Designing Tomorrow’s Microprocessors
Qiong Cai, Josep M Codina, Antonio González, Pedro Marcuello,
Raúl Martínez, Kyriakos Stavrou, Xavier Vera, Polychronis Xekalakis

Intel Barcelona Research Center
Aula Empresa, Facultat d’Informàtica de Barcelona, February 2010
© Intel Corporation, 2010

Course Agenda
- Overview of Today’s Microprocessors and Future Trends
- Design Cycle for Microprocessors
- Methodology for Research in Microprocessors
- Multi-core Processor Architectures
- Parallel Programming
- Systems-on-Chip
- Techniques for Power Reduction
- Reliability
- Hardware/Software Co-designed Microprocessors

Acknowledgment
• Phoenix Project
  – Osman Unsal
  – Oguz Ergin
  – Jaume Abella
  – Pedro Chaparro
  – Javier Carretero
• Raimat Project
  – Tana Ramirez
• Intel
  – J. Tschanz, S. Mitra, S. Iacobovicj, K. Bowman, C. Wilkerson, and many others!

Introduction to new reliability challenges
Robust design and testing challenges
Some solutions

Robust Computing System
Defects, Process variation, Degraded transistors
Radiation, Noise

Inputs
Robust MultiCore System
“Acceptable” Outputs
Power Performance

Design errors, Software failures
Malicious attacks, Human errors
High Reliability Systems

- Aircraft control, fly-by-wire
- Safest
- Automobile
- Simpler is better?
- Nuclear reactors
- Reliable systems
- Military
- Redundant systems
- Space
- Short mission times

What Has Changed in the Last 20 years?

- 1971 - i4004
  - 108 KHz
  - 2300 transistors
- 1989 - i486
  - 33 MHz
  - ~1M transistors
- 2009 – Quadcore i7
  - 2.93 GHz
  - 738M transistors

Moore’s Law

- Transistors are smaller, and weaker
- It’s more complicated to build them
- The more things we have...
- Higher currents
- Higher temperatures
- Easier to have some bugs

New Landscape?

- Reliability was for mission critical

Hardware Failure Taxonomy

- Permanent
  (Defects, out of range parameters, wearout, etc.)
- Temporary
  - Transient
  - Intermittent
    (operation margins, infant mortality, weak parts, process variation, random dopant fluctuation, etc.)
- Radiation
  (Soft errors)
- Non-radiation
  (Power-supply, coupling, EMI, etc.)

Introduction to new reliability challenges

Robust design and testing challenges
Some solutions
**Process Variations**

- **Normalized Leakage (Isb)**
  - 0.18 micron ~1000 samples
  - 30% 20X

**Degradation**

<table>
<thead>
<tr>
<th>Affects</th>
<th>Weakest</th>
<th>Worst Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Electromigration</td>
<td>Connections</td>
<td>Longest “1”</td>
</tr>
<tr>
<td>Stress migration</td>
<td>Connections</td>
<td>Longest Any</td>
</tr>
<tr>
<td>Time-dependent dielectric breakdown (TDDB)</td>
<td>Gate oxide</td>
<td>Widest “0” PMOS “1” NMOS</td>
</tr>
<tr>
<td>Negative Bias Temperature Instability (NBTI)</td>
<td>Gate oxide</td>
<td>Widest “0”</td>
</tr>
<tr>
<td>Large thermal cycling</td>
<td>Package</td>
<td>***</td>
</tr>
<tr>
<td>Short thermal cycling</td>
<td>Unknown</td>
<td>***</td>
</tr>
</tbody>
</table>

**Degradation: Electromigration**

- Wire is made up of metal “grains”

**Defects**

- Void under anchor
- Silicon damage
- Metal2 extrusion/ILD2 crack
- Metal 1 Shunting

**Transistor degradation**

- Traditional:
  - Oxide breakdown
  - Hot electron, etc.
  - Electromigration
- New:
  - Negative Bias Temperature Instability (NBTI)

**Current practice**

- Speed guardbands
- Very expensive

**Process Variations (i2)**

- Worst-case design
- Statistical design consequences
- Very expensive or impractical
- VERY thorough delay test required in predictability of speed limiting paths?

