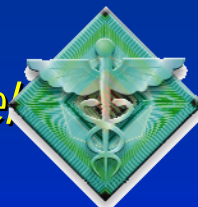


Chip, Heal Thyself



Todd Austin

*Advanced Computer Architecture Lab
University of Michigan*

With

Prof. Valeria Bertacco, Prof. Scott Mahlke
Kypros Constantinides, Smitha Shyam
Mojtaba Mehrara, Mona Attariyan, Sujay Phadke



Chip, Heal Thyself

Todd Austin, October 2007

The BulletProof Project

- Goal
 - Novel design methodologies for the creation of silicon defect tolerant architectures featuring unprecedented low cost*
- Anticipated results, develop architectures that
 - *Detect* and *diagnose* any defects that manifest
 - *Recover* any system computation impaired by defects
 - *Repair* hardware to allow continued operation
- Previous work: defect-tolerant CMP router
- This work: defect-tolerant processing element



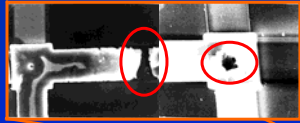
Chip, Heal Thyself

Todd Austin, October 2007

Reliability Challenges

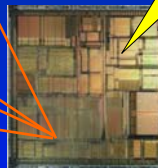
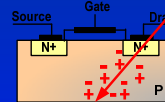
Silicon Defects

(Manufacturing defects and device wear-out)



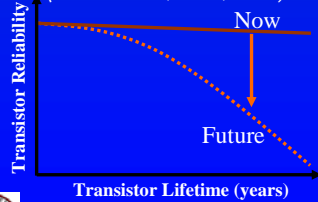
Transient Faults due to Cosmic Rays & Alpha Particles

(Increase exponentially with number of devices on chip)



Transistor Wear-out

(due to TDDB, NBTI, etc...)



Manufacturing Defects That Escape Testing

(Inefficient Burn-in Testing)



Chip, Heal Thyself

Todd Austin, October 2007

The (Bumpy) Road Ahead for Silicon



The lifetime of silicon will be determined by how cheaply and effectively we can make the substrate reliable.



Chip, Heal Thyself

Todd Austin, October 2007

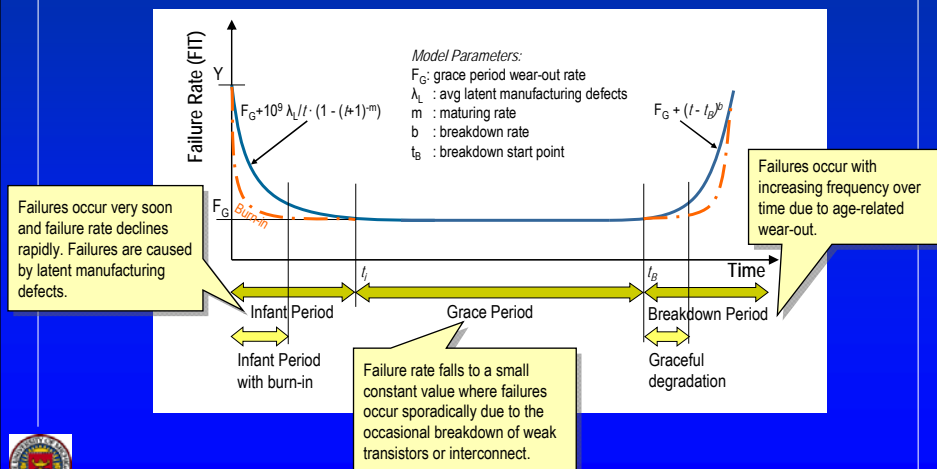
Challenge #1:

Predicting the Future (Fault Modeling)



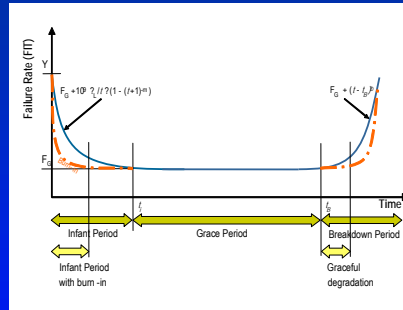
The Bathtub Curve: A generic model for hard failures

- A high-level architect-friendly model of silicon defects, based on the time-tested bathtub curve



