19. Manufacturing Test

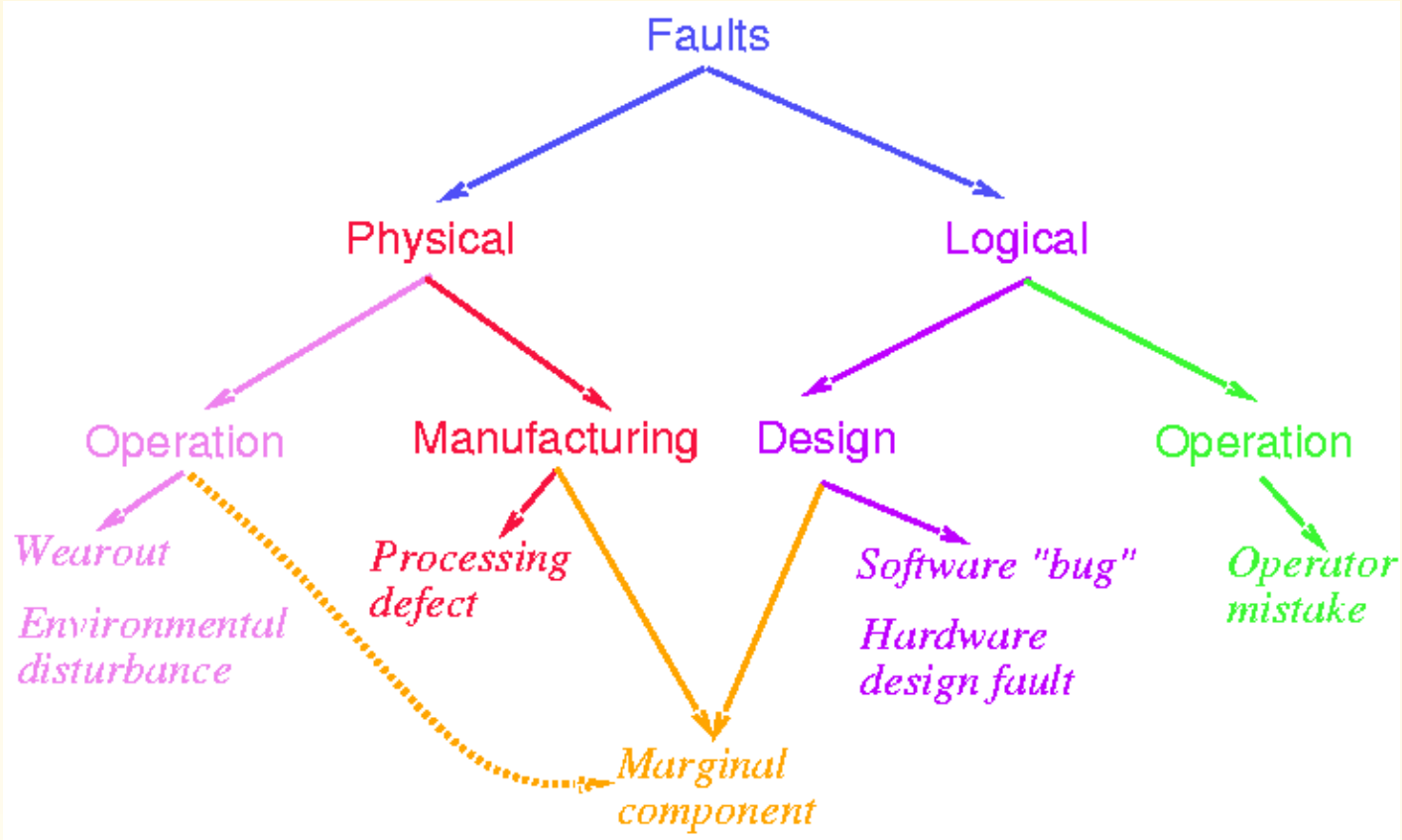
Jacob Abraham

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

VLSI Design
Fall 2020

November 3, 2020

Dealing with Faults



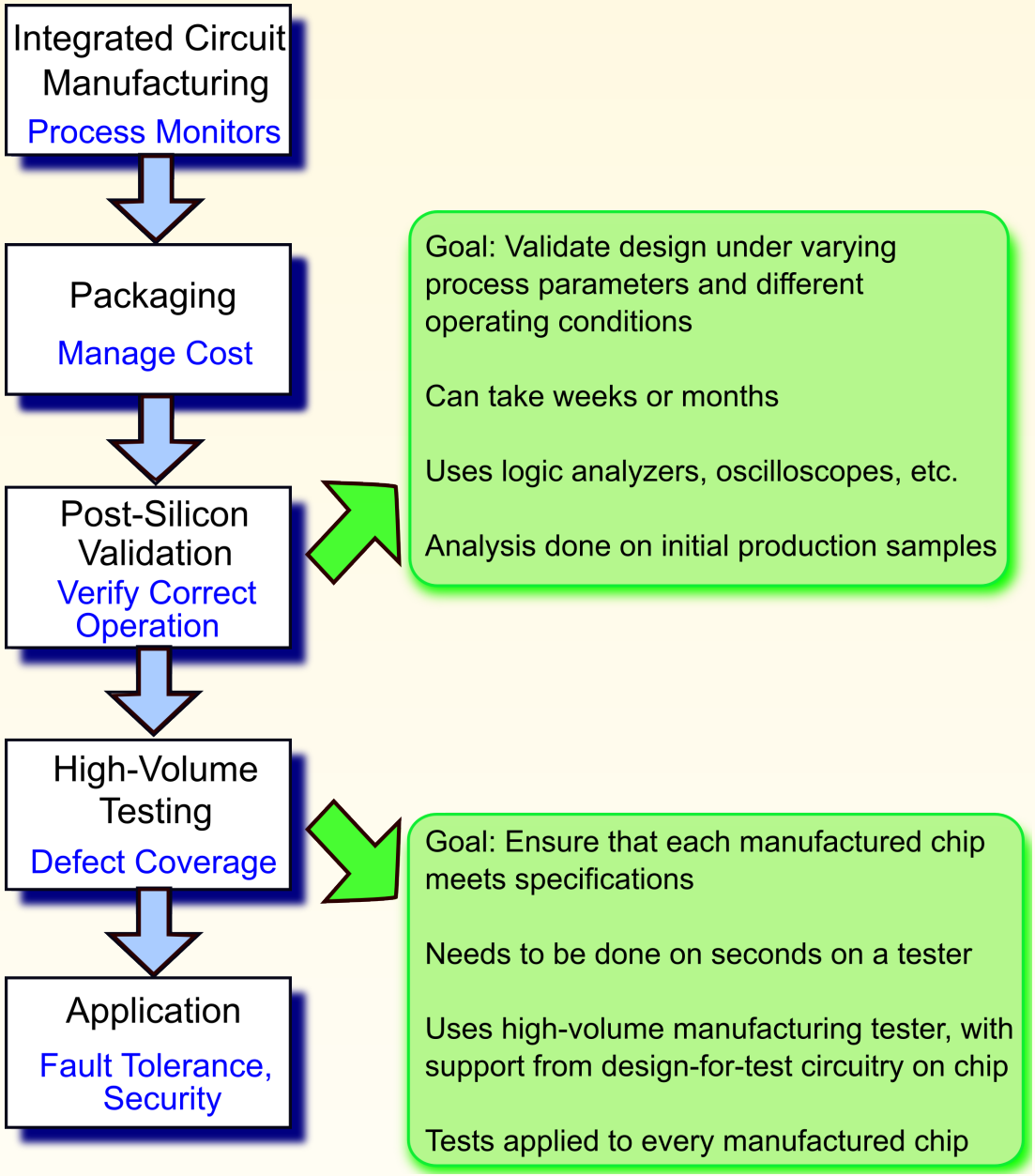
- Design faults: simulation and emulation, formal techniques
- Manufacturing faults, field failures: testing, design for testability
- Operational faults: concurrent error detection and fault tolerance

Silicon Debug

- Test the first chips back from fabrication
  - If you are lucky, they work the first time
  - If not ...
- Logic bugs vs. electrical failures
  - Most chip failures are **logic bugs** from inadequate simulation or verification
  - Some are electrical failures
    - Crosstalk
    - Dynamic nodes: leakage, charge sharing
    - Ratio failures
    - A few are tool or methodology failures (e.g. DRC)
- Fix the bugs and fabricate a corrected chip

- **Silicon debug (or "bringup") is primarily a Non-Recurring Engineering (NRE) cost (like design)**
- **Contrast this with manufacturing test which has to be applied to every part shipped**

Post-Silicon Validation and Manufacturing Test



Shmoo Plots

- How to diagnose failures?
  - Difficult to access chips
    - Picoprobes
    - Electron beam
    - Laser voltage probing
    - Built-in self-test
- Shmoo plots
  - Vary voltage, frequency
  - Look for cause of electrical failures

Clock period in ns on the left, frequency increases going up
Voltage on the bottom, increase left to right

** indicates a failure*

```
1.0   *   *   *   *   *   *
1.1   *   *   *   *   *
1.2   *   *   *   *
1.3   *   *   *
1.4   *   *
1.5   *
      1.0 1.1 1.2 1.3 1.4 1.5
```

Normal
Well-behaved shmoo
Typical Speedpath

```
1.0   *       *       *
1.1       *       *       *
1.2   *       *       *
1.3       *       *       *
1.4   *       *       *
1.5       *       *       *
      1.0 1.1 1.2 1.3 1.4 1.5
```

"Brick Wall"
Bistable
Initialization

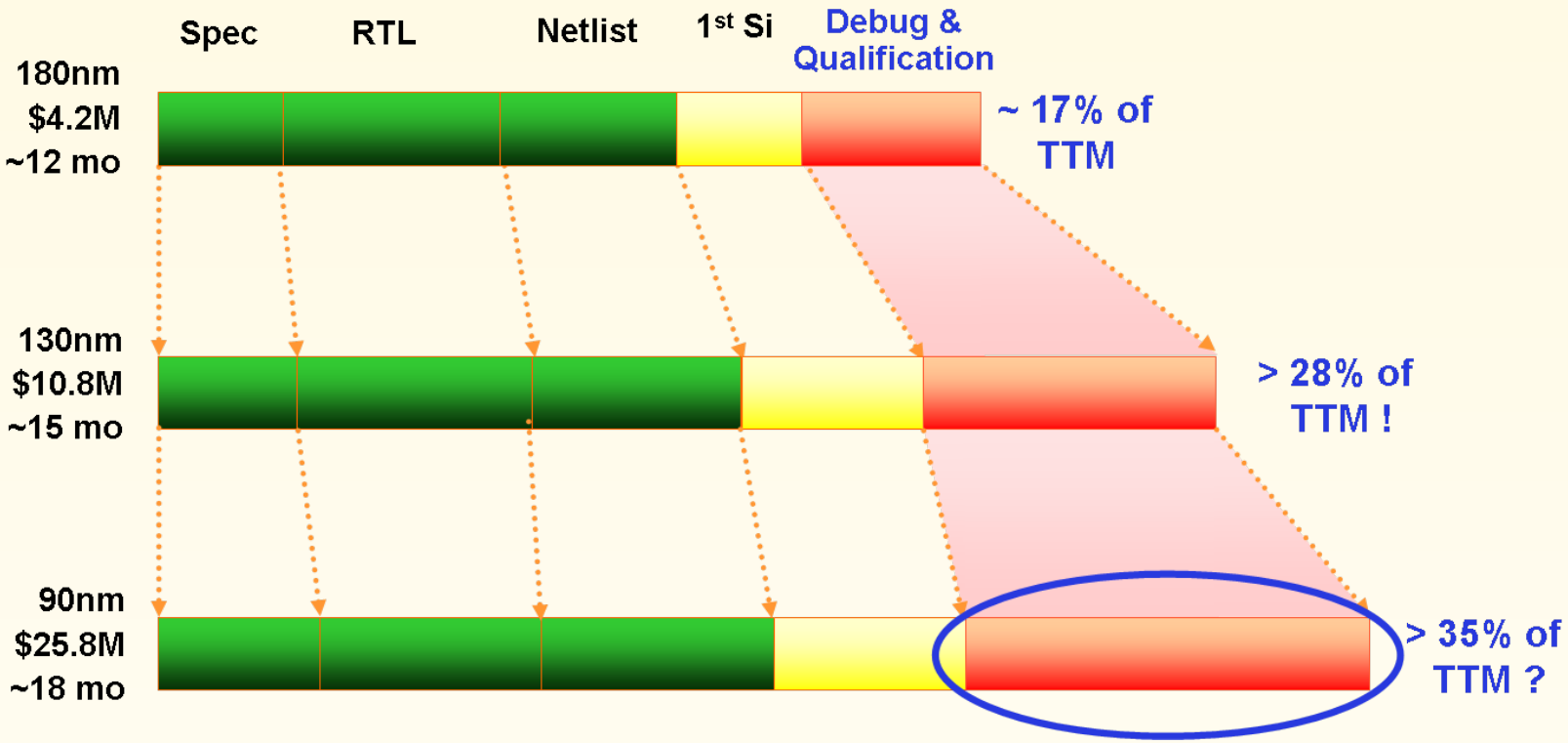
```
1.0           *   *   *
1.1           *   *   *
1.2           *   *   *
1.3           *   *   *
1.4           *   *   *
1.5           *   *   *
      1.0 1.1 1.2 1.3 1.4 1.5
```

