Topic VII

Simulation-Based Verification

系統晶片驗證

SoC Verification

Sep, 2004

2nd International Symposium on

Automated Technology for Verification and Analysis

National Taiwan University (BL 101)

Sunday 31 October - Wednesday 3 November 2004

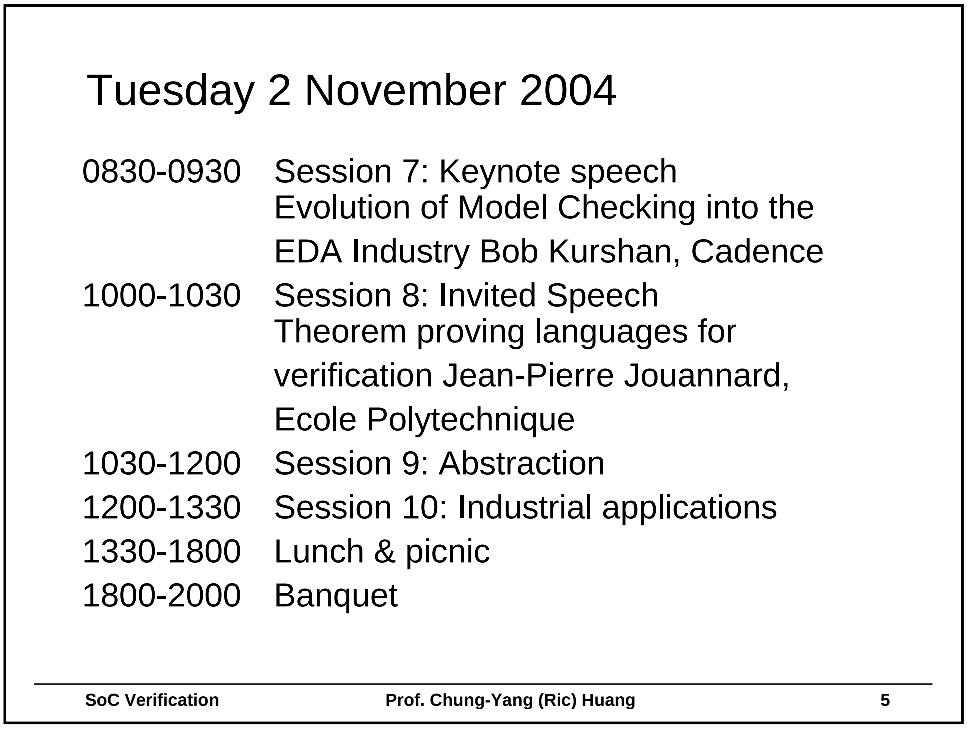
http://cc.ee.ntu.edu.tw/~atva04

Sunday 31 October 2004

0930-1130	Tutorial I Formal Modeling and Analysis of Hybrid Systems Rajeev Alur, Univ. of Penn
1130-1300	Lunch
1300-1500	Tutorial II Assertion-Based Verification - Part A Bob Kurshan, Cadence
1500-1530	Break
1530-1730	Tutorial III Assertion-Based Verification - Part B Pei-Hsin Ho, Synposys
1830-2030	Reception

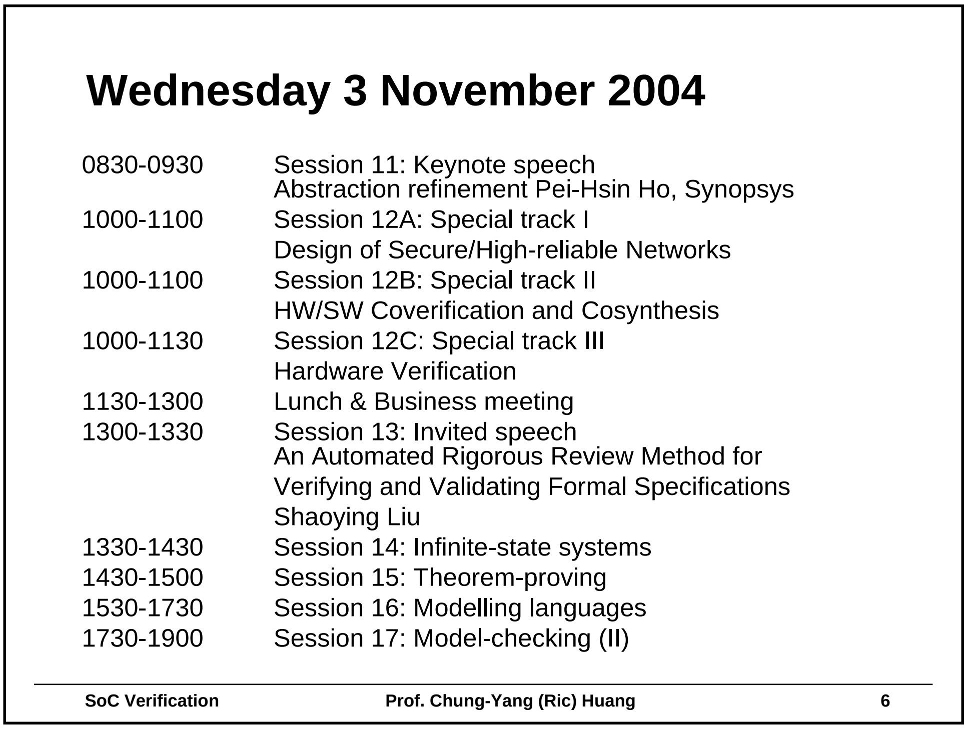
Monday 1 November 2004

0800-0830	Registration and checking-in
0840-0940	Session 1: Keynote speech Games for Formal Design and Verification of Reactive Systems Rajeev Alur, U. of Pennsylvania
1000-1200	Session 2: Model-checking (I)
1330-1400	Session 3: Invited speech Tools for Automated Verification of Web Services Tevfik Bultan, UCSB
1400-1530	Session 4: SAT-based techniques
1600-1730	Session 5: Testing
1730-1850	Session 6: Short papers



Tuesday 2 November 2004

0830-0930	Session 7: Keynote speech Evolution of Model Checking into the EDA Industry Bob Kurshan, Cadence
1000-1030	Session 8: Invited Speech Theorem proving languages for verification Jean-Pierre Jouannard, Ecole Polytechnique
1030-1200	Session 9: Abstraction
1200-1330	Session 10: Industrial applications
1330-1800	Lunch & picnic
1800-2000	Banquet



Wednesday 3 November 2004

0830-0930	Session 11: Keynote speech Abstraction refinement Pei-Hsin Ho, Synopsys
1000-1100	Session 12A: Special track I Design of Secure/High-reliable Networks
1000-1100	Session 12B: Special track II HW/SW Coverification and Cosynthesis
1000-1130	Session 12C: Special track III Hardware Verification
1130-1300	Lunch & Business meeting
1300-1330	Session 13: Invited speech An Automated Rigorous Review Method for Verifying and Validating Formal Specifications Shaoying Liu
1330-1430	Session 14: Infinite-state systems
1430-1500	Session 15: Theorem-proving
1530-1730	Session 16: Modelling languages
1730-1900	Session 17: Model-checking (II)

About Final Projects

- ◆ 32 groups + 1 repeated

1. zChaff SAT solver
   - Comb ATPG(11), Sequential TPG, equivalence checker
2. BDD (CUDD) based
   - Equivalence checker, Floorplanning, channel routing, PCI express, C++ wrapper(2), solver enhancement
3. Combined engine
4. Design verification projects
   - MIPS RISC processor, rapid prototyping, ModelSim, SystemC & SystemVerilog (2)
5. Field study (groups)
   - Open EDA tools, SW verification, formal verification, ABV, analog verification(2)
6. Others
   - IP qualification framework

Initial Project Proposals

- ◆ Due to typhoon and some ambiguity in rules, we will not deduce points for late initial reports this time
  - All students have turned in by Tuesday
  - 1 student overlapped in 2 groups
  - 1 student didn’t turn in hard copy
  - Please refer to TA’s website for details
  - Grades will be posted on class website this weekend
  - I will e-mail my comments to you today

