Key Problems and Research Directions

Advancing integrated circuit technology

- Increasing numbers of transistors
  → very complex designs, difficult to verify

- Smaller geometries
  → more subtle defects, difficulty to achieve test quality

- Increasing use of computers and ICs in society
  → need to deal with failures in applications

Acknowledgement: My graduate students over the years
Integrated Circuits and Systems Group

10 full-time faculty in CAD and IC Design, adjuncts from industry

- Design of CMOS ICs for a variety of applications
  - Low power, high performance digital functions
  - Analog and mixed-signal blocks
  - RF subsystems
- Development of CAD tools
  - Design under variability
  - Design for manufacturability
- Techniques for verifying and testing
  - Digital, analog and RF systems
- Interdisciplinary research projects

Close ties with industry
Example Research (CAD Tools, IC Design)

David Pan: Box Router

Ranjit Gharpurey: Broadband interference sensing and cancellation for RF systems

Arjang Hassibi: Bioluminescence Chip

Hassibi and Abraham: 6-bit, 300MS/sec, 2.7mW ADC with delay lines

Jacob Abraham: 128 point, 6-bit 3GS/sec. FFT
Reliability in the Life of an Integrated Circuit – I

Design “bugs”
Verification (Simulation, Formal)

Design

Process variations, defects
Process Monitors

Wafer

Fabrication
Reliability in the Life of an Integrated Circuit – II

Wafer Probe

Package

Tester

Test cost, coverage
Design for Test, Built-In Self Test

System

Application

Test escapes, wearout, environment
System Self-Test, Error Detection, Fault Tolerance
Analyzing Complex Designs

Need to (implicitly) search a very large state space

- Find bugs in a design
- Generate tests for faults in a manufactured chip

Basic algorithms for even combinational blocks (SAT, ATPG) are NP-complete

Approaches to deal with real designs

- Exploit hierarchy in the design
- Develop abstractions for parts of a design

State-space explosion: A design with 300 state variables has more states than the number of protons in the universe ($10^{80}$)!
Use of ATPG to Check Properties

Some work in checking safety properties

Detecting a “stuck-at-0” on p is equivalent to establishing EFp

Use of ATPG makes it easy to deal with tri-state values, multiple clocks, etc.

Have developed monitor state machines for different types of properties
Abstracting Designs Using Slicing

Static program slicing used for analysis of software (proposed in 1984)

Slice is a subset of the program behavior with respect to some slicing criterion

Slice based on a subset of variables yields a simpler program which is relevant to those variables

Have developed slicing techniques for Verilog
Antecedent Conditioned Slicing for Verification
(work with Vasudevan, Emerson)

Slicing to remove part of design irrelevant to property being verified

- Do not need to preserve program execution where the antecedent is false

Example property for USB 2.0 function core:

\[ G((state == \text{SPEED\_NEG\_FS}) \Rightarrow X((mode\_hs) \land (T1\_gt\_3\_0ms) \Rightarrow (next\_state == \text{RES\_SUSPEND}))) \]

If the machine is in the speed negotiation state, then in the next clock cycle, if it is in high speed mode for more than 3 ms, it will go to the suspend state.
Results on Temporal USB Properties

CPU seconds, 450 MHz dual processor UltraSPARC-II with 1 GB RAM
Manufacturing Test Costs

Why doesn’t the cost of testing a transistor scale like the cost of manufacturing the transistor?
Variations in devices due to subwavelength lithography, random dopant fluctuations, etc.

Experiments on real chips

- Some tests for logic-level “stuck-at” faults do not detect defects unless they are applied at speed

Interconnect opens are resistive (not complete breaks)

- example: Cu interconnect with barrier materials
- effect: delay faults
Effects on Chip?

Changes in delays of paths

Effects could be distributed across paths

- At-speed functional tests are better for delay defects
- **Solution:** at-speed tests
- **Problem:** tester costs
- Need a technique which uses low-cost testers
Software-Based (Native-Mode) Self Test for Processors

- Why not use functional capabilities of processors to replace BIST hardware?
  - No additional hardware
- Reduce test costs by using low-cost testers
- Increase coverage of delay defects and increase yield by testing native
- No issues with excessive power consumption during test

Developed at University of Texas (Int’l Test Conference 1998)

Application to processors at Intel (Int’l Test Conference 2002)
Automatic Generation of Instruction Sequences for Small Delay Defects

- Feedback: heuristics to speed up search
- Phase 1: all paths above a delay threshold
- Phase 2: longest paths through all nodes
- Delay-Based ATPG: generate “TRUE” paths above given delay threshold
- Functional mapping: using verification engine
Results on OR1200 processor

www.opencores.org, synthesized for 0.18µ TSMC process

Results for Phase 1 (paths > 80% of clock)

<table>
<thead>
<tr>
<th>No. of Paths</th>
<th>Drop</th>
<th>Functionally Testable</th>
<th>Functionally Redundant</th>
<th>Time out</th>
</tr>
</thead>
<tbody>
<tr>
<td>27424</td>
<td>12</td>
<td>15118</td>
<td>12106</td>
<td>200</td>
</tr>
</tbody>
</table>

Results for Phase 2
N: % nodes with test for longest path through them

<table>
<thead>
<tr>
<th>Module</th>
<th>Functionally Testable</th>
<th>Functionally Redundant</th>
<th>Rejected Sub-paths</th>
<th>N (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>or1200_ctrl</td>
<td>1826</td>
<td>29191</td>
<td>68087</td>
<td>90.6</td>
</tr>
<tr>
<td>or1200_alu</td>
<td>1427</td>
<td>16985</td>
<td>2716</td>
<td>100</td>
</tr>
<tr>
<td>or1200_lsu</td>
<td>970</td>
<td>4077</td>
<td>3744</td>
<td>100</td>
</tr>
<tr>
<td>or1200_wbmu</td>
<td>1146</td>
<td>2285</td>
<td>2118</td>
<td>100</td>
</tr>
</tbody>
</table>
Test of SoC Cores using Embedded Processor

