



Test Technology Overview

RASSP Education & Facilitation Program

Module 43

Version 3.00

Copyright 1995-1999 SCRA

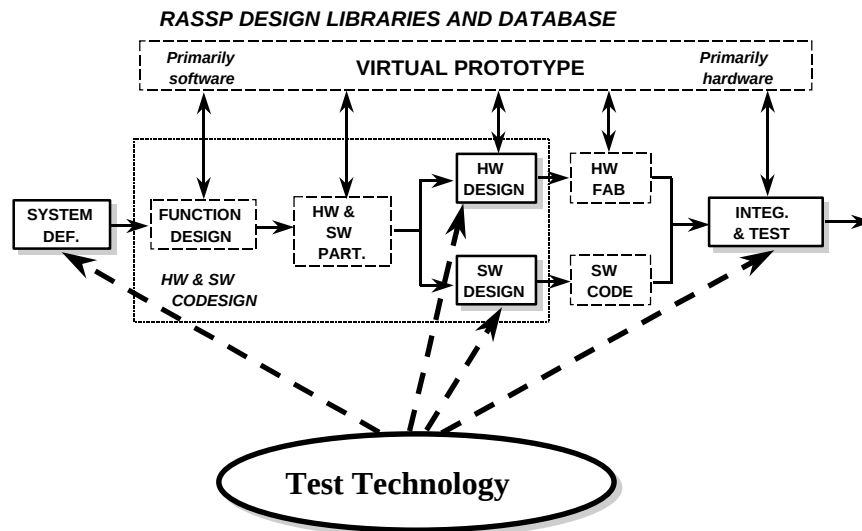
All rights reserved. This information is copyrighted by the SCRA, through its Advanced Technology Institute (ATI), and may only be used for non-commercial educational purposes. Any other use of this information without the express written permission of the ATI is prohibited. Certain parts of this work belong to other copyright holders and are used with their permission. All information contained, may be duplicated for non-commercial educational use only provided this copyright notice and the copyright acknowledgements herein are included. No warranty of any kind is provided or implied, nor is any liability accepted regardless of use.

The United States Government holds "Unlimited Rights" in all data contained herein under Contract F33615-94-C-1457. Such data may be liberally reproduced and disseminated by the Government, in whole or in part, without restriction except as follows: Certain parts of this work to other copyright holders and are used with their permission; This information contained herein may be duplicated only for non-commercial educational use. Any vehicle, in which part or all of this data is incorporated into, shall carry this notice.

Copyright © 1995-1999 SCRA

1

Rapid Prototyping Design Process





Module Goals



- | **To educate the general digital systems designer on the fundamentals of test technology in a manner that will assist him/her in designing testable systems**
- | **Provide information on the goals and techniques for testing systems**
 - m **Provide information on techniques to increase the testability of designs**
 - m **Provide information on how to incorporate these techniques into a general digital design methodology**

Copyright © 1995-1999 SCRA

3

This slide outlines the goals of this module in terms of information provided to the student.



Module Outline



- | **Introduction (Part 1)**
- | **Fault Modeling**
- | **Test Generation**
- | **Automatic Test Pattern Generation Algorithms**
- | **Fault Simulation Algorithms**
- | **Introduction (Part 2)**
- | **I_{DDQ} Testing**
- | **Design for Testability Techniques**
 - m **Ad hoc design for testability techniques**
 - m **Structured design for testability techniques**
 - q **Scan design**
 - q **Boundary scan**

Copyright © 1995-1999 SCRA

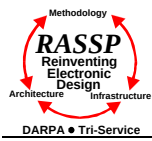
4



Module Outline (Cont.)



- | **Built-In Self Test**
 - m **Definitions**
 - m **Test generation techniques for BIST**
 - m **Signature analysis**
 - m **BIST case study**
 - m **Autonomous Built-In Self Test**
- | **Hierarchical Design for Test**
- | **Synthesis for Test**
- | **DFT Standards**
 - m **IEEE Std. 1149.1**
 - m **IEEE Std. 1149.1b**
 - m **IEEE Std. 1149.5**
 - m **IEEE Std. 1029.1**
 - m **MIL-HDBK-XX47**



Module Outline (Cont.)



- | **Design Flows with DFT**
- | **Summary**
- | **References**



Module Outline



- | **Introduction (Part 1)**
- | **Fault Modeling**
- | **Test Generation**
- | **Automatic Test Pattern Generation Algorithms**
- | **Fault Simulation Algorithms**
- | **Introduction (Part 2)**
- | **I_{DDQ} Testing**
- | **Design for Testability Techniques**
- | **Built-In Self Test**
- | **Hierarchical Design for Test**
- | **Synthesis for Test**
- | **DFT Standards**
- | **Design Flows with DFT**
- | **Summary**

Copyright © 1995-1999 SCRA

7



Introduction The Testing Problem

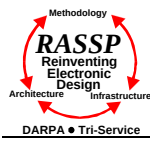


- | **This module presents techniques that are used to detect defects in digital ICs and PC board systems**
- | **The goal of testing is to apply a minimum set of input vectors to each device to determine if it contains a defect**
- | **The first step is to detect defects at the manufacturing level at the earliest point possible**

Copyright © 1995-1999 SCRA

8

This module presents the techniques that are typically used to detect manufacturing defects in ICs.



The Testing Problem

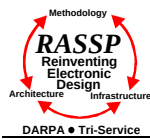


- | **Costs increase dramatically as faulty components find their way into higher levels of integration**
 - m **\$1.00 to fix an IC (throw it out)**
 - m **\$10.00 to find and replace bad IC on a PC board**
 - m **\$100.00 to find bad PC board in a system**
 - m **\$1000.00 to find bad component in fielded system**

Copyright © 1995-1999 SCRA

9

Testing for manufacturing defects incurs costs in the form of time during the design cycle (to generate the necessary test) and time during manufacturing (to actually apply the tests to the devices). However, the costs of letting a defective device be used in a system increase typically by an order of magnitude for each step in the process where the defect goes undetected.



The Testing Problem (Cont.)



- | **Apply a set of test vectors to each device off the manufacturing line and compare outputs to the known good response**
- | **The optimum test set will detect the greatest number of defects that can be present in a device with the least number of test vectors (high defect coverage)**
- | **Types of test sets:**
 - m **Exhaustive** - apply every possible input vector
 - m **Functional** - test every function of the device
 - m **Fault Model Derived** - find a test for every “modeled” fault

Copyright © 1995-1999 SCRA

10

This is the methodology used to test devices during manufacturing. Development of the optimal test set is the most difficult part of the process. The goal is to have the shortest test set possible for maximum fault coverage because a longer test set takes more time to apply to each device (\$\$\$).

There are a number of different approaches to test set generation.



Testing

Which Type is Closest to Optimum?



Consider a 74181 ALU Chip - 14 inputs

| Exhaustive testing

- m **Will detect 100% of detectable faults**
- m **Requires $2^{14} = 16,384$ test vectors**
- m **A 16 bit ALU with 38 inputs would take 7.64 hours to exhaustively test at 10 MHz**

| Functional testing

- m **Will detect 100% of detectable faults**
- m **Total functional testing will take over 448 vectors**
 - q **Each logical mode can be tested with about 8 vectors**
 - q **Each arithmetic mode can be tested with about 20 vectors**
- m **There is no algorithmic way to verify that all functional modes have been tested (designer expertise required)**

Copyright © 1995-1999 SCRA

11

Exhaustive testing consists of applying every possible input combination to a device under test. It is guaranteed to detect all detectable faults, but it is not practical in terms of number of tests required.

Functional testing will also detect every detectable fault if the tests are complete. That means that the test set exercises EVERY functional mode with EVERY possible data set. This method will usually result in a smaller test set size than exhaustive, but it is not practical in the sense that it takes too much time for the designer to write the vectors to ensure that they are complete.



Testing

Which Type is Closest to Optimum? (Cont.)

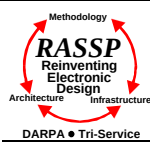


- | **Modeled fault testing (consider single-stuck-at faults)**
 - m **Will detect 100% of detectable modeled faults**
 - m **Requires only 47 vectors**
 - m **Vectors can be generated and analyzed (for fault coverage) using computer programs**
 - m **The number of defects actually detected by this test vector set depends on the quality of the fault model**
 - m **The key is to select a fault model that can be applied to the appropriate level of circuit abstraction (logic level) and that maps to the most possible physical defects**

Copyright © 1995-1999 SCRA

12

The most practical method for test set generation is to develop a model of the physical defects that can occur in the fabricated device at a higher level of abstraction, typically the logic level, and then develop tests for these modeled faults. It is then usually a tractable problem to develop tests of all of the detectable modeled faults. Depending on the quality of the fault model, a test set developed in this manner will most often cover a large percentage of the actual physical defects.



Testing Current Practice



- | **Use fault simulation on functional test set developed during design to determine list of undetected faults**
- | **Use this list of undetected faults as input to ATPG tool**
- | **If fault coverage is unsatisfactory, redesign or use DFT techniques to make undetected faults testable**

OR

- | **Use structured DFT/BIST techniques from the bottom up**

Copyright © 1995-1999 SCRA

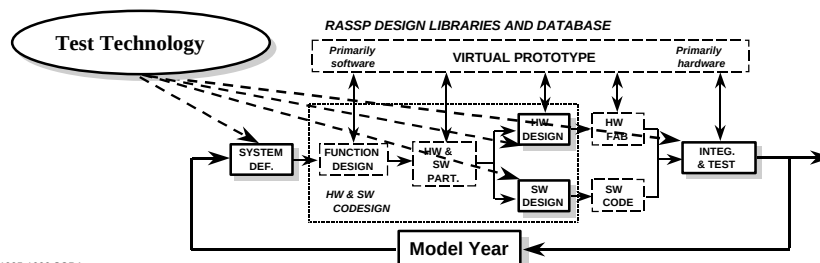
13

For most ASIC designs, the typical practice is to begin with the designer's functional verification test set. This set of vectors is then fault simulated to determine its fault coverage. If it is too low to be acceptable, more functional vectors can be added to exercise the portions of the circuit where the undetected faults lie. Another, perhaps more efficient, approach used is to feed the list of undetected fault to an Automatic Test Pattern Generation program to develop test for them.

Another approach used to to apply Design for Test or Built-In Self-test techniques as part of the design process. IBM's use of Level Sensitive Scan Design is a famous example.

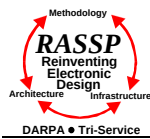
Testing and Rapid Prototyping

- | To be effective, test must be incorporated into the design process at the highest levels
- | To avoid having a major impact on the speed of the design process, techniques for incorporating test must be efficient



Copyright © 1995-1999 SCRA

14



Module Outline



- | Introduction (Part 1)
- | **Fault Modeling**
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Built-In Self Test
- | Hierarchical Design for Test
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | Summary

Copyright © 1995-1999 SCRA

15



Fault Attributes

- **Cause**
 - Specification Mistakes
 - Implementation Mistakes
 - External Component Defects
 - Disturbances
- **Interval**
 - Constant
 - Intermittent
 - Transient

- **Characteristics**
 - Hardware
 - Software
- **Value**
 - Certain
 - Unpredictable
- **Breadth**
 - Local
 - Global

Copyright © 1995-1999 SCRA

16

Presented here is the taxonomy of possible faults in a device.
Ref: [Johnson89]



Fault Modeling



I Goals

- m **Model defects in the device at the highest level of abstraction possible**
 - q **Reduces the number of individual defects that have to be considered**
 - q **Reduces the complexity of the device description that must be used in test generation and analysis**
 - q **Allows test generation and analysis to be done as early in the design process as possible**
- m **Model as high a percentage as possible of the actual physical defects that can occur in the device**

Copyright © 1995-1999 SCRA

17

The trade-off in fault modeling is to develop a model that is as simple as possible to use in test generation, but that models as high a percentage of physical defects as possible.

Fault Models

- | **Stuck-at faults**
 - m Single stuck-at faults
 - m Multiple stuck-at faults
- | **Stuck-open faults**
- | **Bridging faults**
- | **Delay faults**

Here is listed the types of fault models that will be presented. Several other types have been developed, but these are the most commonly used.

Single Stuck-at Fault Model

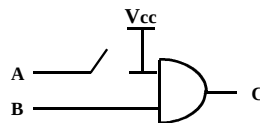
I Assumptions

- m Only one line in the circuit is faulty at a time
- m The fault is permanent (as opposed to transient)
- m The effect of the fault is as if the faulty node is tied to either Vcc (s-a-1), or Gnd (s-a-0)
- m The function of the gates in the circuit is unaffected by the fault

Fault-Free Gate

A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

Fault: A s-a-1



Faulty Gate

A	B	C
0	0	0
0	1	1
1	0	0
1	1	1

The single stuck-at fault model is the most often used fault model. Here are the assumptions or characteristics of the single stuck-at model.

Ref: [Hayes85],[Maly87]



Single Stuck-at Fault Model (Cont.)



I Advantages

- m Can be applied at the logic level or module level
- m Reasonable numbers of faults $2n$ (n =number of circuit nodes)
- m Algorithms for automatic test pattern generation (ATPG) and faults simulation are well developed and efficient
- m Research indicates that the single stuck-at fault model covers about 90% of the possible manufacturing defects in CMOS circuits
 - q Source-drain shorts, oxide pinholes, missing features, diffusion contaminants, metallization shorts, etc.
- m Other useful fault models (stuck-open, bridging faults) can be mapped into (sequences of) stuck-at faults

I Disadvantages

- m Does not cover all defects in CMOS or other devices

Copyright © 1995-1999 SCRA

20

The numerous advantages of the single stuck-at model are the reason it is most often used. It is simple to use during test generation and fault simulation; research has shown that it covers a large percentage of physical defects; and other fault models, that can increase defect coverage, can be mapped into sequences of single stuck-at faults.

There are however, some defects that cannot be covered by the single stuck-at model.

Multiple Stuck-at Fault Model

I Assumptions

- m Same as single stuck-at faults except:
- m 2 or more lines in the circuit can be faulty at the same time

I Advantage

- m If used in conjunction with single stuck-at faults, it covers a greater percentage of physical defects

I Disadvantages

- m Large number of faults $3^n - 1$ (n =number of circuit nodes)
- m Algorithms for ATPG and fault simulation are much more complex and not as well developed
- m Does not cover a significantly larger number of detects that single stuck-at faults

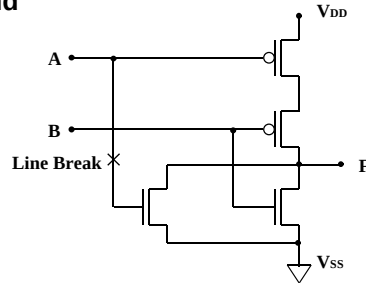
The multiple stuck-at model is sometimes used. It typically doesn't significantly increase the defect coverage enough to offset its major disadvantage of a large number of possible multiple stuck-at fault combinations.

Ref: [Hayes85]

Stuck-open Fault Model

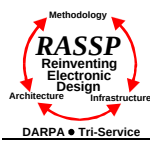
I Assumptions

- m A single physical line in the circuit is broken
- m The resulting unconnected node is not tied to either Vcc or Gnd



- Break above results in a “memory-effect” in the behavior of the circuit
- With AB=10, there is not path from either VDD or VSS to the output
 - F retains the previous value for some undetermined discharge time

The stuck-open fault is popular for use in CMOS circuits. When a defect occurs that breaks a line internal to a CMOS gate, it can result in a “memory effect.” For example, in this circuit, if the output is driven high with a 00 input, and then a 10 input is applied, then the output will stay high (its previous value). However, if the output is driven low with a 01 input, and then the 10 input is applied, then the output will stay low (its previous value) which appears OK.



Stuck-open Fault Model (Cont.)



I Advantages

- m Covers physical defects not covered by single or multiple stuck-at fault models
- m Can be tested with sequences of stuck-at fault tests
 - q In the previous example (Nor gate), apply AB=00 (test for F s-a-0) and force F to Vcc
 - q Apply AB=10 (test for A s-a-0) to force F to Gnd and observe results

I Disadvantages

- m Requires a larger number of tests (sequence for each fault)
- m Algorithms for ATPG and fault simulation are more complex and less well developed
- m Requires a lower level circuit description (transistor level), at least for development of the fault list

Copyright © 1995-1999 SCRA

23

Stuck-open faults can be tested for by sequences of stuck-at fault tests. This will typically result in higher defect coverage.

The major disadvantage of this technique is the increase in the number of tests that have to be applied, although actually finding them is no more complex than single stuck-at testing.

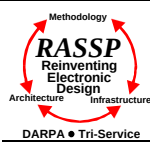
Bridging Fault Model

I Assumptions

- m **Two nodes of a circuit are shorted together**
- m **Usually assumed to be a low resistance path (hard short)**
- m **Three classes are typically considered:**
 - q **Bridging within a logic element (transistor gates, sources, or drains shorted together)**
 - q **Bridging between logic nodes (i.e. inputs or outputs of logic elements) without feedback**
 - q **Bridging between logic nodes with feedback**
- m **Typically not considered is bridging of non-logical nodes between logic elements (transistor shorts across logic elements)**

The bridging fault model is also sometimes used. It is more applicable as line widths get smaller. A hard short between lines is assumed. The model is that of logical lines in the circuit shorted together. The possibility of lines internal to a logic element shorted together is typically not considered because of the more complex circuit model required and the fact that these types of defects are most often covered by other models (stuck-at).

Ref: [Malaiya86]



Bridging Fault Model (Cont.)



I Advantages

- m Covers a large percentage of physical defects - some research indicates that bridging faults account for up to 30% of all defects

m Disadvantages

- m ATPG algorithms are more complex - testing requires setting the two bridged nodes to opposite values and observing the effect

- m Requires a lower level circuit description for bridging faults within logic elements

Copyright © 1995-1999 SCRA

25

Bridging fault models can be used to increase defect coverage. The major disadvantage is, again, the number of faults considered and the impact on test generation time.

Delay Fault Model

I Assumptions

- m The logic function of the circuit-under-test is error free
- m Some physical defect, such as process variations, etc., makes some delays in the circuit-under-test greater than some defined bounds
- m Two delay fault models are typically used:
 - q Gate delay, or transitional fault model
 - q Path delay fault model

Even though a circuit doesn't have any defects that cause incorrect function, it may contain defects that cause slow function. The delay fault models are attempts to model these types of defects and their effect on the circuit. The two models currently used are the gate delay or transitional fault model, where a single gate is assumed to take too long to produce an output, and the path delay fault model, where certain paths in the circuit may take too long to be exercised.

Transitional Delay Fault Model

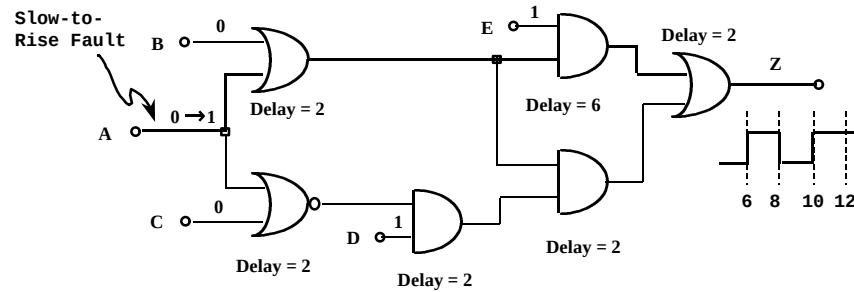
- | **A logical model for a defect that delays either a rising or falling transition on a specific line in the circuit**
 - m **Slow-to-rise**
 - m **Slow-to-fall**
- | **Advantage**
 - m **If a delay fault is large enough, it behaves as a temporary stuck-at fault, and single stuck-at fault testing techniques can be applied**
- | **Disadvantages**
 - m **Two patterns are required for detection *initialization* and *transition detection* (propagation)**
 - m **The minimum delay fault size that can be detected is difficult to determine**

The gate delay, or transitional delay fault model, was the first delay fault model to be developed and is also the simplest. The transitional fault model can be modeled as a temporary stuck-at fault, and tests for it can be developed using this model. Again, the major disadvantage is the complexity of the test generation process and the fact that determining the minimum delay size that can be detected is difficult.

Ref: [Waicukauski87]

Transitional Fault Model

Example of Minimum Delay Fault Detectable



Copyright © 1995-1999 SCRA

© IEEE 1987

[Waicukauski87]

28

This example shows what happens in a typical circuit when an input is changed. Several hazards are visible on the output while the internal circuit nodes settle to their final values. In this example, a delay fault at least as large as the time of measure (12 units) is detectable. It is more difficult to tell how small a delay fault can be before it is detectable because of the hazards.

Figure from [Waicukauski87]

Path Delay Fault Model

- | **A fault model in which the total delays in a path from inputs to outputs in a circuit exceeds some maximum value**
- | **Advantages**
 - m **Detects more delay faults - i.e., in transitional fault model, the delay of a faulty gate may be compensated for by other faster gates in the path**
 - m **Can be used with more aggressive statistical design philosophy**
- | **Disadvantages**
 - m **Large number of possible paths in circuit - exponential with number of gates**
 - m **Algorithms for test generation are more complex and less well developed**

This fault model considers paths in the circuit and tests to see if any path delays exceed some DP_{max} . This algorithm overcomes a potential problem with the transitional fault mode. That is, that the delay of a faulty gate can be compensated by gates in the propagation path that have faster-than-typical delays.

This delay fault model is also consistent with a statistical design philosophy. A statistical design philosophy recognizes that the delays of gates in a circuit are usually not all worst case, but that they fall within a small “typical” range. Using this knowledge, a greater clock speed can be specified by determining the typical delays for all paths in the circuit.

This delay testing method is a form of performance verification. What difference does it make if a single gate is out of tolerance if the paths delays are within DP_{max} ?

Ref: [Lesser80]



Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | **Test Generation**
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Built-In Self Test
- | Hierarchical Design for Test
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | Summary



Test Generation Definitions



Test Vector

An input vector for the circuit-under-test that causes the presence of a fault to be observable at a primary output

Automatic Test Pattern Generation

The process of generating a test pattern for a specific fault using some type of algorithm

Detected Fault

A fault for which a valid test vector has been generated

Undetected Fault

A fault for which a test vector has not been generated

Redundant Fault

A fault for which no test pattern exists (because of redundant logic in the circuit)

Copyright © 1995-1999 SCRA

31

No additional notes for the definitions are needed.



Test Generation Definitions (Cont.)



Fault Coverage

The percentage of total faults for which test patterns have been generated:

$$\text{Fault Coverage} = 100 \times \frac{\text{Number of Detected Faults}}{\text{Total Number of Faults in the CUT}}$$

Fault Efficiency

The percentage of faults that either are detected or **PROVEN** redundant (usually used to measure the effectiveness of a test generator):

$$\text{Fault Efficiency} = 100 \times \frac{\text{Number of Detected Faults} + \text{Number of Redundant Faults}}{\text{Total Number of Faults in the CUT}}$$

Copyright © 1995-1999 SCRA

32

No additional notes for the definitions are needed.



Test Generation Definitions (Cont.)



Controllability

A testability metric that measures the difficulty in driving a node to a specific value

Observability

A testability metric that measures the difficulty in propagating the value on node to a primary output

Testability Measure

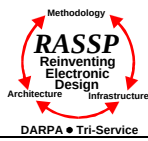
A metric that attempts to determine how difficult it will be to generate a test for a specific line in the circuit. This metric:

- Provides feedback to the designer on testability without actually performing test generation
 - Assists in the test generation process
 - Is based on controllability and observability

Copyright © 1995-1999 SCRA

33

No additional notes for the definitions are needed.



Test Generation Definitions (Cont.)



Sensitization

The process of driving the circuit to a state where the fault causes an actual erroneous value in the device at the point of the fault. E.g., for single stuck-at faults, driving the node to the value opposite the stuck-at value

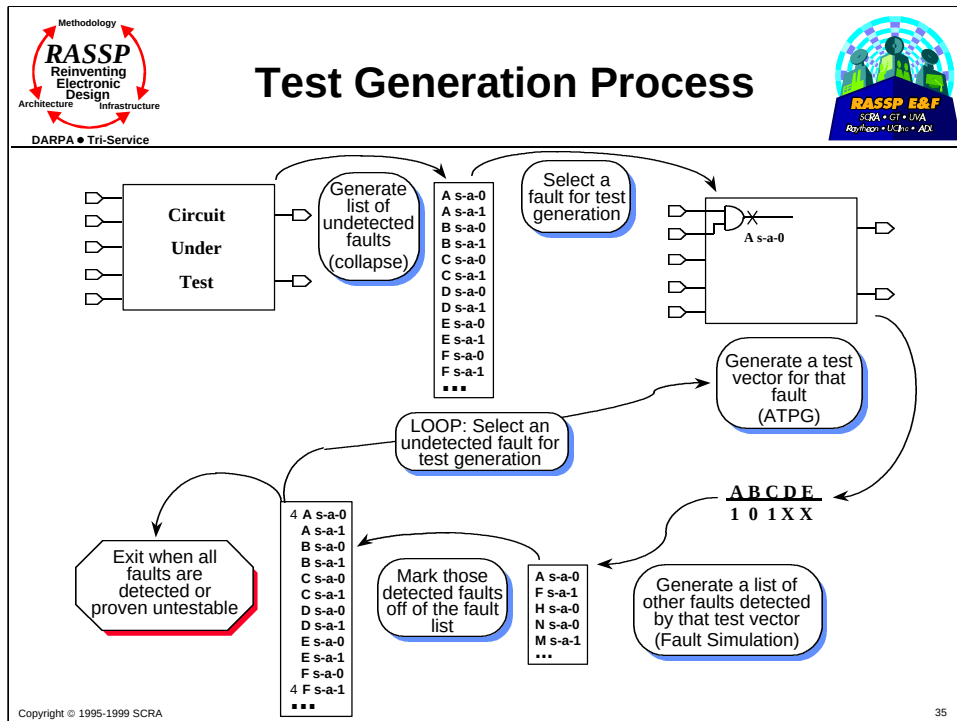
Propagation

The process of driving the circuit to a state where the error becomes observable at the primary outputs

Justification

The process of determining the input combination necessary to drive an internal circuit node to a specified value (consistency)

No additional notes for the definitions are needed.



This slide shows the overall flow of the test generation process. The process begins by generating a list of all possible faults in the circuit under test. One of these faults is selected for test generation. ATPG is performed to generate a test for this fault. Once the test is generated, fault simulation is performed to determine all of the faults that are detected by that vector. These detected faults are removed from the global fault list. Another undetected fault is selected from this list and the process begins again. The loop is exited when all of the faults are either detected or have an ATPG performed unsuccessfully on them (aborted).



