Soft Errors Tutorial

Ilan Beer

IBM Haifa Research Lab

27 Oct. 2008

© 2008 IBM Corporation

Motivation

- As the semiconductors industry progresses deeply into the sub-micron technology, vulnerability of chips to soft errors is growing
- In high reliability systems, as well as in aviation and space, soft errors are already a major issue
- It is believed that in the near future soft errors will become a major issue for more systems
- A relatively hot subject in conferences in recent years
- On the other hand,
  “Fifteen years ago soft errors were the next threat.
  Fifteen years later they are still the next threat”
  Tim Slegel, Distinguished Engineer, IBM

© 2008 IBM Corporation

Outline

- Background – facts about soft errors
- Design solutions for dealing with soft errors
- Similarity and dissimilarity to functional verification

© 2008 IBM Corporation

What are Soft Errors?

- Neither functional problems nor production ones
  - Hence cannot be found during functional verification or production testing
- They occur during normal operation
- The value of a memory element or a logic gate is flipped
- The cause is transient and the error can be fixed by rewriting the correct value

© 2008 IBM Corporation

Two Main Sources

1: Alpha particles

- Emitted from radioactive impurities in silicon and package
- An alpha particle is equivalent to a Helium nucleus
- Fly through silicon, affecting nearby devices
- A few parts per billion are enough to cause problems
- Hit rate depends on the purity of materials used in the production process
- The duration of a strike is about 100 picoseconds

© 2008 IBM Corporation

Two Main Sources (cont.)

2: Cosmic rays

- Mainly neutrons
- Hit silicon and cause emission of alpha (and other) particles
- Only neutrons with enough energy penetrate the atmosphere and reach earth
  - ~15 particles/cm^2-hour at sea level
- Higher flux in higher elevations
  - 300 times more at 10 KM (commercial flight height)
  - Much more in satellites
- 3-5 meters of concrete can provide enough shield

- Also wire crosstalk, voltage surges, etc.

© 2008 IBM Corporation

A Strike Can Cause a Fault

- A strike by a charged particle can change the logic value of a device
- Not every strike flips the value
  - Depends on strike energy and on the charge stored in the device
  - Stored charge depends on transistor capacity (size) and operation voltage
    - Both are reduced as technology progresses
- The fault rates of specific device types (e.g., sram, dram and latches) are calculated using models based on empirical results
- Although the particle is positively charged, it can change a logic value to either '1' or '0'

© 2008 IBM Corporation

Is Small Transistor Size Good or Bad?

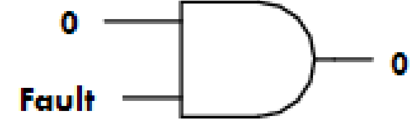
- Smaller size => less capacitance => less charge => increased vulnerability
- Smaller size => less area => less strikes

- These conflicting effects indeed cancel each other to some degree
- However, although feature size shrinks, the number of features grow, so the total system area does not shrink

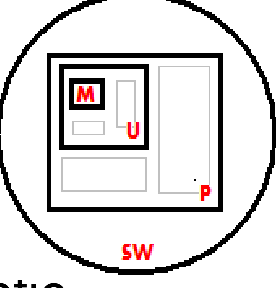
© 2008 IBM Corporation

Faults May Vanish

- Examples
  - Logical masking



  - A register is rewritten after the fault and before being used
  - A unit or thread is inactive at this moment
  - Faults that only affect timing (e.g., in branch prediction)
  - The SW application does not use this value until overwritten



- Vanishing depends on where we measure (macro/unit/processor/system/SW application)
  - All measurements except the application are pessimistic
- In Power6, more than 99% of the faults vanished (beam and simulation)
  - Less vanishing in datapath, more in control logic

