ECE382M.20: System-on-Chip (SoC) Design

Lecture 14 – SoC Verification & Test

Sources:
Jacob A. Abraham

Andreas Gerstlauer
Electrical and Computer Engineering
University of Texas at Austin
gerstl@ece.utexas.edu

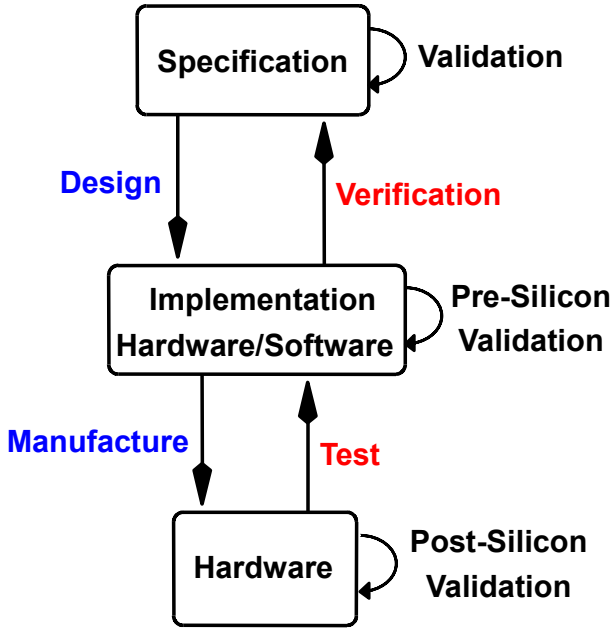

The University of Texas at Austin
Chandra Department of Electrical and Computer Engineering
Cockrell School of Engineering

Outline

- **Introduction**
  - Verification vs. test
- **Verification methods**
  - Verification flow
  - Simulation-based techniques
  - Formal analysis
  - Semi-formal approaches
- **SoC manufacturing test**
  - Design for Test (DFT)
  - SoC testability features
  - Built-in self test

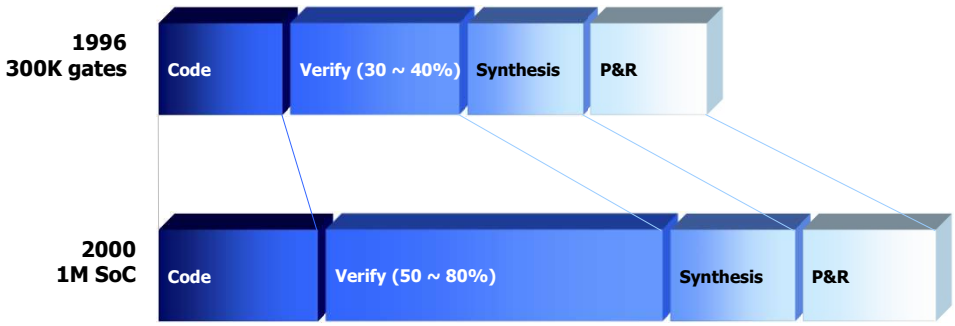
© 2026 A. Gerstlauer

Verification versus Test



Verification Effort

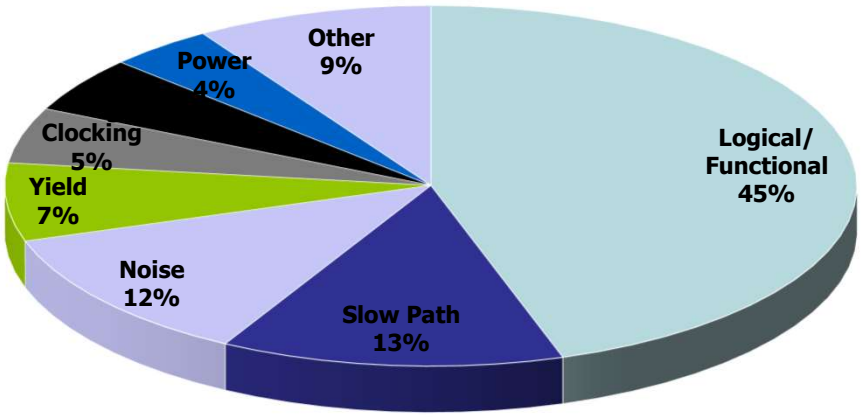
- **Verification** portion of design increases to anywhere from **50 to 80%** of total **development effort** for the design.



Verification methodology manual, 2000-TransEDA

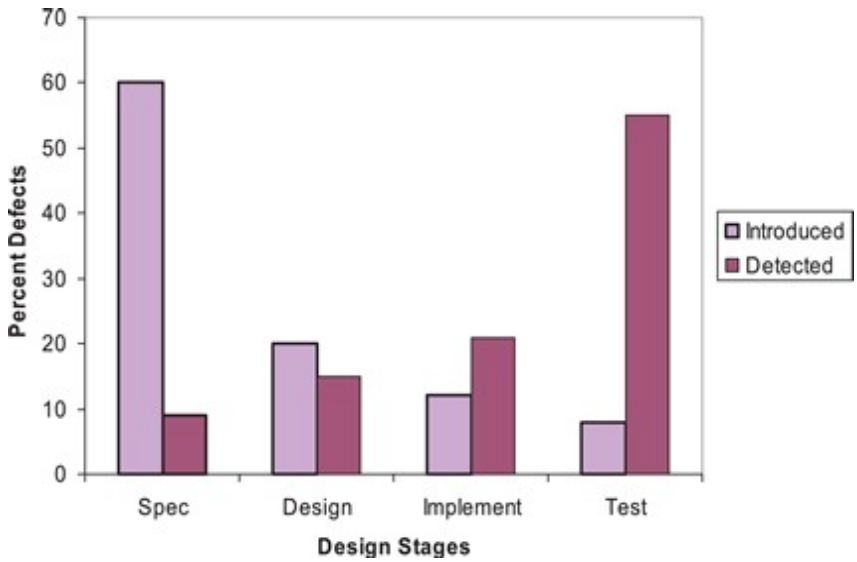
Percentage of Total Flaws

- About 50% of flaws are functional flaws
  - Need verification method to fix logical & functional flaws