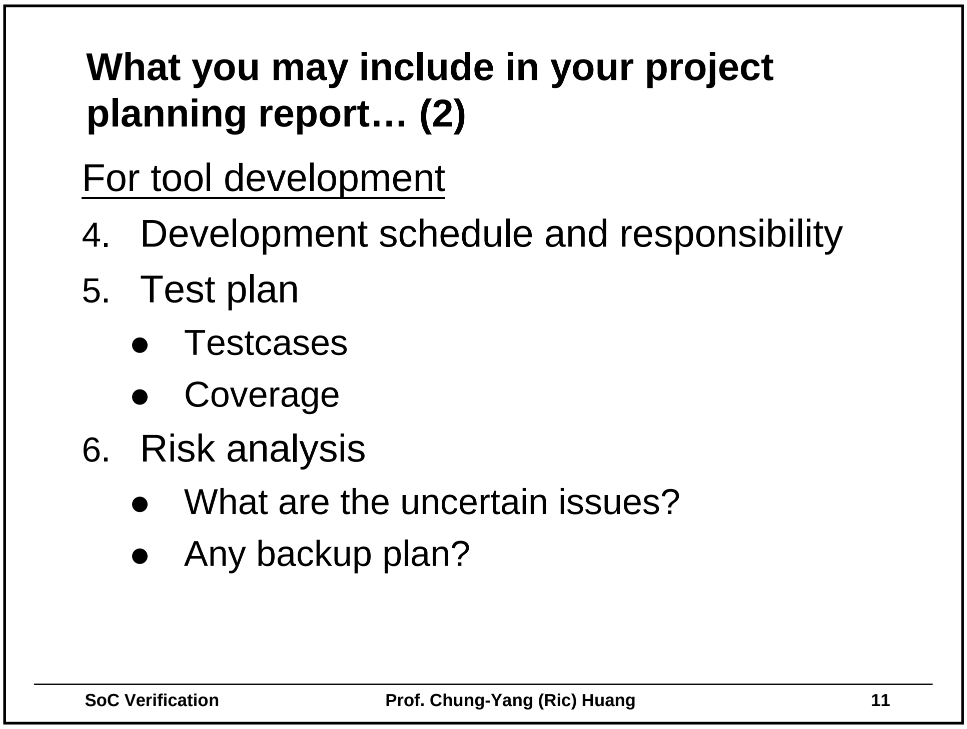
Notes about Handing in your report

- For project reports, please hand in both soft and hard copies (both by due date)
  - Please refer to class website for any last minute changes
  - 1 copy for each group
    - Pick 1 group member's name and id for the e-mail title and attached filename
  - Exact hour on due date will be announced in class and on line
- For homework, please refer to the rules on the class website

What you may include in your project planning report… (1)

For tool development

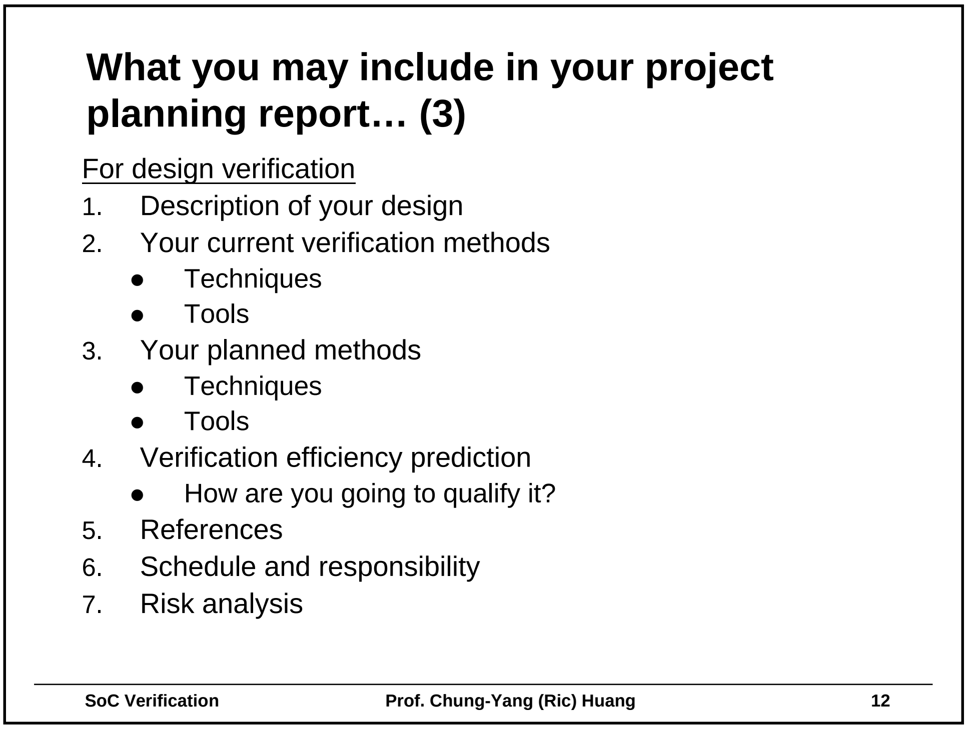
1. User commands and I/O interface
   - Command usage
   - Input language (syntax), parser
   - GUI?
2. Data structures
   - Class architect/hierarchy
   - Interface
3. Major functional components
   - Algorithms (pseudo codes)
   - References
   - Complexity analysis



What you may include in your project planning report… (2)

For tool development

4. Development schedule and responsibility
5. Test plan
   - Testcases
   - Coverage
6. Risk analysis
   - What are the uncertain issues?
   - Any backup plan?



What you may include in your project planning report… (3)

For design verification

1. Description of your design
2. Your current verification methods
   - Techniques
   - Tools
3. Your planned methods
   - Techniques
   - Tools
4. Verification efficiency prediction
   - How are you going to qualify it?
5. References
6. Schedule and responsibility
7. Risk analysis

What you may include in your project planning report… (4)

For field study

1. Description of the field
2. Scope of your study
   - Architecture of this study
   - Sub-topics
3. References
   - Paper
   - Tools/manuals
   - Interviews
4. Schedule and responsibility
5. Risk analysis

HW #1 Behavior Model of Elevator Controller

- Description
  - Use behavior-level language (C/C++ recommended; SystemC, SystemVerilog also OK) to model an elevator controller (e.g. 電機系館的電梯)
- What you need to do
  1. FSM-based specification
  2. Behavior model design (source code)
  3. Testbench
  4. Assertions
  5. [Optional] Coverage analysis
- Problem spec will be put on web by tomorrow
- Due date: 9am, Friday, Nov 12

What we will cover in this topic…

1. The essence of simulation techniques
2. Delay models
3. Types of simulators
4. Steps in simulation-based verification
    1. Preparing input stimuli
    2. Comparing output response
    3. Evaluating the simulation coverages
    4. Debugging the circuit
5. Assertion-based verification
6. SystemVerilog Basics

Remember: Functional vs. Timing vs. Physical Verification

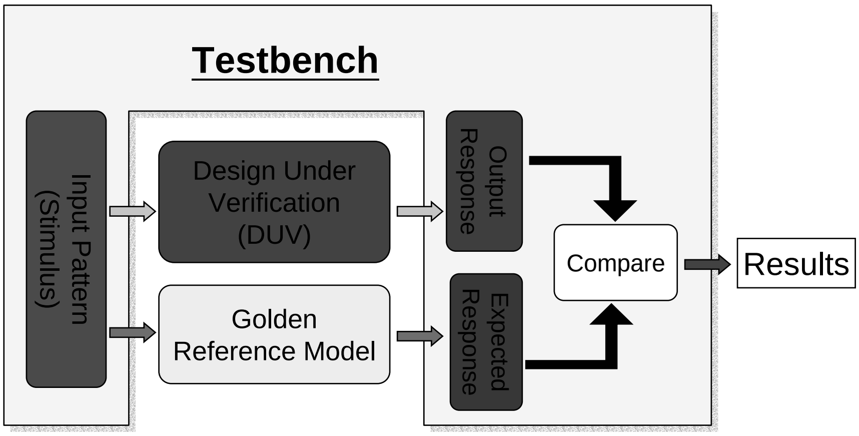
◆ We usually perform functional, timing, and physical separately
- To reduce the complexity
- Their importance differs in different design stages
- Application order
    - Functional → Timing → Physical

◆ Note
- The simulation-based verification in this lecture topic will focus on “functional” only.

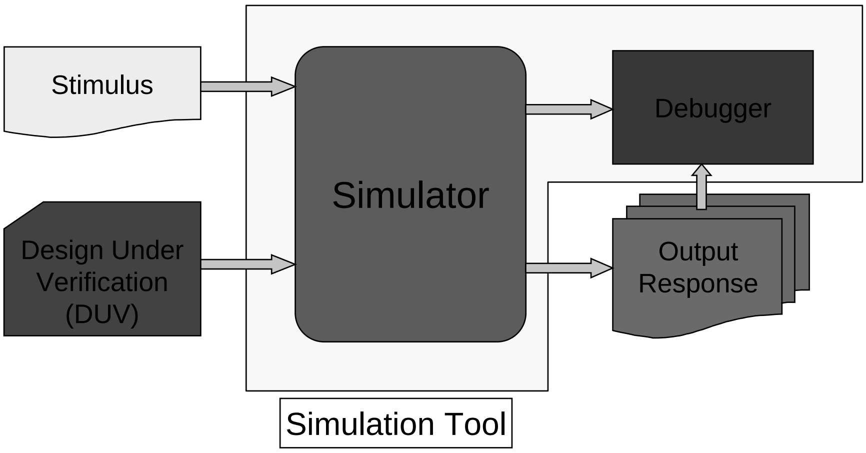
The Fact

- ◆Simulation is by far the most popular method in functional verification
  - ● Intuitive concept
  - ● Easier to learn
  - ● Applicable in various abstract levels
  - ● What you get is what you apply
  - ● Efficient in finding most of the easy bugs
  - ● Quantitative confidence measurement
  - ● Tools are much cheaper

Remember: Simulation-Based Verification



From the Tool’s Viewpoint



Simulator is the core…

What is simulation?