© 2008 IBM Corporation

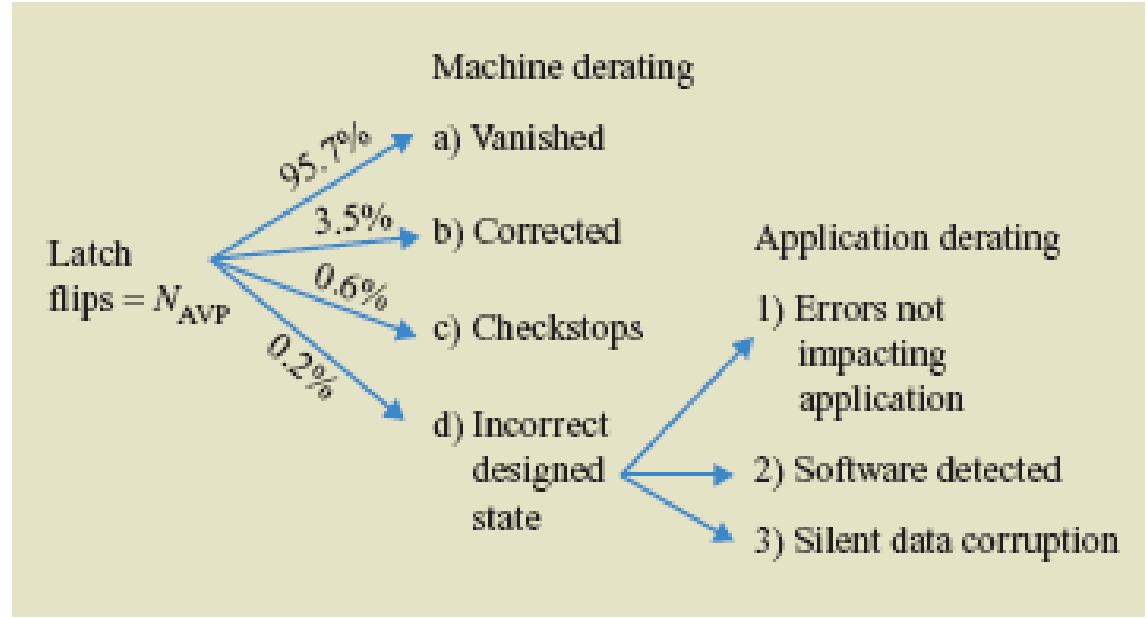
Terminology

- Fault: a bit flip
- Error: a fault that caused harm
  - Propagated to a point where it spoils the computation
  - An error can be either SDC or DUE (or hang)
- SDC: Silent Data Corruption
  - A fault that did not vanish and was not detected
  - Theoretically should be measured at the SW application level
  - We use this term to describe an error at the interface between HW and SW because we limit ourselves to hardware solutions
- DUE: Detected Uncorrectable Error
  - The fault was detected but cannot be corrected nor recovered
  - Usually stops execution; recovery is done by the layers above
- A fault that was corrected or recovered is not an error

© 2008 IBM Corporation

Fate of Faults in IBM Power6

Sanda et al., IBM Journal of Research and Development, Jan. 2008



- Interesting fact: without detection and correction, (d) only doubled!
- These numbers were measured by beam experiments and verified by simulation

