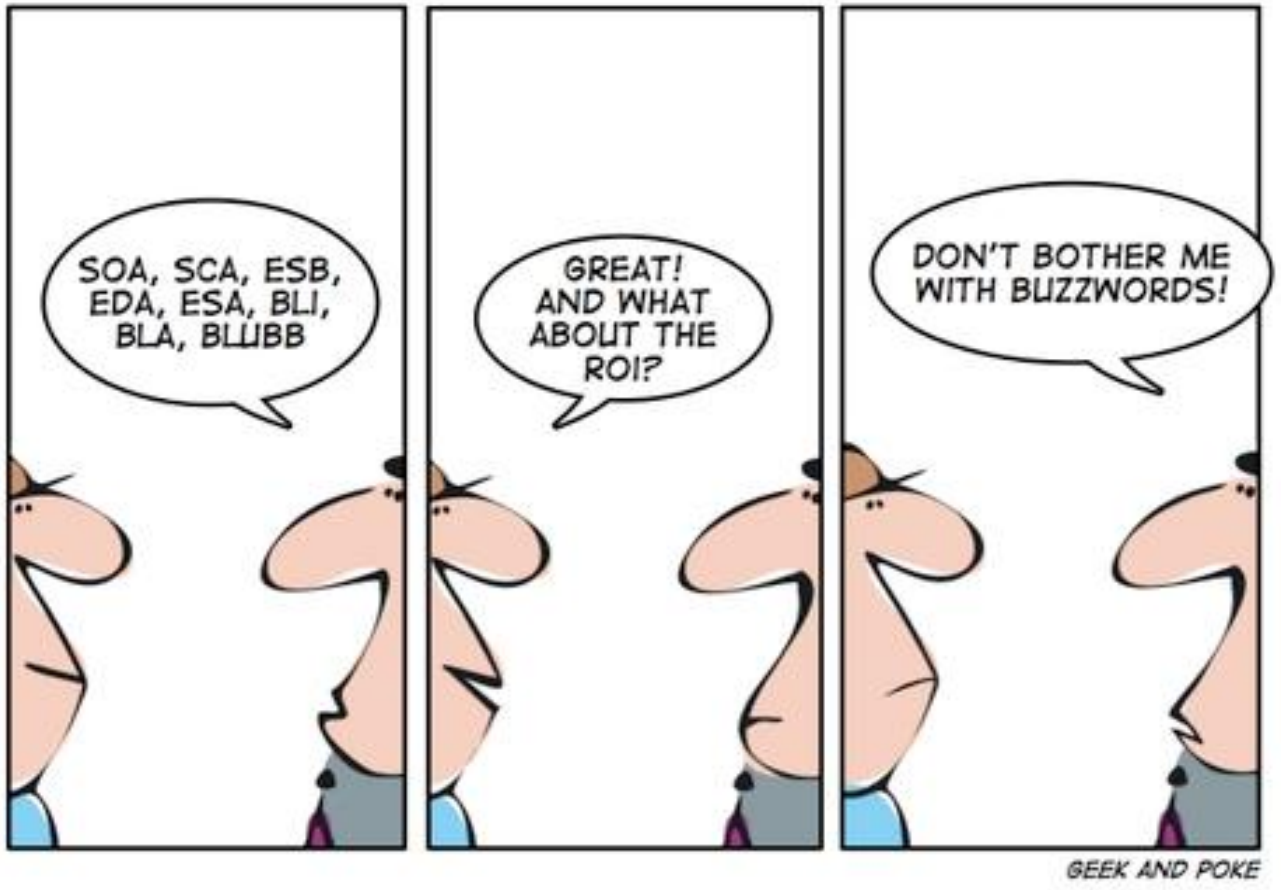
Laurent Arditi, ARM France

DVClub Europe
July 2014

Introduction

- Managers are interested in quality, schedules…

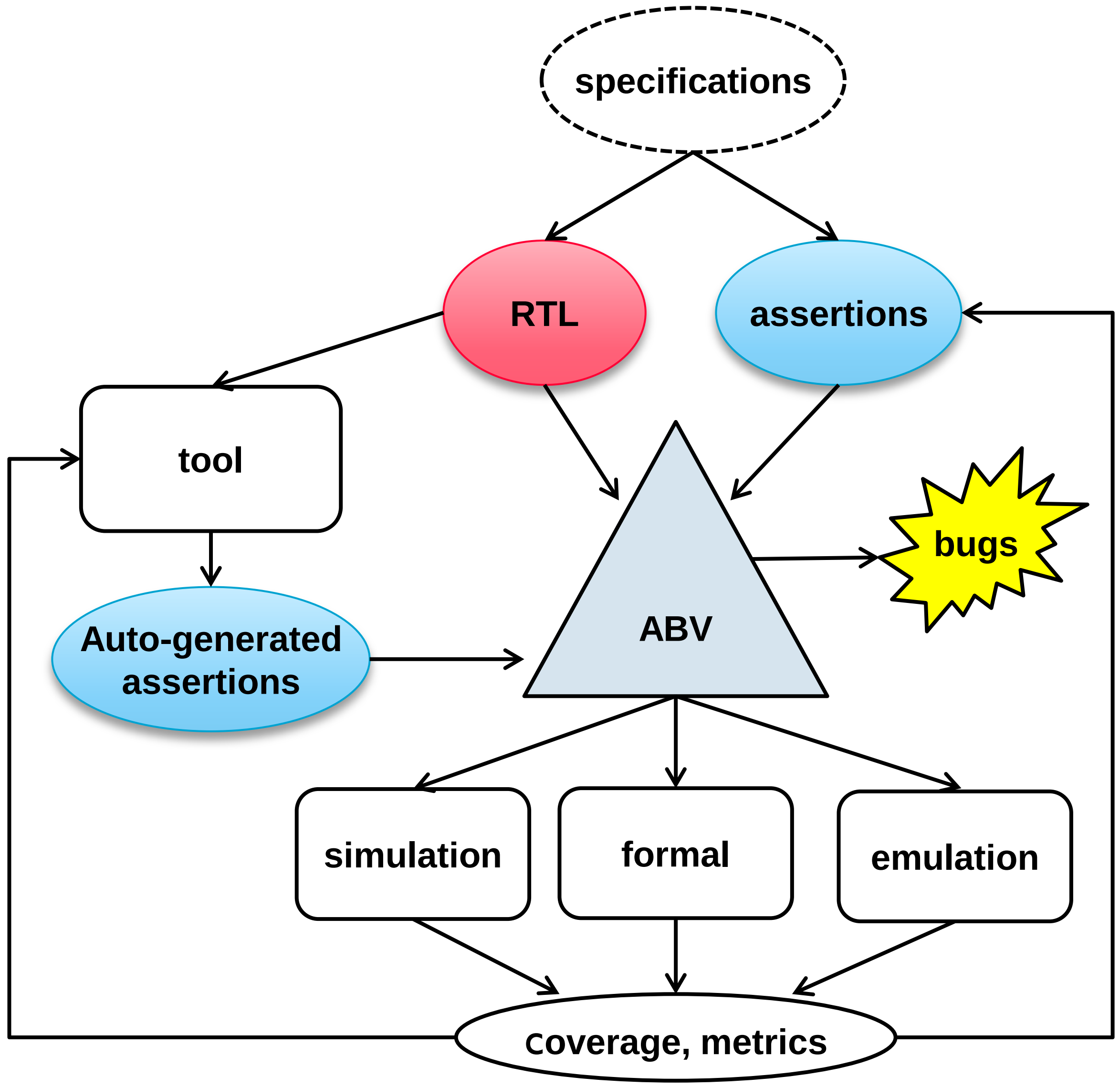
  and Return On Investment (ROI)
- So what's the ROI of ABV, and how do you increase it?


SOA, SCA, ESB, EDA, ESA, BLI, BLA, BLUBB
GREAT! AND WHAT ABOUT THE ROI?
DON'T BOTHER ME WITH BUZZWORDS!
GEEK AND POKE

What is Assertion Based Verification (ABV)?

