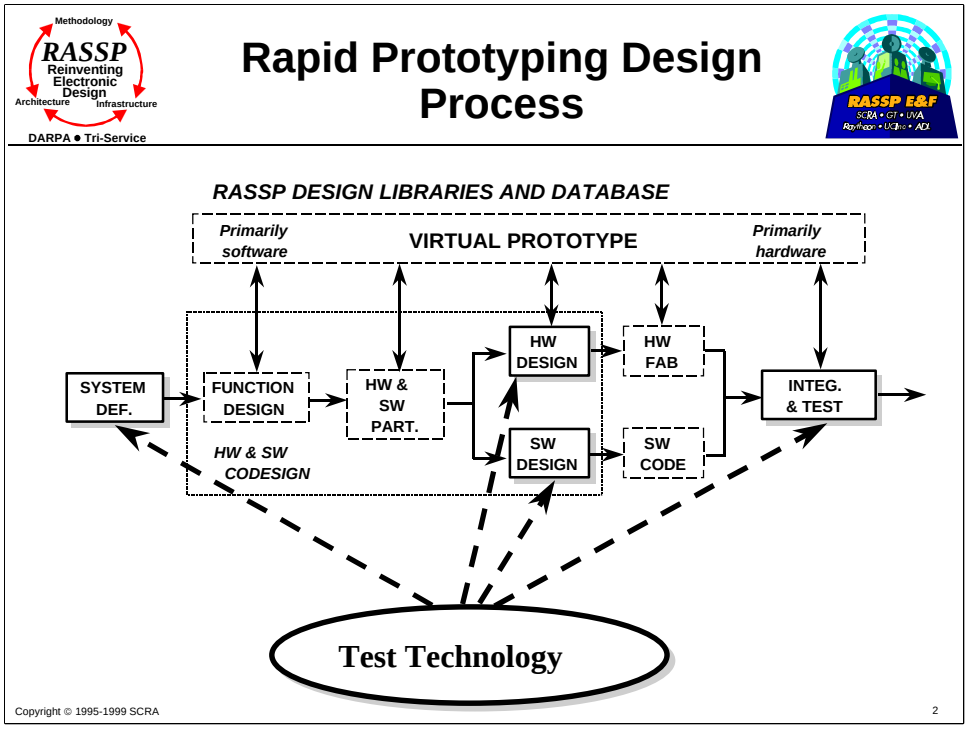
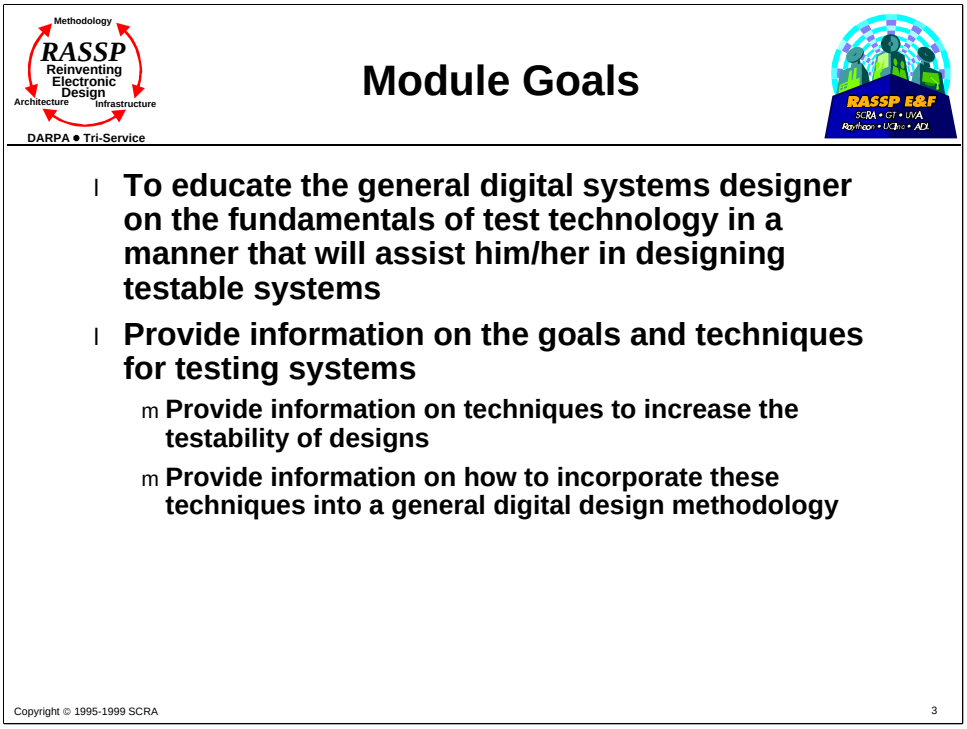
Test Technology Overview

RASSP Education & Facilitation Program

Module 43

Version 3.00





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
  - Ad hoc design for testability techniques
  - Structured design for testability techniques
    - Scan design
    - Boundary scan

Module Outline (Cont.)

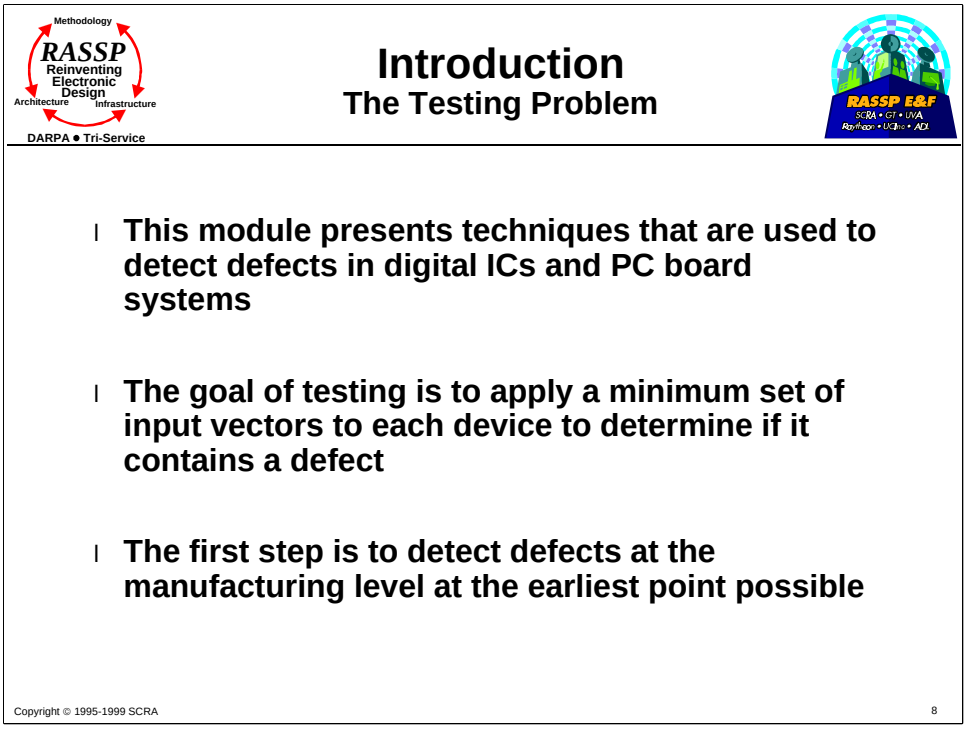
- **Built-In Self Test**
  - **Definitions**
  - **Test generation techniques for BIST**
  - **Signature analysis**
  - **BIST case study**
  - **Autonomous Built-In Self Test**
- **Hierarchical Design for Test**
- **Synthesis for Test**
- **DFT Standards**
  - **IEEE Std. 1149.1**
  - **IEEE Std. 1149.1b**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

Module Outline (Cont.)

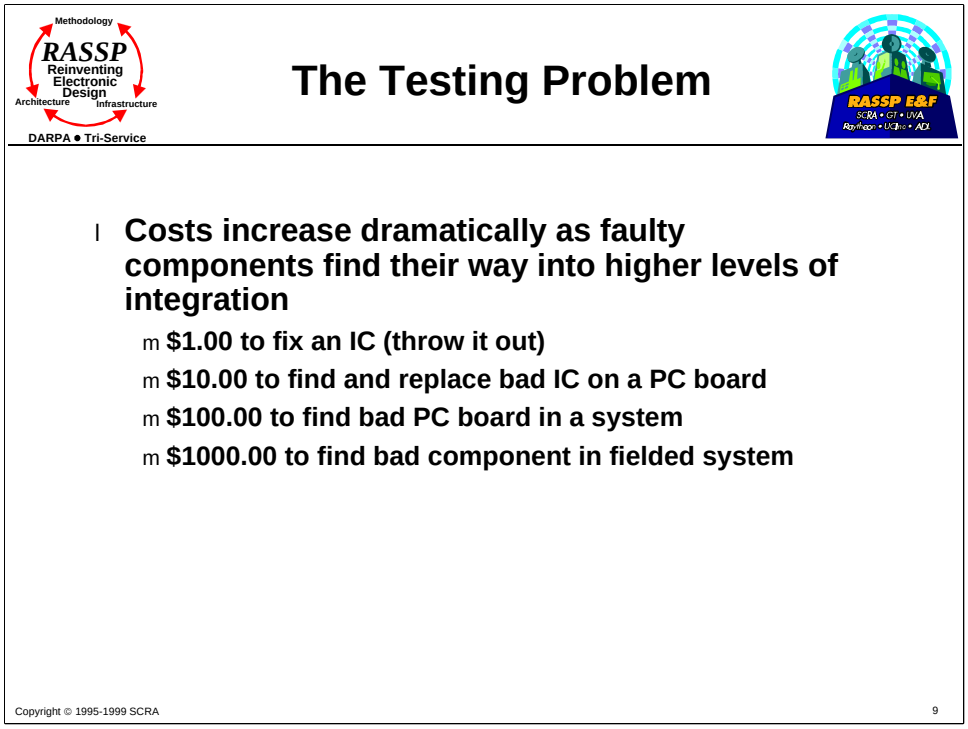
- Design Flows with DFT
- Summary
- References