© 2008 IBM Corporation

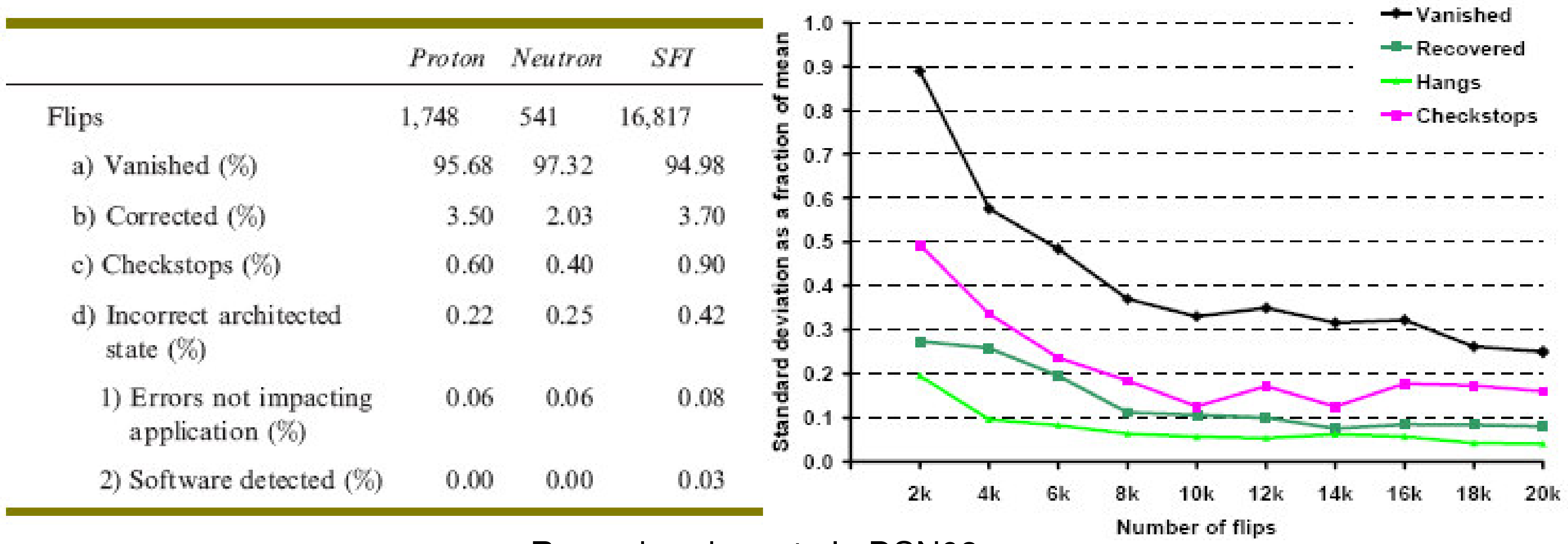
Beam Experiments

- Natural hit rate is too low for measuring derating
- Solution: Accelerated irradiation by protons and neutrons
  - Measuring faults: latches accessed through the scan chain (idle)
  - Measuring errors: by checkers during run
- Pros
  - The system runs at full speed
- Cons
  - A post production activity
    - Too late to affect the design
  - Hard to analyze specific events
  - No targeted injection

© 2008 IBM Corporation

Complementary SFI Experiments

- The number of faults in beam experiments was relatively low (< 2300)
- Statistical fault injection (SFI, described later) converged to similar results



	Proton	*Neutron*	*SFI*
Flips	1,748	541	16,817
a) Vanished (%)	95.68	97.32	94.98
b) Corrected (%)	3.50	2.03	3.70
c) Checkstops (%)	0.60	0.40	0.90
d) Incorrect architected state (%)	0.22	0.25	0.42
1) Errors not impacting application (%)	0.06	0.06	0.08
2) Software detected (%)	0.00	0.00	0.03

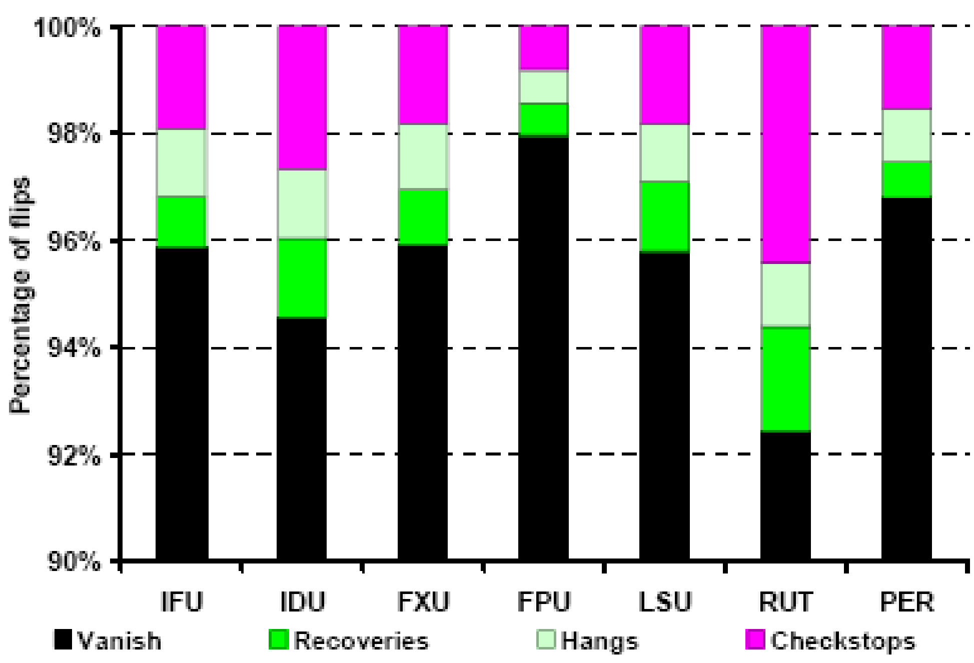
Ramachandran et al., DSN08

© 2008 IBM Corporation

Fate of Faults in Different Units

Ramachandran et al., DSN08

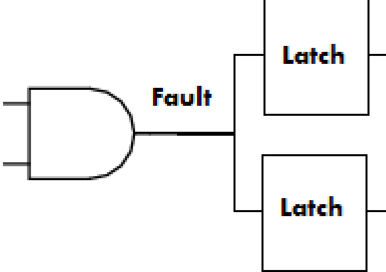
- In IBM Power6
- RUT is the recovery unit
- Note that the scale begins at 90%



© 2008 IBM Corporation

Mainly Single-Fault Model

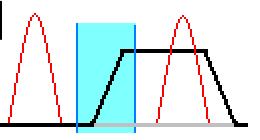
- A single fault at a time is assumed
  - Because fault frequency is relatively low
  - No problem if there are simultaneous faults in two devices with separate detection/correction/recovery logic
- However, a single particle strike may flip multiple bits
  - Special treatment in memory (see later)
  - Less frequent in latches – they are bigger and sparser
- Note: a single fault in a gate can spoil multiple latches
  - Luckily, we currently neglect faults in combinational logic