Test Generation/Testing Disconnect



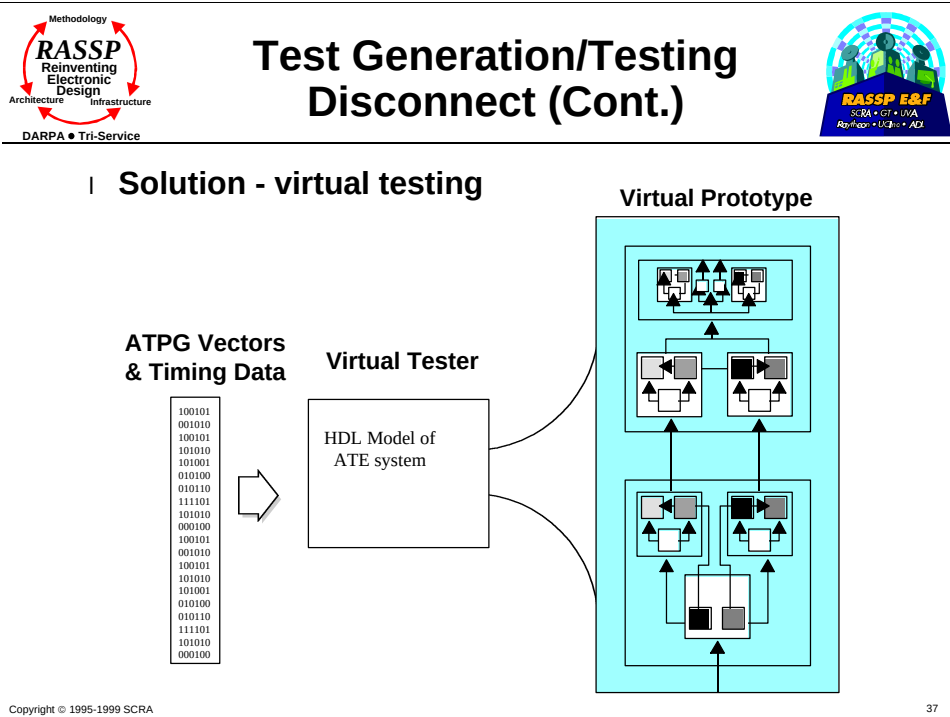
- | **Test vectors developed by the “design team” often must be significantly modified by the “test team”**
 - m **Vectors are incompatible with the Automatic Test Equipment (ATE)**
 - q Test vectors overflow scan-data, format-data, or timing-data memory
 - q Test vector set is not compact enough to fit in pattern memory
 - q Test vectors don’t address bi-directional conflicts
 - q Test vectors included comparisons with tristate values

Copyright © 1995-1999 SCRA

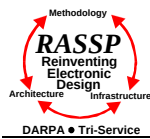
36

Finally, in the area of test generation, there is typically a disconnect between the design team and the test team. That is, the design team may generate what they think is a reasonable manufacturing test set using the techniques discussed, but the test team may have to completely rewrite it due to various tester limitations.

Ref: [Hemmady94]



One possible solution to this problem is the concept of “virtual testing” which is built on top of a virtual prototype of the system-under-test. What is needed is an HDL description of the tester through which the intended test vectors can be run and applied to the virtual prototype. Using this method, a great number of the tester/test set incompatibilities can be found.



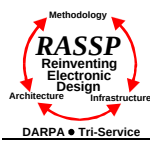
Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | **Automatic Test Pattern Generation Algorithms**
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Hierarchical Design for Test
- | Built-In Self Test
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | Summary

Copyright © 1995-1999 SCRA

38



Automatic Test Pattern Generation (ATPG) Algorithms

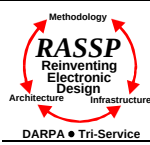


- | **The objective is to automatically generate a test for faults in the circuit-under-test**
- | **Major classes of methods:**
 - m **Pseudorandom**
 - m **Ad-Hoc**
 - m **Algorithmic**
 - q **D-algorithm**
 - q **PODEM**
 - q **FAN and related algorithms**
 - q **Others ...**

Copyright © 1995-1999 SCRA

39

Automatic test pattern generation is used to generate test vectors during the test generation process. There are two major ATPG methods: pseudorandom and deterministic. Algorithmic ATPG is used to generate tests for specific faults in the circuit-under-test.



Pseudorandom Test Generation



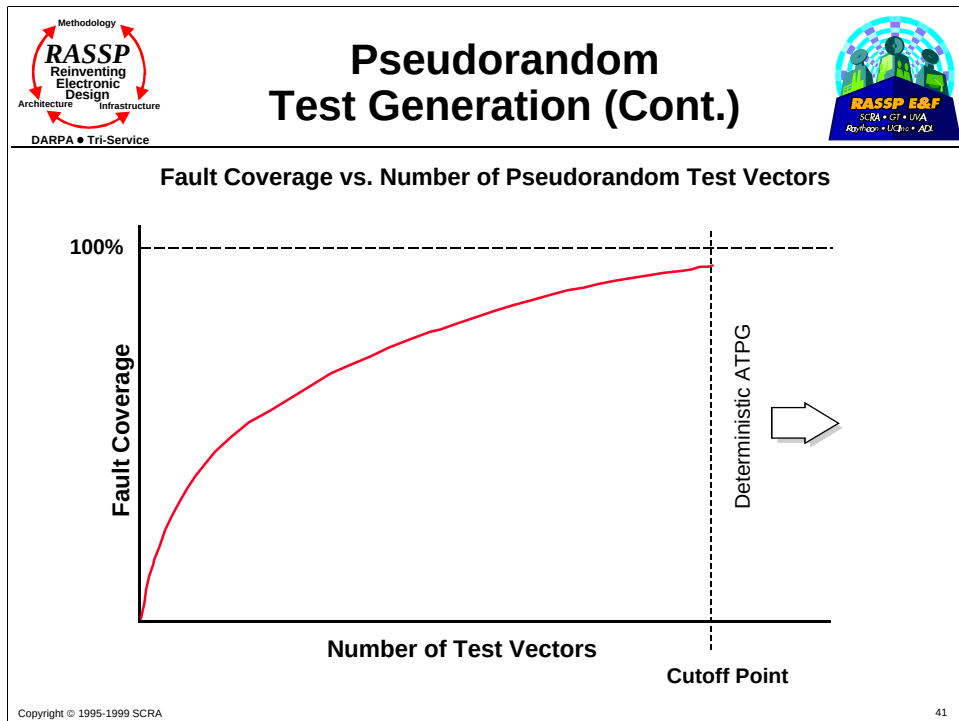
- | **Simply generate an input vector using a pseudorandom number generator and perform fault simulation to determine if it detects the target fault**
- | **The characteristics of the fault greatly influence how well pseudorandom test generation will work**
 - m **Easy-to-detect faults**
 - m **Hard-to-detect faults**
- | **Typically used in the beginning of the test generation process to remove easy-to-detect faults from the fault list**

Copyright © 1995-1999 SCRA

40

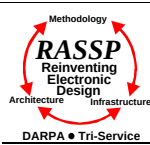
Pseudorandom test generation is the simplest method for generating tests and is typically used initially in the test generation process to quickly remove easy-to-detect faults from the fault list.

The method uses a “pseudorandom pattern generator”, so called because any pattern in the possible set is equally likely; but the pattern set is deterministic in that it can be repeated.



This graph shows the typical fault coverage vs. number of pseudorandom test vectors curve. Pseudorandom test generation is very efficient up to the point where the curve begins to flatten out (circuit dependent), and the deterministic ATPG is typically used to target the remaining undetected faults.

The way this is typically implemented is that pseudorandom tests are generated and fault simulated until two or more successive pseudorandom vectors fail to detect any new faults. Then the pseudorandom process is halted and deterministic ATPG is started.



Ad-Hoc



- | **Uses functional test vectors developed by designers for functional verification and design debugging by:**
 - m **Fault simulating to determine fault coverage**
 - m **Determining locations of undetected faults**
 - m **Adding additional functional tests to exercise areas of design with undetected faults**
 - m **Re-fault simulating and repeating until desired fault coverage is achieved**
- | **No special test generation system is required, only fault simulator**
- | **Utilizes existing vectors and designer expertise**
- | **Achieving high fault coverage may be difficult and time consuming - especially for synthesized designs**

Copyright © 1995-1999 SCRA

42

The Ad-Hoc test generation technique (as previously presented) uses the functional verification vectors as the initial manufacturing test vectors. They are fault simulated to determine fault coverage and undetected faults. If ATPG is not used, then the designer must add functional vectors to the test set to try and achieve higher fault coverage.

This may be especially difficult for synthesized designs because the designer doesn't have the circuit area to behavior correspondence; i.e., he/she may not know what portion of the functionality corresponds to the undetected fault area.

D Algorithm

- | **First algorithm proved “complete” - developed by Roth at IBM in 1966**
 - m **Complete** - can be proven that the algorithm will generate a test for a fault if it exists
- | **Introduced D notation**
 - m **D** - “1” in the good circuit “0” in the faulty
 - m **D’** - “0” in the good circuit “1” in the faulty (Dbar)
- | **PDCF - Primitive D cube of failure** - a set of inputs to a module that will sensitize a specific fault within the module
- | **PDC - Propagation D cube** - a set of inputs to a module that will propagate a D from the inputs to the outputs

The D algorithm was the first algorithm for test generation designed to be programmable on a computer. The D algorithm uses the single stuck-at fault model. Previously developed algorithmic techniques (boolean difference, literal proposition) were too expensive in terms of memory requirements for practical implementation on a computer.

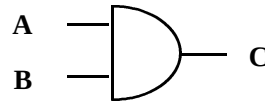
The D algorithm introduced the “D notation” which has been used in most subsequent ATPG algorithms.

Ref [Klenke92]

D Algorithm (Cont.)

D Notation

Good Circuit Value	Faulty Circuit Value	D Value
1	0	D
0	1	D'



AND Gate PDCFs

Fault	PDCF		
	A	B	C
A s-a-0	1	1	D
B s-a-0	1	1	D
C s-a-0	1	1	D
A s-a-1	0	1	D'
B s-a-1	1	0	D'
C s-a-1	X	0	D'
C s-a-1	0	X	D'

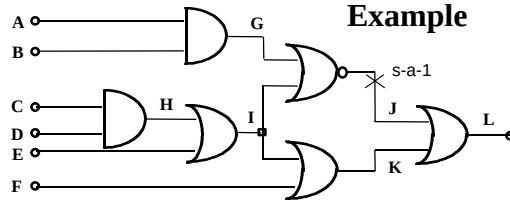
AND Gate PDCs

A	B	C
1	D	D
D	1	D
1	D'	D'
D'	1	D'
D	D	D
D'	D'	D'

This slide introduces the D notation and shows the Primitive D cubes of Failure (PDCFs) and Propagation D cubes (PDCs) for a simple AND gate.

The PDCFs are tests for all of the possible stuck-at faults in the AND gate, and the PDCs are all possible ways to propagate a D or Dbar from the inputs to the output.

D Algorithm (Cont.)



	A	B	C	D	E	F	G	H	I	J	K	L	
TC0											D'		← Initial test cube
TC1								1		X	D'		← Pick PDCF
TC2	1	1						1		X	D'		← Imply from G to A and B
TC3	1	1				0	0	1		0	D'	0	← Set objective K=0 and imply I and F
TC4	1	1			0	0	1	0	0	D'	0	D'	← Imply from I to E and H
TC5	1	1	0	X	0	0	1	0	0	D'	0	D'	← Justify H

	A	B	C	D	E	F	G	H	I	J	K	L	
TC0											D'		← Initial test cube
TC1								X		1	D'		← Pick PDCF
TC2								X		1	D'	1 1	← Imply K and L from I ; no test

Copyright © 1995-1999 SCRA

© IEEE 1992

[Klenke92] 45

This slide is an example of the application of the D algorithm on a target fault, namely, J s-a-1. The process begins by picking all of the possible PDCFs for J s-a-1. This choice, like all in algorithmic ATPG, can be arbitrarily made or made with the assistance of some heuristic (testability measures). After the PDCF is selected and applied, other circuit values that are implied by the values specified in the PDCF are noted. For example, the value of "1" on G requires a "1" on both A and B. After implication of the PDCF, the process of propagating the fault effect to a primary output using the PDCs is begun. The PDC of an OR gate is used to propagate the value on J to the output L. Implication of the value from the PDC on K is then done. This results in the requirement of a "0" on H. This value on H must be justified by setting either C or D to a "0", which results in a complete test.

The second example shows what would happen if the alternate PDCF for J s-a-1 is selected. No test would be possible with this selection, and another solution (namely the first) would have to be tried.

The D algorithm is a "branch-and-bound" algorithm in that at certain points in the algorithm choices as to the solution to be attempted must be made. To make the algorithm complete, each and every choice must be tried. This example illustrates part of the problem with the D algorithm in that choices may be possible at each internal circuit node and the number of solutions to be searched is exponential with the number of circuit nodes.

Ref [Klenke92]



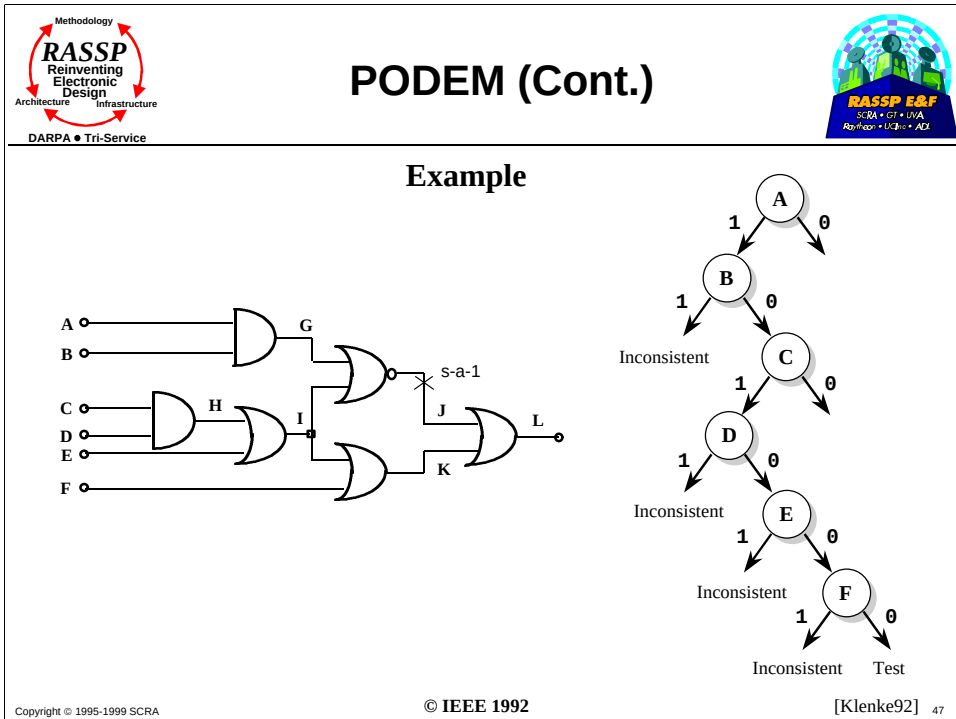
PODEM



- | **Path Oriented *DE*cision Making** - developed in 1981 by Goel to address the problem D algorithm had with XOR gates
 - m D algorithm is exponentially complex to the number of internal circuit nodes - XOR gates make the complexity of the D algorithm approach this limit
 - m PODEM expresses the search space in terms of assignments to the primary inputs only
- | **PODEM is also a branch-and-bound algorithm** which is exponentially complex to the number for circuit inputs - usually a much smaller number than circuit nodes

Copyright © 1995-1999 SCRA
46

PODEM was the first major efficiency enhancement to the D algorithm. PODEM formed the basis for most of the follow-on work in ATPG algorithms. PODEM is still exponentially complex, but its complexity is exponential to the number of circuit *inputs*, not the number of circuit nodes. More importantly, PODEM is more efficient in how it searches this solution space.



This example illustrates the basics of how PODEM orders the search space by applying values at the primary inputs. After an input assignment is selected, a simulation like process is performed to determine what values on circuit nodes are implied by the new input value. If, after implication, a test is no longer possible (fault set a s-a value, no propagation path, etc.) the opposite value for that input is tried. If that also precludes a test, the last input value tried is “popped off of the stack” and the alternate value is tried. This process of “backtracking” insures that PODEM is complete.

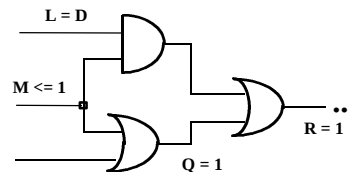
This example uses the simple heuristic of input/value selection of taking the inputs in order and always trying the “1” value first. PODEM in fact has some fairly complex heuristics and procedures that use circuit topology and current state information during the test generation process to select the next input and value to be tried.

More efficient heuristics for this process and earlier determination of the inconsistency of an input combination are focus of much of the follow-on work to PODEM.

Ref [Klenke92]

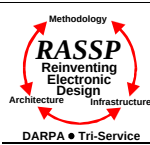
FAN and Related Algorithms

- | **FAN is an improvement on PODEM designed to utilize circuit topology information to increase search efficiency**
- | **Both the D algorithm and PODEM have trouble with areas of reconvergent fanout**
- | **Reconvergent fanout can cause complex interactions between internal circuit nodes**
- | **Example:**



The requirement for $M \leq 1$ to propagate the D value through the AND gate causes R to be set to 1 which terminates the propagation path

FAN was the next major improvement to the PODEM algorithm. FAN attempts to increase the efficiency of PODEM by adding some special techniques to handle the major circuit topology characteristic that causes a problem during test generation - reconvergent fanout.



Others...



- | **Many powerful heuristics have been developed that utilize knowledge of the circuit topology to prune the search space**
 - m Use of testability measures
 - m Learned implications
 - m “Saving” portions of the search space that lead to a successful test
 - m Switching search techniques when excessive backtracking results (propagation first vs. sensitization first, etc.)
- | **These techniques have lead to the development of very fast commercial ATPG systems for combinational circuits**

Copyright © 1995-1999 SCRA

49

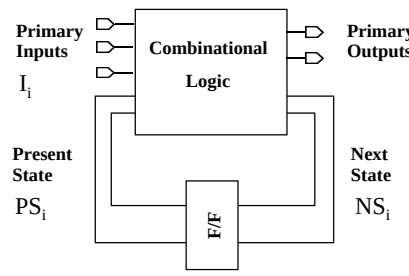
As stated previously, most of the follow-on work the PODEM-FAN has been in the area of improved heuristics and early detection of conflicts during test generation. The bottom line is that very fast test generators for combinational circuits are now available as part of commercial CAD packages (Mentor, Cadence, etc.)



Sequential Circuit Test Generation



Huffman Model



- m Can be treated as a form of combinational logic testing
- m Abstract behavior of the machine in terms of its state transition graph (STG) is also used
- m Problem occurs when the test vector depends on present state lines or the fault effect is propagated to the next state lines

Copyright © 1995-1999 SCRA
50

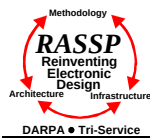
The next problem, currently the focus of much research, sequential circuit test generation. A sequential circuit is basically a block of combinational logic with some of its outputs fed back to the inputs via clocked flip-flops.

The problem with simply using combinational ATPG techniques on the combinational logic block is that a test may require a specific input combination on the Present State lines and the effect of the fault may be only propagated to the next state lines. In both cases, state machine analysis must be performed to determine how to drive the machine to the required Present state and how to differentiate the resulting faulty state from the normal good state.

Sequential Circuit Test Generation (Cont.)

- | **The problem is usually cast into 4 major steps:**
 - m **Step 1 - Combinational test generation - generate test for fault in combinational logic in terms of primary inputs and present state lines (I_i , PS_i)**
 - m **Step 2 - Derive input sequence necessary to drive machine from initial state to state PS_i using STG**
 - m **Step 3 - If effect of fault in step 1 was propagated to next state lines, a faulty state NS_F was created - derive input sequence necessary to differentiate between NS_F and NS_i using STG**
 - m **Construct test sequence by concatenating sequence from step 2, input vector from step 1, and sequence from step 3, and fault simulate**
 - m **Because steps 2 and 3 are typically done using STG of good machine, this is necessary to determine if resulting sequence is a test for the target fault**

The sequential ATPG process is usually modeled as a combination of 4 processes. The first is combinational ATPG for the target fault on the combinational logic block. The second, state justification, is the process of finding a sequence of inputs that will drive the state machine from the reset (or unknown) state to the Present state required by the test above. The third, state differentiation, is the process of finding an input sequence which will cause a different output sequence for the machine in the good state and faulty state as a result of the test found in the first step. And lastly, because the state justification and differentiation processes are typically done using the information about the fault-free machine, the fourth process, sequential fault simulation, is required to determine if the resulting input sequence is in fact a test for the target fault as well as other faults.



Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | **Fault Simulation Algorithms**
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Built-In Self Test
- | Hierarchical Design for Test
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | Summary



Fault Simulation Definitions



Good circuit

A logic model of the circuit-under-test without any faults inserted

Faulty circuit

A logic model of the circuit-under-test with one or more fault models inserted

Fault specification

Defining the set of modeled faults and performing fault collapsing

Fault insertion

Selecting a subset of faults to be simulated and creating the data structures to indicate the presence of the faults

Copyright © 1995-1999 SCRA

53

No additional notes on the definitions are required.

Ref: [Abramovici90]

Fault Simulation Definitions (Cont.)

Equivalent fault

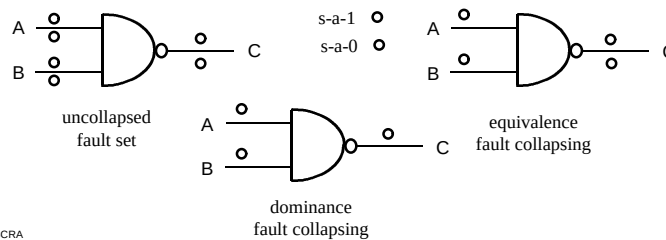
Two faults f_i and f_j are equivalent if there is no test that will distinguish between them

Dominant fault

A fault f_i dominates a fault f_j if every test that detects f_i also f_j detects

Fault collapsing

The process of reducing the fault set by removing equivalent (and dominated) faults



Copyright © 1995-1999 SCRA

54

No additional notes on the definitions are required.

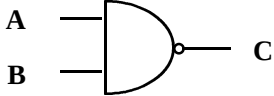
In the figure, A s-a-0 and B s-a-0 are equivalent to C s-a-1 because only the test AB=11 will detect them. Therefore, A s-a-0 and B s-a-0 can be removed from the fault list. Either of the equivalent faults can be removed.

Finally, C s-a-0 can be detected by AB=00, or 10, or 01, and A s-a-1 can only be detected by AB=01 and B s-a-1 can only be detected by AB=10. Therefore, A s-a-1 and B s-a-1 dominate C s-a-0 and C s-a-0 can be removed from the fault list. Dominated faults can be removed from the fault list.



Fault Tables





Fault Table

	f_1	f_2	f_3	f_4	f_5	f_6
$T_1 = 00$			1			
$T_2 = 01$			1	1		
$T_3 = 10$			1		1	
$T_4 = 11$	1	1				1

Good/Faulty Circuit Response

	$f_0 = \text{no faults}$	$f_1 = A s-a-0$	$f_2 = B s-a-0$	$f_3 = C s-a-0$	$f_4 = A s-a-1$	$f_5 = B s-a-1$	$f_6 = C s-a-1$
$T_1 = 00$	1	1	1	0	1	1	1
$T_2 = 01$	1	1	1	0	0	1	1
$T_3 = 10$	1	1	1	0	1	0	1
$T_4 = 11$	0	1	1	0	0	0	1

To construct fault table, XOR f_1 - f_6 columns with f_0 column

Copyright © 1995-1999 SCRA
55

This slide illustrates the construction of a fault table for a simple NAND gate.

Notice that the equivalent and dominance relationship of the faults can be seen from the table. For example, f_1 , f_2 , and f_6 are equivalent. Faults f_4 and f_5 dominate fault f_3 .

Fault Table Reduction

Fault Table

	f ₁	f ₂	f ₃	f ₄	f ₅	f ₆
T ₁	1	1	1			
T ₂			1	1		
T ₃			1		1	
T ₄	1	1				1

To collapse faults, remove all but one equivalent columns and all *dominating* columns

Collapsed Fault Table

	f ₄	f ₅	f ₆
T ₁			
T ₂	1		
T ₃		1	
T ₄			1

Collapsed Fault Table

	f ₄	f ₅	f ₆
T ₁			
T ₂	1		
T ₃		1	
T ₄			1

To collapse tests, remove all but one equivalent rows and all *dominated* rows

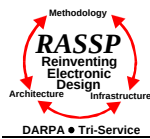
Reduced Fault Table

	f ₄	f ₅	f ₆
T ₂	1		
T ₃		1	
T ₄			1

This slide shows the process of fault table collapsing.

To collapse equivalent faults, remove all but one equivalent column. This results in the removal of the f₁ and f₂ columns. To collapse dominant faults, remove all *dominating* columns notice that the *column* for f₃ dominates the columns for f₄ and f₅ because it has "1"s in all of the places where those columns have "1"s and therefore it can be removed. This terminology is a bit confusing because in terms of faults, f₄ and f₅, *dominate* fault f₃.

To collapse tests, remove all but one equivalent rows and remove all *dominated* rows.



Fault Simulation Algorithms



- | **The goal is to determine the list of faults in a circuit-under-test that are detected by a specific test vector**
- | **The general procedure is to simulate the good and faulty circuits and determine if they produce different outputs**
- | **Consists of five specific tasks:**
 - m **Good circuit simulation**
 - m **Fault specification (fault list generation and collapsing)**
 - m **Fault insertion**
 - m **Fault-effect generation and propagation**
 - m **Fault detection and discarding**

Copyright © 1995-1999 SCRA

57

Fault simulation is one of the most widely used of the test technologies presented herein. Many efficient algorithms for fault simulation have been developed.



Methodology
RASSP
Reinventing
Electronic
Design
Architecture Infrastructure
DARPA • Tri-Service

Fault Simulation Algorithms (Cont.)



RASSP E&F
SCRA • GT • UVA
Rapidgo • USC • ACD

| **Major types:**

- m **Parallel fault simulation**
- m **Deductive fault simulation**
- m **Concurrent fault simulation**
- m **Parallel Pattern Single Fault Propagation (PPSFP)**

Copyright © 1995-1999 SCRA

58

The major types of fault simulation are presented next. Most other work on fault simulation has been in increasing the efficiency of these types of fault simulation or actually paralleling the fault simulation algorithms to be run on parallel/distributed computers.

