

*Sidestepping performance bottlenecks  
and design crises with*  
**Better-Than-Worst-Case design**

---



Todd Austin  
*austin@umich.edu*

Valeria Bertacco  
*valeria@umich.edu*  
Dept. of EECS  
University of Michigan  
Ann Arbor, MI – USA

Krisztian Flautner  
*krisztian.flautner@arm.com*  
ARM, Ltd.  
Cambridge, UK

*Munich - March 7<sup>th</sup>, 2005*

## Introduction

---

Limitations of traditional design approaches  
in light of current technology trends

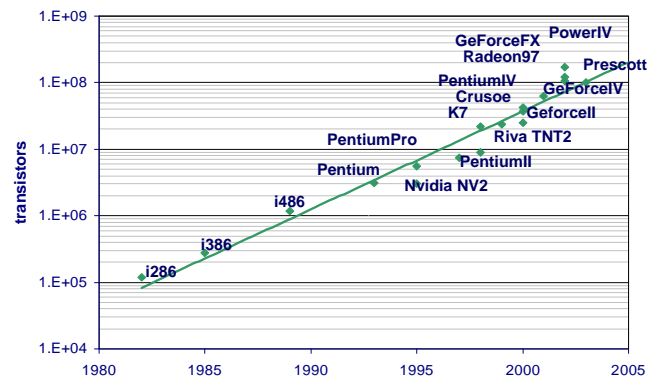
## Pressing Design Challenges in the Nanometer Regime

---

- ❑ Design complexity
    - Billions of transistors lead to untenable designs...
  - ❑ Uncertainty in design parameters
    - Process and temperature variation, supply noise...
  - ❑ Soft errors upset logic and memory
    - Cosmic rays, alpha particles, neutrons, etc...
  - ❑ Power demands
    - Bounding performance, area, battery life
- 

## Design Complexity Trends

---



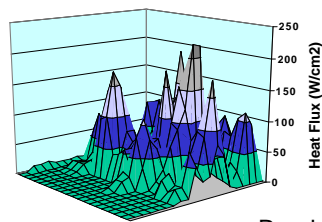
## The Burden of Verification

- ❑ Immense test space
  - Impossible to fully test the system
  - Example: 32 regs, 8k caches, 300 pins =  $2^{132396}$  states
- ❑ Done with respect to ill-defined reference
  - What is correct? Often defined by old designs + gurus
- ❑ Expensive
  - Large fraction of design team dedicated to verification
  - Increases time-to-market, often as much as 1-2 years
- ❑ High-risk
  - Typically only one chance to "get it right"
  - Failures can be costly: replacement parts, bad PR, lawsuits, fatalities

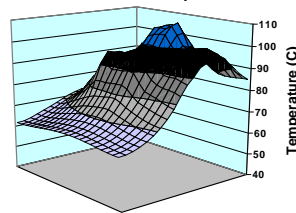
5

## Extreme Variations

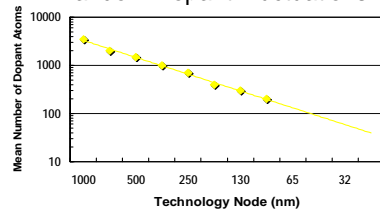
Heat Flux (W/cm<sup>2</sup>)  
Results in Vcc variation



Temperature Variation (°C)  
Results in Hot spots

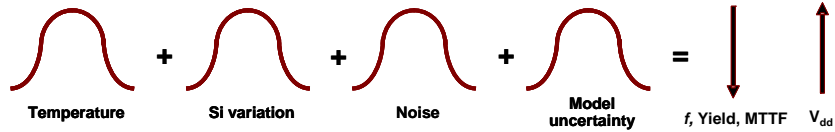


Random Dopant Fluctuations



6

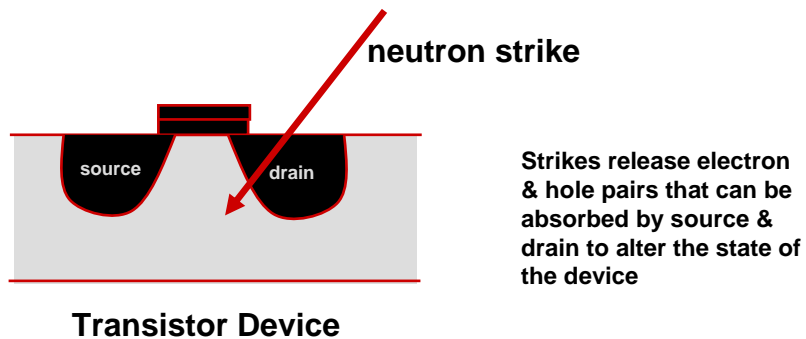
## Uncertainty in Design Parameters



- Uncertainty leads to performance and power overheads
  - Increasing uncertainty with design scaling
  - Intra-die process/temperature variations, inductive noise, deep
  -
- Key Observation: worst-case scenario is highly improbable
  - Significant gain for circuits optimized for common case
  - Efficiency mechanisms needed to tolerate worst-case scenarios

7

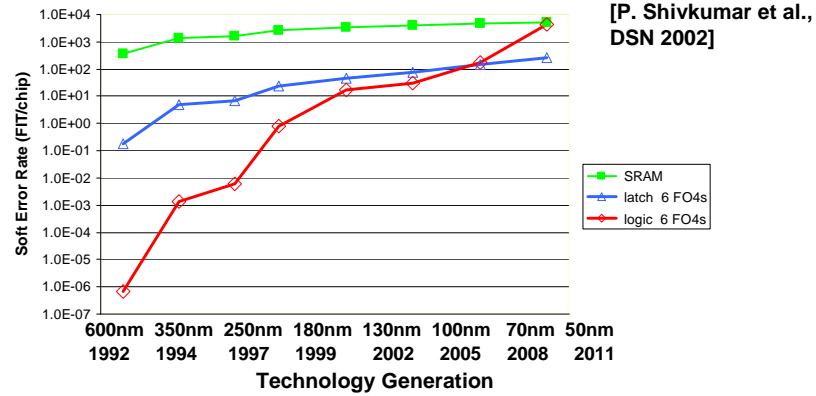
## Impact of Neutron Strike on a Si Device



- Secondary source of upsets: alpha particles from packaging

8

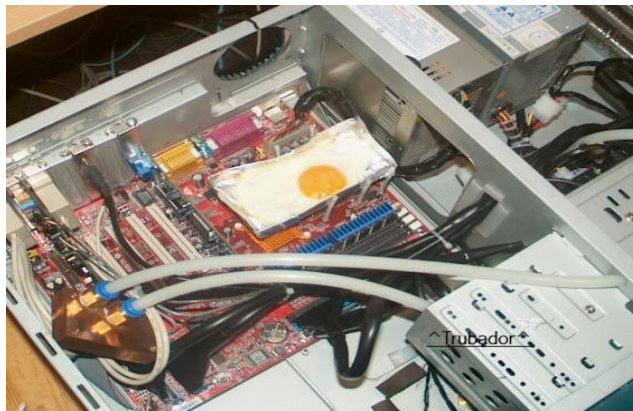
## Soft-Error Trends



- ❑ SER per chip of logic circuits
  - Nine orders of magnitude increase from 600 nm to 50 nm
  - Dominant source of soft errors after 50 nm

9

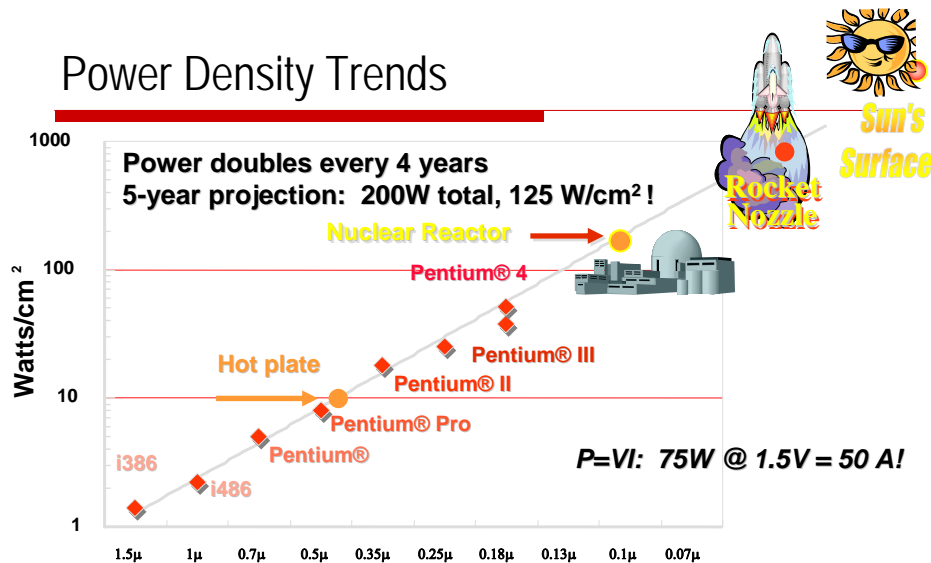
## Fried Egg a la Athlon XP1500+



Source: The New York Times, 25 June 2002

10

## Power Density Trends



\* "New Microarchitecture Challenges in the Coming Generations of CMOS Process Technologies" – Fred Pollack, Intel Corp. Micro32 conference key note - 1999. Courtesy Avi Mendelson, Intel.

11

## Better-Than-Worst-Case (BTWC) design

- ❑ Traditional worst-case design works to avoid errors/faults by assuming worst-case conditions for design validation
- ❑ Better than worst-case design couples a complex designs with a checker component that validates correctness during operation
- ❑ Reduces design effort and enables typical-case optimizations

12

## What is this tutorial about

---

- ❑ **BTWC design**
  - Basic Concepts
  - DIVA Checker
  - Razor Logic
  - Other BTWC solutions
- ❑ **CAD challenges and opportunities**
  - Typical-Case design Optimization (TCO)
  - Circuit-level observability and system-level performance
- ❑ **Open discussion**
- ❑ **Conclusion**

---

13

## Goals of this tutorial

---

1. Introduce and motivate the concept of Better Than Worst-Case design
2. Familiarize the attendees with a number of BTWC designs (ours and others)
3. Introduce efforts (circuit-aware architectural simulation typical case optimization) that highlight the challenges and opportunities that BTWC poses to CAD
4. Facilitate an open discussion on the implications of BTWC design on CAD