© 2008 IBM Corporation

Faults in Combinational Logic

- Have effect only if latched
- Three types of masking can prevent latching
  - Logical masking
  - Electric masking: the fault attenuates before being latched
  - Latch-window masking: fault arrives outside tsetup+thold



- As a result, combinational logic faults are often neglected
- This may change if clock frequency continues to grow
  - tsetup+thold will occupy more of the cycle time
  - Shorter combinational path from latch to latch – less attenuation and less masking
- Some combinational devices in the datapath are protected (e.g., integer arithmetic)

© 2008 IBM Corporation

Measurement Units

- Both faults and errors (faults that escaped) are usually measured by FIT (Faults In Time) units
  - The number of faults/errors in 1 billion hours (~114,000 years)
- FIT is convenient because it is additive
  - The FIT of a system is the sum of FITs of its components
  - Independent random variables with Poisson distribution
    - Assuming that particle strikes are independent events
- FIT is proportional to 1/MTTF (Mean Time To Failure)
  - MTTF is not additive
- A single bit has fault rate of about 1-10 mFIT
  - A 1 GB memory has fault rate of up to 10^9*8*0.01 = 8*10^7 FIT An error every 12.5 hours

© 2008 IBM Corporation

Problem Definition

- FIT in: the anticipated fault rate
- FIT budget: requirement for maximum number of errors
- Derating: the ratio between faults and errors
  - FIT in / FIT of errors
  - Intrinsic derating
    - Faults that “naturally” vanish as a result of the specific design structure
  - Explicit derating
    - Achieved by detection and correction
- Goal: Given FIT in and FIT budget
  - Achieve derating >= FIT in / FIT budget
    - Measure the intrinsic derating
    - Add explicit derating >= FIT in / FIT budget / intrinsic derating

© 2008 IBM Corporation

Design Solutions

© 2008 IBM Corporation

About Design Solutions

- Solutions include prevention, detection, correction, and recovery

- Solutions depend on goals. For example,
  - In some cases, stopping rather than recovery is enough
  - If no delay is allowed, backtracking-based recovery is not an option

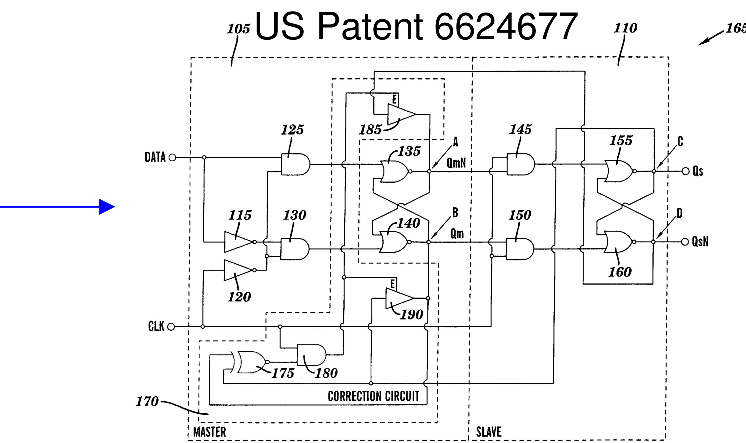
- Solutions depend on what we protect
  - Memory
  - Datapath
  - Control logic

© 2008 IBM Corporation

About Design Solutions (cont.)

Solutions can be provided at various levels

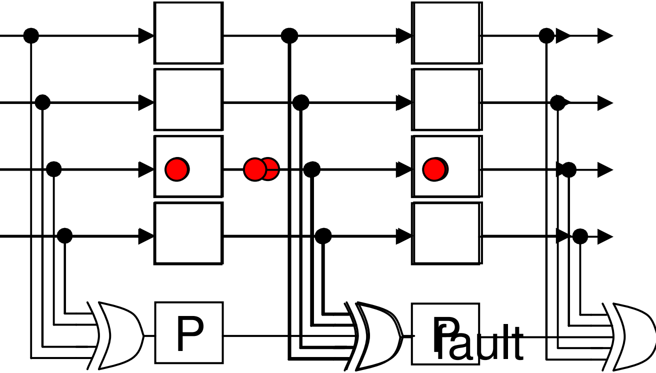
- Physical: add to the chip layers that collect charge (prevention)
- Circuit: hardening of selected latches (prevention)
  - Increase charge by increasing size or adding capacitors
  - Redundancy tricks
- Logic and micro-architecture (next slides)
- Software: calculate twice, sequentially or in parallel
- Combined: detection by hardware, recovery by software



