ASPDAC 2010

Managing Verification Error Traces with Bounded Model Debugging

Sean Safarpour
Vennsa Technologies

Andreas Veneris, Farid Najm
University of Toronto

Outline

- Motivation
  - The debugging problem
- Background
  - Debugging in practice
  - Automated debugging
- Bounded Model Debugging
- Experiments
- Conclusion

The Debugging Problem

- Debugging can take up to 60% of verification
  - Designs and blocks are large
  - Verification environments are complex
  - Design and verification performed by different people
  - Use of 3rd party and legacy IP, many languages, etc.

- Debugging starts as soon as verification fails
  - model checking, simulating testbench, etc.

The Debugging Problem

- Goal of debugging
  - Identify the root cause of failure: localization
  - Make fix such that problem is resolved: remove bug
- Debugging is still not automated in design cycle
  - Performed manually by engineers
  - Similar to decades ago
- Automated design debugging and diagnosis techniques exist for decades: error/fault localization
  - Traditional: Simulation, path tracing, BDDs
  - Recently: SAT, QBF, Max-SAT, unsat cores

The Debugging Problem

Debugging Today:

Automated Debugging in near future:

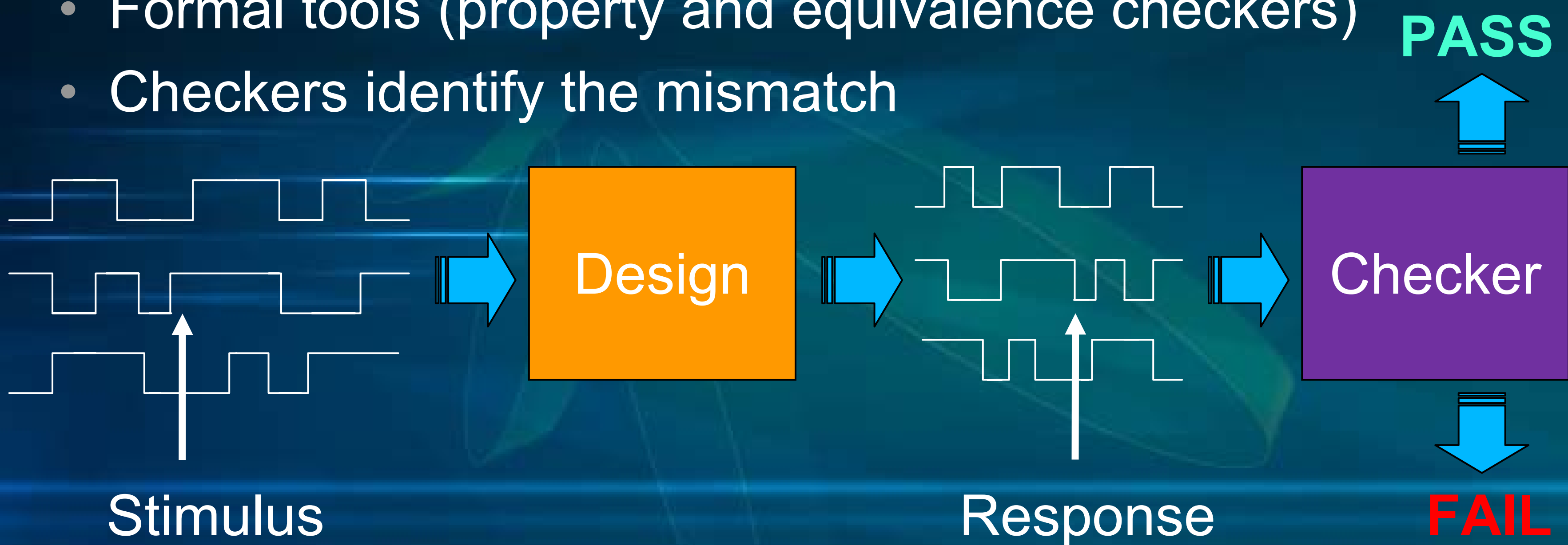
- Main challenge is scaling
  - Handling larger designs, longer trace lengths
  - Memory requirements and run-time
- Bounded Model Debugging aims to reduce the problem size → Tackle larger problems

Outline

- Motivation
  - The debugging problem
- Background
  - Debugging in practice
  - Automated debugging
- Bounded Model Debugging
- Experiments
- Conclusion