---

14

---

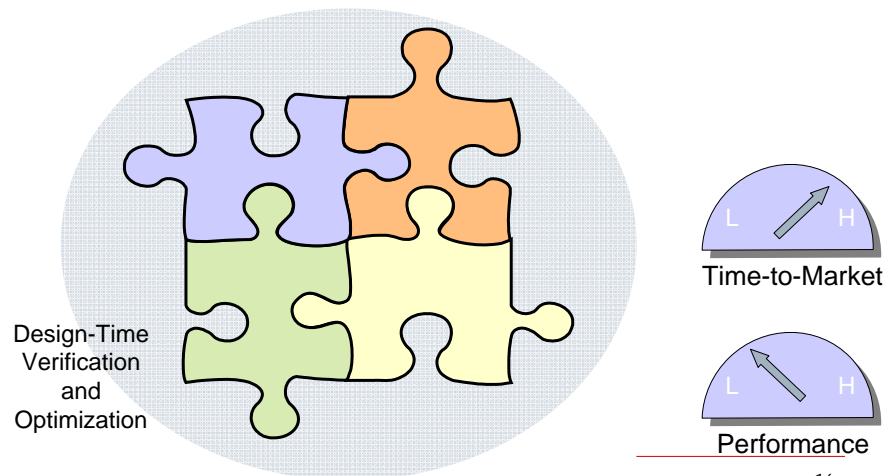
## Better-Than-Worst-Case design

---

15

## Traditional Worst-Case Design

---

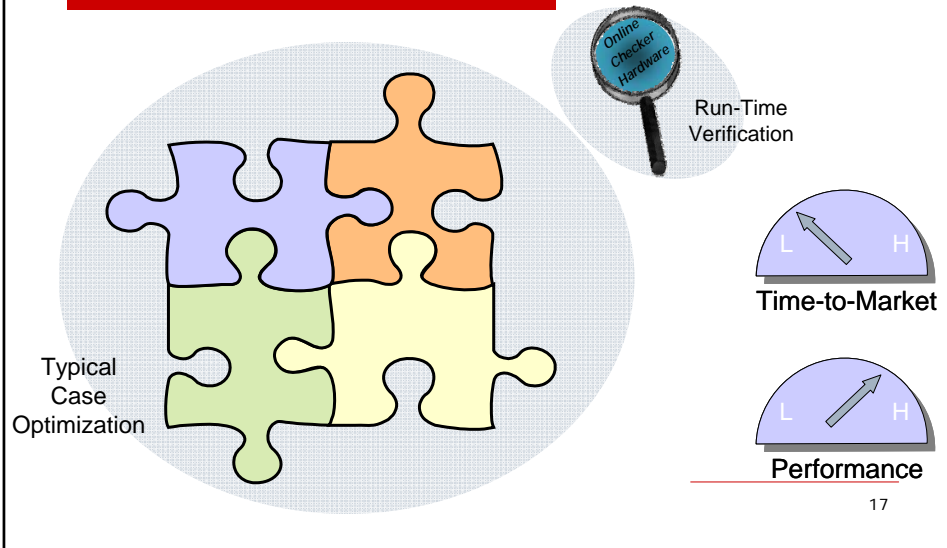


---

16

## Better-Than-Worst-Case (BTWC) design

---



17

## Outline

---

- DIVA Checker
- ❑ Razor Logic
- ❑ Other BTWC designs

---

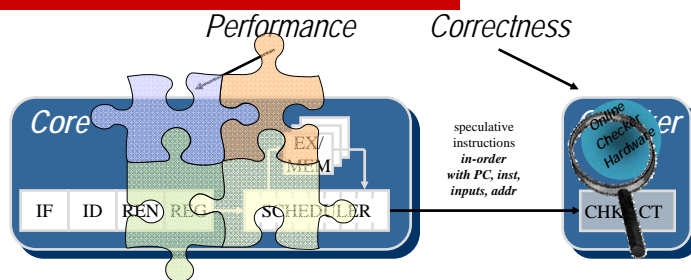
18

## Motivating Observations

- ❑ Online functional verification cover most faults
  - Single-event upsets and noise-related faults
  - Design faults and incomplete implementation
  - Untestable silicon defects and in field circuit failures
  - *Utilize  $N(2)$ -version hardware to detect and correct faults*
- ❑ Increasing speculation reduces exposure to faults
  - Predictors need not be correct, functionally or electrically
  - *Approach leverages a maximally speculative architecture*
- ❑ While complex, processors have simple semantics
  - Need not validate all internals, only exposed semantics
  - *Only check instruction semantics for low overheads*

19

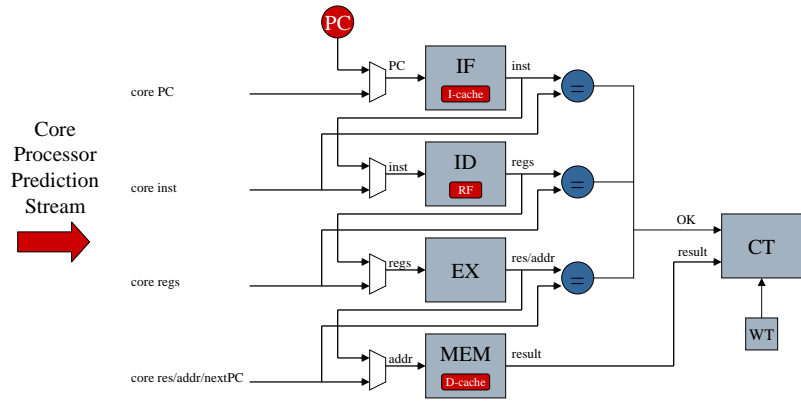
## Example BTWC Design: DIVA Checker [Austin'99]



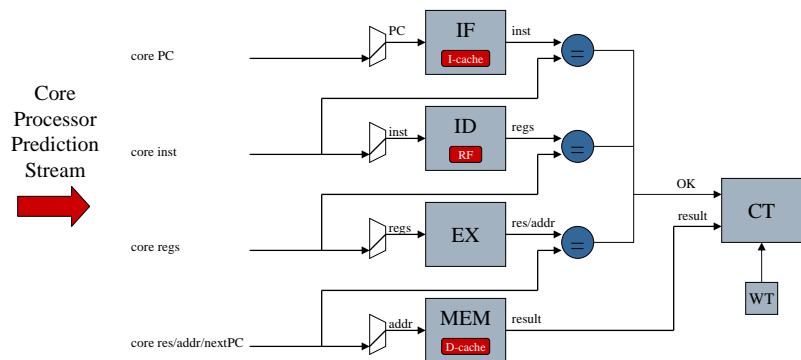
- ❑ All core function is validated by checker
  - Simple checker *detects* and *corrects* faulty results, restarts core
- ❑ Checker relaxes burden of correctness on core processor
  - Tolerates design errors, electrical faults, defects, and failures
  - Core has burden of accurate prediction, as checker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

20

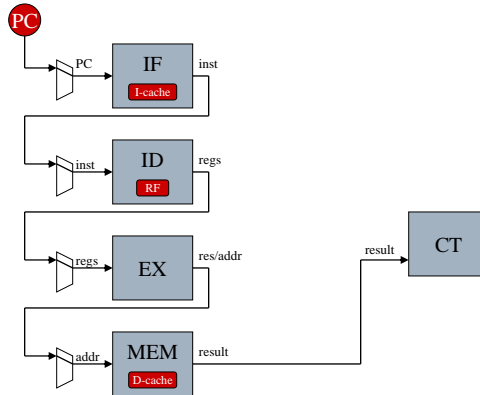
# Checker Processor Architecture



# Check Mode

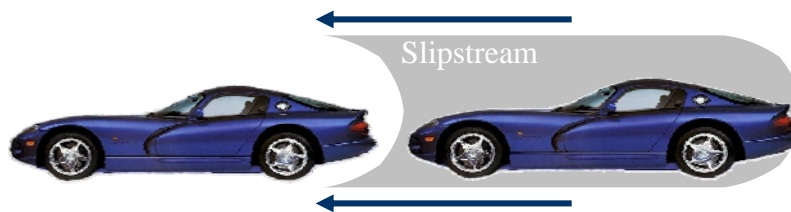


## Recovery Mode



23

## 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

24

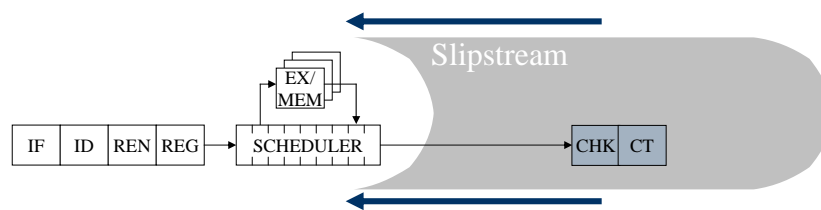
## How Can the Simple Checker Keep Up?



- ❑ 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

25

## How Can the Simple Checker Keep Up?

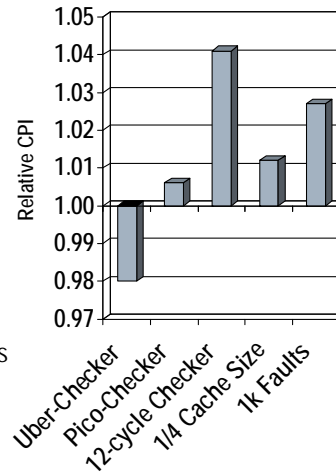


- ❑ Slipstream reduces power requirements of trailing car
- ❑ Checker processor executes inside core processor's slipstream
  - fast moving air  $\Rightarrow$  branch predictions and cache prefetches
  - Core processor slipstream reduces complexity requirements of checker
  - Checker rarely sees branch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

26

## Checker Performance Impacts

- ❑ Checker *throughput* bounds core IPC
  - Only cache misses stall checker pipeline
  - Core warms cache, leaving few stalls
- ❑ Checker *latency* stalls retirement
  - Stalls decode when speculative state buffers fill (LSQ, ROB)
  - Stalled instructions mostly nuked!
- ❑ *Storage hazards* stall core progress
  - Checker may stall core if it lacks resources
- ❑ *Faults* flush core to recover state
  - Small impact if faults are infrequent

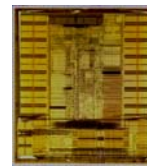


27

## REMORA: Physical Checker Design

- ❑ Physical checker design effort underway
  - Alpha integer ISA subset
  - 4-wide checker, 0.5k I-cache, 4k D-cache
  - Synthesized design (using Synopsys)
- ❑ Physical design estimates
  - 950 MHz clock speed (degree-8 pipe)
  - 12 mm<sup>2</sup> total area in 0.25  $\mu$ m technology
  - 941 mW worst-case power
- ❑ Design also includes:
  - Pipelined checker design, simple core
  - Clock/voltage tuning infrastructure
  - Extensive BIST support

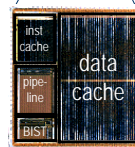
Alpha 21264



REMORA  
Checker

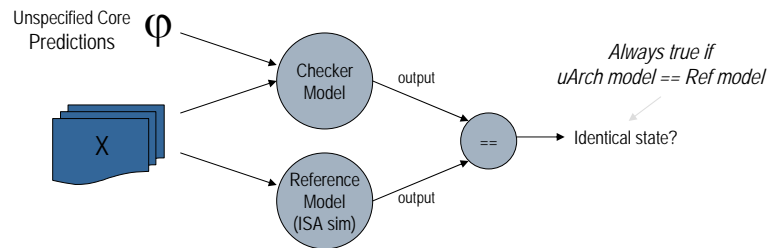
12 mm<sup>2</sup>  
(in 0.25  $\mu$ m)

205 mm<sup>2</sup>  
(in 0.25  $\mu$ m)



28

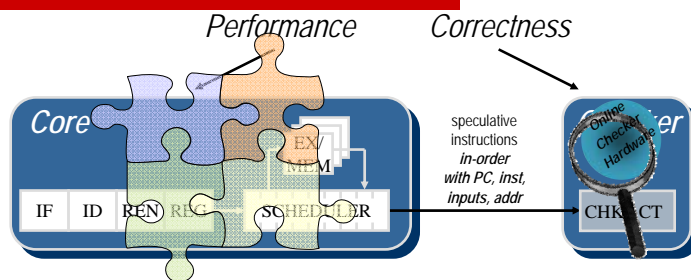
## Verifying the Checker Processor



- ❑ Simple checker permits complete functional verification
  - In-order blocking pipelines (trivial scheduler, no rename/reorder/commit)
  - No “internal” non-architected state
- ❑ Fully verified design using Sakallah’s GRASP SAT-solver
  - For Alpha integer ISA without exceptions
  - With small register file and memory, and small data types

29

## Example BTWC Design: DIVA Checker



- ❑ 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

30

## Outline

---

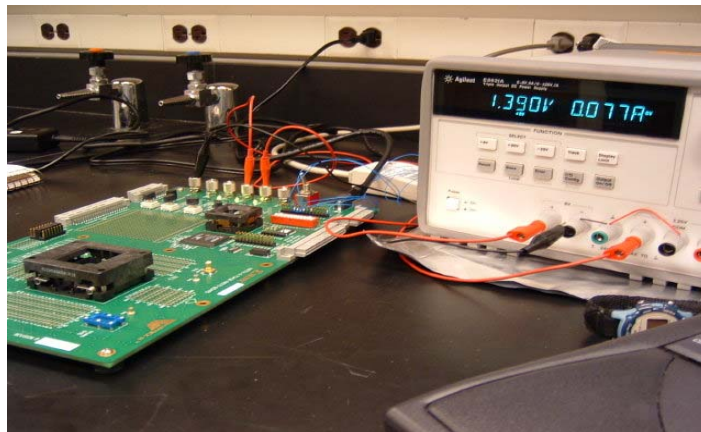
- ❑ DIVA Checker
- Razor Logic
- ❑ Other BTWC designs

