# 17. Design Verification

Jacob Abraham

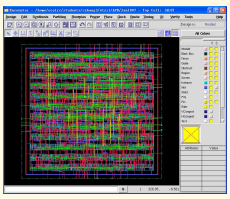
Department of Electrical and Computer Engineering
The University of Texas at Austin

VLSI Design
Fall 2020

October 27, 2020



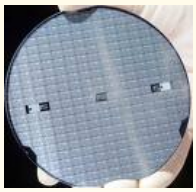
## Reliability in the Life of an Integrated Circuit – I



Design

Design "bugs"
Verification (Simulation, Formal)

Fabrication



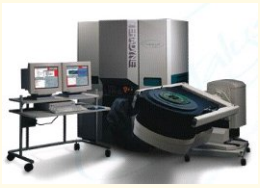
Wafer

Process variations, defects
Process Monitors

## Reliability in the Life of an Integrated Circuit – II

Wafer Probe

Package



Tester

Test cost, coverage
Design for Test, Built-In Self Test

System

Application

Test escapes, wearout, environment
System Self-Test, Error Detection, Fault Tolerance, Resilience



## Analyzing Complex Designs

Need to (implicitly) search a very large state space

- Find bugs in a design – verification process
- Generate tests for faults in a manufactured chip

Basic algorithms for analyzing even combinational blocks (SAT, ATPG) are NP-complete

Approaches to deal with real designs

- Exploit hierarchy in the design
- Develop abstractions for parts of a design

Cost of a new mask set can be on the order of $1+ Million for a large chip

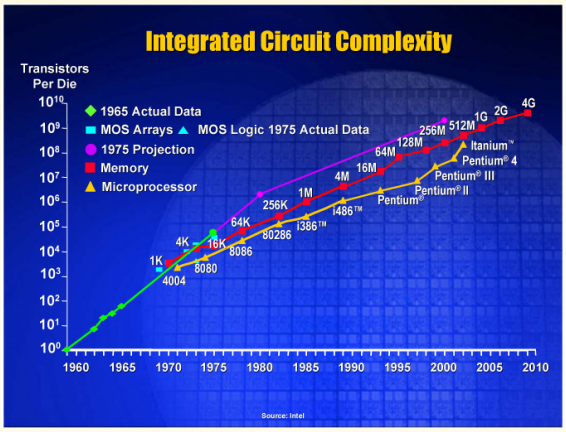
- Cannot afford mistakes
- **Want working "first silicon"**

## Many Aspects of Verification

- Verifying the **functional correctness** of the design
- **Performance** verification
  - Architecture level (number of clocks to perform a function)
- **Timing verification**
  - Circuit level (how fast can we clock?)
- Verifying **power consumption**
- Verifying **signal integrity and variation tolerance**
- Checking **correct implementation** of specifications at each level



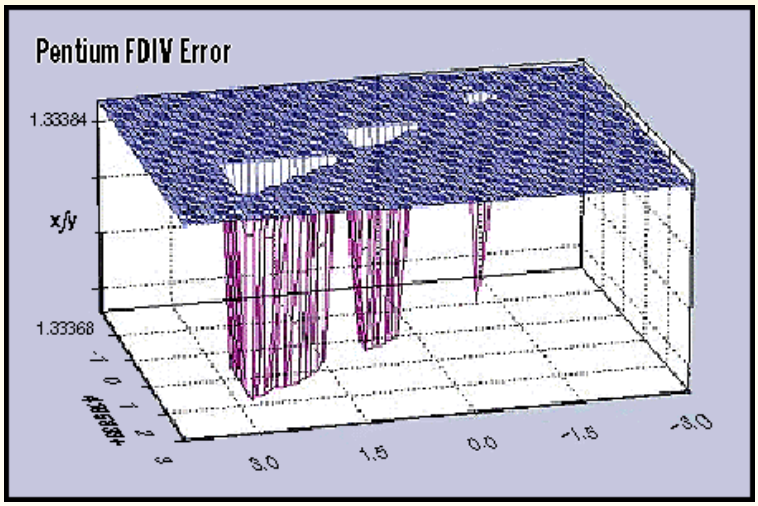
## The Verification Problem




- Need to deal with this complexity
- A subtle bug could produce an incorrect result in a specific state for a specific data input
  - Seen as a "sequence dependency" when simulating a design (specific sequence of inputs to reach the erroneous state)

## The (In)Famous Pentium FDIV Problem

Graph of x, y, x/y in a small region by Larry Hoyle

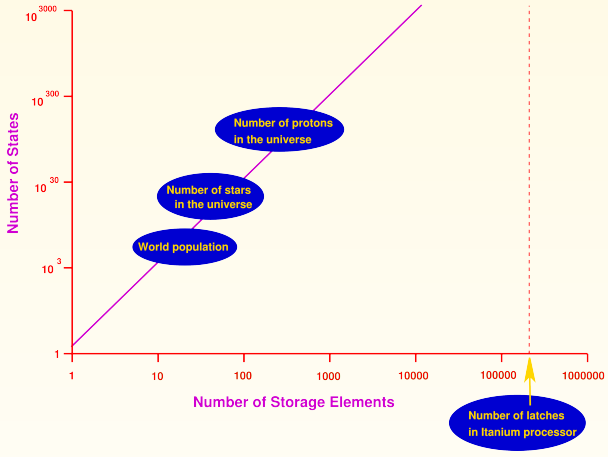





## State-Space Explosion

May need to check a very large number of states to uncover a bug

Problem: the number of protons in the universe is around $10^{80}$, which is less than the number of states for a system with 300 storage elements!