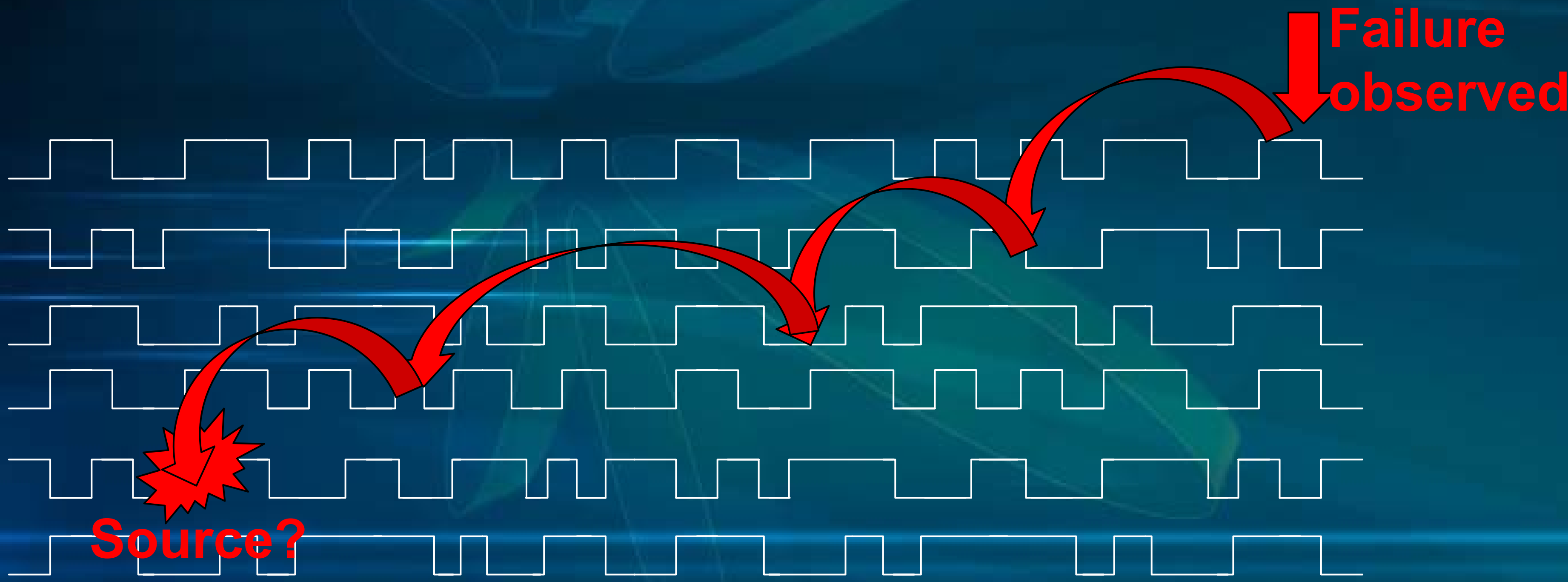
Background

- What leads to debugging?
  - Design behavior does not match expected behavior
- When does this occur?
  - During simulation of design
  - Formal tools (property and equivalence checkers)
  - Checkers identify the mismatch



