Formal weapons for microprocessor verification

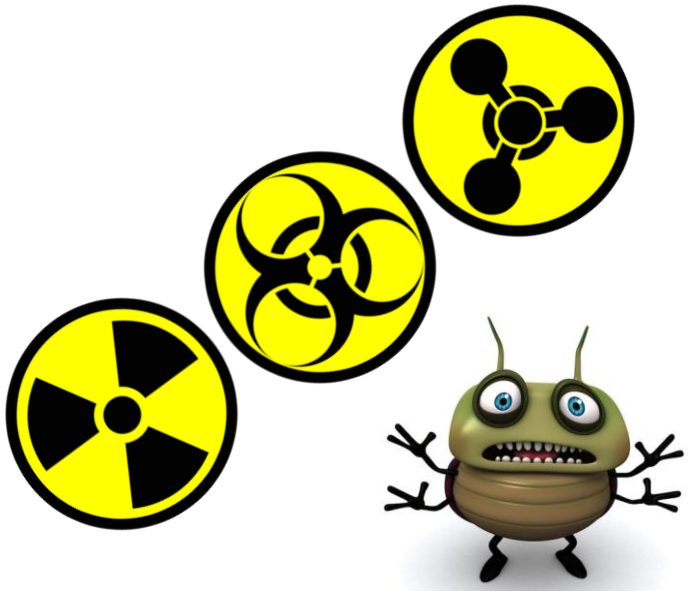
TVS Formal Verification conference, May 2015

Laurent Arditi Olivier Ponsini Anne-Claire Berger

ARM France

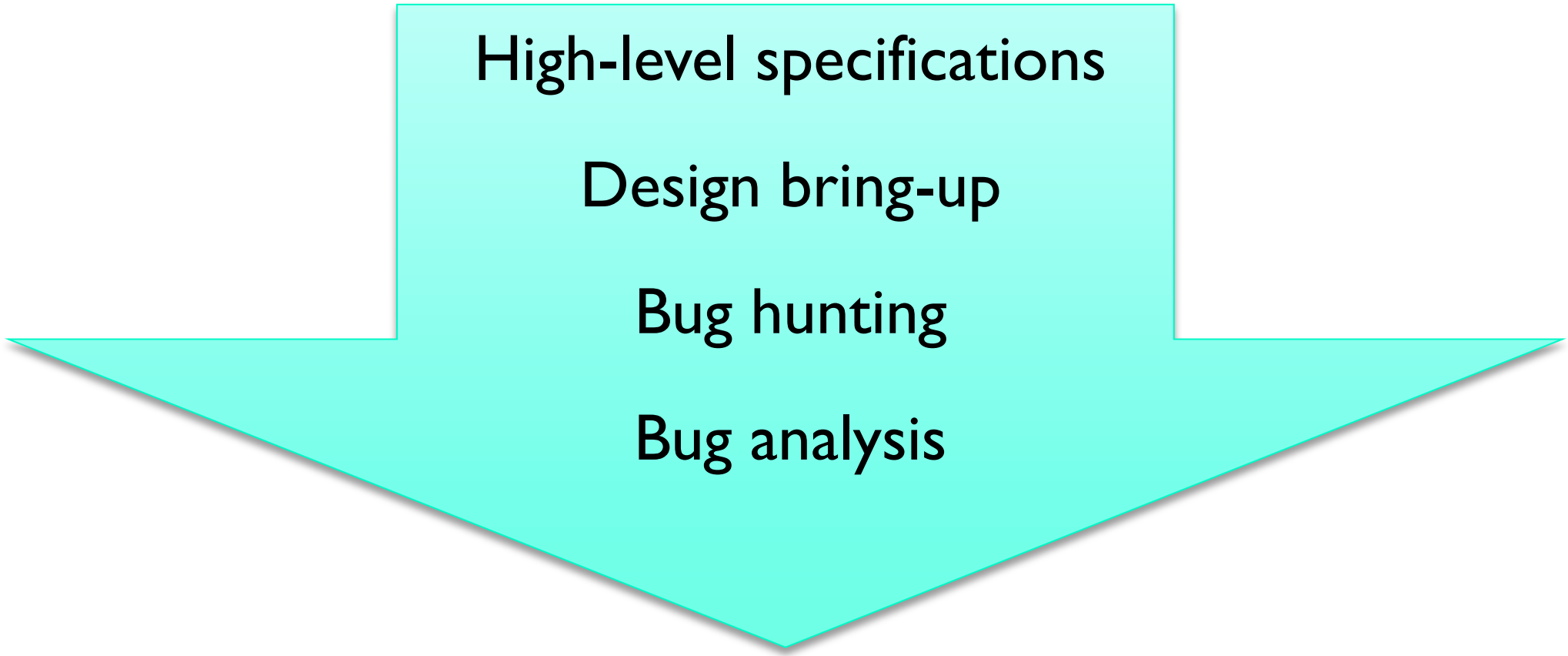
Is formal a weapon of mass destruction (of bugs)?

