Robust Systems
From Today to the \textit{N3XT 1,000X}

Subhasish Mitra

Department of EE & Department of CS
Stanford University
Computational demands exceed Processing capability
World Relies on Computing

- Ensure robust operation
- Meet computation demands
- New application horizon
Research Topics

- **Robust** operation
  - Bugs, reliability, security

- **Revolutionize** nanosystems
  - 1,000X opportunity

- **Program** human brain
  - SNI Big Ideas in Neuroscience initiative
Outline

- Robust operation: silicon CMOS reliability
- Beyond silicon
- Conclusion
Silicon CMOS Reliability Challenges

- Radiation-induced soft errors
  - **Fatal** flip-flop errors

- Early-life failures (ELF)
  - Burn-in: difficult, expensive

- Variations: $V_{dd}$, thermal, circuit aging
  - Worst-case guardbands expensive
Definitions

- **Malfunction** (often referred to as **failure**)
  - Deviation from specified behavior
  - Underlying cause: **failure**

- **Error**: incorrect signal value

- **Fault model**
  - (Logic) representation of effect of failure
System Output Response to Failure

- **Error on output**: non-critical apps. (e.g., games)

- **Fault-secure**: correct outputs or error indication
  - Retry adequate (e.g., banks)

- **Fault masked**: correct outputs
  - Fault in specified class (e.g., spacecraft)

- **Fail safe**: correct or “safe” outputs
Definitions

- **Reliability:** $R(t)$
  - Probability system works correctly up to time $t$

- **Exponential model**
  - $R(t) = e^{-\lambda t}$, $\lambda =$ failure rate

- **Mean Time to Failure (MTTF)**

$$MTTF = \int_{0}^{\infty} t \times R(t) \, dt = \frac{1}{\lambda}$$
Definitions

- **Availability: \( A(t) \)**
  - Probability system works correctly at time \( t \)

- Assume: system repaired after failure
  - Mean Time to Repair (MTTR)

- Steady-state availability: \( \frac{MTTF}{MTTF + MTTR} \)

- How to improve availability?
Error Effects: Vanished

Output
File, display, ...

Golden output
File, display, ...

Start → Finish

Output File, display, … = Golden output File, display, …
Error Effects: Output Mismatch

Silent data corruption (SDC)

Start → Finish

No error indication

Output
File, display, ...

≠

Golden output
File, display, ...

≠
Silent Data Corruption (SDC)

- Error-free
- Output Mismatch

- Output file incorrect
- No error indication
Error Effects: Unexpected Termination

- Divide-by-zero
- Memory access violation
- Application-detected errors
- …
Hang

Start

> 2 × error-free execution time

Does not finish / terminate
Soft Error Effects

- Vanished: 91.2%
- Output Mismatch: 0.8%
- Unexpected Termination: 7.7%
- Hang: 0.3%

All injected flip-flop errors

Detected but Uncorrected Errors (DUE)

[Cho DAC 13]
Soft Error Effects: BZip2 on IBM Power6

- Injected errors
  - Vanished: 95.7%
  - Recovered: 3.5%
  - Checkstop: 0.6%
  - Incorrect architecture state: 0.2%

- No impact: 35%
- Software detected: 50%
- SDC: 15%

[Sanda 08] (graphic contributed by Dr. Pia Sanda, ex-IBM)
Fault-Tolerance: Rich Literature

Expensive

Early pioneers

Early systems
How Low Cost?

**Applications**
- No
- Yes

**Approach**
- **Low-cost** detection / correction
  - BISER, LEAP
- Circuit failure prediction
- **Light-weight** correction
  - Error Resilient System Arch. (ERSA)
  - [Cho IEEE TCAD 12]
Low-Cost Techniques

Circuit Failure Prediction
CASP on-line self-test, diagnostics, self-repair & adaptation

Burn-in challenging

Soft Error Resilience
BISER + LEAP: Errors reduced: 1,000X

Circuit aging margins expensive

Early-life failures (ELF)

Lifetime

Wearout

Time
How Low Cost?

Solution: cross-layer resilience?