## What is a "Bug"?

**Design does not match the specification**

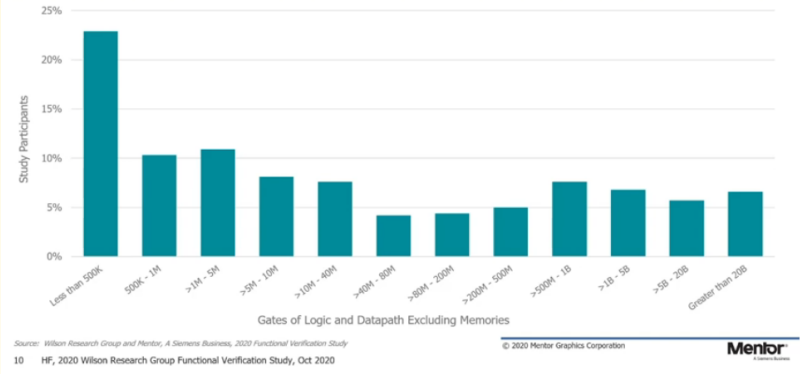
- One problem: complete (and consistent) specifications may not exist for many products
- For example, the difficulty in designing an X86 compatible chip is not in implementing the X-86 instruction set architecture, but in matching the behavior with Intel chips

**Something which the customer will complain about**

- Marketing: **"It's not a bug, it's a feature"**

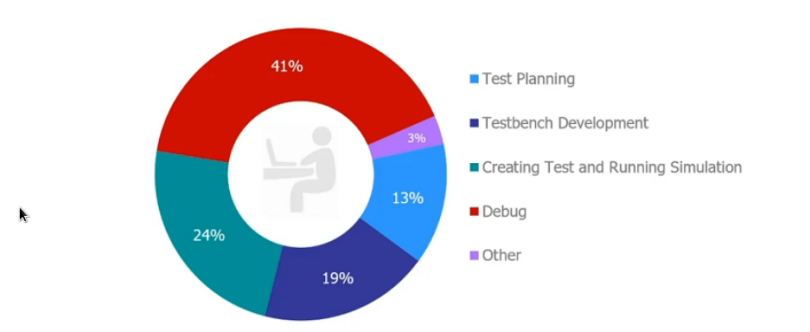


## Mentor/Wilson Group Functional Verification Study




Source: Wilson Research Group and Mentor, 2020 Functional Verification Study – https://go.mentor.com/5ffxz

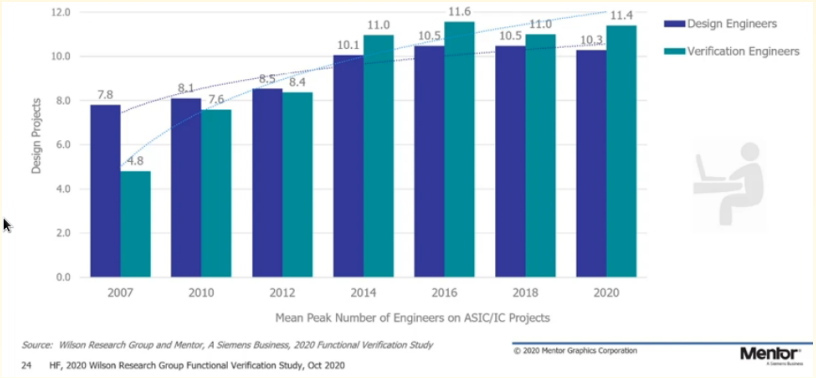
## Percentage of Project Time Spent in Verification – ASICs




Source: Wilson Research Group and Mentor, 2020 Functional Verification Study – https://go.mentor.com/5ffxz



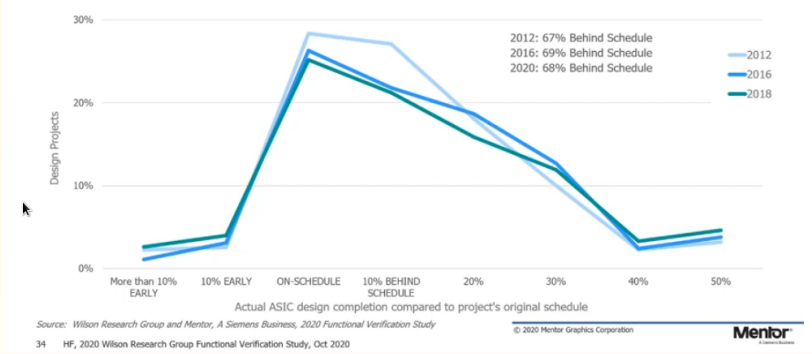
## Mean Peak Number of Engineers on ASIC Projects