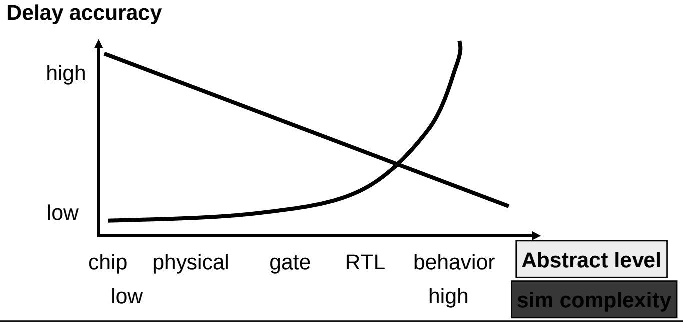
1. To create an environment that mimics the reality
   → What's the reality?
   → The *hardware chip* or the *design?*
2. The environment is *never real*
   → If the reality is the chip
   - It's an *approximation*

   → If the reality is the design
   - Simulation vs. synthesis semantics vs. designer's intention
3. The simulation environment lets the designers interact with the design before it is *transformed* into next level of abstraction

A simulation environment is an approximated model

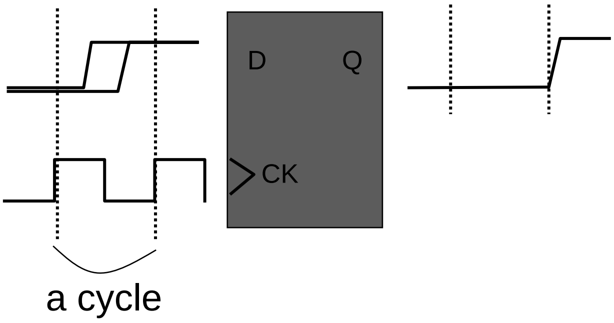
What to consider in the approx. model?

1. Signal delay
   - Accurate delay only exists in chips
     - and it is different from chip to chip
   - Accuracy vs. complexity



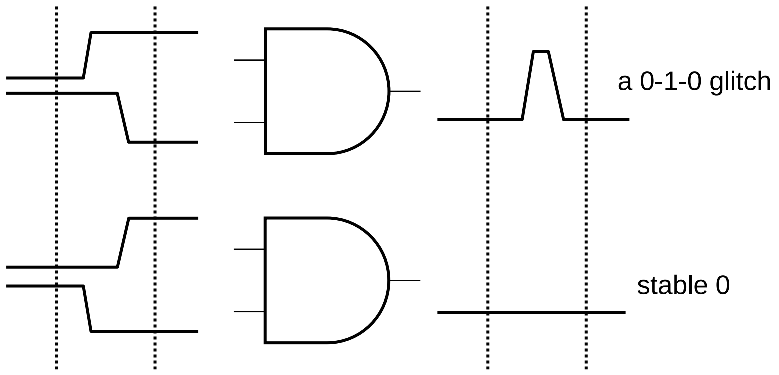
Since we focus on “functional” verification only, we may not care about delays

➔ min requirement: *cycle accuracy*



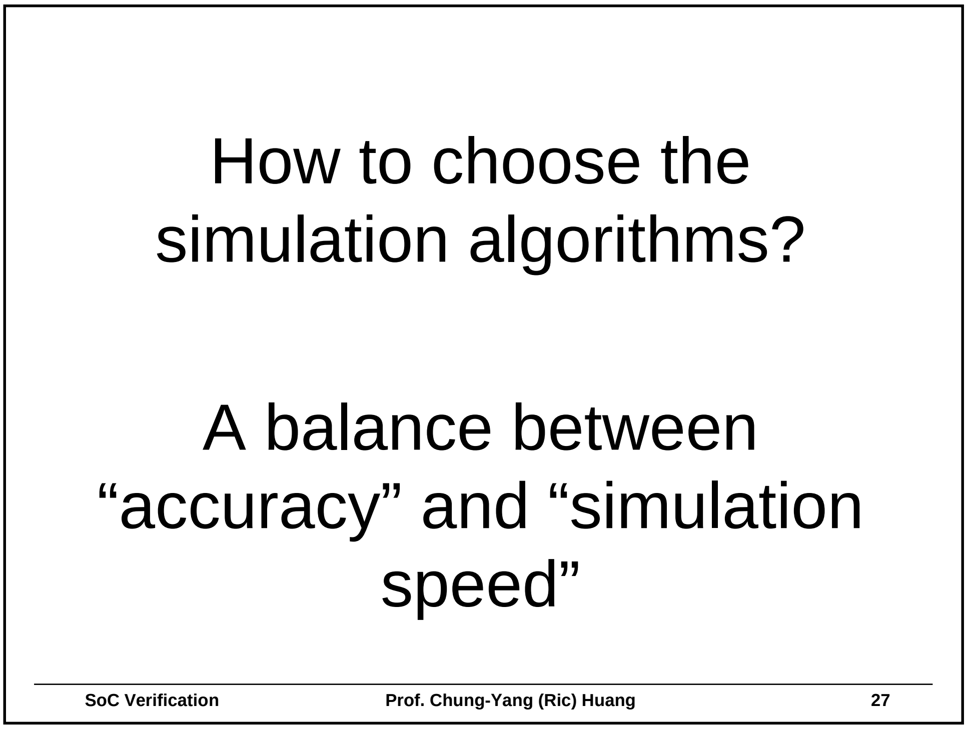
However, we need to pay attention to

1. Glitches (due to race condition)
2. Combinational loops



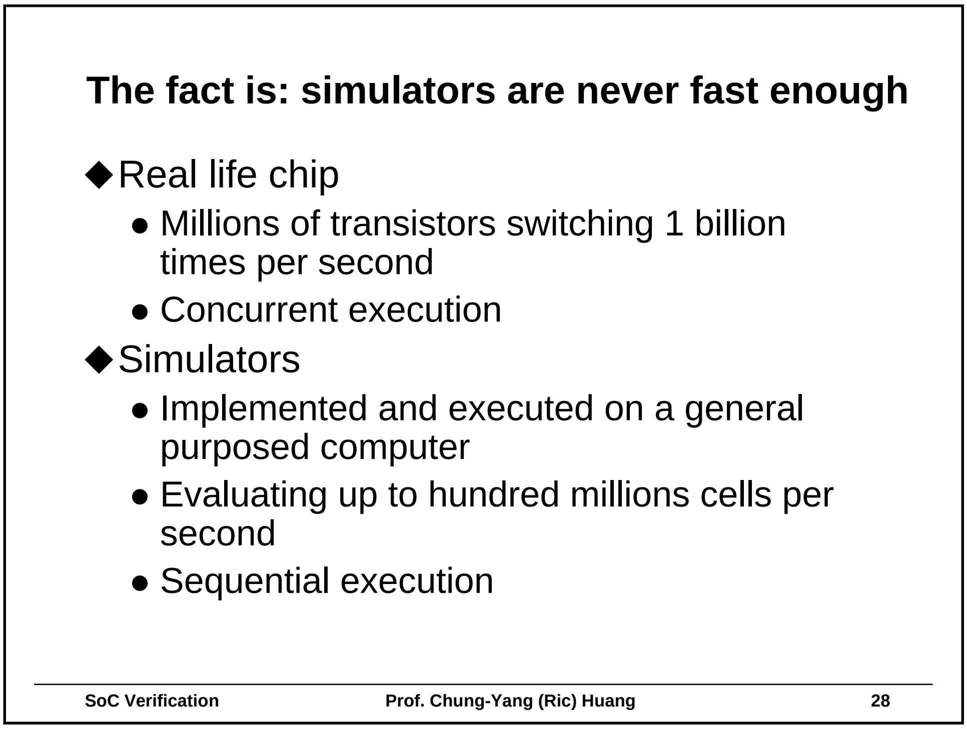
What to consider in the approx. model?