"Wall"
Fails at a certain voltage
Coupling, charge share, races

```
1.0       *   *   *   *   *
1.1           *   *   *   *
1.2               *   *   *
1.3                   *   *
1.4                       *
1.5                       *
      1.0 1.1 1.2 1.3 1.4 1.5
```

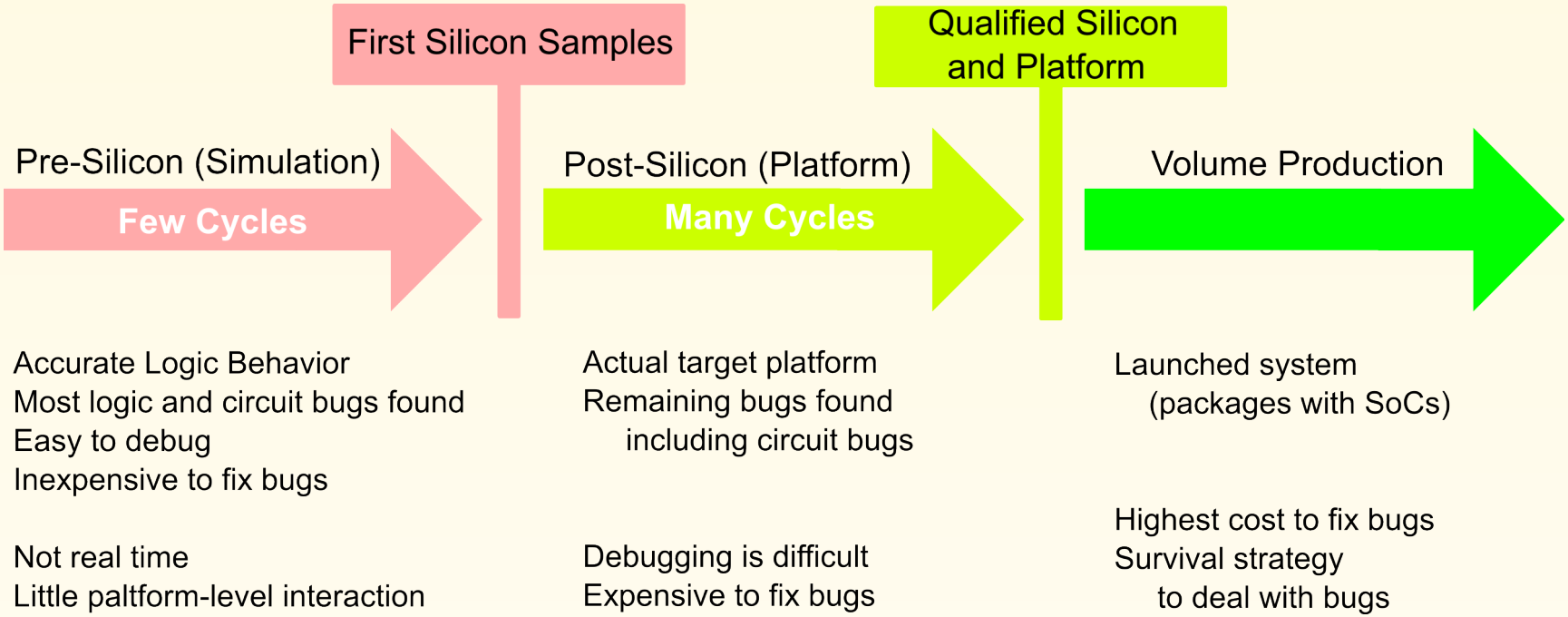
"Reverse speedpath"
Increase in voltage reduces frequency
Speedpath, leakage

Silicon Debug is a Growing Barrier to Market Entry



Source: DAFCA

Validation Domains



Bugs decline in number over development cycle, but cost to fix them increases

Source: N. Hakim, Intel

Where Bugs are Found

Functional bugs (also known as "logic bugs")

- Exist in all manufactured parts (Metric: DPM (defects per million), fatal logic bugs result in 1M DPM)
- 98% found before tape out, 2% post-silicon

Circuit bugs

- Not all parts exhibit failures ($<$ 1M DPM)
- Variable with Voltage, Temperature, Frequency, process and component age
- Computation limits to simulation (not real time) limits extent of variation combinations which can be simulated
- 90% found pre-silicon, 10% post-silicon

Source: N. Hakim, Intel

Validating a Design

Difficulties

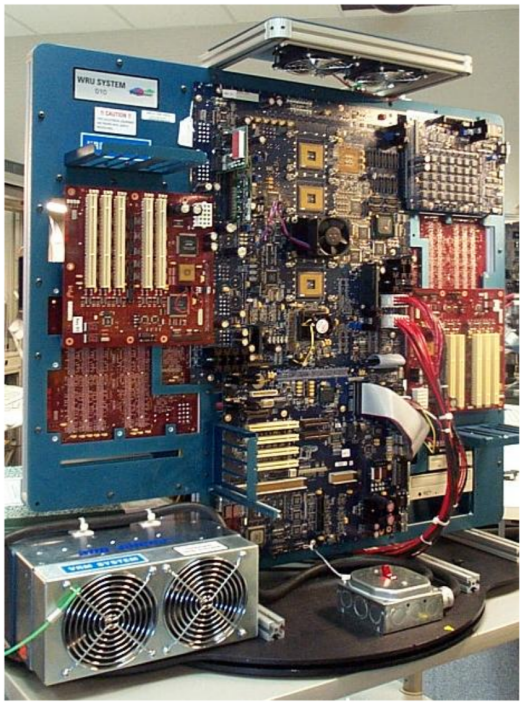
- Lack of visibility to internal blocks and interconnect buses
- High latency between an internal error caused by a fault and its observation at the pins

Solutions

- Software-Based – software monitor routines and processor-specific hardware allow some visibility
- Test Feature-Based – reuse design-for-test (DFT) structures for functional debug
- In-Circuit Emulation – a special "bond-out" version of the device is created that mirrors key internal signals on external device pins
- On-chip Emulation – dedicated debug logic runs in parallel to the normal device logic

Platform Validation Infrastructure

Tests generated in server farm



Tests executed on validation platform

Failures debugged using logic analyzer

Source: N. Hakim, Intel

Classes of Tests

Functional bugs in micro-architecture

- Weighted random instructions
- Architectural simulation patterns
- Random power state transitions
- Directed tests for corner cases
- Multi-core/multi-processor tests
- Tests of virtualization system

Memory subsystem

- Memory channel activation
- Tests for multiple cores/processors
- Directed tests for memory paging, cache coherence

Circuit Bugs

Affect DPM – not all die behave the same way

Timing convergence bugs

- Speed path: circuit operates too slow
- Min-delay: circuit operating too fast (hold times)
- Race: circuit fails due to timing of multiple converging signals

Analog bugs

- Primarily occur in I/O buffers, PLLs, and thermal sensors
- Silicon does not operate in accordance with predicted (simulated) circuit behavior

- Fundamentals for circuit bug hunting
  - Need sufficiently large population of devices
  - Need to vary environmental conditions
  - Need to stimulate stressful system behavior
  - Stimulus is generally functional – failures look just like functional failures

Circuit Bug Root Causes

On-die signal integrity

- Cross-coupling induced noise
- Droop-event induced noise

Power delivery integrity

- High dynamic current events
- Clock gating often results in high dynamic current

Clock domain crossing

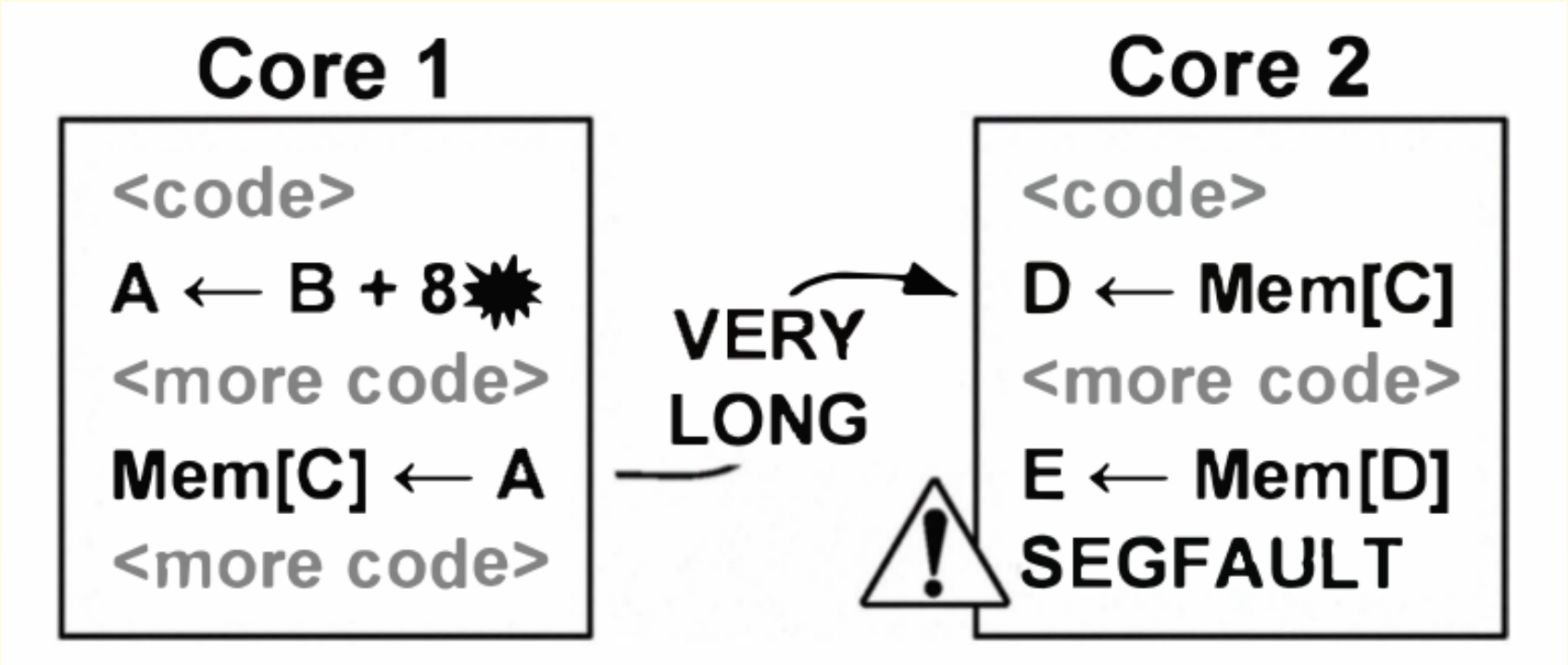
- May cause synchronization and timing issues