**More verification engineers than designers! This is an average: for very large chips, there could be 3–5 times more verification engineers.**

Source: Wilson Research Group and Mentor, 2020 Functional Verification Study – https://go.mentor.com/5ffxz

## Design Completion Compared to Original Schedule in ASICs

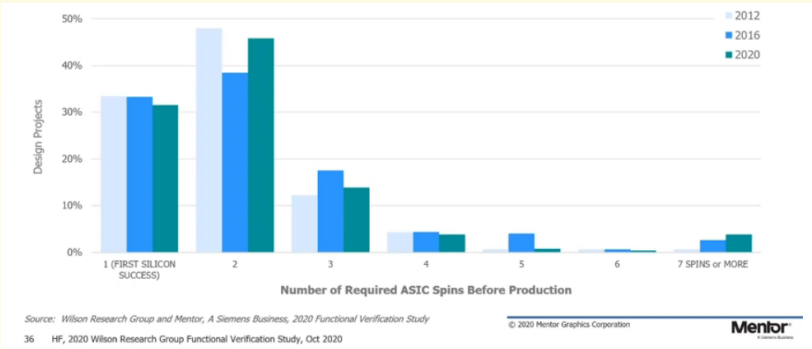



**Delays in verification are a major contributor to delaying schedules**

Source: Wilson Research Group and Mentor, 2020 Functional Verification Study – https://go.mentor.com/5ffxz



## Number of Required ASIC Spins Before Production




Source: Wilson Research Group and Mentor, 2020 Functional Verification Study – https://go.mentor.com/5ffxz

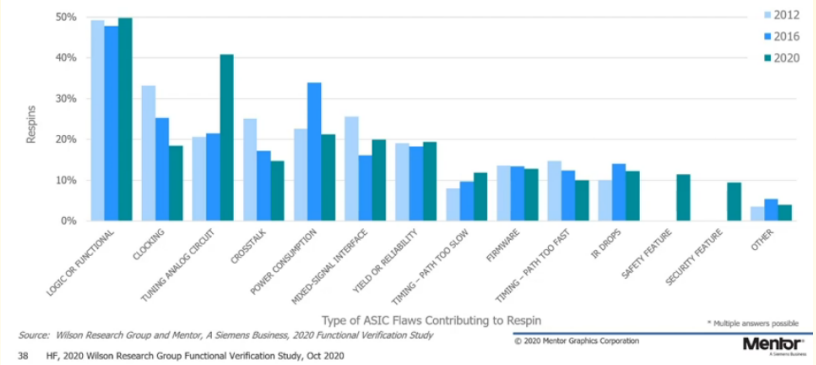
## Design Bug Distribution in Pentium 4

| Type of Bug | % |
|---|---|
| "Goof" | 12.7 |
| Miscommunication | 11.4 |
| Microarchitecture | 9.3 |
| Logic/Microcode Changes | 9.3 |
| Corner Cases | 8.0 |
| Power Down | 5.7 |
| Documentation | 4.4 |
| Complexity | 3.9 |
| Initialization | 3.4 |
| Incorrect RTL Assertions | 2.8 |
| Design Mistake | 2.6 |

- Source: EE Times, July 4, 2001
- 42 Million Transistors
- High-level description: 1+ million lines of RTL
- 100 high-level bugs found through formal verification

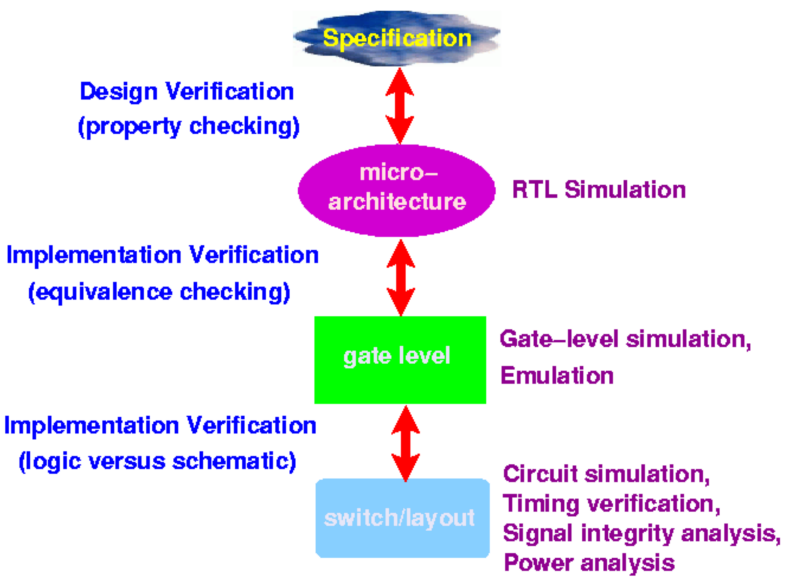


## Flaws Contributing to Respins in ASICs




Source: Wilson Research Group and Mentor, 2018 Functional Verification Study – https://go.mentor.com/5ffxz