- A set of assertions are written to verify an RTL design
- Usually embedded assertions: low-level, directly checking design details
- But, other classes of assertions exist and can be part of ABV:
  - X-propagation
  - Standard protocol checkers
  - High-level assertions
  - …
- These assertions are included in most approaches: simulation, formal, emulation, etc.
- I will show how to achieve good RoI with ABV, using real examples from development of real ARM®-based designs
- With some emphasis on **formal** ABV

What is Assertion Based Verification (ABV)?



Embedded (designers) properties in formal

- Must be written for all verification techniques:
  - Simulation
  - Formal verification
  - Emulation
  - …
- Some rewrites to increase formal efficiency:
  - Use of oracle and pseudo-constants
  - Use structural symmetry
  - Split into smaller assertions
- Interface of the main blocks must be completely defined as assertions and checked with ABV
  - Also used to "verify" the simulation testbenches
  - And for black-boxing of some blocks in formal

Embedded (Designers) Properties in Formal

- Write assertions as implications for coverage results
  - Measure the implication ratio; look for weak modules
  - Push conditions to the left making covers more difficult to reach
  - Do not duplicate similar covers

- Disable assertions for similar instances
  - Reduce cluster at a low risk
  - Disable proven assertions in simulation (some tool issues here)

X-propagation

- Simulation has issues with Xs:
  - X-pessimism `X && !X == X`
  - X-optimism `if (X) … else …`
  - Simulators do not deal correctly with Xs: RTL vs. gate differences
  - Simulation tests masking possible X-propagations because of their preamble
- X-propagation with formal
  - Few hand-written properties, on critical places
  - Standard protocol checkers also looking at Xs
  - Auto-generated X-propagation assertions are very good, too
  - Debugging is very easy, especially compared to simulation
  - Some bugs are not real, but it is good practise to fix them anyway
    - Easier to fix than to analyze the potential implications
    - A bug may mask another one
- Now relying on formal to clean X-propagation

Clocking Verification

- More and more clocking schemes; difficult to verify with standard ABV

- Auto-generated assertions are efficient:
  - No input sampling when no clock enable
  - No output change when no clock enable
  - No request when no active clock
  - Equivalency between normal design, and the non-clock-gated one (SEQ problem)

- Some of these are only doable with formal methods
- Most of the clocking bugs are corner-cases because they are very timing sensitive
  - E.g. enter in retention mode exactly one cycle after a given (and infrequent) event combination
  - Formal is very good to find these

High-level Properties

- Very generic since using generic verification components only
- Examples
  - Cache line duplication; looking at content of models for tagrams
  - Coherency in multi-core CPUs
  - Memory system architectural properties (PDF)

- Don’t try to get full proofs for these, nor expect to get very deep exploration depths

- However, with a bug hunting approach real RTL bugs can be found
- Standard interfaces can be verified using protocol checkers available from the EDA vendors, for both simulation and formal

Tricks to Help ABV

- Design mutations
  - Reduce FIFO depths, change arbitration, etc., in order to maximize the probability of getting to ‘tricky cases’
  - Very good for simulation – but must not be always enabled (increases stalls, etc.)
  - Very good for formal too – if implemented correctly wit on/off abstractions, always good

- Dynamic configurations (configure-by-air-wires)
  - Replace static configuration switches by extra – stable – pins
  - Not always feasible (interface change, etc.)
  - Allow to run different configurations without recompile in simulation
  - Allow a single formal run instead of one per configuration
  - Good to perform an equivalence check between configure-by-air-wires and static configurations

Benefits of Using Early ABV

- Designers using ABV (writing assertions) from the very beginning of the design phase

- Allows validation to use formal before simulation testbench availability, or before full checker availability

- On a GPU design, a collection of new modules was verified using formal only
  - Saved resources for simulation on other modules
  - Allowed to focus more on the integration verification
  - Found just a couple of bugs during integration test, showing the high RTL quality