© 2008 IBM Corporation

Logic and uArchitecture Level Solutions

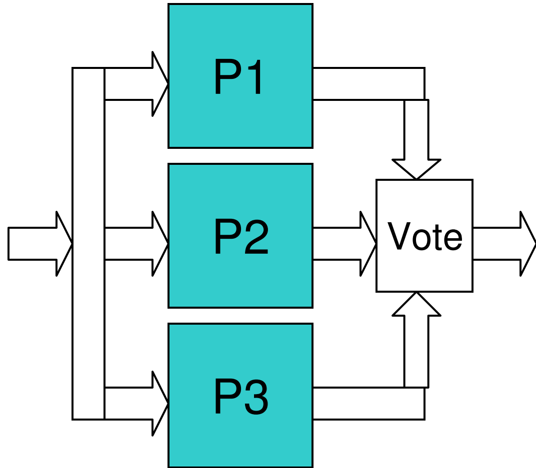
- Detection is based on redundancy
- We want a fault to bring the system to a non-reachable state
  - Bad parity is an example of an unreachable state
- If detection is not done close enough to the point of fault, the fault may escape
  - Downstream, it might manifest itself as a reachable state
  - This is why functional checkers are not good enough for soft error detection



© 2008 IBM Corporation

Very Robust Solutions

- Triple-Modular Redundancy (TMR)
  - Three replicas
  - Compare state/outputs and vote
  - A fault causes no delay
  - Used in satellites
  - Redundancy of more than 200%

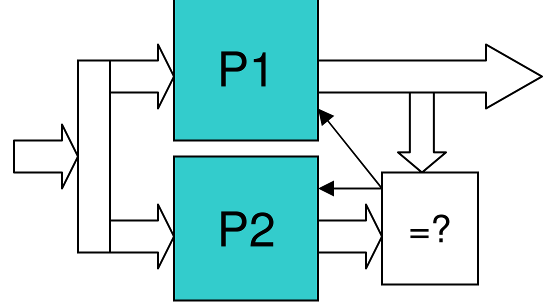


© 2008 IBM Corporation

Very Robust Solutions (cont.)

Duplicate and backtrack

- Two replicas
- The architected state is compared in every cycle
- Backtrack to a safe snapshot
- Used in older generations of IBM z-series processors
- Redundancy of more than 100%

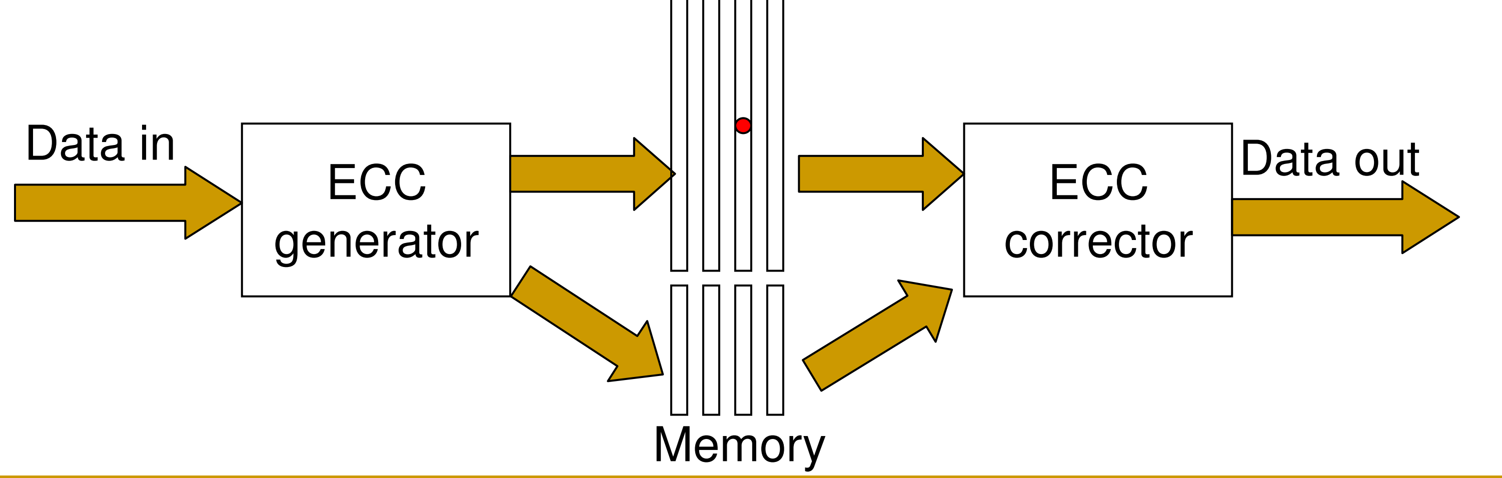


- Both solutions are not limited to a single fault
- Both are very expensive
- They can be applied to parts of the design

© 2008 IBM Corporation

Memory Solutions

- Useful property: A huge number of elements; at any time, only one is accessed through a read/write port
- Solution: Error correction code (ECC)
  - Increased Hamming distance between legal values
  - Usually can correct one fault and detect two
- Cost: A few additional bits per word + ECC generator + ECC corrector



© 2008 IBM Corporation

Memory Solutions (cont.)