2. Signal strengths
   - 0 (ground), 1 (VDD), strong, weak, pull up/down, high impedance, ‘X’, etc
3. Instance types
   - The primitive evaluators in the simulator
   - Accuracy vs. (implementation) complexity
4. Concurrent vs. sequential
5. Digital vs. Analog



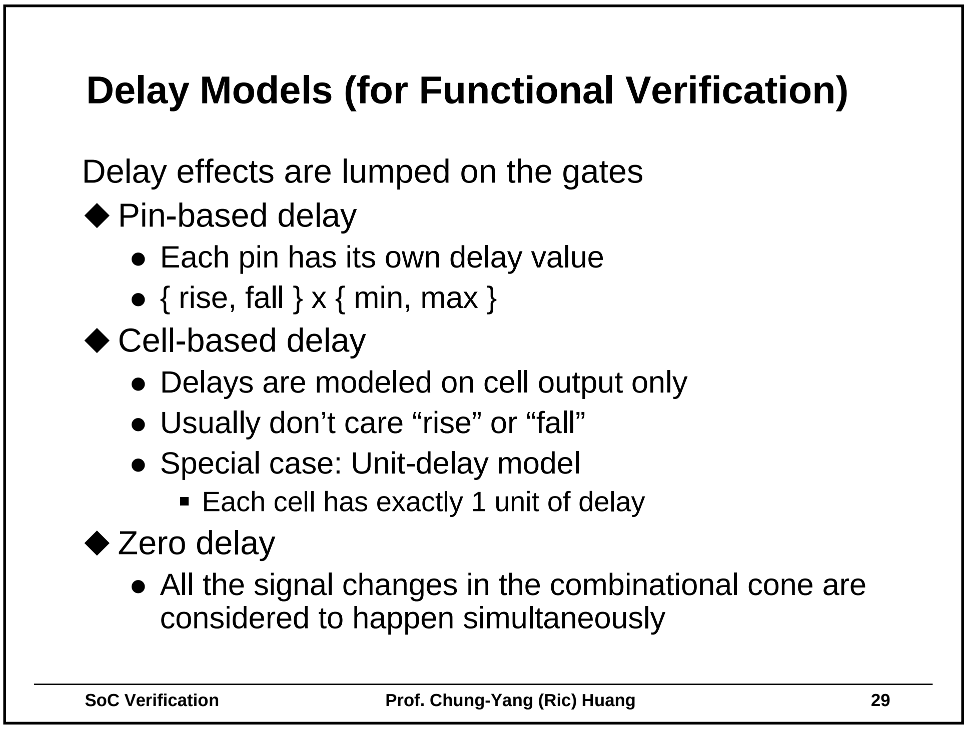
How to choose the simulation algorithms?

A balance between “accuracy” and “simulation speed”



The fact is: simulators are never fast enough

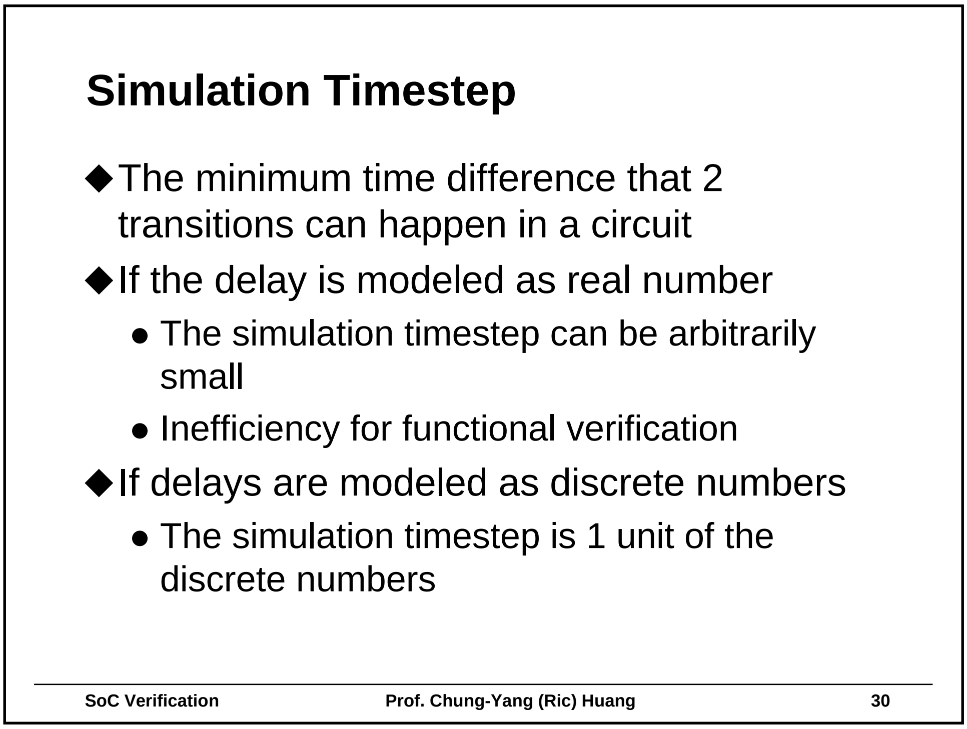
- ◆Real life chip
  - ● Millions of transistors switching 1 billion times per second
  - ● Concurrent execution
- ◆Simulators
  - ● Implemented and executed on a general purposed computer
  - ● Evaluating up to hundred millions cells per second
  - ● Sequential execution



Delay Models (for Functional Verification)

Delay effects are lumped on the gates

- ◆ Pin-based delay
  - Each pin has its own delay value
  - { rise, fall } x { min, max }
- ◆ Cell-based delay
  - Delays are modeled on cell output only
  - Usually don’t care “rise” or “fall”
  - Special case: Unit-delay model
    - Each cell has exactly 1 unit of delay
- ◆ Zero delay
  - All the signal changes in the combinational cone are considered to happen simultaneously



Simulation Timestep

- ◆ The minimum time difference that 2 transitions can happen in a circuit
- ◆ If the delay is modeled as real number
  - The simulation timestep can be arbitrarily small
  - Inefficiency for functional verification
- ◆ If delays are modeled as discrete numbers
  - The simulation timestep is 1 unit of the discrete numbers

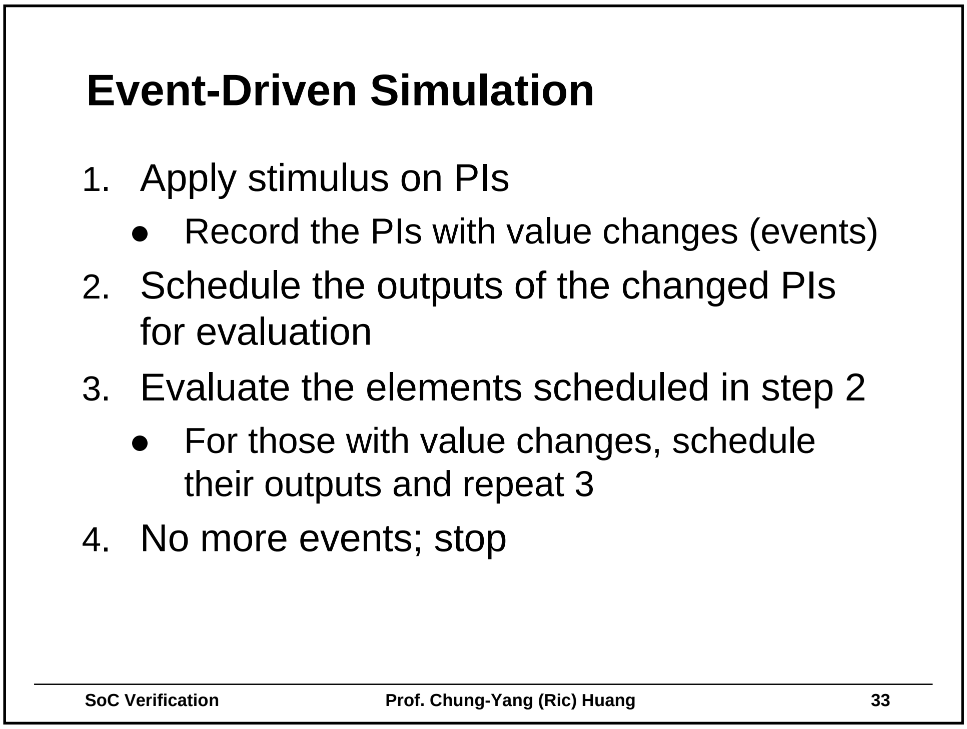
Basic Simulation Algorithm Steps

1. Apply stimulus on PIs
2. For the next timestep, evaluate each cells in the circuit for the possible value changes
3. If there is any cell with value change, go to step 2
4. Otherwise, the circuit is reach a steady state. Either go to 1 for next stimulus, or quit