---

31

## Motivating Study: Voltage vs. Circuit Error Rate

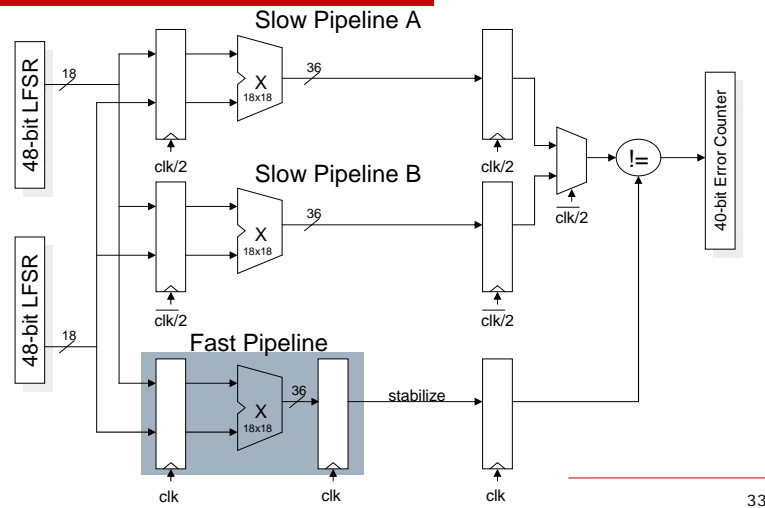
---



---

32

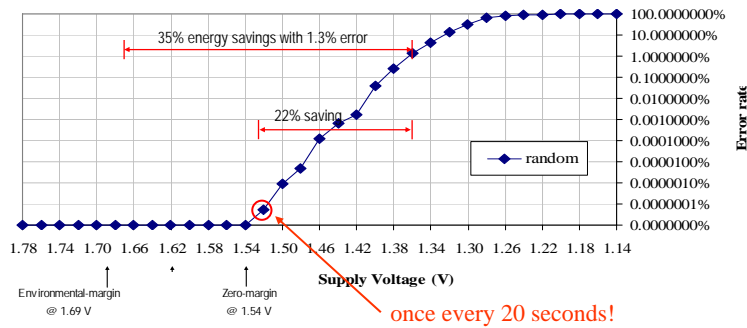
## Circuit Under Test



33

## Error Rate Studies – Empirical Results

18x18-bit Multiplier Block at 90 MHz and 27 C

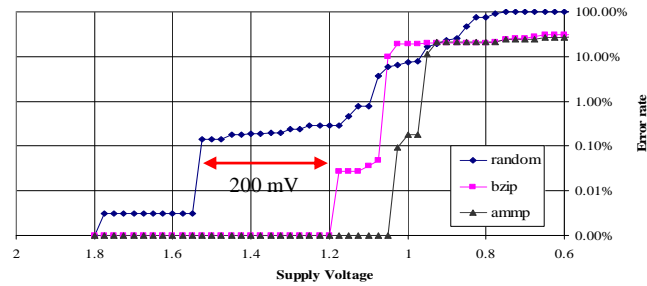


34

## Error Rate Studies – SPICE-Level Simulations

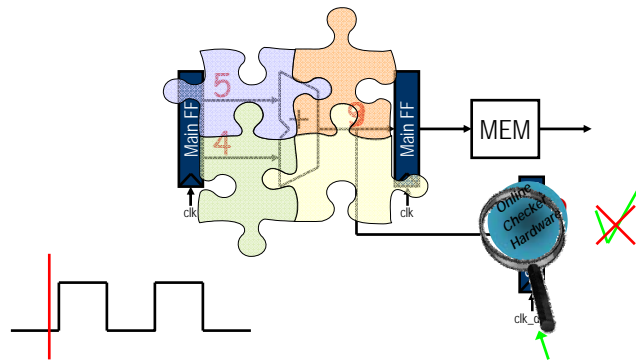
- Based on a SPICE-level simulations of a Kogge-Stone adder

Kogge-Stone Adder at 870 MHz and 27 C



35

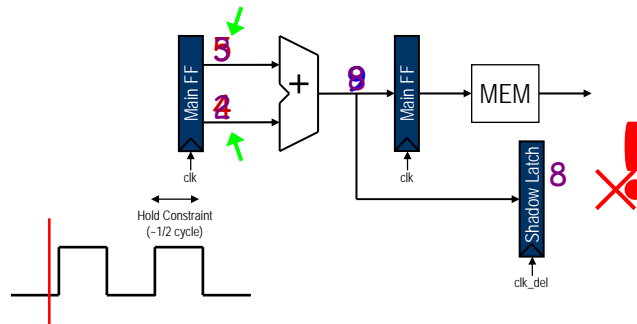
## Another BTWC Design: Razor Logic [Ernst'03]



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

36

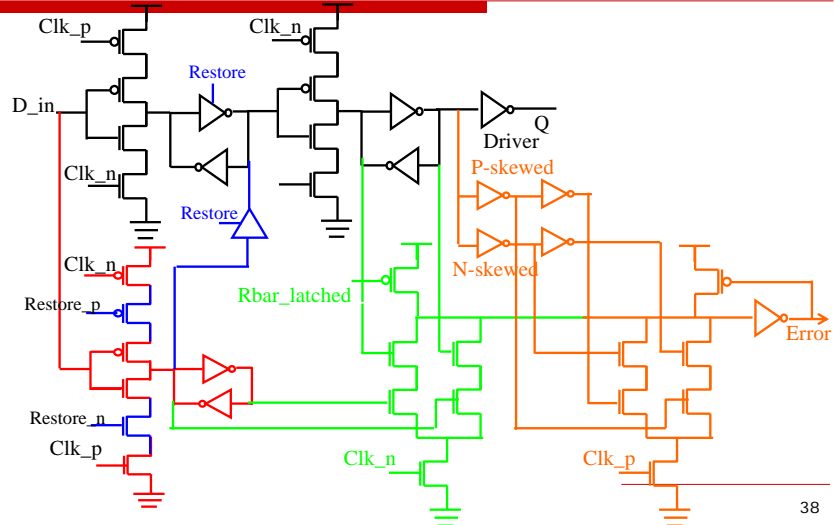
## Razor Short Path Constraint



- Short-path timing constraint prevents shadow latch corruption

37

## Razor Flip-Flop

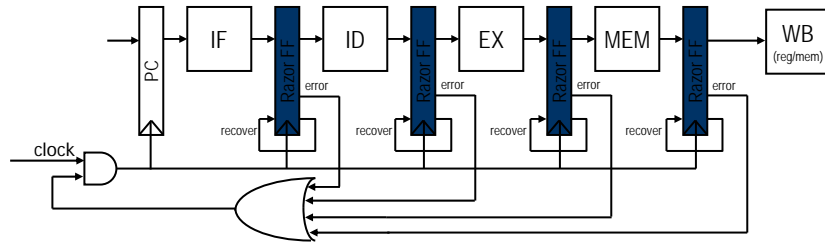


38

## Centralized Pipeline Recovery Control

Cycle: 6

inst6



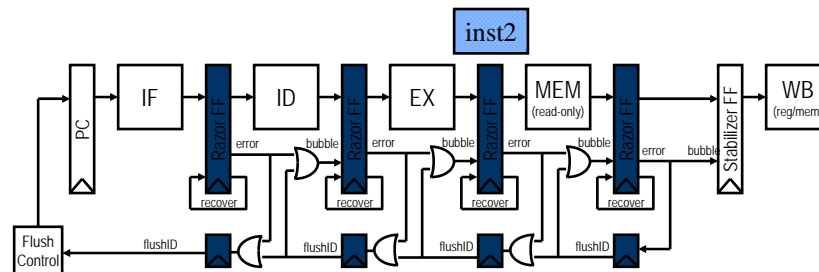
- ❑ Once cycle penalty for timing failure
- ❑ Global synchronization may be difficult for fast, complex designs

39

## Distributed Pipeline Recovery

Cycle: 6

inst4

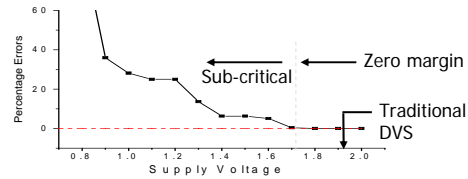


- ❑ Builds on existing branch prediction framework
- ❑ Multiple cycle penalty for timing failure
- ❑ Scalable design as all communication is local

40

## Shaving Voltage Margins with Razor

- *Goal*: reduce voltage margins with *in-situ* error detection and correction

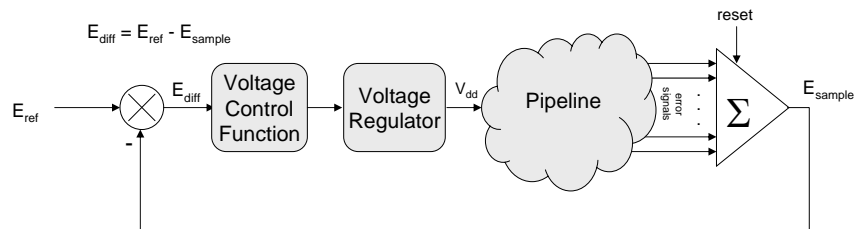


- *Approach*:

- Tune processor voltage based on error rate
- Eliminate margins, run *below* critical voltage
  - Trade-off: power savings vs. overhead of correction

41

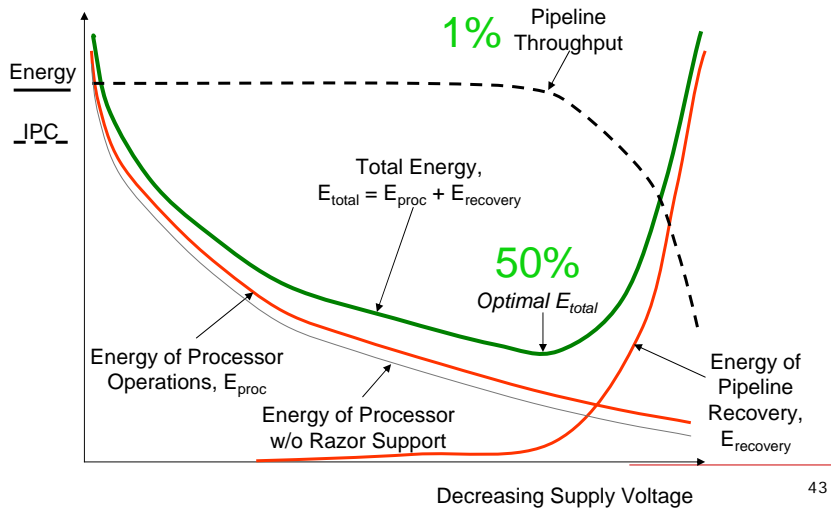
## Razor Opportunity: Typical-Case Energy Reduction



- Energy reduction can be realized with a simple *proportional* control function
  - Control algorithm implemented in software

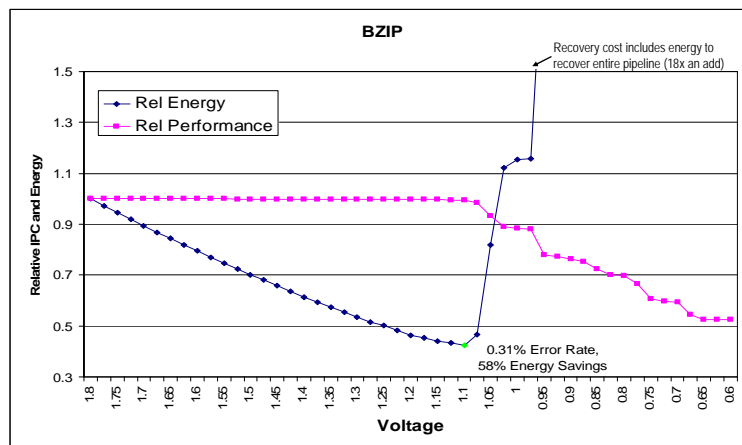
42

## Energy/Performance Characteristics



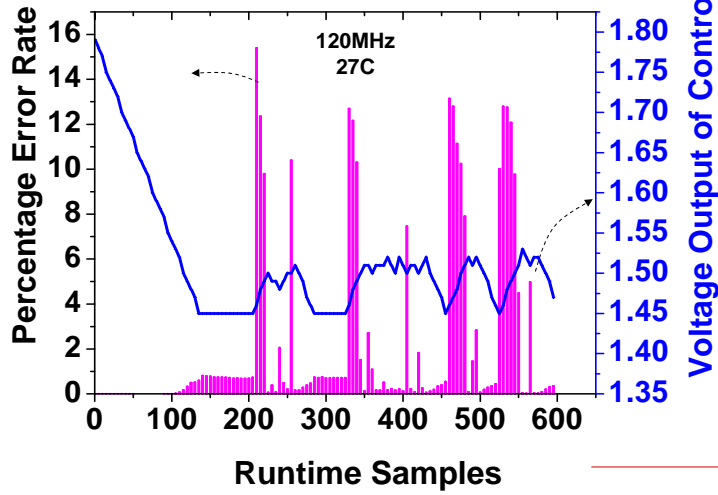
43

## Simulation Results: Optimal Voltage Sweep



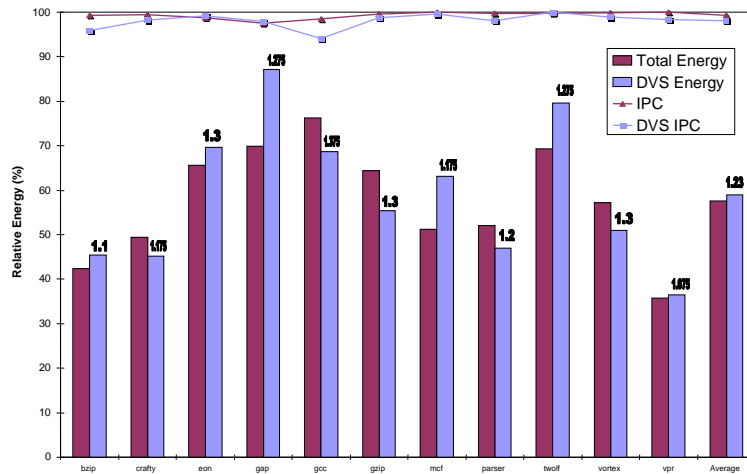
44

## Voltage Controller Performance



45

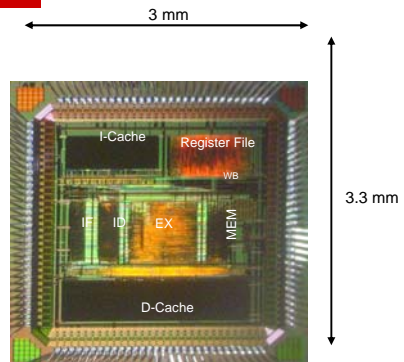
## Simulation Results: Razor DVS Performance



