Software-Based Self-Testing of Processors

Contents crafted by
Professor Dimitris Gizopoulos
University of Piraeus
Computer Systems Laboratory
Talk Outline

- **Integrated Circuits Testing and Processors Testing**
  - Definitions and Cost considerations
  - External Testing vs. Self-Testing
  - Functional Testing vs. Structural Testing

- **Software-Based Self-Testing (SBST) Paradigm**
  - Basic idea
  - Levels of application
  - Targeting different building blocks

- **SBST state-of-the-art**
  - An SBST research evolution
    - From integer arithmetic units to high-performance microprocessors and multiprocessors
  - Academic research work and groups
  - Industrial applications

- **Future Directions and Conclusions**
Processor design has always been a challenge and a driving force for advances.

Processor design objectives:
- functionality
- performance
- power consumption
- size
- configurability
- testability – dependability

Talk focus: **Software-Based Self-Testing** as a mechanism for improved Testability and Dependability.
IC Testing Purpose and Objectives

- Testing – measuring the quality of a manufactured chip
  - Is the chip manufactured as designed? (structural testing)
  - Does it operate in the field of application as specified? (functional testing)
- Testing has been and will always be a very complex but necessary step
- Reject imperfect ICs before shipping product to customers
  - Eliminate/reduce test escapes for product quality

Testing as a Filtering Process

Is Processor Testing a New Challenge?

- Processor Testing is not a new challenge, not a new adventure
  - It has always been a significant part of the processor development flow
  - But today IC (and processor) testing is getting worse

- According to major system companies:
  - testing can get up to 50-60% of the total manufacturing cost
  - “verification and manufacturing testing is the major capital cost, not the billion dollar silicon fabrication lines”

  - It will cost more to test a transistor than to manufacture it.
  - Manufacturing test cost is one of the most difficult challenges semiconductors industry will have to deal with over the next 5-10 years.
  - Testing cost > 50% of total manufacturing cost.
  - Automatic Test Equipment will cost > $20M each.
Increasing test cost
- Test cost may surpass fabrication cost by 2014 (ITRS Roadmap)

High clock speed
DSM technology
→ At-speed testing

IC/tester performance gap
→ Self-test

Moore’s law in danger?
→ Test problems!

Diverse design styles
→ Adaptability

Increasing design complexity
→ Scalability

“A Scalable Software-Based Self-Test Methodology for Programmable Processors”,
Why IC Testing is getting more Difficult?

- **Very deep submicron (VDSM) integrated circuits:**
  - billions of transistors (many functional blocks and storage elements); pin counts increase much slower (reducing pin-to-gate ratio)
    - **problem:** internal blocks are hard-to-access (apply tests and observe responses) but methodologies should test each component
  - very dense structure, “new/immature” processes
    - **problem:** new types of physical failures and defects; fault counts increase (extreme number of fault sites and faulty behavior)
  - very high frequencies and low voltage (extreme conditions)
    - **problem:** comprehensive testing only at actual speed of operation (at-speed testing)
  - vulnerability to environmental factors, e.g. radiation (extreme conditions)
    - **problem:** on-line testing in the field is necessary to verify continuous operation of the IC after fabrication even at sea level

- **Yes, indeed, IC testing is a big problem!**
Basic Integrated Circuits Testing Terms

- Manufacturing testing
- On-line/field testing
- Test pattern/vector and test response
- Controllability, Observability
- Testability
- Design-for-Test (DfT)
- Scan-based testing
- Fault model
  - Stuck-at fault model
  - Delay fault models
    - Transition delay – Path delay
- Fault-free/faulty circuit
- Fault detection
- Fault diagnosis
- Test pattern generation
- Fault simulation
- Fault/Defect coverage
- External testing
- Self-testing
  - or Internal testing
  - or Embedded testing
  - Hardware-based or Software-based
IC Tester or Automatic Test Equipment (ATE)
- Equipment for manufacturing IC testing
- “Big iron” or “Functional” testers

Low-cost tester
- ATE with smaller
  - memory size
  - frequency
  - pin count
Verification vs. Testing

- **Verification**
  - checks if a specification is correctly transformed to a design

- **Testing**
  - checks if a design is correctly manufactured in a chip

- **Fault observation**
  - faulty values transferred to “readable” locations: outputs or memory elements
  - big problem in testing
    - takes large test development and test application times…
  - not a problem in simulation-based verification: any signal observed in the simulator
Major IC Testing Challenges and Objectives