## Design and Implementation Verification






## Verification Approaches

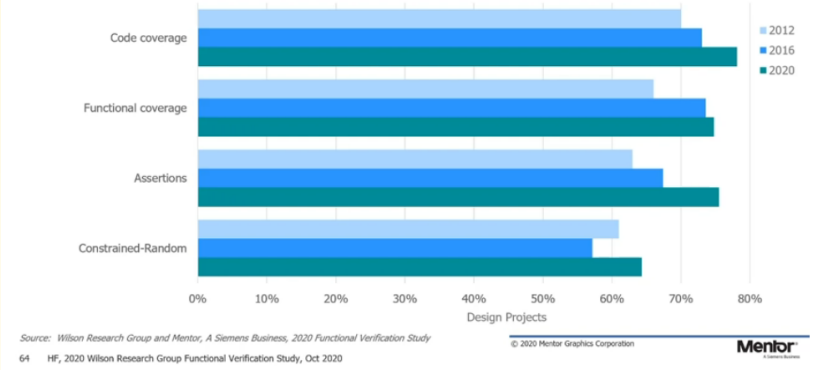
- Simulation (the most popular verification method)
  - Cycle based, functional simulation for billions of cycles
  - Good coverage metrics usually not available
  - Computationally very expensive, slightest optimization has huge impact
- Emulation
  - Capital intensive
  - Map design to be verified on FPGAs
  - Run OS and application at MHz rates
- Formal verification
  - Exhaustive verification of small modules

## Evaluating the Complete Design

- Is there a verification technique which can be applied to the entire chip?
- Only one approach which scales with the design: **Simulation**
- Most common technique now used in industry
- **Cycle-based simulation** can exercise the design for millions of cycles
  - Unfortunately, the question of when to stop simulation is open
  - No good measures of **coverage**
- **Emulation**
  - Used to verify the first Pentium (windows booted on FPGA system)
  - Developing another accurate model is an issue
  - Currently used for post-silicon validation of Intel Atom platform



## Metrics Used to Evaluate Quality of Simulation




Source: Wilson Research Group and Mentor, 2020 Functional Verification Study – https://go.mentor.com/5ffxz

## Simulation Speeds

**Comparing speeds of simulating a microprocessor on a computer**

- Performance timers: 10K – 50K cycles/sec.
- Behavior level: 1000 – 10K cycles/sec.
- R-T level: 20 – 1000 cycles/sec.
- Gate level: 4 – 25 cycles/sec.
- Switch level: 1/4 – 1 cycles/sec.

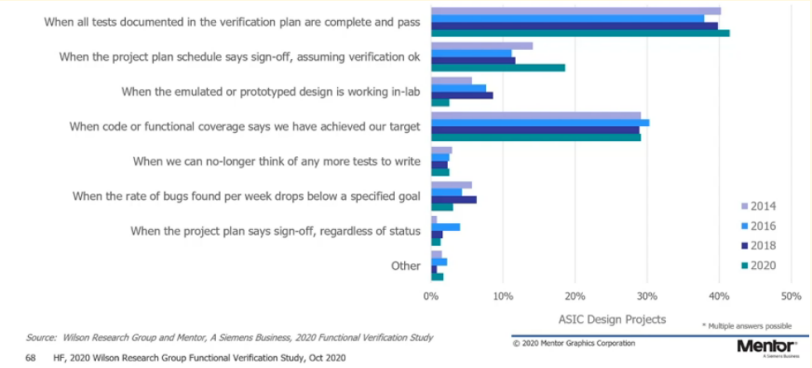


## When are we Done Simulating?

**When do you tape out?**

- Motorola criteria (EE Times, July 4, 2001)
- 40 billion random cycles without finding a bug
- Directed tests in verification plan are completed
- Source code and/or functional coverage goals are met
- Diminishing bug rate is observed
- A certain date on the calendar is reached

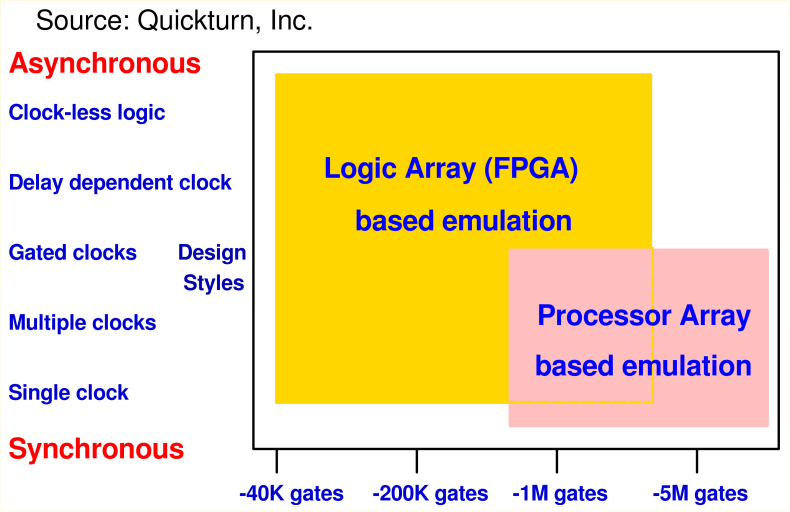
## Signoff Criteria




Source: Wilson Research Group and Mentor, 2020 Functional Verification Study, https://go.mentor.com/5ffxz