46

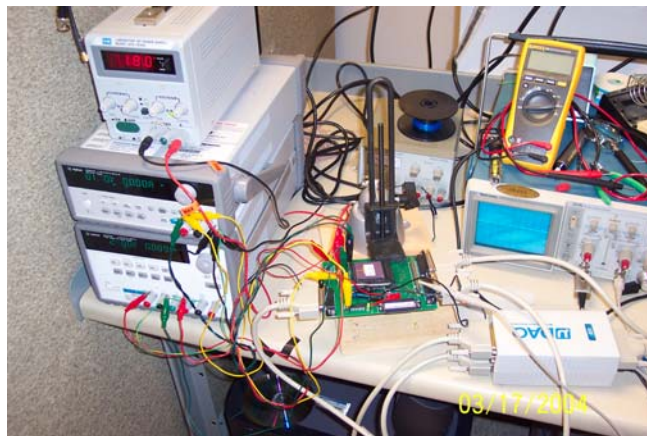
## Razor Prototype Silicon

- ❑ 4 stage 64-bit Alpha pipeline
  - 120 - 160MHz operation
  - 0.18 $\mu$ m technology, 1.8V
- ❑ Razor overhead:
  - Total of 192 Razor flip-flops out of 2408 total (9%)
  - Error-free power overhead:
    - Razor flip-flops: < 1%
    - Short path buffer: 2.1%
  - Recovery power overhead:
    - Razor latch power overhead: 2% at 10% error rate
    - Additional power overhead due to re-execution of instructions



47

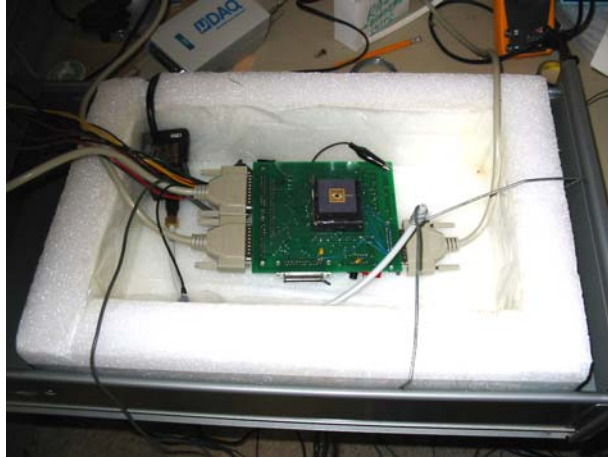
## Razor Prototype Testbed



48

## Razor Prototype Testbed

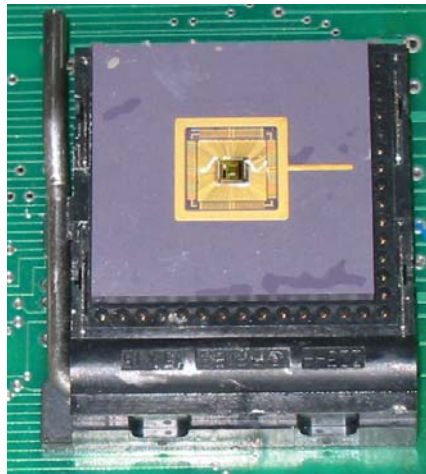
---



49

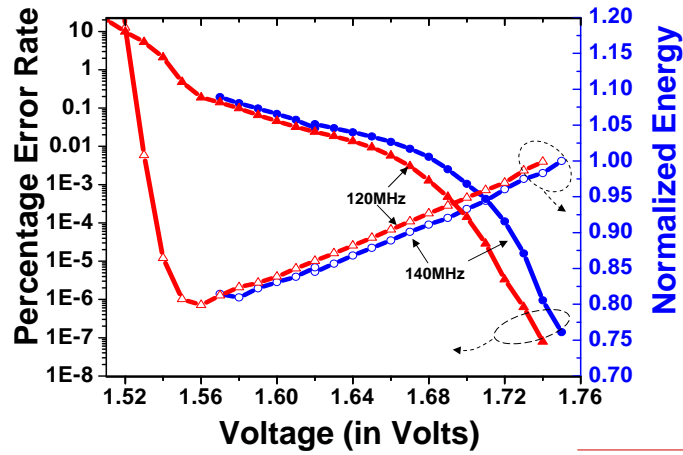
## Razor Prototype Testbed

---



50

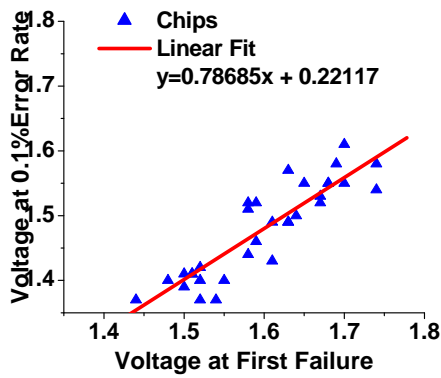
## Error Rate and Normalized Energy Savings



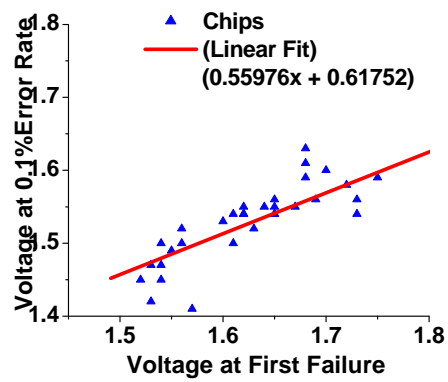
51

## Point of 0.1% Error Rate and Point of First Failure

120MHz



140MHz



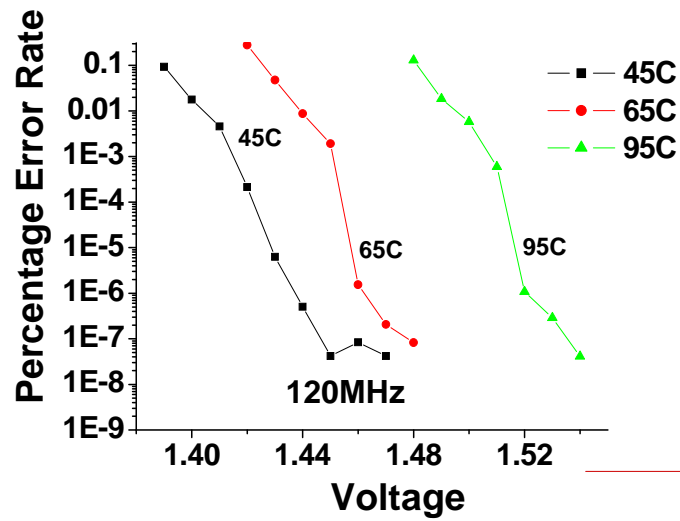
52

## Razor Prototype Testbed



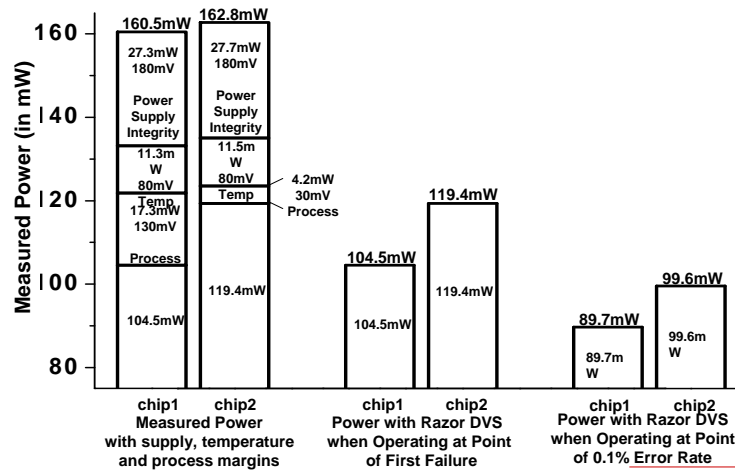
53

## Temperature Margins



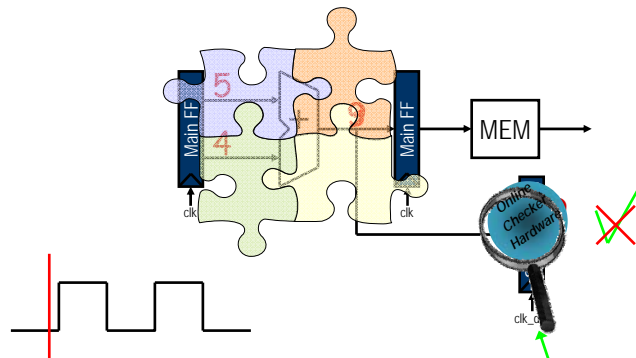
54

## Razor Energy Savings@120MHz,45C



55

## Another BTWC Design: Razor Logic



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

56

## Outline

---

- ❑ DIVA Checker
- ❑ Razor Logic
- Other BTWC designs

---

57

## Other Better Than Worst-Case designs

---

- ❑ Algorithmic-Noise Tolerance, Shanbhag et al.
  - Converting circuit faults to S/N component
- ❑ Approximate Circuits, Lu et al.
  - Architecture-level speculation on computation
- ❑ TEAtime Adaptive Clock, Uht et al.
  - Adaptive clock control
- ❑ On-Chip Self-Calibrating Busses, Worm et al.
  - Error recovery logic for on-chip busses
- ❑ Self-Tuning Circuits, Kehl et al.
  - Early work on dynamic timing error avoidance
- ❑ Time Based Transient Fault Detection, Anghel et al.
  - Double sampling latches for speed testing



**March 2004**

---

58

## Algorithmic Noise Tolerance

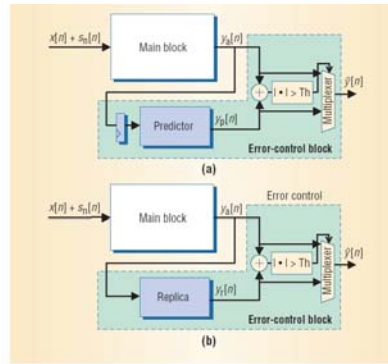


Figure 6. Algorithmic noise-tolerance techniques. (a) A predictor uses the past outputs to generate a statistical estimate of the main block. (b) Reduced-precision redundancy employs a replica of the main filter as an estimator.

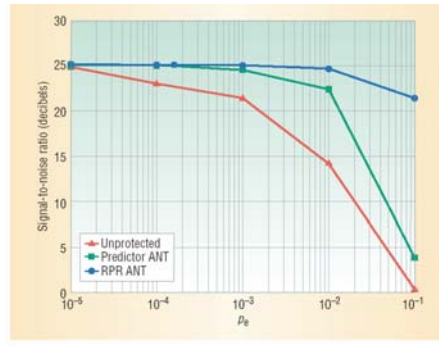


Figure 7. Predictor and reduced-precision redundancy ANT technique performance in the presence of random noise. The signal-to-noise ratio improves by 10 decibels even when each output bit of the filter is independently flipped at an average rate of once every 1,000 samples.

