Post-Silicon Validation of Robust Systems

Subhasish Mitra

Robust Systems Group

Dept. of EE & Dept. of CS

Stanford University

Acknowledgment: Students & collaborators
Robust System Challenges

Technology challenges

- Radiation, Erratic bits
- Aging, Early-life failures

Acceptable results Constraints:
- Power, performance

Inputs

Robust System

Post-Silicon bugs

System complexity
Robust System Design

- Perform correctly
  - Despite complexity & disturbances

- Thorough validation & test ⇐ Today’s focus
- Tolerate imperfect hardware
- Beyond silicon-CMOS: imperfection-immune logic
A Brief History of Bugs

- 1591 – Shakespeare
  
  Henry VI, part III - Act V, Scene II
  
  “… For Warwick was a bug that fear'd us all ….“

- 1878 – Edison
  
  Letter to T. Puskas
  
  “The first step is an intuition – … and then that – ‘Bugs’ as such little faults and difficulties are called show themselves ...”
Post-Silicon Validation Essential

Design → Pre-silicon verification → Fab → Post-silicon validation → High volume

Barcelona TLB bug: 6 months shipment delay
BIOS fix: 10% -20% performance penalty

35% development time 25% design resources

“Post-silicon cost & complexity rising faster than design cost” – S. Yerramilli, V.P., Intel
Electrical Bugs Difficult

- e.g., speed-path, hold time, noise
  - Some voltage, temp., frequency corners
  - Heisenbug characteristics
- LONG debug time – days to weeks
Post-Silicon Bug Localization Challenge

- Pinpoint from system failure (e.g., crash)
  - Bug location, exposing stimulus

Run apps. (OS, games)  
Detect bugs  
Localization dominates cost  
Root-cause & fix  
Localize bugs
Internal Node Access

13K relays (Harvard Mark II)

731M transistors
1,366 pins

thocp.net

history.navy.mil
Failure Reproduction, System Simulation

- System failure reproduction difficult
  - Asynchronous I/Os, clock domains
- Full system simulation for golden response
  - $10^6 - 10^9$ slower than silicon
Key Message: IFRA + QED

- Overcome major barriers
  - $10^6 \times$ improved error detection latency
  - 4X improved error detection
  - 90 – 96% localization accuracy

- Practical
  - Intel® Nehalem + Core™ i7 results
    - Latest high-end Intel architecture

[Park DAC 08, TCAD 09, Hong ITC 10]
Instruction Footprint Recording and Analysis

Design Phase

Special recorders

Record special info.

No

Failure?

Yes

Scan out recorder contents

Post-analyze offline

Localized Bug: (location, stimulus)

Post-Silicon Validation

No failure reproduction
Single failure sighting

No system simulation
Self-consistency
IFRA Hardware for Superscalar Processor

FETCH
- Branch Predictor
- Fetch Queue
- Special IDs
- I-TLB
- I-Cache

DECODE
- Decoders

DISPATCH
- Reg Map
- Reg Free
- Reg Rename

ISSUE
- Instruction Window
- Phys Regfile

EXECUTE
- MUL
- 2xALU
- D-Cache
- 2xBr
- FPU
- 2xLSU
- D-TLB

COMMIT
- Reorder Buffer
- Reg Map

1% area cost
60KB for Alpha 21264

Post-Trigger Generator

Slow wire (scan reuse)

Scan chain
Recording Operation Example

Fetching Operation Example

Special ID Assignment Rule

Instruction Footprints

Decoding Operation Example

Recorder 1

Recorder 2
Instruction Footprint Recorder Design

- Memory dominated
  - Circular buffer
- Simple control logic
  - Compact idle cycles
  - Manage circular buffer
  - Pause recording
## What to Record?

<table>
<thead>
<tr>
<th>Pipeline stage</th>
<th>Auxiliary information</th>
<th>Bits per recorder</th>
<th>Number of recorders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fetch</td>
<td>Program Counter</td>
<td>32</td>
<td>4</td>
</tr>
<tr>
<td>Decode</td>
<td>Decoding results</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Dispatch</td>
<td>Register names residue</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Issue</td>
<td>Operands residue</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>ALU, MUL</td>
<td>Result residue</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>Branch</td>
<td>None</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>Load/Store</td>
<td>Result residue, Memory address</td>
<td>35</td>
<td>2</td>
</tr>
</tbody>
</table>