Process, Voltage, Temperature

- Power state transistors
- Silicon process variation

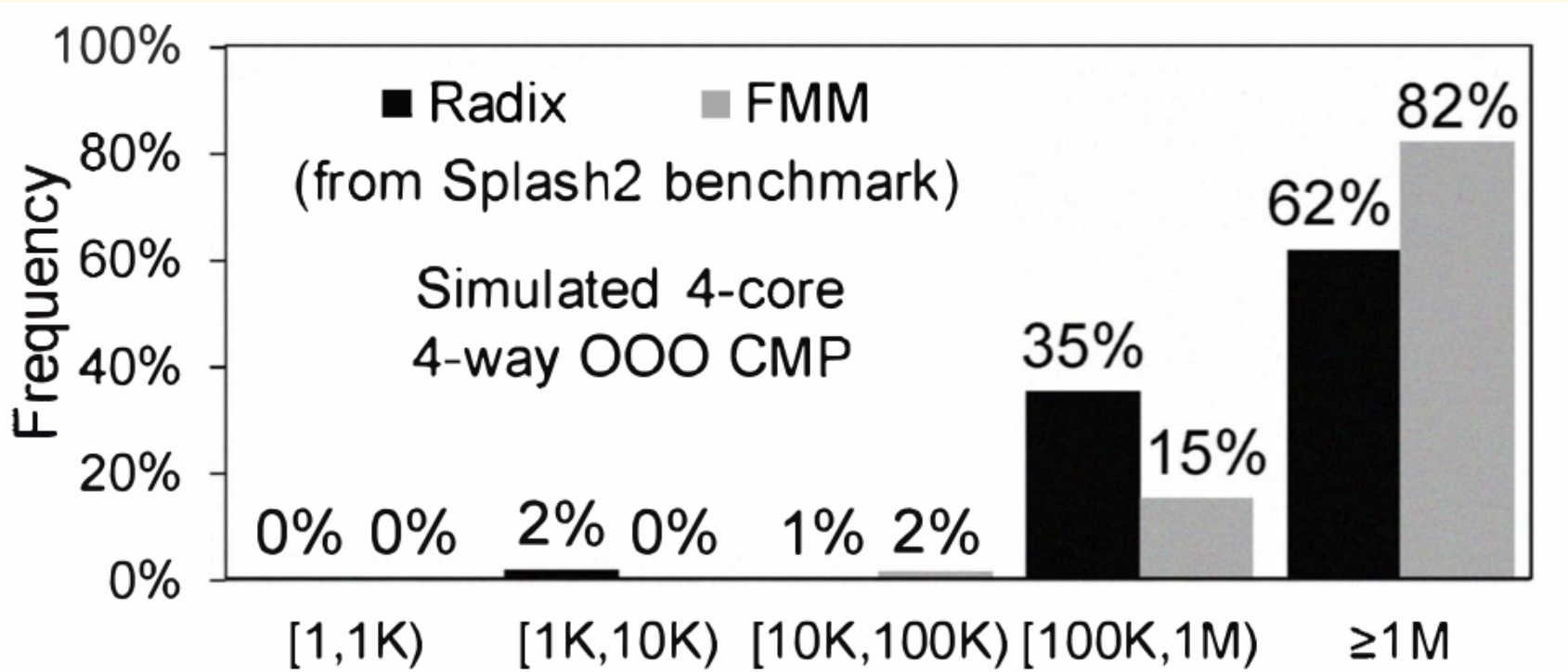
Source: N. Hakim, Intel

Problem: Very Long Error Detection Latencies in Practise

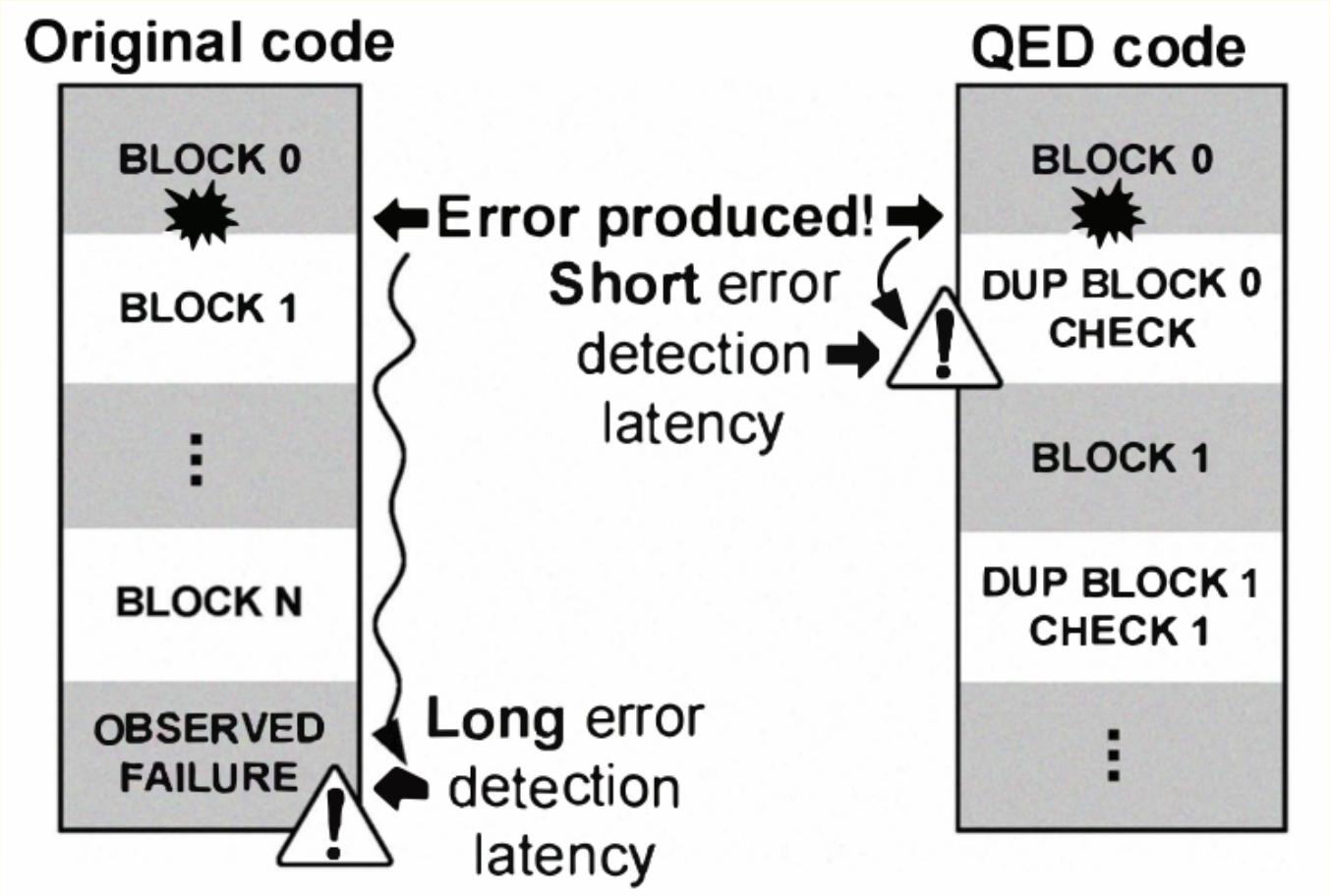


- Inter-core interactions

Distribution of Inter-Core Store-to-Load Latencies



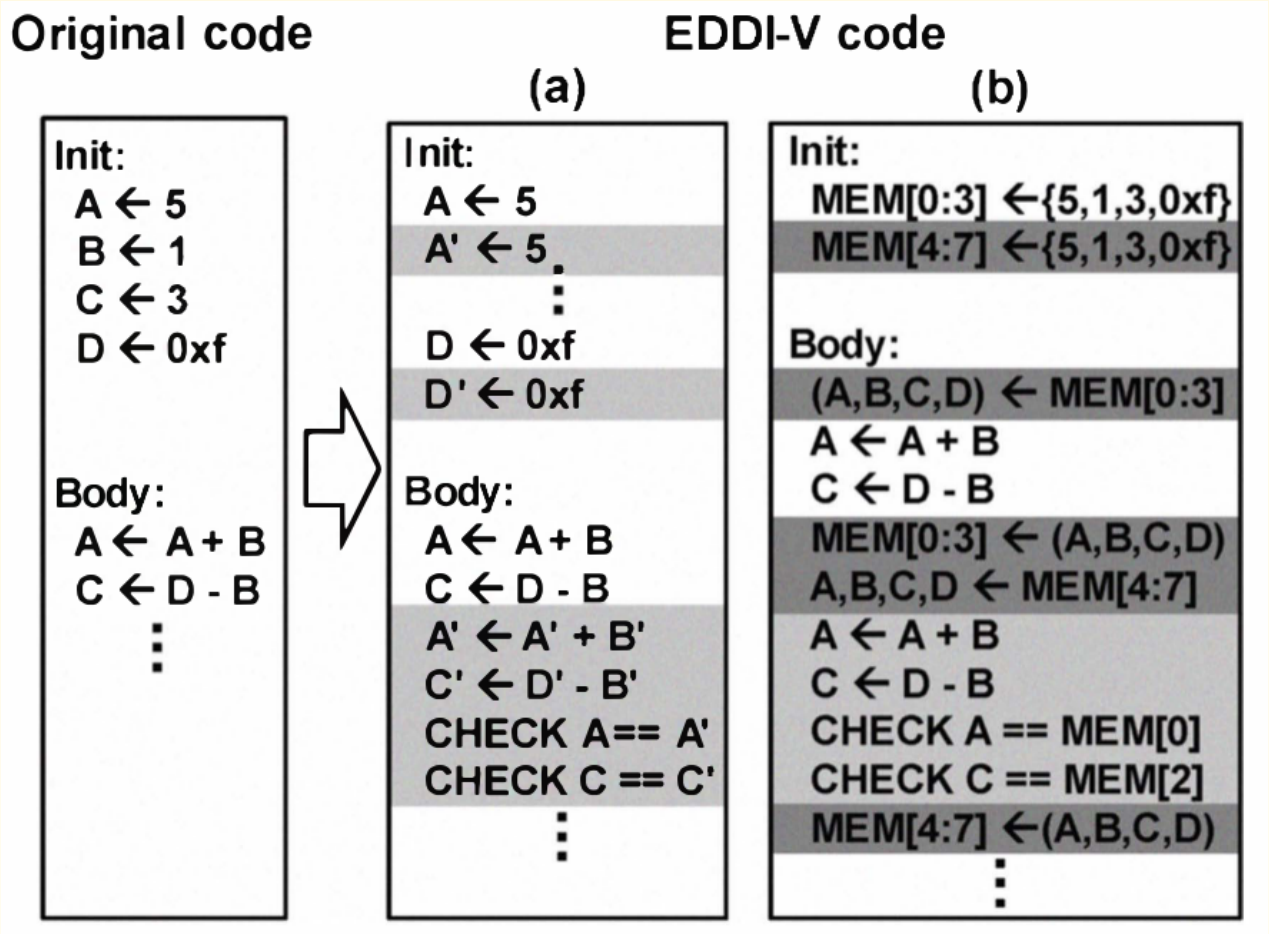
Much Lower Error Detection Latency with EDDI-V-based QED (Quick Error Detection) Test



(EDDI-V: Error Detection by Duplicated Instructions for Validation)

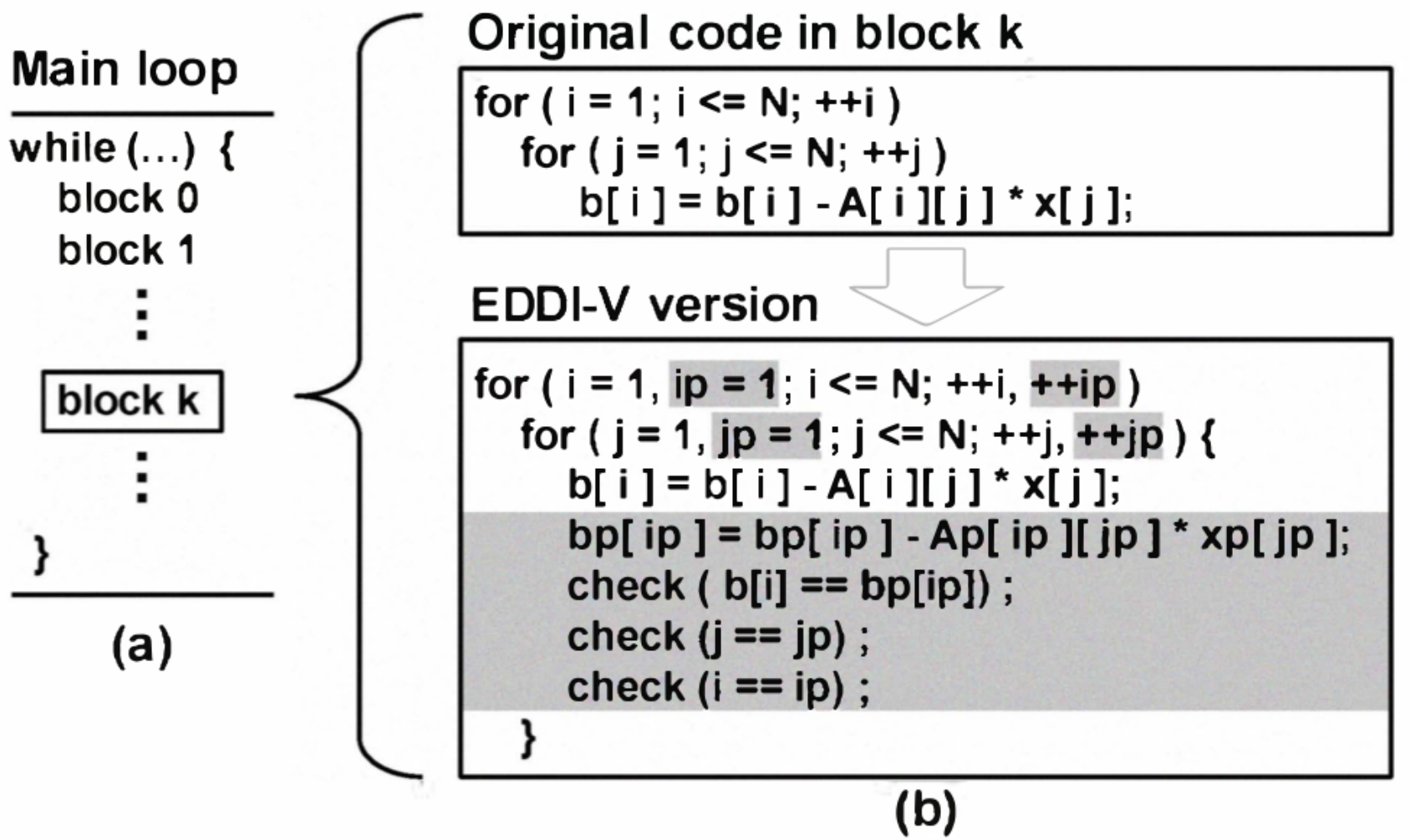
Source: Hong et al., "QED: Quick Error Detection tests for effective post-silicon validation," IEEE International Test Conference, 2010