Module Outline

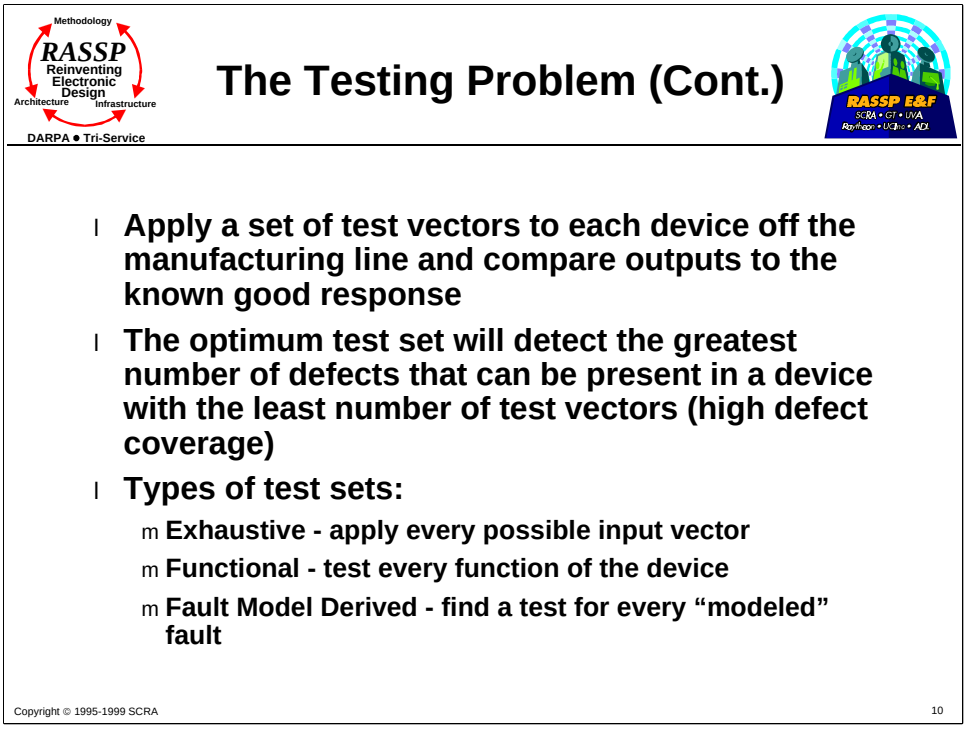
- **Introduction (Part 1)**
- **Fault Modeling**
- **Test Generation**
- **Automatic Test Pattern Generation Algorithms**
- **Fault Simulation Algorithms**
- **Introduction (Part 2)**
- **I_{DDQ} Testing**
- **Design for Testability Techniques**
- **Built-In Self Test**
- **Hierarchical Design for Test**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



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

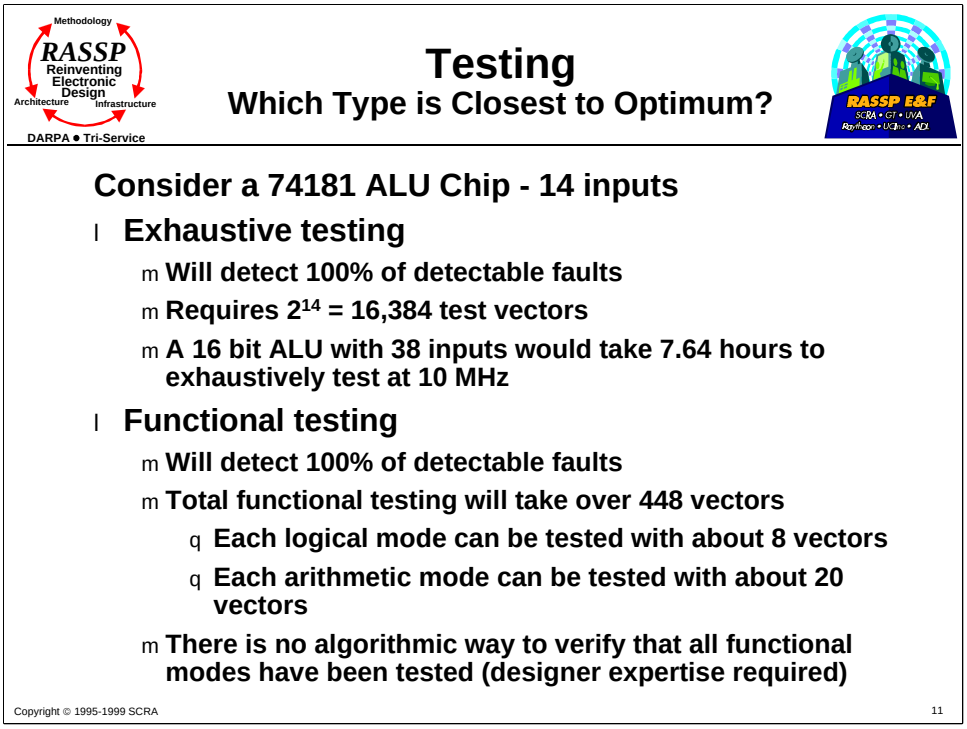


Testing for manufacturing defects incurs costs in the form of time during the design cycle (to generate the necessary test) and time during manufacturing (to actually apply the tests to the devices). However, the costs of letting a defective device be used in a system increase typically by an order of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



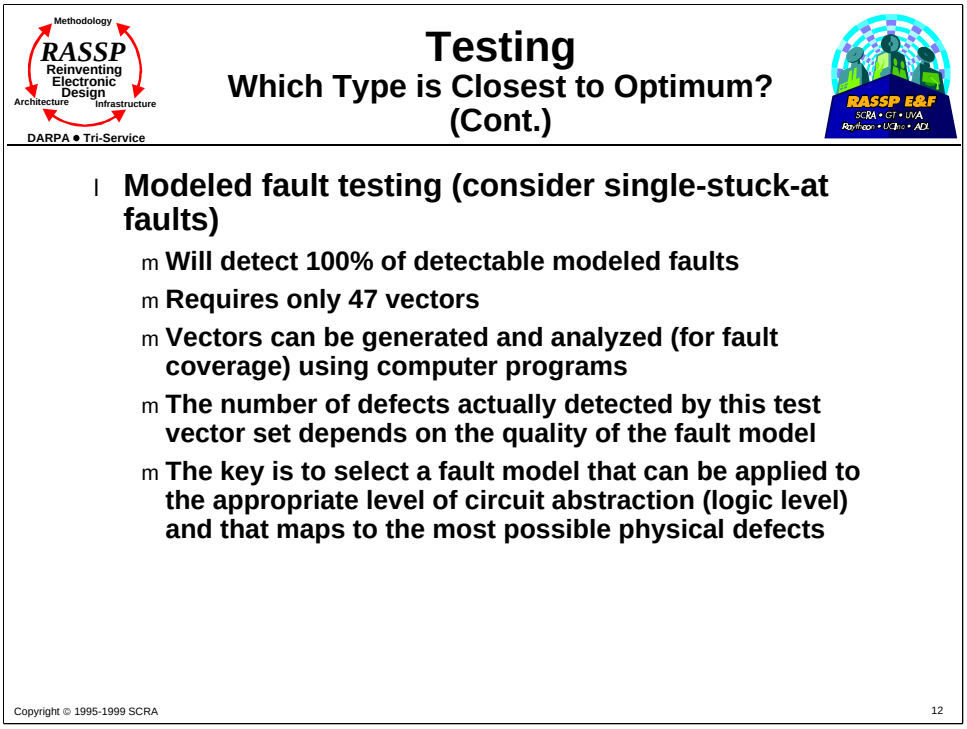
This is the methodology used to test devices during manufacturing. Development of the optimal test set is the most difficult part of the process. The goal is to have the shortest test set possible for maximum fault coverage because a longer test set takes more time to apply to each device ($$$).

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

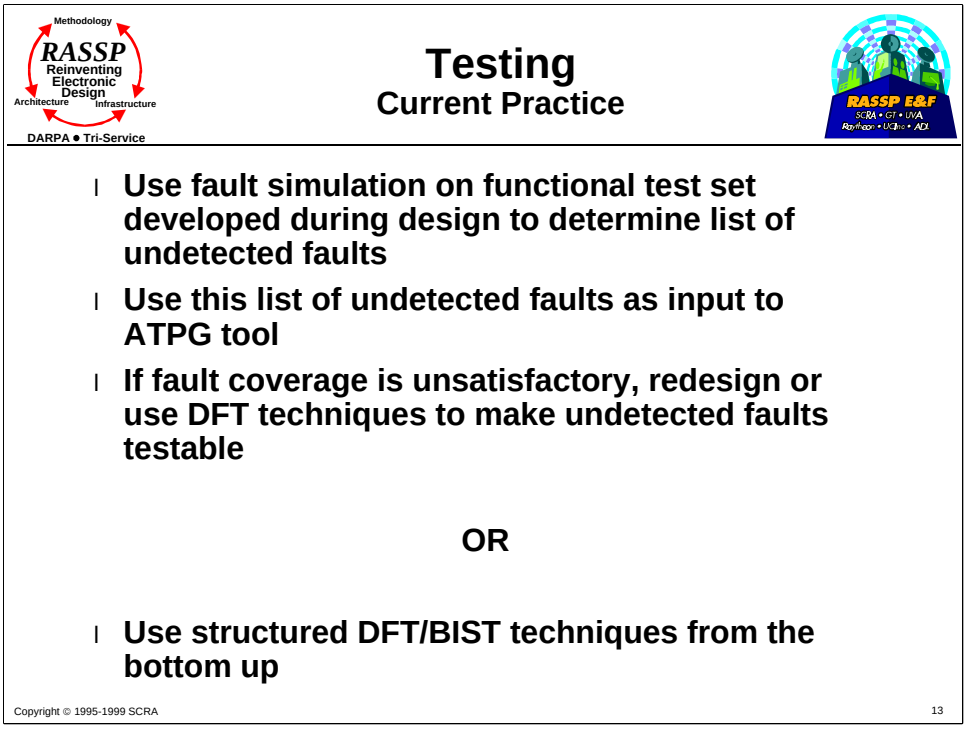


Exhaustive testing consists of applying every possible input combination to a device under test. It is guaranteed to detect all detectable faults, but it is not practical in terms of number of tests required.

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.



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.

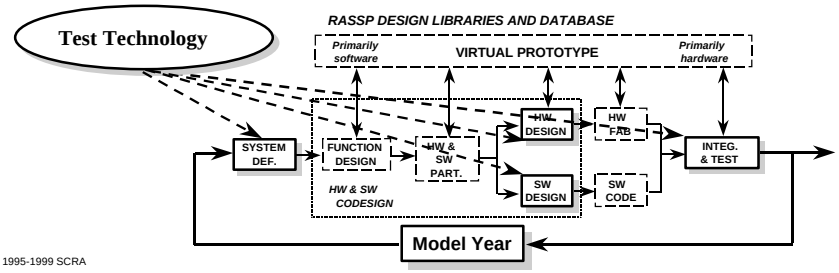


For most ASIC designs, the typical practice is to begin with the designer's functional verification test set. This set of vectors is then fault simulated to determine its fault coverage. If it is too low to be acceptable, more functional vectors can be added to exercise the portions of the circuit where 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.

