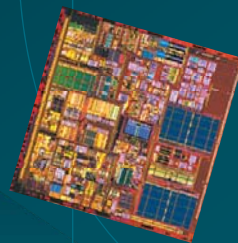
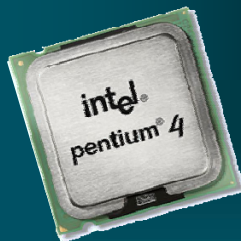


On the Rules of Robust Design (and Why You Should Break Them)

Todd Austin

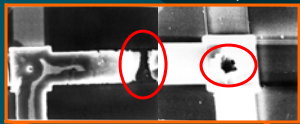
Advanced Computer Architecture Lab
University of Michigan
austin@umich.edu

A long time ago, in a not so far away place...

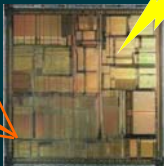


Key Reliability Threats

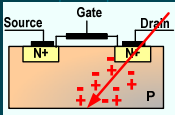
Silicon Defects
(Manufacturing defects and device wear-out)



H/W and S/W Design Errors
(Bugs are expensive and expose security holes)

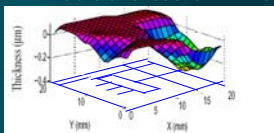


Transient Faults due to Cosmic Rays & Alpha Particles
(Increase exponentially with number of devices on chip)




Parametric Variability
(Uncertainty in device and environment)

Intra-die variations in ILD thickness



Manufacturing Defects
(and Ineffective Burn-in Testing)

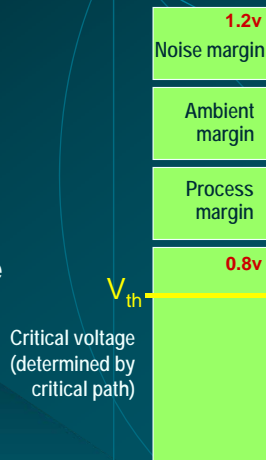


Thermal Runaway

Higher Power Dissipation → Increased Heating → Higher Transistor Leakage → Higher Power Dissipation

The "Rules" of Robust Design

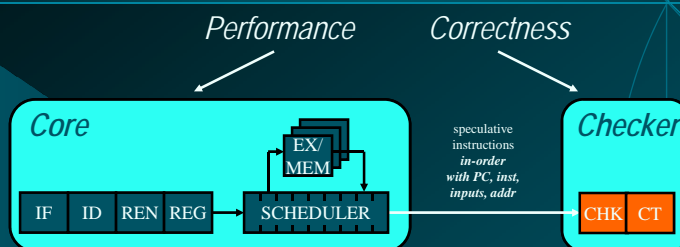
1. Perform functional verification to eliminate design errors
2. Perform electrical verification to ensure robust circuit evaluation
3. Insert design guard bands to ensure no failures during the lifetime of the design



The Burden of Functional Verification

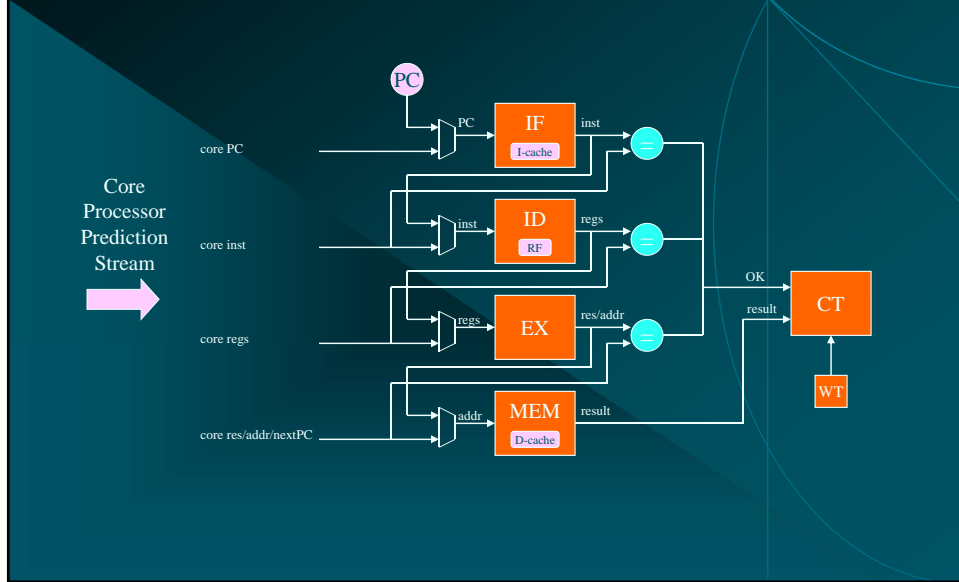
- ❖ Typically consumes two-thirds of all resources
- ❖ Constitutes the primary source of risk to a project
- ❖ It is a thankless and intractable task
- ❖ What if a design could tolerate its design errors?
 - ◆ The verification burden would be lifted
 - ◆ Time-to-market and costs could be slashed