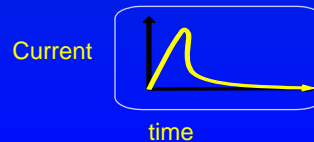
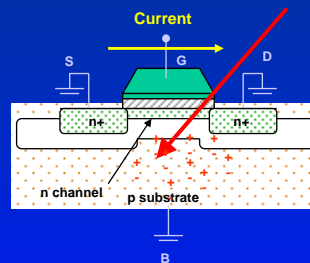
Futures Scenarios We Can Address

- *Future #1:* The failure rate during the grace period begins to rise
 - Low to *moderate*
- *Future #2:* The infant mortality period extends into the lifetime of the product
- Many scenarios will not have on-chip solutions
- Many “doomsday” scenarios exist



A Statistical Model for Transient Faults

- Pulse-based model for transient faults
- Faults injected into combinational logic are classified by duration
 - 20%, 40%, 60%, 80% and 100% of design’s clock period
- Faults injected directly into sequential elements flip their value
- Inter-arrival times for each fault are derived from published data



Challenge #2:

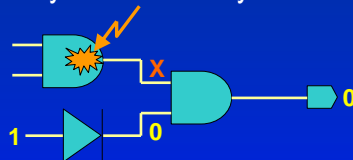
High-Fidelity Resiliency Analysis



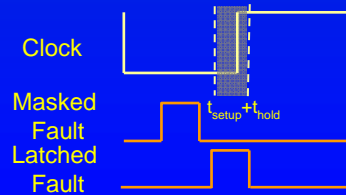
Why is Analysis Hard?

Consider soft error masking...

- *Logic Masking*: the fault gets blocked by a following gate whose output is completely determined by its other inputs

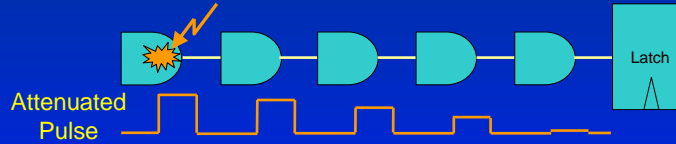


- *Timing Masking*: the fault affects the input of a latch only in the period of time that the latch is not sensitive to its input



Soft Error Masking

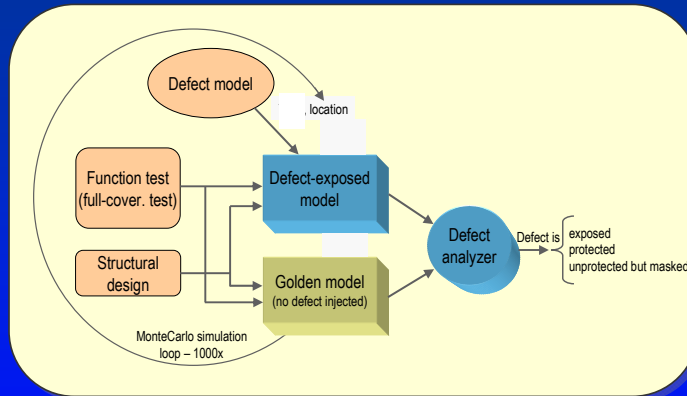
- *Electrical Masking*: the fault's pulse is attenuated by subsequent logic gates due to electrical properties, and does not affect any latch's input



- *Microarchitectural Masking*: the fault alters a value of at least one flip-flop, but the incorrect values get overwritten without being used in any computation affecting the design's output
- *Software Masking*: the fault propagates to the design's output but is subsequently masked by software without affecting the application's correct execution



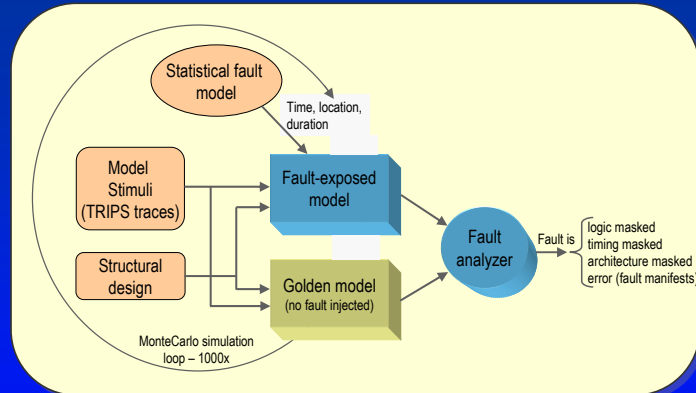
Defect Coverage Analysis



- Asynchronous defect injection at gate level
- Defect coverage analyzed using functional verification tests
- Monte Carlo simulation generates high-confidence estimates