- Simulation does not scale up with design complexity without prohibiting costs
  - A (million) test is just a (million) test
- Formal verification complements simulation
  - Good for finding rare, potentially severe, bugs
  - Good for proving "correctness"
    - Not aiming at completeness
    - Increase confidence
  - Formal-only verification tasks (x-prop, clock enable)
  - Efficient
    - Smaller team
    - Better cluster usage/bugs found ratio than simulation



ARM

Formal in the development process



ARM

High-level specifications

- Verify completeness and correctness at architectural level
  - Coherency protocol in multi-core CPUs
  - Memory system
- Formal specification
  - Prove properties about the specification
  - Success stories with models
  - Still challenging with real implementations
    - Needs helper properties for proofs
    - Finds real RTL bugs

ARM

Design bring-up

- Aid for design during RTL development and on new features
  - Simulation testbenches may not be ready long before new features are already implemented
  - Getting a simple working formal setup is relatively fast
- Basic properties
  - Check the RTL is not completely broken
  - Check assumptions on signal properties and equivalence
- Catch bugs early
- Formal counter-examples easier to debug than simulation failures

Bug hunting

- Find bugs at top and block levels
- No effort on proofs
  - Dedicated tool engines
  - Dedicated techniques
- Very computer intensive
  - Several configurations
  - Thousands of properties
  - Some need days to fail
- Automation and regression
  - Memory usage is unpredictable
  - Cluster friendly

ARM

Bug analysis

- Investigate around a specific bug
  - Reproduce bug
  - Find similar bugs
  - Check bug fixes
- Difficult to hit in simulation
  - Found by human review
  - Observed in field

Formal weapons

- Superlint (Autochecks)
- Protocols
- X-propagation
- Embedded assertions
- FSM
- Clock-gating
- SEC
- System registers
- Coverage

Successfully applied to processors designed at Sophia-Antipolis:

- ARM® CORTEX®-A9 processor
- ARM® CORTEX®-R7 processor
- ARM® CORTEX®-A12 processor
- ARM® CORTEX®-A17 processor
- …

ARM

Superlint (Autochecks)

- Check assertions for:
  - Overflows
  - Out-of-bound indexing
- Automatically generated
- Waiver mechanism is mandatory
- Meticulous lint tool

Protocols

- Certify compliance with standard protocols
  - AXI, ACE, AHB, ATB, APB
- Protocol checkers integrated into EDA soluations
  - Can be used as master or slave
  - Highly configurable
  - Optimized for formal

X-propagation

- Detect and debug X-propagation issues on RTL
- Simulators do not deal correctly with Xs (optimism and pessimism)
- Formal is our exclusive approach
  - Few hand-written assertions
  - Protocol checkers
  - Mostly auto-generated properties
    - Flops with a reset
    - Flop enable conditions
- Proofs are very rare but we find many bugs
  - Debugging is very easy, especially compared to simulation
  - Some false alarms, but it is good practice to fix them anyway