Parallel Fault Simulation

- | The good circuit and a fixed number of faulty circuits N are simultaneously simulated
- | The values of lines in the circuit are packed into the words of the host computer - if the host has a word length of W , then $N=W-1$
- | For a group of F faults, F/N passes are required for fault simulation
- | Bitwise logical operations of the host computer (*and, or, xor, not*) are used to simulate the gates of the circuit-under-test
- | Fault insertion is also handled by bitwise operations of “*masks*”

Parallel fault simulation uses the word width of the computer on which it is run (say, N bits) to simulate $N-1$ faults in parallel. Bitwise logical operation of the computer are used to simulate the logical operation of the gates in the circuit. Special techniques for fault insertion at nodes where faults exist have been developed.

Parallel Fault Simulation (Cont.)

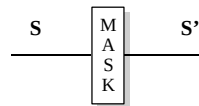
Define masks for signal S and fault in position i of word:

$\text{mask}(S)_i = 1$ if a fault exists on S for the fault in the i th position
 when S is simulated

$\text{fvalue}(S)_i = 1$ if the fault is s-a-1, 0 if the fault is s-a-0

Then a new signal S' with the faults injected can be defined as:

$$S' = S \cdot \overline{\text{mask}(S)} + \text{mask}(S) \cdot \text{fvalue}(S)$$



Model of fault injection

This figure details the process of injecting a fault on line S using a fault mask in parallel fault simulation. The masks are set up for the fault to be injected in the i th position in the word, and only bitwise operations on the word are necessary to perform fault injection. So, for example, if the good value for S is "1" and S is s-a-0, then the mask for the i th bit of S is "1" and the fvalue for S is "0", then:

$$S' = \overline{1} \cdot 1 + 1 \cdot 0 = 0$$

or if the good value for S is "0" and S is s-a-1, then:

$$S' = \overline{0} \cdot 1 + 0 \cdot 1 = 1$$

Ref: [Fujiwara85]



RASSP
Reinventing
Electronic
Design

Architecture Infrastructure

DARPA • Tri-Service

Parallel Fault Simulation (Cont.)

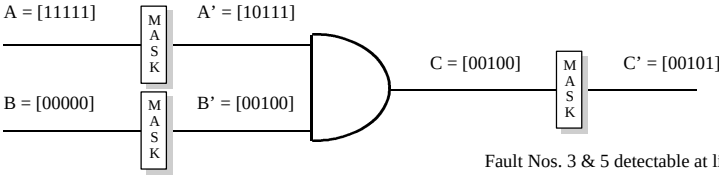


RASSP E&F
SCRA • GT • USA
Rapidgo • U.S. • ADX

Example (Word Width $W = 5$ bits)

Bit Position	Fault
1	Fault-Free
2	A s-a-0
3	B s-a-1
4	C s-a-0
5	C s-a-1

Line	Mask	Fvalue
A	[01000]	[00000]
B	[00100]	[00100]
C	[00011]	[00001]



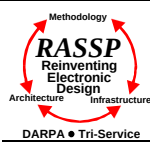
Fault Nos. 3 & 5 detectable at line C

Copyright © 1995-1999 SCRA

61

Here is an example of parallel fault simulation on a AND gate with four faults being injected. The resulting word on the output shows that the faults in the 3rd and 5th bit position have been detected (B s-a-1 and C s-a-1). This can be automatically determined by XORing each bit in the output word with the good circuit value (bit position zero)

Ref: [Fujiwara85]



Deductive Fault Simulation



- | **Explicitly simulates the behavior of the good circuit only**
- | **Simultaneously deduces from the “good” state, all faults that are detectable at any line**
- | **Only one pass through the circuit is needed although it takes a long time to make this pass**
- | **More memory intensive than parallel fault simulation**
- | **For each line in the circuit A , a list of faults L_A that produces the complement of the good state of A is calculated**

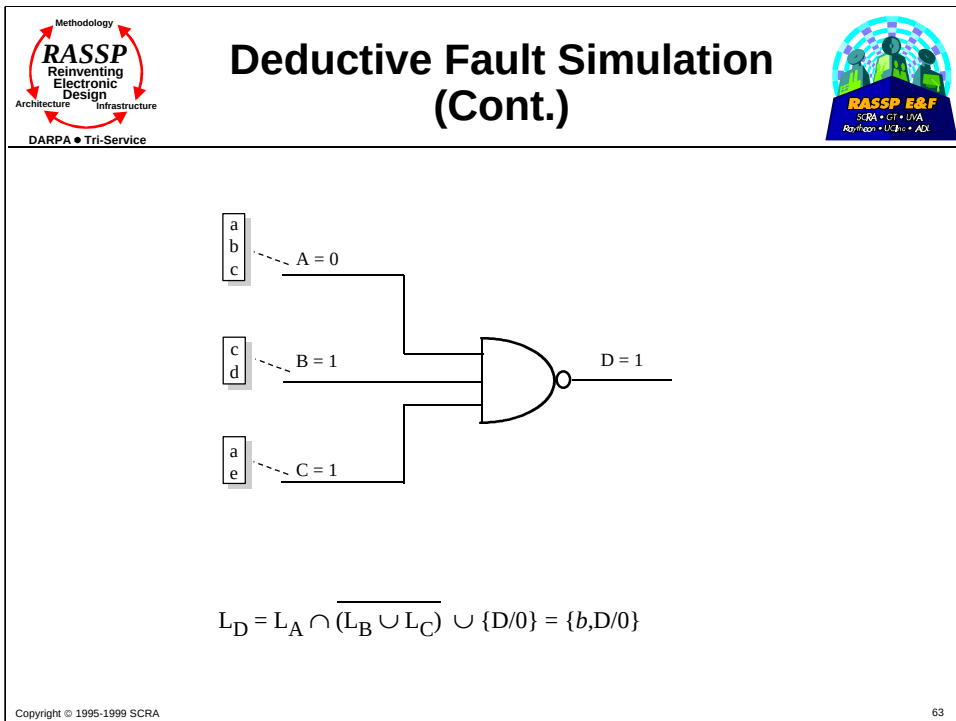
Copyright © 1995-1999 SCRA

62

Deductive fault simulation is another algorithm that appears to be more efficient than parallel fault simulation in terms of run time, but is much less efficient than parallel in terms of memory usage.

The main reason for the lower run time is the fact that only one forward pass through the circuit is needed for good circuit simulation and then one for deducing the faults at each gate. In fact, these two processes can be combined into one total pass.

Ref: [Fujiwara85]



This figure illustrates the deductive fault simulation process for a NOR gate. First, forward simulation is performed to determine all of the good values on the inputs and outputs. Then, using these values, the list of all faults that cause changes in the output are “deduced” from the input values and the gate function.

In this case, only faults that cause the values on input A to change (to “1”) without causing the values of inputs B or C to change (to “0”), will cause the output to change. Also, the faults within the logic element itself that cause the output to change must be considered. Thus, the list of faults visible on the output D is as shown above.

Ref: [Fujiwara85]



Concurrent Fault Simulation



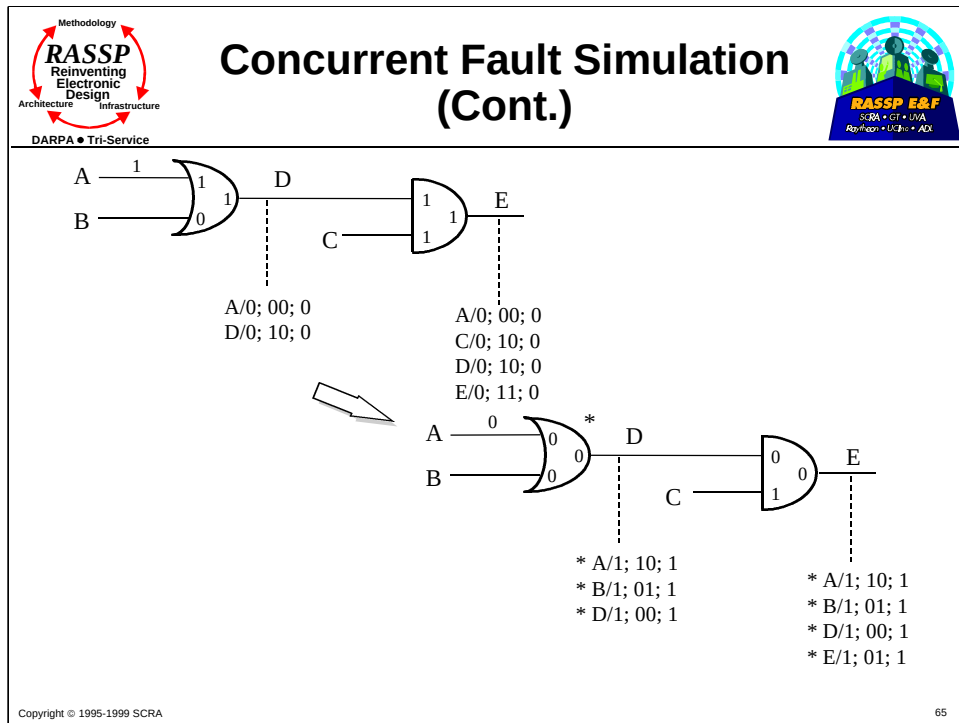
- | **Designed to take advantage of the fact that the behavior of the good and faulty circuits is rarely different**
- | **Needs only one pass through the circuit**
- | **Builds fault list at each node in the circuit (linked list of faults available at that node given the current circuit state)**
- | **Resimulates circuit and reconstructs fault lists in an event-driven manner**
- | **Removes detected faults from fault list**
- | **May require a lot of memory early in the simulation process when many faults are undetected**

Copyright © 1995-1999 SCRA

64

Concurrent fault simulation is another fault simulation algorithm that, like deductive, is more efficient than parallel in terms of runtime, but less so in terms of memory usage. Only a single pass through the circuit is needed during fault simulation.

Concurrent fault simulation maintains a linked list of the “faulty circuits” at each line in the circuit. Only the circuits that do not agree, in terms of their input and output relationships, are explicitly simulated.



In this example the linked lists used in concurrent fault simulation are illustrated. In the upper circuit, the faults A s-a-0, and D s-a-0 affect the input/outputs of the AND gate and appear at the node D, These faults also effect the node E and thus appear in the linked list for that node along with C s-a-0 and E s-a-0.

If line A changes to a value of "0", the linked list at D changes to drop fault A s-a-0 and D s-a-0 and add faults A s-a-1, B s-a-1, and D s-a-1. Since C is still '1', these appear in the linked list for node E as well although the the faults C s-a-0 and E s-a-0 are dropped and the fault E s-a-1 is added.

Ref [Fujiwara85]



Parallel Pattern Single Fault Propagation



- | **A single fault is simulated at a time**
- | **Multiple patterns are simulated in a single pass/word similar to parallel fault simulation**
(single fault propagation - SFP)
- | **A fault is simulated until the faulty values either become identical to the fault-free values or the fault is detected**
- | **SFP takes advantage of the fact that most faults are either detected in a simulation pass or have their effects “die out” fairly rapidly within a small area of the circuit**
- | **PPFSP is currently one of the most efficient fault simulation methods**

Copyright © 1995-1999 SCRA
66

Parallel Pattern Single Fault Propagation is a fault simulation technique that combines two innovations. First, instead of simulating faults in parallel across a word as in parallel fault simulation, PPSFP simulates N patterns (when possible) across a computer word, on one fault at a time.

The second innovation is that during single-fault propagation, if a fault effect disappears, due to a reconvergent fanout problem, for example, simulation of that fault stops. Thus, fault simulation proceeds in an “event-driven manner.” This technique typically results in many fewer gate evaluations than are necessary for the other types of fault simulators. The current fastest fault simulators reported in the literature are PPFSP derivatives.

Ref: [Abramovici90]



Sequential Circuit Fault Simulation

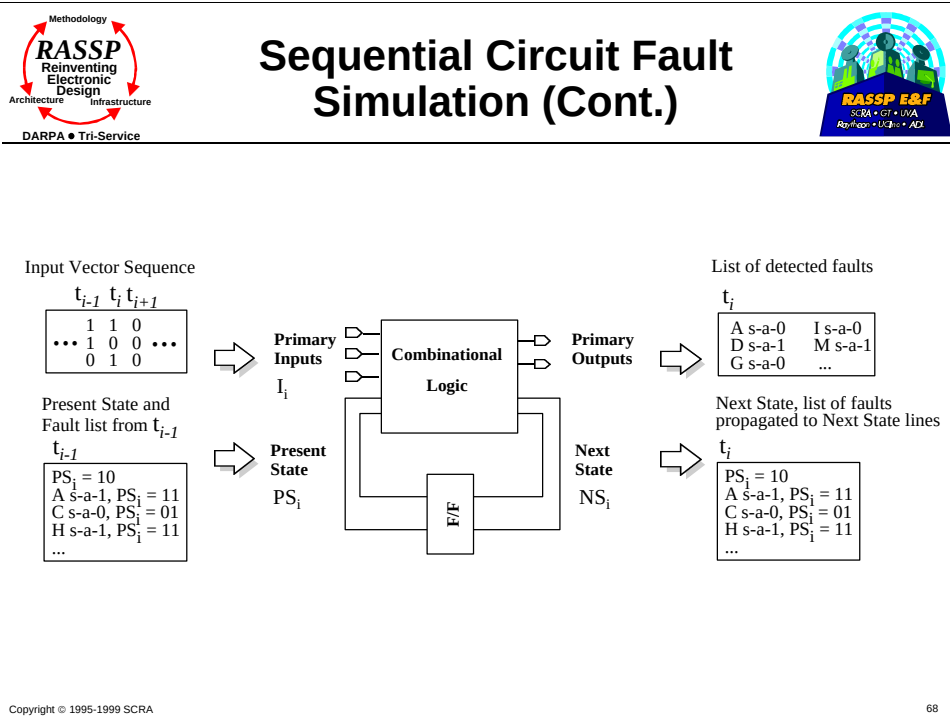


- | Like test generation, fault simulation is more complex for sequential circuits vs. combinational circuits
- | Inputs are applied as a sequence of vectors, one at each time step (clock period)
- | Fault lists propagated to Next State lines at time i must be applied to Present State lines along with faulty values at time $i+1$

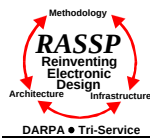
Copyright © 1995-1999 SCRA

67

Like test generation, fault simulation for sequential circuits is more difficult than for combinational circuits. During fault simulation, if a fault is propagated to a next state line, then it and its effect must be propagated to the present state lines and applied with the next input in the sequence. This process can result in the manipulation of large fault lists which hurts efficiency.



This figure graphically illustrates the fault list propagation problem.



Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | **Introduction (Part 2)**
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Built-In Self Test
- | Hierarchical Design for Test
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | Summary

Copyright © 1995-1999 SCRA

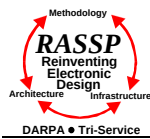
69



Introduction



- | **This section assumes basic familiarity with the following subjects:**
 - m **Testing goals and objectives**
 - m **Single stuck-at faults**
 - m **Basic Automatic Test Pattern Generation (ATPG) algorithms**
 - m **Basic fault simulation algorithms**
- | **A review of these subject can be found in the previous section**



Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | **I_{DDQ} Testing**
- | Design for Testability Techniques
- | Built-In Self Test
- | Hierarchical Design for Test
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | Summary

Copyright © 1995-1999 SCRA

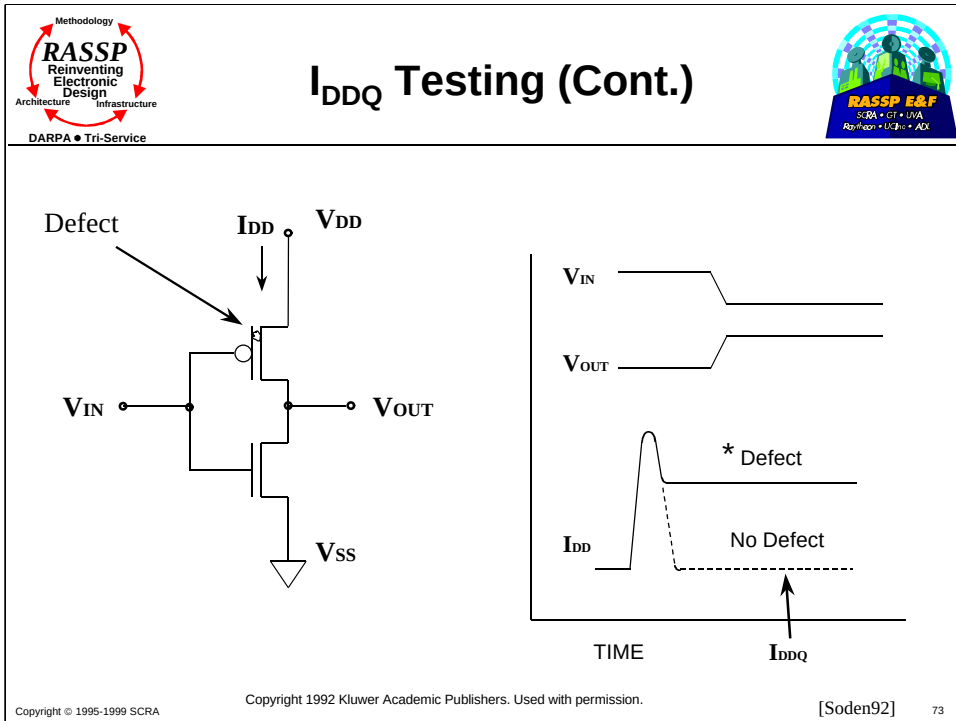
71

I_{DDQ} Testing

- | All of the test techniques discussed thus far use voltage measurement (value) techniques
- | Many defects in CMOS circuits can be detected by current measuring techniques
- | A fully static CMOS gate consumes significant current only when switching
- | Quiescent current for MOS devices (I_{DDQ}) is typically in the fA range
- | Most physical defects will raise that current level by several orders of magnitude or more

IDDQ testing is becoming more prevalent both in research and in new industrial applications. IDDQ testing is based on the physical fact that fault-free CMOS circuits consume VERY LITTLE current in the quiescent state. The presence of faults, under the right conditions, can increase this quiescent current by an order of magnitude which can be used to detect the fault.

Ref: [Soden92]



This figure shows the effect of a defect in terms of a gate-source short in the P transistor of an inverter. When the input voltage goes low such that this transistor turns on, significant current flows between the source and gate such that I_{DDQ} increases dramatically. However, because the gate output goes high as it should, traditional stuck-at fault testing would not detect this defect.

If the defect doesn't affect function, why be concerned about detecting it? The answer is that the defect may be such that after a certain operating time, it will cause the device to fail, causing a detectable fault and thus an error.

Ref: [Soden92]

I_{DDQ} Testing (Cont.)

I Advantages

- m Test generation is easier - faults must be activated, but not propagated to a Primary Output
- m I_{DDQ} testing can detect defects that are not modeled by the stuck-at model
 - q Bridging faults
 - q Gate oxide defects
 - q Shorts between any two of the four terminals of a transistor
 - q Partial defects - defects that do not affect the logic of the circuit, but may effect reliability
 - q Some delay faults
 - q Some stuck-open faults

The major advantages of I_{DDQ} testing include the fact that it can potentially detect faults that are undetectable by other models. Also, test generation can be easier because the fault only has to be activated (sensitized) and it will be detected. The fault effect (value) doesn't need to be propagated to a primary output.

I_{DDQ} Testing (Cont.)

I Disadvantages

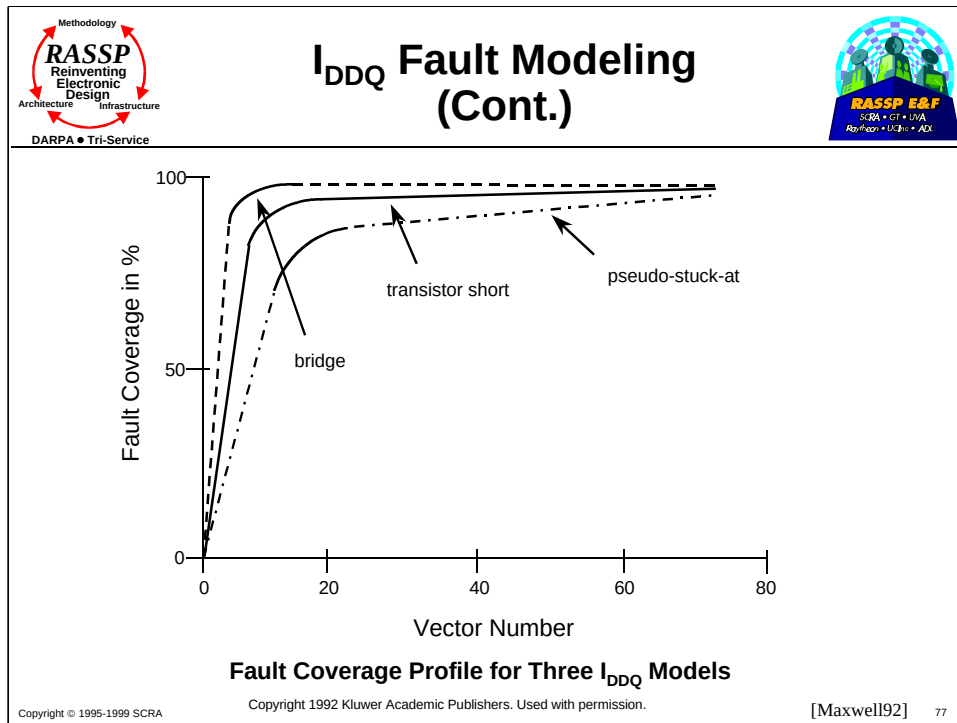
- m Since normal I_{DDQ} is very low, measurements must be very precise
 - q Measurement takes a significant amount of time (1ms)
 - q Setting I_{DDQ} threshold for bad devices is hard
- m Circuit-under-test must contain all static devices (slower), i.e., *no*:
 - q Dynamic circuitry
 - q Pull-ups or pull-downs on I/O buffers
 - q Specialized speed optimized circuitry such as RAM sense amps that draw significant static current

The major disadvantage of I_{DDQ} testing is the measurement of the quiescent current which must be very precise and thus takes a long time relative to value (voltage) measurements. Also, to be suitable for I_{DDQ} testing, certain restrictions must be placed on the design.

I_{DDQ} Fault Modeling

- | **Stuck-at Fault Model**
 - m Drive the faulty node to the value opposite the stuck-at value
- | **Transistor Short Model**
 - m Specific patterns can be derived to test for all possible combinations of shorts between all four terminals
- | **Bridging Fault Model**
 - m Only physically adjacent nodes need be tested
 - m Drive the adjacent nodes to opposite values

There are several fault models that can be used for IDDQ test generation. All of them operate at the logic or transistor level.



This graph shows the fault coverage vs number of test vectors for three IDDQ models. Note that for bridging faults and transistor shorts, very high fault coverage is obtained for a very few vectors. This doesn't mean that this necessarily applies for defect coverage.

Ref [Maxwell92]

I_{DDQ} Test Generation

- | **Every Vector I_{DDQ}**
 - m Utilize logic test patterns developed for voltage-sensing test
 - m Measure I_{DDQ} after every vector
 - m Most useful for first silicon prototype testing
- | **Selective I_{DDQ}**
 - m Perform I_{DDQ} measurement on selected subset of entire vector set
 - m Run entire functional test at speed, but “pause” after vector selected for I_{DDQ} measurement
- | **Supplemental I_{DDQ}**
 - m Add a specific set of vectors designed for I_{DDQ} measurement to the end of the full-speed functional tests

There are several different methods that can be used to develop I_{DDQ} tests. Most are used in conjunction with voltage (value) testing for the best speed/quality tradeoff.



RASSP
Reinventing
Architecture, Methodology,
Design, Infrastructure

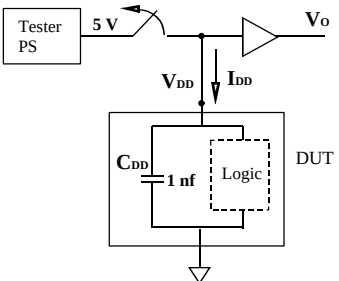
DARPA • Tri-Service

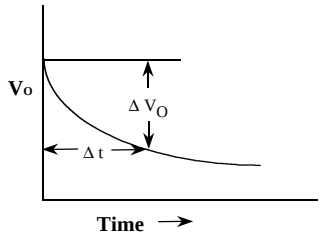
I_{DDQ} Measurement Techniques



RASSP E&F
SCRA • GT • USA
Reprogram • U.S. • ADX

I Off-Chip Measurement Unit (tester power supply)





I On-Chip Measurement Unit

- m Built-in current monitors have been proposed, but not yet widely realized
- m A major consideration is not degrading the at-speed performance of the device-under-test

Copyright © 1995-1999 SCRA
Copyright 1992 Kluwer Academic Publishers. Used with permission.
[Soden92] 79

As noted before, I_{DDQ} measurement is the biggest stumbling block. What's typically done is to apply the test and power from the tester's power supply and then remove it. A defectless device will maintain its output levels for a fairly long time because of the low I_{DDQ} . A device with an I_{DDQ} -detectable fault will discharge faster. This RC time constant measuring method is the main reason I_{DDQ} testing takes so long.

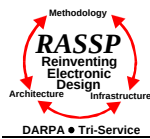
Several techniques for on-chip I_{DDQ} measurement have been theorized, but not implemented. This technique would be necessary for I_{DDQ} BIST.

Ref: [Soden92]

I_{DDQ} Design for Testability

- | **Internal Tri-State Buses**
 - m Short periods of bus contention may be functionally OK, but cause problems with I_{DDQ} testability
 - m Design controllers for non-overlapping bus drivers
- | **Pull-ups and Pull-downs**
 - m Pull-up (down) resistors are commonly used in I/O pads
 - m Eliminate or use separate power supply for I/O pads
- | **Dynamic Circuitry**
 - m Precharge - discharge type logic typically used for high speed design
 - m Ensure all nodes are precharged on every clock cycle
- | **Circuits with Non-Zero Static Current**
 - m Sense-Amps for memory cells, etc.
 - m Avoid or use separate power supply

In order to ensure/improve I_{DDQ} testability, several design constraints must be applied to limit good circuit I_{DDQ} .



