DESIGN & VERIFICATION DVCON 2014 CONFERENCE & EXHIBITION

Practical Approach Using a Formal App to Detect X-Optimism-Related RTL Bugs

Shuqing Zhao, Shan Yan, Yafang Feng
Broadcom Corporation

Agenda

- Problem Statement
- Related Work
- Methodology Outline
- Case Study 1: Power Management Controller
- Case Study 2: Audio Processor
- Summary

X Sources

- What is X? Simulation logic state — unknown (0 or 1)
- X Sources that can potentially hide RTL bugs:
  1. Uninitialized variables—nonresettable registers
  2. Out-of-bound bus bit select or array access
  3. Explicit X assignments that are reachable
     - "Don't care" used for logic minimization purpose
     - Modeling unknown, e.g., memory output or gate output with no power
     - Modeling error cases—should be replaced with assertions
  4. Power-aware semantics specified by UPF/CPF
- Less likely RTL bug sources:
  - Floating wires/ports, multiple drivers—detected by lint tools
  - Timing violations

Problem Statement

- X-Optimism in standard RTL simulation is dangerous and causes bug escape.
  - X-Optimism: “optimistically” resolving X, not matching silicon behavior

```
always @(posedge clk or negedge rst_n)
    if (!rst_n)
        count <= 0;
    else if (count_enable)
        count <= count + 1;
```

4 2 3 1

  - Problematic Verilog constructs: if, case, posedge, negedge
- Gate-level simulation is not adequate.
  - It suffers from X-Pessimism: e.g., (a & ~a) should not be x.
  - It usually is a subset of RTL simulation coverage.
  - Bugs found at this late stage require costly resynthesis or ECOs.

Related Work

- RTL coding style to explicitly propagate X.
  - Readability problem.
  - It's not practical to intercept all combinations.
- 2-state simulation—randomize X to 0 or 1.
  - Complete coverage is not practical.
- Model checking using 4-state:
  - Suffers usual formal capacity limitation—bounded proofs.
  - Depends on completeness of assertions created manually by users.
- Model checking of X checkers is automatically inserted.
  - Basis of our approach

Formal X-Propagation App

- How the formal X-prop app works:
  - Environment input: clocks, resets, modes, X to input ports.
  - Analyzes/elaborates RTL into an internal “netlist.”
  - Automatically creates the X detection assertions on targets:
    - Clocks and resets
    - Output ports
    - Test condition for if/case statements
    - User-specified signals
  - Prove that X’s cannot propagate to targets.
  - A counter-example is generated if any assertion fails.
- Main issue: some assertions will get full proof and some will only get bounded proof.
- Our contribution: a methodology to improve ROI.

X Source-Driven Methodology

Coding
- Async reset, use assertions for X, avoid casex/casez, etc.

Scan for X's
- Lint, "Formal" reset analysis, Structural property analysis, X-assignment reachability

Formal App
- Where can X propagate to?

Debug
- Counter-examples for failed assertions

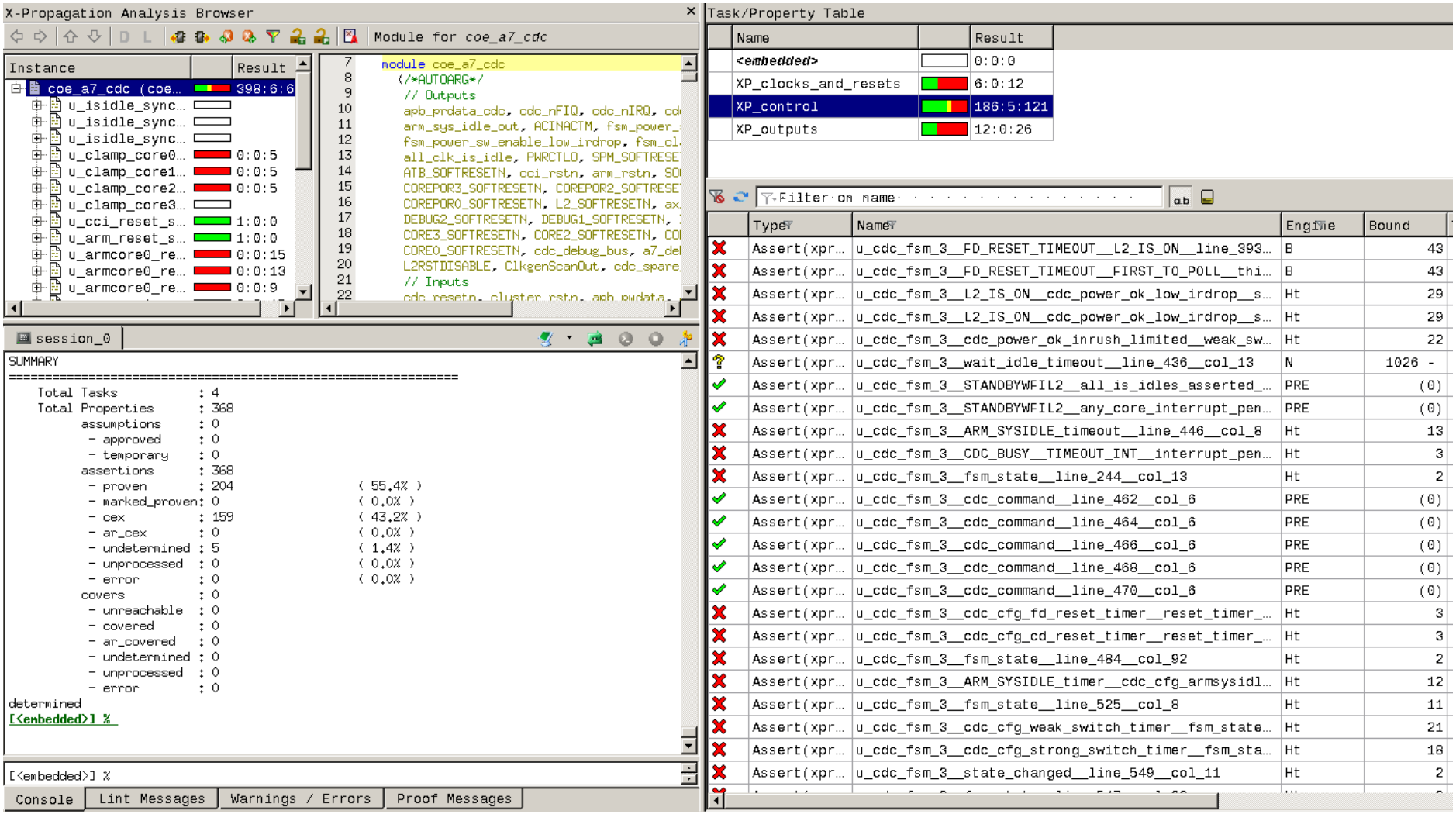
X-prop Sim.
- Cover unproved X assertions
- Power-aware simulation with UPF/CPF