Soft Error Coverage Analysis



- Asynchronous fault injection at gate level with varied duration
- Infrastructure model all possible ways a fault can be masked
- Monte Carlo simulation generates high-confidence estimates



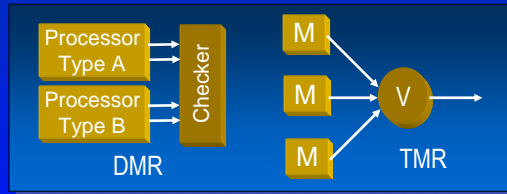
Challenge #3:

Low-Cost High-Coverage Defect-Resilient Architectures



Traditional Defect-Tolerant Techniques

- Used at high-end safety-critical systems
 - Dual Modular Redundancy (look for differences)
 - Triple Modular Redundancy (voting scheme)



- Utilize redundant hardware to validate computation
 - Result in very high area cost
 - Very costly to employ for consumer systems (100-200% overhead)



Novel Reliable Design Strategy:

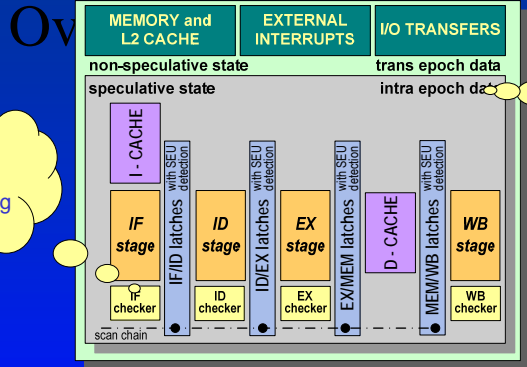
Continuous Online Testing

+

Microarchitectural Checkpointing



BulletProof Pipelines:



On-line distributed testing using checkers

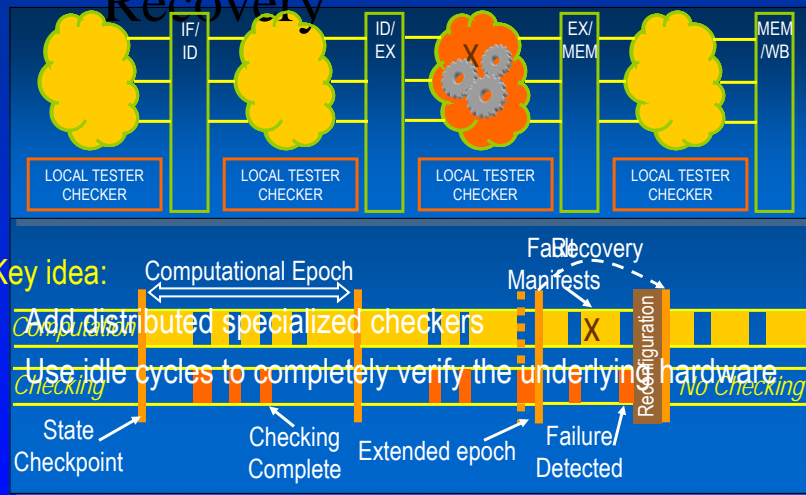
Speculative state during computational epochs

If a component is defective disable it, rollback state, and continue operation in degraded performance mode using remaining resources

Key Insight: For inexpensive defect protection, don't check computation, Instead... Validate H/W is free of defects, otherwise, rollback and recover