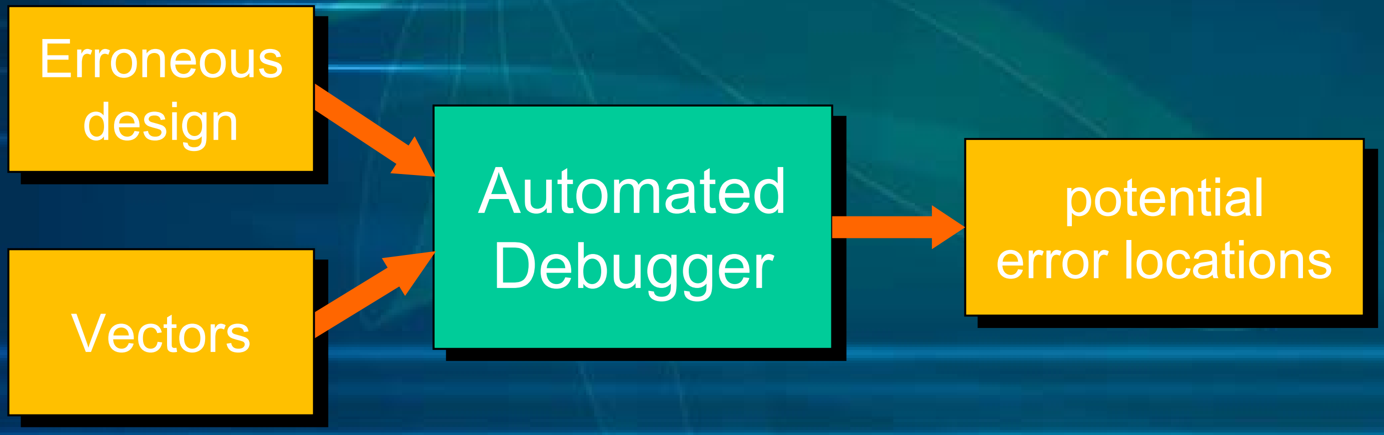
Background

- In practice, engineers debug by investigating sequences from failure point backwards in time
- Process is manual and time consuming
- Inconsistencies or unexpected behavior is analyzed in detail



Background

- Many automated debugging techniques exist
  - Path tracing, simulation, symbolic (BDD), SAT-based
- Debugger can be viewed as a black box
  - Provide erroneous design (with bug)
  - Vectors: stimulus and correct/golden response
  - Output is error locations or suspects



Background

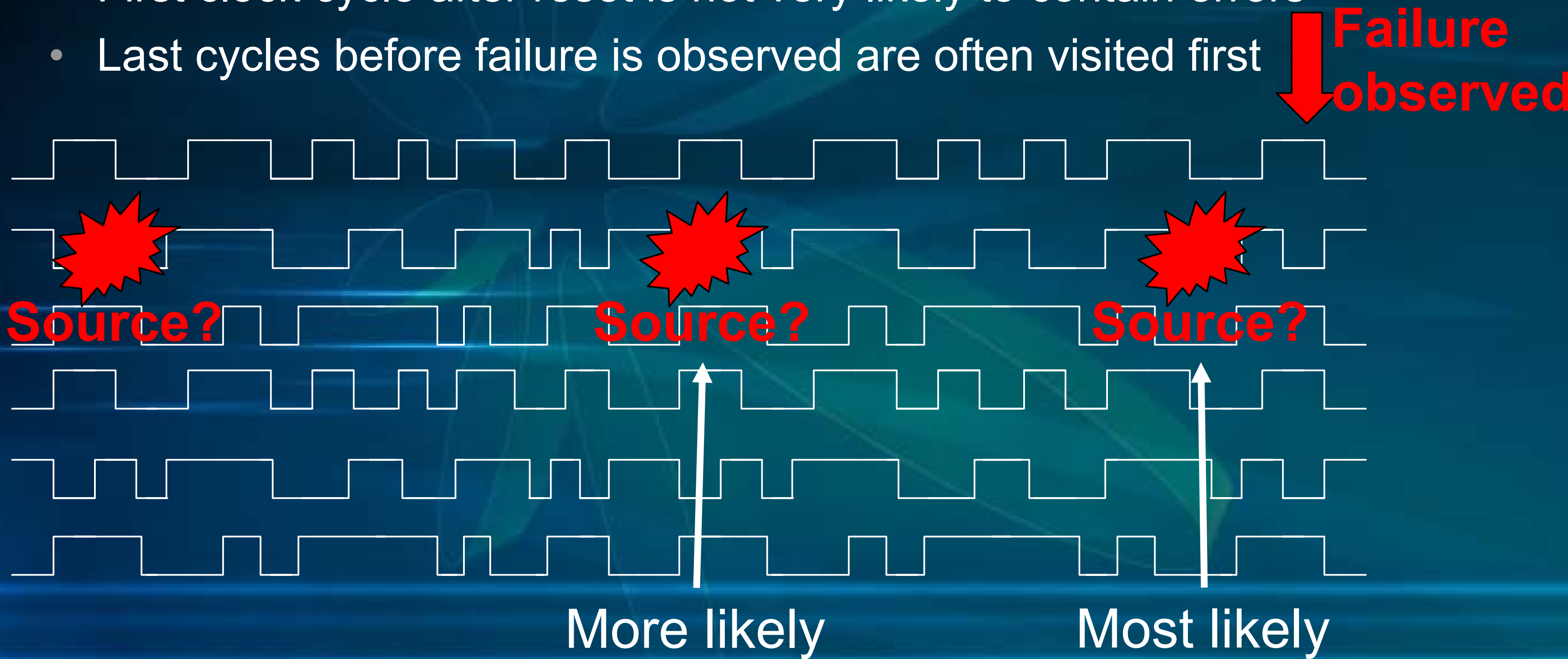
- Automated debugging is hard problem
  - Problem traditionally tackled at gate-level
    - Algorithms work on gate/Boolean models
    - circuits blocks are big: > 100,000 gates
  - Algorithms must model sequential behavior
    - traces are long: > 1000s of clock cycles
  - Complexity grows $O(n^{\#\ \mathrm{errors}})$
    - few high-level errors translate to many low-level errors
- SAT and QBF approaches are most effective