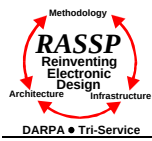
Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | I_{DDQ} Testing
- | Introduction (Part 2)
- | **Design for Testability Techniques**
- | Built-In Self Test
- | Hierarchical Design for Test
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | Summary

Copyright © 1995-1999 SCRA

81



Section Outline



- | **Design for Testability Techniques**
 - m Ad hoc design for testability techniques
 - m Structured design for testability techniques



Design for Testability Techniques



- | **The goal of Design for Testability techniques is to increase the ease with which a device can be tested**
 - m Increase the controllability and observability of internal points in the circuit
- | **Categories of Techniques**
 - m **Ad Hoc (Problem oriented)**
 - q Partitioning
 - q Degating (form of partitioning)
 - q Test points
 - q Bus structured architectures
 - m **Structured Techniques**
 - q Scan Design
 - q Boundary scan

Copyright © 1995-1999 SCRA

83

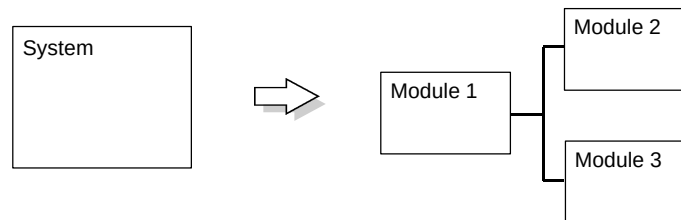
Three major categories of Design for test have been developed. BIST is a category of DFT because, obviously, the most testable chip is one that tests itself. However, it is such a big topic that we will cover it in its own section.

Ref: [Williams83]

Ad Hoc Design for Testability Techniques

I Partitioning - “Divide and Conquer”

- m Physically divide the system into multiple chips or boards
- m On board-level systems, use jumper wires to divide subunits
- m Has major performance penalties

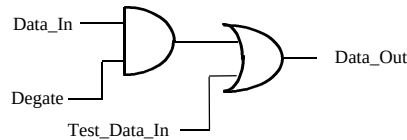


Obviously, anytime you can partition a system into subsystems (physically), the testability is improved. The major problem with this technique is the major performance penalty it incurs.

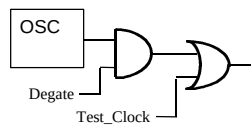
Ad Hoc Design for Testability Techniques (Cont.)

- I **Degating** - another technique for separating modules on a chip/board with lower performance penalties

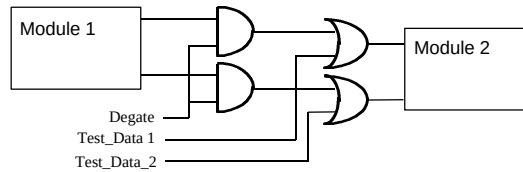
Degating Logic



Data_Out	Degate	Test_Data_In
Data_In	1	0
Test_Data_In	0	<value>



Clock Degating



Module Partitioning

© IEEE 1983
 [Williams83]

Copyright © 1995-1999 SCRA

85

Degating is an on-chip method of partitioning. It allows internal lines on a chip or MCM to be externally controlled in a test mode. It does suffer from the fact that it adds two gate delays to each line that is degated.



RASSP
Reinventing
Electronic
Design

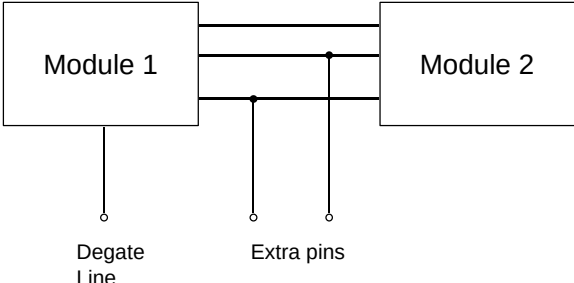
DARPA • Tri-Service

Ad Hoc Design for Testability Techniques (Cont.)



RASSP E&F
SCRA • CT • USA
Rep/agon • U.S. • ADX

| Test Points - insert additional lines to control and observe internal nodes



Copyright © 1995-1999 SCRA
© IEEE 1983
[Williams83] 86

The test points approach is similar to degating except that the external lines are used to both observe (in normal operation) and drive the test nodes. In this case, the degating signal must be able to tri-state the internal nodes (as on an internal bus) so that they can be driven by the external test lines.



RASSP
Reinventing
Electronic
Design

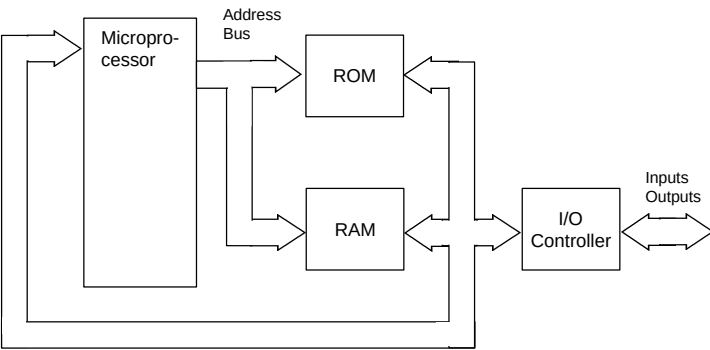
DARPA • Tri-Service

Ad Hoc Design for Testability Techniques (Cont.)



RASSP E&F
SCRA • GT • UVA
Rapidgo • U.S. • ADX

| Bus Structured Architecture - bus adds internal control and observe points



```

graph LR
    MP[Microprocessor] -- Address Bus --> ROM[ROM]
    MP -- Address Bus --> RAM[RAM]
    ROM --> MP
    RAM --> MP
    RAM <--> IOC[I/O Controller]
    IOC <--> IO[Inputs/Outputs]
    
```

Copyright © 1995-1999 SCRA

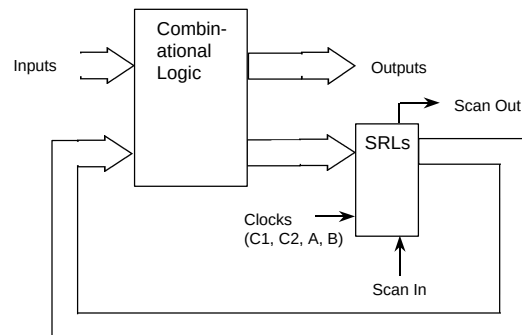
© IEEE 1983

[Williams83] 87

By forcing the design to be bus structured, internal control and observe points are increased and can be used as test points.

Structured Design for Testability Techniques

- | **Scan Design** - the general technique is to make a FSM testable by making the internal state variables controllable and observable
- | This is accomplished by changing the latches for the state bits to scannable latches

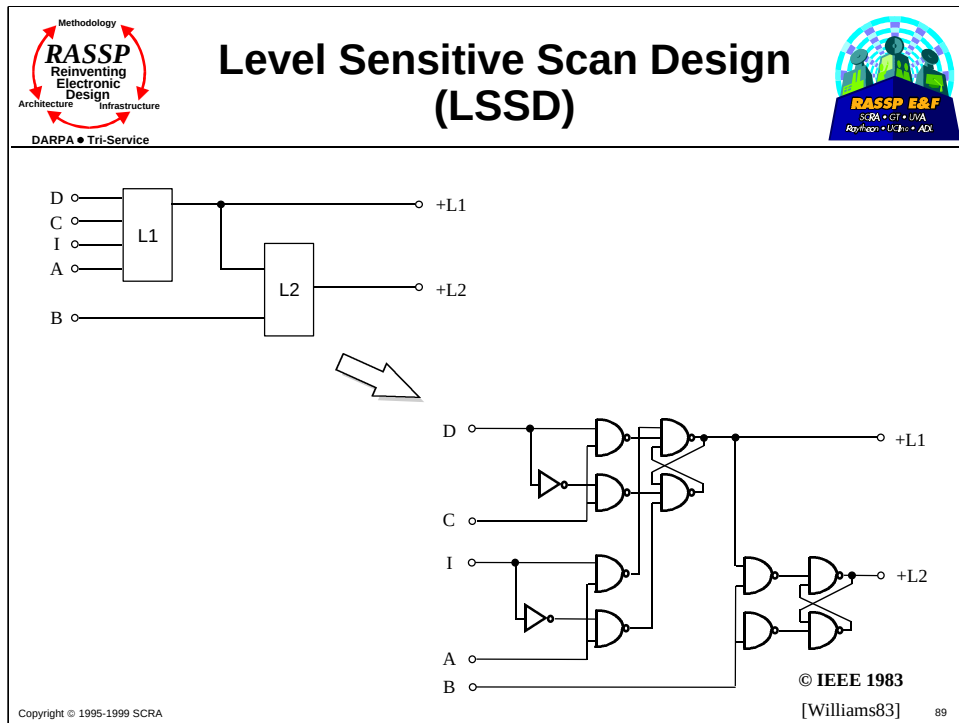


© IEEE 1983

[Williams83] 88

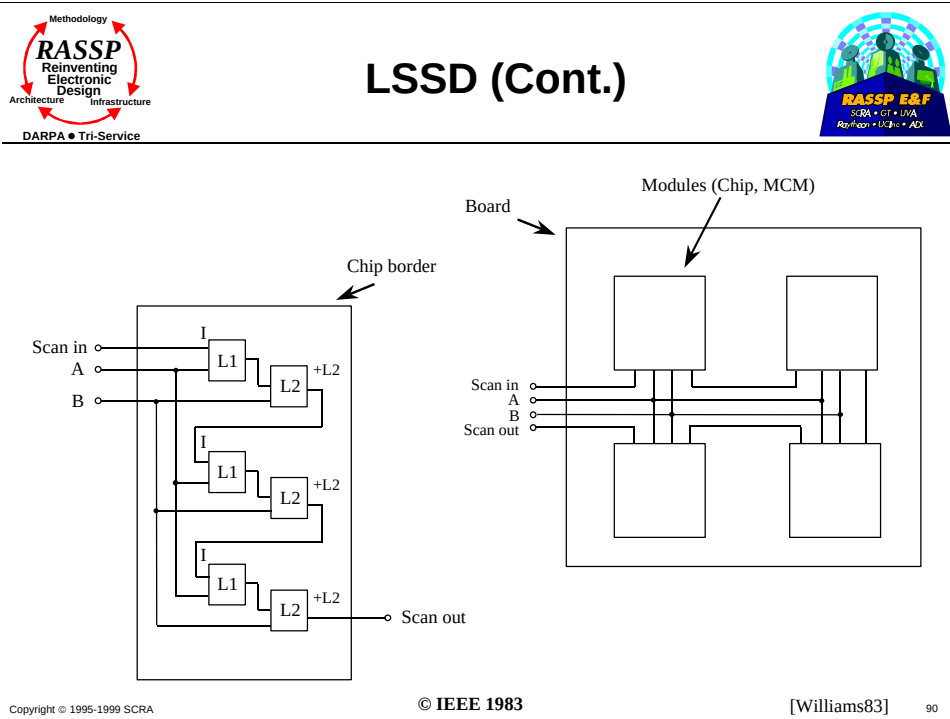
Copyright © 1995-1999 SCRA

Scan design is the most common structured design for testability technique. In it, all of the latches in the design are made externally controllable and observable. Therefore, the testing problem becomes one of combinational logic testing only.

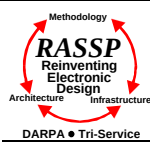


This figure shows the general configuration for of a level sensitive scan design latch as well as a NAND/NOT implementation.

D is the normal data line and C is the normal clock line. Line L1 is the normal output. Lines I, A, B and L2 form the shift portion of the latch. I is the shift data in and L2 is the shift data out. A and B are the two phase, non overlapping shift clocks.



The figure on the left shows how the LSSD registers are configured into a scan chain within a single chip. The figure on the right shows how multiple LSSD chips are configured into a complete scan chain.



LSSD Design Rules



- | **Rule 1: All internal storage is implemented in hazard-free polarity-hold latches**
- | **Rule 2: The latches are controlled by two or more non-overlapping clocks such that latches that feed one another can not have the same clock**
- | **Rule 3: It must be possible to identify a set of clock primary inputs from which the clock inputs to SRLs are controlled either through simple powering trees or through logic that is gated by SRLs and/or non-clock primary inputs**
- | **Rule 4: Clock primary inputs may not feed the data inputs to latches either directly or through combinational logic, but may only feed the clock input to the latches or the primary outputs**

Copyright © 1995-1999 SCRA

91

The use of LSSD requires the adherence to several design rules. A chip with only LSSD latches is not truly LSSD unless it adheres to these rules. The rules are designed to ensure that all internal nodes are controllable and observable via the shift registers.



LSSD

Advantages/Disadvantages

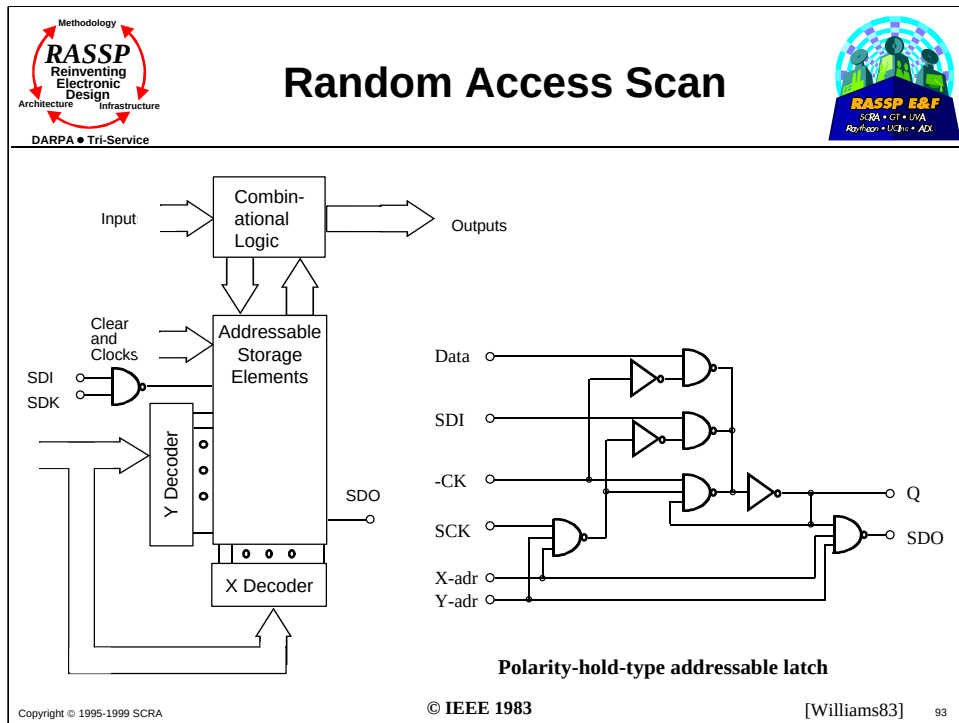


- | **Advantages**
 - m **With LSSD, the testing problem is transformed from one of sequential circuit testing to one of combinational circuit testing**
 - m **By adding controllability/observability to the state variables, LSSD also eases functional testing**
- | **Disadvantages**
 - m **Additional area is required to fabricate the LSSD latches (area overhead)**
 - m **Additional time is required to latch the next state into the LSSD registers (speed overhead)**
 - m **Additional time is required to scan in/out test vectors and responses - at-speed testing is not supported (testing overhead)**
 - m **Clock generation and distribution for LSSD is more difficult**

Copyright © 1995-1999 SCRA
92

The major advantage of LSSD is that it transforms the testing problem from sequential to combinational testing, which is a much more tractable problem.

The major disadvantage of LSSD is probably the speed overhead because it adds several gate delays to the critical path of the design. The testing overhead can be a big problem too because some ASIC vendors charge by the clock cycle for test application.

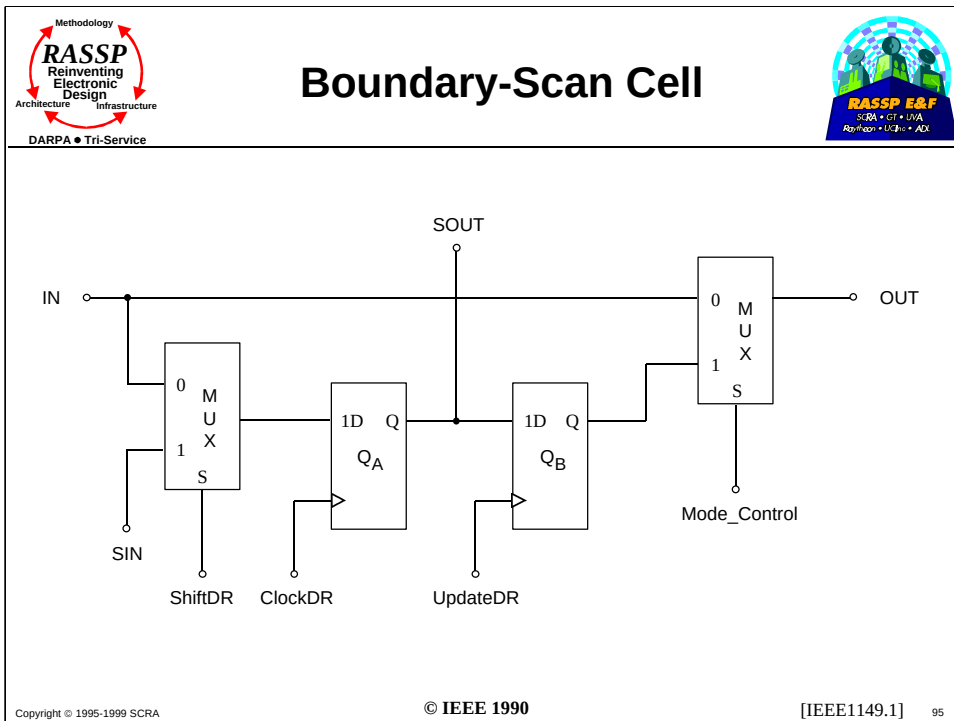


Another scan design approach is random access scan. In this scan approach, each latch in the design is separately addressable and, therefore, individually controllable and observable. The major disadvantage of this technique is the large amount of additional logic required, namely, the scan latches and the X-Y decoder logic.

Boundary-Scan

- | **Consists of adding scan registers to the inputs and outputs of ICs**
- | **Allows for efficient testing at the board level**
 - m Testing of board-level interconnect
 - m Isolation and testing of chips via chip-level BIST or the application of chip-level tests via the test bus
- | **Requires the addition four I/O ports to the chip - Test Access Port (TAP)**
 - m TCK - test clock
 - m TMS - test mode signal
 - m TDI - serial test data in
 - m TDO - serial test data out
- | **Also requires the addition of logic to control the testing process - TAP Controller**

Boundary scan is probably the most widely used DFT technique. It has been adopted by the IEEE as a standard (to be discussed later). It requires the addition of some logic to the chip for control and some additional I/O ports, but the overhead is minimal.



This figure shows the basic structure of a boundary-scan cell. One of these cells is added to each I/O port on the chip. One reason that boundary scan may be popular is that the relative cost of adding the scan cells to the I/O pads is low compared to the cost of adding full scan. For example, only a mux is added to the normal I/O path, which does add some delay, but it's in the context of the already large I/O pad delay. Second, there is some additional logic that has to be added to the Pad buffer, but most of the pad area is dominated by the size of the physical pad, and the additional logic doesn't increase it by much.

Ref: [IEEE1149.1]

Boundary-Scan Cell Modes

Normal Mode: *Mode_Control* = '0'

m Data passes from *IN* to *OUT*

Scan Mode: *ShiftDR* = '1', *ClockDR* = scan clock

m Serial data is shifted in from *SIN* and out to *SOUT*

Capture Mode: *ShiftDR* = '0', *ClockDR* = 1 clock pulse

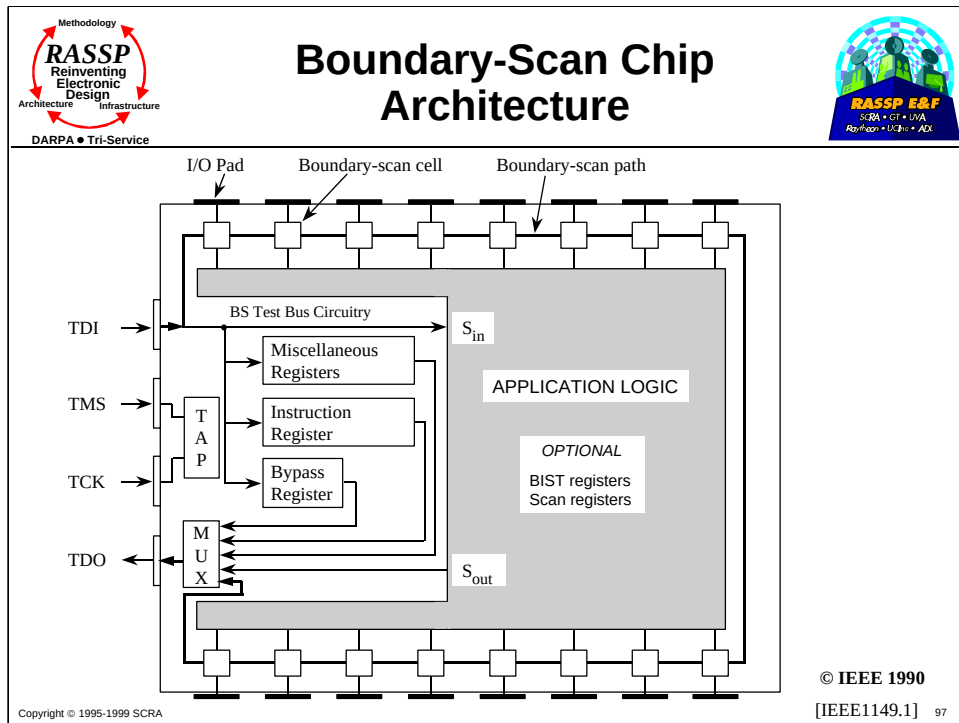
m Data on the *IN* line is clocked into *Q_A*

Update Mode: with *Q_A* loaded, *Mode_Control* = '1', *UpdateDR* = 1 clock pulse

m Data clock into *Q_A* is applied to *OUT*

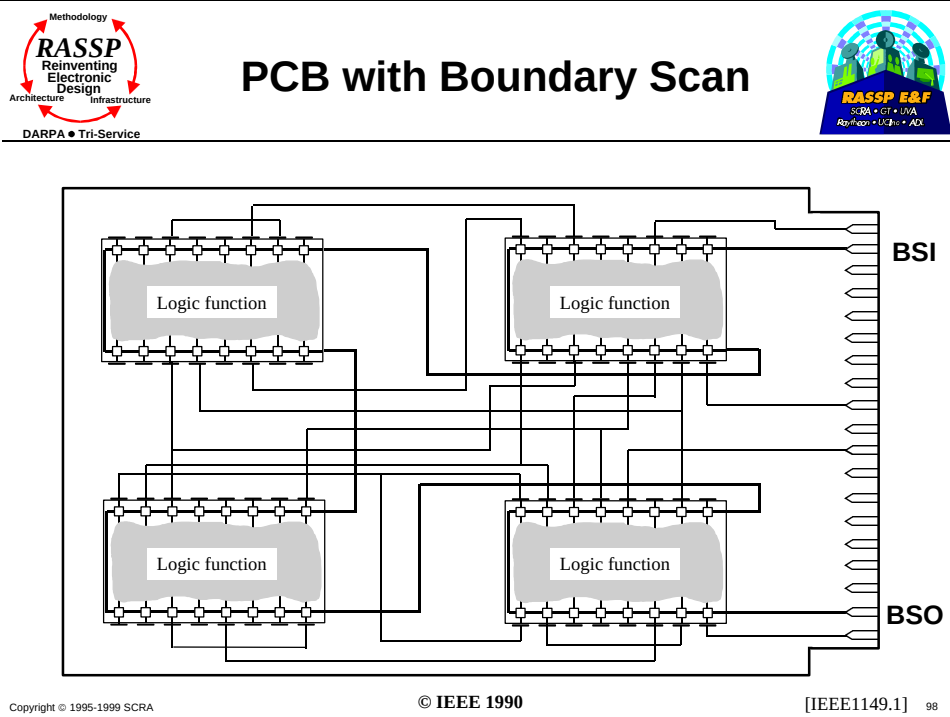
There are four major modes for the boundary-scan cell. The normal mode simply passes inputs to outputs. The scan mode passes data from *SIN* to *Q_A* and from *Q_A* to *Sout*. Capture mode loads the value on the input to *Q_A*. Finally, update mode loads values from *Q_A* to the output.

To shift in and apply data, the scan mode would be selected until the data is shifted in and then one cycle of update mode would be selected. To capture data and scan it out, one cycle of capture mode would be selected followed by the required number of scan cycles.



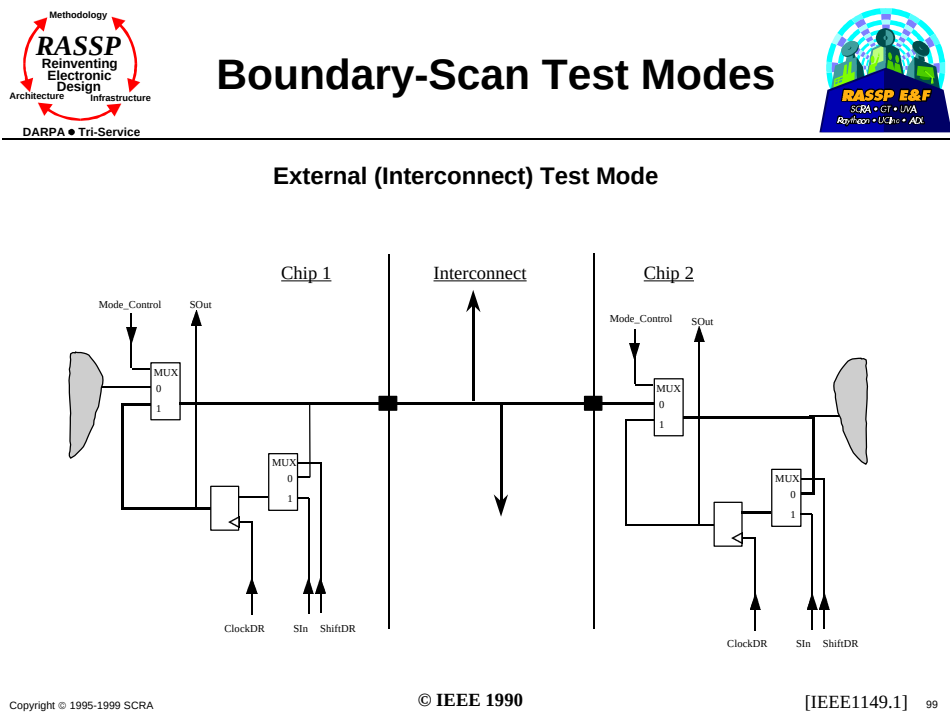
This figure shows the architecture of a boundary-scan-complaint chip. Note that the application logic of the chip itself may include DFT or BIST techniques (scan, BILBO, etc.) and this test logic is controlled by the boundary-scan TAP controller.

