Software-Based Online Detection of Hardware Defects: Mechanisms, Architectural Support, and Evaluation

Kypros Constantinides  
University of Michigan

Onur Mutlu  
Microsoft Research

Todd Austin and Valeria Bertacco  
University of Michigan

Reliability Challenges of Technology Scaling

Suggested Approach

1) Build products out of unreliable components/technologies
2) Provide reliability through very low cost defect-tolerance techniques
In this work we focus on a low-cost technique for detecting and diagnosing hard silicon defects.

Continuous Checking Techniques

- Continuously check for execution errors

- Dual-Modular Redundancy
  - Original Module
  - Copy of the Module
  - Checker

- Processor Checking
  - Main Processor
  - Processor Checker

Shortcomings of continuous checking:

- Redundant computation requires significant extra hardware – high area overhead
- Continuous checking consumes significant energy – pressure on power budget
Periodic Checking Techniques

- Periodically stall the processor and check the hardware
  - If hardware checking succeeds all previous computation is correct
  - Employ checkpointing and roll-back techniques
  - Built-In Self-Test (BIST) techniques to check the hardware

Shortcomings

- Random patterns do not target any specific testing technique (fault model)
- A lot of patterns are needed for good coverage
- Long testing times

Too slow for online testing – High performance overhead

Our Approach – Software-Based Defect Detection

1) Move the hardware checking overhead to software
2) Firmware periodically stalls the processor and perform hardware checking
3) Provide architectural support to the software checking routines

Advantages over hardware-based techniques
- Lower area overhead
- Higher runtime flexibility
  - it can support multiple fault models
  - dynamic tuning of testing process
- Easier to upgrade (software patches)
Access-Control Extensions (ACE) Framework

- Architectural support that enables software access to the processor state (ACE Hardware)
- Special Instructions can access and control any part of the processor state (ACE Instructions)
- Firmware can periodically run directed hardware tests (ACE Firmware)

Accessing The Processor State (ACE Hardware)

- We leverage the existing full hold-scan chain infrastructure
- Full hold-scan chains are employed by most modern processors to improve/automate manufacturing testing
**Accessing The Processor State (ACE Hardware)**

- ACE Instructions can move values from the architectural registers to the scan state and *vice versa*.
- ACE Instructions can swap data between the scan state and the processor state.

**Software-based Testing & Diagnosis (ACE Firmware)**

- **Step 1**: Load test pattern into scan state
- **Step 2**: 3 cycle atomic test operation
  - Cycle 1: Swap scan state with processor state
  - Cycle 2: Test cycle
  - Cycle 3: Swap scan state with processor state
- **Step 3**: Validate test response
**Timeline of Software-Based Testing**

Software-based testing is coupled with a checkpointing and recovery mechanism.

- **Functional software test**
  - Check if the core is capable to run ACE-based testing
  - Limited fault coverage 60-70%
  - Very fast < 1000 instructions

- **Directed ACE-based testing**
  - High-quality testing (ATPG patterns)
  - High fault coverage ~99%
  - Runtime < 1M instructions

**Experimental Methodology**

- OpenSPARC T1 CMP – based on Sun’s Niagara
  - Synopsys Design Compiler to synthesize the OpenSPARC CMP
  - Synopsys TetraMAX ATPG tool for test pattern generation
  - RTL implementation of ACE framework to get area overhead

- Microarchitectural Simulation to get performance overhead
  - SESC cycle-accurate simulator
  - Simulate a SPARC core enhanced with the ACE framework
  - Benchmarks from the SPEC CPU2000 suite
Fault Models used for Test Pattern Generation

- **Stuck-at (0 or 1)**
  - Industry standard fault model for test pattern generation
  - Silicon defects behave as a node stuck at 0 or 1
- **N-Detect**
  - Higher probability to detect real hardware defects
  - Each stuck-at fault is detected by at least $N$ different patterns
- **Path-delay**
  - Test for delay faults that cause timing violations
  - Delay fault can be caused due to:
    - Manufacturing defects
    - Wearout-related defects
    - Process variation

Preliminary Functional Testing

- Fault injection campaign on a gate-level netlist of a SPARC core
- Software functional test – 3 phases (~700 instructions):
  - Control flow check
  - Register access
  - Use all ISA instructions
  - Functional testing coverage is low ~ 62%
  - Undetected faults do not affect the execution of ACE firmware
  - Full coverage provided with further ACE-based testing
**Full-chip Distributed ACE-based Testing**

- Chip testing is distributed to the eight SPARC cores
- Testing for stuck-at and path-delay fault models

Cores [0,1]
Test Instructions: 312K
Coverage: 99.6%

Cores [3,5]
Test Instructions: 405K
Coverage: 98.8%

Cores [2,4]
Test Instructions: 468K
Coverage: 98.7%

Cores [6,7]
Test Instructions: 333K
Coverage: 99.9%

Performance Overhead of ACE-Based Testing

- Performance overhead depends on the fault model used to generate patterns
- ACE framework is flexible to support test patterns from different fault models

**SPEC CPU2000 Average**

![Performance Overhead Chart](chart.png)

Higher quality testing
ACE Framework Area Overhead

- RTL implementation of ACE Framework in Verilog
- Explored several ACE tree configurations
- 8 ACE trees (1 per core) to cover OpenSPARC ~230K ACE accessible bits

Area Overhead:
- 0.7% each tree
- 5.8% for ACE framework

Future Directions – Other Applications

Overhead of ACE framework can be amortized by other applications:
- Manufacturing testing
  - Lower cost of testing equipment
  - Faster testing – testing infrastructure embedded on the chip
- Post-Silicon debugging - direct software access to processor state
Conclusions

- We proposed a novel software-based online defect detection and diagnosis technique
  - Low area overhead: 5.8%
  - High fault coverage: 99%
  - Low performance overhead: 5.5%
- Demonstrated the flexibility of the proposed technique to support:
  - Dynamic trade-off between performance and reliability
  - A number of fault models with varying test quality
- The ACE infrastructure can be a unified framework that provides hardware accessibility and controllability to software

Thank You!

Questions?