EDDI-V Transformations



EDDI-V Transformations: (a) with half of all general-purpose registers reserved, (b) with no registers reserved and register values stored in memory

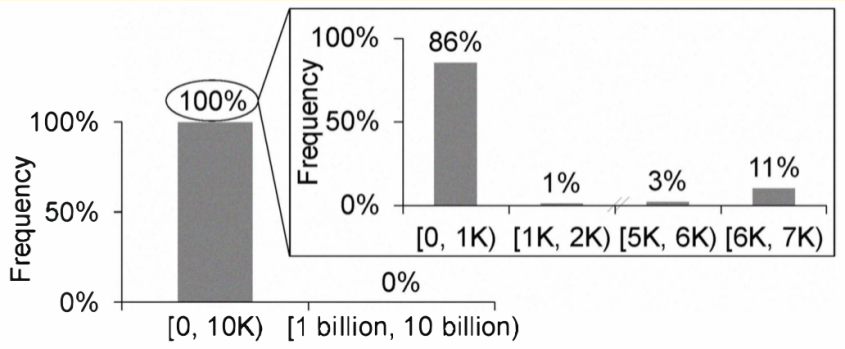
Linpack Experiment



(a) Linpack Program structure
(b) Source code level EDDI-V transformation

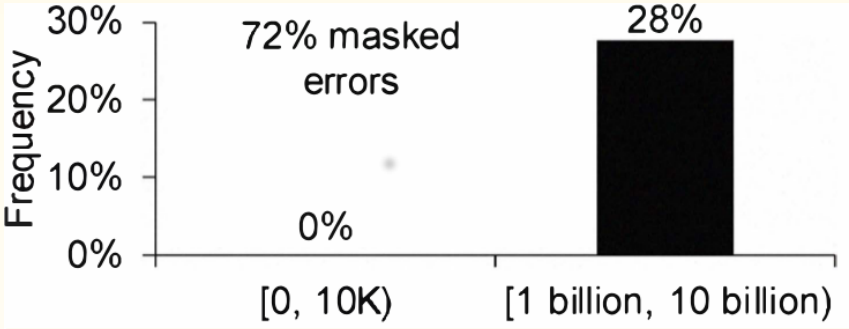
Error Detection Latency Results with Linpack

Linpack test using QED checks



Execution Cycles

Linpack test using only end-result-checks

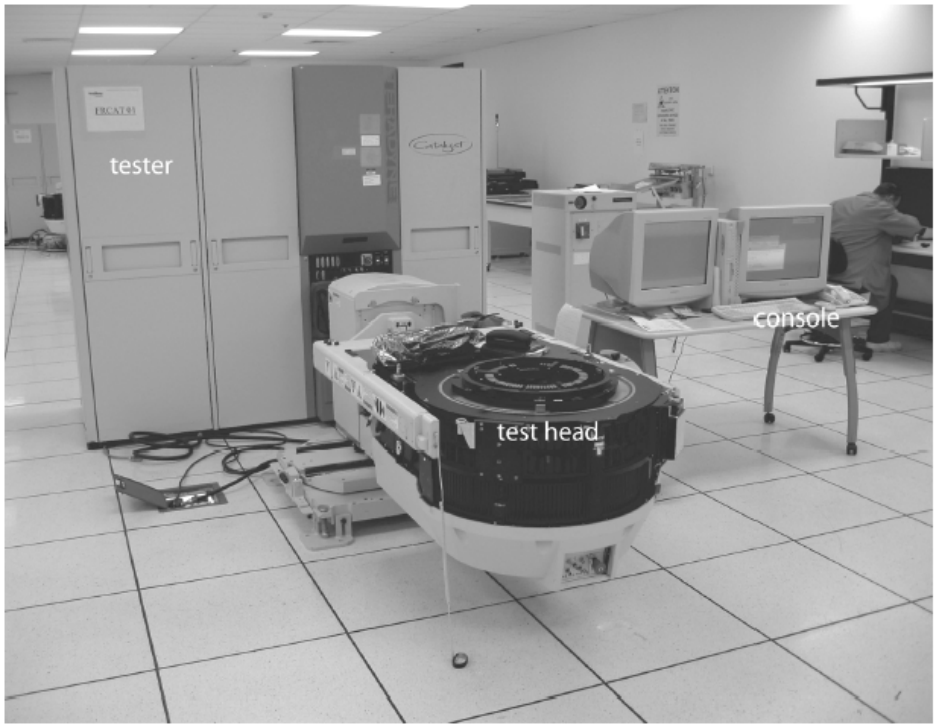


Execution Cycles

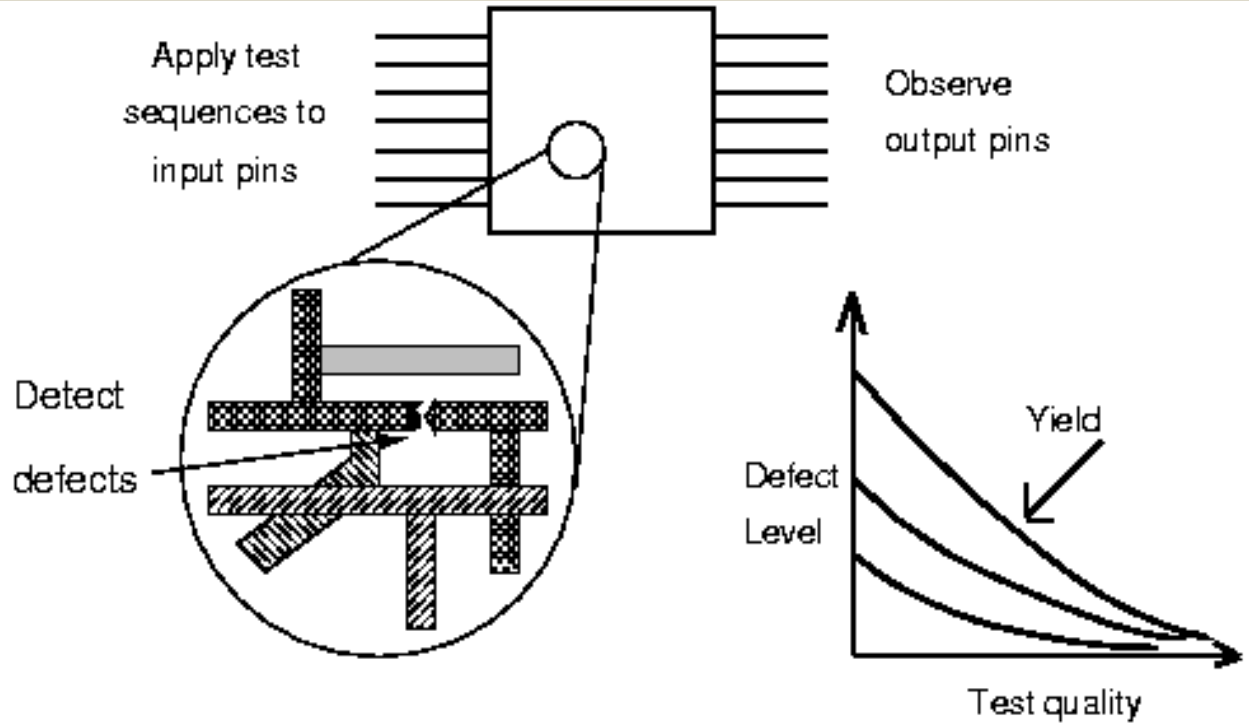
Manufacturing Test

- A speck of dust on a wafer is sufficient to kill chip
- Yield of any chip is $< 100\%$
  - Must test chips after manufacturing before delivery to customers to only ship good parts

- Manufacturing testers are very expensive
  - **Minimize time on tester**
  - Careful selection of **test vectors**



Manufacturing Test



- A test for a defect will produce an output response which is different from the output when there is no defect
- Test quality is high if the set of tests will detect a very high fraction of possible defects
- Defect level is the percentage of bad parts shipped to customers
- Yield is the percentage of defect-free chips manufactured

Fault Models

- Numerous possible physical failures (what we are testing for)
- Can reduce the number of failure types by considering the effects of physical failures on the logic functional blocks: called a FAULT MODEL
- Most widely used fault model: "stuck-at" faults at the gate level
  - Assume that defects will cause the circuit to behave as if lines were "stuck" at logic 0 or 1
- Most commercial tools for test are based on the "stuck-at" model
- Other fault models
  - "Stuck open" model for charge retained on a CMOS node
  - Recent use of the "transition" fault model in an attempt to deal with delays
  - "Path delay" fault model would be better for small delay defects, but the large number of possible paths is an impediment to the use of this fault model

Generating Tests for a Fault

Start with the set of faults of interest, reduce number of faults (use "equivalence", "dominance")
Find a vector or sequence of vectors (sequential circuit, delay tests) which will cause faulty to produce incorrect output
Initial state assumption for sequential circuit (most general assumption: memory elements start with "X" (unknown) state)

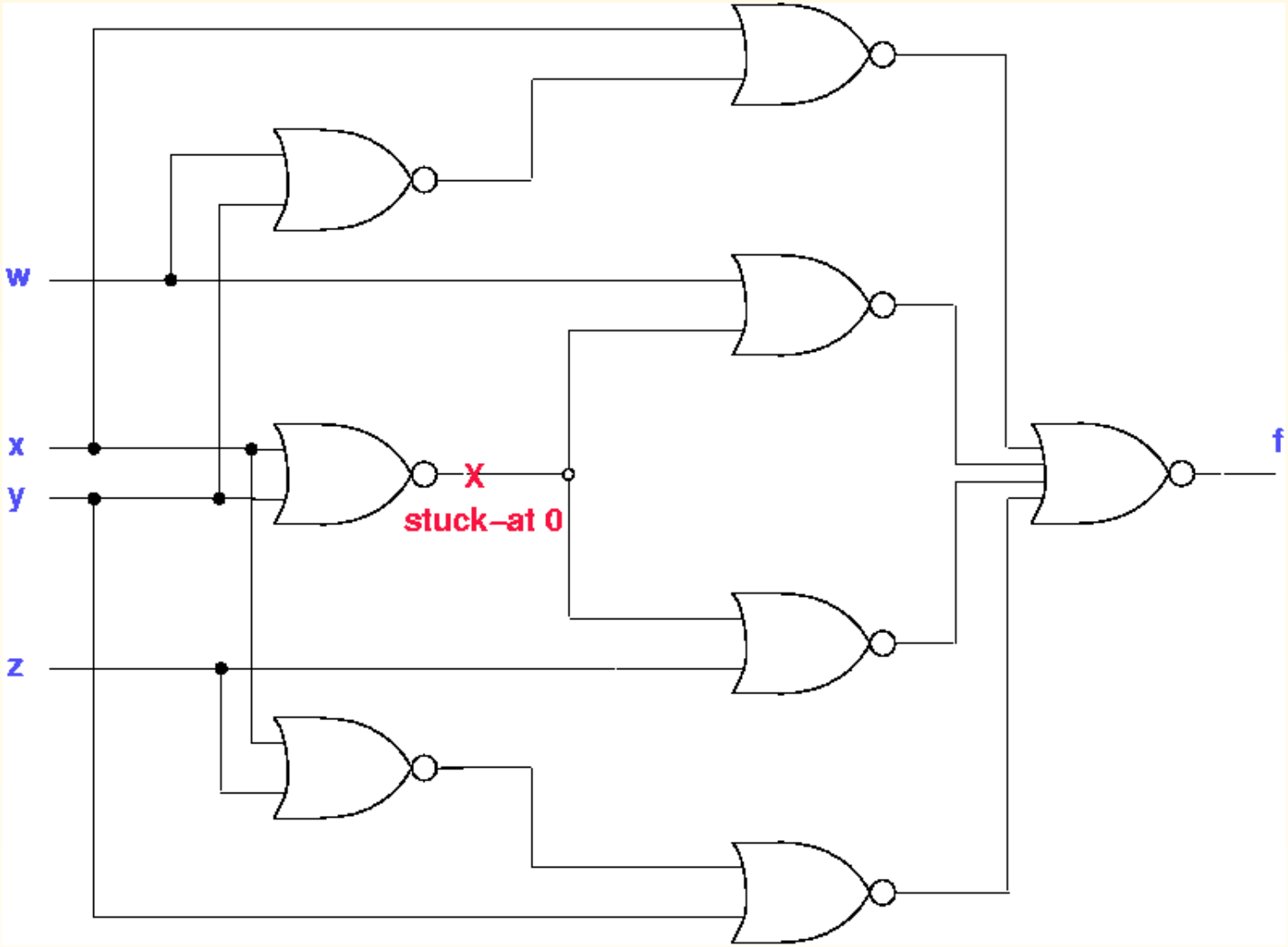
Steps in Test Generation

- **Activate** fault (produce error at fault site)
- **"Sensitize"** path from fault to output (propagate error to output)
- **"Justify"** internal signals to primary inputs
- Choices may exist during sensitization and justification: if conflicts arise, need to **backtrack**
- If no test exists, fault is **redundant**
- **Problem is NP-Complete**