- **What should be optimized:**
  - Test development time (better engineering, EDA tools and DFT)
  - Test application time (use of test equipment is costly)
  - Test data volume (tester memory is expensive)
  - Test application frequency (need test quality: at-speed testing)
  - Fault models (closer to actual silicon defects) – test quality
  - Test power consumption (manufacturing testing, on-line testing)
  - ATE inaccuracies (measurements, frequency gap ATE vs. IC)
  - Yield loss (due to tester inaccuracies and over-testing)

- **Any test methodology tries to optimize some (can’t do all...) of the above**
  - Software-based self-testing tries the same
    - Complements already known test techniques
### External Functional Testing

- **External, ATE-based Testing**
- **Functional Testing**
  - ATE (tester) applies test patterns at chip input pins
  - ATE collects chip responses
  - ATE resembles “external world”
  - Large, expensive tester memory for test data (patterns and responses)
External “Functional” Testing for Processors

- Tester must emulate the “external world” of the processor
  - Mainly the memory system
Functional Testing

- **Functional Testing Facts**
  - Expensive ATE
    - Large, high-speed memories for test patterns/responses
  - At-speed testing
  - Fault coverage for different fault models or no fault model
  - No extra hardware
    - No performance or area overheads
  - Can do performance characterization (speed-binning)
  - May not be easy to excite all circuit areas

- **Functional testing alone is not enough**
Functional testing just asks:
- “Does the chip do what it is designed to do?”

**Structural testing adds structural information**
- Circuits elements and their interconnections
- “Are all circuit elements present and working?”
  - If this is true, and the design is correct, then it must work

**Classic structural testing: scan-based**
Structural Testing Facts

- Can use either high-end ATE or low-cost ATE
- High fault coverage
  - Better access to chip’s interior – with scan (DFT) support
- EDA tools support – full automation
- Extra hardware (area and/or performance penalty: 5-10%)
- Very large test data volume (patterns/responses)
  - Large scan chains
- Can’t do performance characterization (speed-binning)
- May lead to over-testing
  - Testing for faults that will never appear in any functional mode
  - Yield loss

- Structural Testing alone is not enough
Self-testing moves the two test phases (test application and response collection) into the chip itself

- No test patterns/responses are stored in ATE
  - Any ATE can be used or no ATE at all
- At-speed testing
  - Chip itself applies and captures
- Can be either functional or structural
  - May use scan infrastructure or not
  - Scan-based, pseudorandom-based self testing (Fig)
- EDA support
  - Logic BIST and Memory BIST
- Extra hardware (area and performance overheads)
- Extra power consumption
- Over-testing
Self-testing for processors or any processor-based SoC can be

- Hardware-based (as for any IC)
  - Extra hardware is added for test application and response capture
  - Pseudorandom Pattern Generators (PRPG), Linear Feedback Shift Registers (LFSR), Multiple Input Signature Registers (MISR)
  - Scan chains

- Software-based (Instruction-based)
  - Processor instructions/programs are executed to apply test patterns to:
    - Internal components of the processor itself
    - Other SoC components out of the processor (memories, etc)

- Key idea
  - Use the Instruction Set Architecture (ISA) and processor resources (instructions, registers, functional units) to test the processor and surrounding components
## Current Industrial Test Practices

<table>
<thead>
<tr>
<th></th>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Scan</strong></td>
<td>Systematic</td>
<td>Overhead Speed test</td>
</tr>
<tr>
<td></td>
<td>High fault coverage</td>
<td></td>
</tr>
<tr>
<td><strong>BIST</strong></td>
<td>At-speed test</td>
<td>Overhead Power / yield loss</td>
</tr>
<tr>
<td></td>
<td>Low-cost ATE</td>
<td></td>
</tr>
<tr>
<td><strong>Functional test</strong></td>
<td>Speed test Low overhead no over-testing</td>
<td>High-cost ATE Manual test writing Low fault coverage</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Combination of these approaches used in practice
- No one-size-fits-all solution
- SBST is one more contribution to the problem
SBST as a New Test Paradigm

- **Goal** – Enabling **scalable low-cost** at-speed self-test for
  - Processors
  - SoCs containing embedded processors

- **Processor**
  - Re-use of the processor for another task: testing
    - Used as general purpose computing platform for test