Ref: [IEEE1149.1]



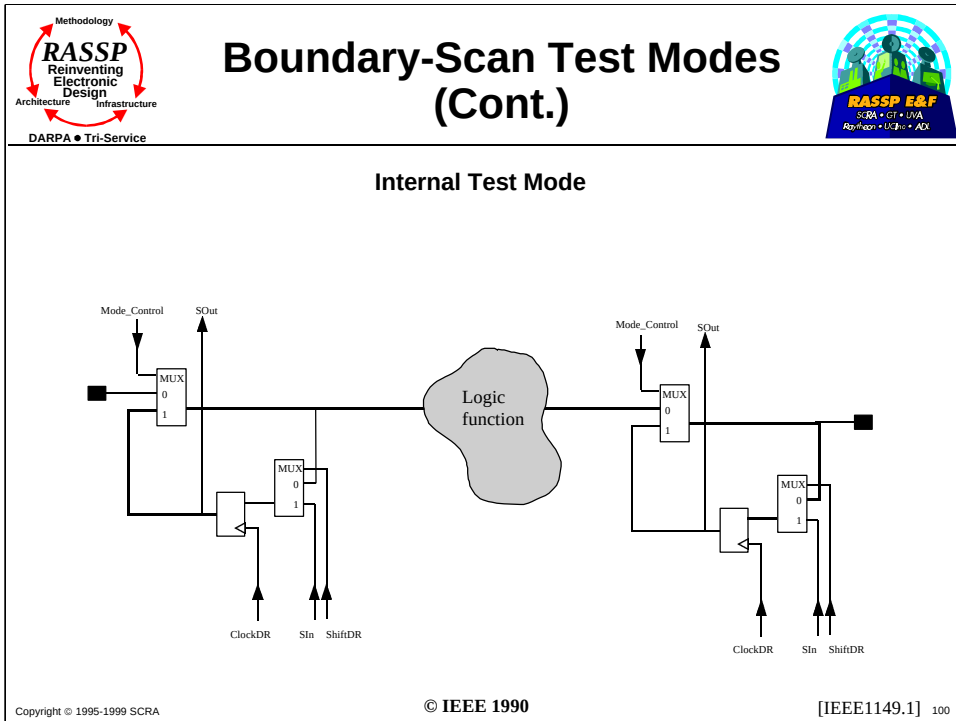
In a PCB with boundary scan, if each chip has scan latches they can be hooked together into one long scan chain. If a board is going to contain some non-scan logic, using scan where possible is still effective because a non-scan chip can be tested via boundary scan if it is surrounded by scan chips.

Ref: [IEEE1149.1]



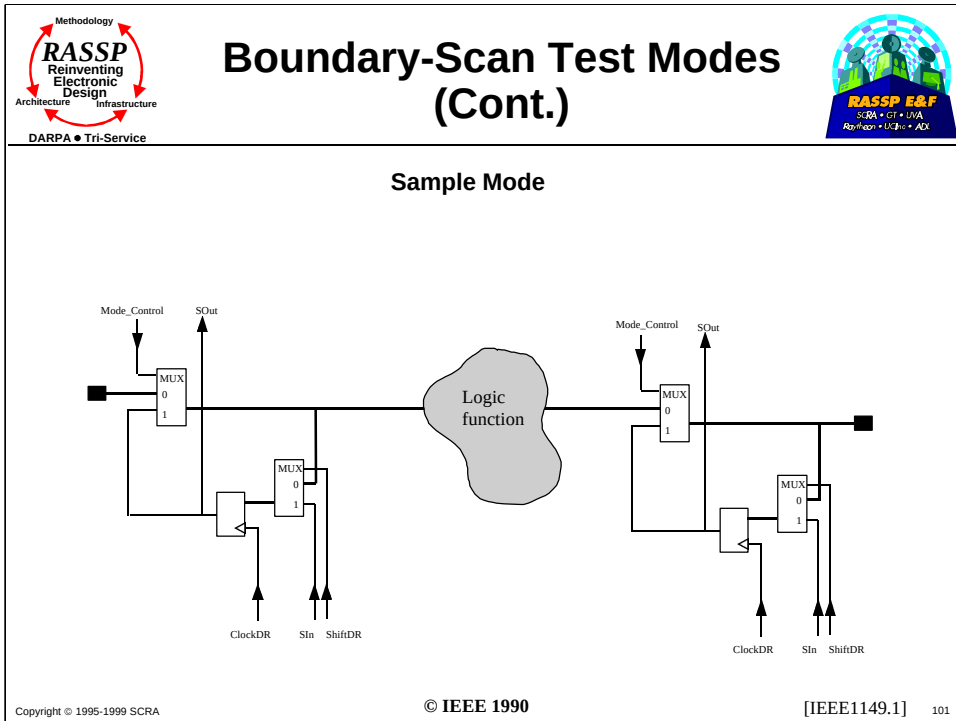
This figure shows the external or interconnect test mode for boundary scan. The scan latches in the source chip are put into the update mode after the test data is scanned in. The scan latches in the destination chip(s) are then put into the capture mode for one cycle and then into the scan mode to scan out the result.

Ref: [IEEE1149.1]



In the internal test mode, after test data is scanned in the scan cells on the chip inputs are placed in the update mode, and then the internal logic is clocked (if required). The scan cells in the outputs are then placed in the capture mode for one cycle and then placed in the scan mode for the required number of cycles to scan the result out.

Ref: [IEEE1149.1]



In the capture mode, all scan cells are placed in the capture mode for one cycle and then in the scan mode for the required number of cycles to scan the result out. This provides a “snapshot” of the I/O state of the device under test at any point in time.

Ref: [IEEE1149.1]



Boundary-Scan Advantages/Disadvantages



I Advantages

- m Area and speed overhead are lower than scan design
- m Boundary-Scan can be used to do functional testing/debugging
 - q IC internal functional tests
 - q IC cluster functional tests
 - q IC/cluster emulation
 - í Control internal buses and nets
 - q Hardware/Software integration tests
 - í Use internal scan to load/examine registers, single step, load microcode, etc.

I Disadvantage

- m Boundary-scan has some area, speed, and testing overhead in the same manner as scan design

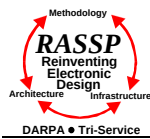
Copyright © 1995-1999 SCRA

102

Boundary-scan has a lower overhead than scan design because, in terms of speed, the increase in normal on/off chip time is much less a percentage increase than is true for scan design. Also, in terms of area, the additional logic is small compared to an I/O pad.

Another major advantage to using boundary scan is that it can also be used to scan in/out functional vectors and responses. This can be useful in a number of design verification tasks.

Boundary-scan does have non-zero area, speed, and testing overheads, and that needs to be considered when adding it.



Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | **Built-In Self Test**
- | Hierarchical Design for Test
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | Summary

Copyright © 1995-1999 SCRA

103



Section Outline



| **Built-In Self Test**

- m **Definitions**
- m **Test generation techniques for BIST**
- m **Signature analysis**
- m **BIST case study**
- m **Autonomous Built-In Self-Test**



Built-In Self Test Definitions



Built-In Self Test (BIST)
The capability of a chip, board, or system to test itself
The goal of Built-In Self Test is to add devices to a design that will allow it to test itself

Built-In-Test Equipment (BITE)
The hardware/software incorporated into a unit to provide DFT or BIST

On-Line BIST
BIST in which testing occurs during normal operation

Concurrent On-Line BIST
A form of on-line BIST in which testing occurs simultaneously with normal function

Nonconcurrent On-Line BIST
A form of on-line BIST where testing is carried out while the system is in an idle state

Copyright © 1995-1999 SCRA105

Thus far, Design for Testability techniques have been discussed. Generic DFT can be considered a passive technique where logic is added to make a design easier for an external tester to test.

Built-In Self Test is an active technique where the device is designed to test itself (with a little help).

Ref: [Abramovici90]

Built-In Self Test Definitions (Cont.)

Off-Line BIST

BIST in which testing occurs when the system is not in its normal operation

Functional Off-Line BIST

Off-line BIST that uses tests based on the functional description of the circuit-under-test

Structural Off-Line BIST

Off-line BIST that uses tests based on the structure of the circuit-under-test

Pseudo Random Pattern Generator (PRPG)

a multi-output device that generates pseudorandom output patterns - usually implemented with a Linear Feedback Shift Register (LFSR)

Multiple-Input Signature Register (MISR)

a multi-input device that compresses a series of input patterns into a (pseudo) unique signature

No additional notes are required for the definitions.



RASSP
Reinventing
Electronic
Design

DARPA • Tri-Service

Test-Pattern Generation for BIST



RASSP E&F
SCRA • GT • USA
Reprogram • UCI • ADX

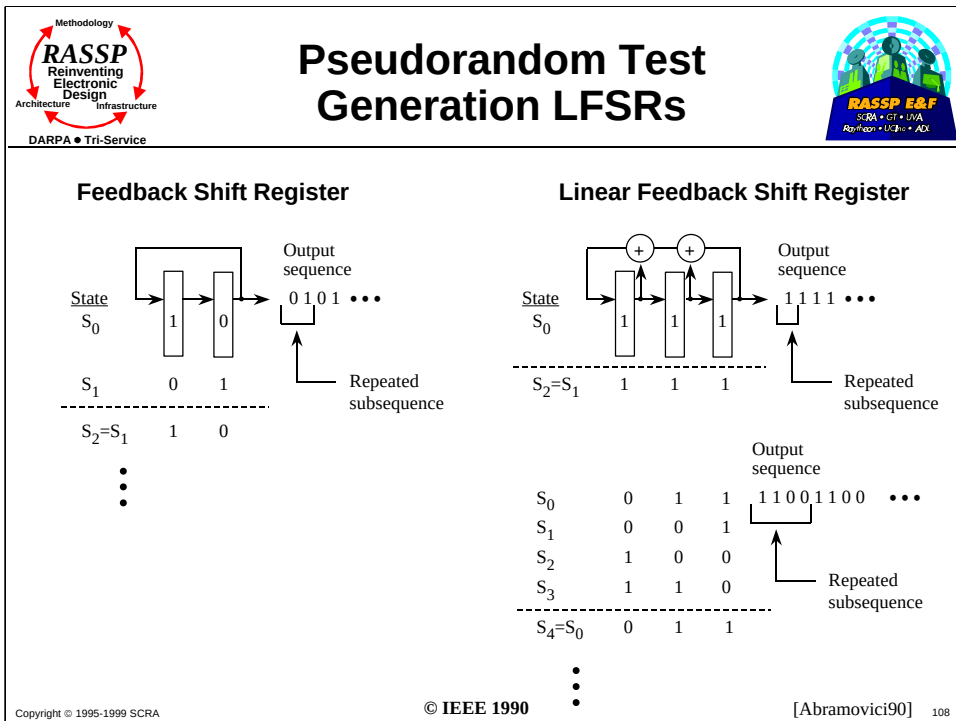
- | **Exhaustive Testing** - apply all 2^n input patterns to a combinational circuit with n inputs
 - m Binary counter can be used as a TPG
- | **Pseudorandom testing** - generate patterns that appear to be random but are in fact deterministic (repeatable)
 - m LFSR used as a TPG
 - m Weighted Pseudorandom Test Generation - LFSR used as TPG with combinational circuit to modify the probability of a "1" or "0" so they are nonuniform
 - m Adaptive Pseudorandom Test Generation - weighted random testing with the weights being modified using output of fault simulation - more than one weight used
- | **Pseudoexhaustive Testing** - segment device and test each portion exhaustively

Copyright © 1995-1999 SCRA

107

There are several ways that test patterns for BIST can be generated. Remember that the device itself is generating the test patterns, so they have to be generated or stored (rarely used) in hardware on chip.

LFSR will be explained further.



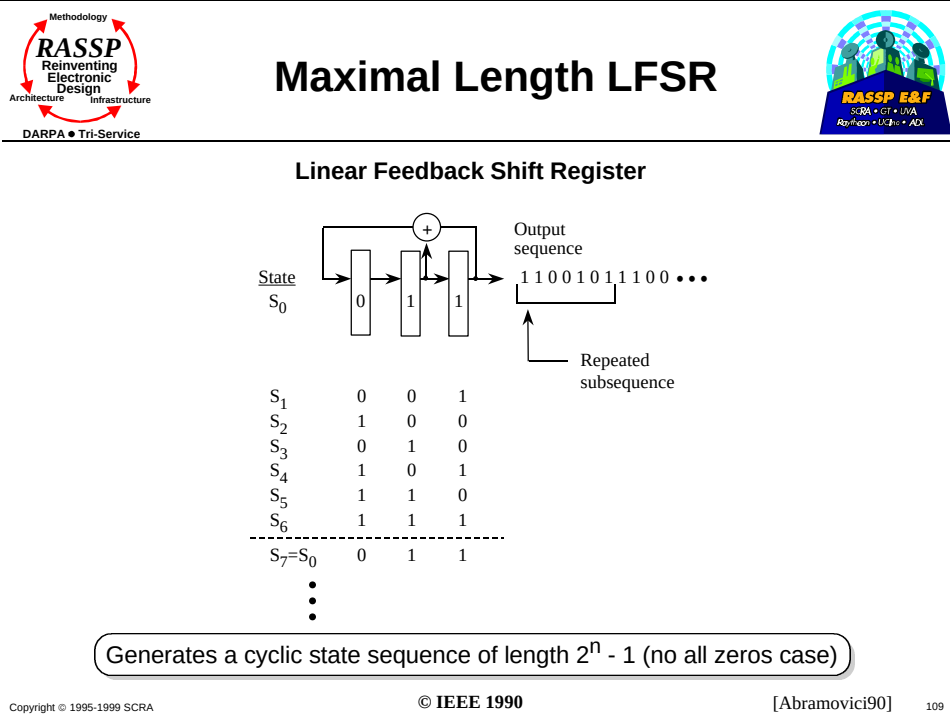
Pseudorandom (likelihood of "1" or "0" is 50%, but patterns are deterministic/repeatable) patterns may be generated by a linear feedback shift register (LFSR).

LFSRs are constructed from:

- unit delays or D flip-flops
- modulo-2 adders
- modulo-2 scalar multipliers

The devices are linear because they preserve the principle of superposition; i.e., its response to a linear combination of inputs is the linear combination of the responses of the circuit to the individual stimuli.

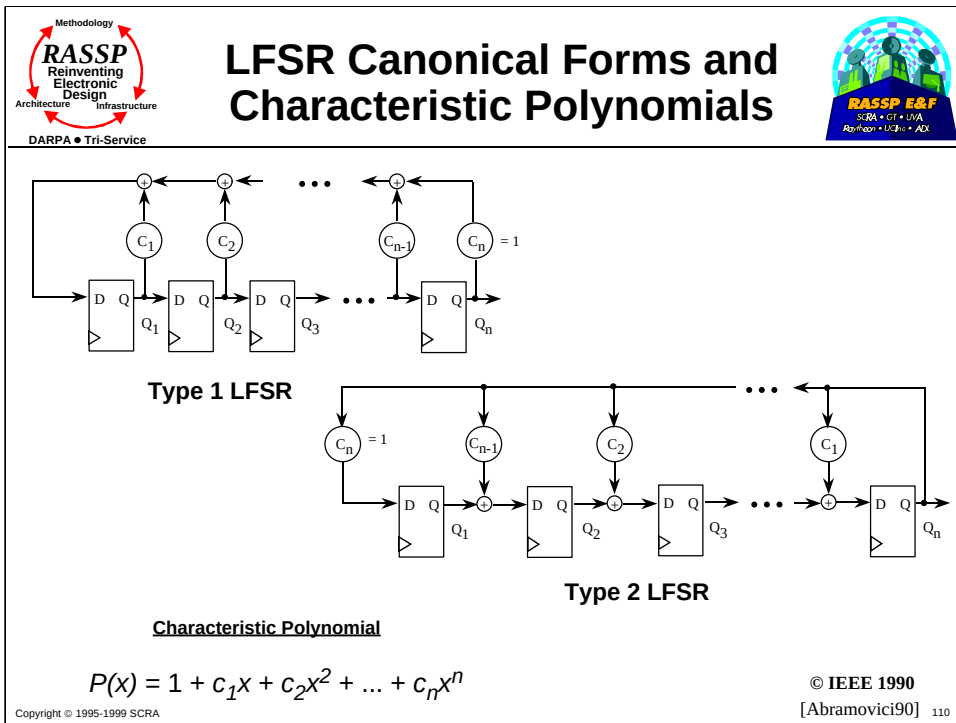
Ref [Abramovici90] page 433



The all zeros case is not possible in this type of LFSR, but notice that the probability of any bit being "1" or "0" is 50% except for that. Therefore, the sequence is pseudorandom in the sense that the probability of a "1" or "0" is approx. 50%, but the sequence is repeatable.

Like a binary counter, all $2^n - 1$ states are generated, but in a "random" order that is repeatable.

Rev: [Abramovici90]



In terms of the mathematical theory, LFSRs have a characteristic polynomial associated with them that can be expressed in terms of the feedback connections. This will be used further in the section on signature analysis.

Ref: [Abramovici90]



Signature Analysis



- | Test Patterns for BIST can be generated at-speed by an LFSR with only a clock input
- | The outputs of the circuit-under-test must be compared to the known good response
- | In general, collecting each output response and off-loading it from the CUT for comparison is too inefficient to be practical
- | The general solution is to compress the entire output stream into a single *signature* value
- | Signature Analysis is a compression technique based on the concept of *cyclic redundancy checking (CRC)*
 - m The simplest form of this technique is based on a single input LFSR

Copyright © 1995-1999 SCRA

111

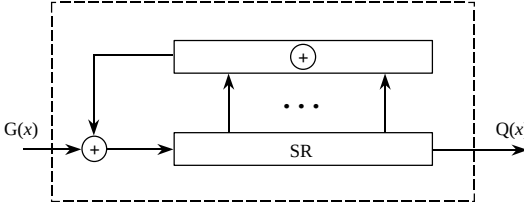
Once the test patterns are automatically applied, the responses must be gathered. The only way to do this practically for true BIST is to compress the responses into a single (we hope) unique value. Signature analysis attempts to perform this function.



DARPA • Tri-Service

Signature Analysis (Cont.)





Initial state: $I(x) = 0$
Final State: $R(x)$

$$\frac{G(x)}{P(x)} = Q(x) + \frac{R(x)}{P(x)}$$

or

$$G(x) = P(x) Q(x) + R(x)$$

number of bit streams that produce a specific signature: $\frac{2^m}{2^n} = 2^{m-n}$ m - bits in input stream
n - bits in signature register

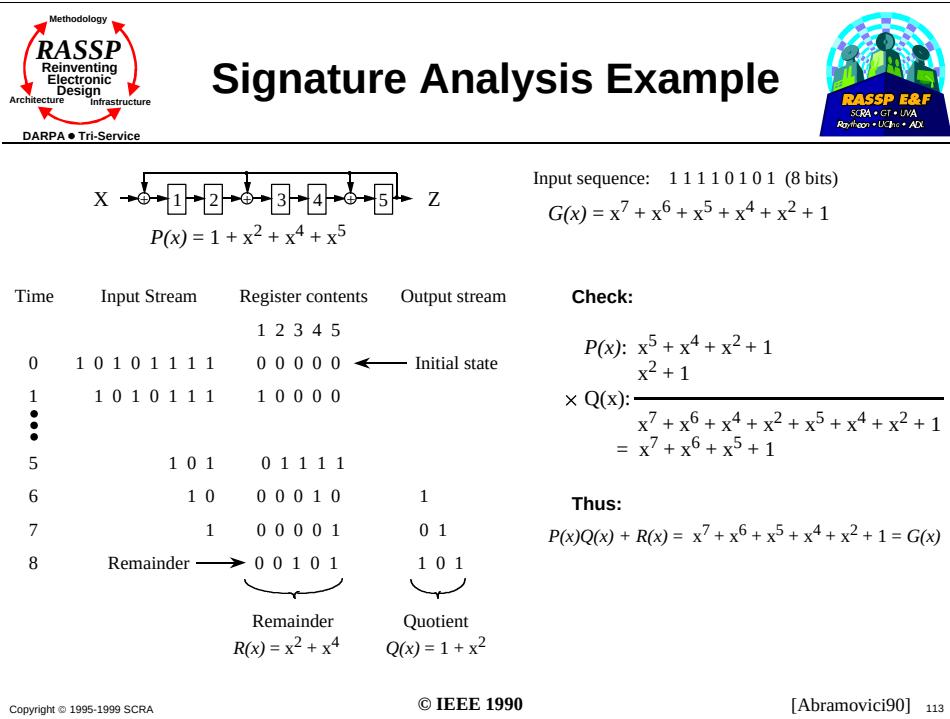
number of erroneous bits streams that produce the same signature as a particular fault-free response: $2^{m-n} - 1$

total proportion of masking error streams: $P_{SA}(M | m, n) = \frac{2^{m-n} - 1}{2^m - 1} \approx 2^{-n}$ for $m \gg n$

Copyright © 1995-1999 SCRA
© IEEE 1990
[Abramovici90] 112

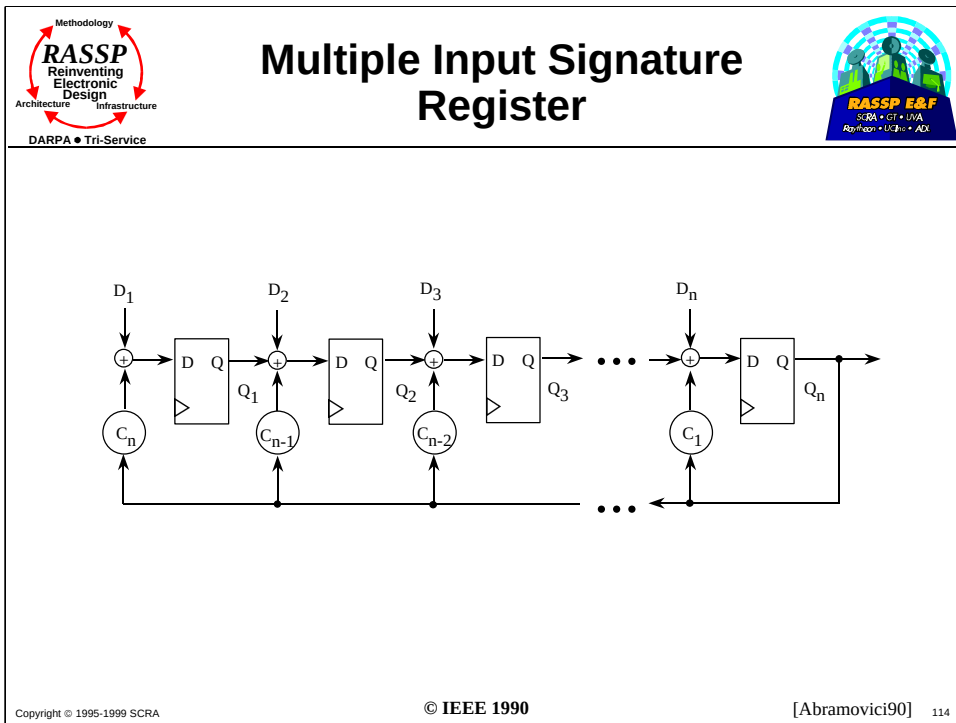
Signature analysis uses an LFSR to compress the input stream to a single value. The basic principal is that the input polynomial (stream) gets divided by the characteristic polynomial of the LFSR, resulting in a quotient (output stream) and a remainder. Because this is basically a “lossy” compression scheme, there is more than one input stream that can generate a specific signature. The occurrence of an erroneous input stream that generates a correct signature is called aliasing. The probability of aliasing as show here is very small, but it is also circuit dependent; i.e., the types of errors generated by faults in the circuit may make aliasing more probable.

Ref: [Abramovici90]

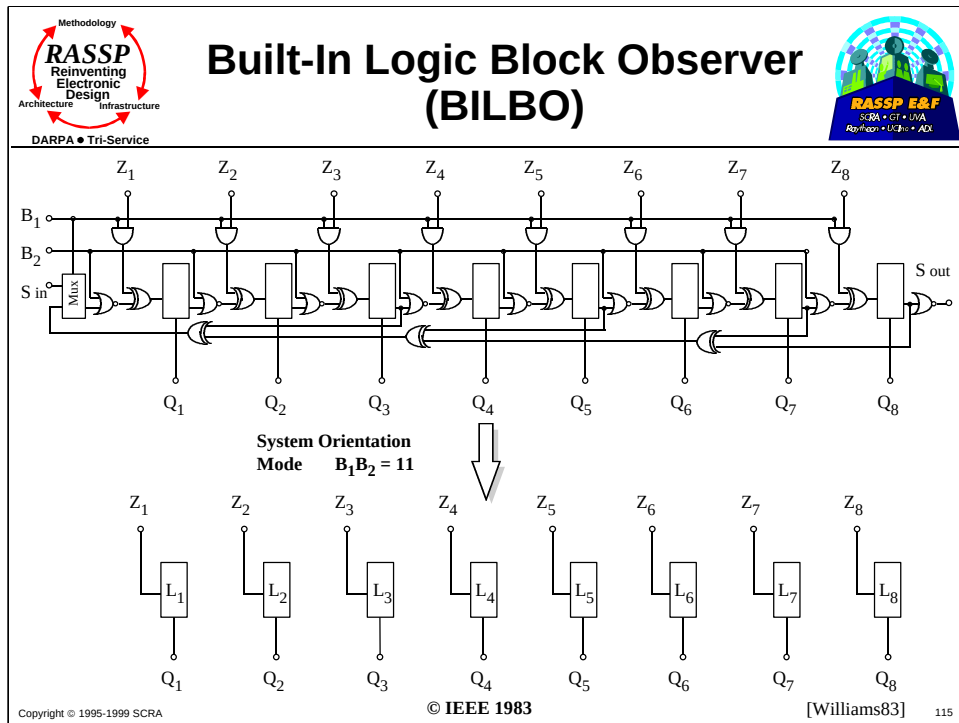


This figure show an example of the hardware implementation of signature analysis and a mathematical check of the result. The audience is referred to the references for a more detailed presentation of the theory.