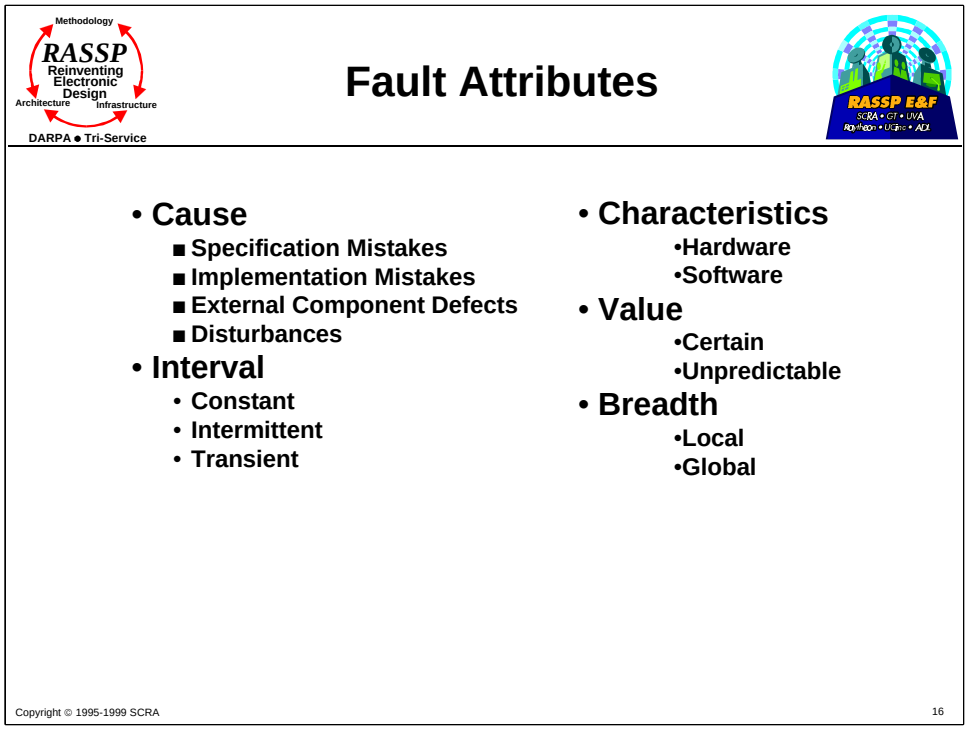
Testing and Rapid Prototyping

- 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



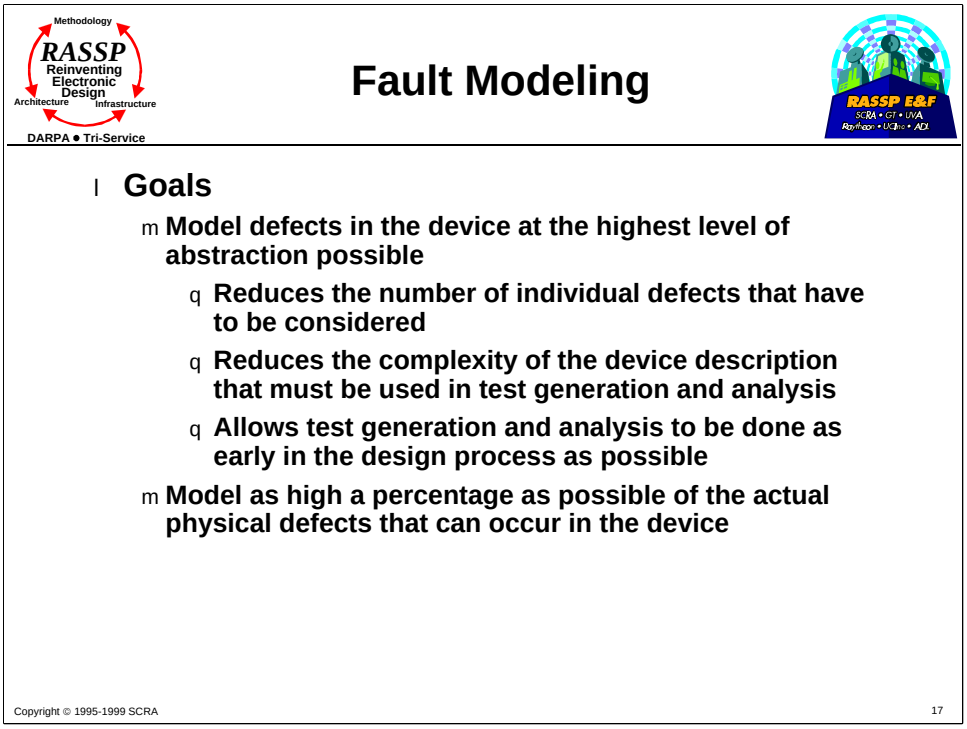
Module Outline

- 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

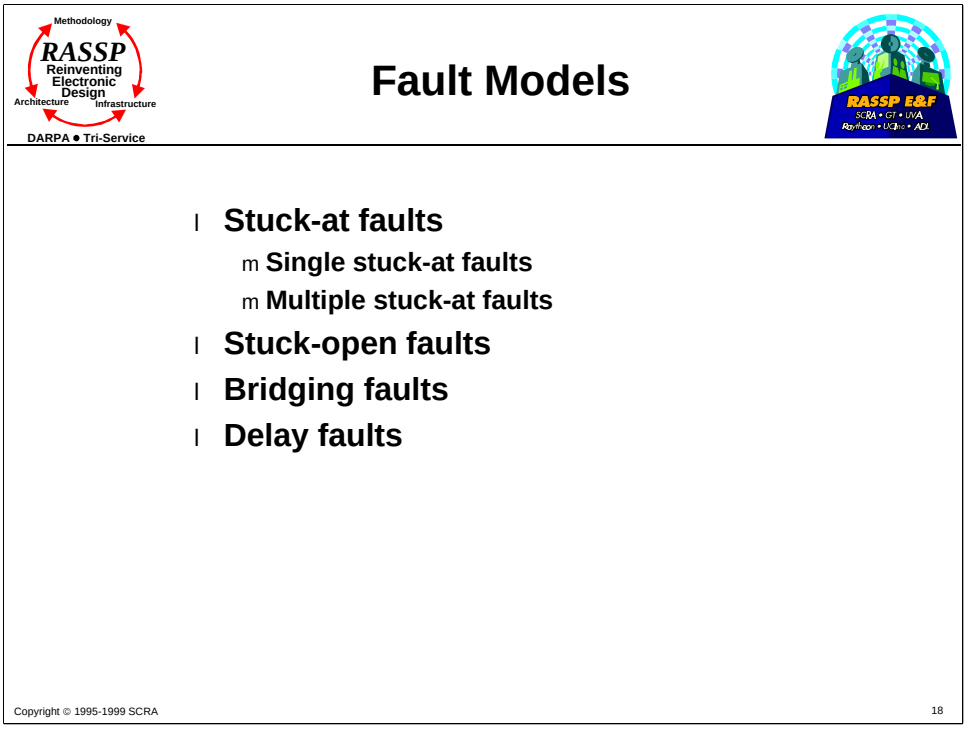


Presented here is the taxonomy of possible faults in a device.

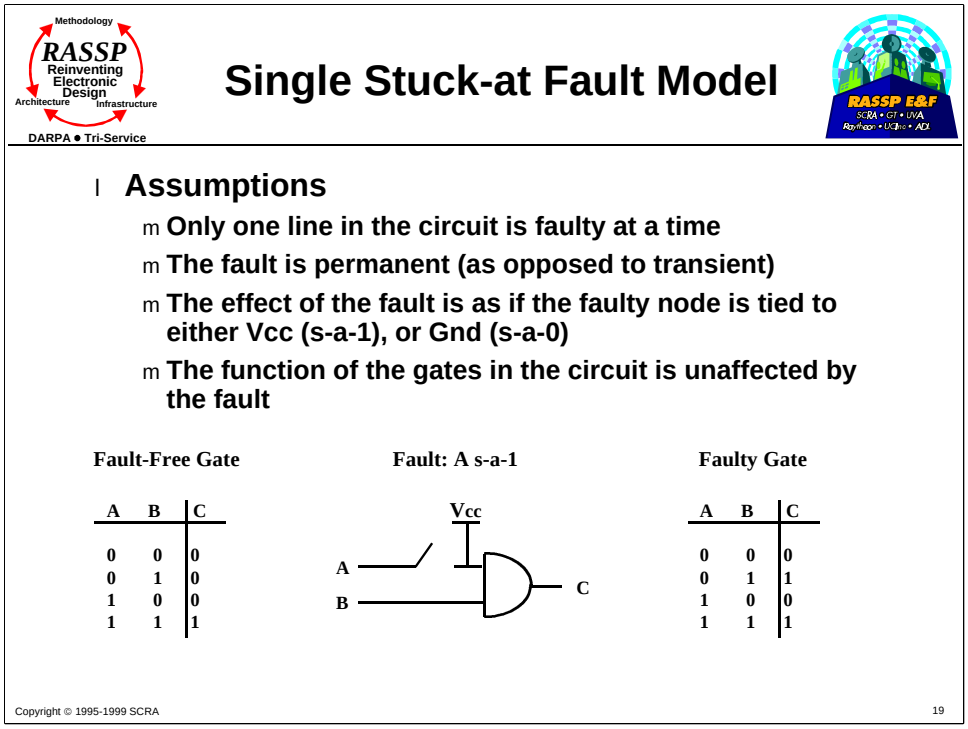
Ref: [Johnson89]



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.

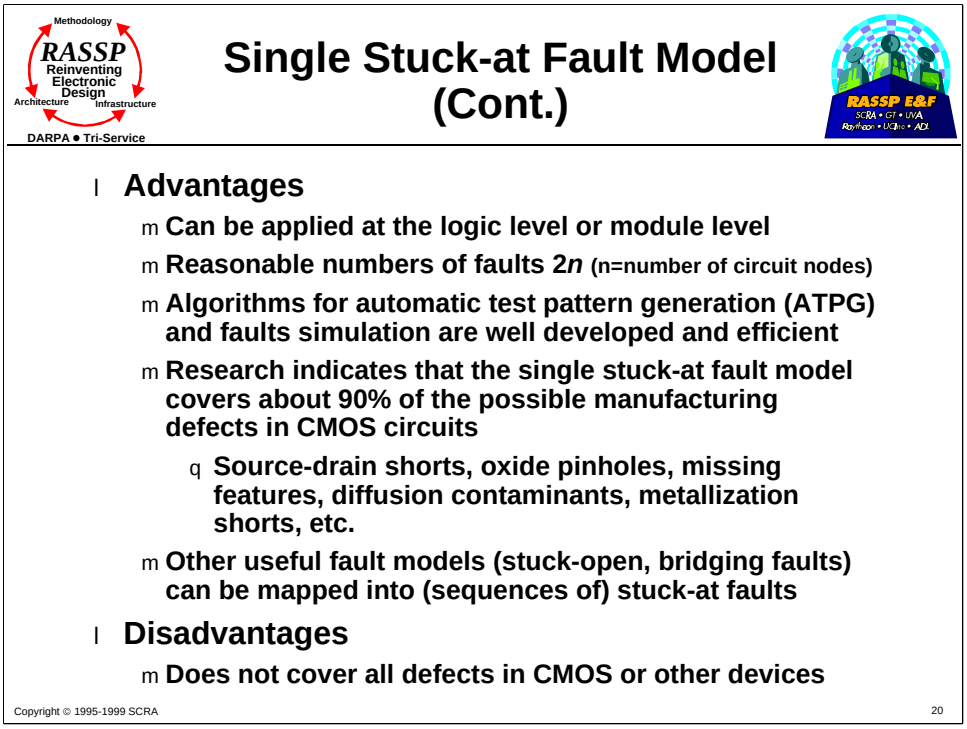


Here is listed the types of fault models that will be presented. Several other types have been developed, but these are the most commonly used.