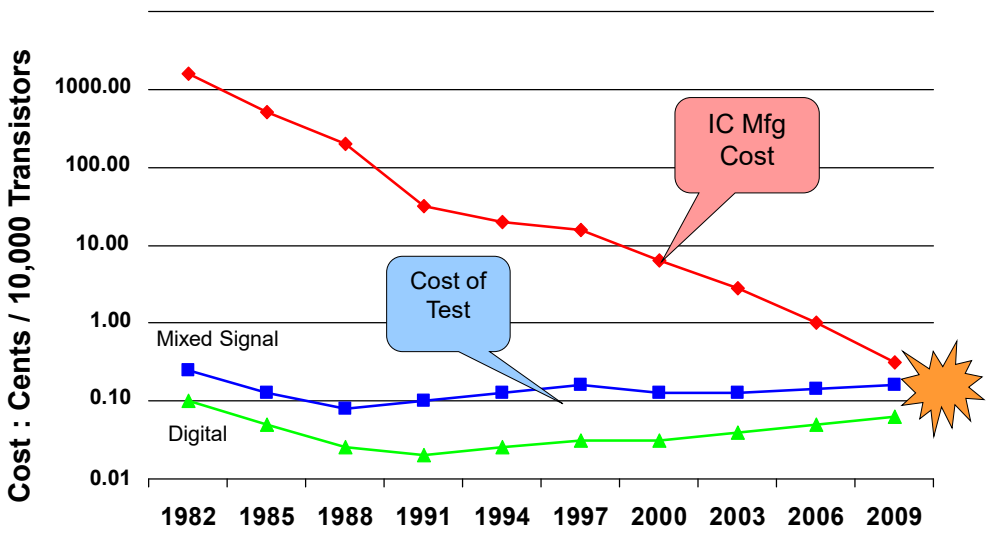


From Mentor presentation material, 2003

"Bug" Introduction and Detection



The Manufacturing Test Problem

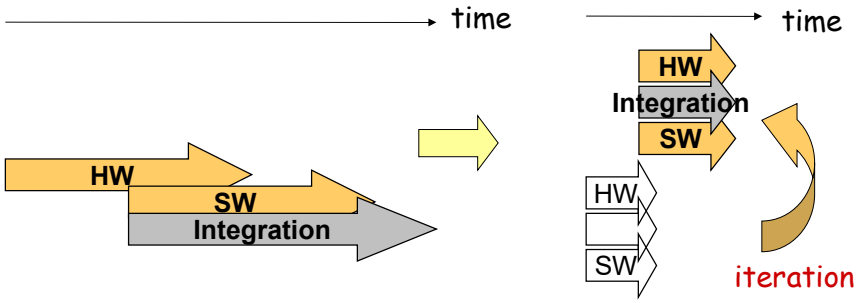


Outline

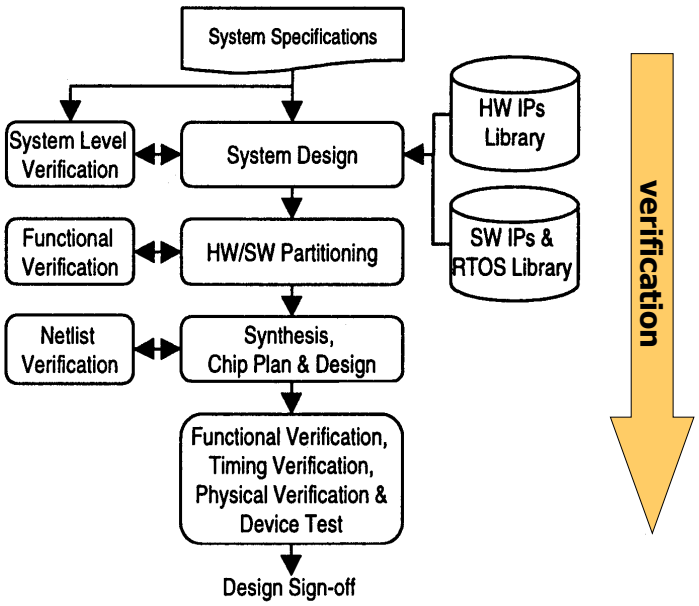
- ✓ Introduction
  - ✓ Verification vs. test
- **Verification methods**
  - Verification slow
  - Simulation-based techniques
  - Formal analysis
  - Semi-formal approaches
- SoC manufacturing test
  - Design for Test (DFT)
  - SoC testability features
  - Built-in self test

HW/SW Co-Design

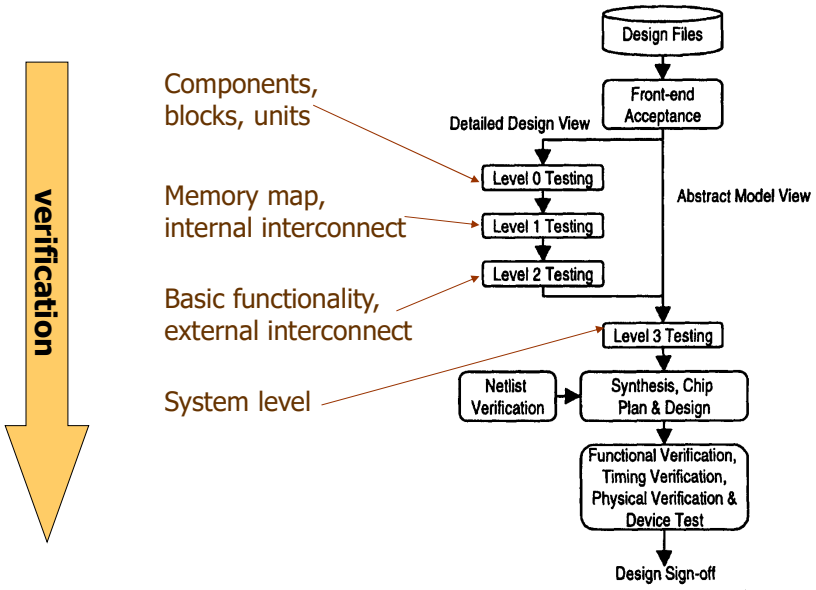
- Concurrent design of HW/SW components
- Evaluate the effect of a design decision at early stage by "virtual prototyping"
- Co-verification