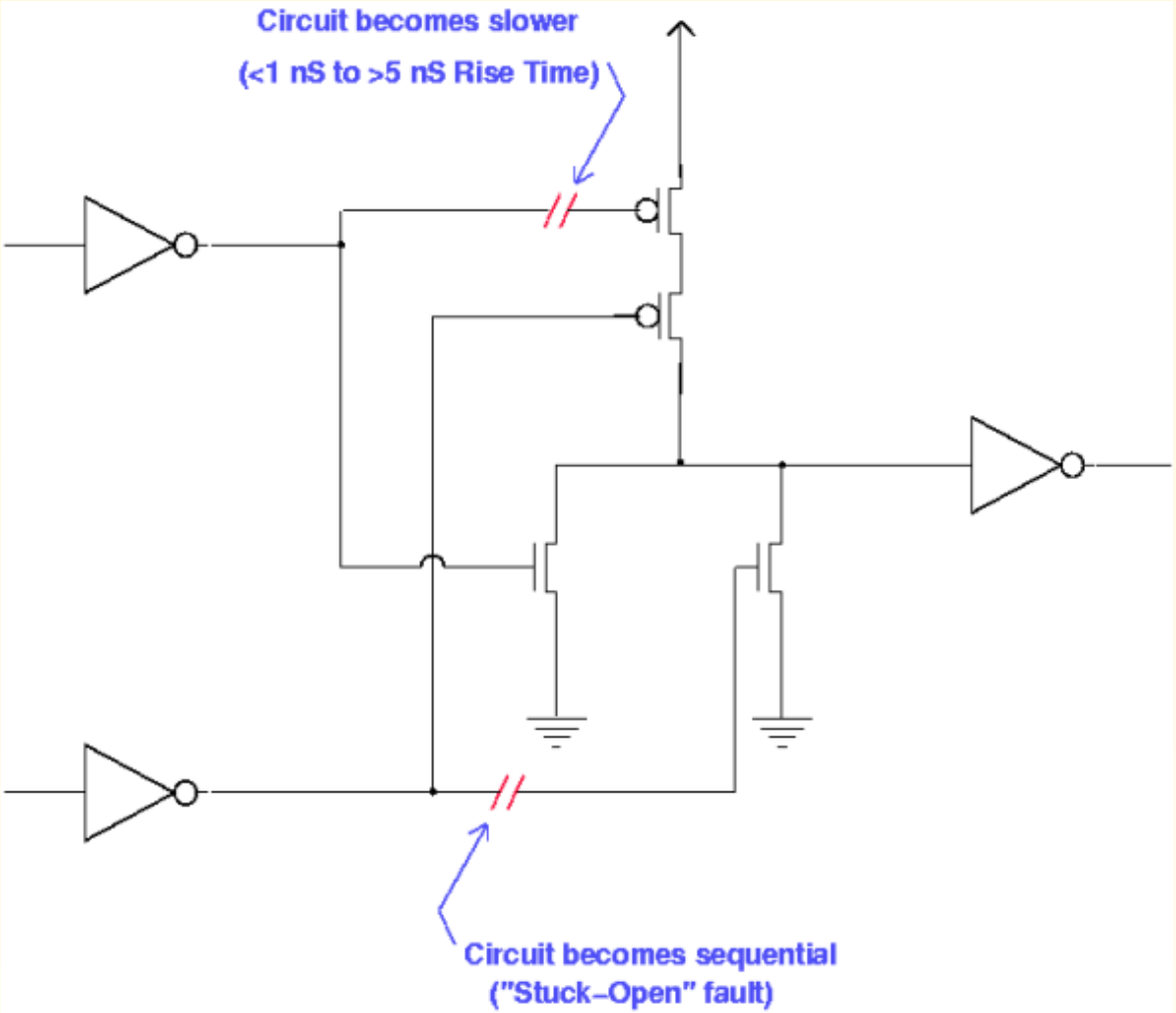
Observability and Controllability

- **Observability**: ease of observing a value on a node by monitoring external output pins of the chip
- **Controllability**: ease of forcing a node to 0 or 1 by driving input pins of the chip
- Combinational logic is usually easier to observe and control
  - Still, NP-complete problem
- Finite state machines can be very difficult, requiring many cycles to enter desired state
  - Especially if state transition diagram is not known to the test engineer, or is too large

Example of Test Generation



Test for Opens in CMOS NOR Gate

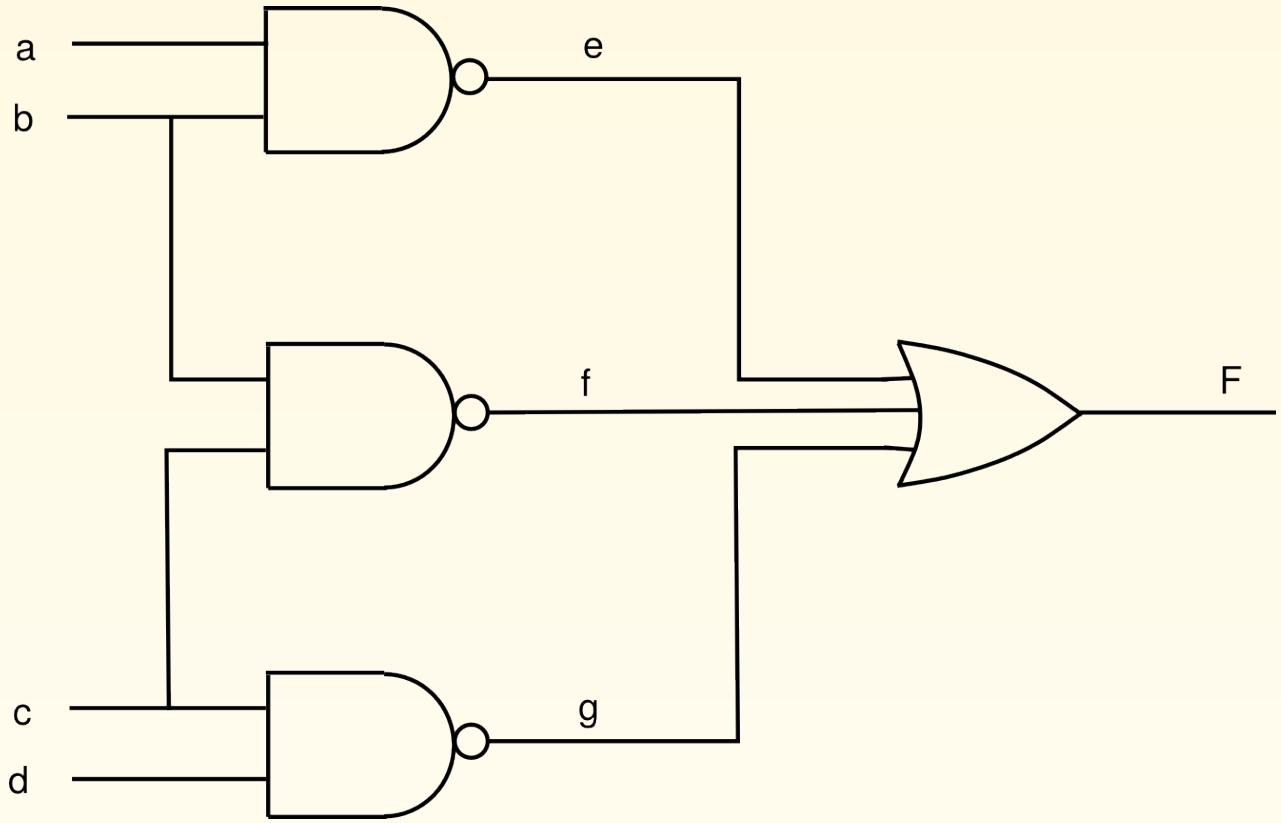


Test for the stuck-open fault at gate of n-channel gate

Pattern 11 (followed by)

Pattern 10

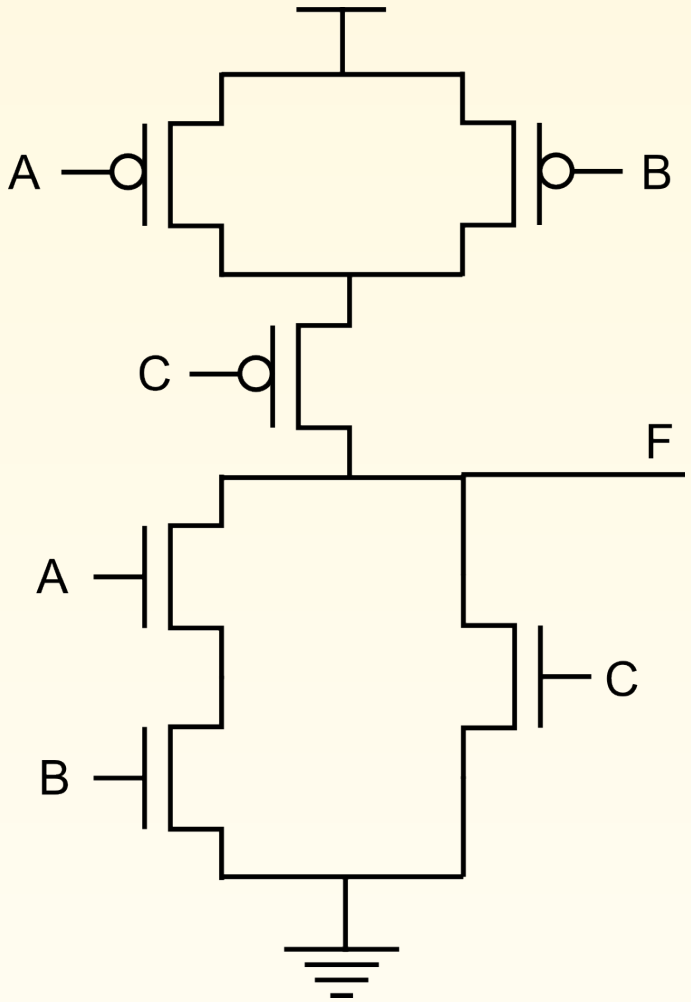
Example



Find tests for:

1. g s-a-0
2. e s-a-1
3. f s-a-0

Example



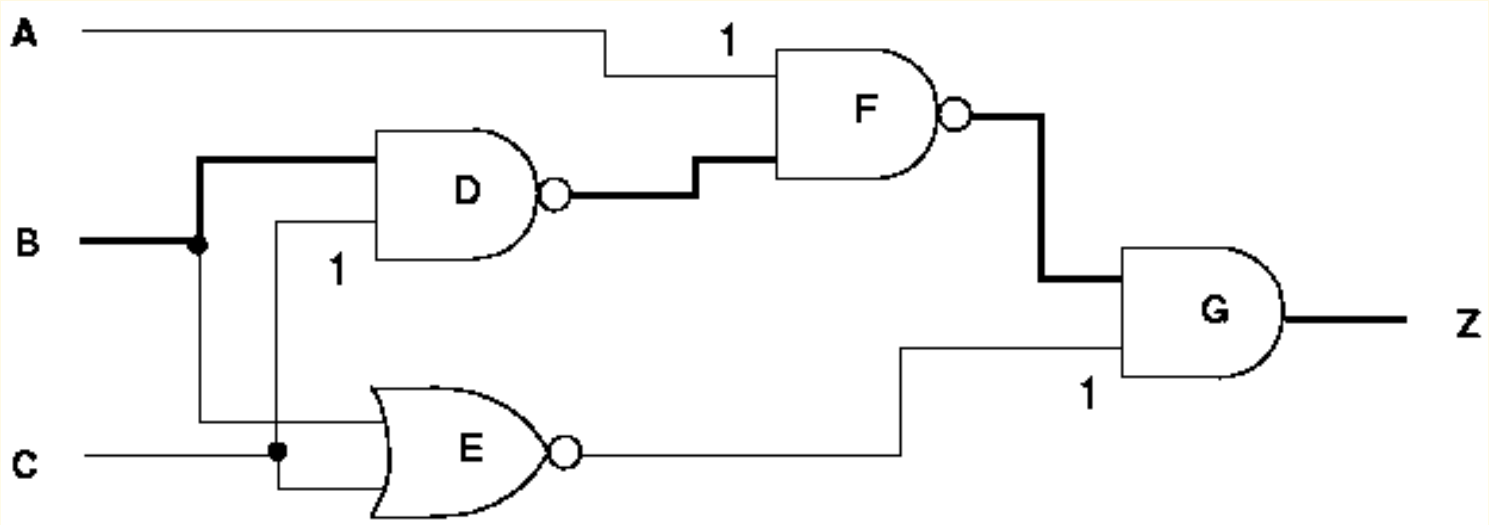
Find tests for:

1. nMOS transistor with input B slow to turn on
2. pMOS transistor with input A stuck ON

Fault Simulation

- Identify faults detected by a sequence of tests
- Provide a numerical value of coverage (ratio of detected faults to total faults)
- Correlation between high fault coverage and low defect level
- Faults considered
  - Generally, gate level "stuck-at" faults
  - Can also evaluate coverage of switch level faults
  - Can include timing and dynamic effects of failures
- Although fault simulation takes polynomial time in the number of gates, it can still be prohibitive for large designs
- Recent research: techniques for **accurate estimation of the fault coverage**

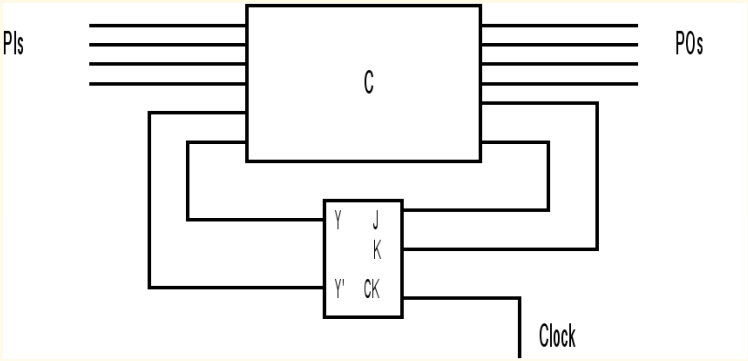
Delay Test and Timing Verification

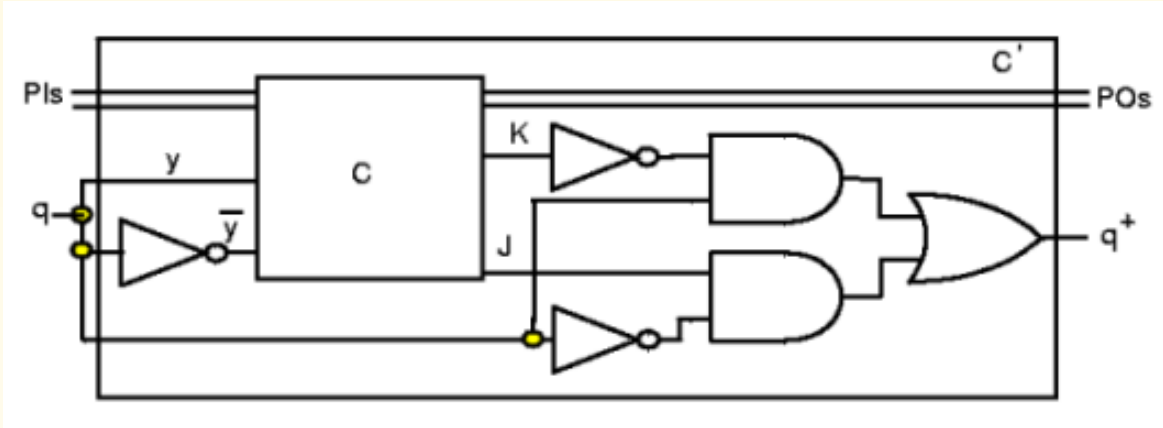


- Static timing analysis (Primetime, for example) only finds **structural** long paths
- **False Path** problem:
  - In order to allow a signal to go through the path,
  - Required Side Inputs: C = 1, A = 1, E = 1
  - Conflict due to C = 1 and E = 1
- Can use modified test generation algorithms to identify longest true paths in a circuit
  - **CRITIC** from UT
  - **Primetime+Tetramax** from Synopsys

Sequential Circuit Test Generation

"Unroll" sequential circuit into an iterative logic array model for one "time frame"





- Single stuck fault in circuit appears in every time frame (may affect propagation of errors)
- Usually assume that the single clock is fault-free
- In general, no prior knowledge of the number of time frames needed for propagation and justification

Functional Test Generation

- Attempt to generate tests when detailed structural information is not available, or for extremely complex systems
- **Useful for verification and speed tests**
- **Requirement for successful functional test:**
  - Check for unintended functions in addition to the correct one
  - Physical failures can cause spurious operations while performing the desired function correctly
  - Ad-hoc functional tests typically do not check for such behavior
- Applied to generating tests for memories, microprocessors

Functional Fault Model for Memories

Memory Arrays

- One or more cells become stuck at 1 or 0
- One or more cells fail to undergo a 0 to 1 or a 1 to 0 transition
- There exist two (or more) cells which are **coupled**
  - A 0 to 1 (or a 1 to 0) transition in a cell (due to a write in that cell) changes the contents of another cell from 0 to 1 or from 1 to 0
  - If transition in cell I changes J, transition in J may not affect the state of cell I
- Multiple cells accessed during READ or WRITE

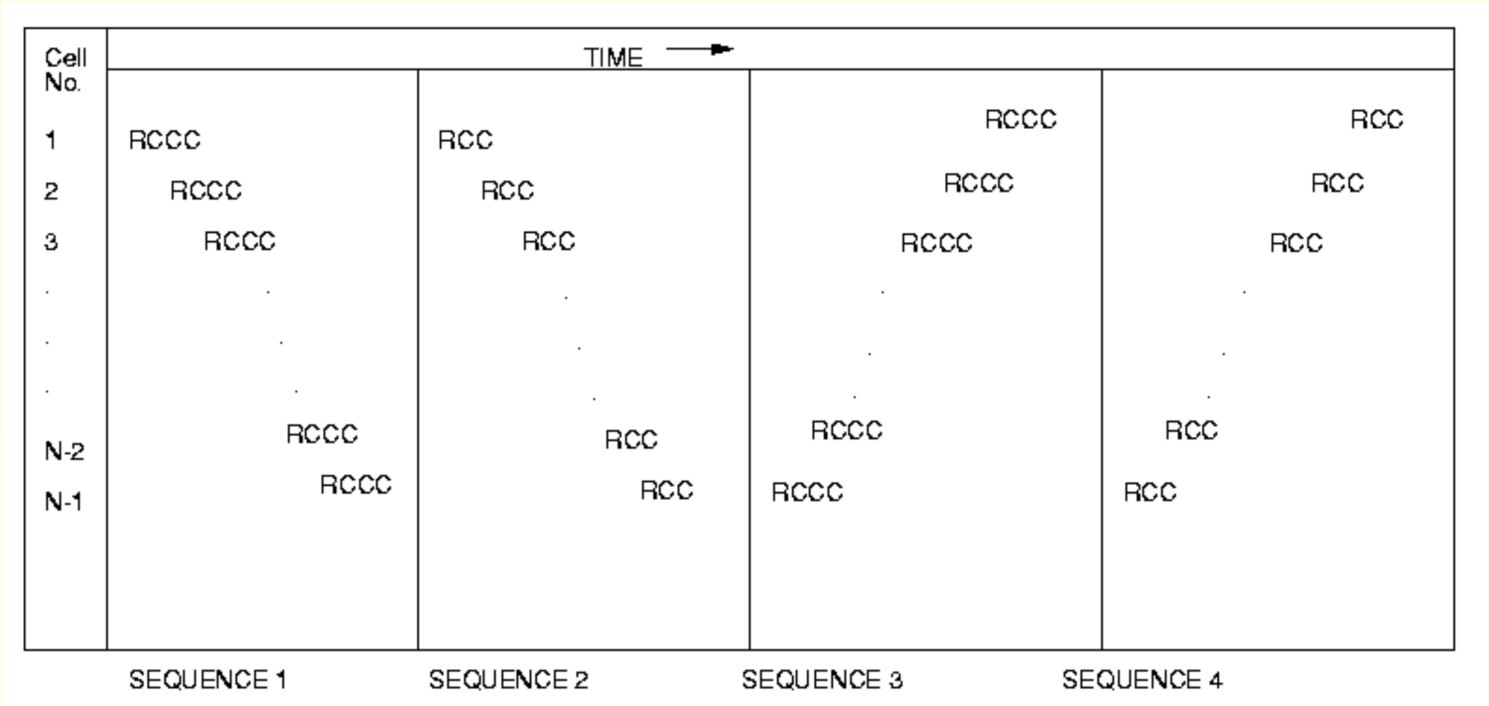
Memory Fault Model, Cont'd

Decoders

- The decoder will not access the addressed cell, and in addition may access non-addressed cells
- The decoder will access multiple cells, including the addressed cell

- Assumption that the combinational logic of the decoder will not be transformed into sequential logic
- **Decoder faults look like memory cell array faults**
- Fault model can be validated by simulating effects of faults at the transistor level

O(n) "March" Test for Memories



R: Read cell and verify; C: Complement cell

Complexity of Test: 14N (N cells)

How would you test for parametric faults, data retention?

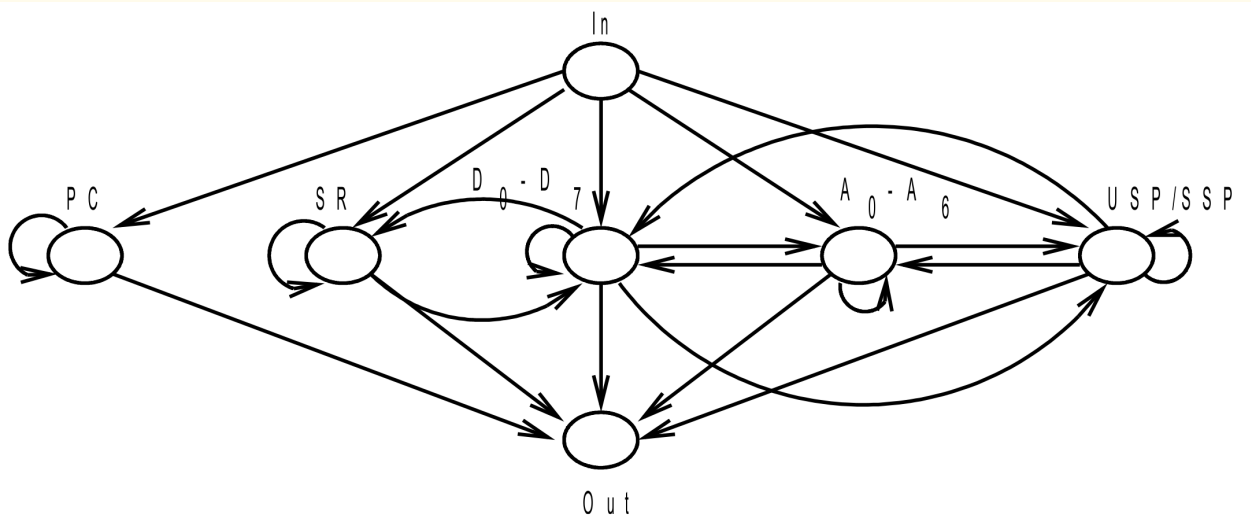
All memory tests are based on this algorithm

Functional Testing of Microprocessors

- Developed in the 80s for generating tests based on information about the instruction set
  - Tests for vendor parts without knowing details
- Tests based on **"functional fault models"** derived from analysis of the effects of low level faults on the behavior of modules
  - Based on functional tests for memories
- Fault models at control sequencing level
- Tests based on high-level information can also be used for **validating design correctness**

Microprocessor Functional Tests

- Microprocessor represented as a graph
  - Node: register or set of equivalent registers
  - Edge: data or information transfer between nodes
- Instructions: sequence of microinstructions (set of micro-operations)
- Tests based on behavior level fault models



Example: data flow graph model of Motorola 68000

Microprocessor Fault Models

Register decoding function – like a decoder

- Decoder will not access the addressed register (or storage cells)
- Decoder may access non-addressed registers or multiple registers, including addressed location (decoder remains combinational)

Instruction sequencing function

- One or more micro-ops are inactive, therefore the instruction is not executed completely
- Micro-ops which are normally inactive become active
- A set of micro-instructions is active in addition to, or instead of, the normal microinstructions

Microprocessor Fault Models, Cont'd

Data transfer function

- Any number of lines can be stuck at 0 or 1
- Any pair of lines i, j can be coupled (a logic value on line i will cause the logic value on line j to be changed)
- More complex effect: A particular pattern on a set of lines can disturb another line (example, capacitive coupling on buses)

Data Manipulation Function

- Assume tests available for the functional blocks (derived separately) using ATPG tools, for example

Testing Instruction Decoding, Control

- Use sequence of instructions Read(R_i) which transfers data in register i to a location in memory without changing the internal state of the microprocessor
  - Check core instructions (Load, Compare, Branch)
  - Check that every register can be loaded and read (without disturbing other registers)
  - Test Load Register instruction for all registers, all addressing modes
  - Check all other instructions
- In self-test mode, compare with stored data, branch to error location if incorrect