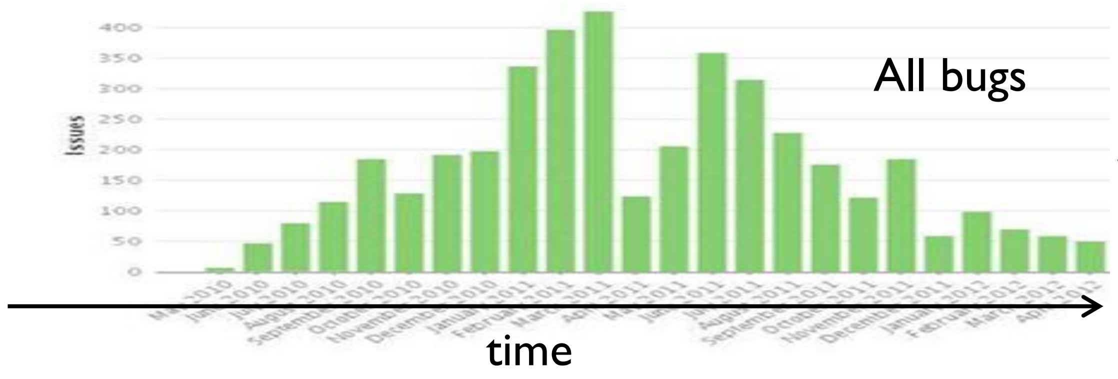
"Formal bring-up was worth the effort and also saved us overall time."

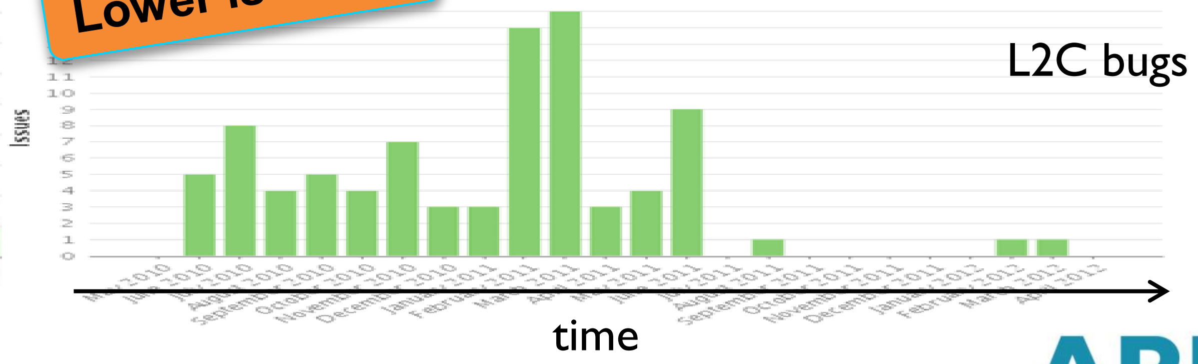
Design Bring-up Benefits

- Another GPU example:
  - No more or no less bugs, but bugs are found much earlier
  - So less RTL changes (code churn), especially late

Block	Formal bring-up usage	Bug density total	Code churn total	Bug density late	Code churn late
L2C	High	6	242	1	10
LSC	Medium	8	577	2	34
LS	No	27	460	13	171
TEX	No	8	369	4	54
JM	No	6	265	3	53
HT	No	10	254	4	71