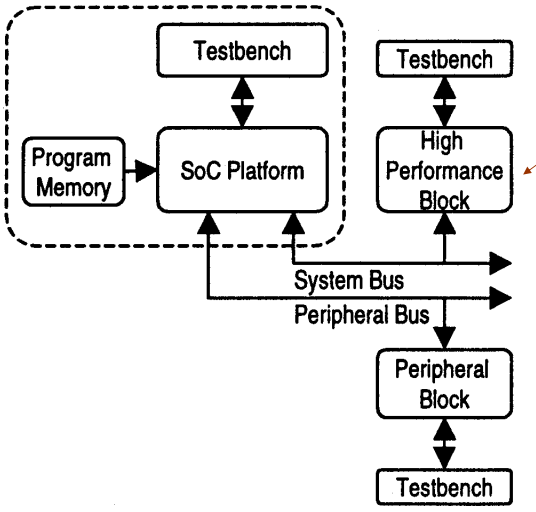
Top-Down SoC Verification



Bottom-Up SoC Verification



Platform Based SoC Verification

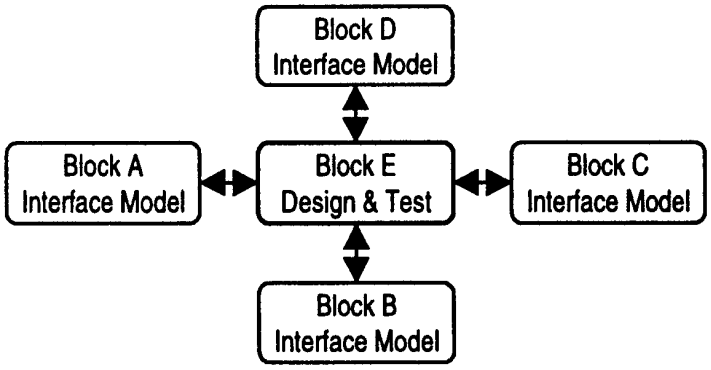


Derivative Design

Interconnect Verification between:

- **SoC Platform**
- **Newly added IPs**

Interface-Driven SoC Verification



**Besides Design-Under-Test,
all others are interface models**

Traditional Simulation

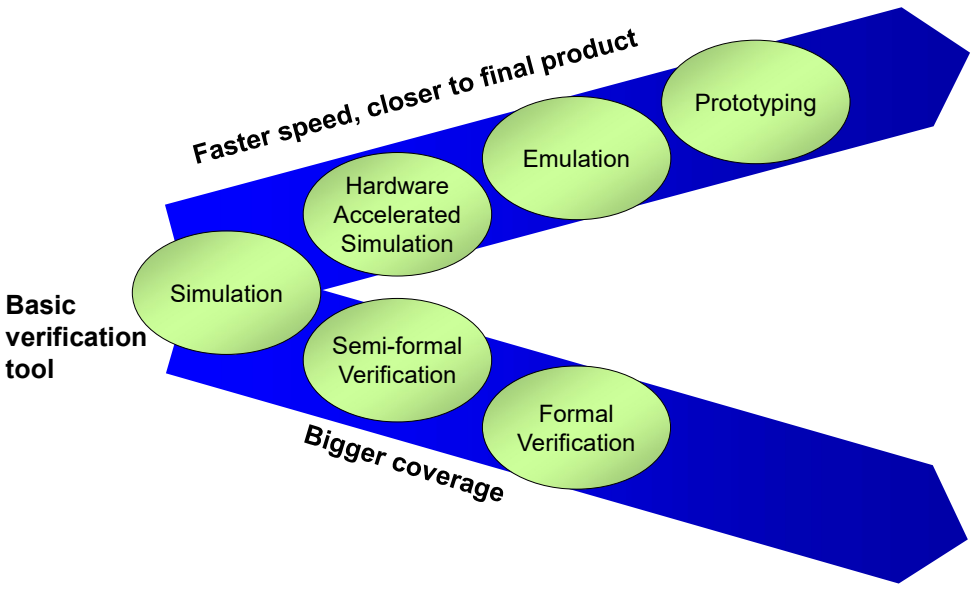


- **Problems of traditional testbench**
  - Real-World Stimuli
  - System-Level Modeling
  - High-Level Algorithmic Modeling
  - Test Automation
  - Source Coverage

Design Verification Methods

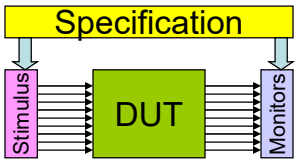
- **Simulation based methods**
  - Specify input test vector, output test vector pair
  - Run simulation and compare output against expected output
- **Formal Methods**
  - Check equivalence of design models or parts of models
  - Check specified properties on models
- **Semi-formal Methods**
  - Specify inputs and outputs as symbolic expressions
  - Check simulation output against expected expression

Verification Approaches



Simulation

- **Create test vectors and simulate model**
  - Simulation, debugging and visualization tools [Synopsys VCS, Mentor ModelSim, Cadence NC-Sim]



- **Inputs**
  - Specification
    - Used to create interesting stimuli and monitors
  - Model of DUT
    - Typically written in HDL or C or both
- **Output**
  - Failed test vectors
    - Pointed out in different design representations by debugging tools

Simulation Technologies

- **Different techniques at varying levels of abstraction**
  - Numerical Simulation (MATLAB)
  - AMS Simulation
  - Transaction-based Simulators
  - HW/SW co-simulation
  - Cycle-based Simulators
  - Event-based Simulators
  - Emulation Systems
  - Rapid Prototyping Systems
  - Hardware Accelerators

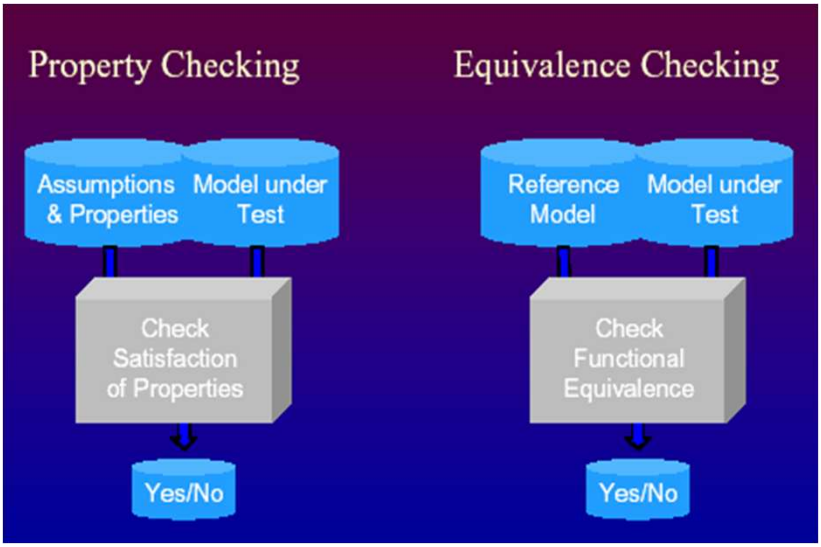
Static Technologies

- **"Lint" Checking**
  - Syntactic correctness
  - Identifies simple errors
- **Static Timing Verification**
  - Setup, hold, delay timing requirements
  - Challenging: multiple sources