Tolerating Design Errors with DIVA [MICRO'99]

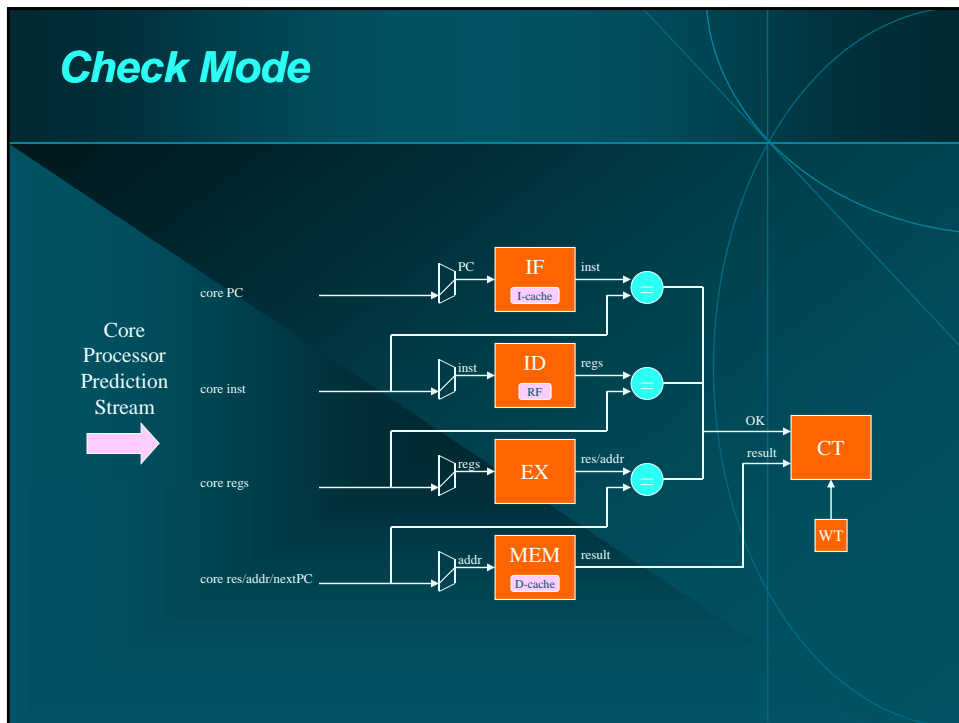


- ❖ All core function is validated by checker
 - ◆ Simple checker *detects* and *corrects* faulty results, restarts core
- ❖ Checker relaxes burden of correctness on core processor
 - ◆ Tolerates design errors, electrical faults, defects, and failures
 - ◆ Core has burden of accurate prediction, as checker is 15x slower
- ❖ Core does heavy lifting, removes hazards that slow checker

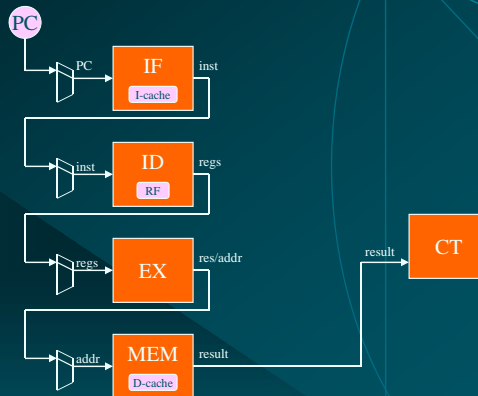
Checker Processor Architecture



Check Mode



Recovery Mode



How Can the Simple Checker Keep Up?