“multiple error resilience techniques from different layers of the system stack cooperate to achieve cost-effective error resilience”

[DARPA, PERFECT BAA 12]

[Borkar Intel, IEEE Micro 05]

[Pedram, NSF 12]

[Gupta IBM, IRPS 14]

[Carter Intel, DATE 10]

[Chandra ARM, DAC 14]

[Henkel, DAC 14]
Existing Resilience Techniques

- Many point solutions
  - Some cross-layer, some single-layer
- Missing
  - End-to-end cross-layer resilience framework
CLEAR
Cross-Layer Exploration for Architecting Resilience

Resilience Library

Application
Software
Architecture
Logic
Circuit

Reliability Analysis / Execution Time Evaluation

Emulation cluster
Stampede supercomputer

Layout Evaluation

28nm
Library cells
SRAM compiler

Synopsys
Design Compiler
PrimeTime
IC Compiler

RTL: ARM, IVM, Leon, OpenSPARC T2 SoC, uncore, accelerators

Reliability, Area, Power, Energy, Clock Frequency, Application Runtime

[Cheng DAC 16]
Today’s Focus

- Radiation-induced soft errors in flip-flops
  - Single-Event Upsets (SEUs)
  - Single-Event Multiple Upsets (SEMUs)
- Combinational logic soft errors not critical
CLEAR: Extensive Study

- Designs: wide variety
  - ARM, LEON3, Alpha, OpenSPARC multi-core SoC, accelerators

- Thorough **flip-flop** error injections
  - FPGA clusters, Stampede supercomputer (522,080 cores)
  - Full workloads (SPEC, PARSEC, PERFECT, proprietary)

- Detailed physical design
  - Wire routing, process / voltage / temperature corners
Many Cross-Layer Questions Answered

- Cross-layer always best?
- All cross-layer solutions equally good?
- Application constraints (e.g., soft real-time)?
- Benchmark dependence?
- Definitive guidelines for new resilience techniques
Key Message

- From black art to science

- 5-50x resilience, 0.2-6% energy cost
  - Circuit + logic + micro-arch. recovery

- Circuit alone (application-guided)
  - ~1% extra energy vs. best cross-layer
CLEAR
Cross-Layer Exploration for Architecting Resilience

Resilience Library

Application
Software
Architecture
Logic
Circuit

Reliability Analysis / Execution Time Evaluation

Emulation cluster

Stampede supercomputer

RTL: ARM, IVM, Leon, OpenSPARC T2 SoC, uncore, accelerators

Layout Evaluation

28nm
Library cells
SRAM compiler

Synopsys
Design Compiler
PrimeTime
IC Compiler

Reliability, Area, Power, Energy, Clock Frequency, Application Runtime
### Representative Resilience Techniques

| Algorithm | 1. Algorithm Based Fault Tolerance (ABFT) Correction  
| 2. ABFT Detection  
| Software | 3. Software Assertions  
| 4. Control Flow Checking by Software Signatures (CFCSS)  
| 5. Error Detection by Duplicated Instructions (EDDI)  
| Logic | 6. Data Flow Checking (DFC)  
| 7. Monitor Cores  
| Circuit | 8. Logic Parity  
| 9. Layout design through Error-Aware transistor Positioning (LEAP)  
| 10. Error Detection Sequential (EDS)  
| Micro-arch. Recovery | 1. Flush  
| 2. Reorder Buffer (RoB)  
| 3. Instruction Replay (IR)  
| 4. Extended IR (EIR)  

**10 error detection/correction techniques + 4 recovery techniques = 798 combinations**
BISER: Built-In Soft Error Resilience

45nm: up to 1,000X benefits

[IEEE Computer 05, TVLSI 06]
Single Error Assumption Inadequate

- Single-event **multiple** upsets (SEMUs)

**LEAP:** Layout by Error Aware transistor Positioning

Errors **Corrected:** SEUs and SEMUs

[Lee IRPS 10, Lilja IEEE TNS 13]
Extensive LEAP Characterization

- Radiation beam experiments
  - 40nm, 28nm, 20nm, 14nm
  - Bulk, SOI
- VDD: nominal, near-threshold

<table>
<thead>
<tr>
<th>Flip-flop</th>
<th>Soft Error Rate (SER)</th>
<th>Area</th>
<th>Power</th>
<th>Delay</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LEAP-DICE</td>
<td>$2 \times 10^{-4}$</td>
<td>2</td>
<td>1.8</td>
<td>1</td>
<td>1.8</td>
</tr>
</tbody>
</table>

[Lee IRPS 10; Lilja IEEE T. Nucl. Sci. 13, SEE 16; Quinn NSREC 15, REDW 15; Turowski SEE 15]
Memory ECC and SEMUs

- Don’t implement multiple error correction blindly
- Multiple *physically adjacent* errors
Memory ECC and SEMUs

Option 1

- Memory interleaving
  - 2 physically adjacent errors
    - Single errors in 2 separate words

- Cost, difficult for smaller geometries

Option 2

- Adjacent bit error correction [Dutta ITC 07]
Memory ECC Challenges

- Performance overhead
  - Pipelining
    - Additional latency, verification effort
  - Detection followed by correction
    - Variable latency, verification effort
- Small distributed memories
Error Masking