Formal Techniques

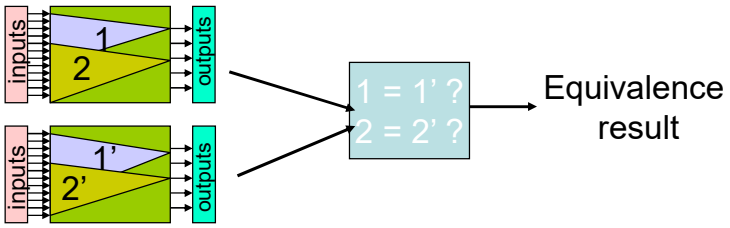
- **Theorem Proving Techniques**
  - Proof-based
  - Not fully automatic
- **Formal Model Checking**
  - Model-based
  - Automatic
- **Formal Equivalence Checking**
  - Reference design ↔ modified design
  - RTL-RTL, RTL-Gate, Gate-Gate implementations
  - No timing verification

Formal Verification of SoCs

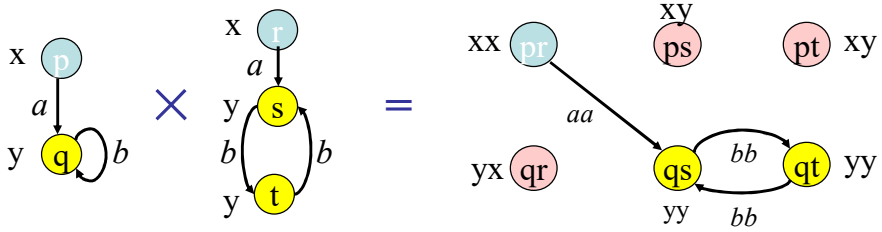


Equivalence Checking

- **LEC uses boolean algebra to check for logic equivalence**

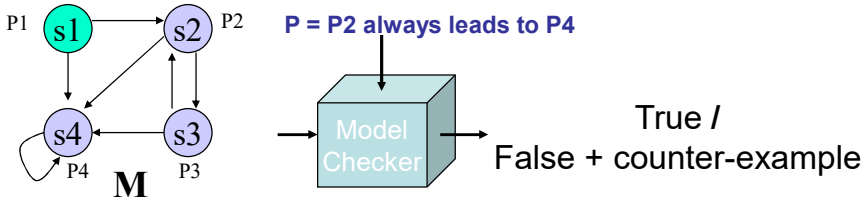


- **SEC uses FSMs to check for sequential equivalence**

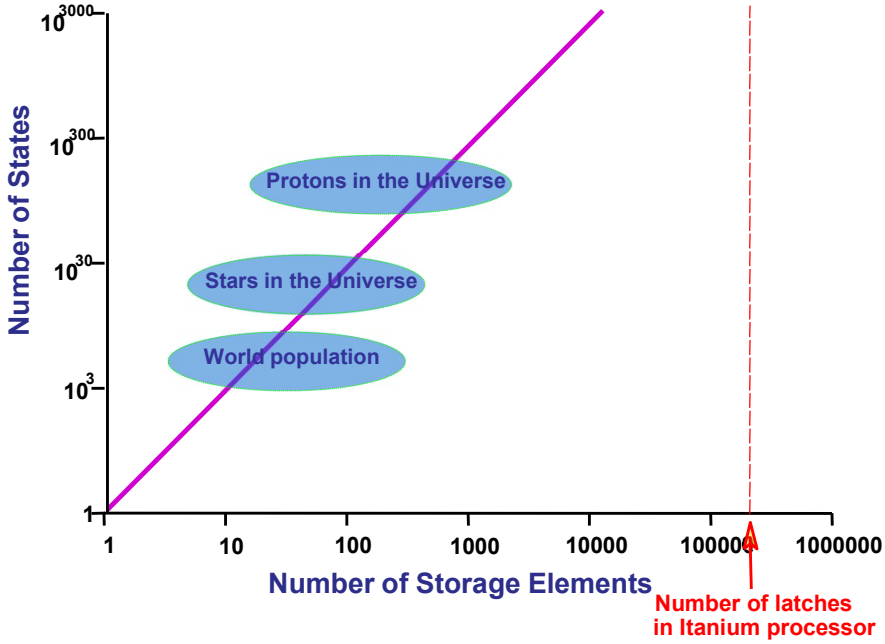


Model Checking

- **Model *M* satisfies property *P*? [Clarke, Emerson '81]**
- **Inputs**
  - State transition system representation of *M*
  - Temporal property *P* as formula of state properties
- **Output**
  - True (property holds)
  - False + counter-example (property does not hold)



State Explosion!



Abstractions to Deal with Large State Spaces

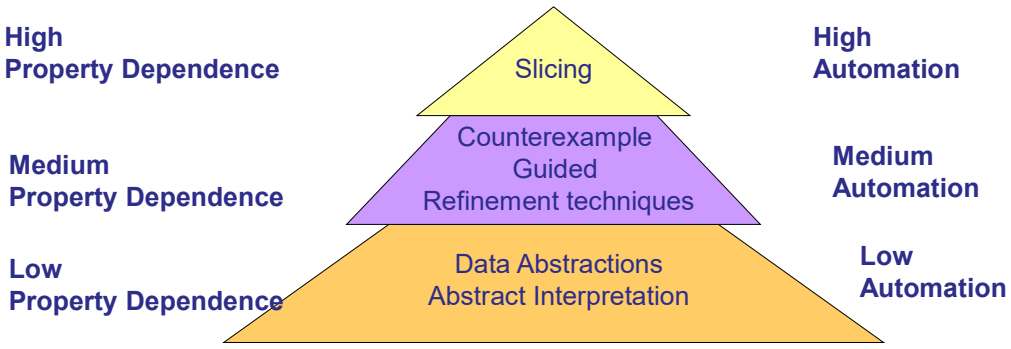
- **Model checking models need to be made smaller**
- **Problem: State-Space Explosion**
- **Smaller or "reduced" models must retain information**
  - Property being checked should yield same result

➢ **Balancing solution: Abstractions**

Types of Abstractions

- **Sound**
  - Property holds in abstraction implies property holds in the original program
- **Complete**
  - Algorithm always finds an abstract program if it exists
- **Exact**
  - Property holds in the abstraction iff property holds in the main program