## Emulation Technologies



## Architecture Verification

- Verify that the design matches the architectural specification
- Extensive testing the common approach
  - Conformance testing
- Approaches used in industry
  - Manually writing tests
  - Generating pseudo-random instruction sequences
  - Using biased pseudo-random instructions
  - Generating instruction sequences from typical workloads
  - Example: to verify an X86 clone, capture instruction trace on another X86 machine is running application



## Example of Verifying Processors

### Verifying PowerPC Processors

- Model based test generator
  - Expert system which contains a formal model of processor architecture, and a heuristic data base of testing knowledge
- Example, testing knowledge accumulated during the verification of the first PowerPC design included about 1,000 generation and validation functions (120,000 lines of C code)
- PowerPC behavioral simulator has about 40,000 lines of C++ code

## MAP1000 Media Processor Verification (Shen, 1999)

### System Architecture

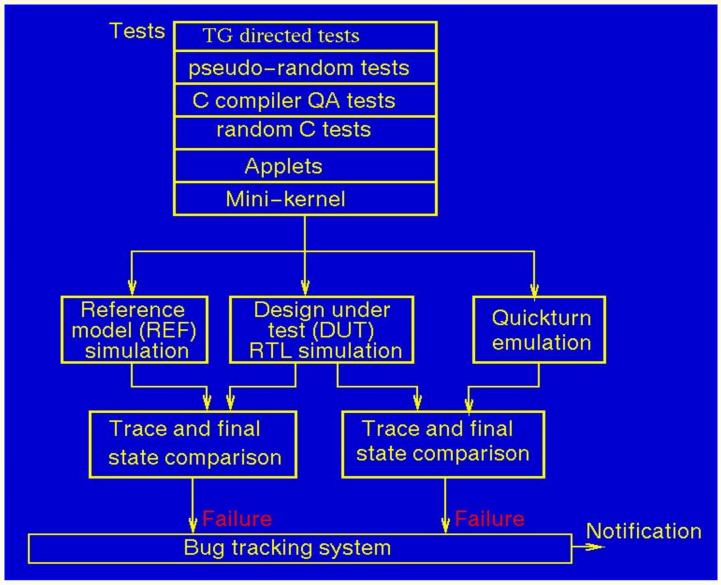
- Many instructions issued in parallel – very long instruction word (VLIW)
- Move scheduling and resource management out of hardware to optimizing compilers
- Specialized function units for serial processing
- Wide fast memory buses
- Large data and instruction caches
- Intelligent memory I/O
  - Including "Strips/Strides" and background transfers
  - Sophisticated data transfer engines

### Verification Challenges

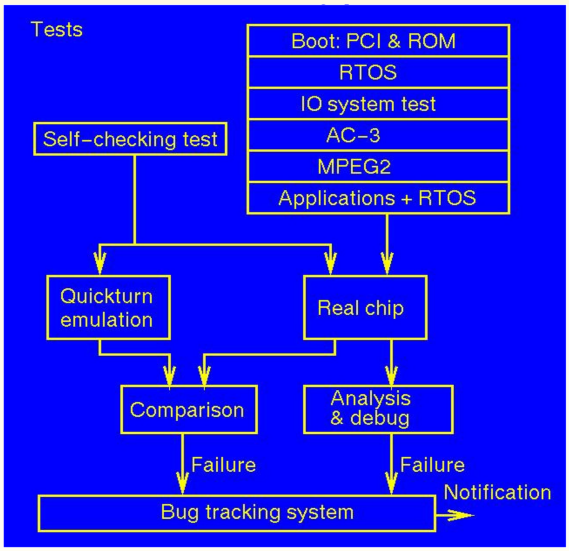
- Complex data/instruction caches, memory I/O mechanisms, large number of functional units