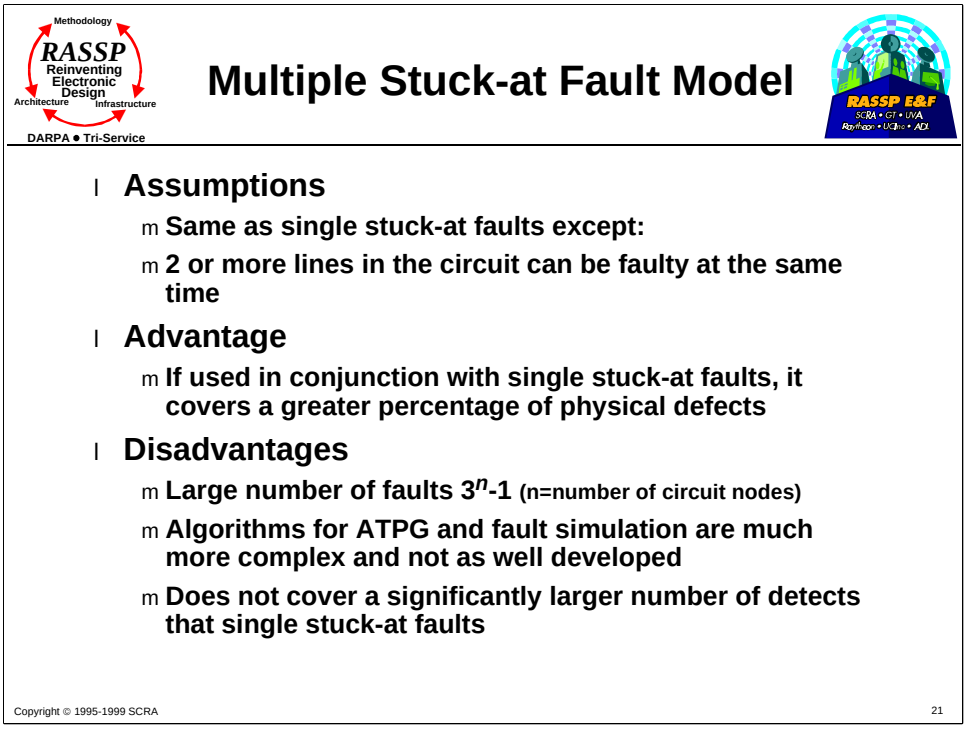
The single stuck-at fault model is the most often used fault model. Here are the assumptions or characteristics of the single stuck-at model.

Ref: [Hayes85],[Maly87]



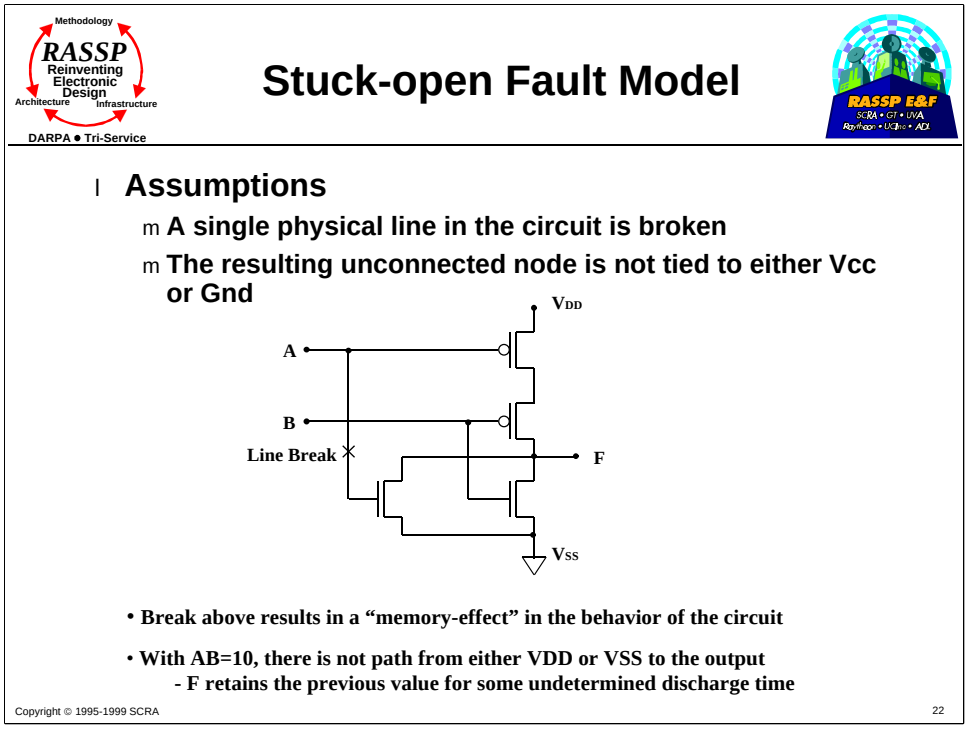
The numerous advantages of the single stuck-at model are the reason it is most often used. It is simple to use during test generation and fault simulation; research has shown that it covers a large percentage of physical defects; and other fault models, that can increase defect coverage, can be mapped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

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

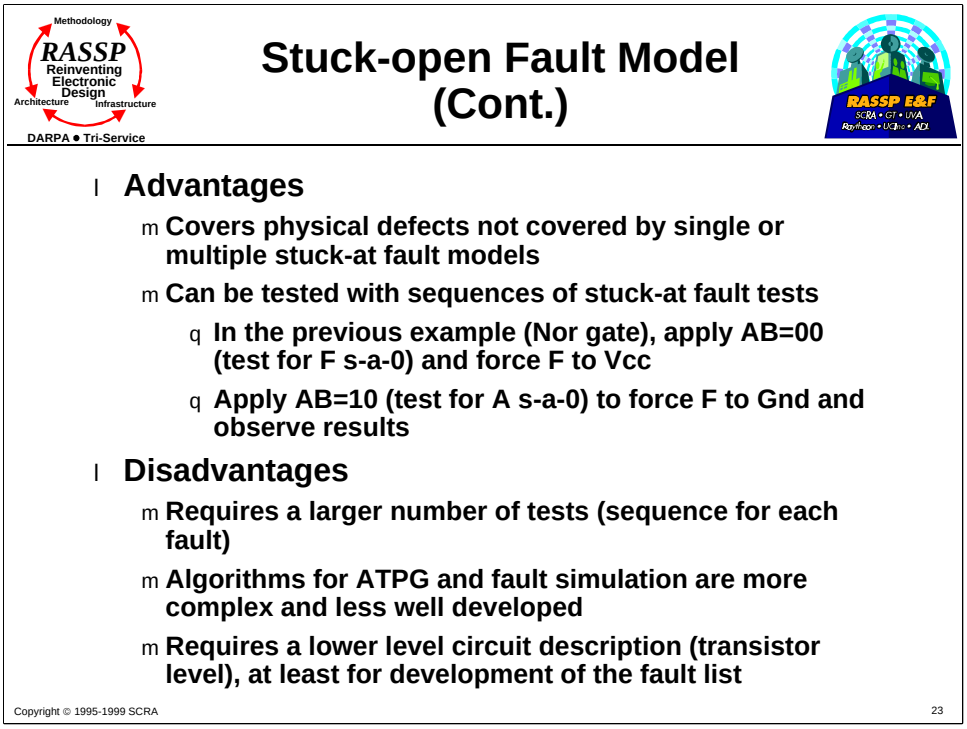


The multiple stuck-at model is sometimes used. It typically doesn't significantly increase the defect coverage enough to offset its major disadvantage of a large number of possible multiple stuck-at fault combinations.

Ref: [Hayes85]

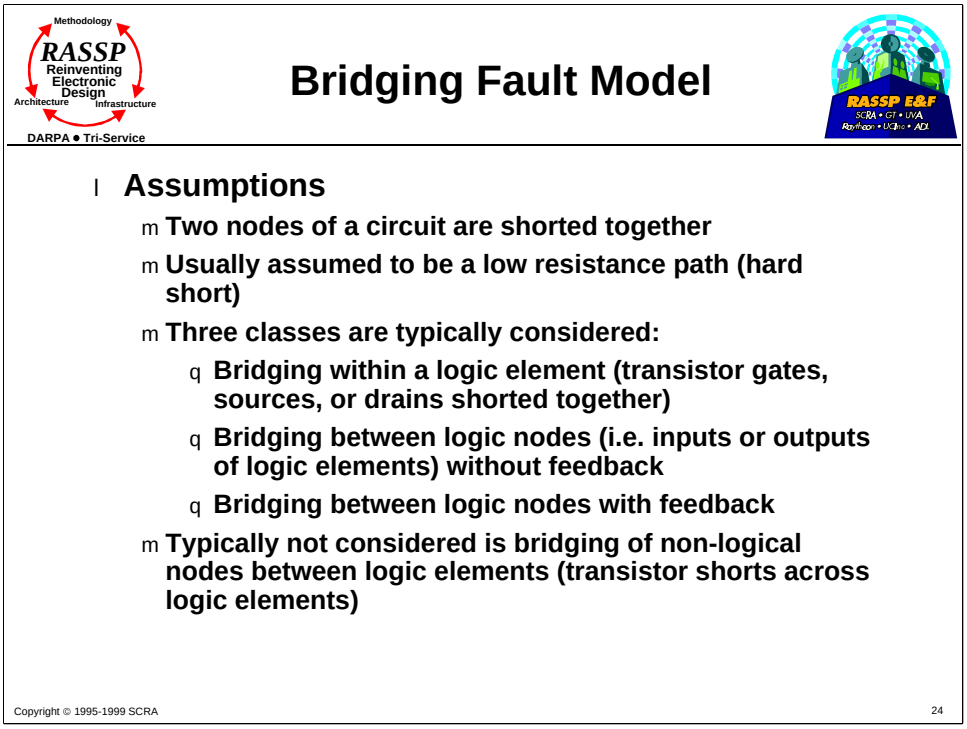


The stuck-open fault is popular for use in CMOS circuits. When a defect occurs that breaks a line internal to a CMOS gate, it can result in a "memory effect." For example, in this circuit, if the output is driven high with a 00 input, and then a 10 input is applied, then the output will stay high (its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