- **Cost issues**
  - Test application – At-speed test with **low-cost testers**
  - Test generation – **Systematic** test program synthesis

**Software-Based Self-Test (SBST)**
At-Speed Test with Low-Cost Testers and SBST

Traditional Functional Test

- Test program
- Simulation trace @ I/O pins
- High-cost tester
- CPU

Software-Based Self-Test (SBST)

- CPU
- memory
- low-cost tester
- Memory trace
- CPU
- memory
- low-cost tester
- Memory trace
- CPU
- memory
- low-cost tester
- Memory trace
Software-Based Self-Testing: Basic Concept
Low speed up/download – High Speed execution
Self-Test Program Execution Steps: Targeting a Module

1. **Test Preparation**
   - CPU core
   - module under test
   - instruction(s) to “load”

2. **Test Application and Response Collection**
   - CPU core
   - module under test
   - instruction(s) to “execute”

3. **Response Extraction**
   - CPU core
   - module under test
   - instruction(s) to “store”
Software-Based Self-Testing: Variants

- **Manufacturing Testing with SBST**
  - Self-test programs/data should be “cache-resident”
    - No external memory cycles, cache controller should be configured ("Cache Loader" needed): no cache misses
      - Necessary for low-cost ATE use
      - No main memory available during testing

- **Periodic On-Line Testing with SBST**
  - Self-test code/data/responses can be stored in ROM, Flash, RAM
  - Self-test takes place periodically or during idle intervals
    - Operating system support
    - Low-cost reliability mechanism for embedded systems

- **SBST for Processor-based SoCs**
  - Processor self-test programs used to test memories, peripherals, etc
Software-Based Self-Testing: Advantages and Challenges

- **No extra hardware**
  - No area, performance, power overheads; non-intrusive

- **At-speed testing**
  - Tests are applied and responses collected at processor’s speed
    - No yield loss
    - Comprehensive defect coverage

- **No over-testing**
  - Testing is performed in functional/normal mode
    - No yield loss due to over-testing

- **“Added Value” to the Product**
  - Can be re-used during system lifetime

- **Memory for self-test program/data**
  - Does it fit in on-chip cache (manufacturing testing)?
  - Does it for in a Flash or ROM (on-line testing)?

- **Is it able to excite faults in all fault locations?**
  - Controllability
  - Observability

- **Can it be easily automated?**
  - Scan-based testing is a very mature automated approach
  - What about SBST?
An SBST Research Evolution Step 1

Microprocessor

Performance Enhancing Mechanisms

Floating Point Datapath and Control
- Add
- Mult
- MAC

Speculative Execution Units
- Branch Prediction
- Prefetching

Deep Pipelines
- Forwarding
- Interlocking

Microprocessor Integer Datapath and Control
- Register Files
- Control Units
- Simple Pipeline
- Integer Arithmetic Units
- ALU
- Add
- Mult

2004-2007
2000-2004
1995-2000
2006+
An SBST Research Evolution Step 2

SoC

Microprocessor

Performance Enhancing Mechanisms

- Floating Point Datapath and Control
  - Add
  - Mult
  - MAC

Speculative Execution Units
- Branch Prediction
- Prefetching

Deep Pipelines
- Forwarding
- Interlocking

Microprocessor Integer Datapath and Control
- Register Files
- Control Units
- Simple Pipeline
- Integer Arithmetic Units
- ALU
- Add
- Mult

Peripheral Controllers
- Controller 1
- Controller 2
- Controller k

2006-2007
Symmetric Multiprocessor

Microprocessor

Performance Enhancing Mechanisms

Floating Point Datapath and Control

Speculative Execution Units

Branch Prediction Prefetching

Deep Pipelines Forwarding Interlocking

Microprocessor Integer Datapath and Control

Register Files

Control Units

Integer Arithmetic Units

Simple Pipeline

ALU

Add

Mult

MAC

L1 Cache

L2 Cache

L1 Cache

L1 Cache

Symmetric Multiprocessor

Microprocessor

Performance Enhancing Mechanisms

Floating Point Datapath and Control

Speculative Execution Units

Branch Prediction Prefetching

Deep Pipelines Forwarding Interlocking

Microprocessor Integer Datapath and Control

Register Files

Control Units

Integer Arithmetic Units

Simple Pipeline

ALU

Add

Mult

MAC

L1 Cache

L2 Cache

L1 Cache
SBST for Processor Integer Datapaths


- **Inputs**
  - Processor ISA
  - Processor RTL description
  - Test sets library