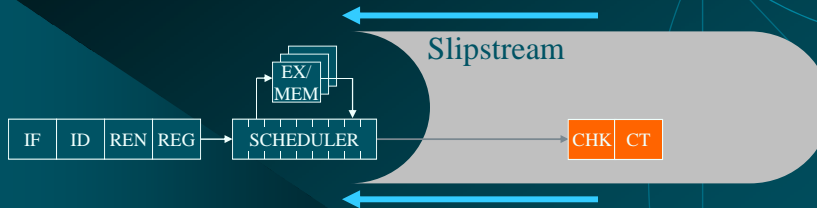
- ❖ Slipstream reduces power requirements of trailing car
- ❖ Checker processor executes inside core processor's slipstream
 - ◆ Fast moving air \Rightarrow branch predictions and cache prefetches
 - ◆ Core processor slipstream reduces complexity requirements of checker
 - ◆ Checker rarely sees branch mispredictions, data hazards, or cache misses

How Can the Simple Checker Keep Up?



- ❖ Slipstream reduces power requirements of trailing car
- ❖ Checker processor executes inside core processor's slipstream
 - ◆ Fast moving air \Rightarrow branch predictions and cache prefetches
 - ◆ Core processor slipstream reduces complexity requirements of checker
 - ◆ Checker rarely sees branch mispredictions, data hazards, or cache misses

How Can the Simple Checker Keep Up?



- ❖ Slipstream reduces power requirements of trailing car
- ❖ Checker processor executes inside core processor's slipstream
 - ◆ Fast moving air \Rightarrow branch predictions and cache prefetches
 - ◆ Core processor slipstream reduces complexity requirements of checker
 - ◆ Checker rarely sees branch mispredictions, data hazards, or cache misses

How DIVA Breaks the Rules

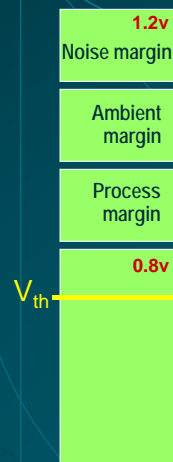
- ❖ **Traditional worst-case design techniques must eliminate all bugs before launch**
- ❖ **Incorporating run-time functional correction mechanisms alleviates the need to find all design bugs**
- ❖ **Leverage slipstreaming benefits to ensure simple checker runs fast**

Thoughts on Rule-Breaking Research

- ❖ **A rule-breaking approach to technical research is effective and engaging**
- ❖ **You will find yourself on very fertile ground**
 - ◆ **Woody Allen: “If you are not failing some of the time, you are not trying hard enough.”**
- ❖ **You will more fully engage your colleagues**
 - ◆ **One half will think your crazy idea will never work**
 - ◆ **One half will be intrigued (with your crazy idea)**
- ❖ **Tips for getting the word out**
 - ◆ **Pick a project name with zero “Google juice”**
 - ◆ **Be an evangelist for your project**

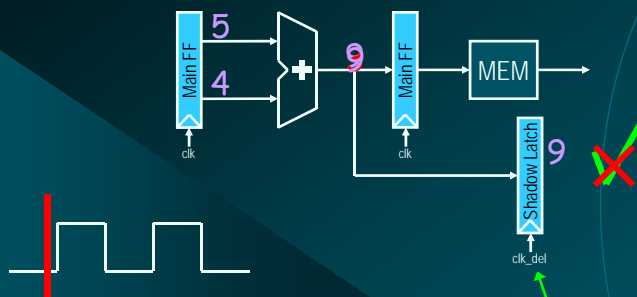
The "Rules" of Robust Design

- ✓ 1. Perform functional verification to eliminate design errors
2. Perform electrical verification to ensure robust circuit evaluation
3. Insert design guard bands to ensure no failures during the lifetime of the design