Abstraction Landscape



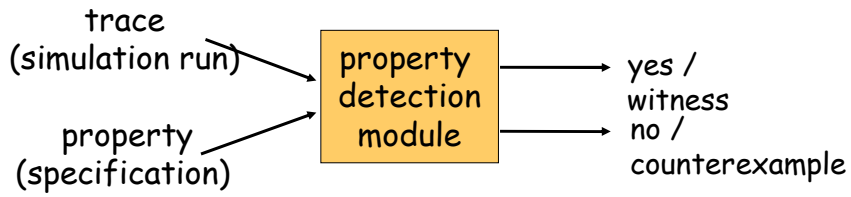
Semi-Formal Methods

- **Executable specification for behavioral modeling**
  - Design Productivity
    - Easy to model complex algorithm
    - Fast execution
    - Simple Testbench
  - Tools
    - Native C/C++ through PLI/FLI
    - Extended C/C++ : SpecC, SystemC

➢ **Verify it on the fly!**
  - Test vector generation
  - Compare RTL Code with Behavioral Model
  - Coverage Test

Assertion-Based Verification

- ***Property Detection*: To decide whether a simulation run (trace) of a design satisfies a given property (assertion)**



e.g., violation of mutual exclusion, $critical_1$ Æ $critical_2$

- **Temporal logic**
  - Example: Properties written in PSL/Sugar

Specifying Properties (Assertions)

- **Open Vera Assertions Language (Synopsys)**
- **Property Specification Language (PSL) (IBM, based on *Sugar*)**
  - Accelera driving consortium
  - IEEE Std. 1850-2005
- **Accelera Open Verification Library (OVL) provides ready to use assertion functions in the form of VHDL and Verilog HDL libraries**
- **SystemVerilog is a next generation language, added to the core Verilog HDL**
  - IEEE Std. 1800-2005

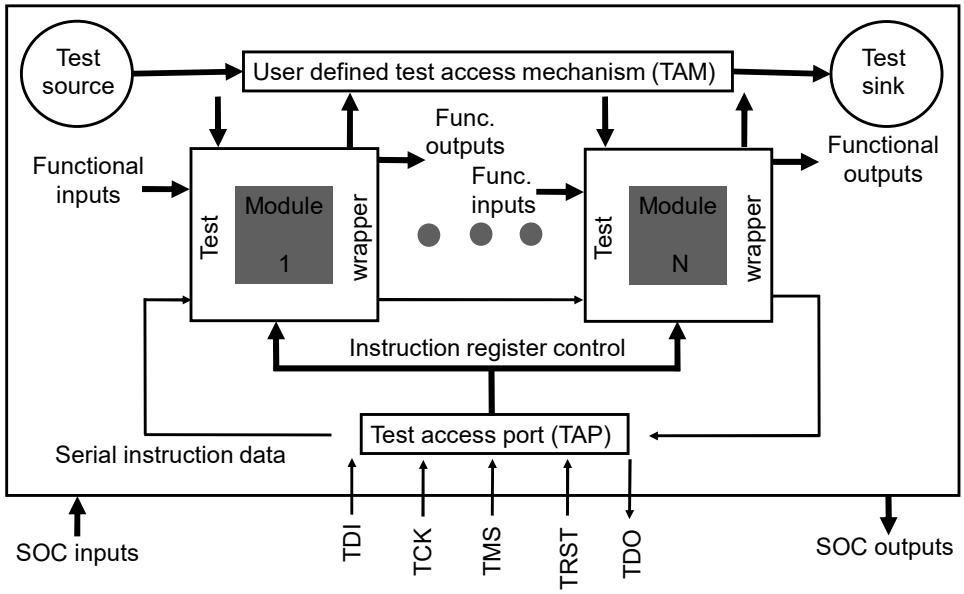
Outline

- ✓ Introduction
  - ✓ Verification vs. test
- ✓ Verification methods
  - ✓ Verification flow
  - ✓ Simulation-based techniques
  - ✓ Formal analysis
  - ✓ Semi-formal approaches
- **SoC manufacturing test**
  - Design for Test (DFT)
  - SoC testability features
  - Built-in self test

Partitioning for SoC Test

- **Partition according to test methodology:**
  - Logic blocks
  - Memory blocks
  - Analog blocks
- **Provide test access:**
  - Boundary scan
  - Analog test bus
- **Provide test-wrappers for cores**
- ➢ **Design for Test (DFT)**

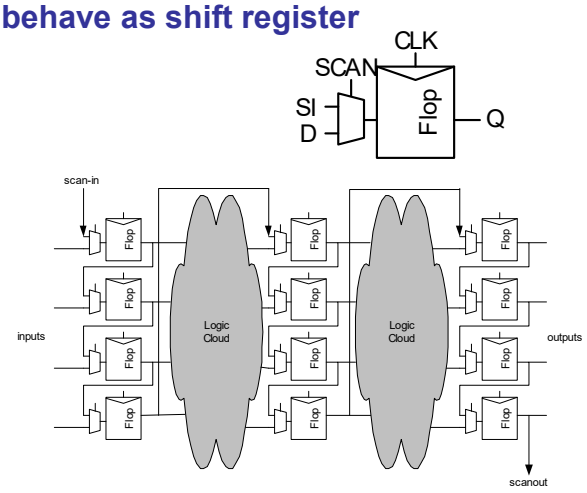
DFT Architecture for SOC



Source: Bushnell and Agrawal

Scan

- **Convert each flip-flop to a scan register**
  - Only costs one extra multiplexer
- **Normal mode: flip-flops behave as usual**
- **Scan mode: flip-flops behave as shift register**
  - Contents of flops can be scanned out and new values scanned in

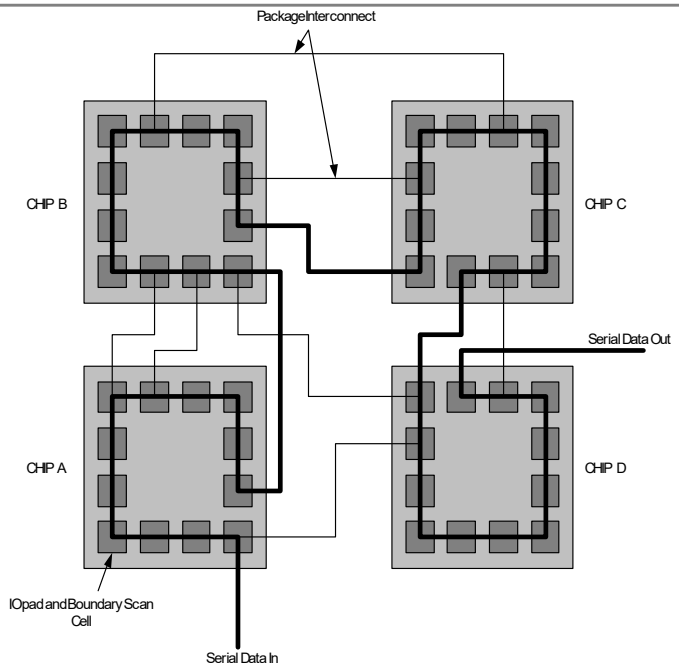


Boundary Scan

- **Testing boards is also difficult**
  - Need to verify solder joints are good
    - Drive a pin to 0, then to 1
    - Check that all connected pins get the values
- **Through-hold boards used "bed of nails"**
- **SMT and BGA boards cannot easily contact pins**