## Pre-Silicon Design Verification

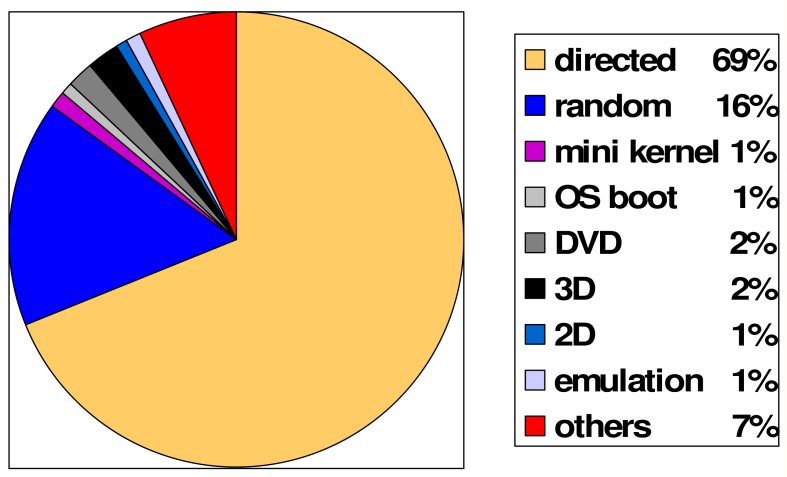


## Post-Silicon Design Verification






## Effectiveness of Tests



## Coverage-Driven Verification

### Attempt to Verify that the Design Meets Verification Goals

- Define all the verification goals up front in terms of "functional coverage points"
  - Each bit of functionality required to be tested in the design is described in terms of events, values and combinations
- Functional coverage points are coded into the HVL (Hardware Verification Language) environment (e.g., Specman 'e')
  - Simulation runs can be measured for the coverage they accomplish
- Focus on tests that will accomplishing the coverage ("coverage driven testing")
  - Then fix bugs, release constraints, improve the test environment
  - Measurable metric for verification effort



## Open Questions

### Are There Better Measures of Coverage?

- Coverage of statements in RTL would be a **necessary** but not **sufficient**
- Coverage of all states is impractical even for a design with a few hundred state variables
- Is there a way to identify a **subset of state variables** that would be tractable, and would lead to better bug detection?
- How would these variables be related to the **behavior** of the design?

## Assertions

- Assertions capture knowledge about how a design should behave
- Used in coverage-based verification techniques
- Assertions help to increase observability into a design, as well as the controllability of a design
- Each assertion specifies
  - legal behavior of some part of the design, or
  - illegal behavior of part of the design
- Examples of assertions (will be specified in a formal language)
  - The fifo should not overflow
  - Some set of signals should be "one-hot"
  - If a signal occurs, then ...



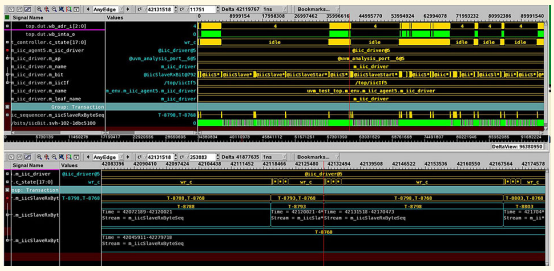
## Simulation Monitors and Assertions

```
assert_never underflow ( clk, reset_n,
    (q_valid==1'b1) && (q_underflow==1'b1));
```

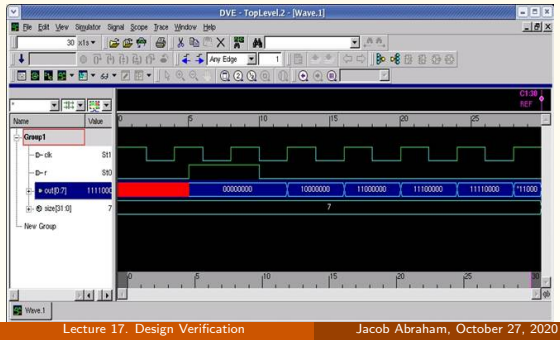

```
module assert_never (clk, reset_
 input clk, reset_n, test_expr;
 parameter severity_level = 0;
 parameter msg="ASSERT NEVER VIOLATION";
// ASSERT: PRAGMA HERE
 //synopsys translate_off
 `ifdef ASSERT_ON
  integer error_count;
  initial error_count = 0;
  always @(posedge clk) begin
   `ifdef ASSERT_GLOBAL_RESET
    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
```

## Commercial Tools

Mentor Graphics



Synopsys






## Formal Verification Approaches

- Theorem Proving: Relationship between a specification and an implementation is regarded as a theorem in a logic, to be proved within the framework of a proof calculus
  - Used for verifying arithmetic circuits in industry
- Model Checking: The specification is in the form of a logic formula, the truth of which is determined with respect to a semantic model provided by an implementation
  - Starting to be used to check small modules in industry
- Equivalence Checking: The equivalence of a specification and an implementation checked
  - Most common industry use of formal verification
- Symbolic Trajectory Evaluation: Properties specified as assertions about circuit state (pre- and post- conditions), verified using symbolic simulation
  - Used to verify embedded memories in industry

## Equivalence Checking

- Most common technique of formal verification used in industry today
  - Typically, gate-level compared with RTL
- Canonical representations, such as Binary Decision Diagrams (BDDs), or Satisfiability Solvers used for the comparison
  - Boolean equivalence checking is NP-complete
  - Multipliers require an exponential number of BDD nodes
- Commercial tools available from many vendors