- No error on outputs
  - **Triple Modular Redundancy (TMR)**

![Diagram showing a voter with three modules connected to it.](image-url)
TMR Reliability

\[ R_{TMR} = R_{voter} \times [R_m^3 + R_m^2 \times (1 - R_m)] \]

- \( R_m \): individual module reliability
- Pessimistic: non-overlapping errors
- Optimistic: correlated / common-mode failures
- TMR MTTF < Simplex MTTF
TMR reliability = simplex reliability

- Time = \( \log_e 2 \times \) Simplex MTTF (perfect voter)
- TMR effective for “short” mission times
- Other options: TMR-Simplex, TMR + Duplex-Repair
Concurrent Error Detection (CED)

- Normal system operation
- Preserve data integrity
  - Correct outputs or
  - Error indicated
    - Incorrect outputs
  - aka fault-secure
Output “Characteristics”

- Output itself: duplication
  - Major challenges if not “fine-grained”
- Output parity
- Output residue
- 1s or 0s count in output word
- Many others (extensive literature)
- Self-checking checkers
Processor Duplication Challenges

- Synchronization !!
- False DUEs when out of sync
  - e.g., error correction event in one processor
  - Mismatch when output pins compared
Single-Bit Logic Parity Prediction

- $P = Z_1 \oplus Z_2 \oplus \ldots \oplus Z_m$

- Disjoint output logic (no logic sharing)
  - Only for combinational logic errors

![Diagram showing parity checker and logic](image)
Multiple-Bit Logic Parity Prediction

- Main purpose: cost reduction (sharing, routing, logic)
  - Can be still expensive

![Parity Checker Diagram]

\[
P_1 = Z_1 \oplus Z_2, \quad P_2 = Z_3 \oplus Z_4
\]
Logic Parity Checking: Flip-Flop Errors

- Logic parity
  - Original design
  - Parity logic
  - Pipeline flip-flops

- Maintain clock period

- Parity group

- Checker

- Predictor

<table>
<thead>
<tr>
<th>Logic parity: Naïve</th>
<th>200 MHz clock speed impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic parity: Incorrect heuristic</td>
<td>80% additional energy impact</td>
</tr>
<tr>
<td>Logic parity: CLEAR heuristic</td>
<td>No clock speed impact</td>
</tr>
<tr>
<td></td>
<td>Minimal energy impact</td>
</tr>
</tbody>
</table>

Parameters: parity size, flip-flop vulnerability, floorplan location, timing path slack, etc.
Parity Prediction for Datapath Circuits

- \( S = A + B \) (\( n \)-bit operation)

- Parity (\( S \)) = \( S_1 \oplus S_2 \oplus S_3 \oplus \ldots \oplus S_n \)
  
  \[
  = (A_1 \oplus B_1 \oplus C_1) \oplus (A_2 \oplus B_2 \oplus C_2) \ldots \oplus (A_n \oplus B_n \oplus C_n) \\
  = (A_1 \oplus A_2 \ldots \oplus A_n) \oplus (B_1 \oplus B_2 \ldots \oplus B_n) \oplus (C_1 \oplus C_2 \ldots \oplus C_n) \\
  = \text{Parity (internal carries)}
  \]

- Parity (internal carries) expensive
  - Several strategies for high-performance adders
Residue Codes for Datapath

- $y = x \mod b$: $y$ is residue of $x$ (modulo $b$)
- Residue $(A + B) = \text{Residue } (A) + \text{Residue } (B)$
- Residue $(A \times B) = \text{Residue } (A) \times \text{Residue } (B)$
- Choice of $b$: Mersenne prime (form $2^m-1$)
  - Coverage, checker complexity
- Issues: bit-wise logic, operand residue, checker cost
  - Often used for multipliers
Application-Specific CED

- LZ compression: loss-less, invertible
  - Compression: complex
  - Decompression: simple

9% area overhead, 0.5% delay overhead [Huang 00]
Error Detection Sequential (EDS)

Slide obtained from K. Bowman, J. Tschanz, et al., Intel
Errors in Processors

- **V_1**: Control flow error
  - Incorrect instruction sequence

- **V_2**: Computational error
  - Incorrect computation

- **V_3**: Memory error
  - Incorrect value or address

- **V_4**: Control flow error
  - Incorrect instruction sequence

Example code:
```
mov r1, 0
mov r2, 1
XOR mov r3, 5
```
Program Representation: Control Flow Graph

\[ \text{i=0; x=1; y=5;} \]
\[ \text{While (i < 10) { } } \]
\[ \text{z = x + y * i; } \]
\[ \text{i = i + 1; } \]
\[ \{ \]
\[ \text{Basic Block (BB)} \]