➢ **Build capability of observing and controlling pins into each chip to make board test easier**

Boundary Scan Example



Boundary Scan (IEEE 1149.1, JTAG)

- **Boundary scan is accessed through five pins**
  - TCK: test clock
  - TMS: test mode select
  - TDI: test data in
  - TDO: test data out
  - TRST*: test reset (optional)
- **Chips with internal scan chains can access the chains through boundary scan for unified test strategy.**

Additional DFT Components

- **Test source: Provides test vectors via on-chip LFSR, counter, ROM, or off-chip ATE.**
- **Test sink: Provides output verification using on-chip signature analyzer, or off-chip ATE.**
- **Test access mechanism (TAM): User-defined test data communication structure; carries test signals from source to module, and module to sink; tests module interconnects via test-wrappers; TAM may contain bus, boundary-scan and analog test bus components.**
- **Test controller: Boundary-scan test access port (TAP); receives control signals from outside; serially loads test instructions in test-wrappers.**

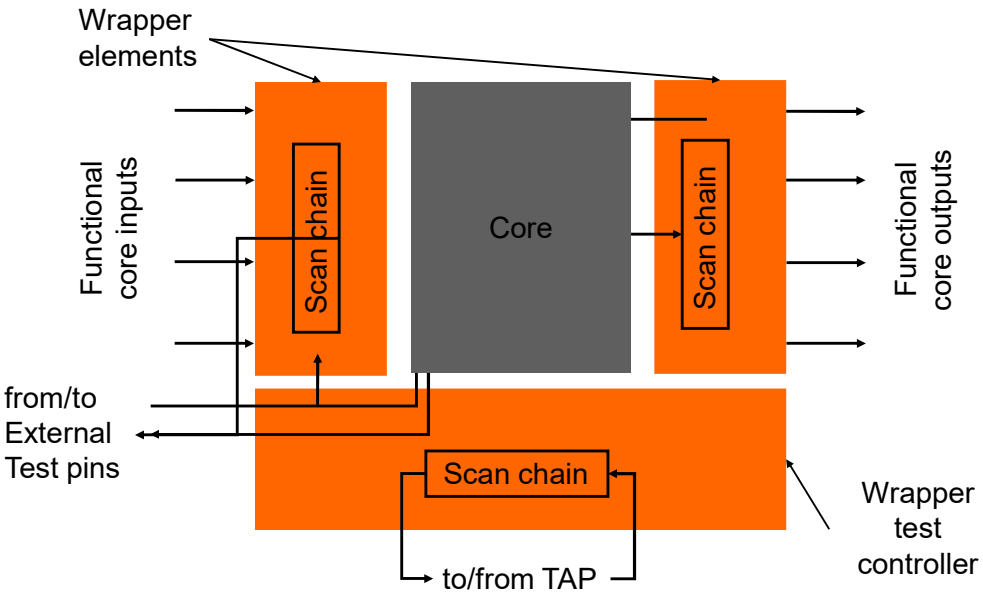
Source: H. Kerkhoff

Test Wrapper for a Core

- **Logic added around a core *to provide test access* to the embedded core**
- **Test-wrapper provides for each core input terminal**
  - An external test mode – Wrapper element observes core input terminal for *interconnect* test
  - An internal test mode – Wrapper element controls state of core input terminal for testing the logic *inside core*
- **For each core output terminal**
  - A normal mode – Host chip driven by core terminal
  - An external test mode – Host chip is driven by wrapper element for *interconnect* test
  - An internal test mode – Wrapper element *observes core outputs* for core test

Source: H. Kerkhoff

A Test-Wrapper



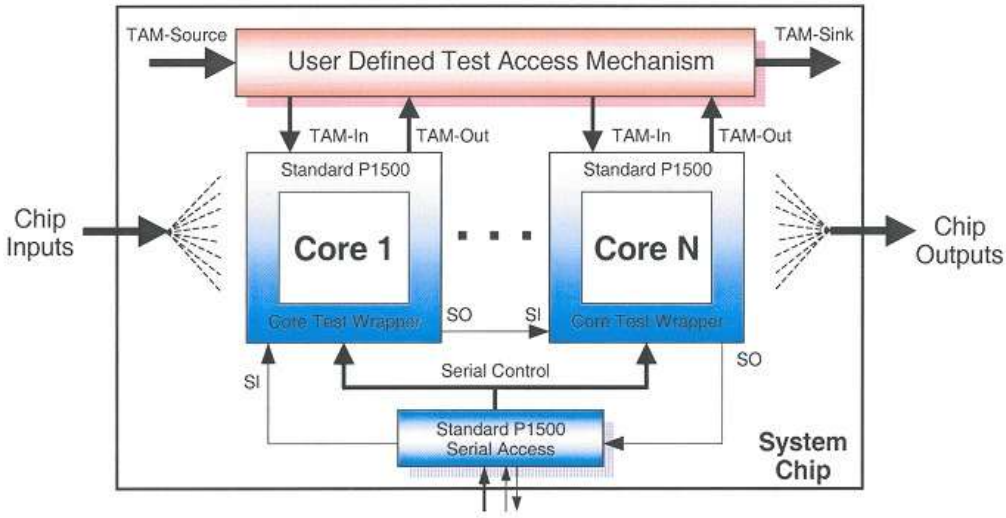
Source: H. Kerkhoff

Goals of IEEE P1500

- Core test *interface* between embedded core and system chip
- Test *reuse* for embedded cores
- Testability guarantee for system interconnect and logic
- Improve efficiency of test between core users and core providers