- **Methodology: three phases**
  - A: Information extraction
  - B: Component classification and test priorities
  - C: Self-test code development for components

- **Phase A: Information Extraction**
  - Processor RTL
  - Processor ISA

- **Phase B: Component Classification and Test Priorities**

- **Phase C: SBST Code Development (per-component)**
  - Test Library

- **SBST Program**
SBST for Processor Integer Datapaths: Phase A


- Extract information using
  - the processor ISA (programmer’s manual)
  - the processor model
    - RT-level description or Architecture Description Language (ADL)
- Processor partitioned as set of components using the RT-level description
- Extracted information (using ISA and RT-level description):
  - the sets of functional component operations (micro-operations)
  - the sets of Basic Test Instructions
    - that excite functional component operations
  - the sets of Peripheral Test Instructions
    - usually instruction sequences instead of single instructions
    - for controlling or observing processor registers
      - setting the operands
      - propagating test response to memory
**Processor Component Classification**
- Functional components
  - Computational (arithmetic/logic operations on data or address operands, i.e. adders, ALUs, barrel shifters, multipliers, MACs etc, multiplier)
  - Interconnect (flow of data or addresses)
  - Storage (feed inputs of data or address computational components and capture their output i.e. register file)
- Control components
- Hidden components

**Test priority assignment for processor components**

**Objective:** reach a high fault coverage ASAP
- Target the most critical-to-test processor components for low test engineering and test application cost

**Criteria**
- Component relative size (large gate count) + Component accessibility

**Data functional components**
- dominate the processor area and are easily accessible

**Low-cost SBST:** data functional is the highest test priority
- Processor fault coverage not sufficient? Proceed to other components
SBST for Processor Integer Datapaths: Phase C Component Self-Test Development


Select instruction (or instr. sequence) for register control

Select instruction that excites comp. operation with specific operands

Select instruction (or instr. sequence) for register observation

Peripheral instructions → Basic test instruction

Phase A (information extraction)

Component test set re-use library

Component self-test routine
SBST for Processor Integer Datapaths: Benchmarks


<table>
<thead>
<tr>
<th>Processor Name</th>
<th>Architecture Implementation</th>
<th>Complexity</th>
<th>Usefulness of Experiment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma</td>
<td>Simple RISC processor, Princeton architecture, 3-stage simple pipeline, no data forwarding, public available.</td>
<td>Simple</td>
<td>First application of the methodology to a RISC processor. Many different synthesized versions used.</td>
</tr>
<tr>
<td>ASIP</td>
<td>RISC processor, Harvard architecture, 5-stage pipeline. Public available limited version.</td>
<td>Average</td>
<td>Application of the methodology on a RISC processor generated automatically by an ASIP suite.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Processor core</th>
<th>Gate count</th>
<th>FC (%)</th>
<th>Size (words)</th>
<th>Time (clock cycles)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Plasma (Synthesis A)</td>
<td>26,080</td>
<td>95.3</td>
<td>853</td>
<td>5,797</td>
</tr>
<tr>
<td>Plasma (Synthesis B)</td>
<td>27,824</td>
<td>95.3</td>
<td>853</td>
<td>5,797</td>
</tr>
<tr>
<td>Plasma (Synthesis C)</td>
<td>30,896</td>
<td>94.5</td>
<td>853</td>
<td>5,797</td>
</tr>
<tr>
<td>ASIP</td>
<td>37,400</td>
<td>92.6</td>
<td>1,728</td>
<td>10,061</td>
</tr>
</tbody>
</table>
SBST Enhancement for Pipelined Processors


- SBST targeting “visible” processor components sufficient for:
  - Non-pipelined processors
  - Simple-pipelined processors
- Are “hidden” mechanisms sufficiently tested as a side-effect?
  - No!
- Experiments on two fully pipelined public available models:
  - miniMIPS and OpenRISC 1200
    - Models available at: http://www.opencores.org
  - MIPS I architecture
  - 5-stage pipeline architecture
    - Hazard detection, data forwarding and pipeline interlock mechanisms
Low Testability of Pipeline Logic (miniMIPS)


Fault Coverage

- > 95%
- < 90%
- < 80%
- < 60%

PGC : Percentage Gate Count

FC of the pipeline logic ~ 72%
Overall FC of the processor ~ 86%
Two Testability Problems: Address Logic and Hazard Detect/Resolve Logic