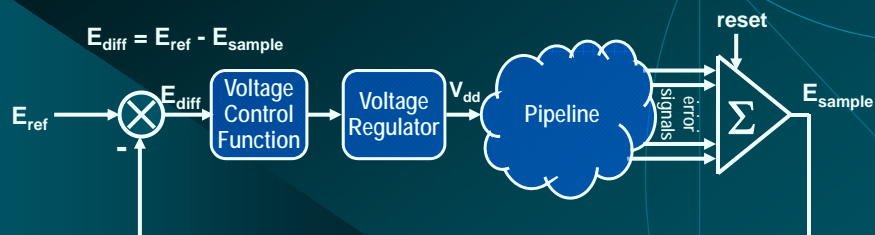
The Excesses of Electrical Verification

Tolerating Electrical Faults with Razor [MICRO'03]



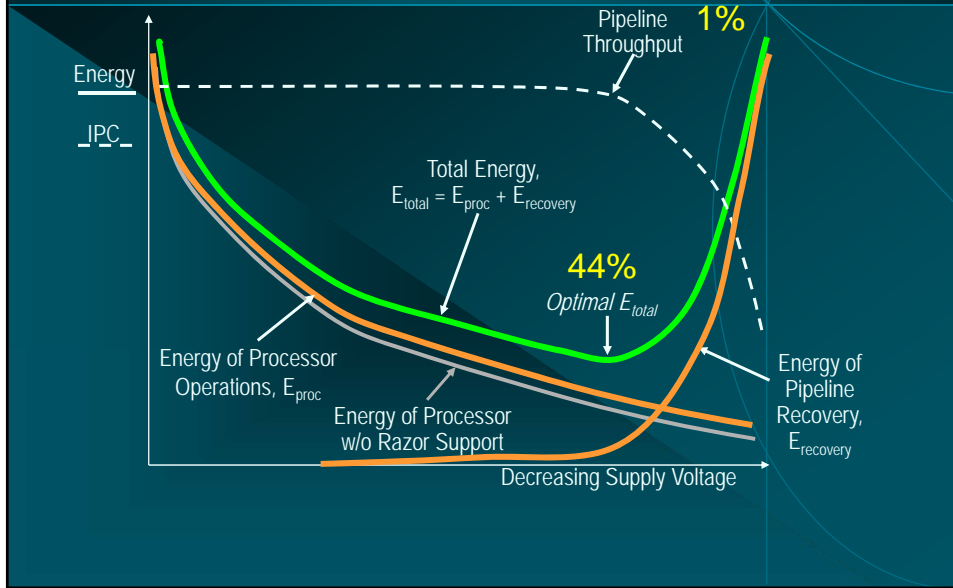
- ❖ Double-sampling metastability tolerant latches detect timing errors
 - ◆ Second sample is correct-by-design
- ❖ Microarchitectural support restores program state
 - ◆ Timing errors treated like branch mispredictions

Razor-Based Dynamic Voltage Scaling



- ❖ Prototype design utilizes a very simple *proportional* control function
 - ◆ Control algorithm implemented in software

Effects of Razor DVS



Razor Prototypes

UMich Razor Prototype
2005: 44% energy savings

Sony Razor Prototype
2008: 32% power savings

ARM Razor Prototype
2010: 52% power savings

I/O

Adaptive FIV control

IRAM

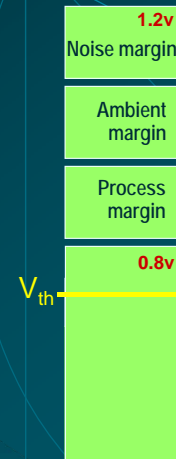
DRAM

Processor Core

external I/O

How Razor Breaks the Rules

- ❖ Traditional worst-case design techniques must observe margin rules for reliable operation
- ❖ Incorporating a timing-error correction mechanisms allow margins to be erased
- ❖ Infrequent use of critical paths allow for even deeper cuts in V_{dd}