Outline

- Motivation
  - The debugging problem
- Background
  - Debugging in practice
  - Automated debugging
- Bounded Model Debugging
- Experiments
- Conclusion

Bounded Model Debugging

- Observation:
  - error source(s) are often excited temporally close to failure points
- Evidence: debugging in industry
  - First clock cycle after reset is not very likely to contain errors
  - Last cycles before failure is observed are often visited first



Bounded Model Debugging

- Two events must occur
  - Error must be excited
  - Erroneous behavior must be propagated to observable point
    - Primary input with correctness model (ex. Assertion)
- Using simplified model, probability of error source can be calculated
  - In given clock cycle *d* the probability is p_d

$$p_d = \prod_{i=1}^{d-1} prop_i \times \prod_{i=1}^{d-1} (1 - obs_i) \times obs_d$$

Bounded Model Debugging

- Bounded Model Debugging is based on concept of error source temporal proximity to failure point
- BMD iteratively solves debugging problem
  - Starts with cycles closest to the failure
  - Uses conventional automated debugger to find error source
  - If required, prior cycles are analyzed
  - Procedure is complete: finds all bugs
- Benefits
  - Requires much less memory
  - Faster run-time
  - Provide user suspects prior to completing the entire process

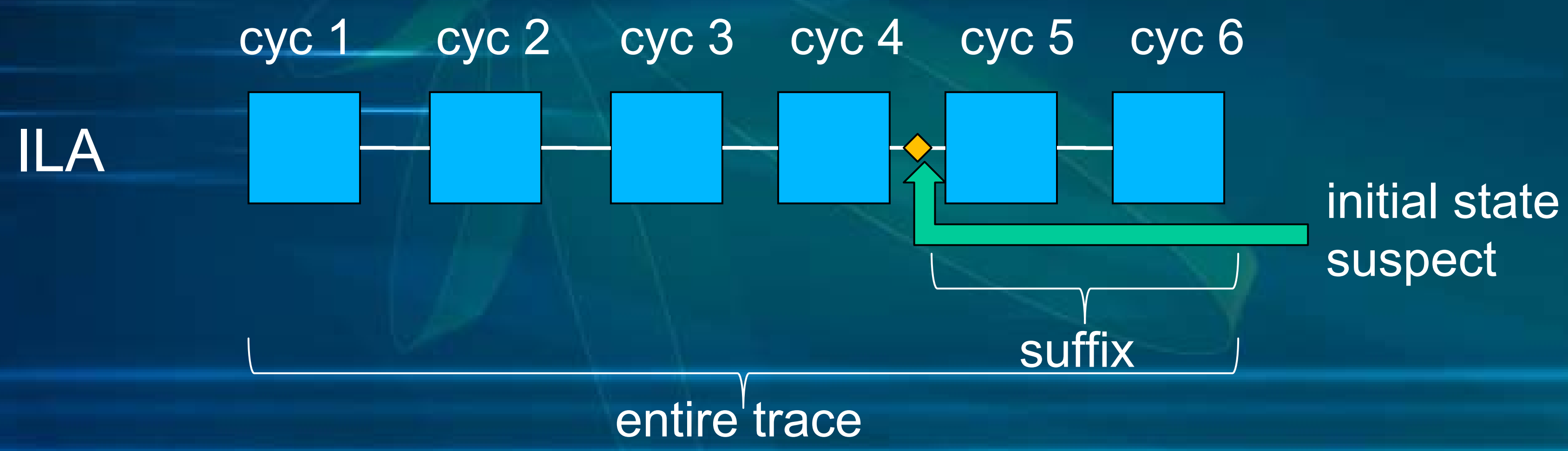
Bounded Model Debugging

Example:

- Simulate design for 100 clock cycles and observe a failure
- Conventional debugging:
  - Build ILA for 100 clock cycles, formulate SAT problem, solve using SAT-solver, results are the suspects
  - Result: Run out of memory for industrial circuits
- Idea: build ILA for 10 cycles
  - Simulate for 90 cycles, solve debugging problem for 10 cycles (suffix)
  - Result: errors between 90 to 100 are found, use much less memory
- Question:
  - What if error sources is excited prior to 90^{th} cycle?