Total storage for all recorders: 60 Kbytes

8-bits ID, 1K entries per recorder
Special Rule for Instruction ID Assignment

- Simplistic schemes inadequate
  - Speculation + flushes, out-of-order, loops
  - Multiple clock domains
- Special rule [Park TCAD 09]
  - ID width: $\log_2 4n$ bits
    - $n = \text{max. instructions in flight}$
    - 8 bits for Alpha ($n = 64$)
- No timestamp or global synchronization
Early Warnings for Post-Triggers MUST

Error after 5 billion cycles (e.g., speedpath)

Failure after 6 billion cycles (e.g., crash)

Too much storage overhead to store 1 billion cycles
Early Warnings for Post-Triggers

- Error after 5 billion cycles (e.g., speedpath)
- Failure after 6 billion cycles (e.g., crash)

- Arithmetic residue, array parity
- Deadlock & segfault
  - Special early warnings pause recording
  - QED: Quick Error Detection

Test Program Execution

Need to capture in recorder storage

Early failure suspect detection

t=0
Post-Analysis Overview

Test program binary

Footprints from recorders

Link footprints

Self-consistency analysis

List of bug location-stimulus pairs
### Footprint Linking

#### Test program

<table>
<thead>
<tr>
<th>PC0</th>
<th>INST0</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC1</td>
<td>INST1</td>
</tr>
<tr>
<td>PC2</td>
<td>INST2</td>
</tr>
<tr>
<td>PC3</td>
<td>INST3</td>
</tr>
<tr>
<td>PC4</td>
<td>INST4</td>
</tr>
<tr>
<td>PC5</td>
<td>INST5</td>
</tr>
<tr>
<td>PC6</td>
<td>INST6</td>
</tr>
</tbody>
</table>

#### Fetch recorder

<table>
<thead>
<tr>
<th>ID: 7</th>
<th>PC5</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: 0</td>
<td>PC0</td>
</tr>
<tr>
<td>ID: 5</td>
<td>PC3</td>
</tr>
<tr>
<td>ID: 6</td>
<td>PC4</td>
</tr>
<tr>
<td>ID: 7</td>
<td>PC5</td>
</tr>
<tr>
<td>ID: 0</td>
<td>PC6</td>
</tr>
<tr>
<td>ID: 4</td>
<td>PC0</td>
</tr>
<tr>
<td>ID: 5</td>
<td>PC1</td>
</tr>
<tr>
<td>ID: 6</td>
<td>PC2</td>
</tr>
<tr>
<td>ID: 7</td>
<td>PC3</td>
</tr>
<tr>
<td>ID: 0</td>
<td>PC4</td>
</tr>
</tbody>
</table>

#### Issue recorder

<table>
<thead>
<tr>
<th>ID: 0</th>
<th>AUX0</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: 7</td>
<td>AUX1</td>
</tr>
<tr>
<td>ID: 6</td>
<td>AUX2</td>
</tr>
<tr>
<td>ID: 5</td>
<td>AUX3</td>
</tr>
<tr>
<td>ID: 0</td>
<td>AUX3</td>
</tr>
<tr>
<td>ID: 7</td>
<td>AUX5</td>
</tr>
<tr>
<td>ID: 7</td>
<td>AUX6</td>
</tr>
<tr>
<td>ID: 4</td>
<td>AUX7</td>
</tr>
<tr>
<td>ID: 6</td>
<td>AUX8</td>
</tr>
<tr>
<td>ID: 0</td>
<td>AUX9</td>
</tr>
<tr>
<td>ID: 7</td>
<td>AUX10</td>
</tr>
</tbody>
</table>

#### ALU recorder

<table>
<thead>
<tr>
<th>ID: 0</th>
<th>AUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID: 7</td>
<td>AUX</td>
</tr>
<tr>
<td>ID: 5</td>
<td>AUX</td>
</tr>
<tr>
<td>ID: 7</td>
<td>AUX</td>
</tr>
<tr>
<td>ID: 6</td>
<td>AUX</td>
</tr>
<tr>
<td>ID: 5</td>
<td>AUX</td>
</tr>
<tr>
<td>ID: 5</td>
<td>AUX</td>
</tr>
<tr>
<td>ID: 6</td>
<td>AUX</td>
</tr>
<tr>
<td>ID: 7</td>
<td>AUX</td>
</tr>
<tr>
<td>ID: 0</td>
<td>AUX</td>
</tr>
</tbody>
</table>

#### Time

- Younger
- Older
Special ID assignment rule ensures:

- Uncommitted instructions uniquely identified
- Relative orders between identical IDs maintained
High-Level Analysis