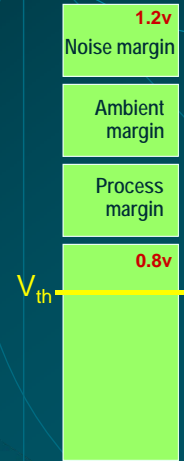


Thoughts on Building (ASIC) Hardware

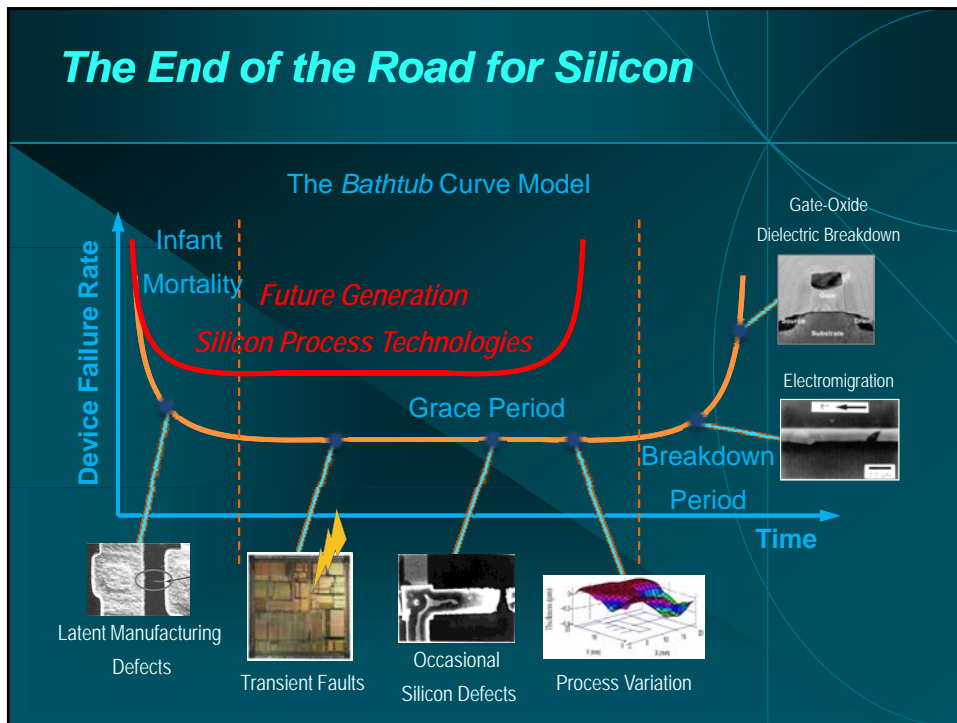
- ❖ Advantages
 - ◆ Sometimes you can't be convincing without a physical demo
 - ◆ If you build it, they (i.e., industry) will come
 - ◆ Custom ASIC == shiny thing
- ❖ Disadvantages
 - ◆ Hardware projects are hungry: they eat money, time and people
 - ◆ There is an *enormous* opportunity cost when you are spending years fleshing out one idea
 - ◆ Often a physical implementation reveals nothing that could not be learned through detailed physical models or FPGA-based implementation

The "Rules" of Robust Design

- ✓ 1. Perform functional verification to eliminate design errors
- ✓ 2. Perform electrical verification to ensure robust circuit evaluation
3. Insert design guard bands to ensure no failures during the lifetime of the design