Example Code Words for 68000 Registers

Register	Code Pattern
D0	11111101111111111111111111111111
D1	11111110111111111111111111111111
D2	11111111011111111111111111111111
D3	11111111101111111111111111111111
D4	11111111110111111111111111111111
D5	11111111111011111111111111111111
D6	11111111111101111111111111111111
D7	11111111111110111111111111111111
A0	11111111111111011111111111111111
A1	11111111111111101111111111111111
A2	11111111111111110111111111111111
A3	11111111111111111011111111111111
A4	11111111111111111101111111111111
A5	11111111111111111110111111111111
A6	11111111111111111111011111111111
USP	11111111111111111111101111111111
SSP	11111111111111111111110111111111

"m-out-of-n" code

AND or OR of any two code words will produce a non-code word

Procedure to Test All Instructions

Load the registers with unique "code words", cw_i, using instruction sequence, WRITE(R_i)

```
for every instruction I {
  for every register Ri {
    Write(Ri) with cwi;
  }
  Execute I;
  for every register Ri {
    Read(Ri);
  }
}
```

Complexity of test patterns: $O(n_i n_r + n_r^4)$
where n_i is the number of instructions and N_r is the number of registers

Data Storage and Transfer Tests

00000000000000000000000000000000
11111111111111111111111111111111

00000000000000001111111111111111
11111111111111110000000000000000

00000000111111110000000011111111
11111111000000001111111100000000

00001111000011110000111100001111
11110000111100001111000011110000

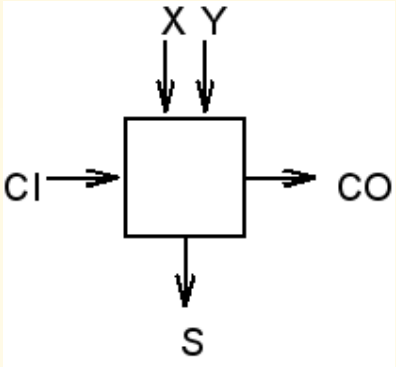
00110011001100110011001100110011
11001100110011001100110011001100

01010101010101010101010101010101
10101010101010101010101010101010

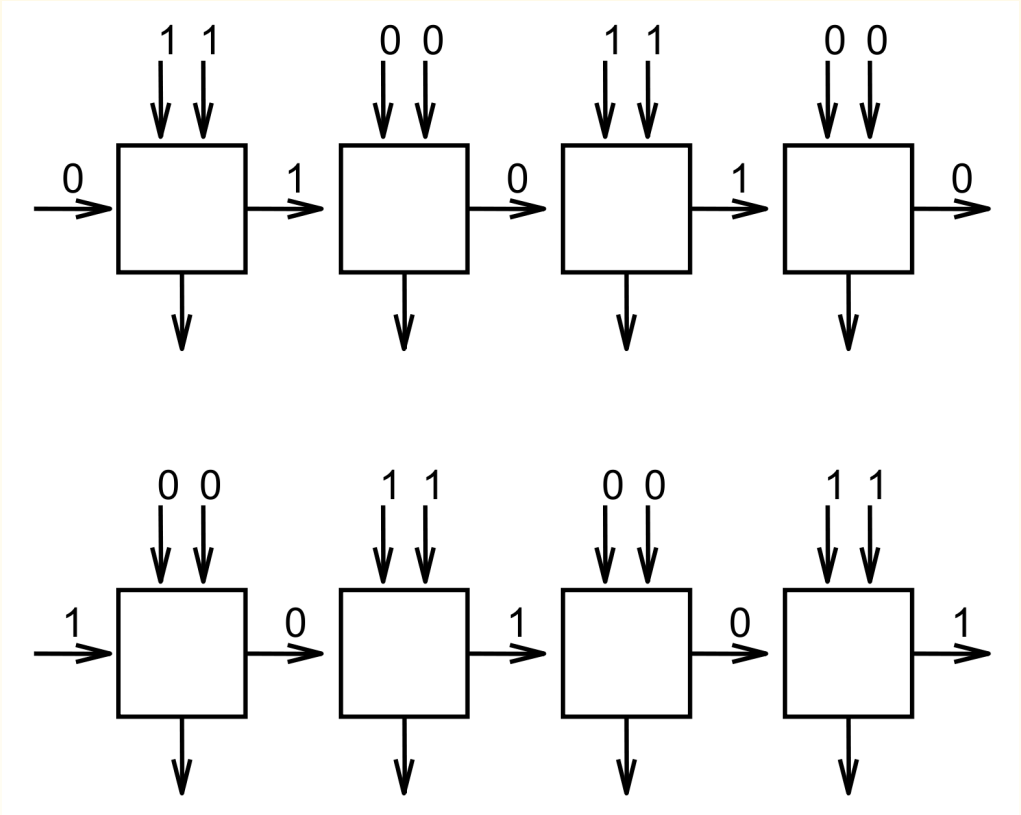
Data patterns to check all logical paths and all registers (32-bit transfer path)

- Test length logarithmic in number of bits (divide and conquer)
- Tests will detect any "stuck-at", short or open in data path

Constant Tests for Ripple-Carry Adder



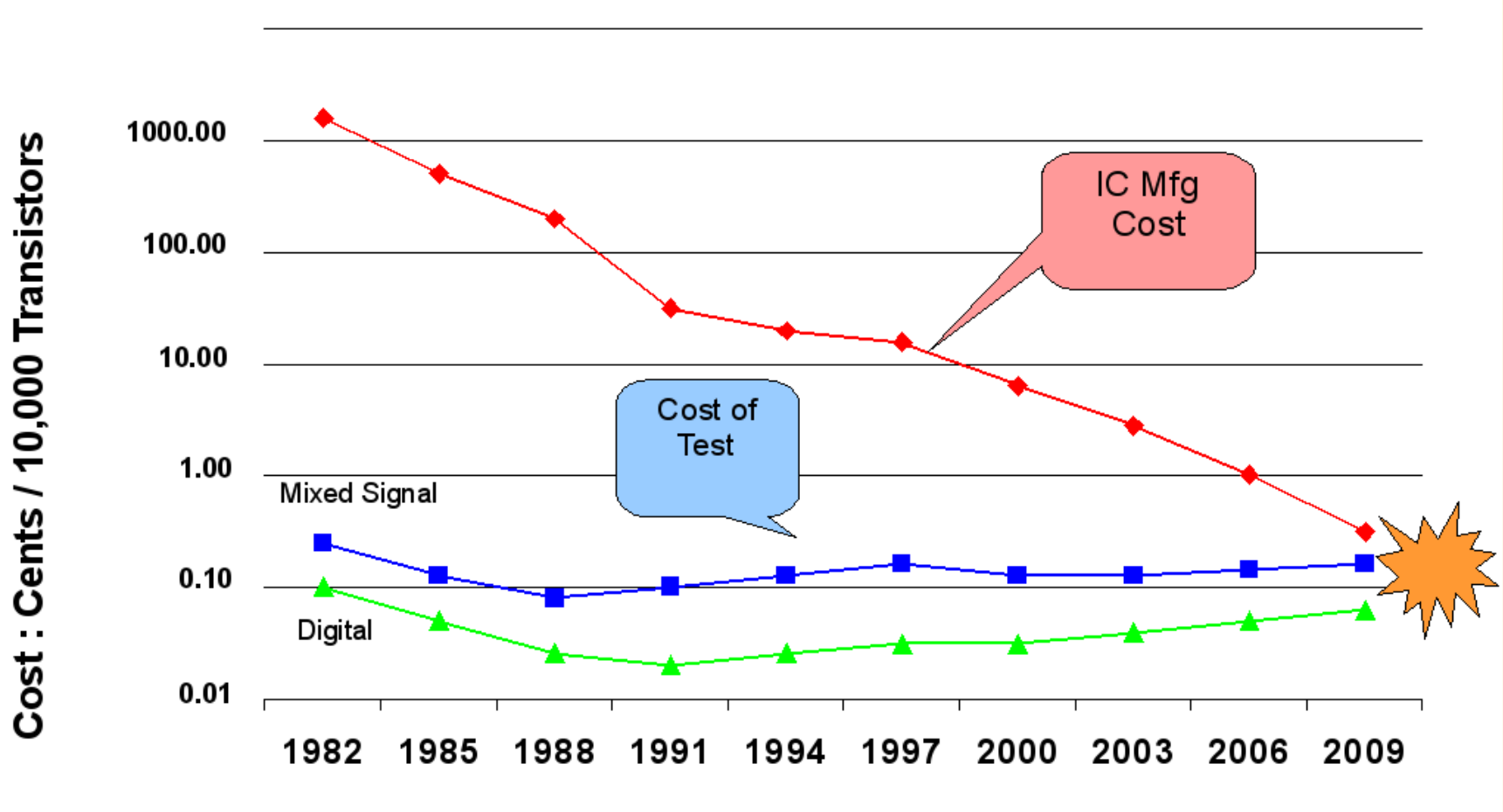
	X, Y			
CI	0 0	0 1	1 0	1 1
0	0,0	0,1	0,1	1,0
1	0,1	1,0	1,0	1,1



Called **C-tests**

Published procedures for generating C-tests for 1- and 2-D Iterative Logic Arrays

Manufacturing Test Costs



Why doesn't the cost of testing a transistor scale like the cost of manufacturing the transistor?

Conventional Test

Test problem simplified by structural, fault-based tests

The stuck-at fault model

- The model allows structural test generation, with a number of faults which is linear in the size of the circuit

Partitioning the circuit

- Partitioning the circuit (with scan latches for example), alleviates the test problem so that test generation does not have to deal with the entire circuit

Will this approach work for Deep SubMicron (DSM) circuits?

Failures May Not Be Hard: Example Resistive Opens

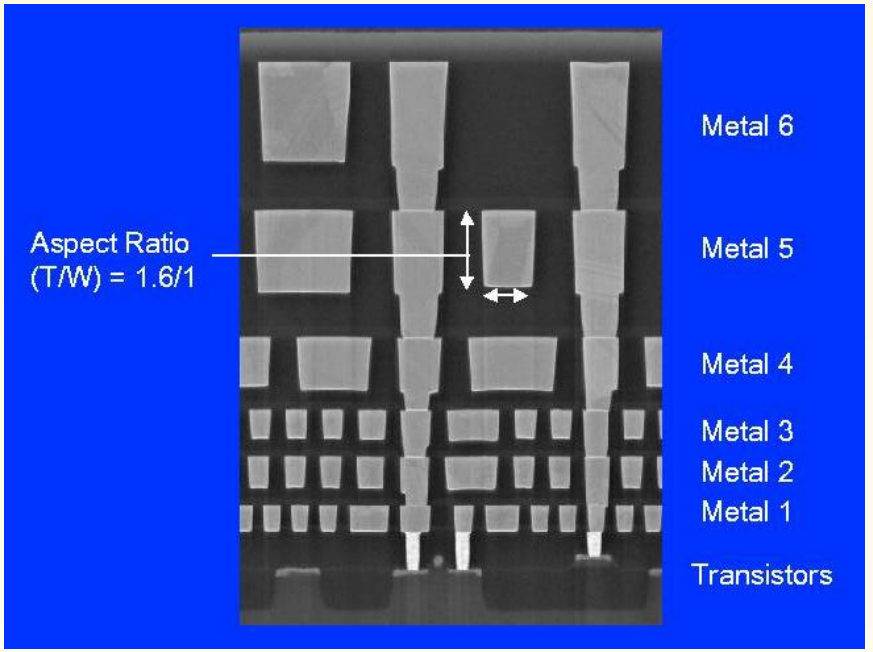
Experiments on real chips

- Some tests for logic-level "stuck-at" faults do not detect defects unless they are applied at speed

Interconnect opens are resistive (not complete breaks)

- Example: Cu interconnect with barrier materials
- Effect: delay faults

Increasing possibility of shorts and crosstalk

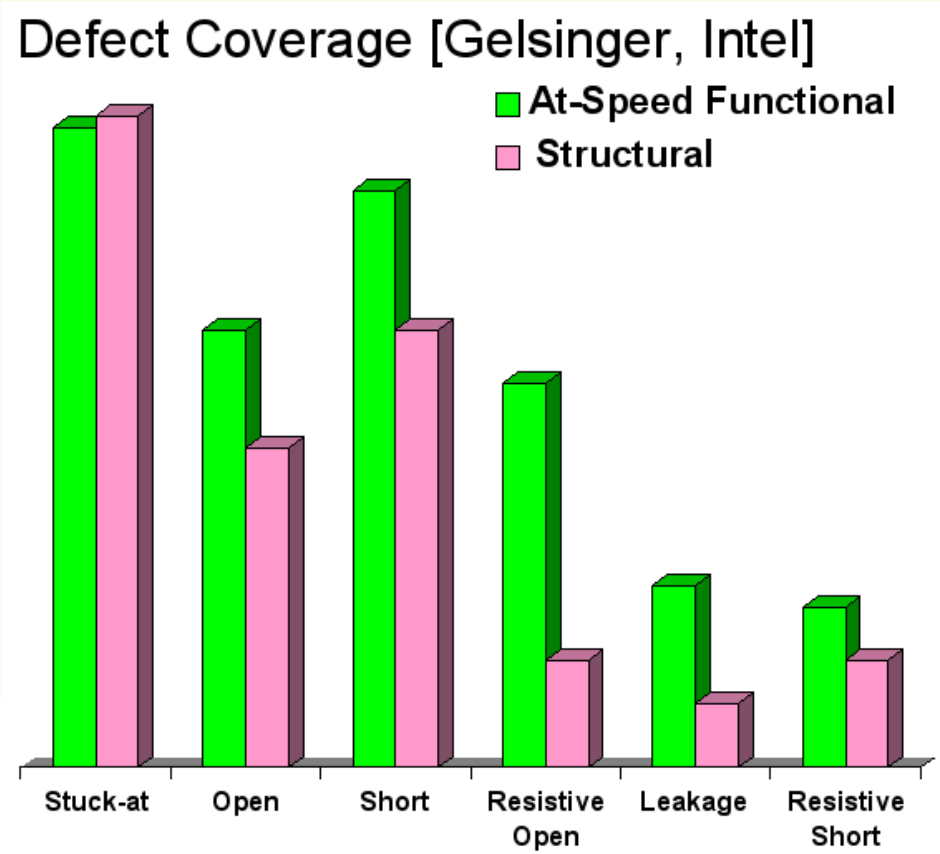


Breaks in Copper interconnect result in resistive opens because the barrier materials preventing interaction of Cu and SiO_2 will still be conductive

Effects on Chip?