Distributed Testing and Recovery

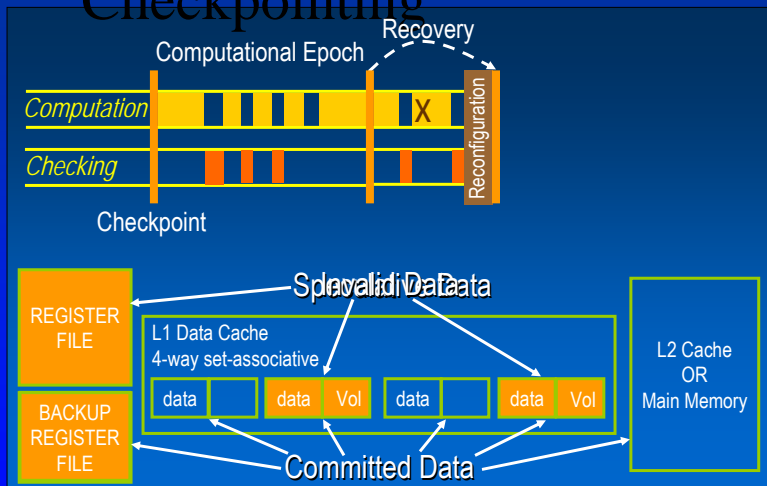


Micro-Architectural Checkpointing

- A mechanism to create coarse-grained epochs of execution
 - Augment each cache block with a *Volatile* bit to indicate speculative state
 - Backup Register File – Single-port SRAM (much simpler and smaller than regular RF)



Micro-Architectural Checkpointing



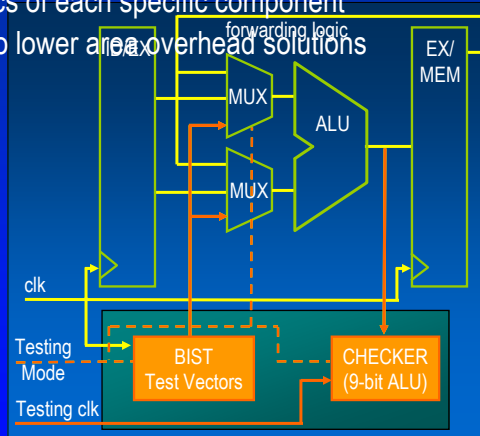
How Long Can Epochs Be?

- A computational epoch must end when:
 - All cache blocks in a set hold volatile data
 - An I/O operation is requested from the OS
- Avg epoch size is in the order of 10,000+ of instructions
- To provide longer computational epochs
 - Add a small fully associative victim cache for speculative data
 - OS classifies I/O requests:
 - *High Priority*: Terminate the epoch
 - *Low Priority*: Hold in a queue – Served at the end of the epoch
 - *Speculative*: Execute speculatively before the end of the epoch
- Avg epoch size in the order of 100,000+ of instructions



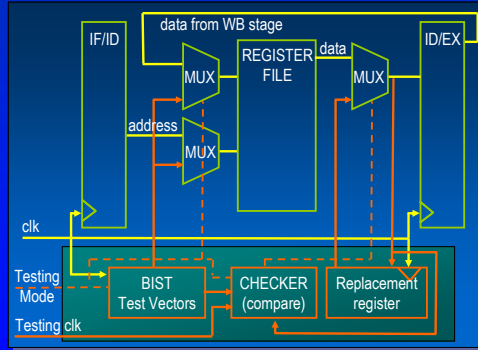
Specialized Distributed Online Testing/Checking

- Special checker for the ALU/forwarding logic component
- Exploits special characteristics of each specific component
- Specialized testing results to lower area overhead solutions
- Built-In Self-Test vectors are sent to ALU
- Output verified by a 9-bit mini-ALU checker
- 4 cycles to fully verify the output of the ALU



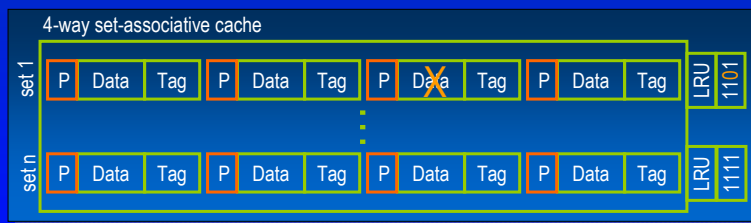
On-line Testing Techniques - Register File Checker

- Four phase split-transaction process:
 - a) Redirect the register under test to the replacement register
 - b) Write a random value (generated by a LFSR) to the register under test
 - c) Read back the written value and compare to the original
 - d) Restore the value of the register under test from the replacement register
- Test address decoders by using different read/write address decoders
- Execute a phase whenever there is an idle read/write port
- Test all 32 registers in 128 clock cycles