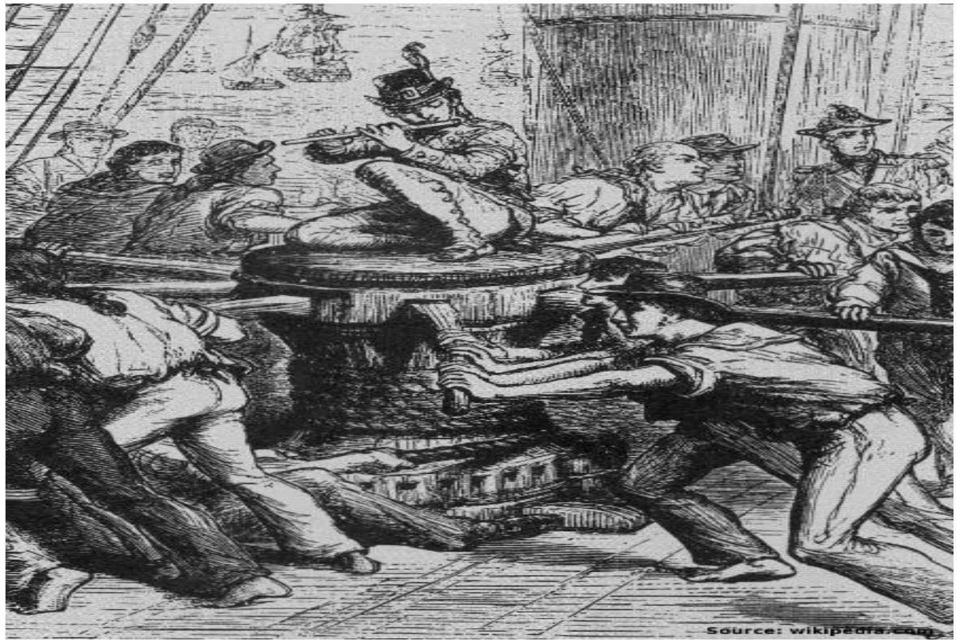
Lower is better





Bug Find / Fix Cycles

- Early phase
  - Unit-level bugs found and fixed by (or close to) designers
  - Bugs easily identified and fixed
  - Usually local scope, so global side-effect risk is minimized
  - Short or non-existent review and tracking process



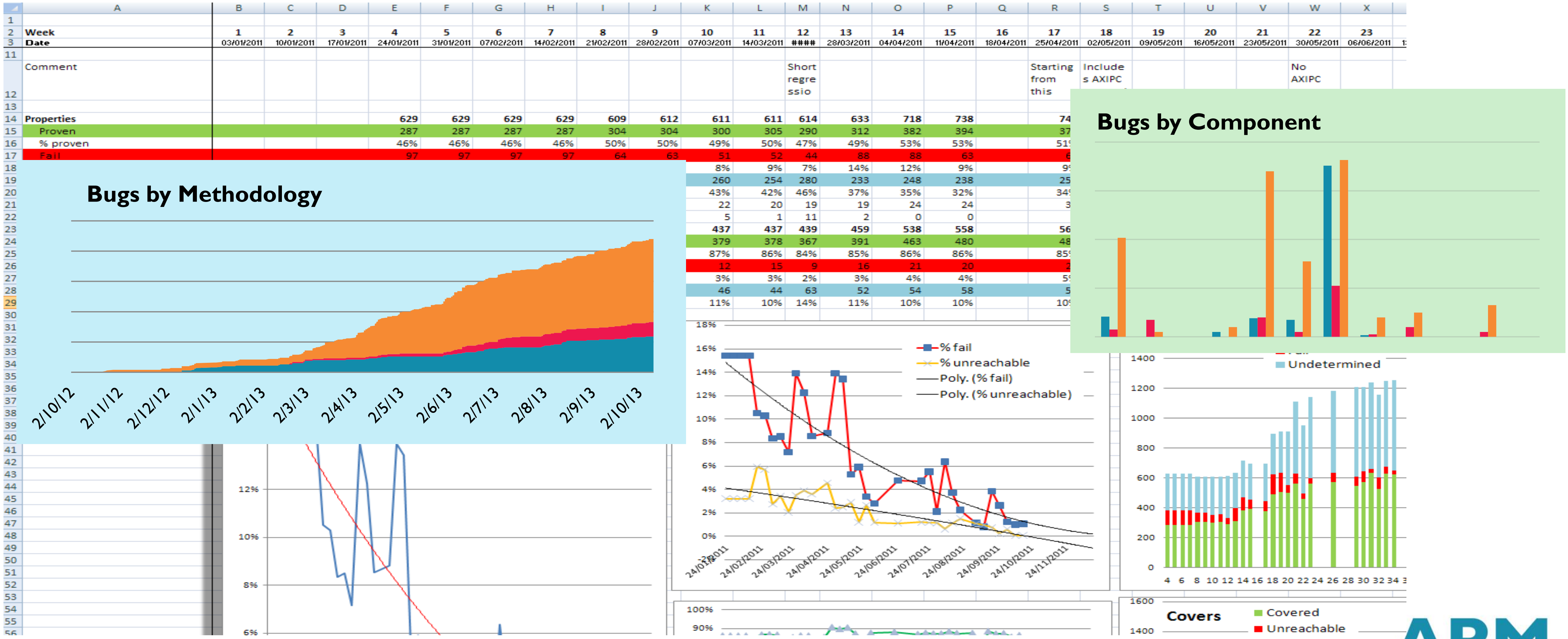
- Late phase
  - Bugs found by validation team
  - Deeply buried unit-level bugs are hard to identify from system-level environment
  - Global side-effect risk from changes is high
  - More restrictive change process (review, code control, re-validate)