\[ \text{mov r1, 0} \]
\[ \text{mov r2, 1} \]
\[ \text{mov r3, 5} \]
\[ \text{L1:} \]
\[ \text{inc r1} \]
\[ \text{bge r1, 10, L2} \]
\[ \text{mul r4, r3, r1} \]
\[ \text{add r5, r2, r4} \]
\[ \text{br L1} \]

\[ \text{L2:} \]
\[ \ldots \]

\[ \text{V1} \]
\[ \text{V2} \]
\[ \text{V3} \]
\[ \text{V4} \]
SIHFT

- Software Implemented Hardware Fault Tolerance
  - Automated by compiler
  - EDDI [Oh, IEEE Trans. Reliability 02]
  - CFCSS [Oh IEEE Trans. Reliability 02b]
  - EDI4 [Oh IEEE Trans. Computers 02]
  - Lots of recent publications
Error Detection using Duplicated Instructions

- Duplicate instructions inside basic blocks
  - Different registers
- Duplicate data structures
- Comparison before memory stores
- Performance penalty 13% - 111%
  - Reduced by Instruction Level Parallelism (ILP)
## EDDI Example

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADD R3, R1, R2</td>
<td>; R3 ← R1 + R2 master</td>
</tr>
<tr>
<td>ADD R23, R21, R22</td>
<td>; R23 ← R21 + R2 shadow</td>
</tr>
<tr>
<td>MUL R4, R3, R5</td>
<td>; R4 ← R3 * R5 master</td>
</tr>
<tr>
<td>MUL R24, R23, R25</td>
<td>; R24 ← R23 * R25 shadow</td>
</tr>
<tr>
<td>BNE R4, R24, Error_Handler</td>
<td>; compare</td>
</tr>
<tr>
<td>ST 0(SP), R4</td>
<td>; store master result</td>
</tr>
<tr>
<td>ST offset(SP), R24</td>
<td>; store shadow result</td>
</tr>
</tbody>
</table>
EDDI Design Choices

- Check after each instruction ?
- Storeless basic blocks (SBB) ?
  - No branch or store except final instruction
- Why SBB ?
  - Correctness defined by program output
  - Erroneous branches: stores skipped ?
    - Check at branches too
CFCSS

- Control Flow Checking using Software Signatures

- Each node
  - Unique signature
- Each edge
  - Transition between 2 signatures
  - Difference function: XOR

\[ d_{12} = s_1 \text{xor} s_2 \]
CFCSS

- Runtime signature $G$
- Basic block $i$ to $j$
  - $G = s_i \text{ XOR } d_{i,j}$
  - Check $G = s_j$

$G_n$ run-time signature at node $V_n$
$s_n$ signature assigned to node $V_n$
$d_n$ signature difference
CFCSS Implementation

- Global variable $G$ holds run-time signature
- Compute & check signature: start of each basic block
CFCSS: Branch Fan-in

- Basic block with multiple predecessors
- Run-time adjusting signature $D$ differentiates fan-in
Error Detection using Diverse Data & Duplicated Instructions

- Duplicated instructions, *data diversity*
  - Expressions in shadows multiply by $k$ (-1, -2, ...)

- $k = -2$: good choice

- Transient errors & most permanent faults detected

- Issues: floating point, pointers
SIHFT Results [Lovelette 02]

- COTS in space: no hardware redundancy
- ARGOS satellite experiment
  - Compare rad-hard processor vs. COTS
- Undetected errors in rad-hard processor
- COTS: 5.55 SEUs / Mbyte / day, 99.7% coverage
- 98.8% successful recovery: software ECC + restart
- COTS + SIHFT: faster than rad-hard
Multi-Threading for CED

- Same application computed by two threads
  - [Rotenberg 99, Saxena 00, Mukherjee 02]

![Diagram](attachment://multi-threading_diagram.png)
What About Recovery?

**Instruction Replay (IR)**

**Flush**

<table>
<thead>
<tr>
<th></th>
<th>Instruction Replay</th>
<th>Flush recovery</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overhead for recovery hardware</td>
<td>16% area, 21% energy</td>
<td>0.6% area, 0.9% energy</td>
</tr>
<tr>
<td>Recovery latency</td>
<td>47 cycles</td>
<td>7 cycles</td>
</tr>
</tbody>
</table>

**Style 1:**

"Instruction Replay"

**Style 2:**

"Flush"

- Cross-layer protected
- LEAP-DICE protected
- Recovery logic
CLEAR
Cross-Layer Exploration for Architecting Resilience

Reliability Analysis / Execution Time Evaluation

Emulation cluster

Stampede supercomputer

RTL: ARM, IVM, Leon, OpenSPARC T2 SoC, uncore, accelerators

RTL:

Layout Evaluation

28nm Library cells

Synopsys Design Compiler

PrimeTime

IC Compiler

Reliability, Area, Power, Energy, Clock Frequency, Application Runtime

Resilience Library

Application

Software

Architecture

Logic

Circuit
Radiation-Induced Soft Errors

Soft errors

Radiation beam testing

Flip-flop error injection

The Los Alamos Neutron Science Center

Simulation / Emulation

≈

Bit-flips

0 ↔ 1
Soft Error Injection

Simulation speed

High-level error injection

Flip-flop
$10^2$ cycles / sec

Architectural register
$10^7$ cycles / sec

Program variable
$10^9$ cycles / sec
Soft Error Injection

Accuracy

Ground truth

High-level error injection

Flip-flop
$10^2$ cycles / sec

Architecture register
$10^7$ cycles / sec

Program variable
$10^9$ cycles / sec
Silent Data Corruption (SDC)

- Error-free
- Output Mismatch
  - Output file incorrect
  - No error indication

Detected but Uncorrected Error (DUE) also considered
Perils of Inaccurate Estimation

**Overestimation**

SDC

**Underestimation**

SDC

Overprotection

Unhappy customers
Little Prior Work

Error injection studies

Quantified comparison

[Rimen FTCS 94]
[Rebaudengo IOLTW 02]
What We Found

- Naïve high-level injections **highly** inaccurate

- How inaccurate?
What We Found

- **Naïve high-level injections** highly inaccurate

- **How inaccurate?**

  - Designs: LEON3 (in-order, single-issue), ALPHA (out-of-order, superscalar)
  - Applications: SPEC 2000
  - Error injection samples: 6 million

[Cho DAC 13]
Inaccuracy Quantification

Undetected output error rate

0.8%

1x

6x

13x

Flip-flop

Architectural register

Program variable
What We Found

- Naïve high-level injections **highly** inaccurate

- How inaccurate?
  - Up to 45X
  - Neither optimistic nor pessimistic

[Cho DAC 13]
What We Found

- Naïve high-level injections highly inaccurate

- How inaccurate?
  - Up to 45X
  - Neither optimistic nor pessimistic

- Why inaccurate?
  - Only 3% flip-flop error propagations modeled

[Cho DAC 13]
Uncore Components

OpenSPARC T2 SoC

- Uncore: 12%
- Memories: 76%

[Li ITC 13]

Intel i7 quad-core SoC

- Uncore: 39.8%
- Processor cores: 60.2%

[Gupta USENIX 12]
Existing Work

Errors in processor cores

Errors in uncore?

HUNDREDS of publications

Distribution Statement A: Approved for Public Release, Distribution Unlimited
Uncore Soft Errors: First Extensive Study

- New error injection: fast & accurate
  - 20,000x speedup vs. RTL

- Reliability impact: uncore ≈ processor cores
  - BUT, long error propagation latency
Lots of CLEAR Results

798 total combinations

Energy cost (%)

% SDCs covered
High-Level Enough?

Silent Data Corruption (SDC) Rate

- Data-flow check
- Monitor cores
- Assertions
- CFCSS
- EDDI
- ABFT correct
- ABFT detect

- Architecture
- Software
- Algorithm

Improvement:
- 1x
- 19x
- 38x

Circuit Logic Architecture Software Algorithm

0 2 4 6 8

Silent Data Corruption (SDC) Rate
Cross-Layer Combinations

Error bar: 0.5% energy cost (additive)
Cross-Layer Combinations

LEON3 energy cost (%)

SDC improvement

LEAP-DICE + logic parity + flush recovery

Error bar: 0.5% energy cost (additive)
Cross-Layer Combinations

Error bar: 0.5% energy cost (additive)
Cross-Layer Combinations

LEON3 energy cost (%)

SDC improvement

Error bar: 0.5% energy cost (additive)

LEAP-DICE + logic parity + flush recovery
Cross-Layer Combinations

LEON3 energy cost (%)

SDC improvement

LEAP-DICE + logic parity + flush recovery

LEAP-DICE + logic parity: special optimization

Error bar: 0.5% energy cost (additive)
Architecture & Software: Too Expensive

LEON3 energy cost (%)

SDC improvement

2x 5x 50x

EDDI + …

DFC + …

CFCSS + …

Assertions + …
Circuit-only (Application-guided): Highly Effective

LEON3 energy cost (%)

SDC improvement

LEAP-DICE + logic parity + flush recovery
Circuit-only (Application-guided): Highly Effective

Circuit-only: LEAP-DICE error correction (application-guided insertion)

LEON3 energy cost (%)

SDC improvement

LEAP-DICE + logic parity + flush recovery

Algorithm
Software
Architecture
Logic
Circuit
Circuit-only (Application-guided): Highly Effective