Bounded Model Debugging

- Question:
  - What if error sources is excited prior to 90th cycle?
- Answer:
  - Must increase suffix length (to 20, 30, etc.)
- Completeness
  - Need special type of suspects to identify incomplete solutions
  - *Initial State Suspects* determine if error may be excited in prior cycles

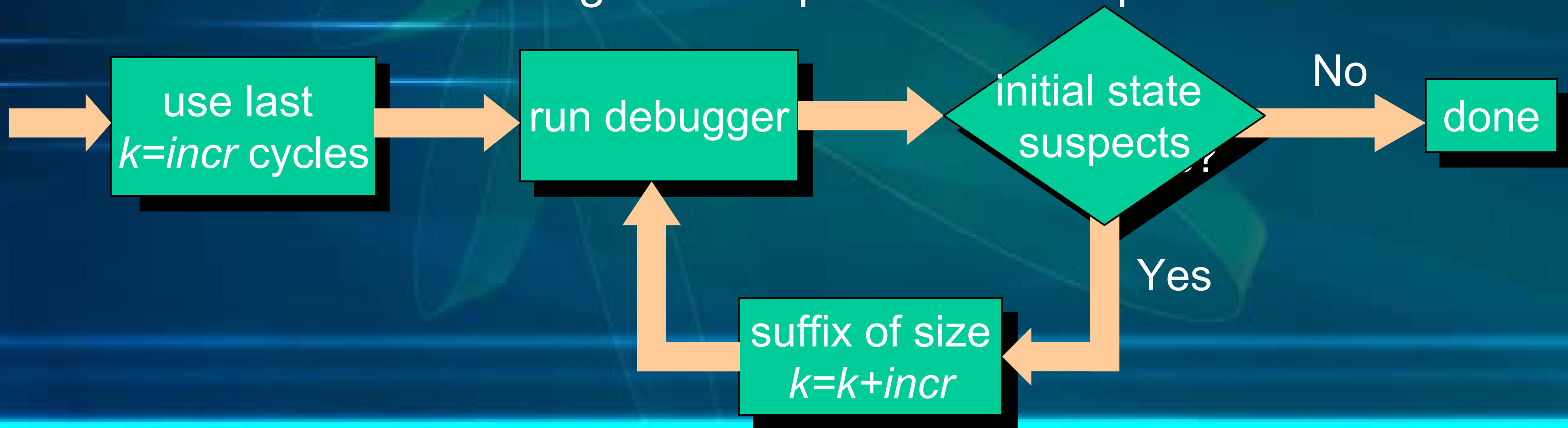


Bounded Model Debugging

- Initial State Suspects
  - Used to find whether an initial state value can be error source
  - Can be a simple MUX on initial states in SAT-based debugging
- Impact on error cardinality
  - If looking for N errors, with BMD must look for N + # state elements
  - Complexity of debugging grows exponentially with cardinality
    - Big problem!
- Solution
  - Group initial state suspect together
  - If one initial suspect is found need to increase suffix length
  - Error cardinality increases to N + 1

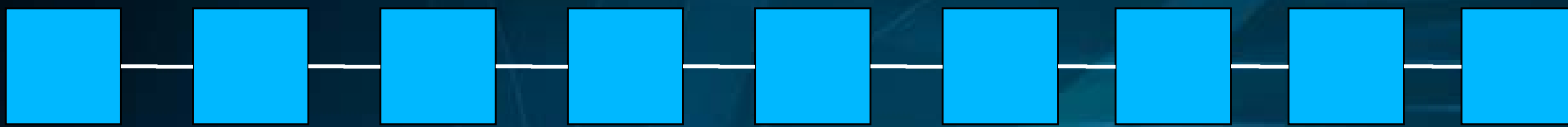
Bounded Model Debugging

- Formulation
  - Consider the last *incr* clock cycles: *suffix*
  - Use initial state suspects to identify errors excited prior to last *incr* cycles
  - Solve debugging problem using suffix
  - If initial state suspects are found as solution
    - Increase suffix length and repeat above steps