ARM

Embedded assertions

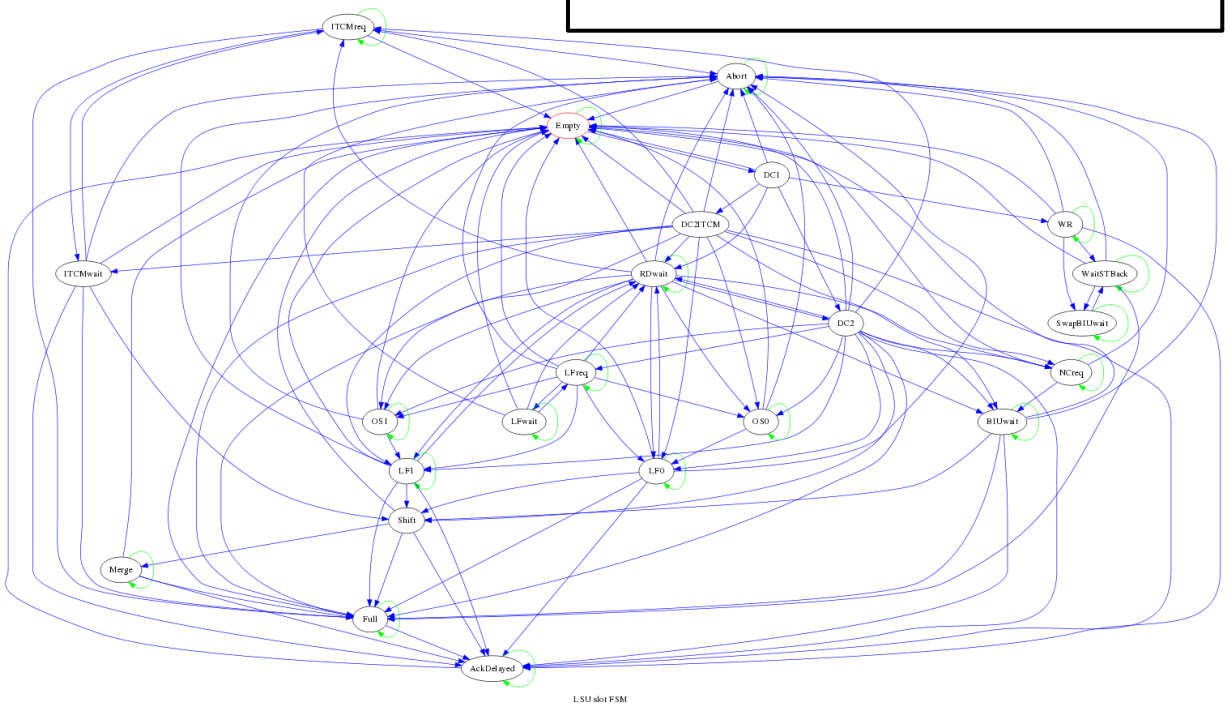
- Check designers' assertions
  - Written for both simulation and formal
- Maximize the use of implications
  - Yields coverage points for free
- Assertion density metrics
  - Modules relatively to each others
- Several flows
  - Simple flows for designers
  - Regression flows: cluster friendly for x1000 properties
  - Soak flows: "random" assertions running for days

Finite State Machines

- Check consistent behavior of implemented FSM
- Simple textual FSM specification
  - States
  - Transitions
- Automatic generation of properties
  - State reachability
  - Transition conformity

lsu_slot_fsm_val.fsm

```
STATE Empty   : slot_state_is_empty
STATE LF0     : slot_state_is_lf0_hit
STATE DC1     : slot_state_is_dc1_q
...
INIT Empty
TRANS Empty : DC1
TRANS LF0 LF1 DC2 : Shift
...
```