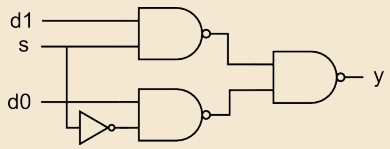
## Equivalence Checking

- Validate that the implementation of a module is consistent with the specification
  - Can use simulation or formal techniques
  - Combinational or sequential modules

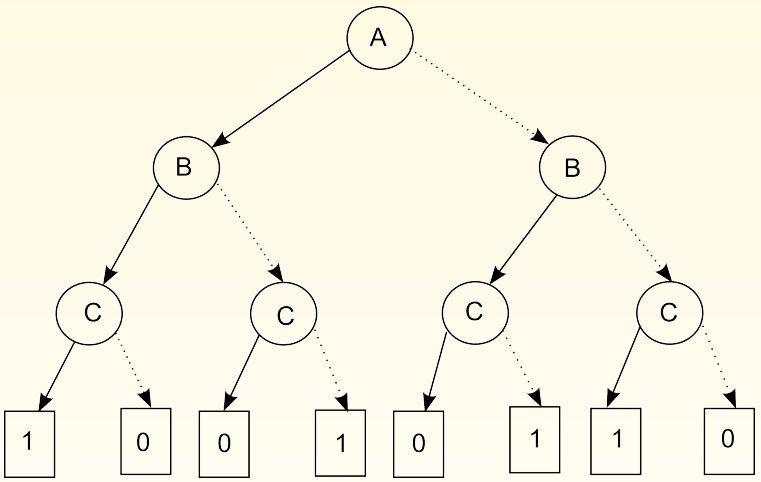
**Example: Specification in RTL**

```
module mux(input s, d0, d1,
           output y);
assign y = s ? d1 : d0;
endmodule
```

**Example: Implementation at the gate level**

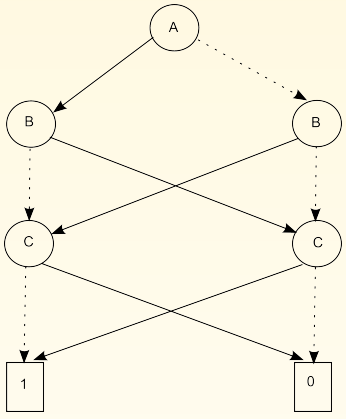


## Decision Tree for $A \oplus B \oplus C$






## Reduced, Ordered BDD (ROBDD)




$F = A \oplus B \oplus C$

Reduced, Ordered BDDs (ROBDDs) are canonical

Can represent sets of states, state-transition relations, etc.

Structure and complexity of ROBDDs for *Symmetric Functions?*

## Satisfiability (SAT) Solvers

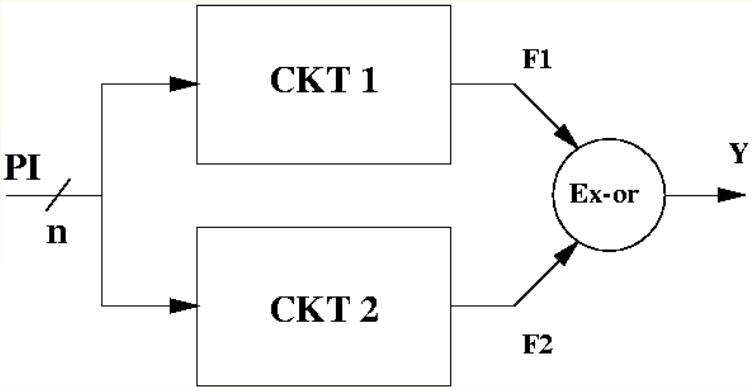
### Can a Boolean Function be Satisfied?

- Cast an equivalence checking problem as a SAT problem
- Starts by converting Boolean formula into the Conjunctive Normal Form (CNF) – (product of sums)
  $(a+b+c)(a+\overline{e}+f)(\overline{c}+\overline{d}+g)\ldots$
- Goal is to find an assignment satisfying every term (if any clause is 0, there is no satisfying assignment)
- Commercial and Open SAT solvers available
- Most verification tools now use BDDs + SAT
- Some bring in ATPG ideas – called "structural SAT"



## Use of ATPG for Equivalence Checking

- Use a tool (Automatic Test Pattern Generator) which generates manufacturing tests
- Detecting a "stuck-at-0" fault at Y (requires an input which generates a 1 on Y) will prove inequivalence of the two circuits
- Approach is not memory limited (like BDDs)



## Design Verification

- Digital systems similar to **reactive programs**
- Digital systems receive inputs and produce outputs in a continuous interaction with their environment
- Behavior of digital systems is concurrent because each gate in the system simultaneously evaluating its output as a function of its inputs