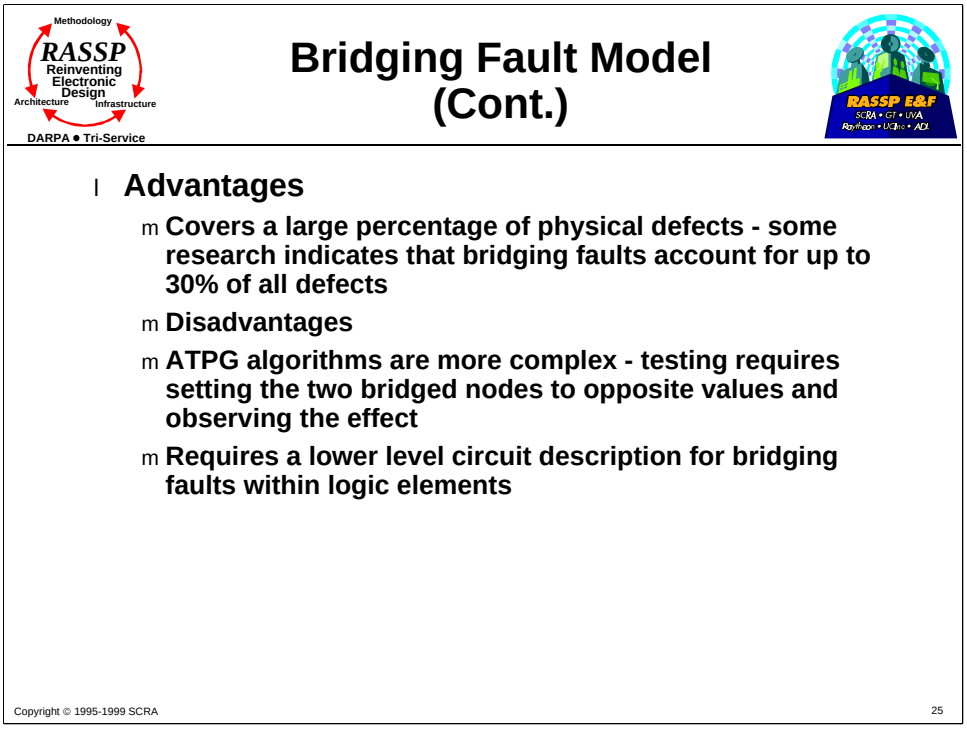
Stuck-open faults can be tested for by sequences of stuck-at fault tests. This will typically result in higher defect coverage.

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 that single stuck-at testing.

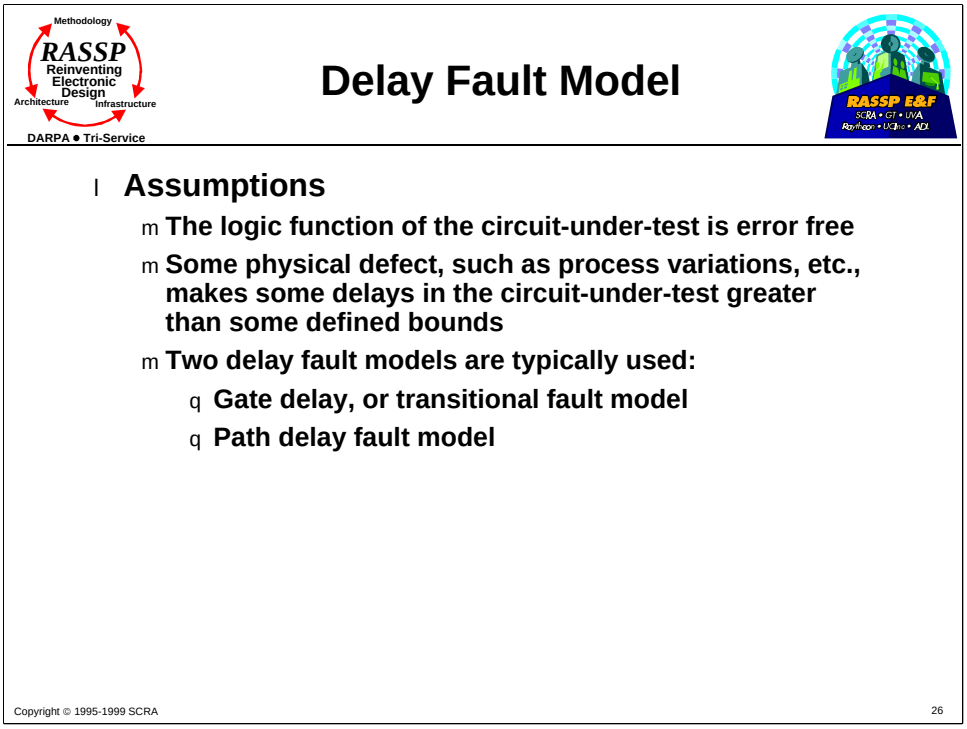


The bridging fault model is also sometimes used. It is more applicable as line widths get smaller. A hard short between lines is assumed. The model is that of logical lines in the circuit shorted together. The possibility of lines internal to a logic element shorted together is typically not considered 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).

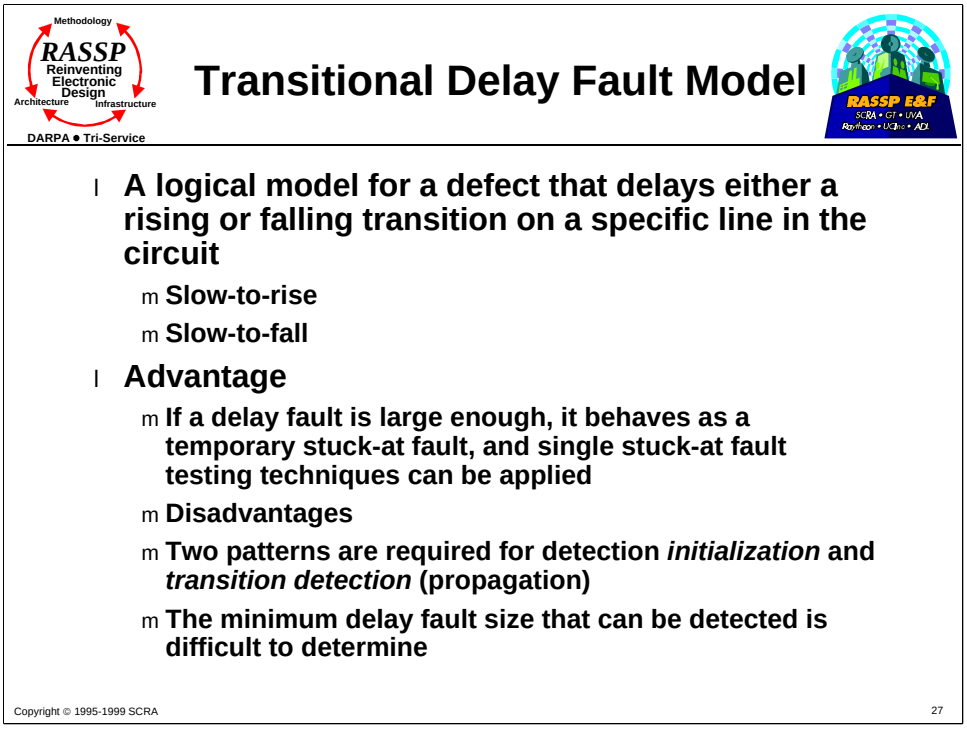
Ref: [Malaiya86]



Bridging fault 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.



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.

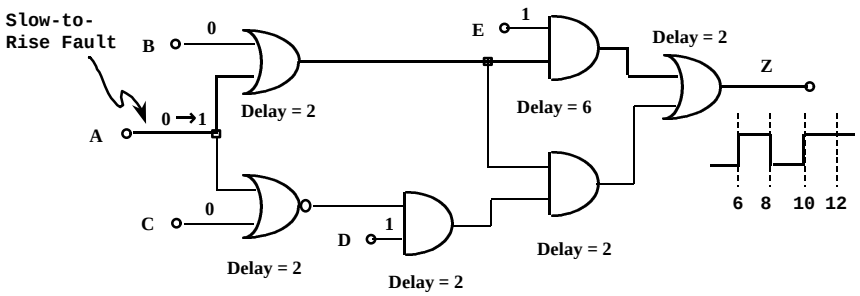


The gate delay, or transitional delay fault model, was the first delay fault model to be developed and is also the simplest. The transitional fault model can be modeled as a temporary stuck-at fault, and tests for it can be developed using this model. Again, the major disadvantage is the complexity of 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

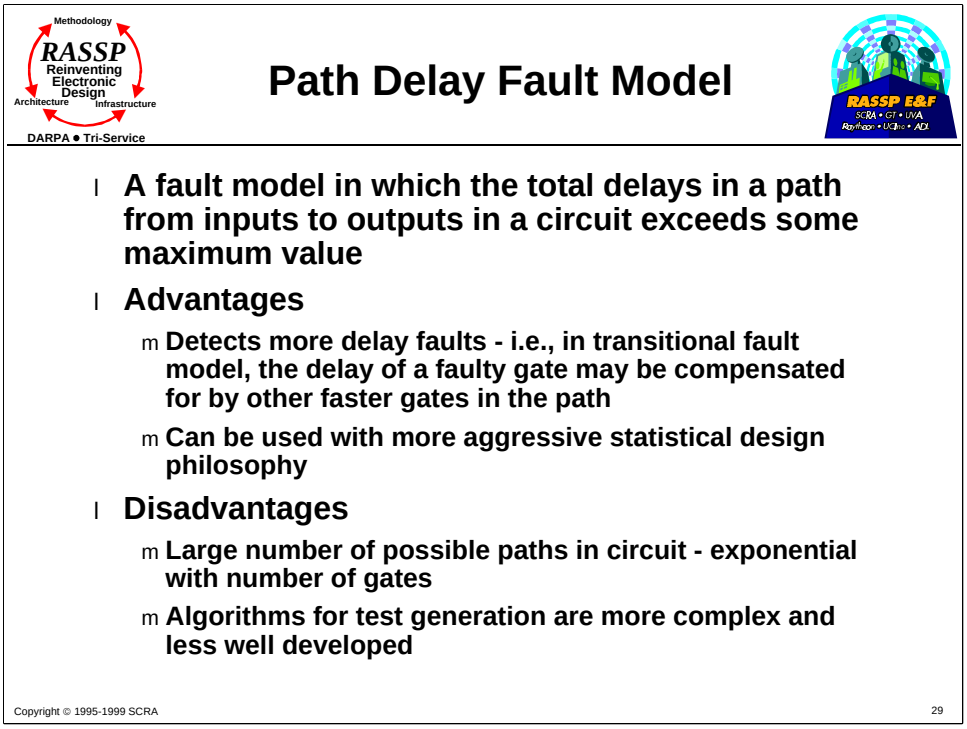


© IEEE 1987

[Waicukauski87]

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]



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.