[Shanbhag '04]

## Approximate Circuits

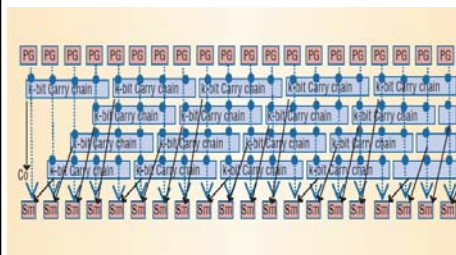


Figure 1. Sample 32-bit approximation adder. The adder has the usual carry, propagate, generate, and sum

circuits but also implements a carry chain with 29 4-bit carry blocks and three boundary cells.

[Lu '04]

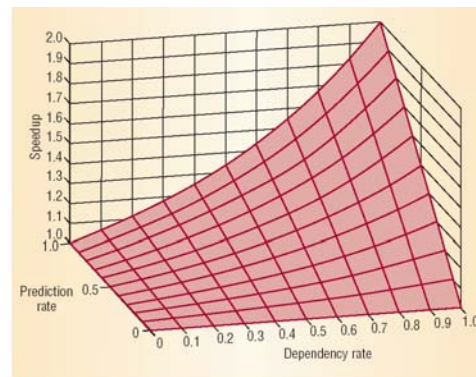
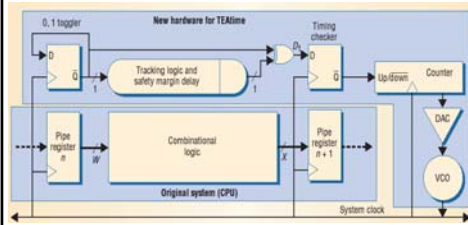


Figure 2. Speedup of speculative execution as a function of prediction rate and instruction dependency rate. The functional unit writeback-bandwidth-occupancy rate is 50 percent.

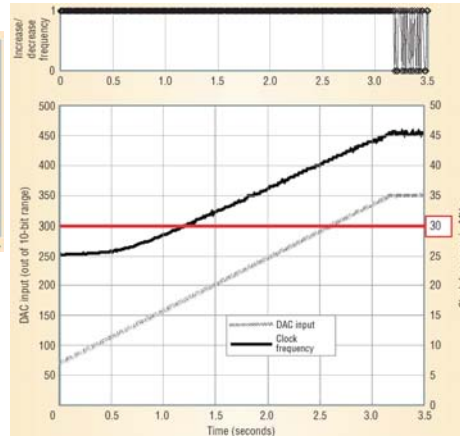
# TEAtime Adaptive Clock



**Figure 1. TEAtime CPU or digital system modifications. TEAtime adds a toggler, tracking logic and safety**

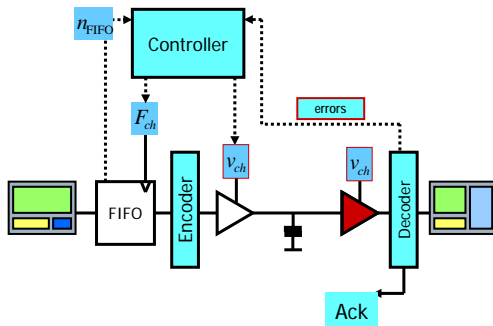
**margin delay, timing checker, up-down counter, digital-to-analog converter, and voltage-controlled oscillator.**

[Uht '04]

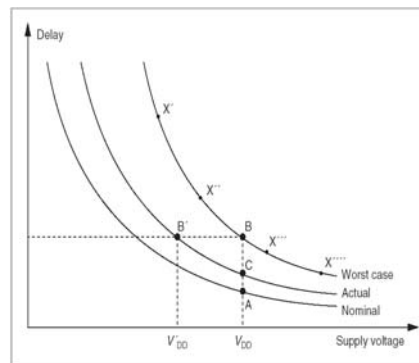


61

# On-Chip Self-Calibrating Busses



**Figure 2. The basic idea of a self-calibrating, point-to-point, unidirectional on-chip interconnect: the classic static scheme, with a FIFO buffer to decouple the two subsystems (a); the proposed self-calibrating scheme, with the elements needed to achieve the desired goals (b).**

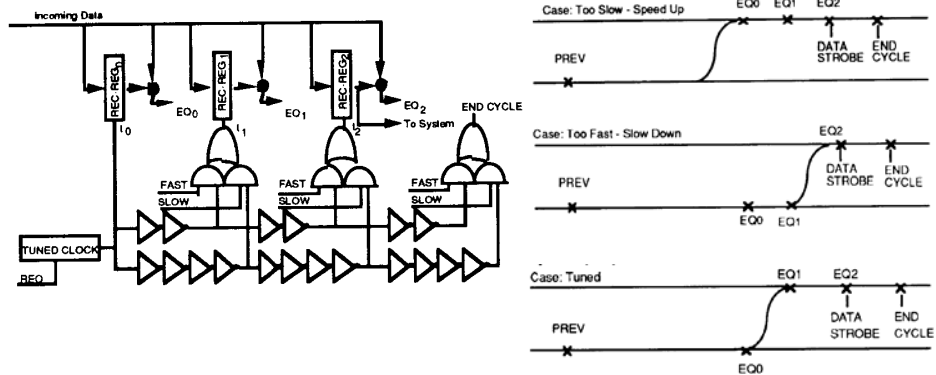


**Figure 1. Delay and voltage relation for nominal, actual, and worst-case design. The worst-case design typically wastes resources—usually silicon area and, more critically, energy. Traditional dynamic voltage-scaling techniques would select only points such as X to X''.**

[Worm '04]

62

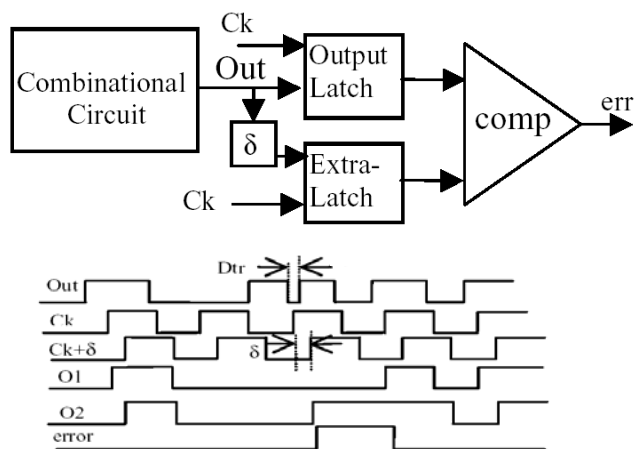
## Self-Tuning Circuits



[Kehl '93]

63

## Time Redundancy Based Transient Fault Detection



[Anghel '00]

64

---

## CAD opportunities for BTWC

---

65

## Key observation

---

In a BTWC context

*infrequent faults in the core design are tolerable.*

- A fault is infrequent if:
  - Environmental conditions trigger it rarely
  - Normal system operation activate a faulty configuration rarely

---

66

## CAD opportunities

---

- ❑ Synthesis
  - Optimize performance/power for the most common scenarios (*typical-case optimization*)
  - Flexible synthesis tools - Optimization constraints can be relaxed
  - Finer granularity of synthesis objectives – probability density curves
  
- ❑ Verification
  - Accurate evaluation- Statistical analysis of execution scenarios
  - Verification focuses on “frequent” transactions/ path of execution
  - Verification focuses on critical components
  - Safety mechanisms detect “rare” problems at a performance cost
    - Performance and verification are intertwined

---

67

## Outline

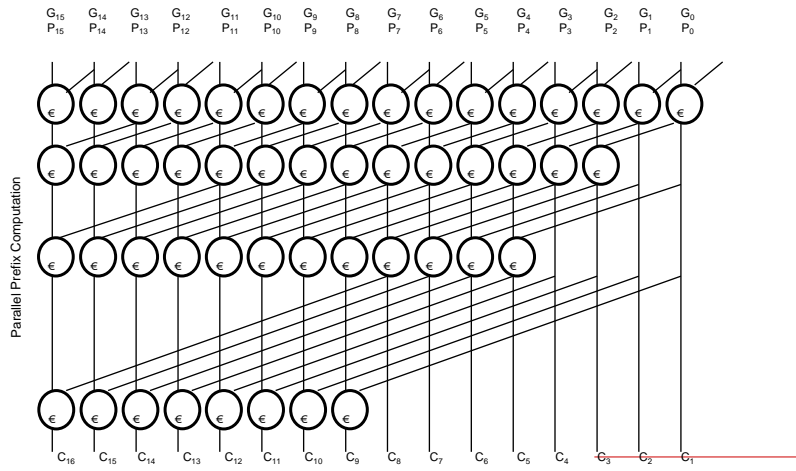
---

- *Synthesis* - Typical-Case Optimized Adders
- ❑ *Performance/verification* - Circuit-Aware simulation
- ❑ *Verification* - Beta-release processors

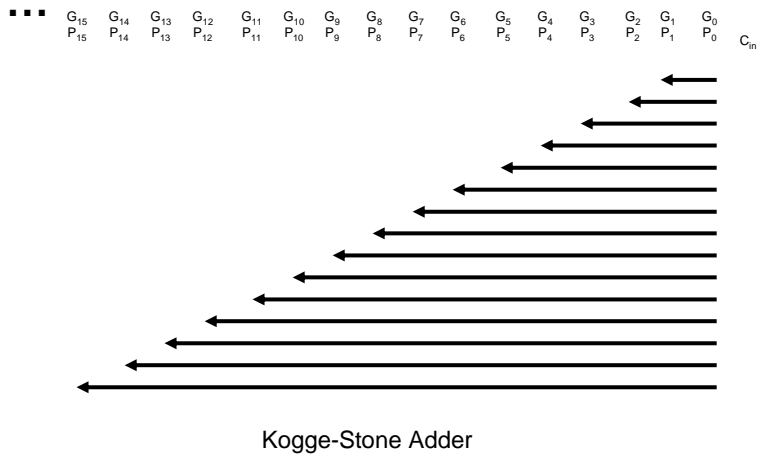
---

68

# Kogge-Stone Adder

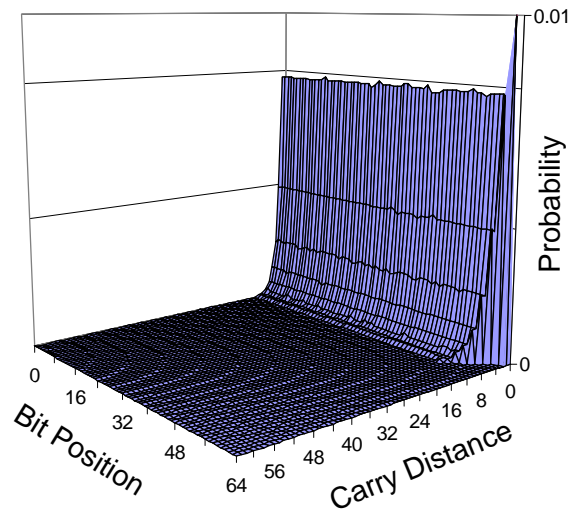


# BTWC Opportunity: Typical-Case Optimized Adder



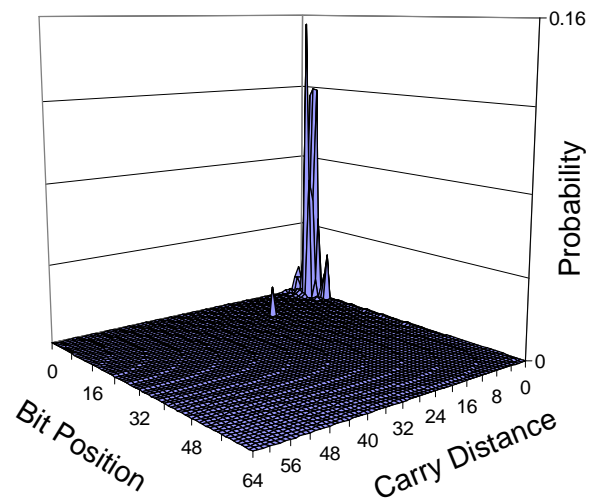
## Carry Propagations for Random Data

---

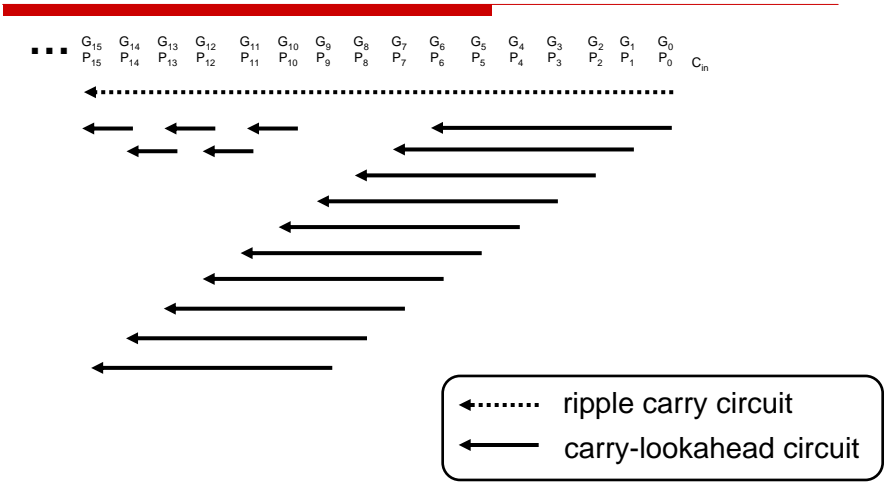


## Carry Propagations for Typical Data

---



# Typical Case Optimized Adder



# Benefits of Typical Case Optimization

Adder Topology	Latency (in gate delays)		
	<i>Worst-Case</i>	<i>Typical-Case</i>	<i>Random</i>
Kogge-Stone	8	5.08	7.09
TCO Adder	128	3.03	3.69