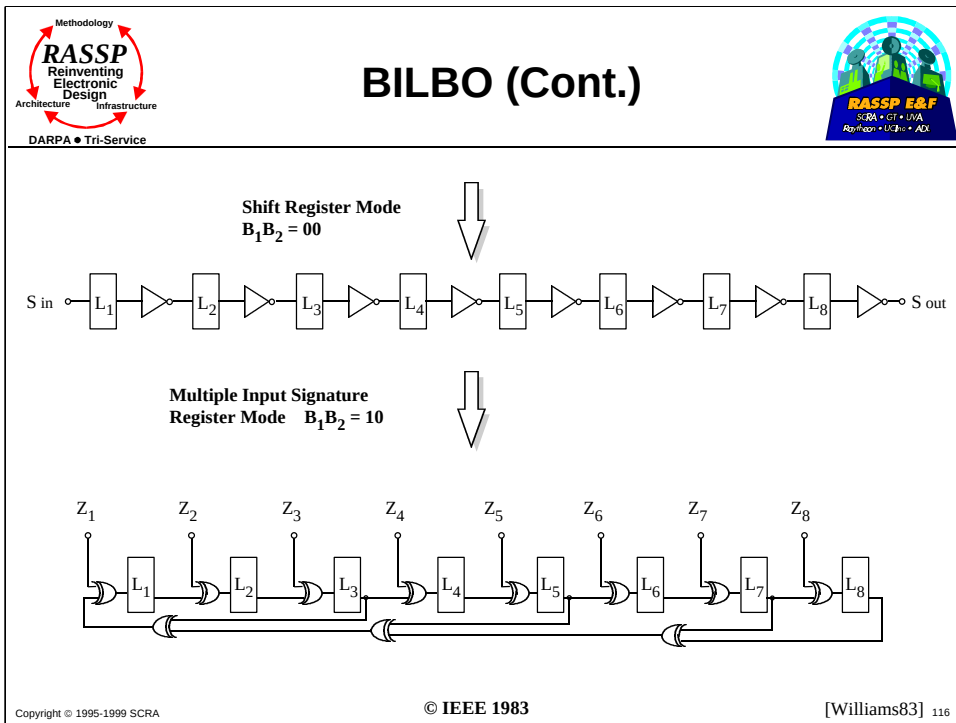
Ref: [Abramovici90]



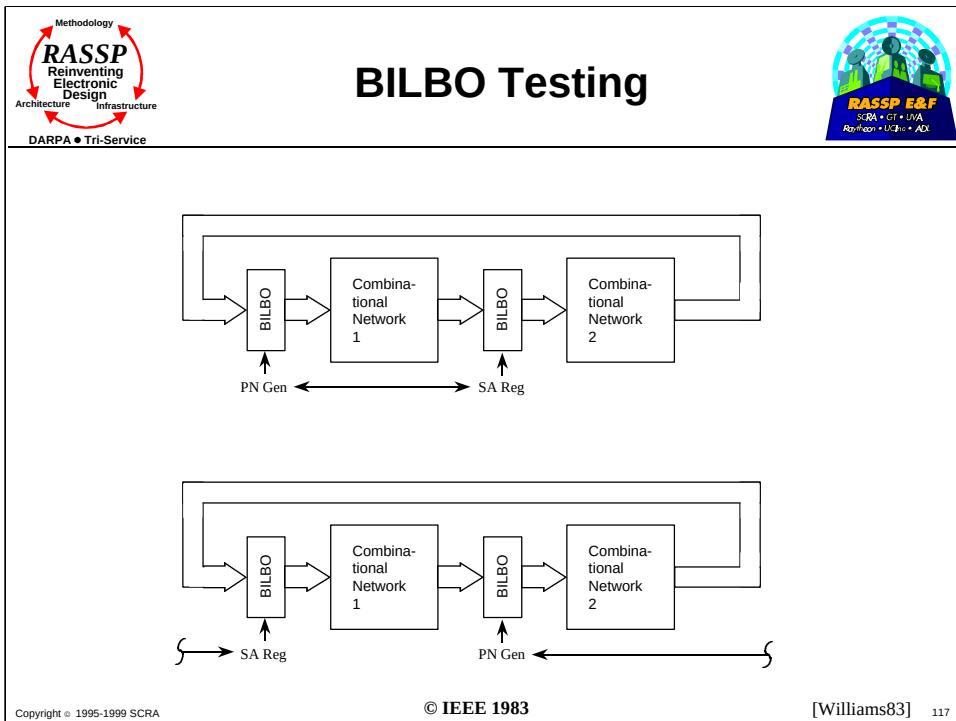
In order to reduce the amount of hardware required to compress a multiple bit stream, a multiple input signature analysis register can be used. The theory presented in the literature shows that the functionality in terms of aliasing probability is unchanged for this implementation.



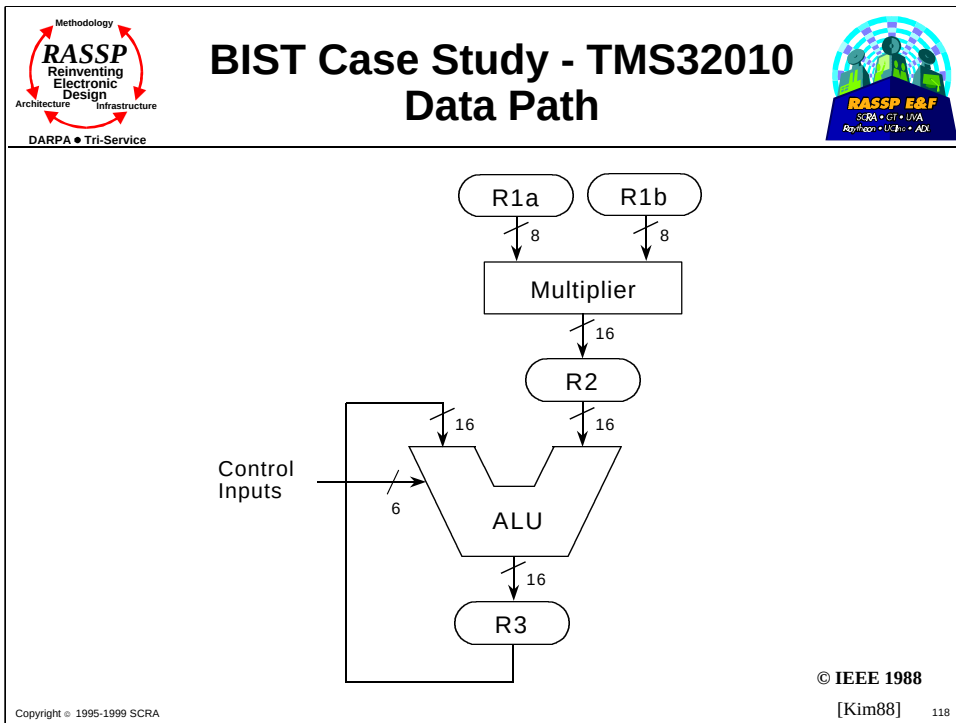
This carefully drawn figure details the implementation of a Built-In Logic Block Observer (BILBO). A BILBO is not a small fictional troll, but a logic block that can function as a normal state register, a scan register, a PseudoRandom Pattern Generator (PRPG), or a Multiple Input Signature Register (MISR), depending on the state of the mode inputs.



This figure show the shift register and the MISR modes. Note that, in the MISR mode, if the Z inputs are held constant at “0”, the BILBO functions as a PRPG.

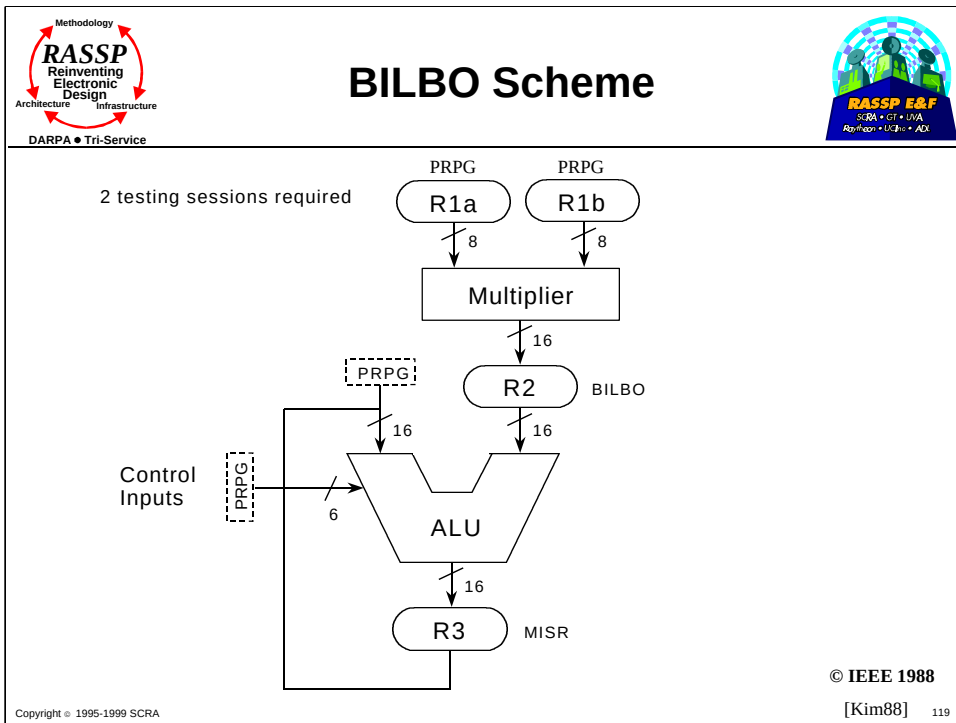


These figures show how replacing the two registers in this design with BILBOs will facilitate testing. To test each combinational device, the BILBO on the inputs is set to MISR mode with constant “0”s on the inputs (I'm not sure how) to function as a PRPG. The BILBO on the outputs is put into a MISR mode to function as a signature analyzer. This testing requires two testing “sessions”, one for each combinational block.

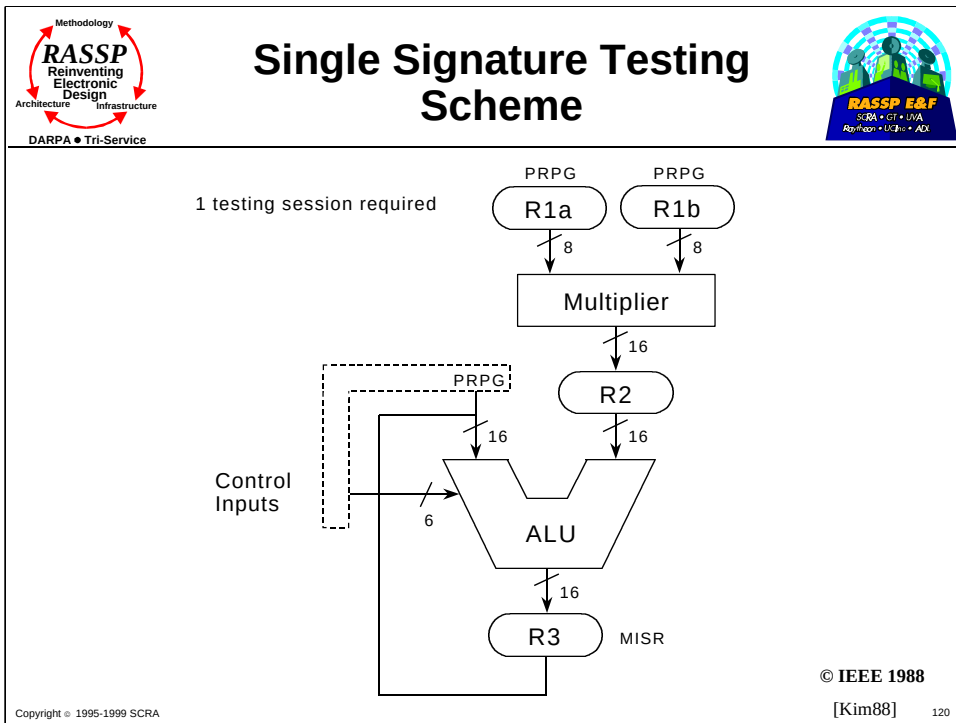


This is a case study in the literature which describes various configuration of BIST for a section of the TMS32010 data path shown here.

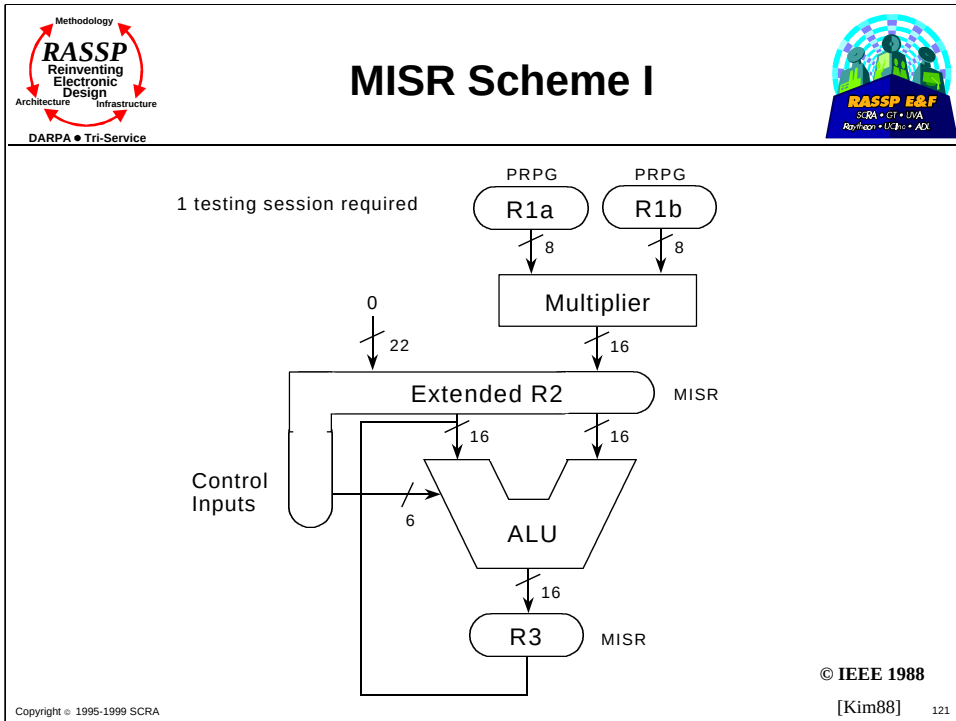
Ref: [Kim88]



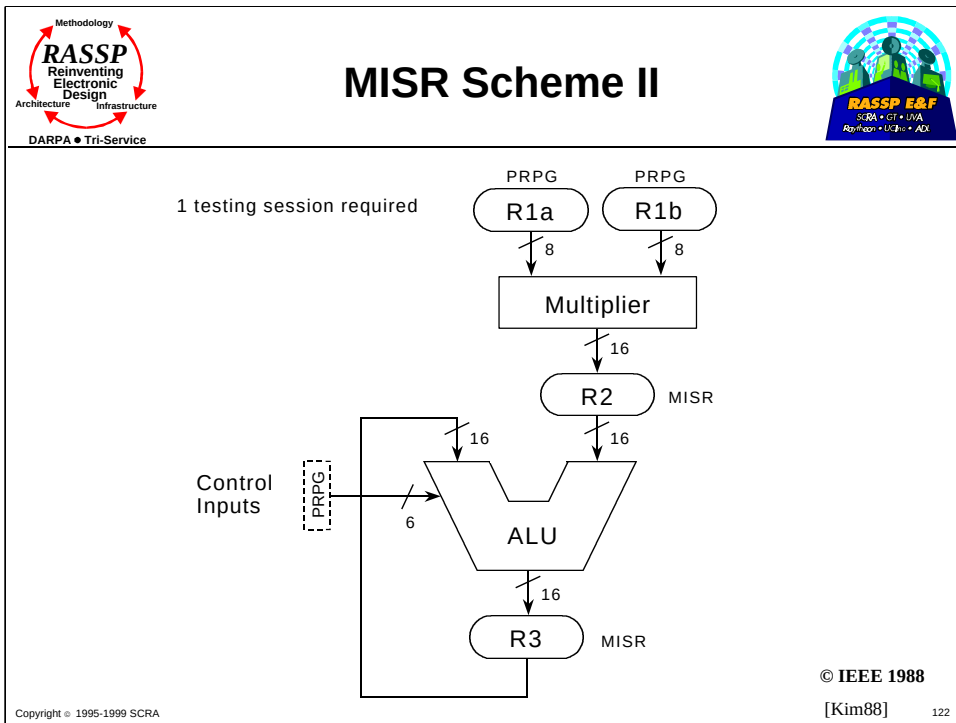
This figure shows the normal BILBO scheme where R1a, R1b, R2, and R3 are replaced with BILBO registers. Also, a 6 bit PRPG is required for the ALU mode bits, and a 16 bit PRPG is required for the leftmost ALU bits. Two testing sessions are required, one for the multiplier and one for the ALU.



In this scheme, R2 is unmodified and the outputs of the Multiplier during testing are used as partial test inputs to the ALU for testing. Again, a 6 and 16 bit PRPG is required, but R2 is not a BILBO. Only one testing session is required.



In this scheme, R2 is extended to a 38 bit MISR and the first 22 bits are held constant at "0" during testing. Only one test session is required.



In this scheme, R2 is again an MISR, but it is only 16 bits wide. The output of the R3 MISR is fed back around to the leftmost bits of the ALU. Only an additional 6 bit PRPG for the mode bits is required. This scheme uses the least amount of hardware.



Test Case Results



No. of test patterns	BILBO scheme	Single signature testing	MISR scheme I	MISR scheme II
Average	2,177	> 3,000	1,457	1,378
Minimum	830	-	634	721
Maximum	3,619	-	2,531	2,121
Fault coverage (%)	100	64.5	100	100

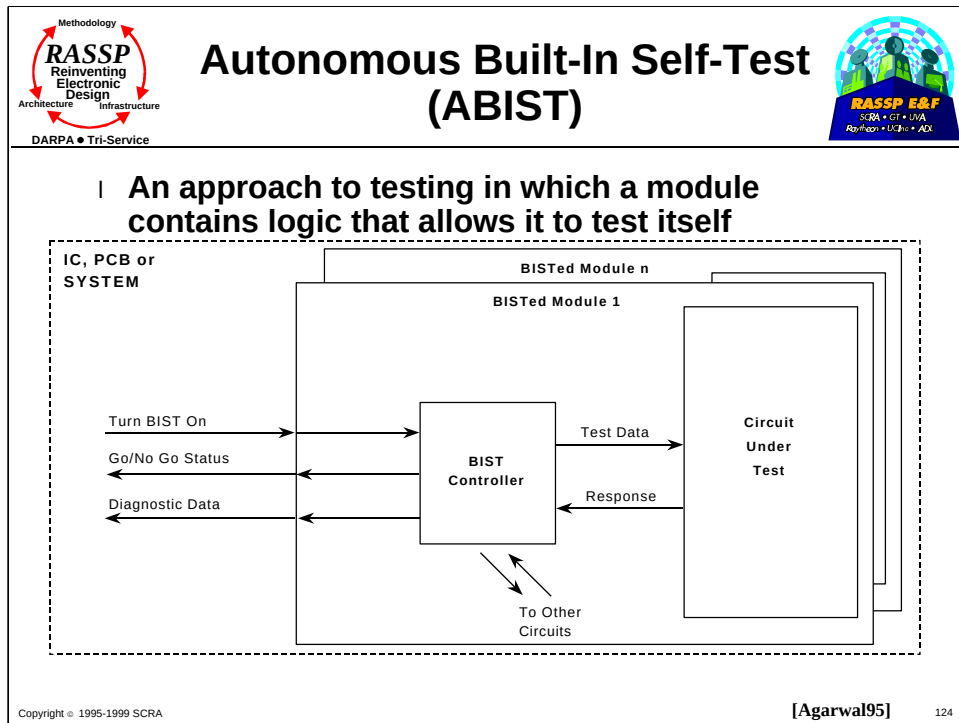
Copyright © 1995-1999 SCRA

© IEEE 1988

[Kim88] 123

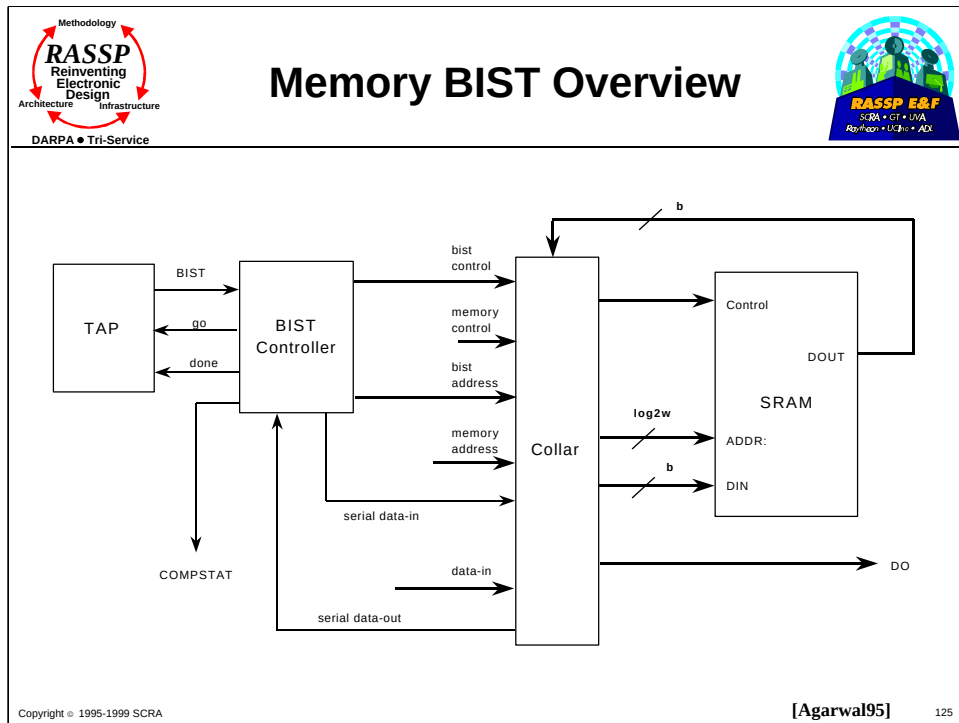
This table show the results of the various schemes. Twenty different runs with different seeds for the PSRGs were done. For designs 1, 3, and 4, the simulation was stopped when fault coverage reached 100%. For design 2, fault coverage saturated at 64.5%.

Note that design 4 produced equivalent results for number of test vectors, but required less hardware.



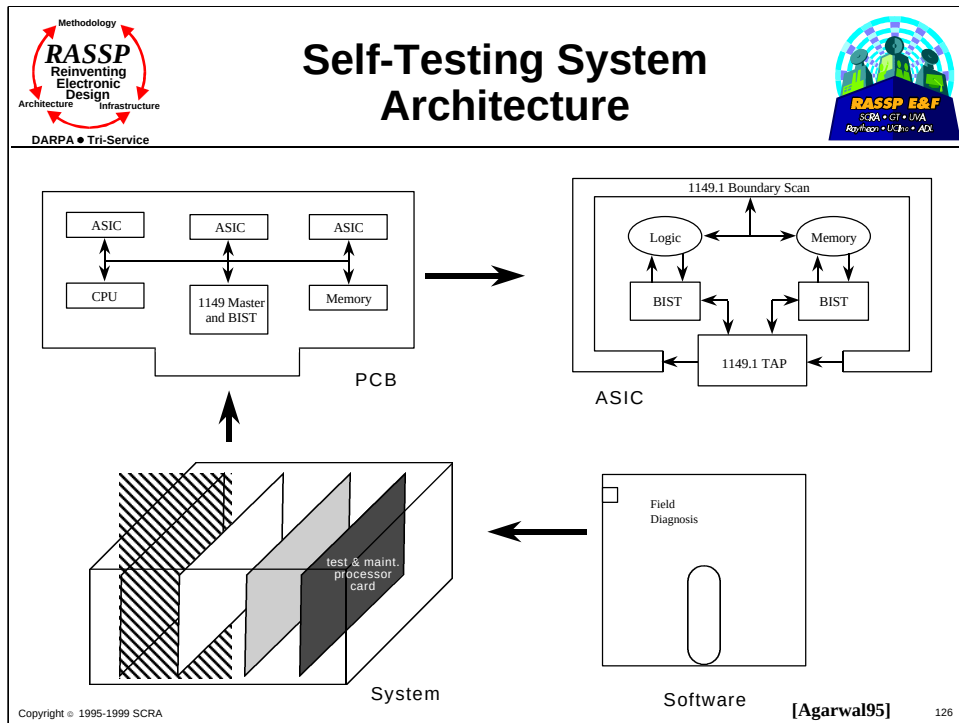
A novel concept being developed for RASSP by LogicVision is Autonomous Built-In Self-Test. In ABIST, normal BIST techniques are built into the circuit-under-test, but a controller is added that will completely control that BIST hardware to configure it as required and run the required test sessions. It also compares the resulting response(s) to provide a single go/no go output back to the next level of test hardware.

Ref: [Agarwal95]

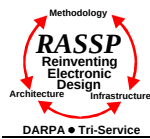


This figure shows the configuration of an ABIST SRAM developed by LogicVision. The collar is a parameterizable part that can be configured for various sizes of SRAMs. It contains the shift registers necessary to apply the test data to the SRAM core.

LogicVision has a tool, ICRAMBIST, that automatically generates VHDL or Verilog descriptions of these parts.



This figure illustrates the necessity of building in testability at all levels, from chip, to board, to system, to diagnostic software.



Generations of Test Solutions



I Completely External Testing:

- m **Generation 1: Functional verification vectors used as manufacturing test vectors (fault simulate & supplement with additional functional vectors to raise fault coverage)**
- m **Generation 2: Scan insertion and Automatic Test Pattern Generation**

I Electronic Systems Test Automation (ESTA)

- m **Generation 3: Automatic Built-In Self-Test (ABIST) using 1149.X**
- m **Generation 4: Hierarchical and integrated BIST (HIBIST) from chip to system**

Copyright © 1995-1999 SCRA

[Agarwal95]

127

As previously stated, the current state of the art in testing is generally either fault simulation of functional vectors (and the adding some) or structured DFT (LSSD) with ATPG. LogicVision and others are trying to push the state of the art of testing into the next step, Electronic Systems Test Automation, where testability and test are built-in from the start.



Benefits of ESTA



- | **Reduces time to market during design and manufacturing**
 - m Production test development time can be reduced from weeks to days
 - m Simplifies and manages the test development process
 - m Handles very complex designs
 - m Compatible with HDL and synthesis-based design flow
- | **Reduces cost over full product test lifecycle**
 - m Reduces tester capital costs
 - m Spans the test hierarchy from IC to MCM/PCB to system
 - m Allows test data to be manipulated for reuse at higher levels
 - m Permits field diagnosis

Copyright © 1995-1999 SCRA

[Agarwal95]

128

There are major benefits to the incorporation of ESTA, some of which are stated here...