- No need to backtrack; no delay
- Problem: One strike can change multiple adjacent bits
  - Solution: Interleave, so that bits of the same word are placed far from each other
- Problem: Faults may accumulate in the course of time
  - Solution: Periodic refresh
- A perfect solution for ~80% of the system area
- Consumes nothing from the FIT budget
- Considered as a solved problem; we will not deal with it further

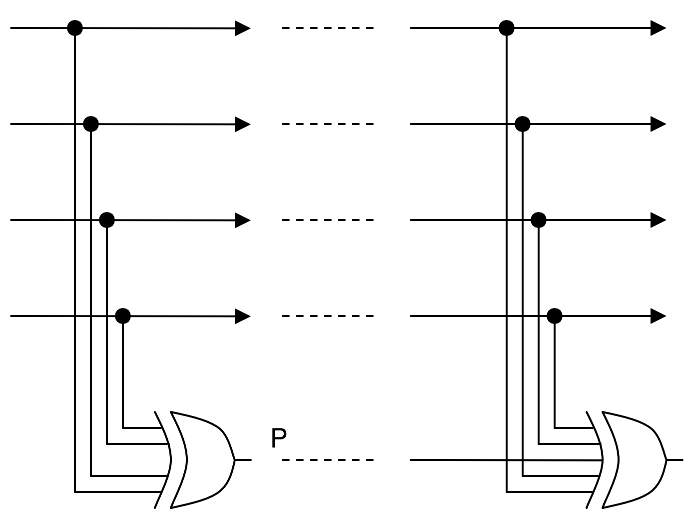
© 2008 IBM Corporation

Datapath Solutions

- Common elements
  - Wide buses that move data without modifying it, possibly through multiplexers
    - Parity-based detection (*)
    - CRC for packets (no extra lines, no need to detect in between)
    - ECC is not applicable
  - Arithmetic operations (integers)
    - Residue checking (calculation modulo a small integer)
- Satisfactory design methodology and implementation
  - As long as the elements fall into one of the above classes
- Usually has relatively low FIT budget
- Occupies ~80% of the non-memory area

© 2008 IBM Corporation

Parity



© 2008 IBM Corporation

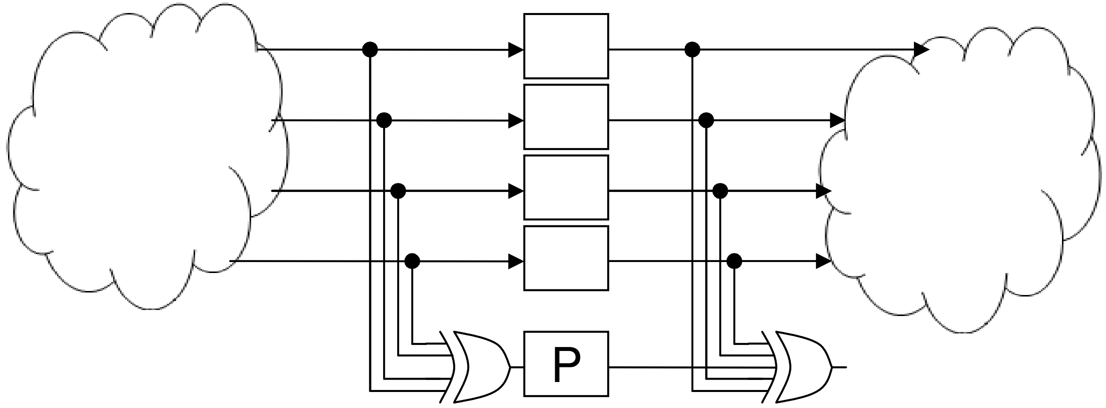
Control Logic Solutions

- Only latches are protected, not combinational logic
- The goal is to verify that the value read from the latch is the last value written into it
- A common solution is parity prediction and checking (*)
- Designers also use functional checkers but they don't know what is covered
- Have to find a compromise between detection quality and cost in area/power/timing/wiring
- Immature design methodology and implementation
- Usually has high FIT budget relative to its size

© 2008 IBM Corporation

Parity Prediction

- Calculate parity of latch inputs (!) on write
- Check parity of latch outputs in every cycle (not on read)