Typical-case performance much better than worst case, relevant in a TCO context

## Outline

- ❑ *Synthesis* - Typical-Case Optimized Adders
- *Performance/verification* - Circuit-Aware simulation
- ❑ *Verification* - Beta-release processors

75

## Simulation for BTWC

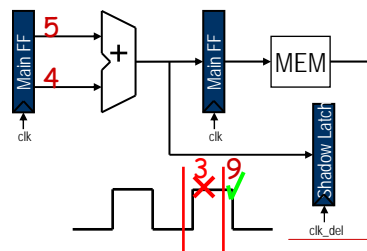
**Electrical**, transient phenomena affect the performance of BTWC designs

Simulation tools need to be "electrically accurate"

Functional correctness is re-defined in a **statistical** context

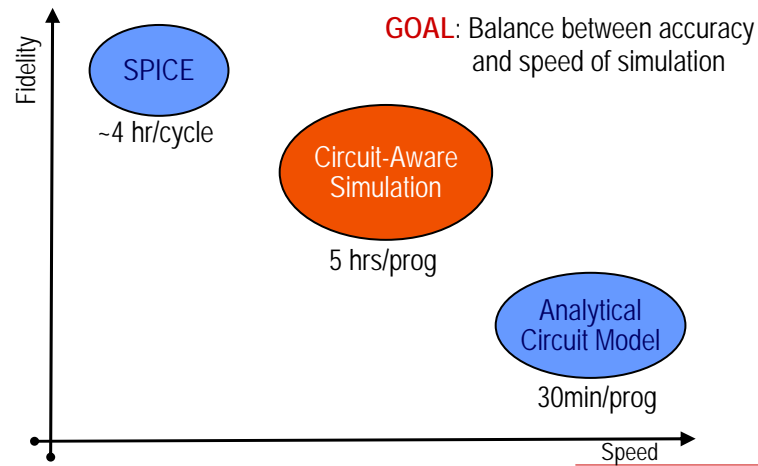
Need ability to gather statistical simulation data

Example: Razor latches



76

## Key Challenges: Speed and Fidelity



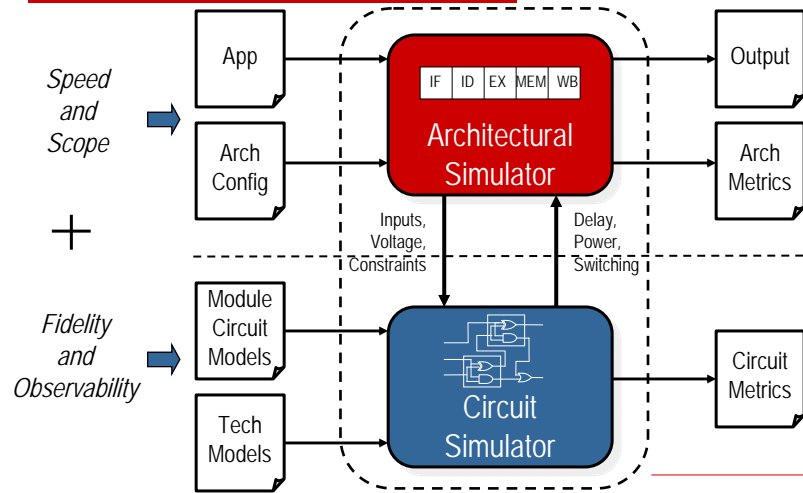
77

## Circuit-Aware is not only for BTWC designs

- ❑ There is a recent trend in computer architecture design toward system that can adapt to circuit-level phenomena
  - e.g., di/dt, thermal throttling
- ❑ These novel circuit-aware architectural optimization share a modeling requirement of detailed circuit
  - Needs to be interaction between architectural state and circuit behavior ( e.g., device switching activity, detail timing information of pipeline states )
- ❑ Analytical circuit modeling has been widely used
  - Simple and fast
  - At the cost of accuracy

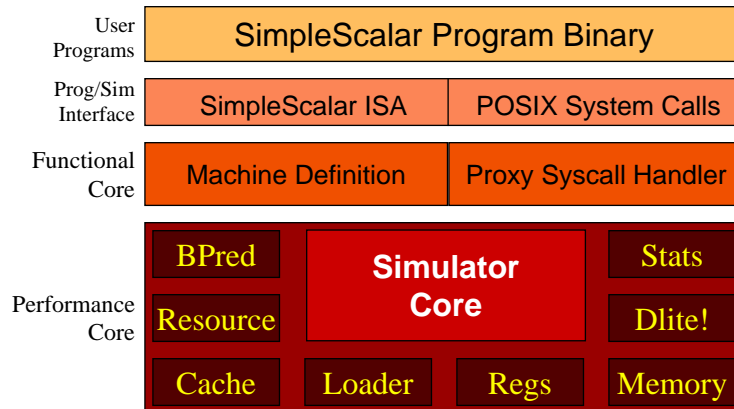
78

## Circuit-Aware Architectural Simulation Platform Overview



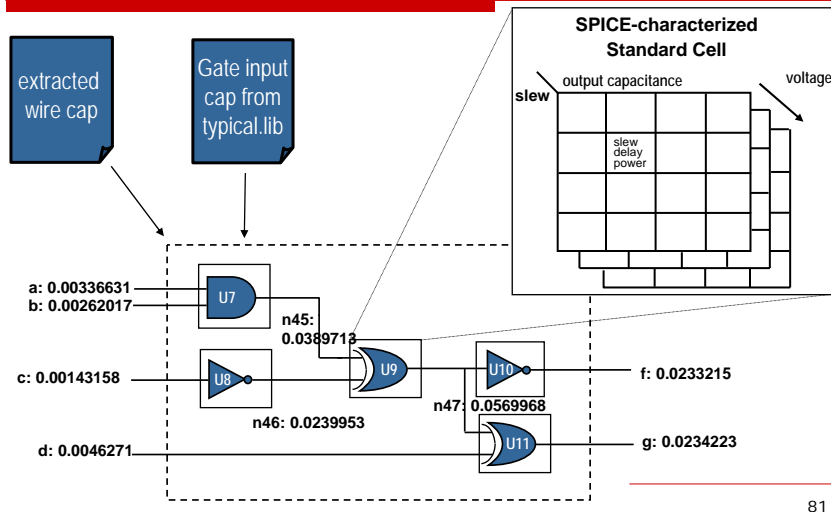
79

## Architectural Simulator Structure



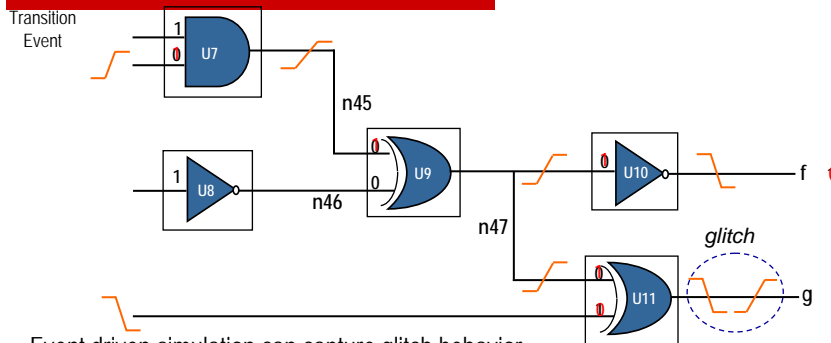
80

## Standard Circuit Simulation Approach



81

## Event Driven Implementation



- Event driven simulation can capture glitch behavior
- Validation against a set of SPICE simulation
- Error rates are consistently less than 11%, with most less than 3%
- The initial speed of simulation without optimization is **150 insts / sec**

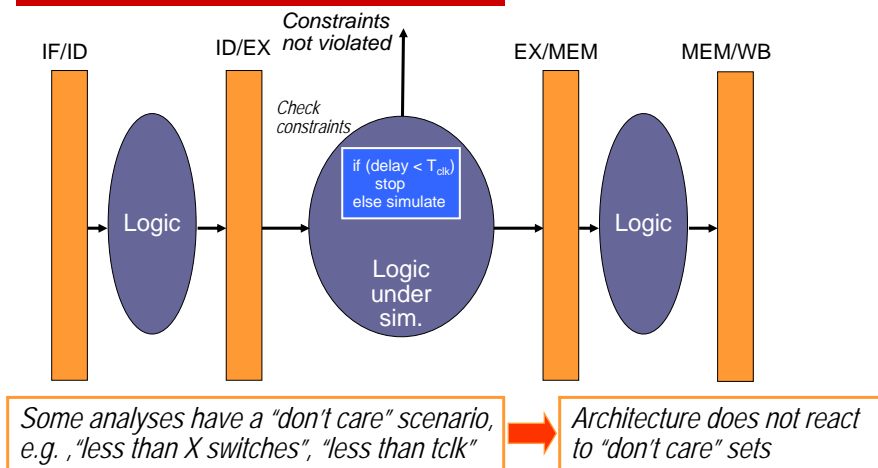
82

## Some serious performance boosters

- ❑ Constraint-based pruning
  - Static pruning
  - Dynamic pruning
- ❑ Circuit timing memoization  
(a.k.a. *Cashing of electrical simulation results*)
- ❑ SimPoints

83

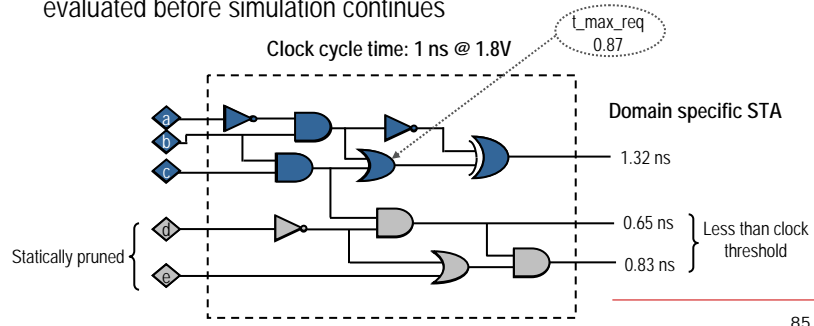
## Constraint-Based Pruning – An overview



84

## Static Constraint Pruning

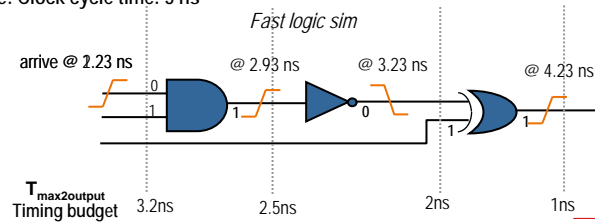
- At each new voltage and temperature, domain specific STA computes worst case values of the constraints measure and can be pruned where the constraints cannot be violated
- Whenever the supply voltage is changed, constraint based pruning is re-evaluated before simulation continues



## Dynamic Pruning

- During simulation, an event can be dropped if a particular input vector causes a transition such that:  $t_{\text{event}} + T_{\text{max2output}} < T_{\text{constraint}}$ 
  - Guarantees that simulation will reach output net without a timing violation
  - Must still perform logic simulation to compute circuit state