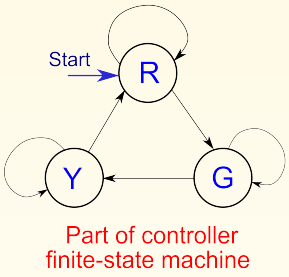
### Check Properties of Design

- Since specification is usually not formal, check design for properties that would be consistent with the specification
- Safety "something bad will never happen"
- Liveness Property: "something good will eventually happen"
- Temporal Logic and variations commonly used to specify properties
- Example: Linear Temporal Logic (LTL) or Computation Tree Logic (CTL)



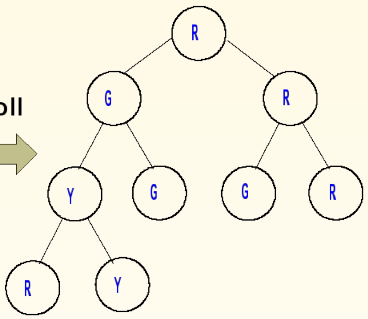
## Example of Computation Tree

Traffic light controller




Part of controller finite-state machine

Unroll




Computation Tree

## System Verilog Assertions (SVA)

**SVA**

- Assertions: Predicates placed in program
- Immediate and Concurrent Assertions
- assert, assume, cover, expect constructs

**Immediate Assertions**

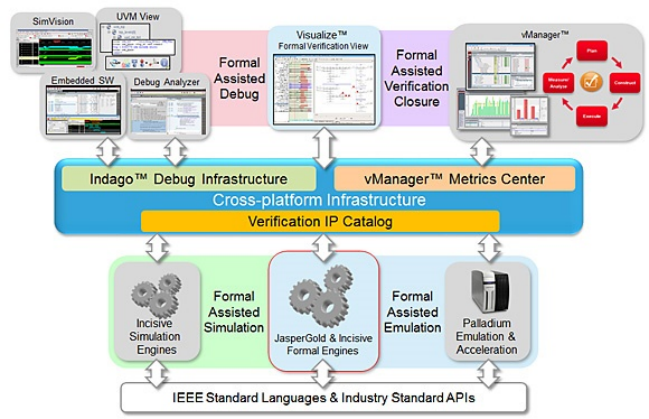
assert (a == b);

**Concurrent Assertions**

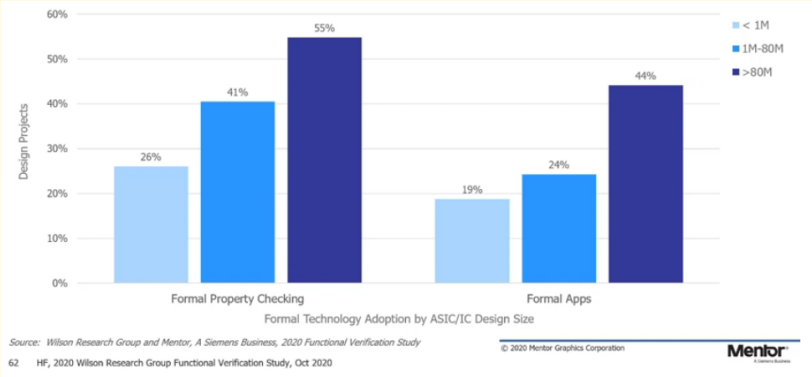
assert property (@(posedge clk) req |→ ack);



## Cadence Formal Verification



## Adoption of Static (Formal) Techniques in ASICs




Source: Wilson Research Group and Mentor, 2020 Functional Verification Study – https://go.mentor.com/5ffxz



## Dealing with State Explosion

### Verification is a very difficult problem

- Even combinational equivalence checking problems (ATPG, SAT) are NP-complete
- Checking sequential properties is only possible for small designs
- Additional problem of generating correct "wrappers" for the module being verified

### How can we deal with the complexity?

- Use more powerful computers?
  - Computers double in capability (assuming we can program multi-core processors) every couple of years
  - Adding one state variable to a design doubles its states
- Exploit hierarchy in the design
- Develop powerful abstractions

## Program Slicing

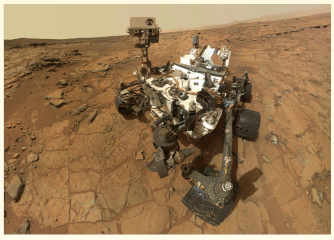
### A Slice of a Design

- Represents behavior of the design with respect to a given set of variables (or slicing criterion)
- Proposed for use in software in 1984 (Weiser)
- Slice generated by a control/data flow analysis of the program code
- Slicing is done on the **structure** of the design, so scales well
- "Static analysis"



## Dramatic Example of Design Bug Detection and Recovery


BAE RAD750 (133 MHz)



Science Lab on Mars

- Mars Science Laboratory – MSL ("Curiosity") relies extensively on BAE RAD750 chip running at 133 MHz
- During cruise to Mars (circa January 2012), MSL processes are unexpectedly reset – no code bug is found (7 months remaining before landing)
- Misbehavior eventually traced to processor hardware malfunction: instruction flow depends on processor temperature
- Only possible fix was via software – done successfully