Link footprints

<Linked footprints, Test program binary>

Control-flow

Data-dependency

Decoding

Load/Store

<Initial location, Initial footprint>

Low-level analysis

<Bug location, Bug stimulus>
Low-Level Analysis

Link footprints

High-level analysis

Microarchitecture independent

Microarchitecture dependent (manually generated)

<Bug location, Bug stimulus>
Low-Level Analysis: Decision 1

Test Program Binary

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>R0 ← R1 + R2</td>
</tr>
<tr>
<td>14</td>
<td>Jump to 20</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>R0 ← R3 + R6</td>
</tr>
<tr>
<td></td>
<td>Read-After-Write</td>
</tr>
<tr>
<td></td>
<td>Hazard</td>
</tr>
<tr>
<td>64</td>
<td>Jump to R0</td>
</tr>
</tbody>
</table>

Fetch recorder

<table>
<thead>
<tr>
<th>ID</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
</tr>
</tbody>
</table>

ALU recorder

<table>
<thead>
<tr>
<th>ID</th>
<th>Reg. value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
</tr>
</tbody>
</table>

Issue recorder

<table>
<thead>
<tr>
<th>ID</th>
<th>Reg. values</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>0</td>
</tr>
</tbody>
</table>

Residues of values match? No simulation required

No
Low-Level Analysis: Decision 2

Test Program Binary

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>R0 ← R1 + R2</td>
</tr>
<tr>
<td>14</td>
<td>Jump to 20</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>20</td>
<td>R0 ← R3 + R6</td>
</tr>
<tr>
<td></td>
<td><strong>Read-After-Write</strong></td>
</tr>
<tr>
<td></td>
<td><strong>Hazard</strong></td>
</tr>
<tr>
<td>64</td>
<td>Jump to R0</td>
</tr>
</tbody>
</table>

Fetch recorder

<table>
<thead>
<tr>
<th>ID</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
</tr>
</tbody>
</table>

Dispatch recorder

<table>
<thead>
<tr>
<th>ID</th>
<th>Reg. names</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1  2  0</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>-  0  -</td>
</tr>
</tbody>
</table>

Residues of physical register names match? No simulation required

No
Low-Level Analysis: Decision 3

Test Program Binary

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>R0 ← R1 + R2</td>
</tr>
<tr>
<td>20</td>
<td>R0 ← R3 + R6</td>
</tr>
<tr>
<td>24</td>
<td>Jump to 64 if R3=0</td>
</tr>
<tr>
<td>64</td>
<td>Jump to R0</td>
</tr>
</tbody>
</table>

Fetch recorder

<table>
<thead>
<tr>
<th>ID</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
</tr>
</tbody>
</table>

Dispatch recorder

<table>
<thead>
<tr>
<th>ID</th>
<th>Reg. names</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>

Physical register names match with previous producer? Yes

No simulation required
Low-Level Analysis: Remaining Decisions

Link footprints

High-level analysis

<Bug location, Bug stimulus>
Example Localized Bug – Location

Pipeline Register

Decoder

Arch. Dest. Reg | Rest of pipeline reg.

Read Circuit | Write Circuit

Reg. Mapping

Rest of modules in dispatch stage

Bug Location
### Example Localized Bug – Stimulus

#### Test Program Binary

<table>
<thead>
<tr>
<th>Address</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>R0 ← R1 + R2</td>
</tr>
<tr>
<td>14</td>
<td>Jump to 20</td>
</tr>
<tr>
<td>20</td>
<td>R0 ← R3 + R6</td>
</tr>
<tr>
<td>24</td>
<td>Jump to 64 if R3=0</td>
</tr>
<tr>
<td>64</td>
<td>Jump to R0</td>
</tr>
</tbody>
</table>

#### Fetch recorder

<table>
<thead>
<tr>
<th>ID</th>
<th>PC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>14</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
</tr>
<tr>
<td>3</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>64</td>
</tr>
<tr>
<td>5</td>
<td>00</td>
</tr>
</tbody>
</table>

The diagram illustrates the flow of execution, starting at address 10 and progressing through the instructions, with arrows indicating the Stimulates Bug sequence.
After Localization ? – Circuit Debug

- Target
  - Pin-pointed design blocks & vicinity
    - Enhanced controllability & observability

- Test patterns
  - From bug-exposing stimulus
  - Legacy patterns

- Circuit debug support – effective techniques exist
  - Clock shrink, clock stretch, PICA, LADA
Simulation Flow