**Defects**

- M4-M4 Short
- Poly stringer
- M4 Void Formations

Source: [Spinellis ETW 2002]
Bathtub Curve

Infant Mortality

- Burn-in becomes less effective
  - Latent defects remain dormant
  - Higher level of variations aggravates the problem
  - Power limits maximum temperature, so many defects remain dormant

Next Generation Burn-in & Test System for Athlon Microprocessors: Hybrid Burn-in, Mark Hillier, Burn-in & Test Socket Workshop, 2001

Source of Noise

- Crosstalk
- Power supply noise
- Substrate noise
- Soft errors
- Electromagnetic interference
- Thermal noise

Transistor Device

Latching Window

Soft Errors

- Neutrons strike on Si device
- Alpha particles from packaging
- They impact latches
- They impact combinational logic

Soft Errors (2)

- Soft Errors can cause problems in different ways
  - Change the data value in the Caches and Memory
  - Corrupt the execution of instruction due the flip of data in the pipeline registers.
  - Change the character of a SRAM-Based FPGA circuit (Firm Error)
  - Datapath logic SET (Single Event Transient) caught by registers/memory

Soft Errors (3): Evidence

- Error logs of large servers
- Sun Microsystems, 2000 (from Baumann, IRPS 2002)
  - Cosmic ray strikes on L2 cache
    - Mysterious crashes of Sun flagship servers
    - Companies affected
      - Baby Bell (Atlanta), America Online, Ebay, & dozens others
      - Verisign moved to IBM Unix servers (for the most part)
Soft Errors (i4): Doomsday

Altitude of 30,000 feet on a route crossing the north pole both cause increase in neutron flux.

Four 1M 130nm SRAM-based FPGAs, it would be subject to 0.074 upsets per day = 324 hours between upsets.

Assume one such system on-board each commercial aircraft, 4,000 civilian flights per day, 3 hours average flight time.

Nearly 37 aircraft will experience a neutron-induced SRAM-based FPGA configuration failure during the duration of their flight.

Summary: VLSI Trends & Reliability

<table>
<thead>
<tr>
<th>Reliability Problems</th>
<th>VLSI Trends</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power supply, Signal integrity problems</td>
<td>High speed, low voltage, large current</td>
</tr>
<tr>
<td>Process variation (die-to-die, intra-die)</td>
<td>Many transistors, nano-fabrication, high speed</td>
</tr>
<tr>
<td>Manufacturing defects</td>
<td>Many transistors, new material, burn-in issues</td>
</tr>
<tr>
<td>Degradation over time</td>
<td>Large current, increased electric field, new material, thin oxide, stress void, High-K</td>
</tr>
<tr>
<td>SEUs due to radiation</td>
<td>Many transistors, low-voltage, high speed</td>
</tr>
</tbody>
</table>

Testing

Validation
- Find out if something goes well

Testing
- One mechanism to validate

Coverage
- How many errors we catch...
- % of the area covered
- % of the transistors covered
- % of inputs covered

Testing (2)

Application level:
- Programs are run and their outputs verified. Easy to do, but the coverage is very low

Functional level:
- Each individual block is tested with different inputs. Coverage increases, but many timing/state related faults are missed

Structural level:
- Inputs are injected to force failures to show up in the different nodes of the block. The coverage is much higher, but it is complex producing proper inputs for high coverage

Testing (3)

- Observability
  - How easy is to propagate a failure in a node to the outputs
  - Internal nodes can be latched to increase observability
- Controllability
  - How easy is to set a node to a given value
**Testing Challenges**

- Technology issues/integration consequences
  - Testing time
  - Yield vs Debug
  - Low Vcc: decrease margins, more noise
- "New" on-die complex structures
  - For instance, interconnection network, validate the coherence protocol
  - DVS/DVFS

**Design For Testing (DFT)**

- DFT consists in
  - Increasing the observability and controllability of circuits
  - Putting hardware in place to allow testing (e.g. scan pins, BIST, etc.)
  - Space redundancy
    - Self-checking blocks: some properties of the outputs are checked for correctness (e.g. residue, parity, ECC)
    - Full hardware replication: operations are repeated in replicated hardware (e.g. triple modular replication, DIVA, etc.)
  - Time redundancy
    - Outputs are latched twice at different times to detect timing errors

---

**High Availability Building Blocks**

- **Fault Tolerance**
  - Concurrent Repair
  - Design Verification
  - System Integration
- **Fault Avoidance**
  - Failure Masking
  - Reliability
  - Integration

**System Design Technology**

**Data Integrity**

- Failure Masking
- System Design Technology

**Detect & Isolate**

- Recover
- Detect & Isolate

**Error Masking**

- No error on outputs
  - Triple Modular Redundancy (TMR)
  - N-Modular Redundancy (NMR)

**Error Detection and Fault Isolation**

- Fault avoidance
  - Rigorous design and verification
  - Comprehensive testing
- Instantaneous error detection
  - Arrays, controls, latches, dataflow, interface
  - Redundancy
  - Inline checking
  - ECC for memory and caches
  - CRC for I/O links and disks