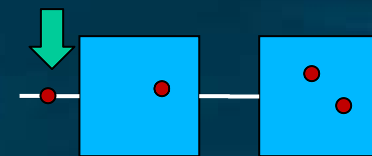


Bounded Model Debugging

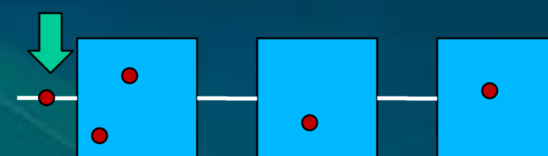
Example: trace of length 9 requires too much memory



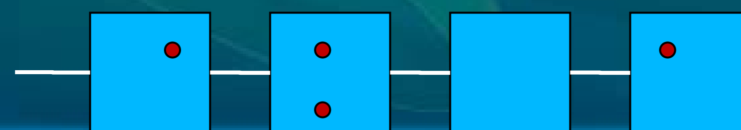
Use suffix of 2 cycles: Find initial state suspects



Use suffix of 3 cycles: Find initial state suspects



Use suffix of 2 cycles: Find all solutions



Outline

- Motivation
  - The debugging problem
- Background
  - Debugging in practice
  - Automated debugging
- Bounded Model Debugging
- Experiments
- Conclusion

Experiments

- Designs in Verilog from OpenCores.org and industrial designs
- Correct behavior provided by testbench with expected model or by assertions
- Functional errors are injected into designs
  - Incorrect statements (operator, concatenation, etc.)
  - Incorrect module port connections
  - Wrong state transition, etc.
- Results compared against SAT-based hierarchical debugger with/without proposed technique

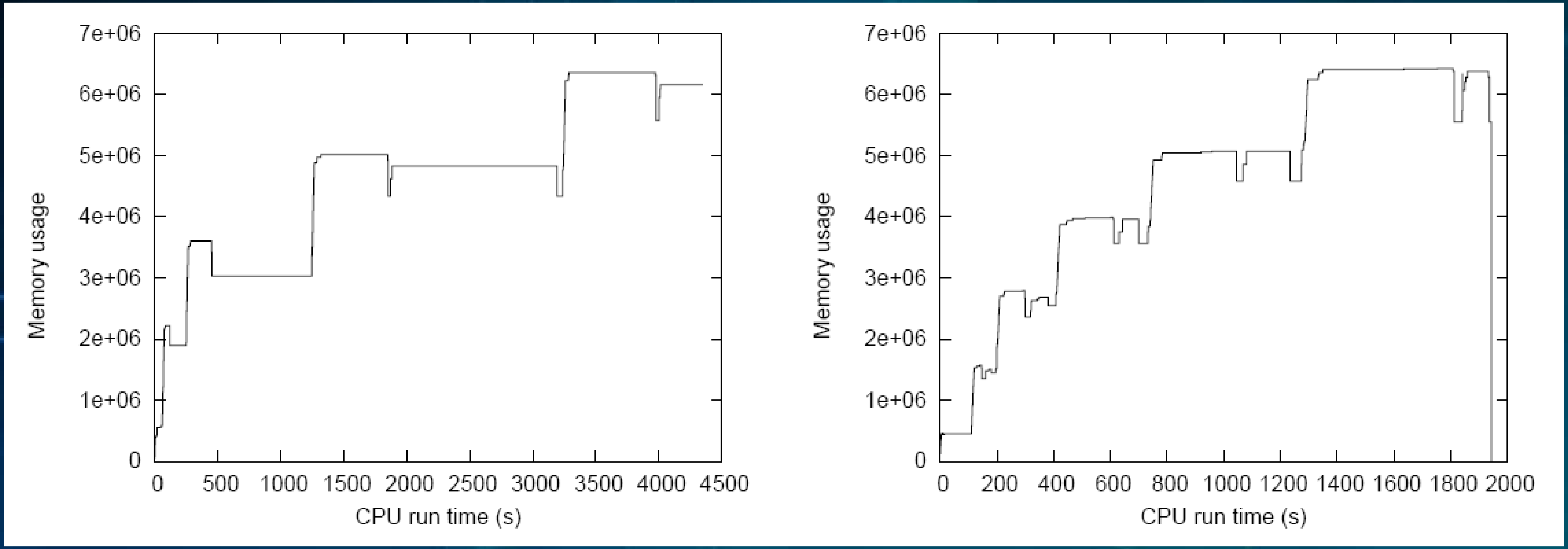
Bounded Model Debugging

circuit name	Run-time (s)	BMD iters	# solutions	Imrpovement (X)
ac97_ctrl-1	204.57	10	7	6.09
fpu1	201.01	4	6	10.49
mem_ctrl-2	28.35	1	11	126.98
rx_comm1	452.97	1	30	7.95