Obviously, this algorithm is not optimal

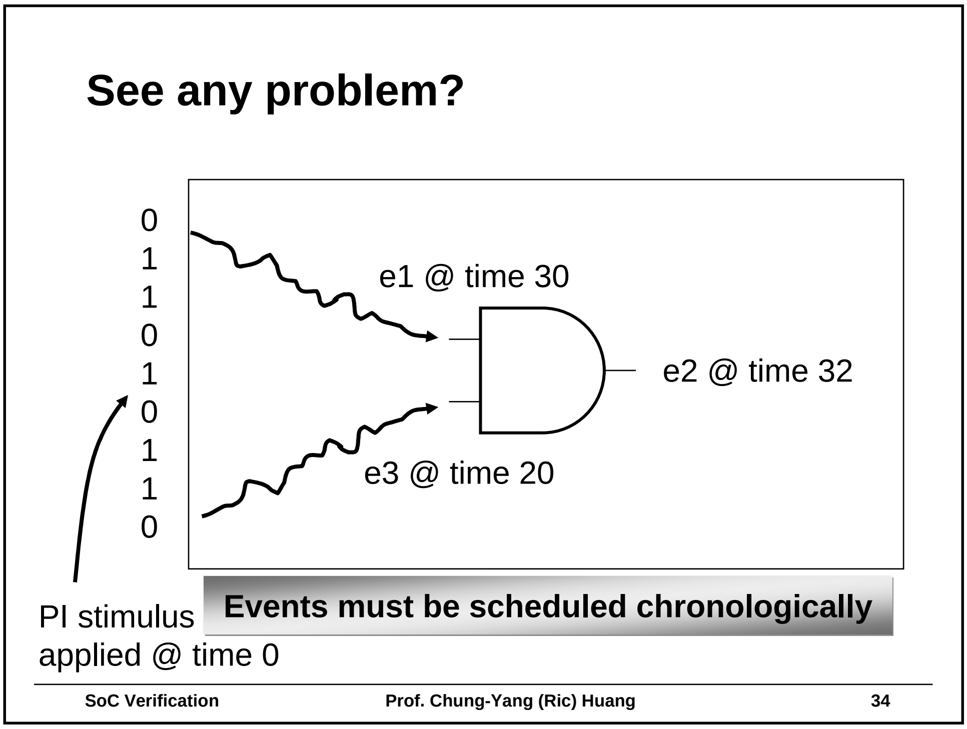
If none of the inputs of a cell has value change,
we don’t need to evaluate this cell
➔ value change = “event”

If a gate has value change
➔ schedule its outputs



Event-Driven Simulation

1. Apply stimulus on PIs
   - Record the PIs with value changes (events)
2. Schedule the outputs of the changed PIs for evaluation
3. Evaluate the elements scheduled in step 2
   - For those with value changes, schedule their outputs and repeat 3
4. No more events; stop

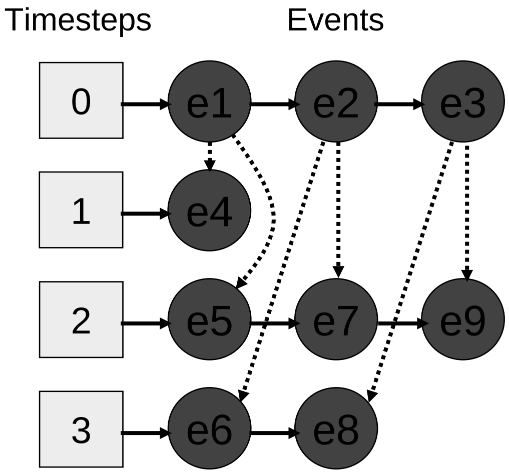


See any problem?

PI stimulus applied @ time 0

Events must be scheduled chronologically

Event Wheels on Simulation Timestep



Event-Driven Simulation

- ◆ Advantages
  - Delay calculation can be as accurate as possible
  - Asynchronous effects (like glitches) can be caught
- ◆ Disadvantages
  - Still not optimal (a cell can be evaluated many times in a cycle)

Optimizations for Event-Driven Simulation

- If we only care about the cycle-accuracy of the simulation
  - Only the steady states of the circuit matter
- If we assume the entire design meets the setup and hold requirements of all FFs
  - Everything must be stable in the cycle
- If we assume the active clock edge is the only significant event in changing the state of the design
  - Everything must be synchronous

Use 0-delay model to optimize the simulation

Cycle-Based Simulation

- Cells are assumed to be 0-delay
- Optimizations
  - Evaluate cells in topological order
    - Each cell is evaluated at most once in a cycle
  - Compile the circuit to its equivalent by removing the redundancy
    - e.g. AND gate with ‘1’ in its inputs

Cycle-Based Simulation

◆ Advantages
- Better performance than event-driven simulation
- Can be applied to larger design

◆ Disadvantages
- Only work for synchronous circuit
- Cannot detect asynchronous effects

Usually needs to work with STA tools

Types of Implementation for Simulators

1. Interpreted code simulators
   - Circuit is modeled as a netlist of primitive elements like AND, OR, etc
   - Each element has its own evaluation function
   - Simulation is performed as an event propagation on the netlist
2. Compiled code simulators
   - Circuit is compiled into an executable with simulation microinstructions
   - Run the test pattern on the executable

➔ Compiled code simulators are usually faster

Parallel Pattern Simulation

◆Pack several test patterns in an integer (e.g. 32 bits)

◆Simulate these test patterns simultaneously

➔ Great speedup

➔ But speedup less if it creates too many events

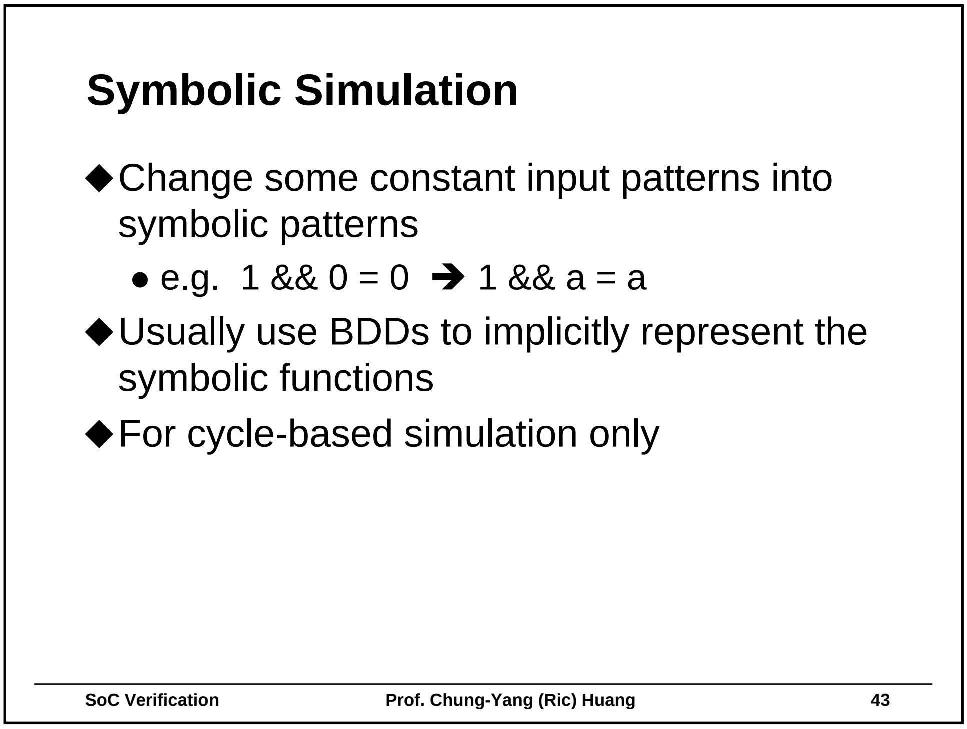
- Each bit change is an event

➔ However, may only work for cycle-based simulation

Parallel Simulation

◆Use multi processes or threads to simulate different parts of the circuit at the same time