Benefits of ESTA (Cont.)



| Enhances quality

- m Permits very high fault coverage tests
- m Supports at-speed test
- m Tests embedded or hard-to-reach functionality
- m Supports validation of test structures and data

And the rest of which are stated here (note functional testing, too!).



Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Built-In Self Test
- | **Hierarchical Design for Test**
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | Summary

Copyright © 1995-1999 SCRA

130



Hierarchical Design For Test



- | **Testability must be considered at all levels of the design**
 - m ASIC/FPGA
 - m MCM
 - m Board
 - m System
- | **System-level test requirements must be propagated down to the lower level of the design to exploit design options at these levels**
- | **Additional DFT features could be added to an ASIC to assist in testing other chips connected to it**
- | **The alternative is to rigidly specify DFT rules for each level of the design**
 - m **Does not support COTS use**

Copyright © 1995-1999 SCRA [MABadir94] 131

Hierarchical Design for Test requires that testing be considered at all levels of the design process.

For example, if during system design a COTS processor that doesn't include boundary scan is specified, boundary scan can be added/required on all interface chips around it to effectively make the processor boundary scan testable.

If rigid structured DFT is used from the ground up, the problem is easier; but that doesn't support COTS parts.

Ref: [Abadir94]

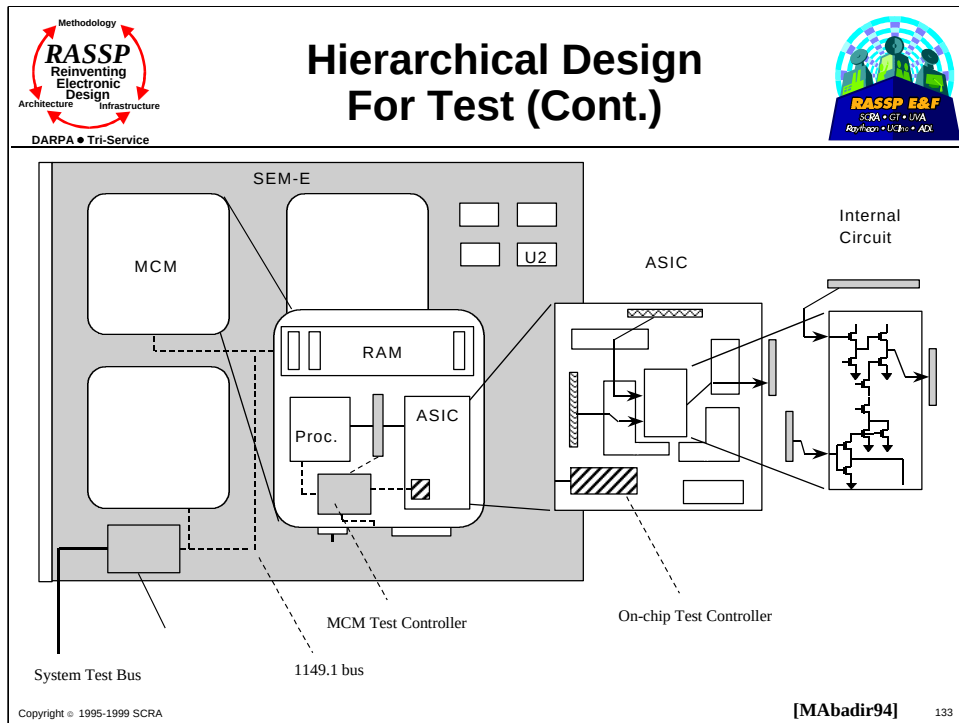


Hierarchical Design For Test (Cont.)



- | **Testability requirements must be captured at an early stage of the design process**
- | **Standard interfaces should be used to simplify testing interconnects**
- | **A hierarchical test strategy has three main components:**
 - m **Partitioning of the design into independently testable units (test kernels)**
 - m **Specification of test methods for each test kernel**
 - q **Full scan, functional testing, boundary scan**
 - m **Setting of targets for trade-off parameters**
 - q **Fault coverage, test application time, test cost**

Some further points about HDFT are stated here.



This slide simply illustrates the concept of HDFT at a system (SEM-E) level. Note that not all parts must have DFT, but if requirements are global, other parts can have additional testing functionality added to meet the global requirements.



Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Built-In Self Test
- | Hierarchical Design for Test
- | **Synthesis for Test**
- | DFT Standards
- | Design Flows with DFT
- | Summary

Copyright © 1995-1999 SCRA

134



Synthesis for Test



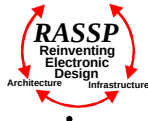
- | **Synthesis for test is:**
 - m **Synthesizing inherently testable circuits**
 - m **Incorporating BIST circuitry into a behaviorally synthesized circuit**
 - m **Constraining logic synthesis to conform to structured DFT techniques like scan**
 - m **Developing a complete test program for a synthesized circuit**
- | **The definition depends on the user and on the level of synthesis supported**
 - m **Gate-level test synthesis**
 - m **RTL test synthesis**

Copyright © 1995-1999 SCRA

135

The definition of synthesis for test depends on the user, but for commercial tools, it can be broken down into two broad categories; gate level, and RTL synthesis for test.

Ref [Aitken95]



Elements of Synthesis for Test



- | **Methodology selection** - selection of the overall set of rules, structures, and design constraints to be used in including testability in the design process
- | **Test structure insertion** - the actual insertion of the selected test structures into the description of the circuit-under-test
- | **Circuit verification** - insuring that the test circuitry and the circuit-under-test operate as intended during test as well as normal operation
- | **Program preparation** - generation of the actual test programs used during manufacture - generation of test patterns, construction of scan strings, etc.

Copyright ©

Ref [Aitken95]



Gate-Level Synthesis for Test



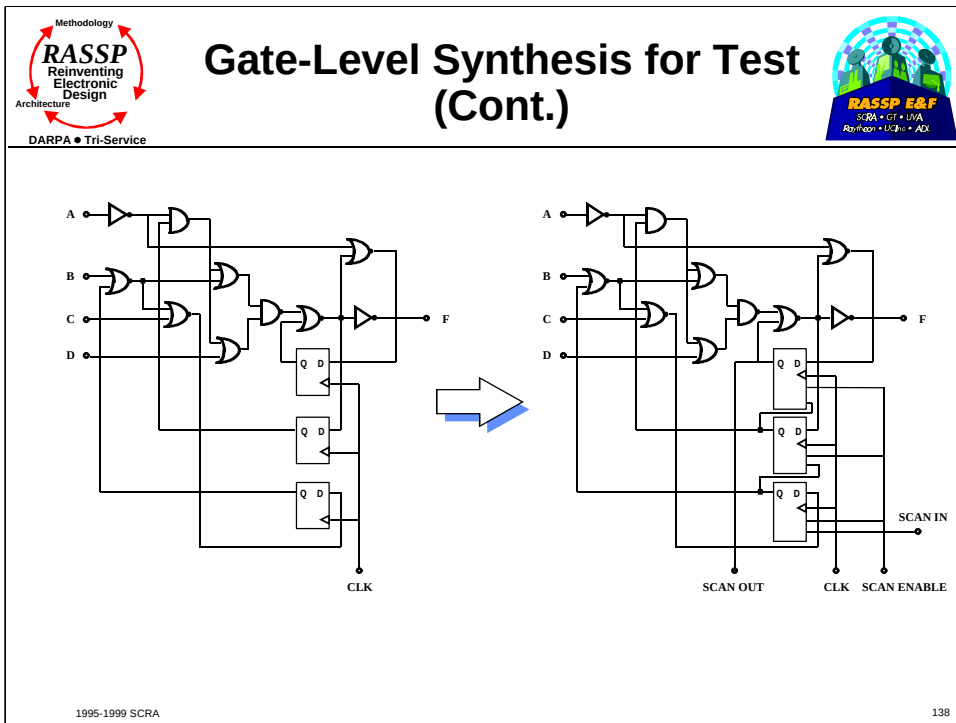
- | **Most mature technology - commercially available**
- | **Five separate activities:**
 - m **Methodology selection - usually a limited number of fixed DFT method available**
 - m **Automatic test structure insertion**
 - q **Select which flip-flops belong in which scan chain(s)**
 - q **Connect successive elements in the chain(s)**
 - q **Connect global signals**
 - m **DFT rule checking - check for design techniques which could cause a problem with the selected methodology**
 - m **Automatic test pattern generation - generate tests for the circuit-under-test given the DFT method and fault model**
 - m **Test vector translation - reformatting test vectors for specific ATE equipment used**

1995-1999 SCRA

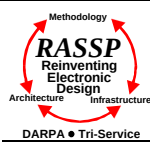
137

Gate level synthesis for test is fairly straight forward in that it consists of selection of a testability measure (usually some form of scan full or partial), and then automatic insertion and test pattern generation and test program construction (building the scan chains).

Ref [Aitken95]



This slide shows the output that would be typical of a gate level synthesis process for a state machine without and with synthesis for with full-scan as the testability methodology and the tool will insert the scan flip-flops and connect the scan chain.



RTL Synthesis for Test



- | **Tools less mature from a commercial standpoint**
- | **Activities:**
 - m **Methodology selection** - selection of a test methodology for a specific module(s)
 - m **Automatic test structure insertion** - insertion of RTL statements necessary to generate DFT circuitry
 - m **Test structure verification** - tools may be available to actually fix DFT violations
 - m **Test program preparation** - conversion of RTL level test vectors to gate level compatible vectors may be difficult

Copyright © 1995-1999 SCRA

139

Commercial tools for RTL synthesis for test are much less mature although University tools are more mature.

In RTL synthesis, methodology selection may be more flexible. A larger number of potential methodologies may be available and different methodologies may be used on separate modules in the same circuit.

Automatic test structure insertion consists of adding the necessary statements to the RTL description so that the test structures are synthesized.

In RTL test synthesis, the test structure verification tools may be able to not only check for untestable structures such as gated clocks, but may in fact, be able to fix them.

Finally, test program construction may not be available for RTL descriptions because of the difficulty in translating the high level stimulus into gate level test vectors.

University tools for RTL synthesis for test concentrate on generating testable structures. This involves construction of a register adjacency graph and then changing it to make the structure more testable. This is done by minimizing the sequential depth between input and output registers, maximizing the input and output registers) and minimizing the number of cycles.



Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Built-In Self Test
- | Hierarchical Design for Test
- | Synthesis for Test
- | **DFT Standards**
- | Design Flows with DFT
- | Summary

Copyright © 1995-1999 SCRA

140



Section Outline



| DFT Standards

- m IEEE Std. 1149.1
- m IEEE Std. 1149.1b
- m IEEE Std. 1149.5
- m IEEE Std. 1029.1
- m MIL-HDBK-XX47



DFT Standards



- | **IEEE Std. 1149.1 - Test Access Port and Boundary-Scan Architecture**
 - m Defines the architecture of the TAP and Boundary Scan cells
 - m IEEE 1149.1b - defines the Boundary-Scan Description Language (BSDL)
- | **IEEE Std. P1149.2 - Extended Serial-Digital Interface Standard**
 - m Defines a scheme that supports board-level interconnect testing and internal-scan testing of components
- | **IEEE Std. P1149.3 - Real Time Test Bus Standard**
 - m Proposed to define standards for real-time testability bus (work discontinued)

Copyright © 1995-1999 SCRA

142

This slide lists some for the approved/proposed IEEE testing standards. Note that P1149.3 is currently inactive.



DFT Standards (Cont.)



- | **IEEE Std. P1149.4 - Mixed-Signal Test Bus Standard**
 - m Proposed to extend the concept of boundary-scan to analog and mixed signal devices
- | **IEEE Std. P1149.5 - Module Test and Maintenance Bus Standard**
 - m Defines specifications for a serial test and maintenance bus for systems with two or more modules plugged into a backplane
- | **IEEE Std. 1029.1 - Waveform and Vector Exchange Specification (WAVES)**
 - m Defines standard for VHDL description of stimulus vectors and responses
- | **MIL-HDBK-XX47 Testability Analysis Handbook**
 - m Defined the DoD view of Design for Test

Copyright © 1995-1999 SCRA

143

This slide lists some for the approved/proposed IEEE testing standards. MIL-HDBK-XX47 is a preliminary document that provides information on the DoD view of DFT.



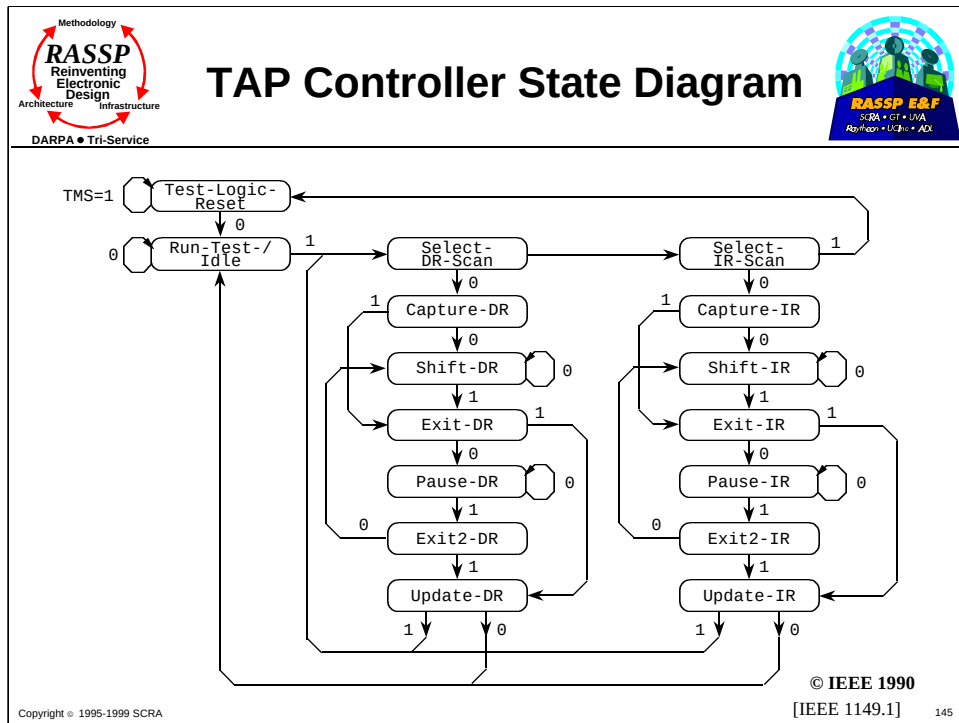
IEEE Std. 1149.1 Standard Test Access Port and Boundary-Scan Architecture



- | Defines standard terms
- | Defines the boundary-scan cell architecture, TAP controller architecture, and board-under-test architecture described previously
- | Defines the test modes described previously
- | Defines alternative cell and board architectures
- | Defines the actual state diagram for the TAP controller
- | Defines timing relationships for controller states and TAP port events
- | Defines the functionality of the Instruction, Test Data, and Bypass registers

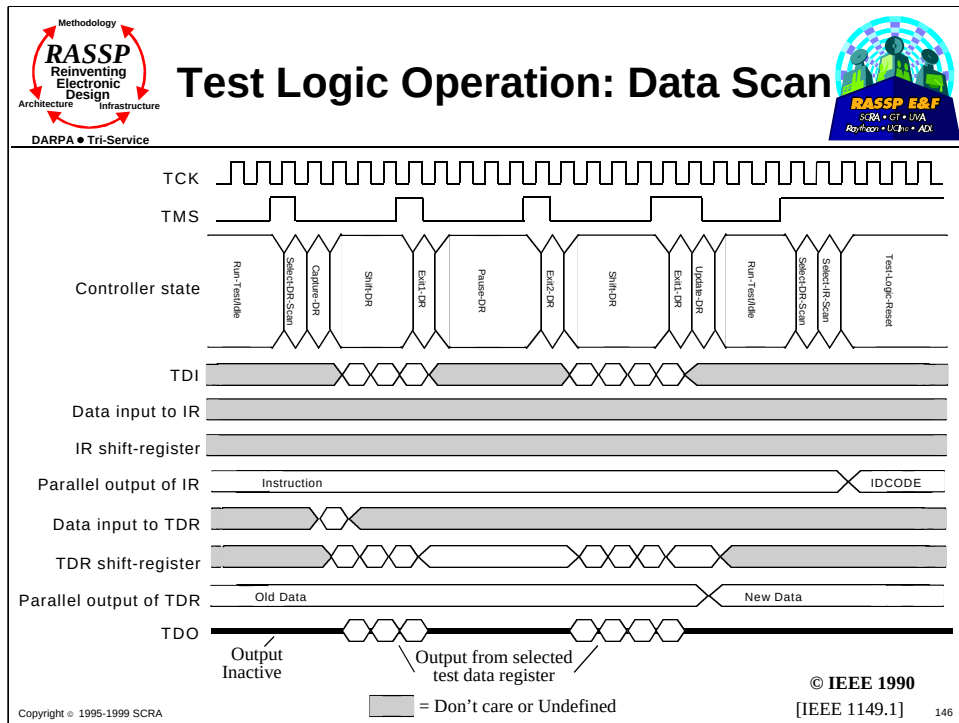
Copyright © 1995-1999 SCRA144

Here are listed some of the major contents of the IEEE 1149.1 standard. The architecture described previously for the boundary-scan cells/architecture are defined in this standard.



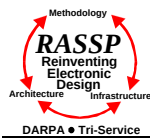
Additional details such as the TAP controller state diagram are defined in the standard to insure interoperability between parts that are designed to the standard.

Ref: [IEEE1149.1]



Timing diagrams such as this one are also defined to ensure interoperability.

Ref: [IEEE 1149.1]



IEEE Std. 1149.1b BSDL



- | **BSDL is intended to provide a machine-readable means of representing some parts of the documentary information specified in 1149.1**
- | **The goal of the language is to facilitate communication between companies, individuals, and tools that need to exchange information on the design of test logic; for example:**
- | **Suppliers can supply BSDL descriptions of components that support 1149.1 to purchasers**
 - m **ATPG tools could use BSDL description of components on a board to generate test vectors**
 - m **BSDL descriptions of 1149.1-compliant components could be used for synthesis**

Copyright © 1995-1999 SCRA

147

The 1149.1b standard is an appendix to 1149.1 that defines the Boundary-Scan Description Language. BSDL is designed to standardize the way boundary-scan architectures are described.

IEEE Std. 1149.1b BSDL (Cont.)

- | **BSDL is a subset of IEEE Std 1076-1993 VHDL**
- | **A standard VHDL package STD_1149_1_1994 is defined**
 - m **Attributes**
 - q **TAP Ports**
 - q **TAP Instructions**
 - m **Types**
 - q **Cell Data**
 - q **ID Codes**
 - m **Constants**
 - q **Cell Types**

BSDL is basically a set of standard VHDL attributes which are defined in a package. Instances of these attributes are associated with a VHDL entity of a boundary-scan-compatible part to describe the architecture of the boundary-scan components. This includes the scan cells, TAP, test instructions available, etc.



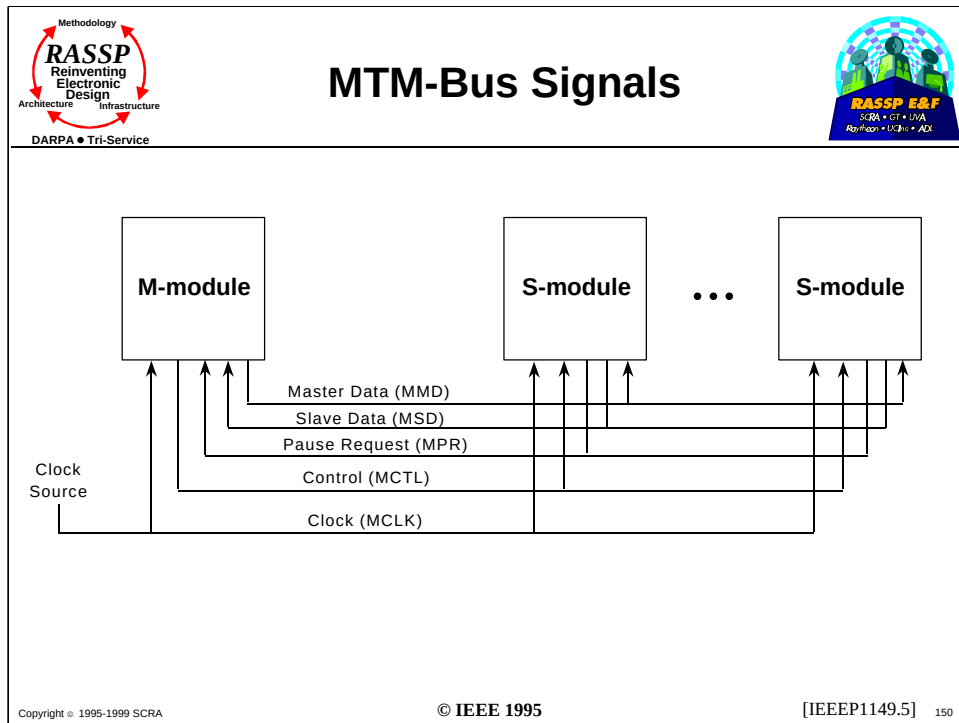
IEEE Std. P1149.5 Standard Module Test and Maintenance (MTM) Bus Protocol



- | **Defines a standardized serial, backplane Test and Maintenance bus**
- | **Intended for use in the test, diagnosis, and maintenance of electronic subsystems and modules**
 - m **Module Test**
 - m **Subsystem Test**
 - m **Subsystem Diagnosis**
 - m **Software/Hardware Development**
- | **Defines MTM-Bus architecture**
- | **Defines MTM-Bus protocol**
 - m **Physical layer**
 - m **Link layer**
 - m **Message layer**

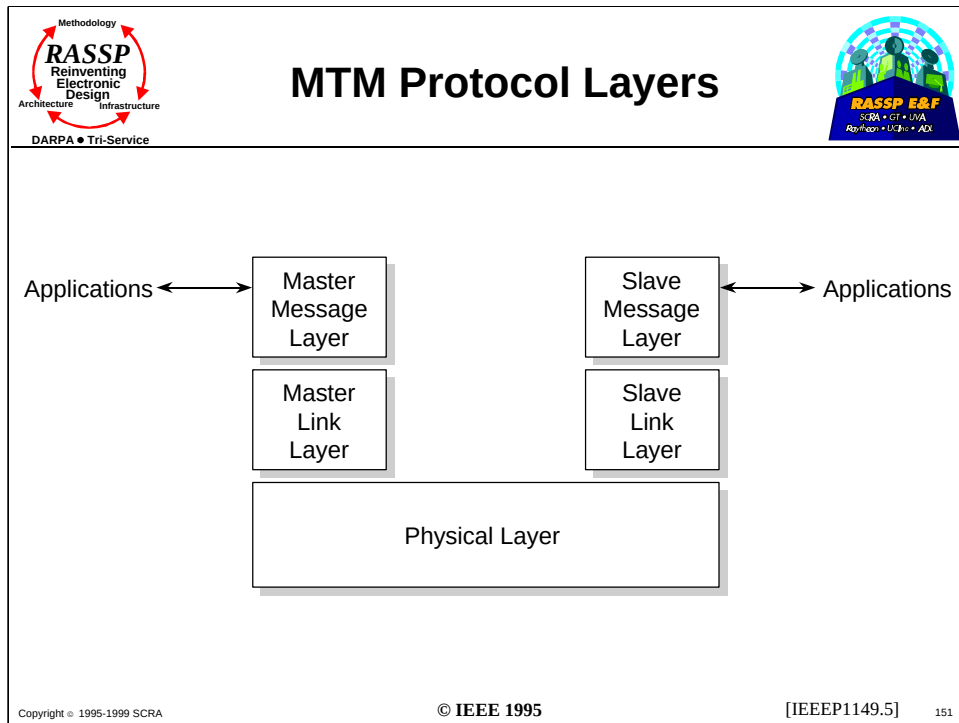
Copyright © 1995-1999 SCRA149

The IEEE P1149.5 MTM standard defines test interoperability at the next higher level from 1149.1.



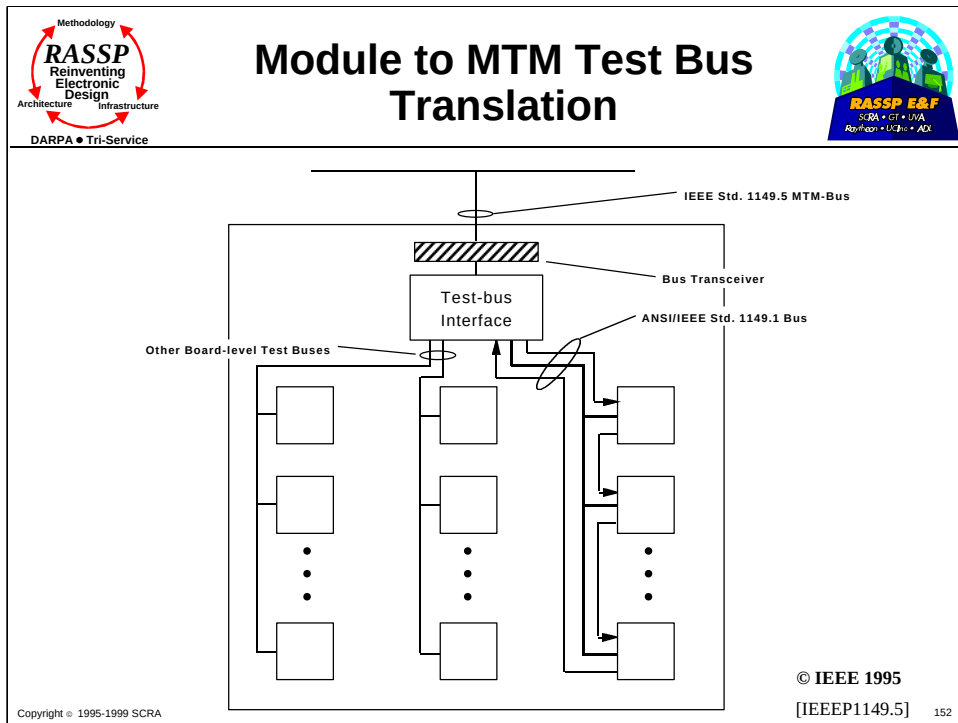
This figure shows the MTM bus and how it is used to connect testable systems together so that they can transmit and receive test data over the same bus.

Ref: [IEEEP1149.5]



The standard also defines a fairly detailed protocol for transmitting messages over the MTM bus. Different messages lengths and instructions are also defined.