Wishbone and 128-bit AES designs from opencores.org

Validation vectors: random values encrypted/decrypted

<table>
<thead>
<tr>
<th>AES Core</th>
<th>Inputs</th>
<th>69</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Outputs</td>
<td>33</td>
</tr>
<tr>
<td></td>
<td>Combinational primitives</td>
<td>9225</td>
</tr>
<tr>
<td></td>
<td>Sequential primitives</td>
<td>1119</td>
</tr>
<tr>
<td></td>
<td>Stuck-at faults</td>
<td>64070</td>
</tr>
</tbody>
</table>

Result of Mapping AES tests to ARM instructions (one case)

<table>
<thead>
<tr>
<th>Test</th>
<th>Size (bytes)</th>
<th>Fault coverage(%)</th>
<th>Original Coverage(%)</th>
<th>No. of Cycles</th>
<th>Original Cycles</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9128</td>
<td>90.15</td>
<td>90.35</td>
<td>7816</td>
<td>7435</td>
</tr>
</tbody>
</table>

The University of Texas at Austin
Jacob A. Abraham
April 9, 2009
Testing Analog/Mixed-Signal/RF Circuits

Have to deal with continuous signals

Customers want a guarantee of specifications

A defect may or may not affect the desired behavior of a chip

Tests are for the specifications, not for defects

Similar trend in digital: testing for distributed path delays

Test costs very high to test every specification

Find simple test measures, calibrate using accurate measurements
Loopback + DFT Scheme

Provide dynamic performance parameters of individual signal paths

- Avoid yield loss due to fault masking

DFT circuitry – analog filter and adder on loadboard or on chip

- Characterize harmonic distortion and noise parameters
Validation: Hardware Measurements

Broadband modem IC
  - Tx/Rx data rates up to 80MSPS
Programmable 3-pole filter
  - Bypassed in normal mode
Faults injected by
  - Reconfiguring TX/RX gain
  - Sweeping supply voltage, input amplitude

![Image of hardware](image_url)

![Graphs of SINAD](image_url)

(a) DAC channel
(b) ADC channel
RF Built-In Test using Amplitude Detectors

Alternate test methodology

High input impedance (7.6KOhm@1GHz) for detector

Detector output mapped to RF circuit specifications

Low frequency output signal (sampling frequency of 10MHz for mixer test, DC for amplifier test)

Strong correlations with RF circuit parameters
Die Photo of 940 MHz GSM Transceiver

0.18µ CMOS technology

10 MHz output from sensors used to predict specifications

Receiver: \(0.5 \times 1.2 \ mm^2\); Detector: \(0.06 \times 0.072 \ mm^2\)

Area overhead: 1.4%
Measurement Setup

Die wire-bonded to RF PCB, with on-board tuning knobs
Agilent E8257D signal generator
Agilent E4448A spectrum analyzer
Tektronix DPO 7104 digital oscilloscope
LabView and Agilent VEE automation programming
Measurement Results: LNA in RX

RMS Error between prediction and measurement was 0.09dB for LNA Gain and 0.15dBm for LNA IIP3

Relative error was less than 5%
System-Level Fault Tolerance

- Application-specific techniques for numerical computations
- Algorithm-Based Fault Tolerance (Abraham and students, from 1982)
- Overhead decreases as system becomes larger
- Checksum extended to analog subsystems (Chatterjee, 1991)
Control Flow Error Detection using Assertions (CEDA)

- Correct execution of a system disrupted by
  - hardware faults
  - malicious attacks
- Many instances of incorrect execution detected by monitoring control flow
- Developed techniques with low overhead to detect deviations in program flow (one target: embedded systems)
- Fault injection in SPEC benchmark programs showed our approach to be much better than existing techniques
- Demonstrated applicability to security
Control Flow Errors (CFEs)

Execution of incorrect sequence of instructions
- Transient or permanent faults
- External attacks (example: buffer overflow attacks)

Detection techniques
- Hardware – cost too high for some applications
- Software – existing techniques have high performance overhead for reasonable cost
CEDA: Control Flow Error Detection through Assertions

S: global runtime signature register
- updated at the beginning and end of each node
- each update either an XOR or an AND operation

Se: expected value of S at each point in the program
- calculated at compile time

Check point: S is checked against its expected value
- detects CFE if one occurred
- not required inside every node

Node signature: expected value of S inside a node

Node exit signature: expected value of S immediately after exiting a node
Integration with GCC

1. GCC language dependent processing
2. GCC language independent and machine independent processing
3. Generate control flow representation
4. Determine check points depending on error latency required
5. Calculate expected node signatures and node exit signatures
6. Insert update statements to reflect expected signatures
7. Insert check statements at check points
8. GCC machine dependent processing
Application to Fault Tolerance

Application to SPEC benchmarks

Prior techniques, YACCA and CFCSS also implemented

Faults injected using Fiesta
  - gdb based fault injection tool

Fault models:
  - branch deletions
  - branch creations
  - branch operand corruptions
Application to Security

Most external attacks will subvert control flow of the system
- Buffer overflow attacks and viruses will result in unwanted system calls

Functions protected by CEDA cannot be called from unauthorized points in the code

Security experiments
- Protected a library function by implementing CEDA
- Buffer overflow test suite developed by Wilander et al.
- 18 different types of buffer overflow attacks to call the library function

Results:
- Detected and prevented all buffer overflow attacks
- Previous prevention techniques not applicable to more than 50% of these attacks