ARM

Clock gating and slow clock verification

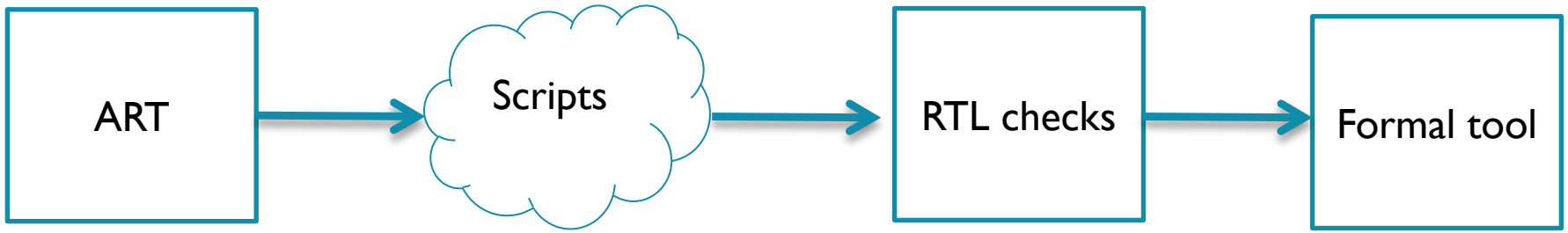
- Check the logic introduced for clock gating
  - Complex clocking schemes in low-power designs
  - Difficult for simulation
- 3 verification goals
  - Regional clock gating
  - Output stability
  - Input sampling
- Dedicated properties
  - Some hand-written properties for regional clock gating
  - Automatic generation from script for other assertions
- Found many bugs
  - Proved much more efficient than simulation

Sequential Equivalence Checking

- Prove that two different circuit descriptions exhibit exactly the same functional behavior
  - Design mutations
  - Same design with and without a given feature
  - Optimizations in Floating-Point units
  - Mid-level clock-gating
- Can only be handled by formal methods
- Advanced techniques used to get exhaustive proofs

System registers

- Check accesses to system registers
  - Initial value
  - Read and write accesses
- A common register description reference: ART
  - Maintained by designers
  - Used for design, simulation, and formal verification
- An automatic flow to generate assertions
  - Saves time
  - Ensures exhaustiveness



ARM

Coverage

- Check reachability of RTL code branches and statements (Cover Items)
- Allows various analyses
  - Reports dead code
  - Generate waivers for simulation code coverage
  - Help filling the latest % in simulation code coverage
  - Increases confidence in the formal environment not over-constraining the design
  - Shows parts of the design weakly covered by assertions
- Towards a formal coverage metric
  - Reports Cover Items explored by abstraction engines
  - Approximates progress of formal verification effort

Formal helpers

- Mutations
  - RTL changes to reach corner-cases in fewer cycles (e.g. FIFO reduction).

    Used in simulation too. Non-deterministically enabled in formal
- Initial value and other abstractions
  - Skip “configure and populate” cycles to reach interesting cases faster
  - Skip irrelevant logic

Considerations for formal efficiency

- Check properties on an internal part of the design in isolation
  - Smaller design size, fewer cycles needed to reach interesting scenarios
- A complete specification of internal interfaces is costly
  - But good for informal validation too and helpful when the design must be modified
- Leverage the diverse formal techniques
  - A portfolio of engines with a smart selection, a parallel race and good orchestration
  - Efficient usage of the cluster

Conclusion and perspectives

- Designers' feedback
  - Few false alarms
  - Debugging formal fails is often easier than simulation fails
  - Undetermined properties are well accepted, provided exploration depth is reasonable
- Managers' feedback
  - Formal is part of validation sign-off
  - Reporting status and showing progress is crucial
- New formal weapons
  - IP-XACT verification
  - Security verification
  - UPF/CPF low-power aware verification

Thank you

ARM