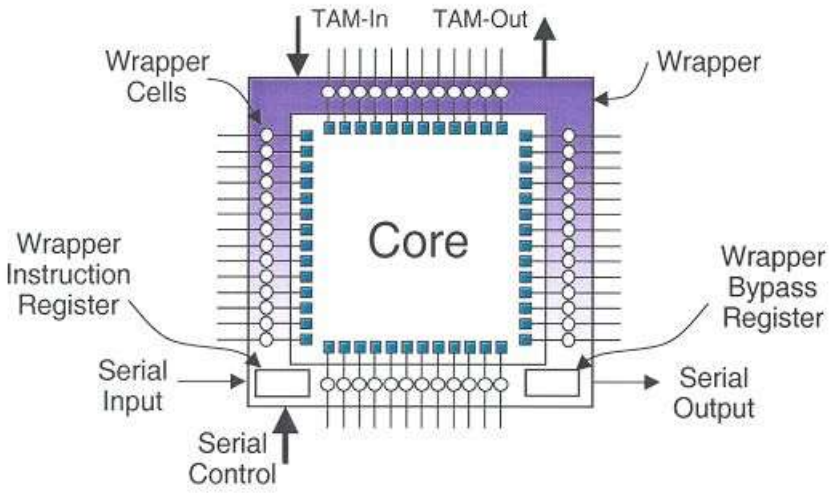
Source: H. Kerkhoff

Set-up of P1500 Architecture



Source: H. Kerkhoff

Core including Wrapper Cells



Source: H. Kerkhoff

Built-In Self Test (BIST)

- **Increasing circuit complexity, tester cost**
  - Interest in techniques which integrate some tester capabilities on the chip
  - Reduce tester costs
- **Approach:**
  - **Pseudo-random (or pseudo-exhaustive) pattern generator (PRPG)** on the chip
  - Compress test responses into "**signature**" using **multi-input shift register (MISR)**

➢ **Integrating pattern generation and response evaluation on chip – BIST**

Test Access Mechanisms (TAMs)

- **Non-functional access**
  - Uses a kind of access to core not allowed during the normal functional operation
  - Generally based on scan chains or other design for test (DFT) structures → Slow serial access
  - Direct access to core test pins through external pins → Fast, but high pin overhead
  - Can also use the embedded processor as the test source/sink → Needs wrappers around the core under test
- **Functional access**
  - Embedded processor is the test source/sink → No DFT structures or wrappers around the cores

Functional TAMs

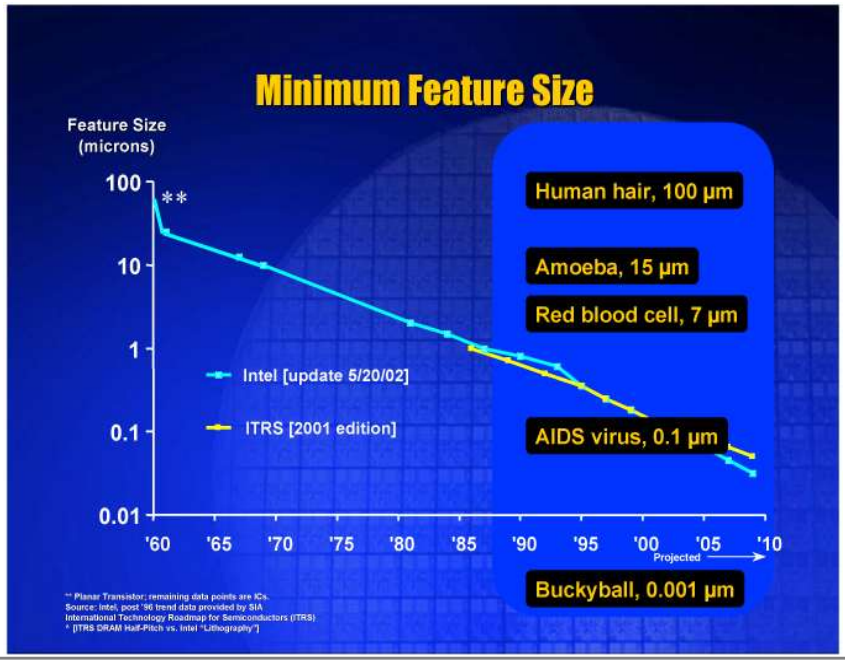
- **Software-Based Self Test (SBST): Use the intelligence of the embedded processor to test the SOC**
- **At-speed tests are possible**
- **Cores in the SOC can be of three kinds**
  1. White box -- internals visible, structure changeable
  2. Grey box – all the internals visible, but structure of the core cannot be changed
  3. Black box – no internals visible, no change can be made on the core
- **Any methodology for testing black box cores should not depend on knowledge of the core's internals**

Why is Conventional Test Successful?

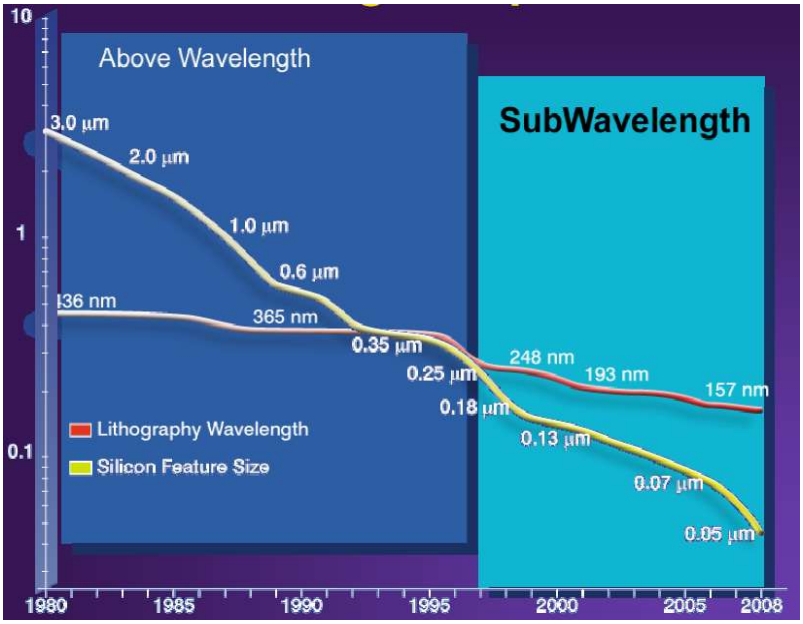
- **Two innovations have allowed test to keep up with complex designs**
  - The stuck-at fault model
    - The model allows structural test generation, with a number of faults which is linear in the size of the circuit
  - Partitioning the circuit
    - Partitioning the circuit (with scan latches for example), alleviates the test problem so that test generation does not have to deal with the entire circuit