- Masked/silent error
  - No
    - Any failure detected?
      - No
        - Warm up for a million cycles
      - Yes
        - Inject error
          - Short error latency?
            - No
              - Complete miss
            - Yes
              - Post-analyze
                - Exact localization
                - Localization with candidates
Bug-Localization Results: Alpha 21264

Total candidates: 200,000+
(200 design blocks) x (1,000+ error appearance cycles)

Correct localization (96%)

Complete miss (4%)

Exact localization (78%)

Avg. 6 candidates (22%)
Post-Analysis Decision Diagram Generation

Link footprints

HLA1  HLA2  HLA3  HLA4

Microarchitecture independent

Microarchitecture dependent (manually generated)

Bug locations + exposing stimulus

手动生成昂贵且容易出错

Towards automation → BLoG
BLoG – Bug Localization Graph

Design
- Insert recorders

Bug Detection
- Record footprints until failure
- Footprint linking
- μarch-independent analysis

Offline Analysis
- Traverse BLoG

Localized Bug

Microarch. description

BLoG
BLoG Nodes & Edges

4 Cores

Register value
Last PC
MUX
16
μop
Predicted PC
Current PC
=M
Misprediction?
REG
REG

Register value
Last PC
P
16
μop
Predicted PC
Current PC
M
Predicted PC
C
Misprediction?
BLoG Node Types – Storage Structures

Random-access

- **R**
  - Data
  - Control
  - Address
  - e.g., register file

Associative

- **A**
  - Data
  - Control
  - Tag
  - e.g., TLB

Queue

- **Q**
  - Data
  - Control
  - e.g., reorder buffer
BLoG Node Types – Non-storage Structures

Modifying

M
Data 1
Data 2
? e.g., decoder

Connection

C
Data 1
Data 1 e.g., pipeline registers

Select

S
In 1 In 2
Control e.g., forwarding path
Out
BLoG Node Types – Non-storage Structures

Protected

Data 1

Checker

Data 2

e.g., residue protected ALU

Default

Data in

Control

Data Out

Everything else
BLoG Edge Dependencies

- Last PC
- Current PC
- Predicted PC
- Misprediction?
- Register value
- 16
- \( \mu \text{op} \)
- Schedule Rec
- Operand
- Fetch Rec
- Commit Rec
Example Self-Consistency Check 1

No simulation required

IF (Z ≠ X) AND (Z ≠ Y)
Example Self-Consistency Check 2

ELSE IF \( (C_i = C_j \text{ AND } Z_i = X_i \text{ AND } Z_j = Y_j \text{ for any } i, j) \)

No simulation required
Example Self-Consistency Check 3
BLoG Traversal Example

<table>
<thead>
<tr>
<th>ID</th>
<th>AUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>23</td>
</tr>
<tr>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>AUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>123</td>
</tr>
<tr>
<td>3</td>
<td>234</td>
</tr>
<tr>
<td>4</td>
<td>434</td>
</tr>
<tr>
<td>6</td>
<td>788</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>AUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>AUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
</tr>
<tr>
<td>7</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>AUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>13</td>
</tr>
<tr>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>4</td>
<td>43</td>
</tr>
<tr>
<td>7</td>
<td>78</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>AUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>34</td>
</tr>
<tr>
<td>5</td>
<td>44</td>
</tr>
<tr>
<td>6</td>
<td>88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>AUX</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>128</td>
</tr>
<tr>
<td>5</td>
<td>247</td>
</tr>
<tr>
<td>4</td>
<td>345</td>
</tr>
<tr>
<td>3</td>
<td>783</td>
</tr>
</tbody>
</table>
Simulation Results on Intel Nehalem

Possible candidate space: 269,000

= (269 design blocks) x (1,000 error appearance cycles)

Correct localization (90%)

Complete miss (10%)

Exact localization

Average 6 out of 269,000 possible candidates

(62%)

(38%)
Multi-Core SoCs Challenging

- Example 1: Complex interactions

Core 1

\[
\text{<code>}
A \leftarrow B + 8
\text{<more code>}
\text{Mem}[C] \leftarrow A
\text{<more code>}
\]

Core 2

\[
\text{D} \leftarrow \text{Mem}[C]
\text{<more code>}
E \leftarrow \text{Mem}[D]
\text{SEGFAULT}
\]

very long
Inter-Core Store-to-Load Latencies

TOO LONG for bug-localization

FMM
Splash Benchmark
4-core 4-way OOO CMP

% of total occurrences

100% 80% 60% 40% 20% 0%

Number of cycles

1K 10K 100K 1M >1M
Error Detection Latency: BIG CHALLENGE

Billions of cycles detection latency

Example 2: NIC protocol bug

Monitor packet drop rate

Dropped packet rate (sec⁻¹)