- Concurrent execution
- More complicated scheduling scheme



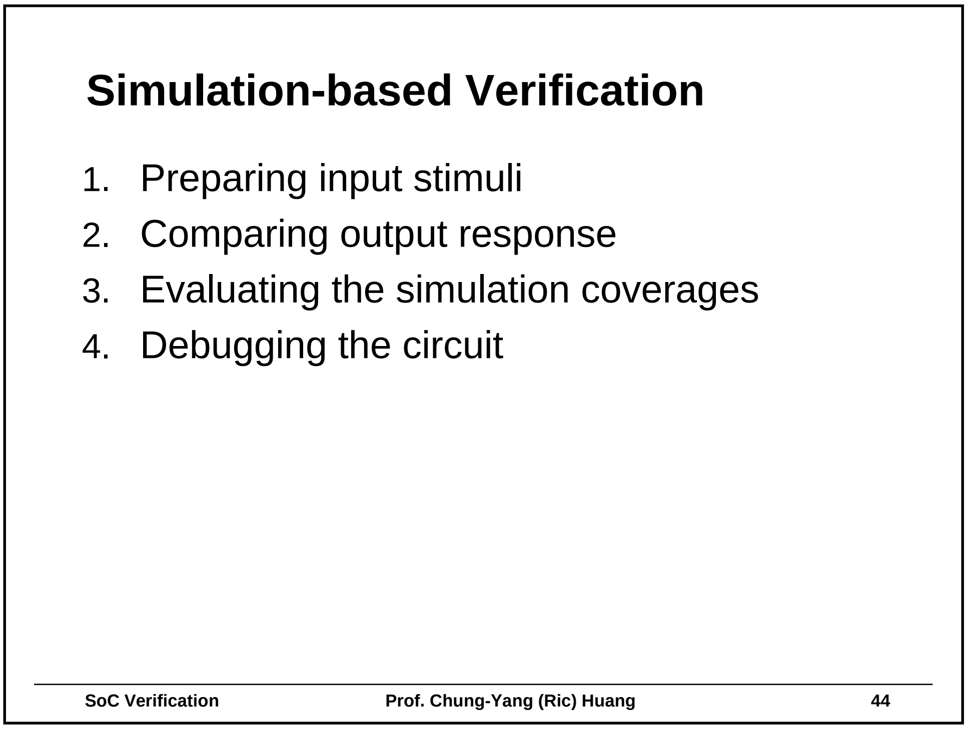
Symbolic Simulation

◆Change some constant input patterns into symbolic patterns

- e.g. 1 && 0 = 0 ➔ 1 && a = a

◆Usually use BDDs to implicitly represent the symbolic functions

◆For cycle-based simulation only



Simulation-based Verification

1. Preparing input stimuli
2. Comparing output response
3. Evaluating the simulation coverages
4. Debugging the circuit

Simple Stimulus

- Explicit values directly applied on PIs
- Test pattern usually explicitly written in HDL
  - Manual pattern

```
initial begin // initilaise the input and output buffers
    for(tmpCounter = 0; tmpCounter <= 2048;
         tmpCounter = tmpCounter + 1) begin
        XmitBuffer[tmpCounter] = 8'b00000000;
    end
    GenCrc16Err    = FALSE;
    Crc16ErrMask   = 16'hffff;
end
```

  - May use with “random” system call

Complex Stimulus

- Input pattern may depend on the response of the output or other part of the circuit
  - e.g. Req needs to wait for Ack
- Need API from simulator and to write a small HDL around the DUV

➔ Testbench authoring (e.g. e, Vera, SystemVerilog)

Monitoring Simple Output Response

- ◆For manual input pattern
  - ● Expected output response can be derived
  - ● Automatic monitoring is very important
  - ➔ Regression test
- ◆For random input pattern
  - ● Need a golden reference model to generate the corresponding outputs
  - ● Important to catch unexpected bugs

Complex Output Response

- ◆Output response may not only be a pattern
  - ➔ can be temporal formula
  - ● e.g. Handshaking
- ◆Need to have a monitoring circuit at the output
  - ➔ Assertion-based verification
  - ➔ Can be extended to monitoring internal signals

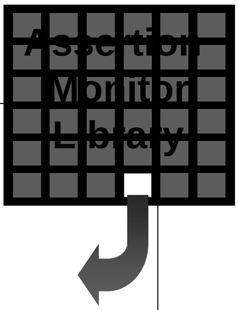
Open Verification Library (OVL)

- ◆Offer HDL library as the monitor circuit
- ◆Open source; free to download
  - ● http://www.verificationlib.org/
- ◆31 types of assertions

OVL Example --- assert_never

[redacted]

[redacted]



```
module assert_never (clk, reset_
  input clk, reset_n, test_expr;
  parameter severity_level = 0;
  parameter msg="ASSERT NEVER VIOLATION";
// ASSERT: PRAGMA HERE
  //synopsys translate_off
  `ifdef ASSERT_ON
    integer error_count;
    initial error_count = 0;
    always @(posedge clk) begin
      `ifdef ASSERT_GLOBAL_RESET
        if (`ASSERT_GLOBAL_RESET != 1'b0) begin
      `else
        if (reset_n != 0) begin // active low reset_n
      `endif
          if (test_expr == 1'b1) begin
            error_count = error_count + 1;
            `ifdef ASSERT_MAX_REPORT_ERROR
              if (error_count<=`ASSERT_MAX_REPORT_ERROR)
            `endif
                $display("%s : severity %0d : time %0t : %m", msg, severity_level, $time);
            if (severity_level == 0) $finish;
          end
        end
    end
  `endif
 //synopsys translate_on
endmodule
```

source: Harry Foster

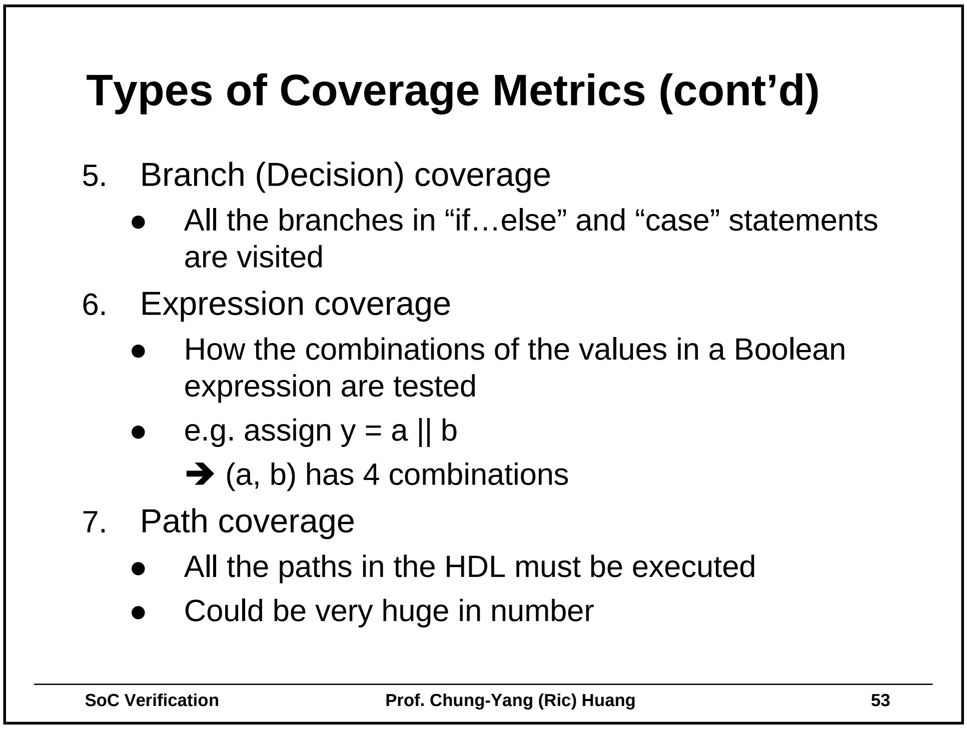
Coverage Metrics

DUT: usually RTL or up

1. A quantitative measure to assess the quality of a test suite
   - [Assume] Test completeness or verification confidence is proportional to the simulation coverage on certain attribute of the design
2. Can also be a hint to generate more vectors on the area where the test is not sufficient

Types of Coverage Metrics

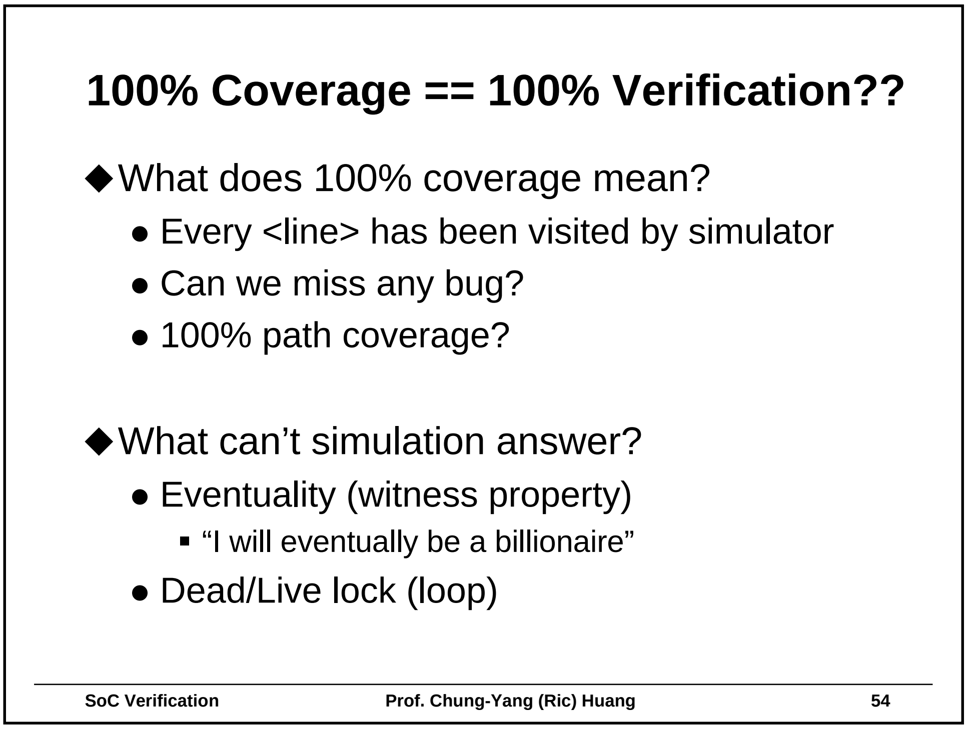
1. Statement coverage
   - Each line of the HDL code must be covered
2. Toggle Coverage
   - Signal in a design is toggled
   - May be used for gate-level design
3. State Machine Coverage
   - All the legal states in FSM are visited
   - All the legal transitions in FSM are traversed
4. Triggering (Event) Coverage
   - All the signals in the sensitivity list are triggered