Under Investigations: ABV Coverage

- *Verify the Verification*: how good is my verification, and when am I done?
  - Techniques exist, based on code mutations:
    - If the RTL code is modified, is the change detected by simulation tests, or by assertion failures in formal?
    - Needs lots of reruns; difficult on big designs
  - Or, using formal techniques to answer:
    - What is statically covered by the asserts?
    - What is really covered when proving the asserts?
    - What is covered by the undetermined asserts?
  - Help understand where/how to push ABV further:
    - Parts of the design which are insufficiently covered by the existing assertions
    - Assertions which are not strong enough, and not explored enough

What to Show to Your Manager?

~~Nothing… until *'the design is fully-proven'*~~



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
1																								
2	Week	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
3	Date	03/01/2011	10/01/2011	17/01/2011	24/01/2011	31/01/2011	07/02/2011	14/02/2011	21/02/2011	28/02/2011	07/03/2011	14/03/2011	####	28/03/2011	04/04/2011	11/04/2011	18/04/2011	25/04/2011	02/05/2011	09/05/2011	16/05/2011	23/05/2011	30/05/2011	06/06/2011
11																								
12	Comment												Short regre ssio					Starting from this	Include s AXIPC				No AXIPC	
13																								
14	Properties				629	629	629	629	609	612	611	611	614	633	718	738		74						
15	Proven				287	287	287	287	304	304	300	305	290	312	382	394		37						
16	% proven				46%	46%	46%	46%	50%	50%	49%	50%	47%	49%	53%	53%		51						
17	Fail				97	97	97	97	64	63	51	52	44	88	88	63		6						
18											8%	9%	7%	14%	12%	9%		9						
19											260	254	280	233	248	238		25						
20											43%	42%	46%	37%	35%	32%		34						
21											22	20	19	19	24	24		3						
22											5	1	11	2	0	0								
23											437	437	439	459	538	558		56						
24											379	378	367	391	463	480		48						
25											87%	86%	84%	85%	86%	86%		85						
26											12	15	9	16	21	20		2						
27											3%	3%	2%	3%	4%	4%		5						
28											46	44	63	52	54	58		5						
29											11%	10%	14%	11%	10%	10%		10						

Reporting ABV Results

- Collect data on a regular basis, show the cost and benefit evolutions

- Some useful numbers:
  - Number and density of assertion per block
  - Percentage of covered, proven, fails, etc.
  - Highlight the covered and fails, not the proven (managers don't care about provens!)
  - Number of bug founds
    - By different techniques (simulation, formal, emulation, FPGA)
    - By classes of assertions
    - By milestones
  - Exact cost of engineering work and infrastructure
  - Compare all techniques

Conclusion

- ABV is a powerful verification methodology
    - Fits well for both simulation and formal
    - Delivers a high RoI, if used carefully
    - Must be completed with other checkers (for simulation)
    - Can be completed with automatically generated assertions for specific verification aspects

- Needs information about ABV coverage and completeness
    - Techniques exist, but are not mainstream yet