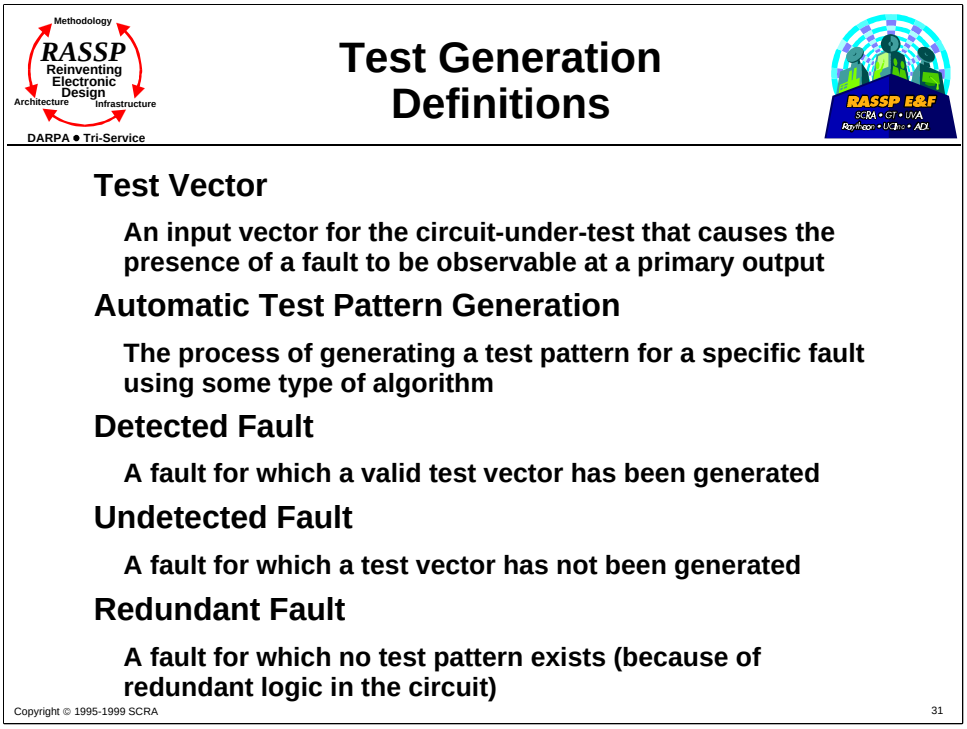
This delay fault model is also consistent with a statistical design philosophy. A statistical design philosophy recognizes that the delays of gates in a circuit are usually not all worst case, but that they fall within a small “typical” range. Using this knowledge, a greater clock speed can be specified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

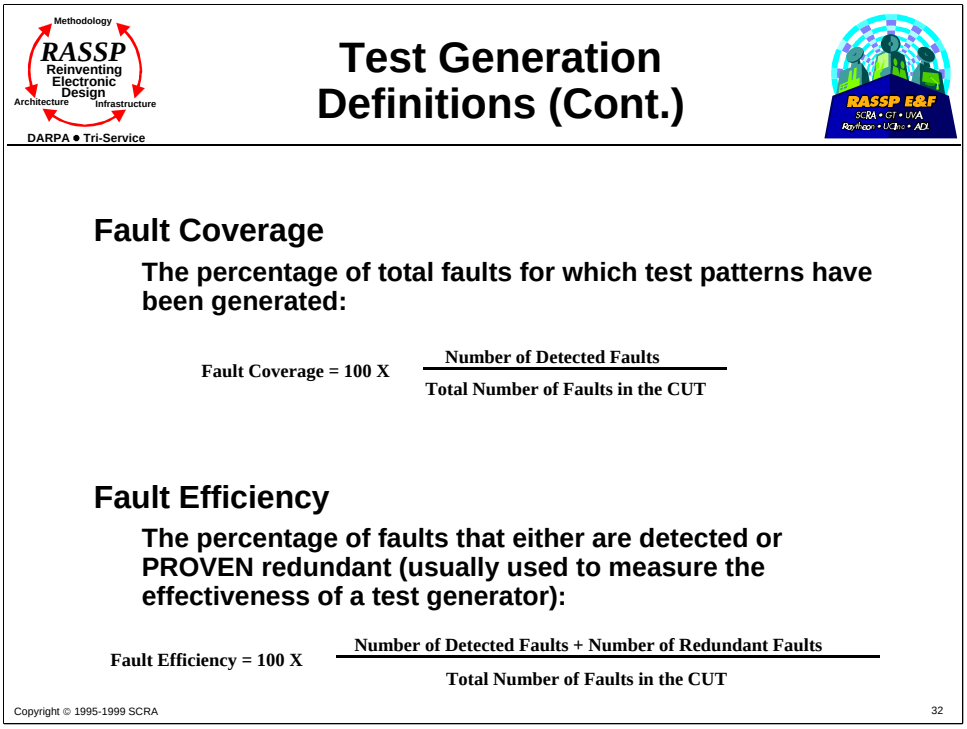
Ref: [Lesser80]

Module Outline

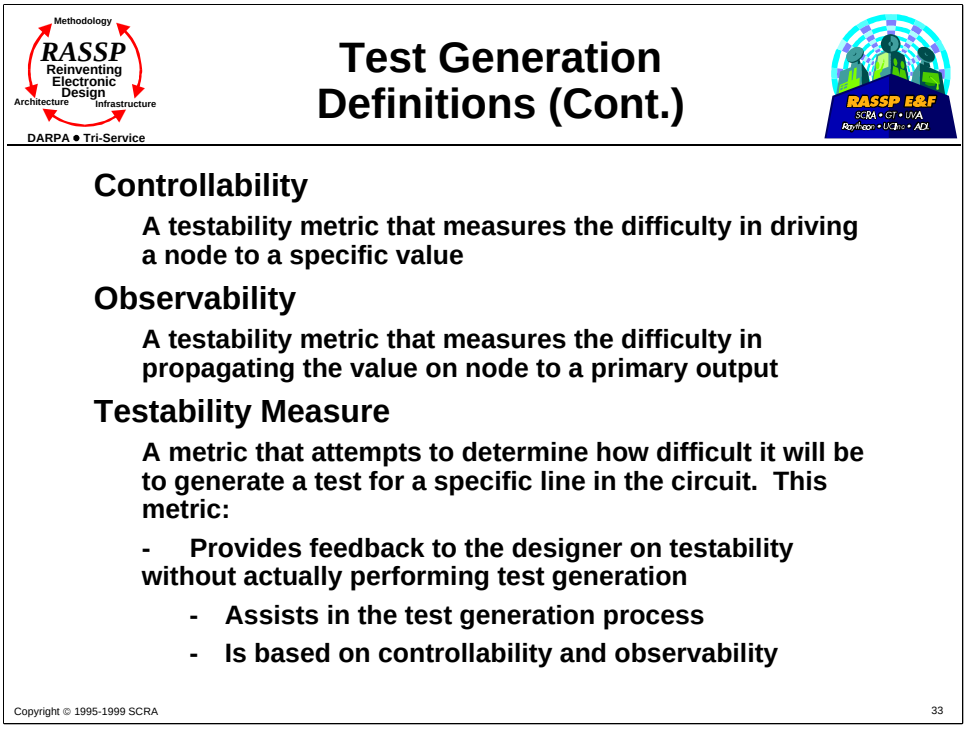
- Introduction (Part 1)
- Fault Modeling
- **Test Generation**
- Automatic Test Pattern Generation Algorithms
- Fault Simulation Algorithms
- Introduction (Part 2)
- I_{DDQ} Testing
- Design for Testability Techniques
- Built-In Self Test
- Hierarchical Design for Test
- Synthesis for Test
- DFT Standards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



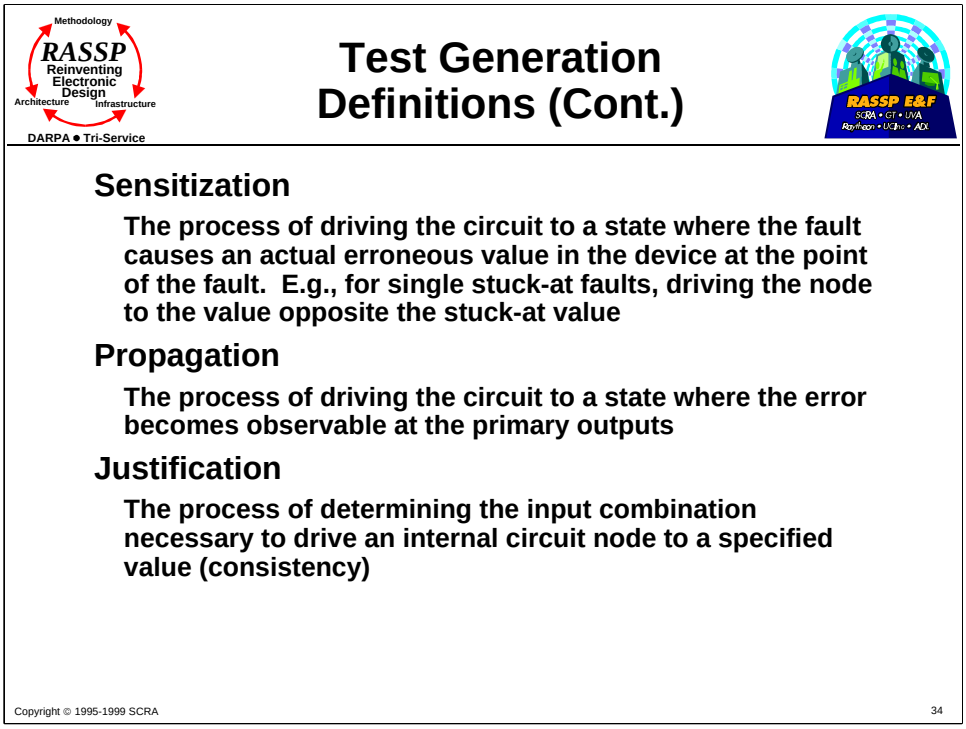
No additional notes for the definitions are needed.