---

**Some solutions**

**Introduction to new reliability challenges**

**Robust design and testing challenges**

**Module 1**

**Module 2**

**Module 3**

**Voter**

**Voted Outputs**
### CED Structure

**Input** → **Function** → **Output Characteristic Predictor** → **Output** → **Checker** → **Error**

- **Parity, residue coding**

### Spatial Replication: LockStep Like

- Duplicate instances of the pipeline; mirroring complete processors
- Compare states:
  - Chip-External Detection: monitor and compare the two processors' external behavior at the chip pins (i.e., all address and data traffic exiting the lowest level of on-chip cache)
  - Cache Interface Comparison: monitor and compare the address and data traffic entering the interface of the cache
  - Full-State Comparison at Checkpoint Creation: the complete set of changes to architectural state are compared

### Spatial Replication: LockStep Like

- Duplicate instances of the pipeline; mirroring complete processors
- Compare states:
  - Chip-External Detection: monitor and compare the two processors' external behavior at the chip pins (i.e., all address and data traffic exiting the lowest level of on-chip cache)
  - Cache Interface Comparison: monitor and compare the address and data traffic entering the interface of the cache
  - Full-State Comparison at Checkpoint Creation: the complete set of changes to architectural state are compared

### Temporal Replication

**RMT: Redundant threads execute at the same time to check for errors**

- Hardware implementation on SMT
- Multi-core based RMT using fingerprint

- Compare signatures of instruction sequences only
- Fewer compares are needed -> reduce thread communication
- Optimized versions: selective/partial replication

### Temporal Replication: Software Assisted

- SIHT (software-implemented hardware fault-tolerance)
  - It introduces the redundant operations and checks/assertions
  - Hardened values and subroutines
- SWIFT approach
  - VLIW
  - Reclaims unused instruction-level resources present during the execution of most programs

### Coding Like

- **Parity**
  - Cheap solution
- **ECC**
  - Used in large caches
  - Extended protection against multiple strikes and multiple upsets
  - Interleaving, scrubbing

### Coding Like Logic - ALUs

- Residue codes
  - Uses residues with coverage close to the one achieved by FUs replication at a fraction of the area & power
- Mod D residue of number N: remainder of N divided by D
  - For arithmetic ops: \((X \text{ op } Y) \text{ mod } D = ((X \text{ mod } 3) \text{ op } (Y \text{ mod } 3)) \text{ mod } 3\)
- Example:
  - \(X = 46238; 46238 \text{ mod } 3 = 2\)
  - \(Y = 56788; 56788 \text{ mod } 3 = 1\)
  - \(X+Y = 103206; 103206 \text{ mod } 3 = 0 (2 + 1 \text{ mod } 3 \text{ is } 0)\)
**Faults Mitigation**

- Transient faults
  - Selected hardening of circuit nodes
  - Add extra capacitance to selected feedback nodes
- NBTI
  - Reduce duty cycle in PMOS transistors through content inversion
- Electromigration
  - Dummy activity in idle cycles to balance current in both directions

**Fault Tolerant System**

<table>
<thead>
<tr>
<th>Post-Si Validation</th>
<th>QuadUse Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detects Degradation Errors</td>
<td>Core-Independent</td>
</tr>
<tr>
<td>Defects/Bugs/Hard Faults</td>
<td>Online/Offline Mode</td>
</tr>
<tr>
<td>Soft Errors</td>
<td>Combine with SW/HW approaches</td>
</tr>
<tr>
<td>Pre-Si Validation, Functional Validation</td>
<td></td>
</tr>
</tbody>
</table>

**CONCLUSIONS**

**Multi-layer Approach**

- A lot has been said about solutions that combine circuit, µarch, and SW approaches
  - Few to none seen
  - Each layer tries to do its best... probably paying a high price
- We need to design solutions bottom-up considering all different layers
  - Clearly identify error detection and recovery requirements for each level
  - Each layer contributes with its own detection and recovery capabilities
  - Co-design CKT/HW/SW solutions!

**Error Detection**

- Software level
  - Check assertions and tolerate some errors
- OS level
  - Task allocation (verified/not verified, low/high activity...)
- Microarchitecture level
  - Codes for data and self-testable blocks
- Circuit
  - Delay detectors
- Technology level:
  - Reduce variations and susceptibility to transient faults, wearoutability

**Adapt the System**

- Reconfiguration
  - Transistor level (manufacturing time)
    - Spare transistors
  - Circuit level
    - Spare cache lines
  - Microarchitecture
    - Memory, processor, routers, networks
  - Software
    - Components that can be virtualized (e.g., bypass levels in inorder cores)
  - OS
    - Task allocation (heterogeneity)