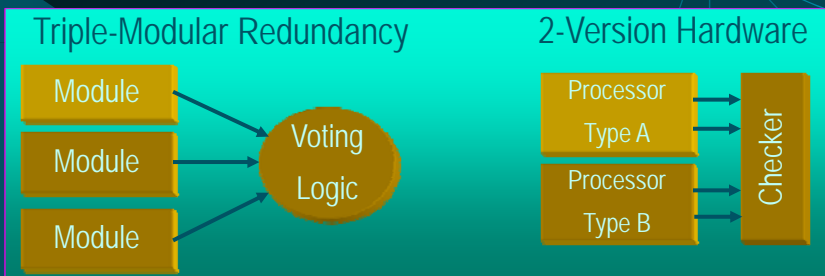


The End of the Road for Silicon



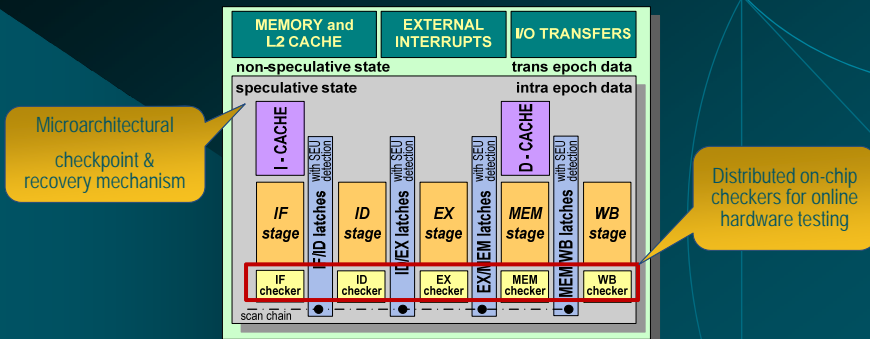
\$\$ Traditional Techniques \$\$

- ❖ Used at high-end life-critical systems (e.g., aviation)



- ❖ Utilize redundant hardware to validate computation
 - ◆ Redundant computation bears high area/power cost
 - ◆ Very costly to employ for mainstream systems

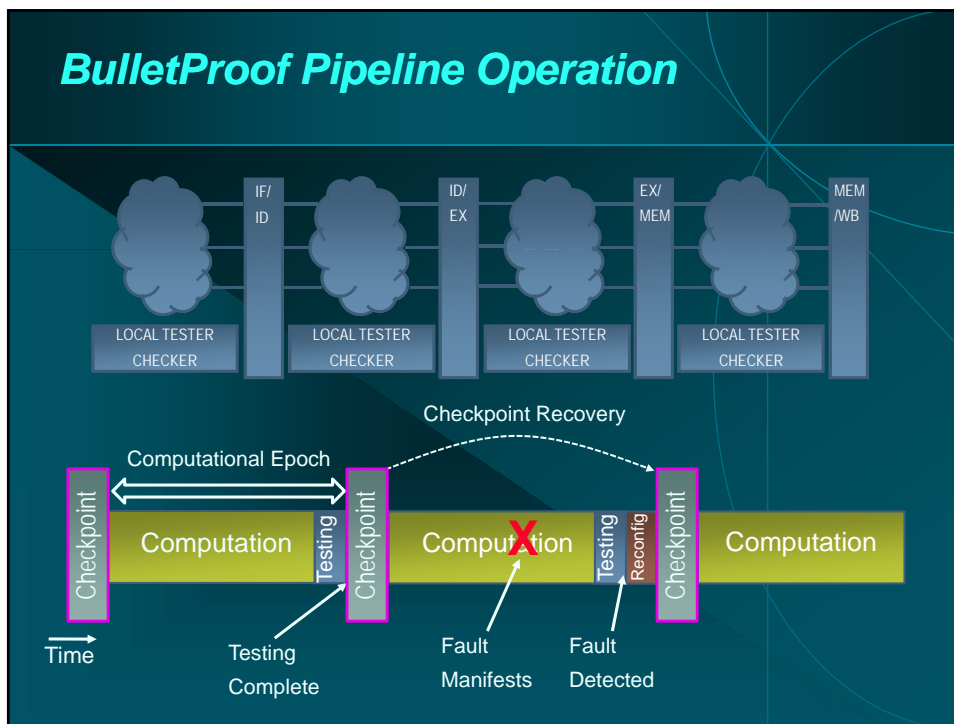
Ultra-Low Cost Defect Protection with BulletProof [ASPLOS'06]