Case Study 1: PM Controller

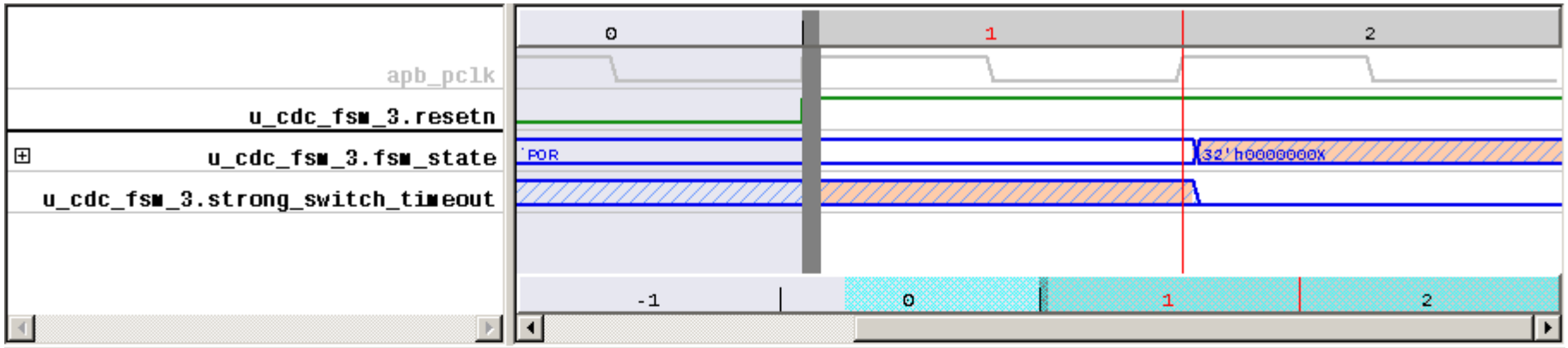
- Background
  - DUT: Critical power management controller for quad core app processor
  - Presilicon verification
- Process
  - Ran very comprehensive UVM constraint random simulation.
  - Used formal reset analysis to check for uninitialized flops.
  - Used the formal X-prop app to scan for targets X can propagate to.
- Results
  - Reset analysis found four flops that were not initialized by reset.
  - X-prop app detected one flop that caused a nasty bug in the state-machine transition after reset.

DESIGN & VERIFICATION DVCON 2014 CONFERENCE & EXHIBITION

XPROP App GUI Report



RTL Bug Counter Example



Source Pane

Search the Source ...

Why at iteration 1 for *u_cdc_fsm_3.fsm_state* (1

```
243
244         case (fsm_state)
                  `POR
245          `POR: begin //POR state
246             if (strong_switch_timeout) begin
                     1'bx
247                if (this_is_core_0)
                        1'b0
248                  fsm_state <= `RESFDM;
249                else
250                  fsm_state <= `RESFD_WAIT; // no default here, go one way or the other
251             end
252             else fsm_state <= `POR;
253          end
254
255          `RESFDM: begin  // Resume from full dormant as master (core_0 only)
256             if (FD_RESET_TIMEOUT & (cdc_command == `MDEC)) begin
257                fsm_state <= `RUN;
258             end
```

Case Study 2: Audio Processor

- Background
  - DUT: audio processor module
  - Post-silicon analysis of bug escape
  - Bug: one of the channels was hanging the system.
- Process
  - Gatesim reproduced the bug.
  - Used the X-prop app to scan for targets X can propagate to.
- Results
  - X-prop app confirmed out-of-bound array access due to an RTL coding error.
  - The SPS app caught the same bug in a shorter runtime.
  - Xprop app proved absolute safety of a software workaround.

Summary

- Conclusions
  - X-optimism causes RTL bugs to be missed from simulation.
  - Formal approach is exhaustive but has limitations.
  - We recommend an X source-driven formal approach.
  - We found show-stopper bugs using this approach.
- Future Plan
  - Run X-prop app + low-power formal app to detect X issues from UPF.
  - Study formal code coverage relationship for bounded proof.
- Acknowledgements
  - Jennifer Hwang from Broadcom

Thank you!