Ref: [IEEEP1149.5]



A bus transceiver is required to move data from the MTM bus to the 1149.1 bus and back as well as to talk to other non-standard test busses.

Ref: [IEEEP1149.5]



IEEE Std. 1029.1 Waveform and Vector Exchange Specification (WAVES)



- | **A language for specifying stimulus and response patterns for digital electronic systems**
- | **Provides for unambiguously exchanging such patterns between and among design and test environments**
- | **Can represent various levels of waveform detail**
 - m **Simulator event trace data**
 - m **Highly structured test vectors typical of automatic test equipment**
- | **WAVES is a subset of IEEE Standard 1076-1987 VHDL**
- | **WAVES declarations and procedures declared in separate package - instantiated with circuit-under-test in test bench**

Copyright © 1995-1999 SCRA
153

WAVES is a subset of VHDL that is intended as a means for unambiguously specifying waveforms for digital systems - both stimulus and response.

WAVES can represent signals a various levels of abstraction, from simple lists of simulator outputs, to complex descriptions suitable for use in ATE equipment.

WAVES consists mainly of separate packages that are instantiated in a test bench for a device-under-test.

IEEE Std. 1029.1 WAVES (Cont.)

I **WAVES Package:**

m **Required Declarations**

- q **Logic Value** - values to be applied directly to DUT
- q **Pin Codes** - “input” values used to list waveform
- q **Test Pins** - list of pins in the waveform

m **Waveform Generator Procedure** - reads “input values” and applies them to the DUT

- q **Values can be build in to the generator procedure**
 - ı **Simple list**
 - ı **Generator algorithm**
- q **Values can be read from an external file**

There are standard WAVES packages that define procedures like apply. The user defined WAVES package (which uses the predefined packages) consists of required definitions and the waveform generator procedure.

The required definitions include the logic value to be applied to the DUT, the pin codes, which is the values for the pins defined by the DUT description, and the actual list of pins of the DUT.

The waveform generator procedure actually applies the values to the DUT. The values can be included in the waveform generator procedure in the form of VHDL code, or read from an external file. If included in the code, the waveform can be simply listed, or can be described algorithmically (as is typically done in descriptions used for ATE equipment).



MIL-HDBK-XX47 Testability Analysis Handbook



- I **Outlines the DoD documents that relate to DFT**
 - m **MIL-STD-2165A, Testability Program for Systems and Equipment**
 - m **MIL-STD-499A, Engineering Management**
 - m **MIL-STD-470A, Maintainability Program Requirements for Systems and Equipments**
 - m **MIL-STD-1388-1A, Logistics Support Analysis**
 - m **MIL-STD-785, Reliability Program for Systems and Equipments Development and Production**
 - m **MIL-HDBK-59, Department of Defense Computer-Aided Acquisition and Logistics Support (CALS) Program Implementation Guide**
 - m **MIL-STD-1814, Integrated Diagnostics, Roadmap, and Requirements Document**

MIL-HDBK-XX47 outlines the DoD documents that outline the process and product of Design for Test for Military systems. Some detail is included to show how these documents relate to the process and each other.



MIL-HDBK-XX47 (Cont.)



I Test Design and Assessment

- m Incorporate embedded and external test capability for a system or equipment that will satisfy testability performance requirements
- m Assess the level of test effectiveness that will be achieved for a system or equipment
- m Ensure the effective integration and compatibility of this test capability with other diagnostic elements

I Outputs

- m Product fault detection and fault isolation performance levels that achieve the specified test effectiveness requirements

The second “half” of the MIL-HDBK-XX47 handbook details the DFT design flow and techniques that is defined by the appropriate documents or that adheres to these documents. Some anecdotal information is included.



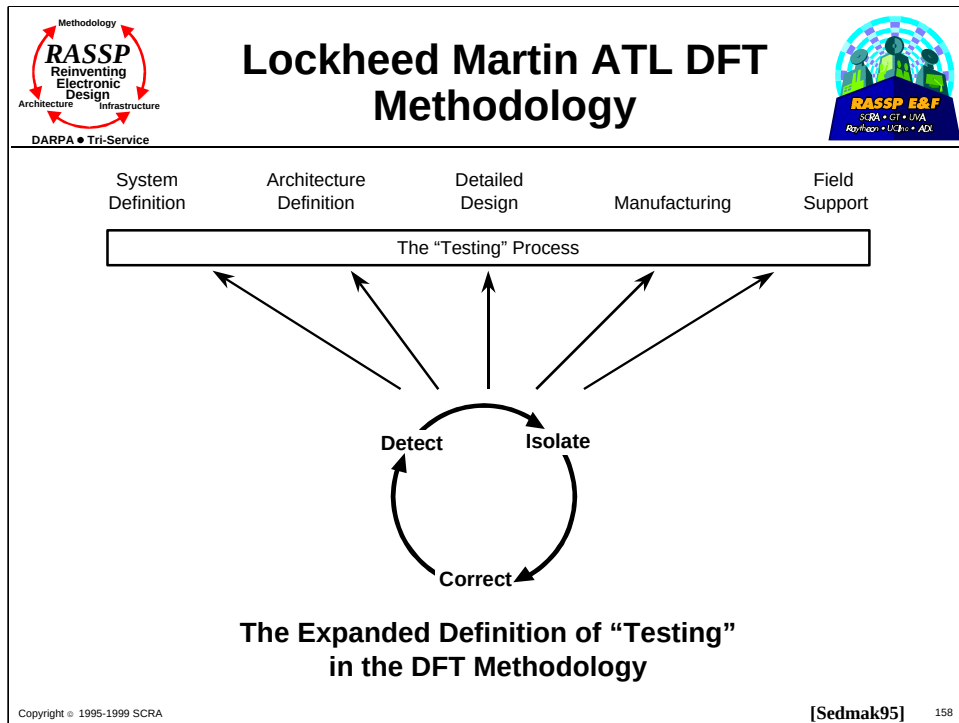
Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Built-In Self Test
- | Hierarchical Design for Test
- | Synthesis for Test
- | DFT Standards
- | **Design Flows with DFT**
- | Summary

Copyright © 1995-1999 SCRA

157



This slide shows the basis of the LMC-ATL Test methodology. That is, testing is reduced to three basic processes, Detection, Isolation, and Correction, and they are applied all along the product life-cycle.

Ref [Sedmak95]



ATL DFT Methodology

Terms and Definitions



Prediction
The assessment of the degree of compliance to test requirements through analysis or comparison with similar library elements
 — Examples of means: Circuit level testability measures, topological dependency models

Verification
The assessment of the degree of compliance to test requirements through simulation or closed form proof
 — Examples of means: Deterministic fault simulation, exhaustive test coverage proofs

Measurement
The assessment of the degree of compliance to test requirements through monitoring test performance on a physical item under test
 Examples of means: Automatic fault history logging, ATE-based data collection

Copyright © 1995-1999 SCRA
[Sedmak95] 159

Closed form proofs are used when provable fault coverage is possible (ie. pseudoexhaustive testing)

Automatic fault history logging automatically registers which fault have been detected, isolated, or corrected at each step in the process.

Ref [Sedmak95]



ATL DFT Methodology

Additional Terms and Definitions



Metrics

Quantitative parameters used for requirements specification, compliance tracking, tradeoff analysis, and selection

TMAT

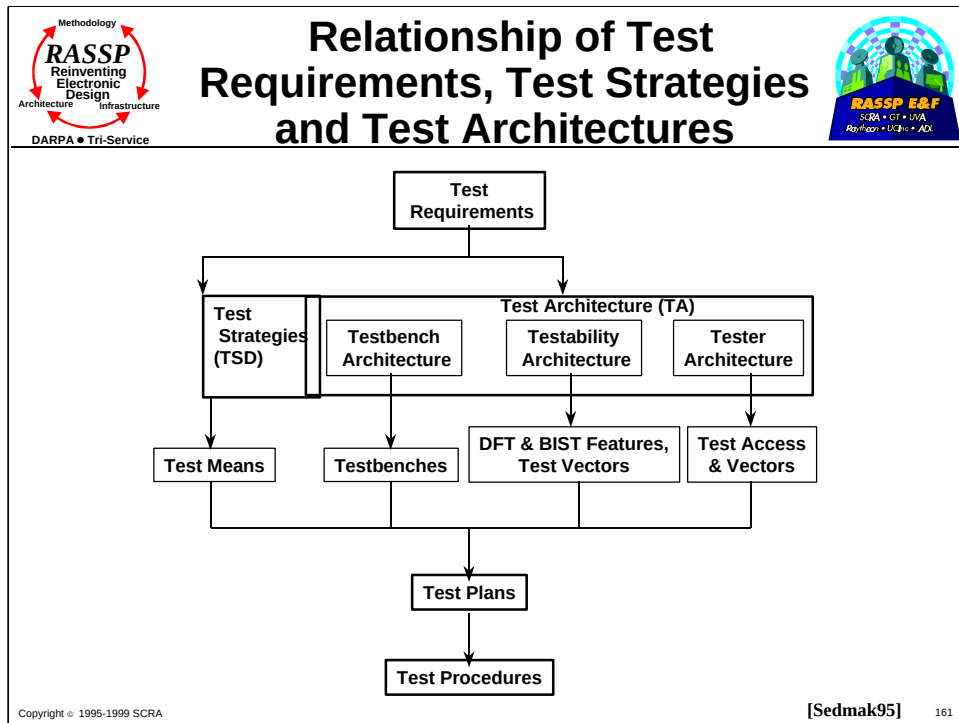
The Test Metrics/Tool Application Table, used in the Methodology to select a metric and tool(s), or method, to predict, verify, or measure compliance

Copyright © 1995-1999 SCRA

[Sedmak95] 160

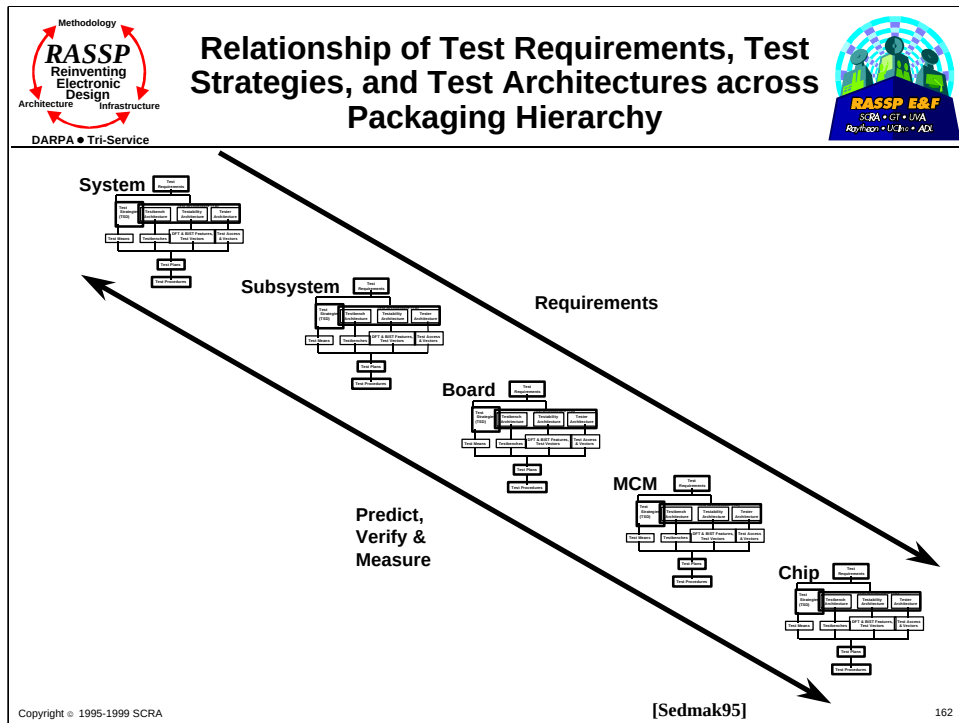
TMAT can be generic, type of tool is listed, not necessarily specific vendor.

Ref [Sedmak95]



Increasing levels of detail & development go down in this diagram.
 Testbenches define testing during VP stage.
 Test Strategies and Test Architecture are codeveloped!
 Testability Architecture determines what is in the system for test.
 Tester Architecture determines what the tester looks like.

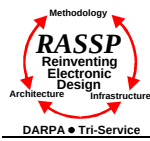
Ref [Sedmak95]



Requirements (TSD, TA) are flowed down which preserves test philosophy.

PVM measures are flowed back up (feedback).

Ref [Sedmak95]



Test Strategy Diagram (TSD)



- | **Used to bridge from requirements to implementation and through to manufacturing and field support**
 - m **Organizes and manages the key data generated during, and flowing between the methodology process steps**
 - m **Based on simple three dimensional spreadsheet type structures and operations**
 - m **Fault populations flow through various test means until they are Detected, Isolated and Corrected**

Ref [Sedmak95]



Test Strategy Diagram (Cont.)



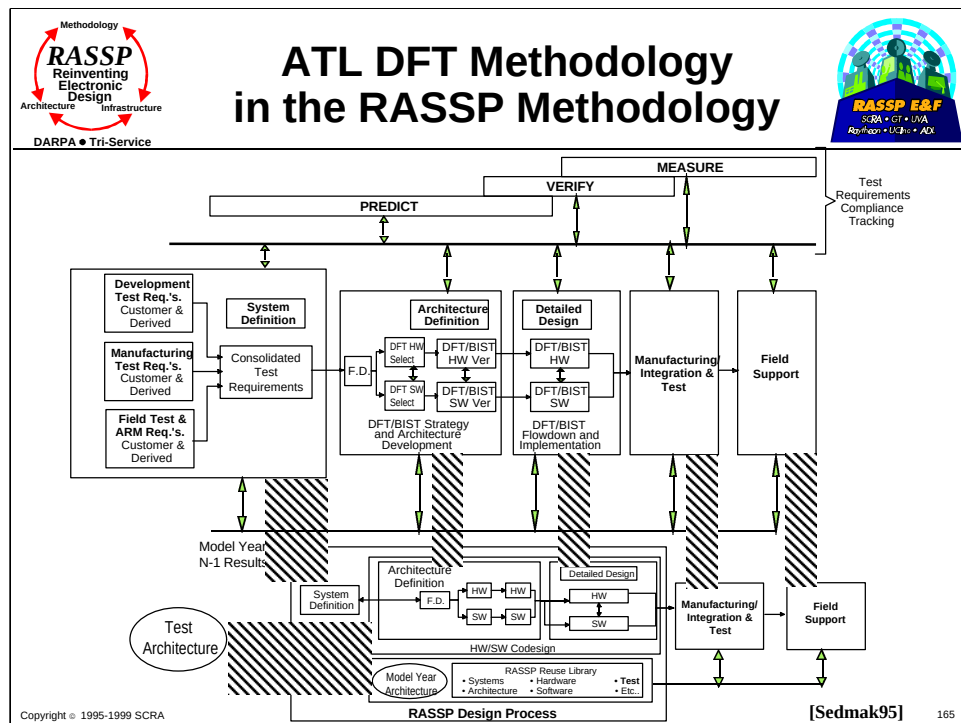
- | **Examples of applications include:**
 - m **Capturing quantitative test requirements**
 - m **Developing test strategies and making tradeoffs, including analysis of the mixing of test means**
 - m **Flowing requirements down to lower assembly levels**
 - m **Capturing predicted, verified, and measured (PVM) test performance data and comparing it against requirements for compliance tracking**
 - m **Rolling up test performance PVM data from lower levels for system level analysis**
 - m **Performing parametric analysis such as test time, product quality**

Copyright © 1995-1999 SCRA

[Sedmak95]

164

Ref [Sedmak95]



Management commitment (time, money) is required. Test Requirements Compliance Tracking allows continuous checking of compliance tracking throughout the design phases. The DFT methodology is integrated into the design flow (diagram doesn't show this well)!

Test Requirements are driven by cost analysis and technology. Consolidation of test requirements forces groups to merge requirements, resolve conflicts, and generate a singular test strategy. It also documents requirements in one place.

Architecture definition - actual test strategy chosen based on consolidated requirements - fault coverage, use of COTS, etc.

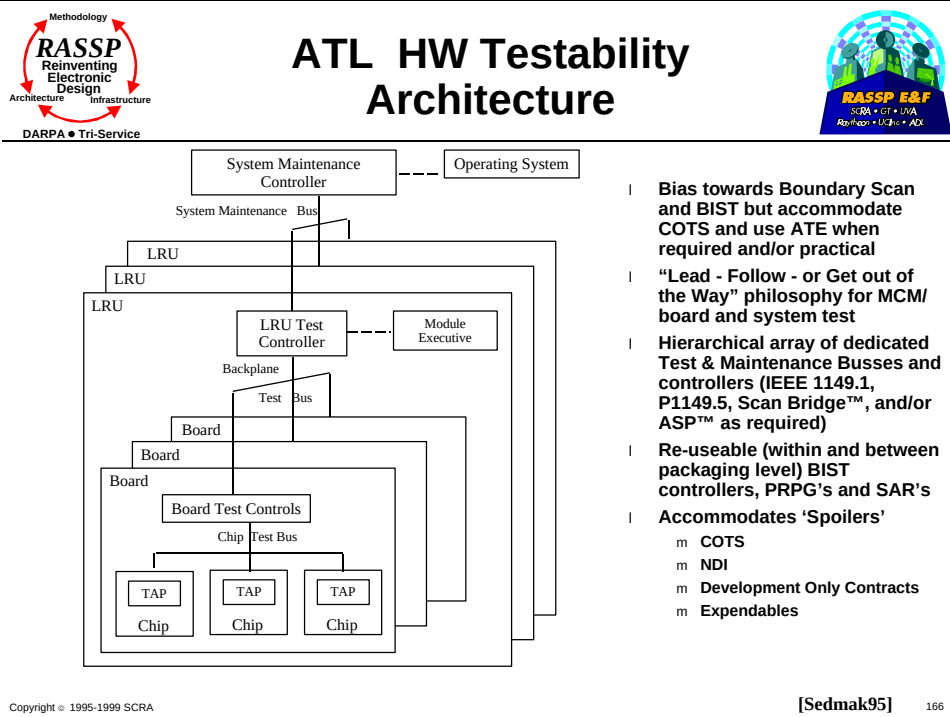
Detailed design phase - test architecture is flowed down. Fault simulation is performed.

Manufacturing - exploits DFT methodology, gains feedback on actual defects (MYA, reuse!)

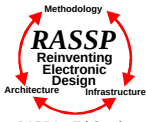
Field Support - gain feedback on actual field failures (MYA, Reuse!)

Meta data for components (contained in RASSP Reuse Data Management System-RRDM [Aspect Technologies]) contains information on "success" of techniques in previous systems to guide reuse in similar systems.

Ref [Sedmak95]



Ref [Sedmak95]



RASSP
Reinventing
Electronic
Design

DARPA • Tri-Service

RASSP Concepts which DFT Leverages



RASSP E&F
SCRA • GT • USA
Rep/agon • USMC • ADX

RASSP Concept	Benefit	DFT Leverage	Benefit
Model Year Architecture & Standard Virtual Interface	<i>Low cost technology upgrades over model years and across products</i>	Embed Testability Architecture into MYA	<i>Facilitate singular test philosophy & ease of upgrades</i>
Virtual prototype	<i>Early verification of top down, Hierarchical model of system</i>	Embed BIST resources into VP	<i>Early test & debug of BIST functions</i>
HW/ SW Co-Design	<i>Simpler integration & test & Improved product quality</i>	Capture Testability Architecture in Performance Models & DFG's	<i>Early development of test functions facilitates HW/SW Integration</i>
Enterprise Infra-Structure	<i>Automation and control of process and re-use of components and data</i>	Embed DFT steps in workflows and re-use libraries	<i>Integration of DFT into RASSP</i>

RASSP concepts provides a good framework for integrating design with test.

Copyright © 1995-1999 SCRA

[Sedmak95] 167

DFG's - data flow graphs (ie. PGM)

-primitives for power-on self-test, diagnostics added to command program.

Ref [Sedmak95]



Benefits of Integrating DFT into RASSP



Approach	Benefit
1. Consolidate test requirements	1. Reduce overall test development efforts and cost
2. Incorporate TSD construct	2. Bridge requirements to implementation
3. Extend concept of re-use	3. Minimize impact of test on schedule and cost
4. Implement conformance checking (via TSD).	4. Consistent framework for feedback of model year results.
5. Model test resources in VP	5. Concurrent test development shortens schedule
6. Integrate test architecture with MYA	6. Consistent use; Model year upgrades of test resources



Design-For-Test Integration Tasks

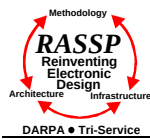


- | **Work Flows and Activity Definitions**
- | **DFT Tool Integration's**
 - m CAT or Test Specific (Fault simulation, Testability Analysis, DFT/ BIST insertion, ATPG, ...)
 - m CAE or Re-use of functional engineering tools (HDL Entry, SW development, simulation, Data Management, ...)
- | **Re-use Libraries**
 - m Object Class Hierarchy (DOCH)
 - m Contents of re-use elements (what data constitutes a re-use element)
 - m Initial population of critical elements
- | **Templates and standards for test related product data**
- | **Training**
 - m Benefits
 - m Process
 - m Tools

Copyright © 1995-1999 SCRA

[Sedmak95]

169



Module Outline



- | Introduction (Part 1)
- | Fault Modeling
- | Test Generation
- | Automatic Test Pattern Generation Algorithms
- | Fault Simulation Algorithms
- | Introduction (Part 2)
- | I_{DDQ} Testing
- | Design for Testability Techniques
- | Built-In Self Test
- | Hierarchical Design for Test
- | Synthesis for Test
- | DFT Standards
- | Design Flows with DFT
- | **Summary**

Copyright

170



Summary



- | **A background on general testing was provided**
 - m Fault Modeling
 - m Test Generation
 - m Faults Simulation
 - m I_{DDQ} Testing
- | **A background on Design for Test was provided**
 - m DFT Techniques
 - m Built-in Self Test Techniques
 - m Hierarchical DFT Techniques
 - m Synthesis for Test
- | **Finally, a background on how testing and DFT is used in the design process was provided**
 - m DFT Standards
 - m Design flows with DFT

Copyright © 1995-1999 SCRA

171



References



- [Aitken95] Aitken, R. C., "An Overview of Test Synthesis Tools," *IEEE Design and Test of Computers*, pp. 8-15, Summer 1995.
- [Abadir94] Abadir, M., G. McFall, S. Spencer, K. Drake, J. Evans "A Hierarchical Design for Test (DFT) Methodology for the Rapid Prototyping of Application Specific Signal Processing (RASSP)," *Preliminary Working Document*, Version 0.73, August 10, 1994.
- [Abramovici90] Abramovici, M., M. A. Breuer, A. D. Friedman, Digital Systems Testing and Testable Design, IEEE Computer Society Press, 1990 ; © IEEE 1990
- [Agarwal95] Agarwal, V. K., "Hierarchical and Integrated Build-In Self-Test CAD Tools," *Proceedings of the RASSP PI Meeting* January 9-13, 1995.
- [Fujiwara85] Fujiwara, H. Logic Testing and Design for Testability, The MIT Press, 1985.
- [Hayes85] Hayes, John P., "Fault Modeling," *IEEE Design and Test of Computers*, April, 1985, pp. 37-44.
- [Hemmady94] Hemmady, S., "VLSI Test Automation: Fact or Fiction?," *ASIC & EDA*, September 1994, pp. 56-58.
- [IEEE] All referenced IEEE material is used with permission.
- [IEEE1149.1] IEEE 1149.1 Standard Test Access Port and Boundary-Scan Architecture; © IEEE 1990
- [IEEEP1149.5] IEEE P1149.5 Standard Module Test and Maintenance (MTM) Bus Protocol; ; © IEEE 1995
- [Johnson89] Johnson, B. W., Design and Analysis of Fault Tolerant Digital Systems, Addison-Wesley, 1989.
- [Kim88] Kim, K., D. S. Ha, "On Using Signature Registers as Pseudorandom Pattern Generators in Built-In Self-Testing," *IEEE Transactions on Computer-Aided Design*, Vol. 7, No. 8, August 1988, pp. 919-928 ; © IEEE 1988

Copyright © 1995-1999 SCRA

172

References (cont.)

- [Klenke92] Klenke, R. H., R. D. Williams, J. H. Aylor, "Parallel-Processing Techniques for Automatic Test Pattern Generation," *IEEE Computer*, January 1992, pp. 71-84 ; © IEEE 1992
- [Lesser80] Lesser, J. D., J. J. Shedletsky, "An Experimental Delay Test Generator for LSI Logic," *IEEE Transactions on Computers*, March, 1980, pp. 235-248.
- [Richards97] Richards, M., Gadiant, A., Frank, G., eds. *Rapid Prototyping of Application Specific Signal Processors*, Kluwer Academic Publishers, Norwell, MA, 1997
- [Sedmak95] Sedmak, R., J. Evans, "Tutorial on the RASSP Design-for-Test ability Methodology," *2nd Annual RASSP Conference*, 1995.
- [Soden92] Soden, J. M., C. F. Hawkins, R. K. Gulati, and W. Mao, "IDDQ Testing: A Review," *Journal of Electronic Testing: Theory and Applications*, Kluwer Academic Publishers, Norwell, MA, 1992, pp. 5-17.
- [MAbadir94] Abadir, M. S., "Hierarchical Design for Testability Tools for RASSP," *Supplemental presentation material for the Martin Marietta RASSP Semi-Annual Government Review*, February, 23 & 24, 1994.
- [Maxwell92] Maxwell, P. C., R. C. Aitken, "IDDQ Testing as a Component of a Test Suite: The Need for Several Fault Coverage Metrics," *Journal of Electronic Testing: Theory and Applications (JETTA)*, Kluwer Academic Publishers, Norwell, MA, Vol. 3, No. 4, December 1992, pp. 19-30.
- [Maly87] Maly, W., "Realistic Fault Modeling for VLSI Testing," *Proceedings of the 24th ACM/IEEE Design Automation Conference*, 1987, pp. 173-180.
- [Malaiya86] Malaiya, Y. K., A. P. Jayasumana, R. Rajsuman, "A Detailed Examination of Bridging Faults," *IEEE*, 1986, pp. 78-81.
- [Waicukauskis87] Waicukauskis, J. A., et. al., "Transitional Fault Simulation," *IEEE Design & Test*, April, 1987, pp. 32-38; © IEEE 1987
- [Williams83] Williams, T. W., K. P. Parker, "Design for Testability - A Survey," *Proceedings of the IEEE*, Vol. 71, No. 1, January 1983, pp. 98 - 112; © IEEE 1983