- **Address-related logic is poorly tested**
  - Address registers of the pipeline stages and components that handle addresses, i.e. PC, Bus controller

- **Low controllability**
  - Fault excitation requires every address line to be set to both 0 and 1

- **Low observability**
  - Fault propagation can be done only through the IF and MEM stages

- **Same problem occurs also in non-pipelined processors**
  - Accentuated in pipelined models. More than 60% of the undetected faults of the pipeline

- **Hazard detection and resolution logic is poorly tested**
  - Hazard detection and interlocking logic
  - Forwarding paths

- **Fundamental difficulty: “invisible” to programmer**
- **Not explicitly targeted**

- **Pipeline stages that involve forwarding paths (i.e. EX, MEM) present much lower fault coverage in comparison with other stages (i.e. IF, ID)**
Solution for Controllability of Address Logic

For the low controllability of the address related faults

Solution:
- Execution of SBST code located in different regions of the entire memory space

Key idea:
- SBST program is partitioned into multiple code segments virtually stored and executed from different memory regions

Constraint: a huge memory model should not be used
- Fault simulation: large simulation time
- Actual testing: huge memory in the loadboard

For the low observability of the address related faults

Considering that the address bus of the processor is observed during testing
- Two different paths for fault propagation
  - Instruction fetch stage (path 1) and Execute and then instruction fetch stage (path 2)

Considering that responses can be captured only through data bus
- Two different paths for fault propagation
  - Jump-and-link (path 3) and Exceptions (path 4)

Solution for Controllability of Address Logic (cont.)


Self-test program
4KB
(relative addresses)

Partitioned into
4 code segments

Virtual memory space
4GB
(virtual address)

Partitioned in 1024 regions

Physical memory
4MB
(physical address)

Each region is mapped to the same physical space

Virtual addresses are translated to physical address
Solution for Hazard Detect/Resolve Logic

- Low testability of hazard detection and forwarding

Solution:
- Execution of test sequences that increase pipeline activity

Key idea:
- Use existing SBST routines: operands diversity
- Create multiple instantiations: code variants

Basic code

```
lw    R1, Imm1(R0)
addi   R2, R1, Imm2
add    R4, R3, R2
sw     R4, Imm3(R0)
```

Def-use pairs

```
1
2
3
```

Enhanced code

```
lw    R1, Imm1(R0)
addi   R2, R1, Imm2
nop
add    R4, R3, R2
nop
nop
sw     R4, Imm3(R0)
```
Experimental Results on SBST Enhancement


<table>
<thead>
<tr>
<th>Processor</th>
<th>Original FC % (stuck-at)</th>
<th>Enhanced FC % (stuck-at)</th>
</tr>
</thead>
<tbody>
<tr>
<td>miniMIPS (A)</td>
<td>86.58</td>
<td>95.08</td>
</tr>
<tr>
<td>miniMIPS (B)</td>
<td>81.51</td>
<td>94.00</td>
</tr>
<tr>
<td>OpenRISC</td>
<td>80.36</td>
<td>90.03</td>
</tr>
</tbody>
</table>

- Overall fault coverage for pipelined processors
- miniMIPS (A): includes a fast parallel multiplier (original design)
- miniMIPS (B): without multiplier
- Enhanced code size: 1.6K (from 1.2K) for miniMIPS and 3.1K (from 2.2K) for OpenRISC
- Enhanced code execution cycles: 7K cycles (from 6K) for miniMIPS and 57K cycles (from 44K cycles) for OpenRISC
SBST for Processor Floating Point Units

- **Testability conditions for complex FP blocks**
  - E.g. Two-path FP adder

- **Main problem for SBST Development:**
  - Bit manipulation code to apply test sets to integer components inside the FP unit
    - Exponent/Significand
  - FP instruction sets do not support bit manipulation
    - Use integer instructions instead

- **Applied to MIPS processor with FP support**
  - 96.4% fault coverage
  - Only 2.5K words program

---


SBST for Speculative Execution Units (SpEUs)

- Used for performance optimization (Branch & Value Prediction, Data Prefetch)

Testing of SpEUs is particularly challenging due to:
- Self-correction mechanisms. Produce same functional result even if faulty
  - Only performance degradation due to incorrect speculations