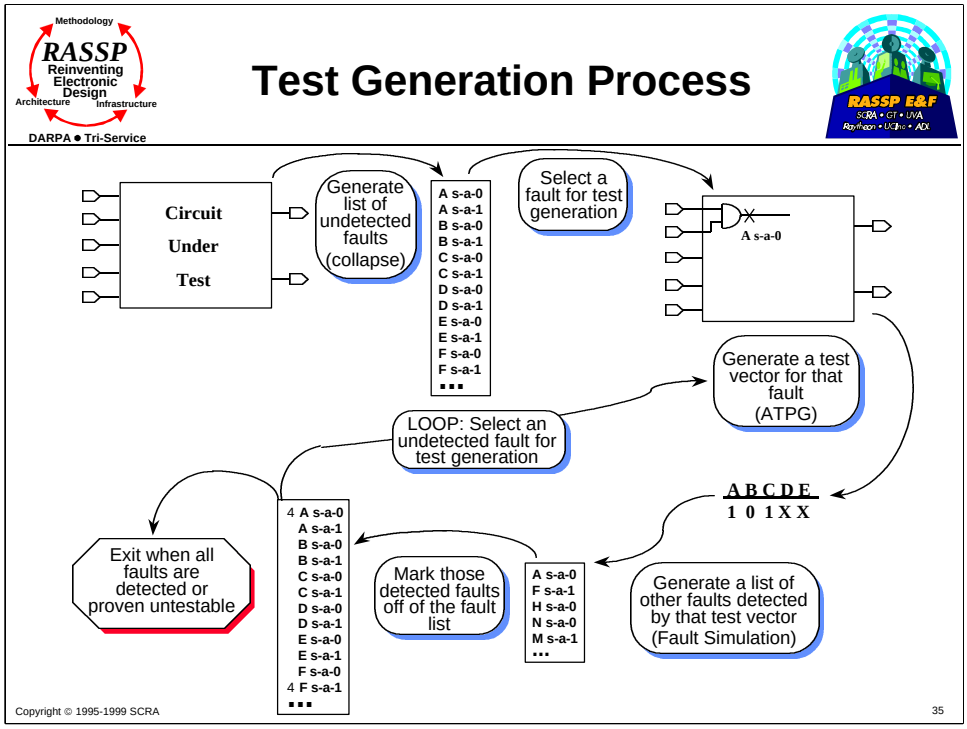
No additional notes for the definitions are needed.



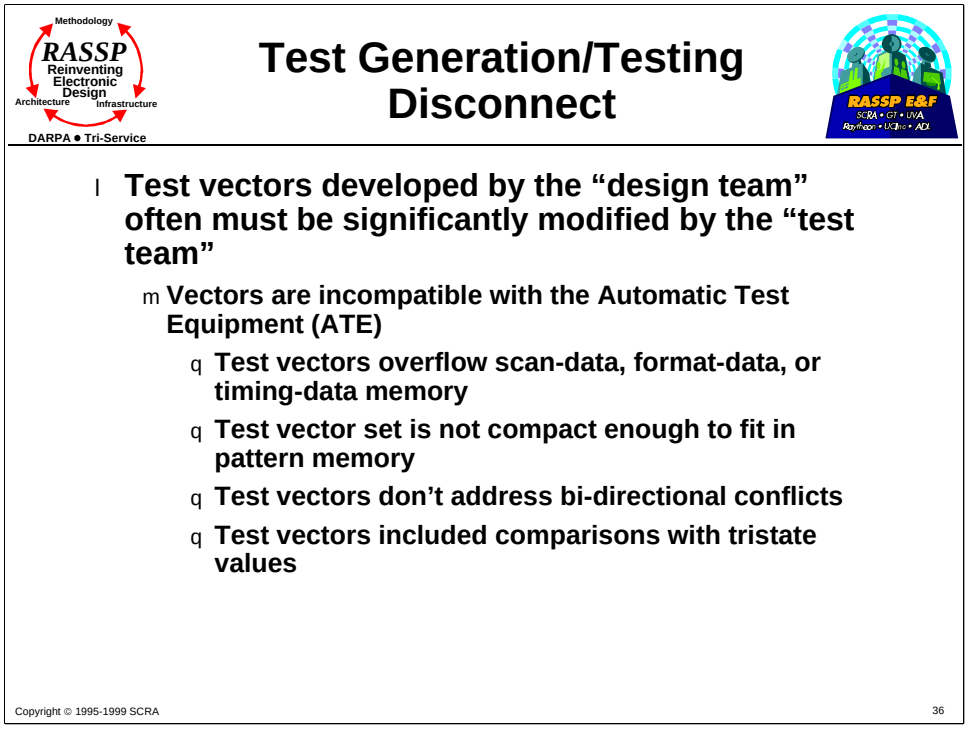
No additional notes for the definitions are needed.



No additional notes for the definitions are needed.

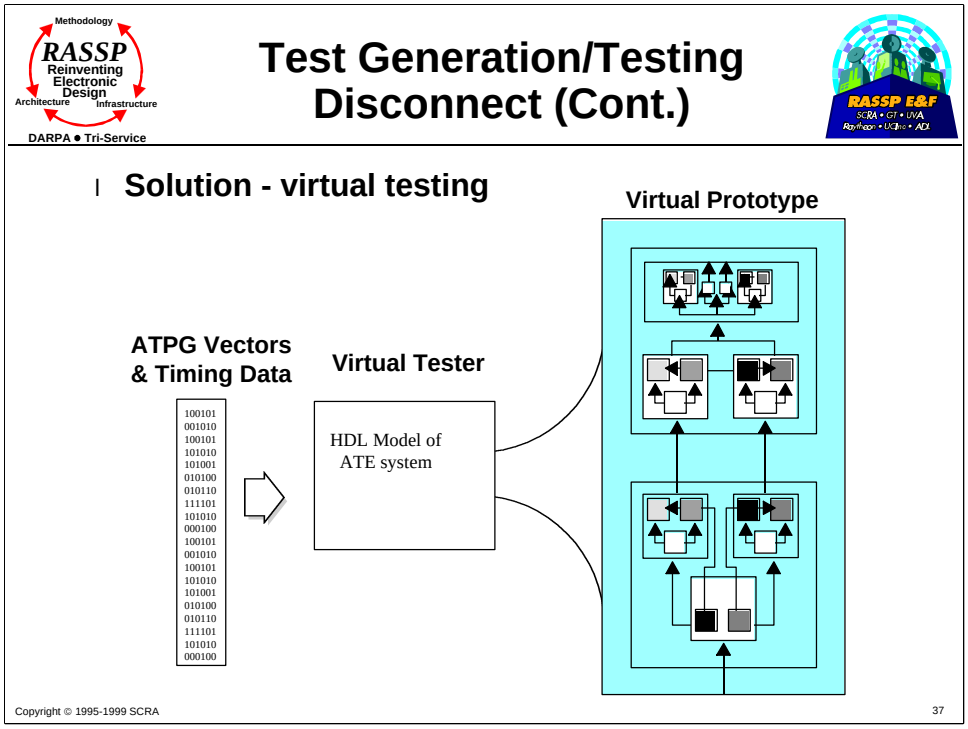


This slide shows the overall flow of the test generation process. The process begins by generating a list of all possible faults in the circuit under test. One of these faults is selected for test generation. ATPG is performed to generate a test for this fault. Once the test is generated, fault simulation 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).



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.

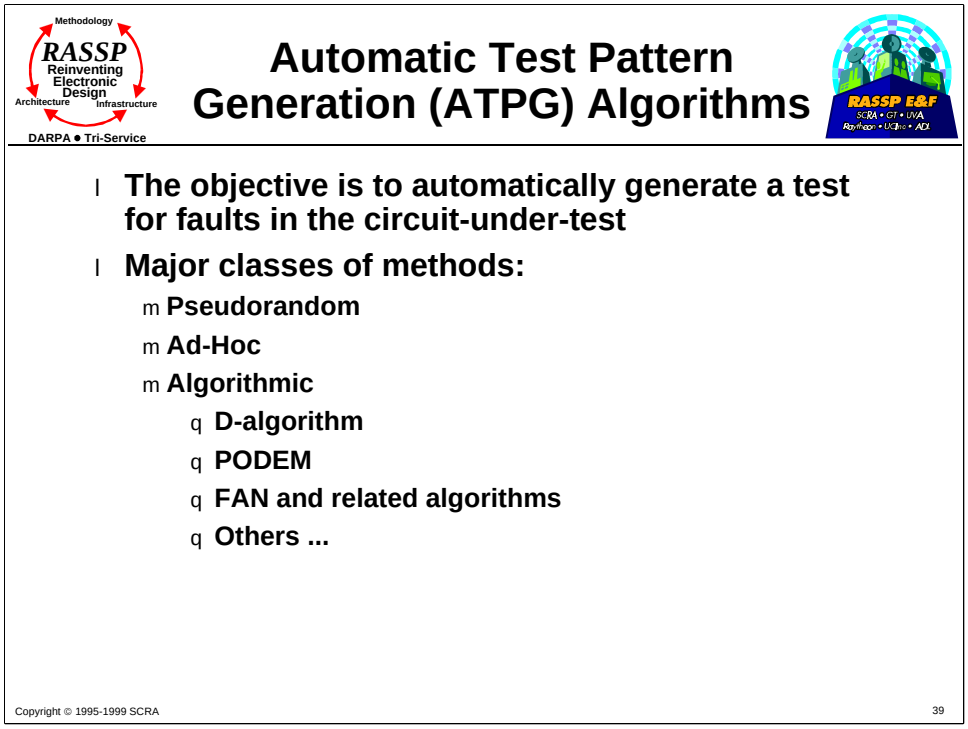
Ref: [Hemmady94]



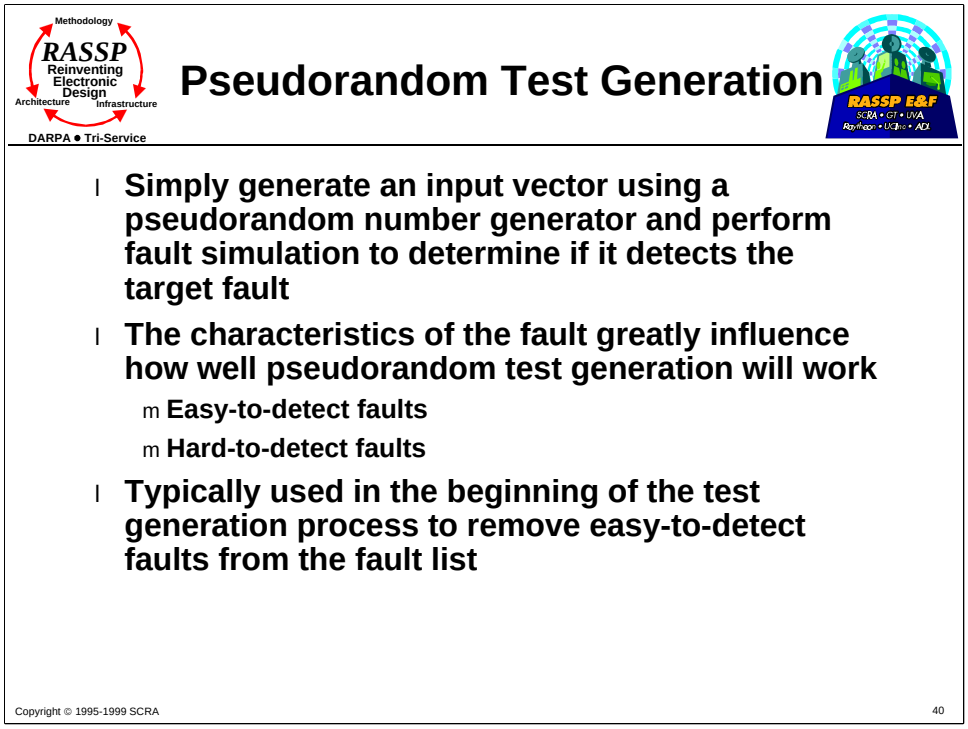
One possible solution to this problem is the concept of “virtual testing” which is built on top of a virtual prototype of the system-under-test. What is needed is an HDL description of the tester through which the intended test vectors can be run and applied to the virtual prototype. Using this method,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