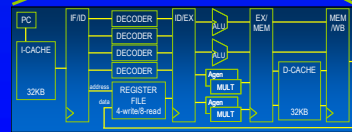
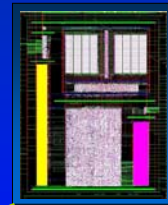
On-line Testing Techniques – Cache Line Checker

- A single parity bit is associated with each cache block (data+tag)
- *Write*: generated and store parity - *Read*: compute and verify parity
- Defective cache lines are disabled by using bit-masks in the LRU logic
- Periodically reset bit-masks to avoid soft-errors being represented as silicon defects



Experimental Methodology - Baseline Architecture

- Baseline Architecture:
 - 5-stage 4-wide VLIW architecture, 32KB I-Cache, 32KB D-Cache
 - Embedded designs: Need high reliability with high cost sensitivity
- Circuit-Level Evaluation:
 - Prototype with a physical layout (TSMC 0.18um)
 - Accurate area overhead estimations
 - Accurate fault coverage area estimations
- Architecture-Level Evaluation:
 - Trimaran toolset & Dinero IV cache simulator
 - Average computational epoch size
 - Performance while in graceful degradation
- Benchmarks
 - SPECINT2000, MediaBench, MiBench



On-line Testing Techniques

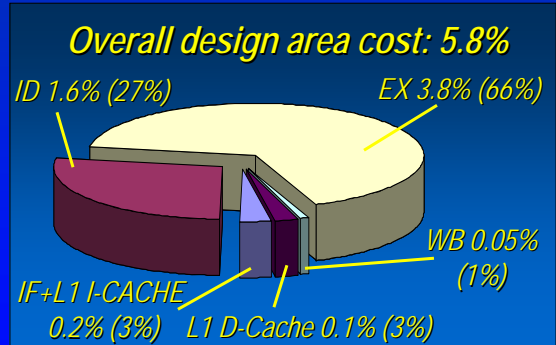
Component-specific Testers/Checkers

- Decoders
 - 63 test vectors – Majority checker
- ALU/Address Generation Unit
 - 20 test vectors – 9-bit mini-ALU checker
- Multiplier
 - 55 test vectors – Residue checker
- Register File
 - Replacement register – 4-phase checking process
- Caches
 - Parity bit for each cache block (data+tag)



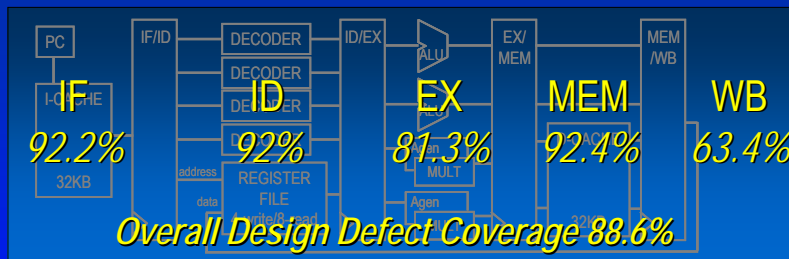
Area Overhead Summary

- Overhead calculated using a physical-level prototype
 - Place & routed synthesized Verilog description of the design
- EX stage dominates area cost contribution
 - Functional unit checkers
 - Test vectors
- Next is ID stage
 - Decoders checkers
 - Test vectors
 - Backup register file
- The rest is:
 - Cache parity bits
 - Cache *Volatile* bits
 - Testing logic



Design Defect Coverage

- *Defect Coverage*: total area of the design in which a defect can be detected and corrected

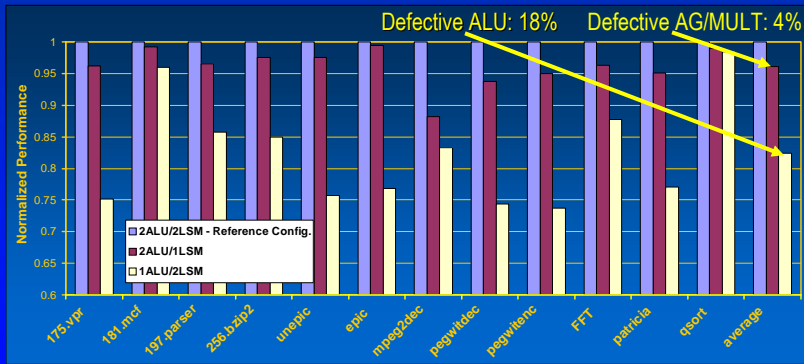


- The unprotected area of the design mainly consists:
 - Resources that do not exhibit inherent redundancy
 - Interconnect (i.e., wire-buses connecting the components)
 - Control logic