Software-based self-testing methodology using processor hardware resources
- Performance monitoring hardware and Processor exceptions mechanism

Propagate address value to processor outputs

Collect information about the SpEU

Results
- 96%-97% for Branch Prediction Unit (with 1-bit and 2-bit prediction schemes
- ~93% fault coverage for the entire MIPS processor including the BPUs
SBST for Communication Peripherals in SoCs

- **Embedded processor in SoC used to self-test communication peripherals**
  - Commonly used in a wide range of System-on-Chip (SoC) families
  - Occupy a significant portion of the SoC

- **Application**
  - UART and Ethernet cores (with 16 and 32 deep FIFOs): 92% to 95% fault coverage for the peripheral cores with 2K to 6K word programs.

---


---

**User Actions**
- Identify core’s register set
- Identify operating modes
- Select appropriate operating mode
- Identify error conditions
- Identify bus parameters
- Identify processor ISA details

**Methodology Steps**
- Generate tests for the operating modes
- Generate tests for the Tx & Rx FIFOs
- Generate tests for the error handling logic
- Generate tests for the bus interface logic
- Transform pseudocode into actual instructions

**SBST code**
- Test routines templates
- Pseudo test routines
- Final test routines
### SBST Research

- **Other major academic research groups on SBST**
  - The University of Texas at Austin, USA
    - Prof J. Abraham, Pioneer in the area of functional testing for processors since the 80’s
    - On-going high quality research on the topic
  - University of California San Diego and Santa Barbara, USA
    - Profs K-T. Cheng, S. Dey and L.-C. Wang
  - Politecnico di Torino, Italy
    - Prof M. Sonza Reorda
  - Nara Institute of Science and Technology, Japan
    - Prof H. Fujiwara
  - Princeton University, USA
    - Prof N. Jha
  - Case Western Reserve University, USA
    - Prof C. Papachristou
  - University of Stuttgart, Germany
    - Prof H.-J. Wunderlich

- **Industrial published work on SBST**
  - Intel (IEEE International Test Conference 2002)
  - Sun (IEEE International Test Conference 2006)
  - NEC (ACM/IEEE Design Automation Conference 2003 and 2006)
Pioneer work on processor functional testing

Based on a functional fault model
- No structural fault simulation/coverage results
- No automation

Back in 1998: VERTIS
- Exercise functional operations of processor; no fault model
- Applicable to design validation and manufacturing testing

Results
- Viper 32-bit processor, 4600 gates, 251 ffs, 94.04% fault coverage in open-loop (using tester)
- GL85 (8085 model), 6300 gates, 244 ffs, 90.20% fault coverage in open-loop mode, 86.70% in closed-loop (self-test) mode with 360,000 instructions
SBST @ UT Austin (cont.)


- **Fully automated flow for:**
  - mapping of pre-computed module tests to instruction sequences
  - propagation of module test responses to primary outputs

- **Uses for test application:**
  - temporal logic descriptions for component tests
  - bounded model checking (BMC) from formal verification
    - looks for counter-example on the negated temporal logic properties
    - counter-example gives the instruction sequence
    - solutions may exist although counter-example may not be found always

- **Uses for fault propagation:**
  - Boolean difference formulation
  - Linear time logic passed to a bounded model checker again

- **Results:**
  - 82% fault coverage on OpenRISC 1200 (speculate 90%+ test coverage)
Automation-oriented approach

Using

- Test program templates (parameterized)
- Controllability and observability constraints
- Virtual constraint circuits (VCC)
  - To generate module-level tests
- Synthesis of self-test programs
  - From module-level tests

Experimental results

- First results on Parwan toy-processor (888 gates, 53 ffs)
  - 91.42%
- Then, only on one module Tensilica’s Xtensa (ALU+shifter+logic)
  - 95.2% fault coverage using a self-test program of 20K bytes running for 27K cycles (approx. 25K faults in the ALU)
SBST @ UC San Diego (flow)


Instruction Set Architecture (ISA)

Set of test program templates: T

RTL description of processor

Set of combinational MUTs: M

∀m ∈ M

Template ranking: T_m

t ∈ T_m

1. Simulation-based constraint extraction

C_m,t

2. Constrained test generation (virtual constraint circuit)

P_m,t

3. Test program synthesis

TP_m,t

4. Processor-level fault simulation

Y

More t in T?

N

Acceptable fault coverage?

Y

More m in M?