➢ **Do these two assumptions hold for Deep SubMicron (DSM) circuits?**

IC Technology

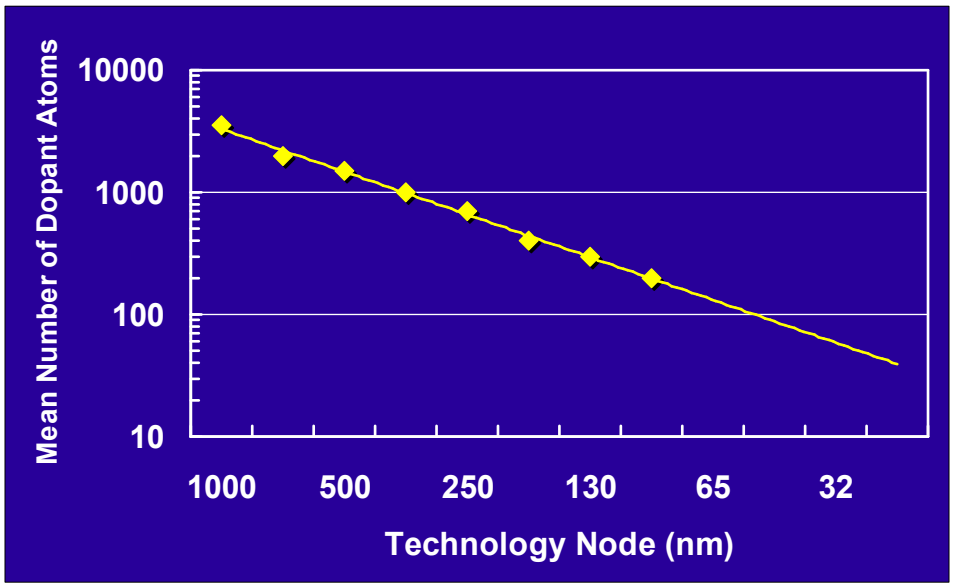


Features Smaller than Wavelengths



Source: Raul Camposano, Synopsys

Random Dopant Fluctuations

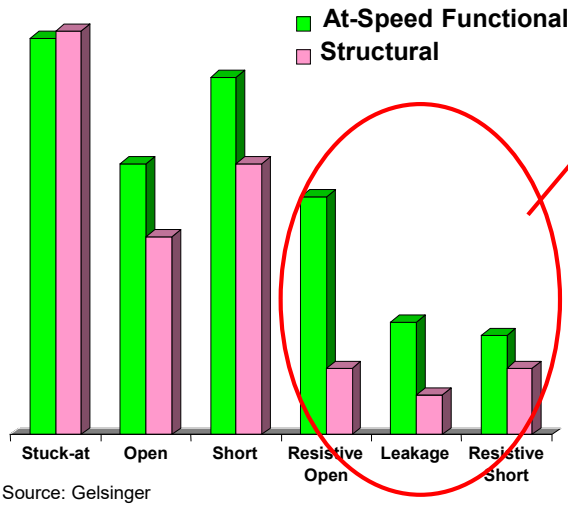


Defects in DSM Technologies

- **Experiments on real chips (e.g., Stanford)**
  - Stuck-at tests do not detect some defects unless they are applied at speed
- **Resistive opens comprise the bulk of test escapes in one production line**
  - Likely in copper interconnect – cause delay faults
- **Delay faults identified as the cause of most test escapes on another line**
  - Speed differences of up to a factor of 1.5 can exist between fast and slow devices - problems with "speed binning"
- **Increasing possibility of *shorts* and *crosstalk***

Effects on Chip?

- **Change in delays of paths**
- **Effects could be distributed across paths**



Source: Gelsinger

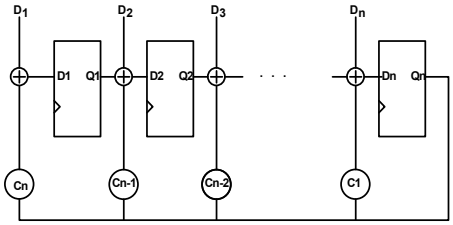
At-speed functional tests better for delay defects

Solution:

- **At-Speed tests**
  - Tester Cost?
- **Apply "Native Mode"?**
  - Can use low-cost testers

Native-Mode Built-In Self Test

- **Functional capabilities of processors can be used to replace BIST hardware – [UT Austin, ITC'1998]**
  - Application to self-test of processors at Intel – FRITS method applied to Pentium 4, Itanium [ITC'2002]



Hardware for MISR

```
for each data value Di {
   Shift_Right_Through_Carry(S);
   if (Carry) S = XOR(S, polynomial);
   S = XOR(S, Di);
}
```

Software implementation of MISR

Native-Mode Self Test for Processors

- **Random instructions can be run from cache and results compressed into a signature**
- **Implementation in Intel FRITS system showed benefits for real chips (Pentium 4, Itanium)**
- **Technique can be used for self-test of an embedded processor in a System-on-Chip**

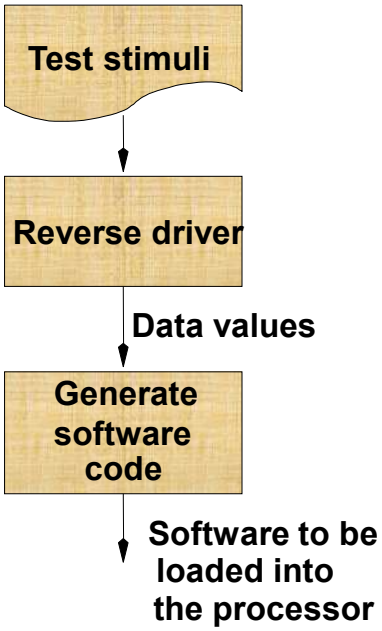
➢ **Is it possible to now use this processing capability to test other modules (digital, analog/mixed-signal and RF) on the SoC?**
  - First, can the processor test be improved to detect realistic defects, e.g., small delays?

Are Random Tests Sufficient?

- **Intel implementation involved code in the cache which generated random instruction sequences**
- **Interest in generating instructions targeting faults**
  - Possible to generate instruction sequences which will test for an internal stuck-at fault in a module [Gurumurthy, Vasudevan and Abraham, ITC 2006]
- **In order to deal with defects in DSM technologies, need to target small delay defects**
  - Recent work: automatically generate instruction sequences which will target small delay defects in an internal module [Gurumurthy, Vemu, Abraham and Saab, European Test Symposium (ETS) 2007]

Approach to Testing Cores

- **Uses functional TAM**
- **Uses pre-existing vectors**
- **Generates software to be loaded on to the embedded processor**
  - Reverse driver that produces given test vectors for core



[Gurumurthy, Sambamurthy and Abraham, Int'l Test Synthesis Workshop (ITSW) 2008]

Summary

- **Verification vs. test**
- **Verification methods**
  - Simulation-based
  - Formal equivalence and model checking
  - Semi-formal assertion-based verification
- **Manufacturing test**
  - Design for test (DFT)
  - Scan chains, JTAG boundary scan, test wrappers
  - Built-in self test