Circuit-only: LEAP-DICE error correction (application-guided insertion)

LEAP-DICE + logic parity + flush recovery
Circuit-only (Application-guided): Highly Effective

Circuit-only: LEAP-DICE error correction
(application-guided insertion)

LEON3 energy cost (%)

SDC improvement

LEAP-DICE + logic parity + flush recovery
Circuit-only (Application-guided): Highly Effective

- **Circuit-only: LEAP-DICE error correction**
  - (without application-guided insertion)

- **Circuit-only: LEAP-DICE error correction**
  - (application-guided insertion)

- **LEAP-DICE + logic parity + flush recovery**

- LEON3 energy cost (%)

- SDC improvement
What About ABFT?

Circuit-only: LEAP-DICE error correction (application-guided insertion)

ABFT challenging for general-purpose cores

LEON3 energy cost (%)

SDC improvement

- LEAP-DICE + logic parity + flush recovery
- ABFT correction + LEAP-DICE + logic parity + flush recovery
CLEAR Insights

- Hidden costs & inefficiencies
- Implementation matters
- Inaccurate analysis
Example: “Hidden” Costs

Error Detection Sequential (EDS) + recovery

Not just flip-flop overhead
Routing, recovery impact

<table>
<thead>
<tr>
<th>Flip-flop</th>
<th>Area</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LEAP-DICE</td>
<td>2</td>
<td>1.8</td>
</tr>
<tr>
<td>EDS</td>
<td>1.5</td>
<td>1.4</td>
</tr>
</tbody>
</table>

LEON3 energy cost (%) vs SDC improvement

2x | 5x | 50x |

SDC improvement: LEAP-DICE (errors corrected)
Example: Inefficiencies

- Few Flip-flops Protected (Data Flow Checking (DFC): 57%)
- Low SDC Coverage per Flip-flop (Data Flow Checking (DFC): 30%)

Result: Low SDC Improvement

[Meixner, MICRO 07]
Example: Implementation Matters

Logic parity
- Original design
- Parity logic
- Pipeline flip-flops

Parameters: parity size, flip-flop vulnerability, floorplan location, timing path slack, etc.

<table>
<thead>
<tr>
<th>Logic parity</th>
<th>200 MHz clock speed impact</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic parity: Naïve</td>
<td>200 MHz clock speed impact</td>
</tr>
<tr>
<td>Logic parity: Incorrect heuristic</td>
<td>80% additional energy impact</td>
</tr>
<tr>
<td>Logic parity: CLEAR heuristic</td>
<td>No clock speed impact</td>
</tr>
<tr>
<td></td>
<td>Minimal energy impact</td>
</tr>
</tbody>
</table>

- maintain clock period
- parity group
- checker
- predictor

Logic parity: Naïve

Logic parity: Incorrect heuristic

Logic parity: CLEAR heuristic
Example: Inaccurate Analysis

- Software assertions: SDC improvement
  - Prior publications: 3.9x
    - Inaccurate error injection
  - Accurate analysis: 1.5x

<table>
<thead>
<tr>
<th></th>
<th>Flip-Flop error injection</th>
<th>Register Uniform error injection</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDC improvement</td>
<td>1.5x</td>
<td>4.8x</td>
</tr>
</tbody>
</table>

[Sahoo DSN 08, Hari DSN 12]
How About Benchmark Dependence?

- 50 \(<training, evaluation>\) pairs
  - Training: 4 SPEC, Evaluation: 7 SPEC

<table>
<thead>
<tr>
<th></th>
<th>Trained SDC improvement</th>
<th>Evaluated SDC improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5x</td>
<td>50x</td>
</tr>
<tr>
<td>Trained SDC improvement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluated SDC improvement</td>
<td>4.8x</td>
<td>39x</td>
</tr>
</tbody>
</table>

Add “lightweight” hardening (e.g., LHL)

<table>
<thead>
<tr>
<th>Extra energy cost (additive)</th>
<th>2%</th>
<th>1%</th>
<th>0.8%</th>
</tr>
</thead>
<tbody>
<tr>
<td>Final SDC improvement</td>
<td>19x</td>
<td>152x</td>
<td>1,326x</td>
</tr>
</tbody>
</table>
## Light-Hardened LEAP (LHL)

<table>
<thead>
<tr>
<th>Flip-Flop</th>
<th>Soft Error Rate (SER)</th>
<th>Area</th>
<th>Power</th>
<th>Delay</th>
<th>Energy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Baseline</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>LHL</td>
<td>$2.5 \times 10^{-1}$</td>
<td>1.2</td>
<td>1.1</td>
<td>1.2</td>
<td>1.3</td>
</tr>
<tr>
<td>LEAP-DICE</td>
<td>$2 \times 10^{-4}$</td>
<td>2</td>
<td>1.8</td>
<td>1</td>
<td>1.8</td>
</tr>
</tbody>
</table>
Target for Future Resilience Techniques

LEON3 SDC Improvement

IVM SDC Improvement

Energy Cost (%)

Target

LEAP-DICE error correction

LEAP-DICE + logic parity + flush recovery
Outline

- Robust operation: silicon CMOS reliability
- Beyond silicon
- Conclusion
THE FUTURE OF COMPUTING PERFORMANCE

Game Over or Next Level?