- Without BMD:
  - 3 time-out, 17 memory-out
  - 11 cases where bug is found out of 31
- With BMD:
  - 8 time-out, 0 memory-out
  - 29 cases where bug is found out of 31

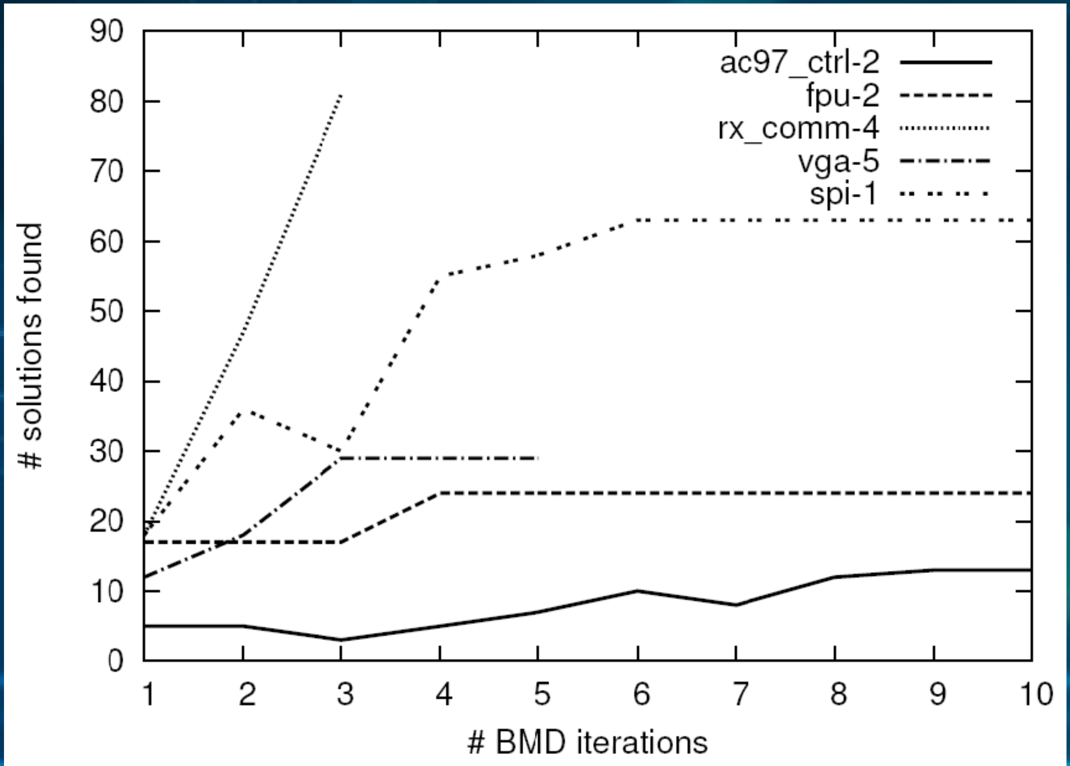
Bounded Model Debugging

- Behavior of BMD iterations on memory and run-time
  - memory usage increases uniformly
  - run-time increases (not uniformly)



Bounded Model Debugging

- Behavior of BMD iterations on # solutions found
  - Number of solutions tends to plateau
  - If plateau point is less than entire trace length, memory and run time savings are expected



Outline

- Motivation
  - The debugging problem
- Background
  - Debugging in practice
  - Automated debugging
- Bounded Model Debugging
- Experiments
- Conclusion

Conclusion

- Industrial debugging problems are challenging
  - Large memory requirements and slow performance
- Need efficient debugging techniques
- We introduce bounded model debugging (BMD) to improve current techniques
- BMD can tackle problems with very long traces
- Performance improves by up to two orders of magnitude
- Can solve 2.7 times more problems