- Both timing and wiring complexity limit the number of protected latches per parity bit
- Natural grouping does not always exist

© 2008 IBM Corporation

Recovery (as in IBM Processors)

- Central recovery in a separate unit
- Takes snapshots of the architected state
- On fault detection
  - Backtracks to a valid snapshot
  - Reverts to the first instruction discarded
  - Resets the relevant units to valid states (e.g., flushes pipelines)
  - Stops if recovery is impossible
- Detection devices provide information for focused recovery and fault analysis
- Recovery imposes time limits on detection

© 2008 IBM Corporation

Over-Detection

- In Power6
  - With detection: 3.5% corrected, 0.6% DUE, 0.2% SDC
  - Without detection: only 0.4% SDC (0 detected; 0 DUE)
- Most of the detected faults could vanish
- Too many recoveries – not a big problem
- Too many DUEs – a big problem
- Reason for over-detection
  - We protect devices, not against bad states
  - Protecting against bad states would cost much more
- DUE minimization by deferring until instruction retires
  - No delay problems because there is no recovery

© 2008 IBM Corporation

Verification Aspects

© 2008 IBM Corporation

Verification Goals

- Main goal
  - Verify that the system meets the FIT budget requirements while retaining functional correctness
- Sub-goals
  - Measure system derating
  - Measure derating/vulnerability of sub-components
    - For optimal use of resource budgets
  - Verify that the detection logic indeed detects
  - Verify that the recovery mechanism indeed recovers
  - Verify that the soft error handling logic does not impair functional correctness

© 2008 IBM Corporation

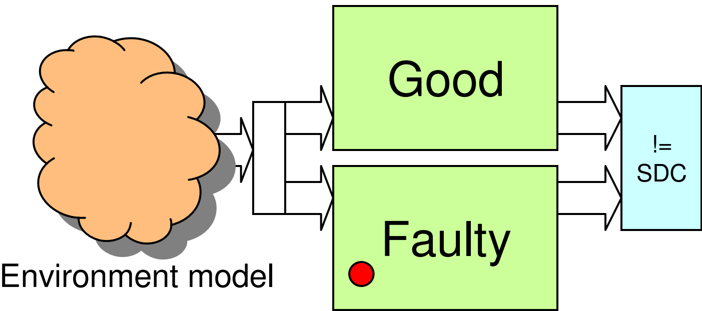
Similarity of Soft Error Verification to Functional Verification

- RTL of a design is given
- Valid stimuli (tests) are generated
  - Injected faults can be regarded as part of the test
- We verify that the design behaves correctly for all inputs, including the presence of faults
- Use of interface checkers or expected results
- Coverage is possibly measured
- The state space is huge
  - Fault injection increases the state space even further.
    For exhaustive verification, a fault should be injected to every latch in every reachable state

© 2008 IBM Corporation

A Simplistic Approach

- Use sequential equivalence
- In the faulty copy, allow one latch to change its value in one arbitrary cycle (use non-determinism)
- The two copies should be equivalent even in the presence of one fault



© 2008 IBM Corporation

Why Simplistic?

- Soft error requirements are statistical
  - Not every fault needs to be detected or recovered (FIT budget)
  - Only typical workloads are interesting, not corner cases
- After recovery, the faulty copy no longer behaves as the copy without a fault
  - It is reset to a valid state, probably not a state of the design without a fault (e.g., the pipeline is flushed)
- We cannot neglect the state explosion problem
  - The comparison should be applied to the entire design in order not to be pessimistic

© 2008 IBM Corporation

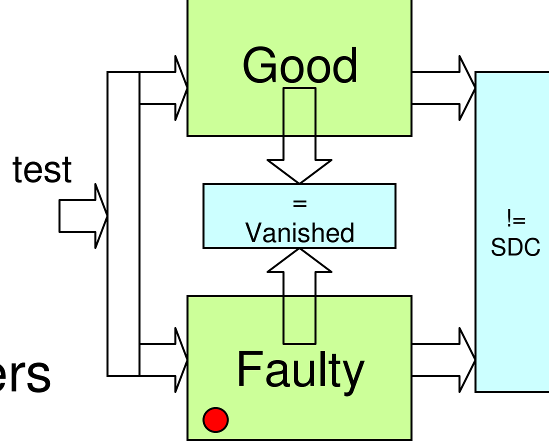
A Practical Approach: Statistical Fault Injection (SFI)

- Inject faults one at a time
- Simulate enough cycles to allow one of these
  - Vanishing
  - Detection by the soft error checkers + recovery or stopping
  - Detection by "interface" checkers (SDC)
- After enough runs, measure output FIT
  - The average "latch intrinsic FIT" of latch injections that resulted in SDC, multiplied by the total number of latches
- Use this procedure to report areas that are more vulnerable
  - Same calculations as above, but for specific parts
  - This can help designers invest effort and resources efficiently