Changes in delays of paths

Effects could be distributed across paths



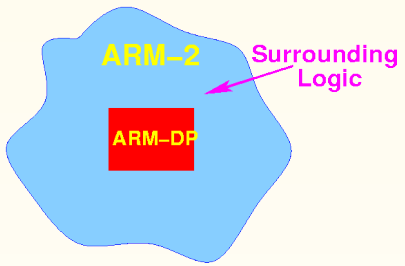
- At-speed functional tests are better for delay defects
- Solution: at-speed tests – run tests in the same environment as normal operation
- Problem: tester costs
- Need a technique which uses low-cost testers

Many companies now incorporating application-level tests in the manufacturing test flow

Reducing Test Complexity

- Generate tests at a higher level of abstraction?
  - Fault models at the Register Transfer Level
  - Tests may not detect DSM defects (e.g., delays)
- Exploit the design hierarchy
  - **Target one module at a time for test at the structural level** (can deal with opens, shorts, paths in module)
  - **Problems: accessing the module from the design boundary (complexity of the rest of the design)**
  - Can add logic to facilitate access to embedded modules ("design for testability"), **use "Slicing"**
- Experiment to determine extent of problem
  - Generate test for module by itself, then while embedded

	Gates	FFs	PIs	POs	Faults
ARM-2	16029	1270	63	67	99198
ARM-DP	8893	295	199	161	51824



Test Generation

Module by itself

- Sequential ATPG can easily deal with a module, example the ARM-DP by itself
- Results using commercial ATPG tool (on HPUX-715, 125 MHz processor)
  - Fault Coverage: 99.70%
  - ATPG Efficiency: 99.93%
  - Test generation time: 33.1 seconds
  - Test length: 822 cycles

Test generation for embedded module

- Sequential ATPG cannot deal with a module when it is embedded in even a moderately complex design
- Results on ARM-DP when it is embedded in ARM-2
  - Fault Coverage: 17.66%
  - ATPG Efficiency: 17.66%
  - Test generation time: 316,199 seconds

Tests for Small Delay Defects

Need to test **paths** in the circuit to detect small delay defects

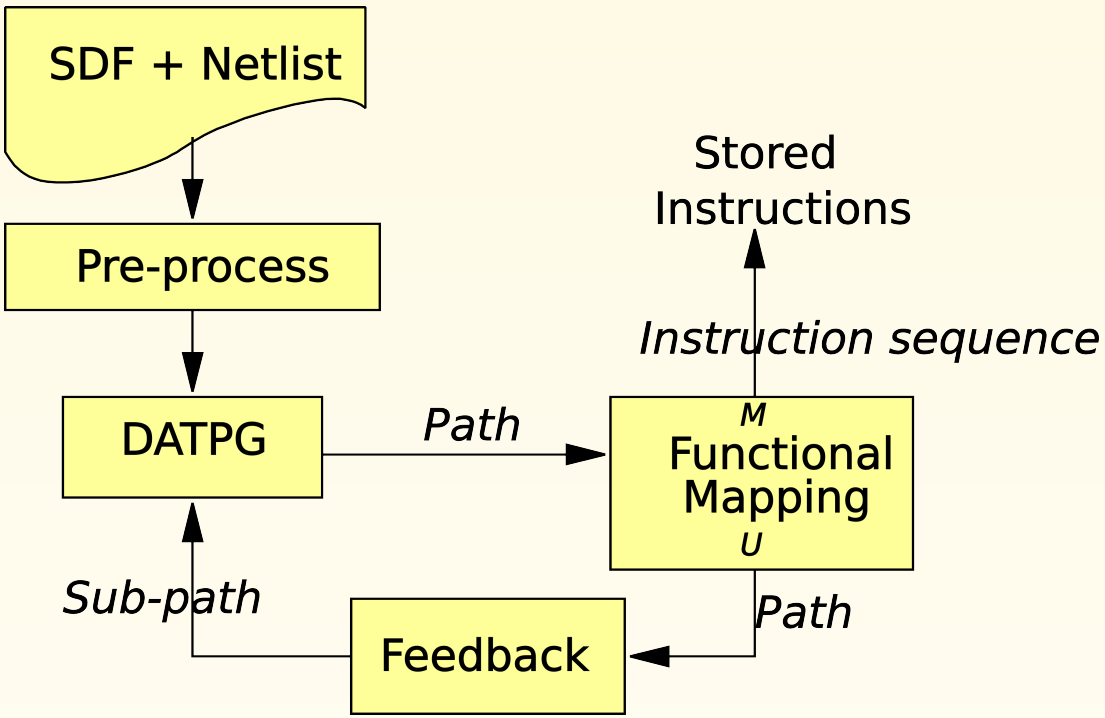
However, the number of paths in a circuit can be exponential in the number of nodes

Solution: test the longest path through every node

- This will detect the smallest possible delay increase which will cause the circuit to fail

Total number of tests is linear in the number of nodes

Automatic Generation of Instruction Sequences for Small Delay Defects



- Phase 1: all paths above a delay threshold
- Phase 2: longest paths through all nodes
- Delay-Based ATPG: generate "TRUE" paths above given delay threshold
- Functional mapping: **using verification engine**

- Feedback: heuristics to speed up search

Results on OR1200 processor

www.opencores.org, synthesized for 0.18μ TSMC process

Results for Phase 1 (paths $>$ 80% of clock)

No. of Paths	Drop	Functionally Testable	Functionally Redundant	Time out
27424	12	15118	12106	200

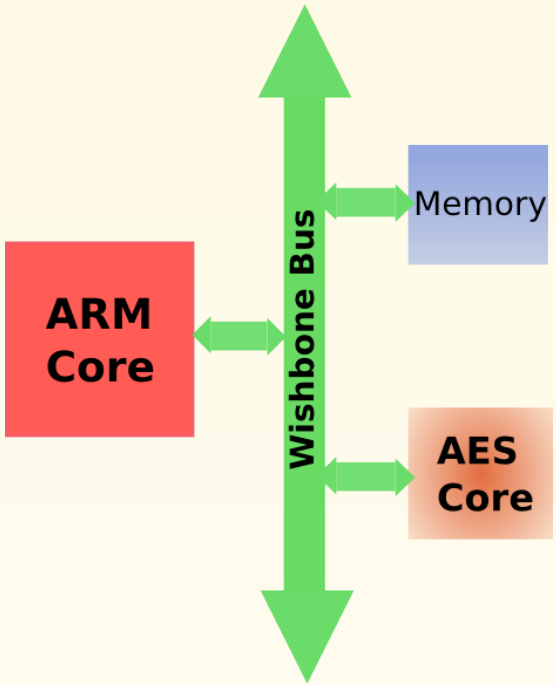
Results for Phase 2
N: % nodes with test for longest path through them

Module	Functionally Testable	Functionally Redundant	Rejected Sub-paths	N (%)
or1200_ctrl	1826	29191	68087	90.6
or1200_alu	1427	16985	2716	100
or1200_lsu	970	4077	3744	100
or1200_wbmux	1146	2285	2118	100

Test of SoC Cores using Embedded Processor

Wishbone and 128-bit AES designs from opencores.org

Validation vectors: random values encrypted/decrypted



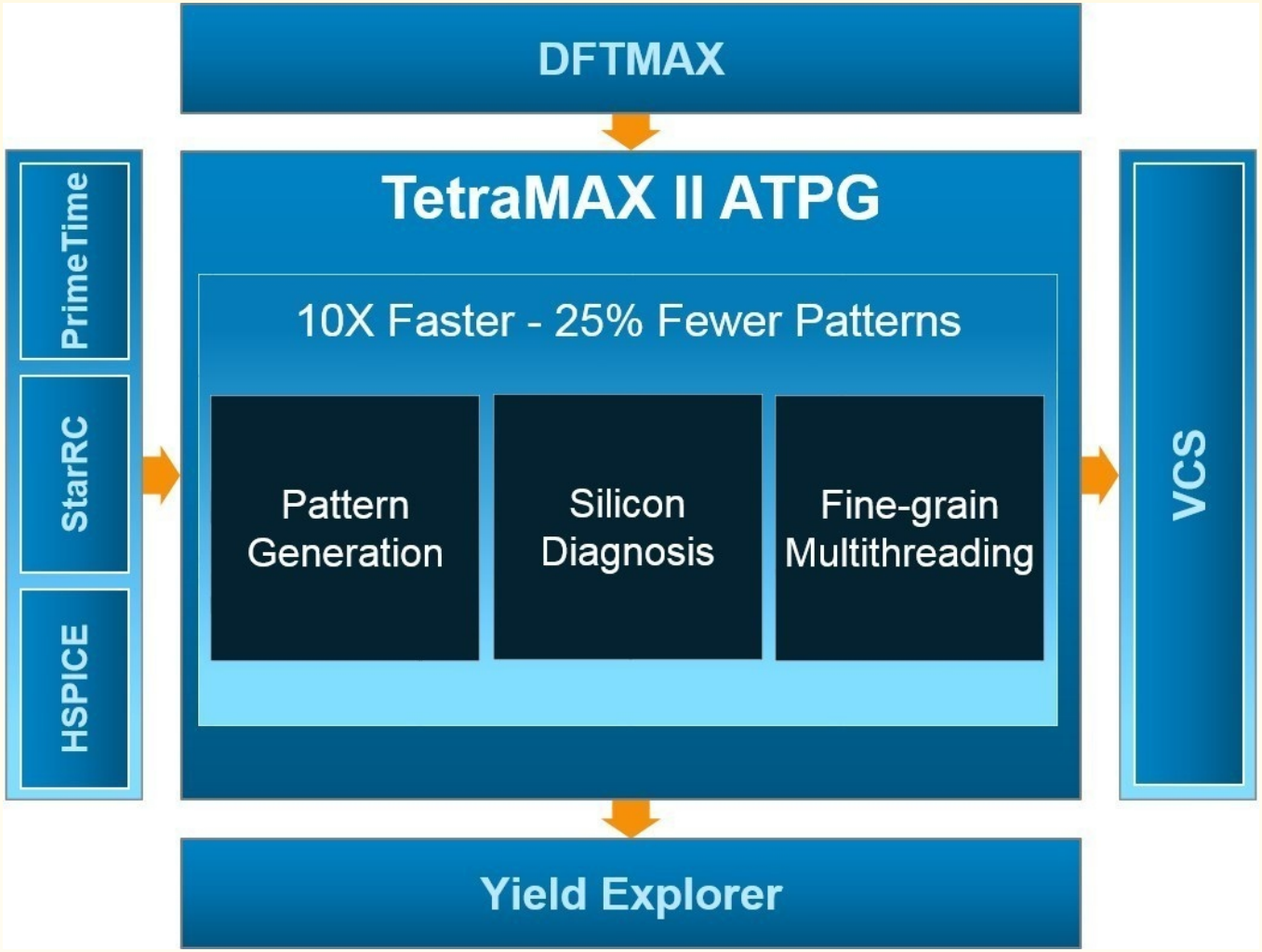
AES Core

Inputs	69
Outputs	33
Combinational primitives	9225
Sequential primitives	1119
Stuck-at faults	64070

Result of Mapping AES tests to ARM instructions (one case)

	Size (bytes)	Fault coverage(%)	Original Coverage(%)	No. of Cycles	Original Cycles
Test	9128	90.15	90.35	7816	7435

Commercial Test Generation Tool



Source: Synopsys, Inc.