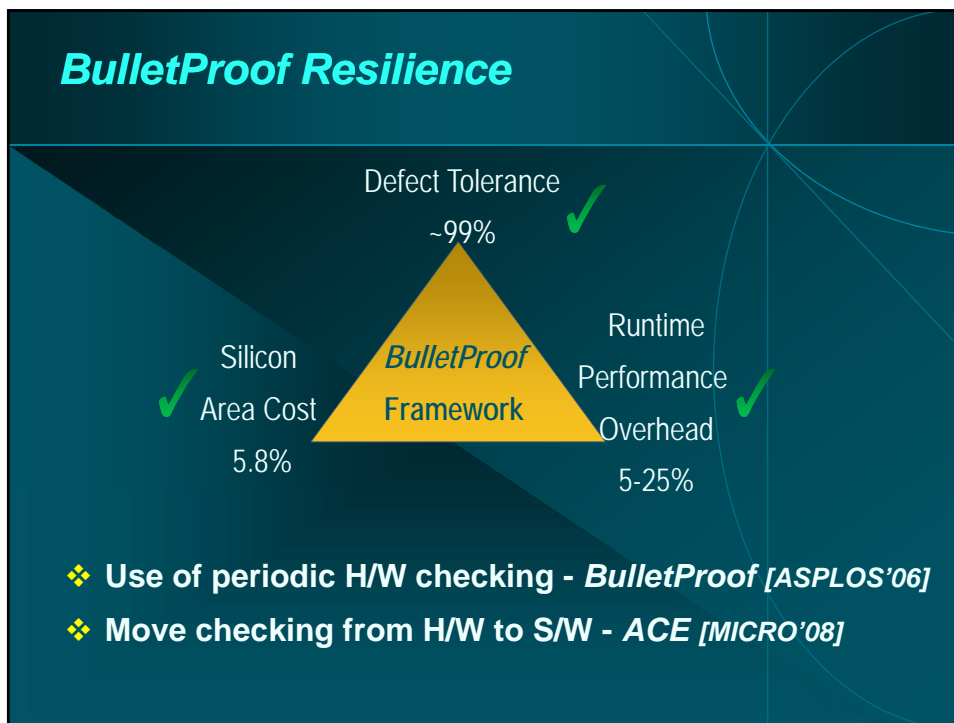
The BulletProof Approach – Periodically check the hardware



BulletProof Pipeline Operation



BulletProof Resilience

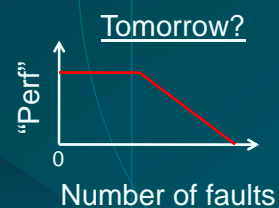
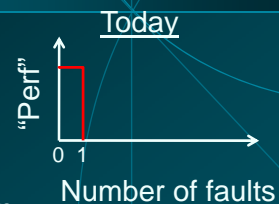


How BulletProof Breaks the Rules

- ❖ Traditional fault-tolerant designs replicate hardware, resulting in very high cost
- ❖ Incorporating online manufacturing-grade testing and checkpointing into the design allows high coverage detection and correction of defect-related faults
- ❖ High-quality testing infrastructure is much less expensive than replication, leading to ultra-low cost defect tolerance solutions

Thoughts on Designing Resilient CMPs

- ❖ Components of a resilient design
 - ◆ Fault detection and diagnosis
 - ◆ Program recovery
 - ◆ System Repair
- ❖ Properties of “ideal” resilient design
 - ◆ Inexpensive detection and diagnosis
 - ◆ Inexpensive recovery
 - ◆ System “performance” proportional to fraction of working devices
- ❖ CMP trends favor many of these requirements
 - ◆ Checkpointing facilitates recovery
 - ◆ “Many”core trend smooths failure impact



Summary

- ❖ Rules were made to be broken
 - ◆ This is one important aspect of “research”

- ❖ New Rules for Robust Design
 1. Utilize dynamic verification to reduce the burden of functional verification
 2. Employ timing speculation to reduce energy demands and speed timing closure
 3. Tighten guard bands and prolong the lifetime of silicon with ultra-low cost defect tolerance techniques

Questions