Samuel H. Fuller and Lynette I. Millett, Editors
Improve Computing Performance

System integration

Device performance
Option 1: Better Transistors

- Few experimental demos
- Transistors ≠ system
Option 2: Design Tricks

- Limited "tricks"
- Complexity $\rightarrow$ design bugs
Improve Computing Performance

Target: $1,000 \times$ performance

New innovations required

System integration

Multi-cores

Power management

Device performance
Solution: NanoSystems

Transform new nanotech into new systems enable new applications

New devices
New fabrication
New sensors
Abundant-Data Explosion

“Swimming in sensors, drowning in data”

- Mine, search, analyze: near real-time
  - Data centers, mobile phones, robots
Today’s System Bottlenecks

- **Separate** compute & memory chips
- Not enough on-chip memory
- Capacity & bandwidth critical
Abundant-Data Applications

Huge memory wall: processors, accelerators

Energy Measurements

Genomics classification
- 95%
- 5%

Natural language processing
- 82%
- 18%

Intel performance counter monitors 2 CPUs, 8-cores/CPU + 128GB DRAM
Abundant-Data Applications

Huge memory wall: processors, accelerators

Deep Learning Accelerators

- AlexNet (CNN): 15% Compute, 85% Memory
- ResNet-152 (CNN): 20% Compute, 80% Memory
- Language Model (LSTM): 8% Compute, 92% Memory
Nano-Engineered Computing Systems Technology

Energy-Efficient Abundant-Data Computing: The N3XT 1,000×

Next-generation information technologies will process unprecedented amounts of locally structured data that overwhelm existing computing systems. N3XT improves the energy efficiency of abundant-data applications 1,000× by using new logic and memory technologies, 3D integration with fine-grained connections, and new architectures for computation immersed in memory.
N3XT NanoSystems

Computation immersed in memory
N3XT NanoSystems

Computation immersed in memory

Increased functionality

Memory

Computing logic

Fine-grained, ultra-dense 3D

Impossible with today’s technologies
N3XT Computation Immersed in Memory

3D Resistive RAM
Massive storage

1D CNFET, 2D FET
Compute, RAM access

MRAM
Quick access

1D CNFET, 2D FET
Compute, RAM access

1D CNFET, 2D FET
Compute, Power, Clock

thermal

No TSV
Ultra-dense, fine-grained vias
Silicon compatible
Carbon Nanotube FET (CNFET)

CNT: \( d = 1.2\text{nm} \)