Types of Coverage Metrics (cont'd)

5. Branch (Decision) coverage
   - All the branches in "if…else" and "case" statements are visited
6. Expression coverage
   - How the combinations of the values in a Boolean expression are tested
   - e.g. assign y = a || b

     ➔ (a, b) has 4 combinations
7. Path coverage
   - All the paths in the HDL must be executed
   - Could be very huge in number



100% Coverage == 100% Verification??

◆What does 100% coverage mean?

- Every <line> has been visited by simulator
- Can we miss any bug?
- 100% path coverage?

◆What can't simulation answer?

- Eventuality (witness property)
  - "I will eventually be a billionaire"
- Dead/Live lock (loop)

Any better coverage metrics?

◆Key point coverage

- Select a set of “important” signals in a circuit, make sure all the value combinations have been tested
- e.g. 30 signals ➔ 1G combinations
- May not be feasible in reality
- How to pick the importance signals?

◆Assertion coverage

- How many time the assertions in the circuit are tested?

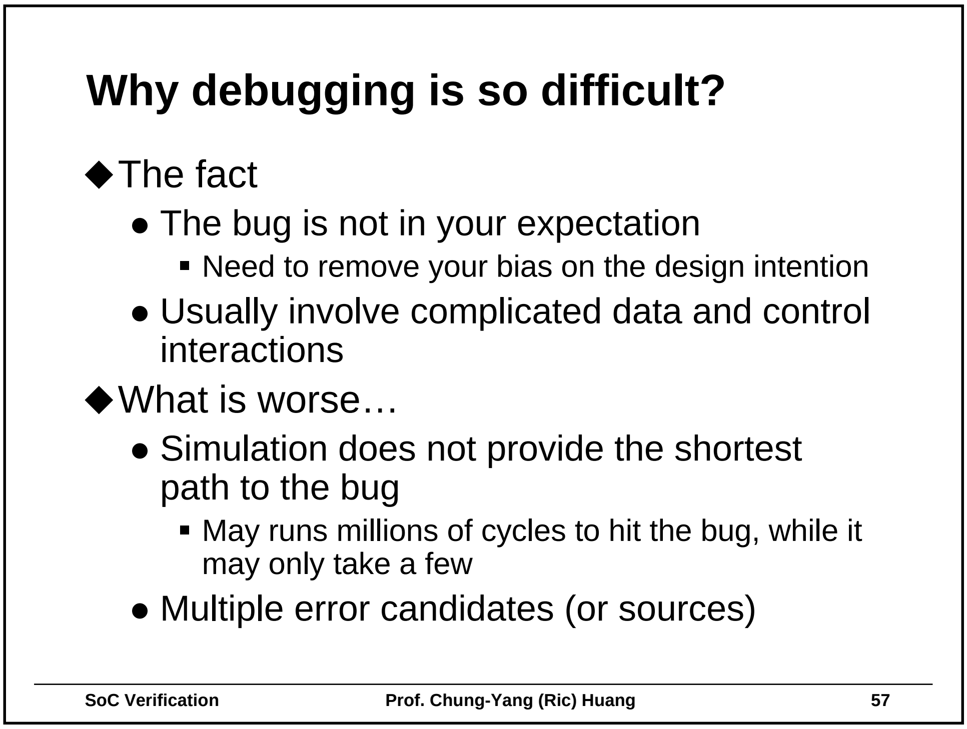
Debugging the Circuit

◆When the output response shows a mismatch, there may be a bug

- A bug in the circuit or the reference model?

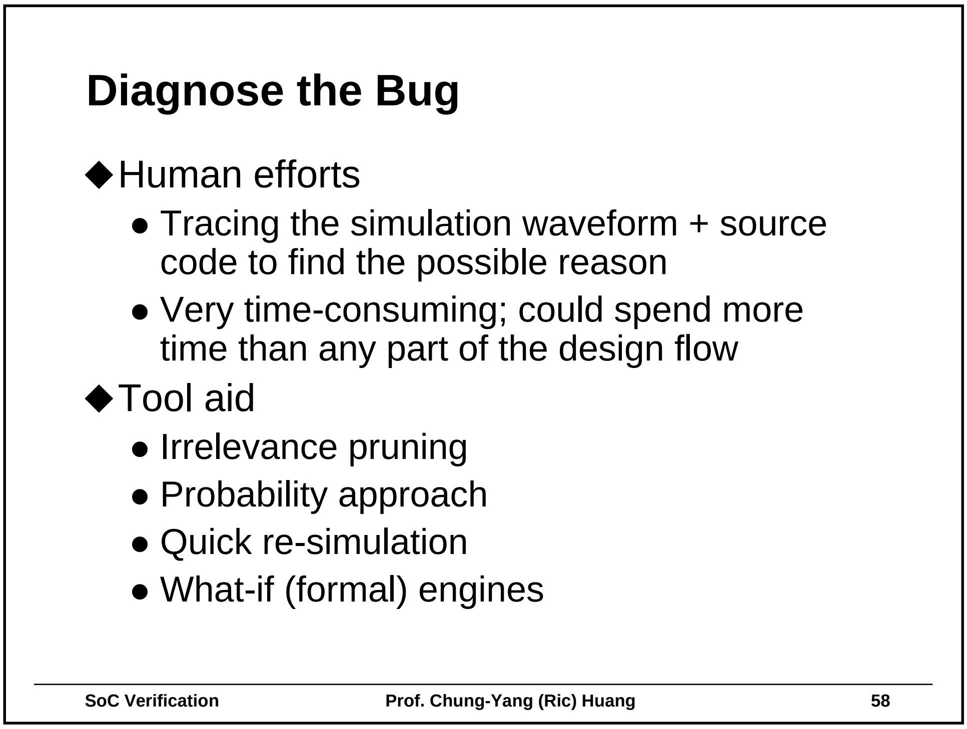
◆Debugging utilities

- Waveform viewer
- Schematic viewer
- Source-code/value annotation
- FSM bubble diagram or STG
- Step debugger



Why debugging is so difficult?

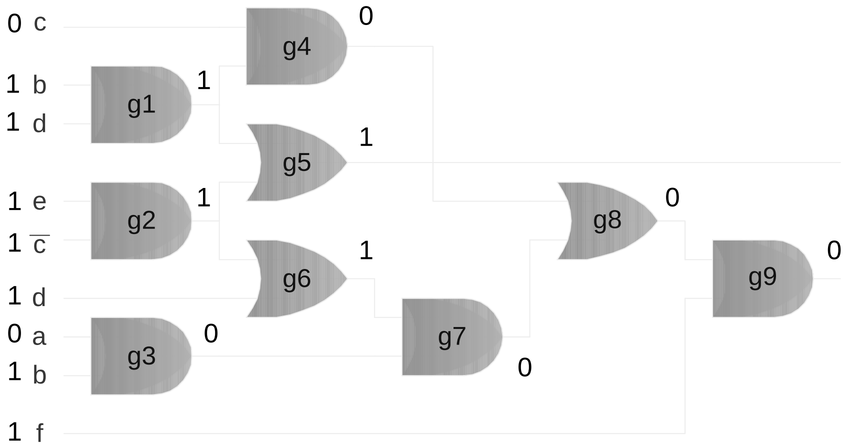
- ◆The fact
  - ● The bug is not in your expectation
    - ▪ Need to remove your bias on the design intention
  - ● Usually involve complicated data and control interactions
- ◆What is worse…
  - ● Simulation does not provide the shortest path to the bug
    - ▪ May runs millions of cycles to hit the bug, while it may only take a few
  - ● Multiple error candidates (or sources)