Example: Clock cycle time: 5 ns



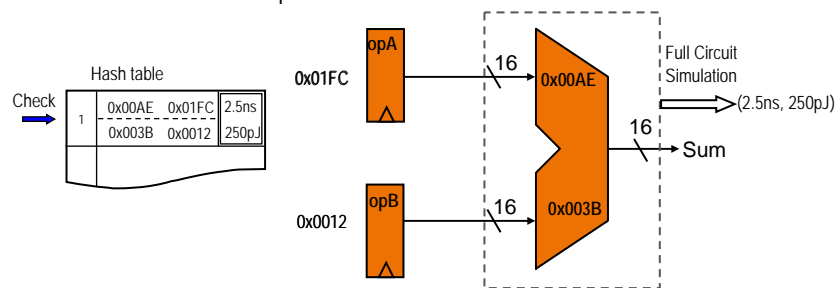
## Constraint-Based Circuit Pruning

- ❑ In our case study of 200Mhz Razor system
  - At 1.8V nominal voltage, pruning eliminated 64% of prime inputs of circuit
  - At most highly constrained voltage 1.4V, 24% of prime inputs of circuit is eliminated
- ❑ Static and Dynamic pruning achieve **445 instructions per second**

87

## Circuit Timing Memoization

- ❑ Remember previous circuit evaluations, reuse results if they recur
- ❑ Leverage value locality by recording switching history
  - Previous vector encodes the internal node values of circuit, input vector indicates new input transition



88

## Circuit Timing Memoization

---

- ❑ Size of hash table is limited by 256MB
- ❑ Dynamic reordering hash bucket chains
  - Bringing most recently referenced (MRU) element to the head of chain, reduces average number of hops
  - At most 50% hit rate on average
- ❑ Per-opcode input vector filtering mechanism
  - Observation:
    - load/store instructions ignore “operand B”
  - Each instruction opcode indicates with mask which do not influence stage logic evaluation
  - 70% hit rate on average

---

89

## SimPoint Analysis

---

- ❑ After marshalling all optimization technique, we achieve approximately **1000 instructions per second**
- ❑ We deploy a recent developed simulation sampling technique, SimPoint
  - Uses Basic Block Distribution Analysis to extract representative samples(10 million insts) of original benchmark (1 billion insts)
  - Drastically reduces the number of instructions observed to characterize the program's performance
  - Error analysis indicates an error of less than 10% (typically less than 3%) for a wide variety of benchmarks

***A full benchmark execution can be completed within 5 hours***

---

90

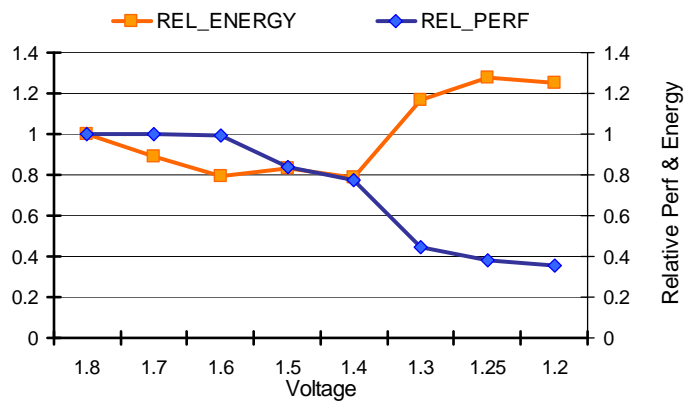
## Circuit-aware simulation to evaluate Razor

- ❑ Initial simulators utilized a hand-generated EX-stage circuit model
  - insufficient performance
- ❑ Challenge: instruction latency (in cycles) depends on circuit evaluation latency
  - Cycle count may vary with input, voltage, temperature, process variation
- ❑ *Circuit-Aware Architectural Simulation* combines architectural and circuit simulation
  - SimpleScalar architectural-level simulation
  - Gate-level timing simulation of per-stage logic blocks

91

## Case Study: Razor Timing Speculation

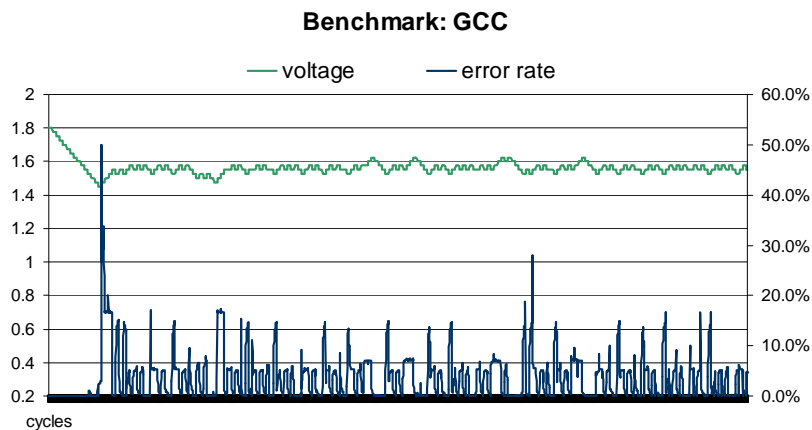
### Benchmark: GCC



92

## Case Study: Razor Timing Speculation

---



93

## Circuit-Aware simulation in summary

---

- ❑ Challenge is integration between architectural simulation and circuit simulation
  - Must balance fidelity and speed of simulation
- ❑ Three optimizations were utilized to enhance speed of simulation
  - Constraint-based Pruning
  - Circuit Simulation Memoization
  - SimPoint Simulation Sampling
- ❑ Demonstrated with Razor pipeline simulations
  - Razor architecture reacts cycle-by-cycle to circuit-level phenomenon

94

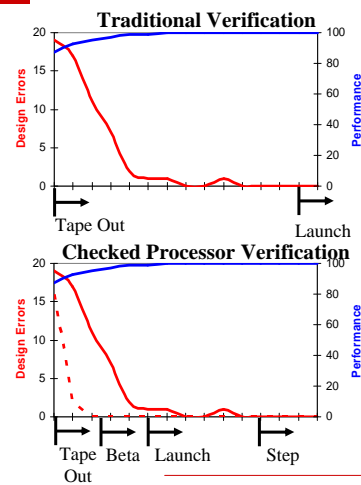
## Outline

- ❑ *Synthesis* - Typical-Case Optimized Adders
- ❑ *Performance/verification* - Circuit-Aware simulation
- *Verification* - Beta-release designs

95

## Beta-Release Designs

- ❑ Traditional verification stalls launch until debug complete
- ❑ Checked processor verification could overlap with launch
  - Beta-release when checker works
  - Launch when performance stable
  - Step as needed without recalls



96

## Additional CAD Opportunities

---

- ❑ For synthesis:
  - Typical-case library characterization (e.g., pdf of delay)
  - Synthesize design for target performance, power, etc...
  - TCO-style optimizations possible for macro-modules
- ❑ For verification:
  - Full formal verification for checker components
  - Profile-directed simulation-based verification for core
- ❑ For testing:
  - Checker component can facilitate software-based manufacturing test of core components

---

97

---

**Open discussion**

---

98

---

## Conclusion

99

## Conclusions

---

- ❑ Better than worst-case design abandons traditional worst-case design constraints
- ❑ Couples complex designs with checkers
- ❑ Enables CAD opportunities for typical-case optimization
- ❑ Requires tool support for observability, synthesis and verification

For more information:

<http://www.eecs.umich.edu/razor>

100

## References

---

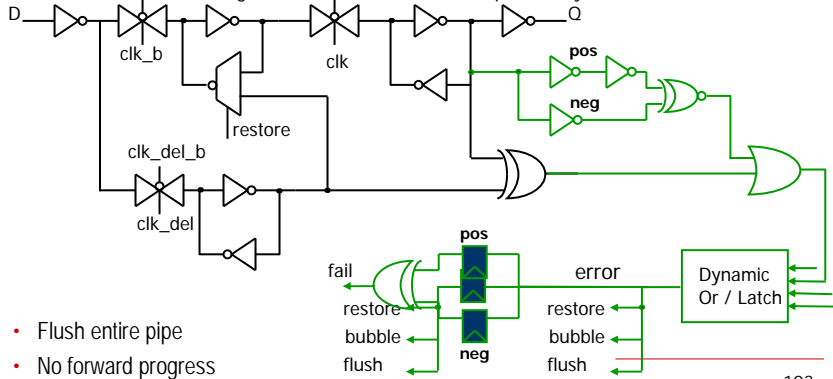
- Todd Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design," ACM/IEEE 32nd Annual Symposium on Microarchitecture (MICRO-32), November 1999.
- D. Ernst, N. S. Kim, S. Das, S. Pant, T. Pham, R. Rao, C. Ziesler, D. Blaauw, T. Austin, T. Mudge, and K. Flautner, "Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation," in the 36th Annual Int'l Symposium on Microarchitecture (MICRO-36), December 2003.
- Shanbhag, N.R., "Reliable and efficient system-on-chip design," Computer, Vol.37, Iss.3, Mar 2004.
- Uht, A.K., "Going beyond worst-case specs with TEAtime," Computer, Vol.37, Iss.3, Mar 2004.
- Austin, T.; Blaauw, D.; Mudge, T.; Flautner, K., "Making typical silicon matter with Razor," Computer, Vol.37, Iss.3, Mar 2004.
- S.-L. Lu, "Speeding up processing with approximation circuits," Computer, Vol.37, Iss.3, Mar 2004.
- Worm, F.; lenne, P.; Thiran, P.; DeMicheli, G., "A Robust Self-Calibrating Transmission Scheme for On-Chip Networks," IEEE Trans. on VLSI Systems, Vol. 13, Iss. 1, January 2005.
- T. Kehl. Hardware self-tuning and circuit performance monitoring, in Proceedings of International Conference on Computer Design, 1993.
- L. Anghel and M. Nicolaidis, "Cost reduction and evaluation of temporary faults detecting technique," in Proceedings of the conference on Design, automation and test in Europe (DATE-2000), March 2000.

---

## Supplemental Materials

## More Details on Meta-Stability

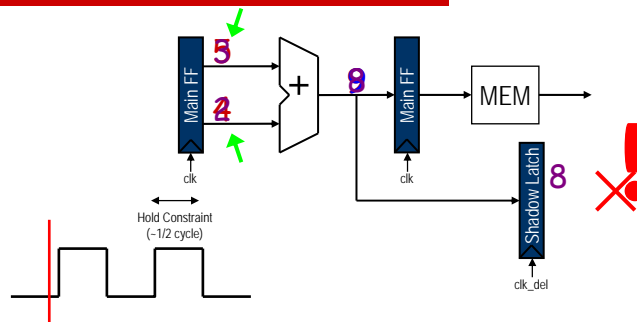
- Sub-critical operation invites meta-stability
  - Meta-stability detector itself can become meta-stable
  - double latch error signal to obtain sufficient small probability



- Flush entire pipe
- No forward progress
- Reduce frequency

103

## Razor Short Path Constraint

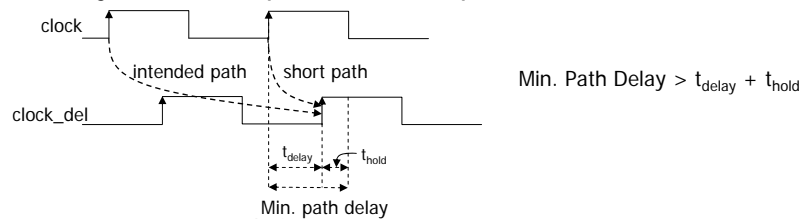


- Double-sampling metastability tolerant latches detect timing errors
  - Second sample correct-by-design, use guarantees forward progress
- Microarchitectural support restores correct program state
  - Timing errors treated in the same way as branch mispredictions

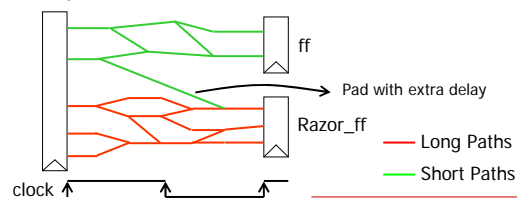
104

## Overcoming Short Path Constraints

- ❑ Delayed clock imposes a short-path constraint



- Razor necessary only for latches on slow paths
- Pad fast path for latches with mixed path delays
- Trade-off between DVS headroom and short path constraints