CNFET

**Energy Delay Product**

- \( \sim 10\times \) benefit

Full-chip case studies

[IBM, IMEC, Stanford, other commercial]
CNFET Inverter

P+ Doped

N+ Doped

VDD

OUT

GND

INPUT
Big Promise, Major Obstacles

- Process advances alone inadequate

Mis-positioned CNTs

Metallic CNTs

Imperfection-immune paradigm

[Zhang IEEE TCAD 12]
CNT Growth *circa* 2005

- Highly mis-positioned
First Wafer-Scale Aligned CNT Growth

Quartz wafer with catalyst → Aligned CNT growth

Quartz wafer with CNTs

99.5% aligned CNTs

Stanford Nanofabrication Facility

[Patil VLSI Tech. 08, IEEE TNANO 09]
Wafer-Scale CNT Transfer

High-temperature CNT growth

900 °C

Before transfer

Quartz

2 µm

CNTs

Low-temperature circuit fabrication

120 °C

After transfer

SiO₂/Si

2 µm

[Patil VLSI Tech. 08, IEEE TNANO 09]
1. Grow CNTs
Mis-Positioned CNT-Immune NAND

1. Grow CNTs

2. Extended gate, contacts

CRUCIAL

[Patil IEEE TCAD 09]
Mis-Positioned CNT-Immune NAND

1. Grow CNTs
2. Extended gate, contacts
3. Etch gate & CNTs
4. Dope P & N regions

- Arbitrary logic functions
  - Graph algorithms

Etched region essential

[Patil IEEE TCAD 09]
Imperfection-Immune VLSI

Mis-positioned CNTs

- Arbitrary logic functions

Metallic CNTs

- Scalable m-CNT Removal

- Erased (relaxed node)
- Scaled circuits

[Patil Symp. VLSI Tech. 08, TCAD 09, Shulaker IEDM 15]
Most Importantly

- VLSI processing
  - No per-unit customization

- VLSI design
  - Immune CNT library
CNT Computer

[Shulaker Nature 13]
CNT Computer

- Turing-complete processor: entirely CNFETs
10× EDP, BUT…

How can we do better?
N3XT Computation Immersed in Memory

3D Resistive RAM
Massive storage

1D CNFET, 2D FET
Compute, RAM access

MRAM
Quick access

1D CNFET, 2D FET
Compute, RAM access

1D CNFET, 2D FET
Compute, Power, Clock

thermal

No TSV

Ultra-dense, fine-grained vias

Silicon compatible
Many Nano-scale Innovations

Memory & logic devices

- 3D Resistive RAM (RRAM)
- MoS$_2$
- Vertical MOSFETs
- <1 nm
- 2D FETs: large-area monolayer MoS$_2$

Embedded cooling

- Phase change: hotspots suppressed
- Evaporative Wicking
- 30 µm thick
- Vertical metal nanowire arrays
3D Integration

- Massive ILV density $\gg$ TSV density

TSV (chip stacking) - Through silicon via (TSV)

Dense, e.g., monolithic - Nano-scale inter-layer vias (ILVs)
Realizing Monolithic 3D

- Low-temperature fabrication: $< 400 \, ^\circ\! C$
Device + Architecture Benefits

Naturally enabled

Emerging logic + Emerging memory + Monolithic 3D integration

[Wei IEDM 09, 13, Shulaker VLSI Tech 14]
3D NanoSystem

Wafer-scale design + fabrication

[Shulaker Nature 17]
3D NanoSystem

>2 Million CNFETs, 1 Mbit Resistive RAM

[Shulaker Nature 17]
3D NanoSystem

- Interwoven compute + memory + sensing

[Shulaker Nature 17]
3D NanoSystem

Abundant data: Terabytes / second

- Millions of sensors
- Ultra-dense vertical connections
- CNTs
- 1 Megabit RRAM
- CNT computing logic
- In-situ classification accelerator
- Extensive, accurate

[Shulaker Nature 17]
3D NanoSystem Results

Principle Component Analysis (PCA)

[Shulaker Nature 17]
N3XT Simulation Framework

Joint technology, design & app. exploration

Heterogeneous technologies

System-level analysis

Architecture exploration

Energy, exec. time

Thermal

Physical design, yield, reliability

Abundant-data apps

[Aly IEEE Computer 15, Hwang CODES/ISSS 17]
Massive Benefits:
Deep Learning, Graph Analytics, ...

IBM graph analytics
DeepDive
TensorFlow

2D
- 64 GB off-chip DRAM
- DDR3 interface
- 64 processor cores
- SRAM cache

Single-chip N3XT
- 64 GB on-chip 3D RRAM
- "Simple" interface
- STTRAM cache
- 64 processor cores
Massive Benefits: Deep Learning, Graph Analytics, ...

~1,000× benefits, existing software

![Bar chart showing benefits in energy and execution time for various algorithms: PageRank, SSSP, Connected Components, BFS, Logistic Regression, Linear Regression. The benefits are listed as follows: 851x, 656x, 400x, 510x, 700x, 970x.]
Massive Benefits: Deep Learning, Graph Analytics, ...

851× benefits

- Energy: 37×
- Exec. Time: 23×
Massive Benefits: Deep Learning, Graph Analytics, ...

- 100× – 1,000× benefits (energy × execution time)

- Deep learning accelerators

System-Level Benefits

Chip stacking: 2 - 4× benefits

[Hwang CODES/ISSS 17] LSTM: Long Short Term Memory, CNN: Convolutional Neural Network
Complement with Software Solutions

Co-optimized s/w + h/w

DSL compiler

Learning: key architectural concept

Runtime optimization

Yield, reliability

Cross-Layer Resilience

DSL = Domain-Specific Language
More Opportunities

Co-optimized hardware + software

Brain-inspired

Technology innovations

Outline

- Robust operation: silicon CMOS reliability
- Beyond silicon
- Conclusion
Thanks to Research Group
Thanks to Sponsors & Collaborators
## Conclusion

- Robust systems
  - New solutions: elegantly simple, effective

### Silicon CMOS Reliability

- BISER, LEAP
- Failure prediction
- CLEAR cross-layer

### Beyond silicon

- CNFET nanosystems
- *N3XT* monolithic 3D
- 1,000X opportunity