Performance Under Degraded Mode Execution

- The system recovers from a defect by disabling the defective component
- Losing an ALU results in average 18% performance degradation
- Losing an Addr. Gen/MULT unit results in average 4% perf. degradation

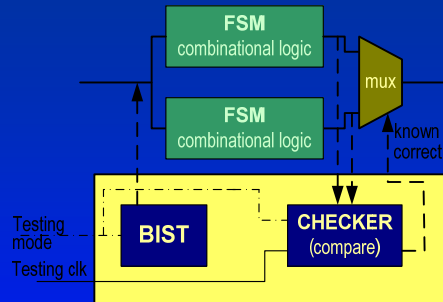


Challenge #3.5:

*Lower-Cost Higher-Coverage
Fault-Resilient Architectures*



Control Logic Protection



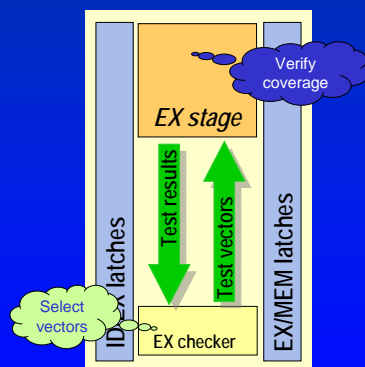
- DMR approach
- BIST techniques used to localize the fault
- Checker is smaller than a third copy of the FSM
- Lower cost than traditional TMR



Reflexive Self-Testing

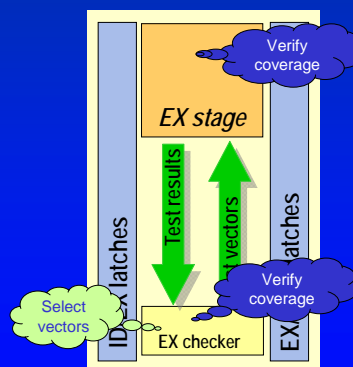
Traditional Testing

- Checker generates tests to fully cover design block



Reflexive Self-testing

- Checker generates tests to fully cover design block AND CHECKER



- Relies on single-defect model



Hot Off the Press

- Overall area overhead: 14.1%
- Overall design coverage : 95.2%
- And the quest for lower-cost higher-coverage continues...
 - To appear in MICRO 2007: S/W based continuous testing
 - Utilizes ACE instruction extension that probes internal state
 - Significant improvement to control logic coverage
 - Tests run continuously with application/OS software
 - Same checkpoint/recovery mechanisms utilized
- Overall area overhead: 5.1%
- Overall design coverage: 99.2%
- Overall performance impact: < 5%

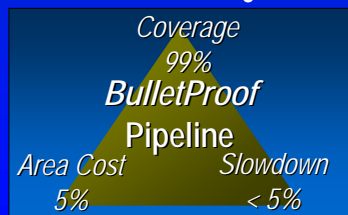


Chip, Heal Thyself

Todd Austin, October 2007

Conclusions

- *BulletProof* pipeline takes a new direction in fault-tolerant design
 - First ultra-low cost defect protection mechanism for microprocessor pipelines
 - Propose the combination of on-line distributed testing with microarchitectural checkpointing for low-cost defect protection
- Implemented a physical-level prototype of the technique
 - *Area cost*: ~ 5%
 - *Coverage*: 99% coverage for first defect
 - *Slowdown*: Cost of continuous testing limited to ~5%



Chip, Heal Thyself

Todd Austin, October 2007

Questions



Future Work

- Adding support for low-cost transient fault protection
- Increasing overall design coverage (control, etc.)
- Leveraging existing facilities in chip-multiprocessors (many cores, global checkpointing)
- Migrate detection and diagnosis mechanisms to software (trade-off silicon overheads for runtime)



Reliability Challenges with CMOS Scaling

Growing concerns that designers will face major reliability challenges as CMOS scales in the nanometer regime

Device Wear-out:

- Metal electro-migration (weak interconnects, fractures, shorts, voids)
- Hot carrier degradation (weak transistors)
- Time-Dependent Dielectric Breakdown (transistor failures)



Average Product Lifetime ↓



Work in Progress

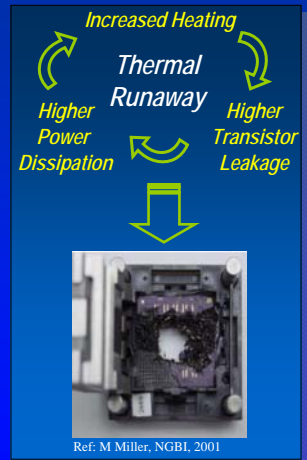
- Improve design coverage of the technique
 - Low-cost defect-tolerance techniques for control logic and interconnect
- Reduce the area overhead of the testing infrastructure
- Examine the applicability of our technique to desktop and server microprocessors
- Add support for soft error protection
- Utilize the testing infrastructure for other value-added capabilities



Reliability Challenges with CMOS Scaling

Manufacturing defects that escape testing:

- Device scaling increases device infant mortality rates
- Burn-in testing: Devices are stressed with high voltage and temperature in order to screen out weak parts
- With technology scaling burn-in testing becomes less effective because of thermal run-away effects

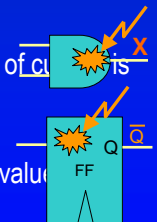


Silicon Failures that Escape Testing



Why is Resiliency Analysis Hard? One Example...

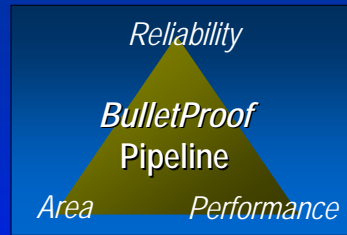
- *Soft errors*, also called *transient faults* and *single-event upsets*(SEU)
 - Processor execution errors caused by high-energy neutrons resulting from cosmic radiation and alpha particles radiation
 - Appears to be a reliability threat for future technology processors
- When a particle strikes a circuit element a small amount of charge is deposited
 - *Combinational logic node*: a very short duration pulse of current is formed at the circuit node
 - *State holding element (FF/SRAM cell)*: flip the stored value



Our Approach: BulletProof Pipeline

Goals:

- Area Cost
 - Ultra low-cost solution
- Provided Reliability
 - Support recovery from first defect
- Performance
 - After recovery the system still operates in degraded performance mode



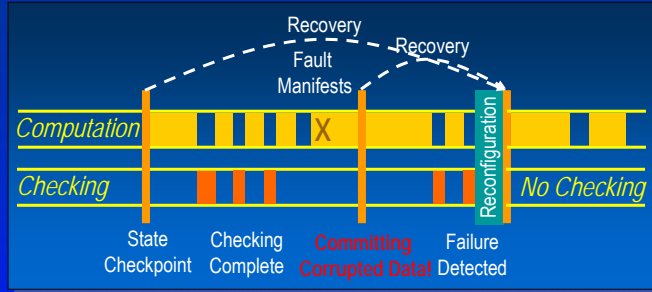
BulletProof Pipeline Overview

- Employ microarchitectural checkpointing to provide a *computational epoch*
- *Computational Epoch*: a protected period of computation over which the underlying hardware is checked
- Use on-line distributed testing techniques to verify the hardware is free of defects, on idle cycles
- If a component is defective disable it, rollback state, and continue operation under a degraded performance mode on remaining resources

For inexpensive defect protection, don't check computation, Instead... Validate H/W is free of defects, otherwise, rollback and recover.



Two-Phase Commit



- Support a sliding rollback window (keep last 2 checkpoints)
- Recover by restoring the oldest checkpoint
 - Maintain two epochs in local cache hierarchy by having two per cache block
 - Add an extra backup register file for the architectural state

Volatile bits



On-line Testing Techniques

On-line Distributed Testing:

- Specialized tester-checker for each major component in the pipeline
- Exploit special characteristics of each specific component
- Specialized testing results to lower area overhead solutions

Decoder Checker:

- Exercise test vectors on idle cycles
- Detect failures using a majority checker
- 63 test vectors to cover stack-at-0/1 faults
- Test all 4 decoders in 126 clock cycles