© 2008 IBM Corporation

Identifying Escapes and Vanished Faults

- Identifying vanished faults is important
  - Efficiency: If a fault vanished, a new fault can be injected
  - Correctness: When simulation ends, we want to know if the fault is latent in the internal state, in which case it is a potential SDC
- Solution
  - Lock-step simulation of good and faulty machine
  - Compare internal state each cycle (or every k cycles)
- A by product
  - By comparing the architected state every cycle we can find SDCs
  - More conservative than interface checkers
  - Sometimes over-pessimistic
- The solution works only as long as there is lock-step
  - E.g., not after recovery



© 2008 IBM Corporation

Some Problems to be Solved

- Intelligent sampling of the injection space
- Identifying protected latches to avoid unnecessary injections (alternatively, verifying designers' annotations)
- Knowing how many runs are needed to converge to statistically meaningful results?
- Optimizing simulation to allow more injections
- Writing simulation checkers that identify all escapes
- Selecting typical workloads
- Identifying vulnerable areas in the design
- Verifying the recovery process (both recovery unit and pervasive branches)
- Finding a good compromise between FIT budget and resource budgets

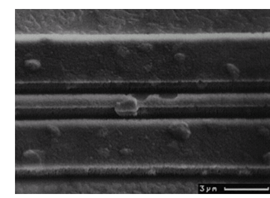
© 2008 IBM Corporation

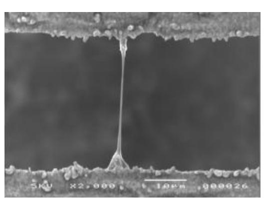
Longer Term Goal: Automatic Generation of the Detection Logic

- Currently
  - Intensive effort for the designer
  - Inserted logic should be verified
  - Logic is not necessarily efficient
- Automatically generated logic
  - Will be correct by construction
  - Might be more efficient
  - Will save designer effort
  - Will shorten time to market
- Naïve generation is already possible
- The challenge is to close the loop of generation-measurement in order to find the best compromise between FIT budget and area/power/timing budgets

© 2008 IBM Corporation

Hard Errors



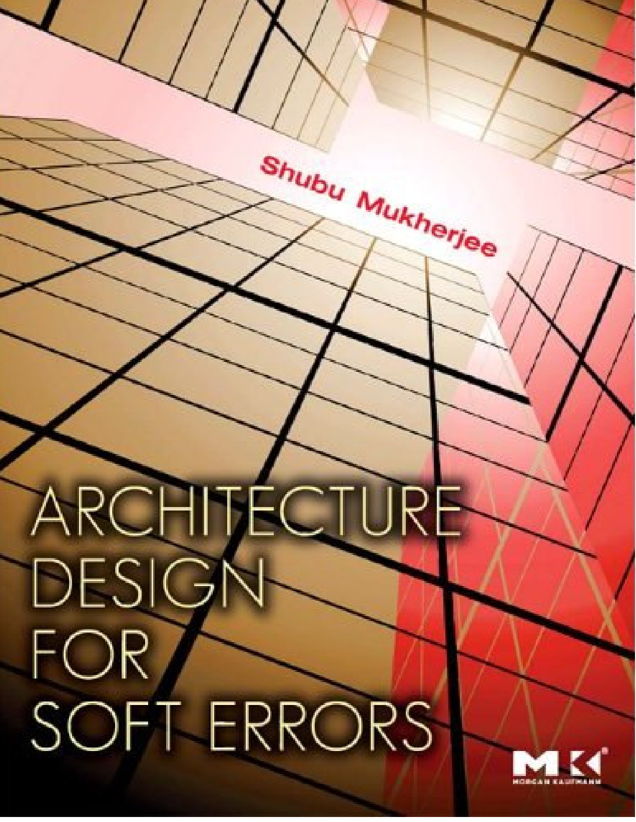


- If we reduce the soft error rate to very low levels, we may face the next barrier: hard errors
  - Wire wear-out by electromigration
  - Gate oxide wear-out
- Intrinsic vulnerability is very low but there is no derating
- May result in SDC or DUE; no vanishing; no recovery
- In memory, correction by ECC (one fault)
- Can be detected with soft error detection logic
- However, we protect only latches against soft errors
- Hard errors in combinational logic cannot be neglected
- Solutions
  - Protect the combinational logic in hardware – expensive!
  - BIST + SW

© 2008 IBM Corporation

Recommended Book (new)

- Architecture Design for Soft Errors
  by Shubu Mukherjee (Intel)



Thank you

© 2008 IBM Corporation