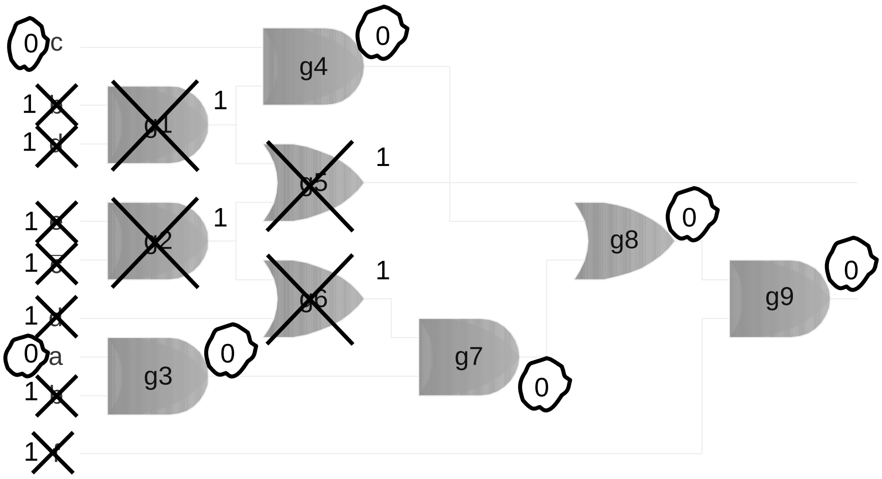
Diagnose the Bug

- ◆Human efforts
  - ● Tracing the simulation waveform + source code to find the possible reason
  - ● Very time-consuming; could spend more time than any part of the design flow
- ◆Tool aid
  - ● Irrelevance pruning
  - ● Probability approach
  - ● Quick re-simulation
  - ● What-if (formal) engines

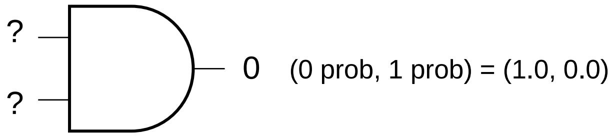
Irrelevance Pruning



Irrelevance Pruning

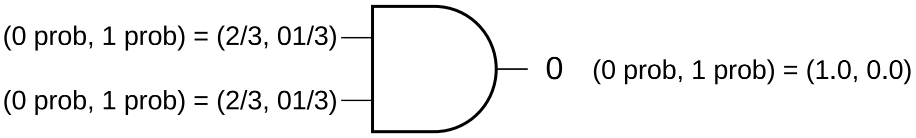


Probability Approach

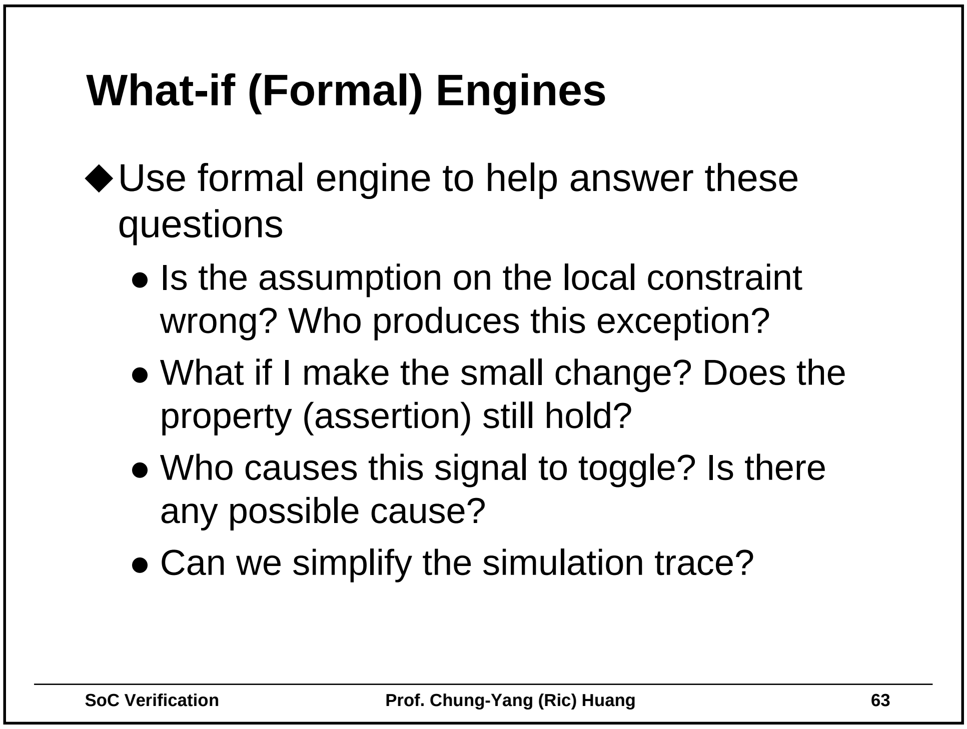


- For AND output to be 0,
  input can be (0, 0), (0, 1), (1, 0)
  - ➔ 2/3 to be ‘0’, 1/3 to be ‘1’

Probability Approach

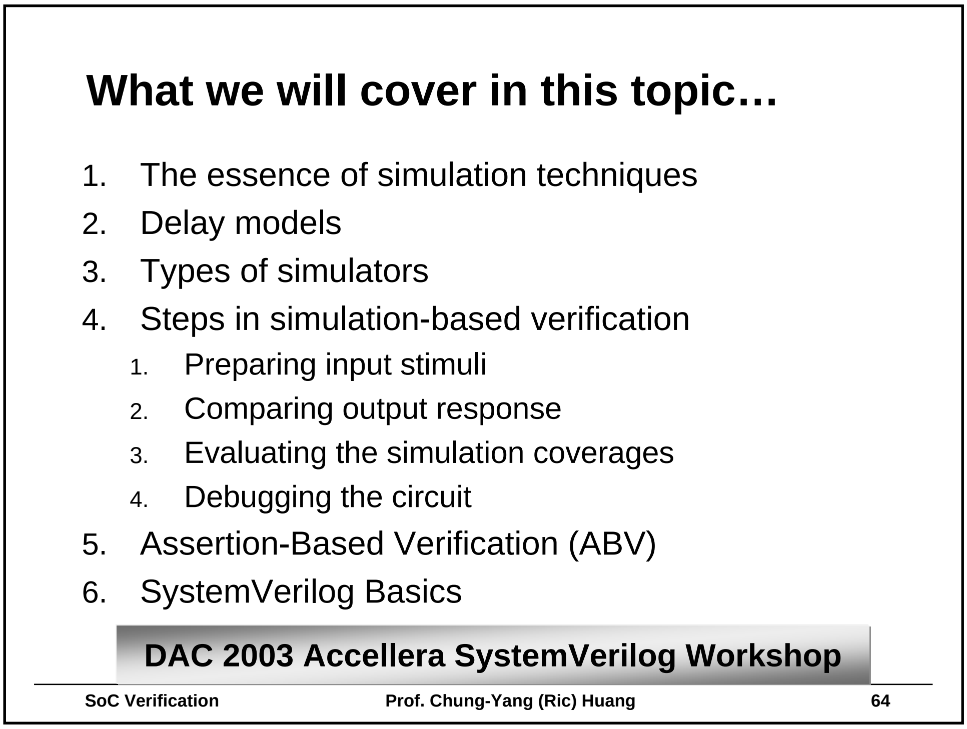

(0 prob, 1 prob) = (2/3, 01/3)
(0 prob, 1 prob) = (2/3, 01/3)
0
(0 prob, 1 prob) = (1.0, 0.0)

- For 2-input AND, if output is (p0, p1)
  - ➔ Input is (2/3 * P0, p1 + 1/3 * p0)
- Can compute the probabilities until inputs
  - ➔ Generate another test vector to witness the bug
  - ➔ Hopefully shorter
- Problem: reconvergence will cause the probability to be inaccurate



What-if (Formal) Engines

- Use formal engine to help answer these questions
  - Is the assumption on the local constraint wrong? Who produces this exception?
  - What if I make the small change? Does the property (assertion) still hold?
  - Who causes this signal to toggle? Is there any possible cause?
  - Can we simplify the simulation trace?



What we will cover in this topic…

1. The essence of simulation techniques
2. Delay models
3. Types of simulators
4. Steps in simulation-based verification
   1. Preparing input stimuli
   2. Comparing output response
   3. Evaluating the simulation coverages
   4. Debugging the circuit
5. Assertion-Based Verification (ABV)
6. SystemVerilog Basics

DAC 2003 Accellera SystemVerilog Workshop