105

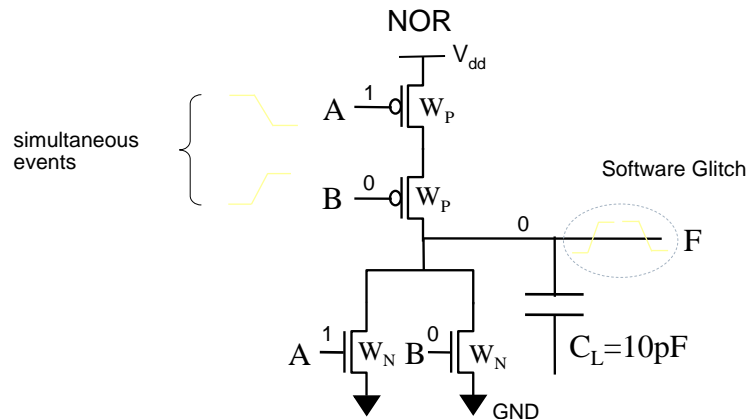
## Power Overhead of the Razor Flip-Flop

<i><b>Error Free Operation</b></i>	
Standard Flip-Flop Energy (static/switching)	49fJ / 125fJ
RFF Energy (static/switching)	60fJ / 203fJ
<i><b>Error Detection and Recovery</b></i>	
Energy of RFF per error event	260fJ

- ❑ 38% error-free latch overhead (assuming 20% switching activity)
- ❑ 42% latch overhead with errors (20% switching, 1% error rate)
- ❑ Overhead mitigated by latch-frugal architecture

106

## Simultaneous Events



Cancel a pair of close events that may cause software static glitch, which cannot occur in real circuits; *source of inaccuracy*

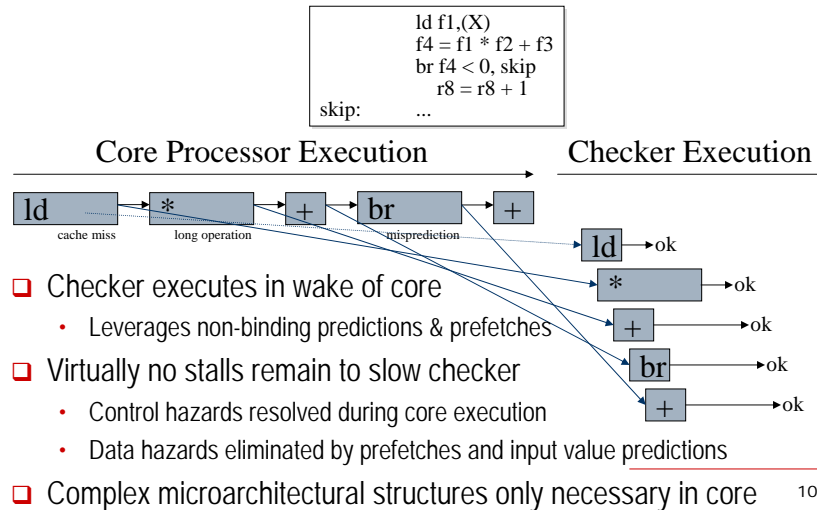
107

## Accuracy of Simulators

- ❑ Accuracy of simulation
  - Validation against a set of SPICE simulation with number of circuit topology at varied voltages and input slew rates
  - Error rates are consistently less than 11%, with most less than 3%
- ❑ The initial speed of simulation without optimization is 150 instructions per second (comparable to VCS)
- ❑ VCS stands for Verilog Compiler Simulator

108

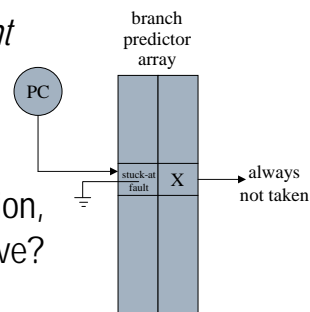
## Speeding the Checker with Core Computation



109

## Motivating Observations

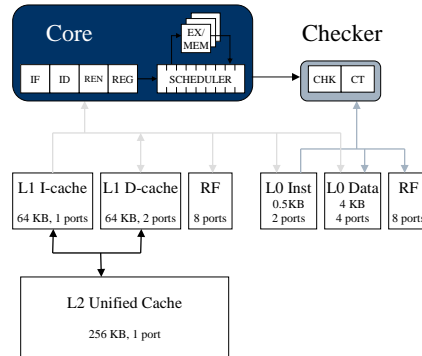
- ❑ Speculative execution is *fault-tolerant*
  - Design errors, timing errors, and electrical faults only manifest as performance divots
  - Correct checking mechanism will fix errors
- ❑ What if all computation, communication, control, and progress were speculative?
  - Any incorrect computation fixed
    - *maximally speculative*
  - Any core fault fixed
    - *minimally correct*



110

## Optimized System Architecture

- ❑ Performance impacts eliminated
  - Checker RF allows core commit
  - No storage hazards
  - Few checker cache misses
  - Less expensive core storage architecture (same as baseline)
- ❑ Core cache failures affect checker



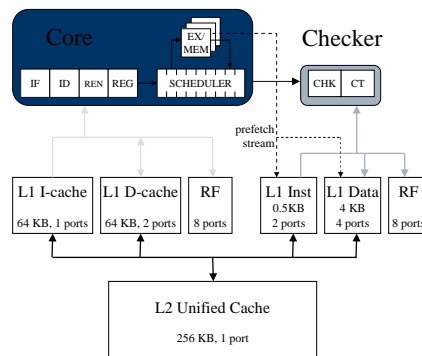
### Slowdowns

**-0.4%** Best: -3.2%  
Worst: 0.2%

111

## Fully Decoupled System Architecture

- ❑ Checker fully decoupled
  - Core L1 caches may fail
  - All L2 writebacks from checker
  - Core caches flushed on fault
  - Core accesses and misses warm up checker caches
- ❑ Eliminates common mode core cache failures
  - But, generates more L2 traffic
  - Further optimizations possible



### Slowdowns

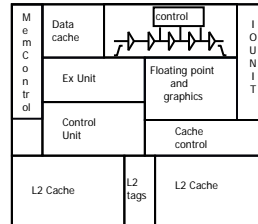
**1.2%** Best: 0%  
Worst: 6.7%

112

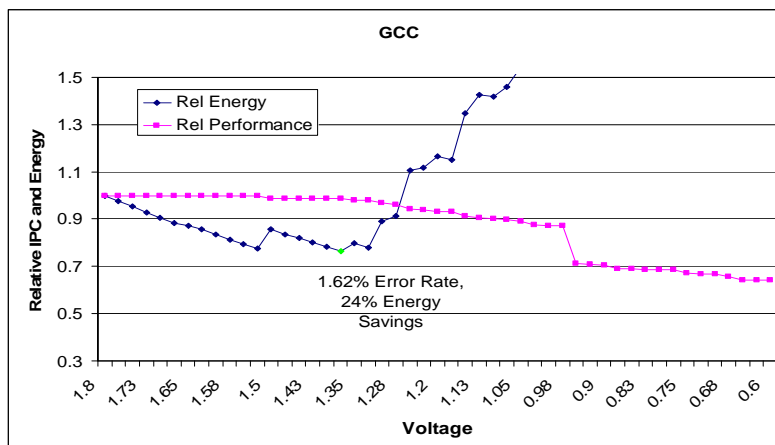
# Intelligent Energy Management

## Slack detector

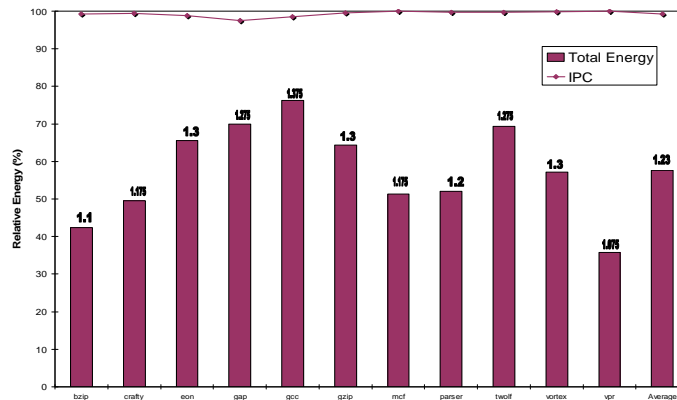
- Automatic tuning mechanism
  - ARM's *Intelligent Energy Manager (IEM)*
  - Processor voltage automatically tuned to external ambient conditions
  - Inverter chain designed to track most restrictive critical path, margin still required



# EX-Stage Analysis – Optimal Voltage Sweep

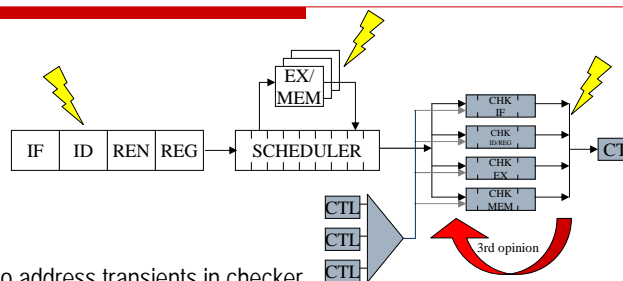


## Simulation Results: Energy-Optimal Voltage



115

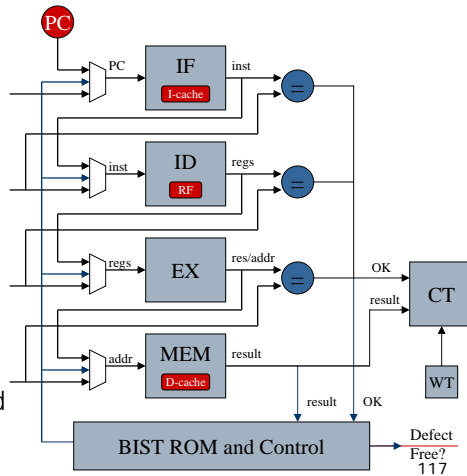
## Low-Cost SER and Noise Protection



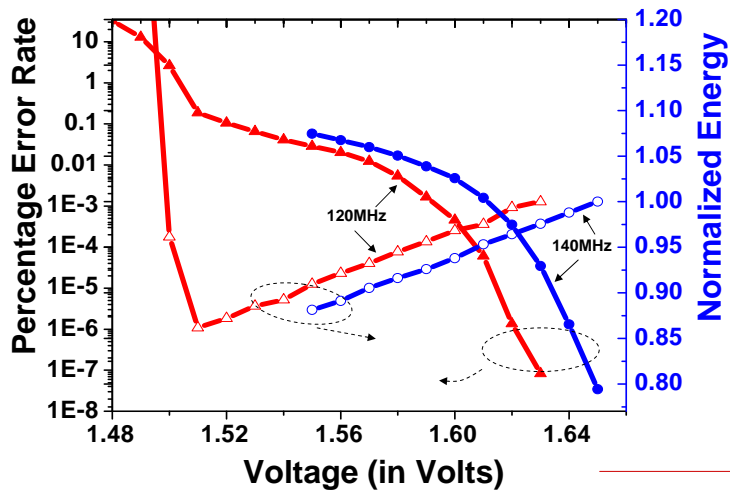
- ❑ Only need to address transients in checker
  - Checker detects and corrects noise-related faults in core
  - Core processor designed without regard to strikes (e.g., no ECC...)
- ❑ Recycle checker inputs suspected core fault
  - If no error on third execution, transient strike in checker processor
  - If error on third execution, core processor fault occurred (e.g., SER, design error)
- ❑ Protect critical checker control with triple-modular redundant (TMR) logic
  - TMR on simple control results in only 1.3% larger checker (synthesized design)<sub>1,6</sub>

## Fully Testable Microprocessor Designs

- ❑ Checker structure facilitates manufacturing tests
  - All checker inputs exposed to built-in-self-test logic
  - Checker provides built-in test signature compression
- ❑ Checker can be fully tested with small BIST module
  - less than 0.5% area increase
- ❑ Reduces burden of testing on core
  - Missed core defects corrected
  - Checker acts as core tester



## Error Rate and Normalized Energy Savings for Chip1



## Measured Power and Energy

---

120MHz 27C	Point of First Failure		Point of 0.1% Error Rate	
	Power (mW)	Energy per Instruction (Power/IPC/Freq) (pJ)	Power (mW)	Energy per Instruction (Power/IPC/Freq) (pJ)
Chip1	104.5	870	89.7	740
Chip2	119.4	990	99.6	830