N

Done
New perspective for SBST

- Instead of using SBST for defect/fault screening group suggests its use for at-speed functional testing and speed-binning
  - Because a functional-based approach alone may not be sufficient for full testing
- Random Test Program Generation (RTPG) combined with Target Test Program Generation (TTPG)
  - TTPG uses simulation to develop learned models for the modules around the module under test
    - Thus improve the efficiency of generated test programs
    - Reduces complexity of test generation
  - For Random/boolean modules Boolean learning is used
  - For Datapath/arithmetic modules Arithmetic learning is used

Experimental results

- OpenRISC processor’s ALU
  - 94.94% fault coverage, 97.40% test coverage (~20K stuck-at faults in the ALU)
Innovative automatic self-test program generation approach
- Genetic algorithms based evolution framework based on fault simulation and gate-level netlist
- μGP evolutionary algorithm: improves and fine-tunes self-test programs
- Results: 90.77% for i8051 (12K gates netlist)

Group's SBST-related research extends to:
- SoC testing using embedded processors
  - Memories, peripherals
- Silicon-debug combining SBST and scan-based test
- Software-based self-diagnosis (SBSD)
Recently focused on delay fault testing for pipelined processors

- Uses graph-theoretic approach of pipeline to develop path delay fault tests
  - Pipeline Instruction execution graph (PIE graph)
- Constraint-based test development using ATPG

Experimental results
- VPRO, 16-bit, pipelined
- DLX, 32-bit, pipelined
- Detection of all functionally testable path delay faults
  - 100% efficiency

PIE graph (Pipeline Instruction Execution graph)
- Size of PIE graph may grow fast

ISA and RTL description
PIE-graph construction
Path classification and constraint extraction
Constrained TPG
Test instruction sequence generation
FRITS – Functional Random Instruction Testing at Speed

Functional BIST for Intel Microprocessors
- Key objective is to enable functional testing using low-cost testers
  - Low-cost testers have been used for structural testing (scan-based)
  - DFT applied to enable functional testing on low-cost, low-pin count testers
  - Automatic tool that generates self-test programs

Experimental results
- Pentium 4: ~70% total fault coverage
  - Components fault coverage 20%-90%
- Itanium: ~85% total fault coverage

SBST can be a great supplement to
- Full functional test
- Scan-based structural test
FST Test Sequence and FRITS Kernel Execution Flow

1. Assert Reset
2. Load Functional Test Into Cache using Cache-Loader with processor in Test Mode
3. Transition processor into Normal Mode with Cache-able Starting Address
4. Execute FST Test
5. Transition processor into Test Mode and Unload results
6. Test Sequence Generation
7. Test Data Generation
8. Test Execution & Result Compression
9. Data loop count met? (NO)
10. Instruction loop count met? (NO)
11. Data loop count met? (YES)
12. Instruction loop count met? (YES)
13. End
Cache-Resident approach for Processor Testing
- *Load&Go* testing or *Cache Resident* testing

Microprocessors complexity
- More on-die cache
- High-speed, serial interfaces
  - Non-determinism of serializer/deserializer (SerDes) circuits
    - “packets” may come later or earlier in time
- 4 or 8+ cores

Functional BIST for Sun Microprocessors
- Key objective *again* is to enable functional testing using low-cost testers
  - Sun wants to move from big-iron functional testers to low-cost ATE
- Used it for *UltraSparc T1* (8 cores, 32 threads)
- This can be done if non-deterministic SerDes and memory access is avoided
  - Thus cache-resident testing
    - Only size and architecture of the on-chip cache is the limitation
SBST @ Sun (cont.)

- Determining the contents of the cache (L2) using RTL simulation
  - Pass 1 and pass 2
- Easy case for direct-mapped caches
  - More tricky in associative caches
- Test conversion for Load&Go test

---

Conclusions and Open Issues

Software-Based Self-Test can enable low-cost, high-quality functional test

- At-speed self-test with low-cost testers
- Scalable, adaptable, systematic, automatable SBST methodologies have been presented in literature
- Demonstration on commercial processors and several good quality research benchmarks
- Complement for
  - Scan tests – Detecting speed defects
  - Random functional tests – Top-off mechanism for improving fault coverage
- Need more research or improvements
  - SBST for superscalar and out-of-order execution processors
  - SBST for multiprocessors
  - Power-aware SBST
  - SBST Periodic On-Line Testing