Time (10⁹ cycles)
Idea: Quick Error Detection (QED)

- Guarantee short error detection latency
- Automatically transform existing validation tests
  - Wide variety
    - Software-only, hardware-assisted
- Compatible with
  - IFRA
  - Trace buffers

[Hong ITC 10]
QED Validation Test Principle

Original

Snippet 0
Snippet 1
Snippet 2
... (snippets n)
Observed Failure

Error produced

😊 Low latency
😊 Easier localization
😊 Better coverage possible

Long error detection latency

QED

Snippet 0
Snippet 0'
Check
Snippet 1
Snippet 2
Snippet 1'
Snippet 2'
Check

Short error detection latency
QED Validation Test Principle

Original

Snippet 0
Snippet 1
Snippet 2

Snippet n
Observed Failure

QED

Snippet 0
Snippet 0'
Check

Snippet 1
Snippet 2
Snippet 1'
Snippet 2'
Check

error produced

😊 User-configurable check granularity

Short error detection latency

Long error detection latency
# QED Implementation: Wide Variety

<table>
<thead>
<tr>
<th></th>
<th>Code change</th>
<th>Hardware change</th>
<th>Check granularity</th>
<th>Electrical bugs (E)</th>
<th>Logic bugs (L)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EDDI</strong></td>
<td>😞 Some</td>
<td>😊 None</td>
<td>😞 Limited</td>
<td>😞 E</td>
<td>😞 L (some)</td>
</tr>
<tr>
<td><strong>EDDI-M</strong></td>
<td>😞 Some</td>
<td>😊 None</td>
<td>😊 Flexible</td>
<td>😞 E</td>
<td>😞 L (some)</td>
</tr>
<tr>
<td><strong>SW RMT</strong></td>
<td>😞 None</td>
<td>😞 Some</td>
<td>😞 Fine</td>
<td>😞 E</td>
<td></td>
</tr>
<tr>
<td><strong>HW RMT</strong></td>
<td>😞 None</td>
<td>😞 Some</td>
<td>😞 Fine</td>
<td>😞 E</td>
<td></td>
</tr>
<tr>
<td><strong>BLADE</strong></td>
<td>😞 Some</td>
<td>😊 None / Some</td>
<td>😊 Flexible</td>
<td>😞 Both L &amp; E</td>
<td></td>
</tr>
</tbody>
</table>
Intel® Core™ i7 Hardware Results

- **Error detection latency (clock cycles)**
  - 0 - 10K: QED
  - 1-10 Billion: No QED

- **Detected error count** (normalized)
  - Improved error detection: 4X
  - Improved error latency: $10^6$ X

- Temperature controller
- In-Target probe (ITP)
### QED Shmoo Results for Intel® Core™ i7

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>0.9</td>
<td>1.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.4</td>
</tr>
<tr>
<td>1.0</td>
<td>0.1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.3</td>
</tr>
</tbody>
</table>

- **No boot**
- **Detected by QED / crash**
- **Pass**
- **Coverage improved by QED**

#### Original test
- Pass
- Error detection
- Crash

#### QED test
- Pass / Quick detection
- Quick detection
- Crash / Quick detection
Robust System Design

- Perform correctly
  - Despite complexity & disturbances

- Thorough validation & test ⇐ Today’s focus
- Tolerate imperfect hardware
- Beyond silicon-CMOS: imperfection-immune logic
Tolerate Imperfect Hardware: Ultra Low Cost

Circuit Failure Prediction
New failure signatures discovered
1% area, 1% power, 3% system performance

BISER & LEAP flip-flops
Self-correcting
Errors reduced: 2,000X

Burn-in difficult
\( I_{ddq} \) ineffective

Early-life failures (ELF)

Radiation-induced errors

Transistor aging
Guardbands expensive

Failure rate

Time

Aging
Carbon Nanotube Logic: Big Promise, BUT

- Major barriers: inherent imperfections at nano-scale
  - Imperfection-immune logic overcomes barriers
    - Practical, elegantly simple
- First experimental demonstration
  - VLSI computation & storage circuits

Collaborator: Prof. Philip Wong, EE
Conclusion

- Robust system design: efficient techniques practical
- IFRA + QED
  - Overcome major barriers
    - $10^6 X$ improved error detection latency
    - 4X improved error detection
    - 90 – 96% localization accuracy
  - Practical
    - Intel® Nehalem + Core™ i7 results