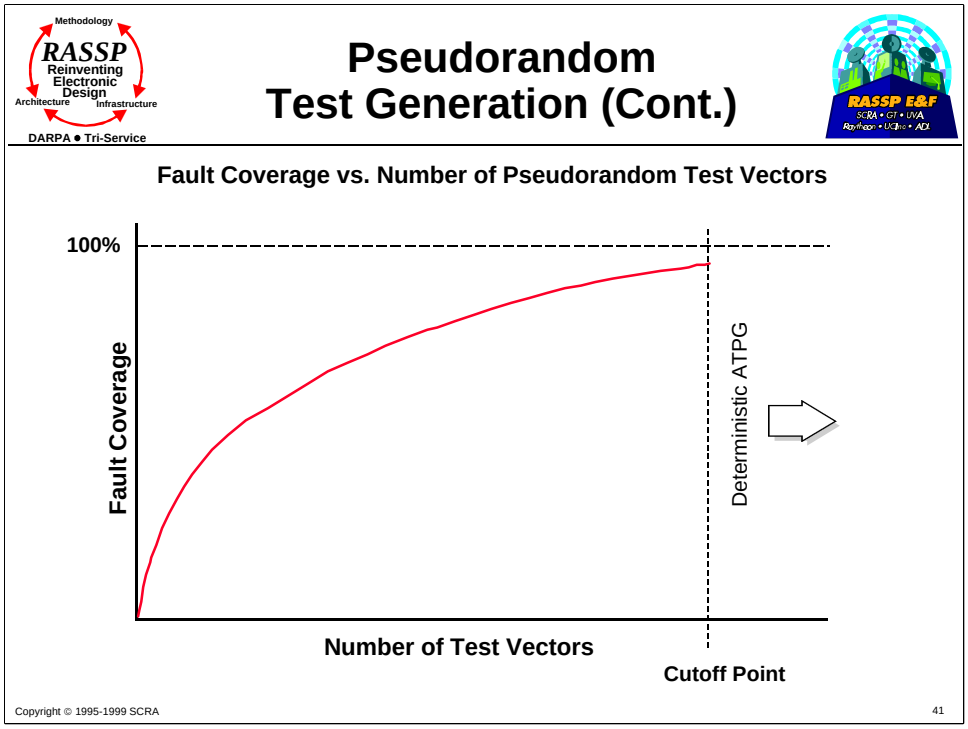


Automatic test pattern generation is used to generate test vectors during the test generation process. There are two major ATPG methods: pseudorandom and deterministic. Algorithmic ATPG is used to generate tests for specific faults in the circuit-under-test.



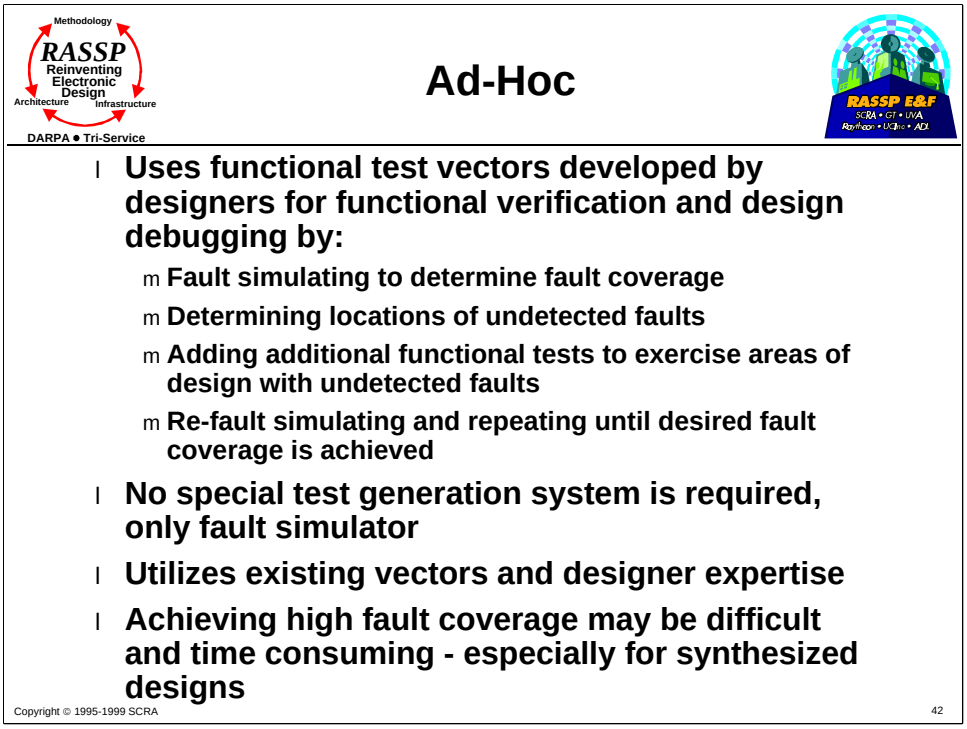
Pseudorandom test generation is the simplest method for generating tests and is typically used initially in the test generation process to quickly remove easy-to-detect faults from the fault list.

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.



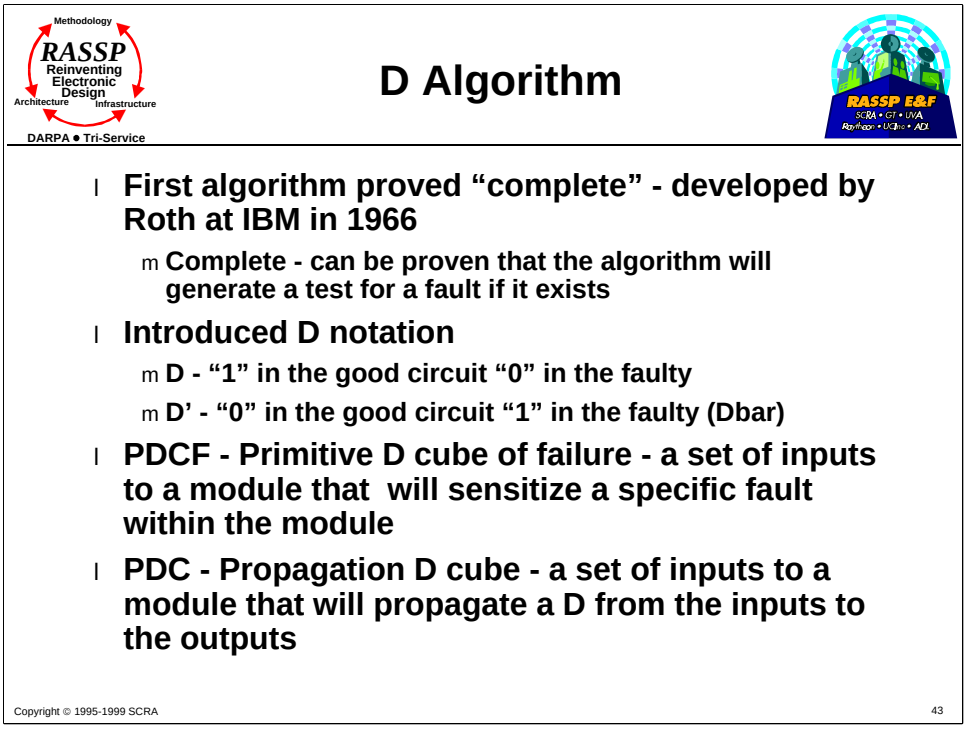
This graph shows the typical fault coverage vs. number of pseudorandom test vectors curve. Pseudorandom test generation is very efficient up to the point where the curve begins to flatten out (circuit dependent), and the deterministic ATPG is typically used to target the remaining undetected faults.

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.



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.



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.

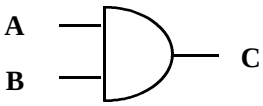
The D algorithm introduced the "D notation" which has been used in most subsequent ATPG algorithms.

Ref [Klenke92]

D Algorithm (Cont.)

D Notation

Good Circuit Value	Faulty Circuit Value	D Value
1	0	D
0	1	D’



AND Gate PDCFs

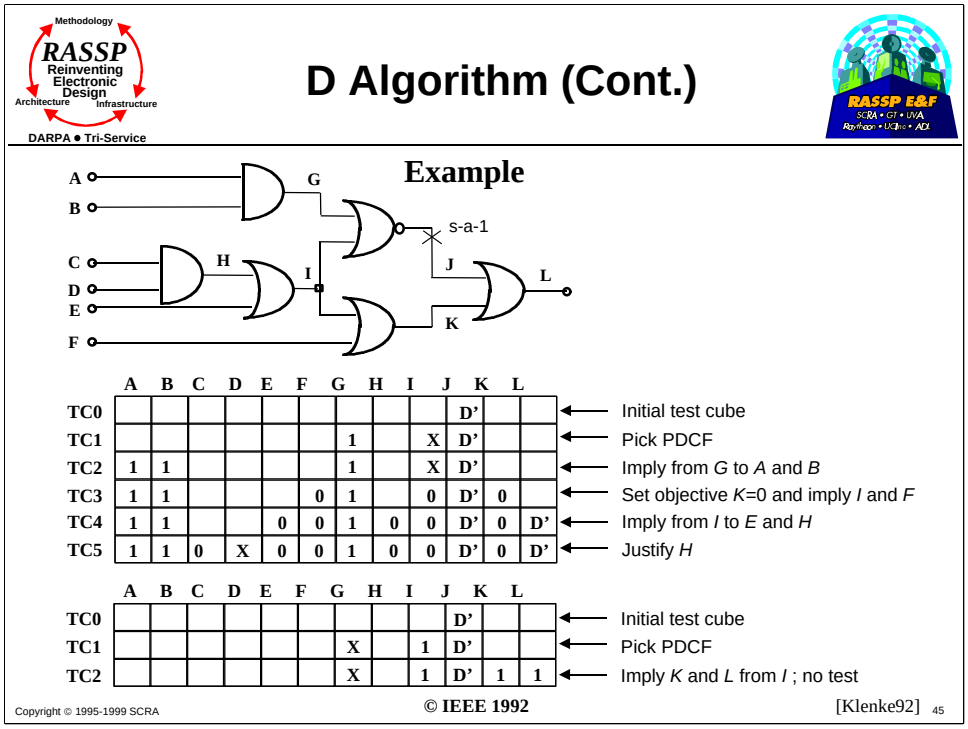
Fault	PDCF A	B	C
A s-a-0	1	1	D
B s-a-0	1	1	D
C s-a-0	1	1	D
A s-a-1	0	1	D’
B s-a-1	1	0	D’
C s-a-1	X	0	D’
C s-a-1	0	X	D’

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.

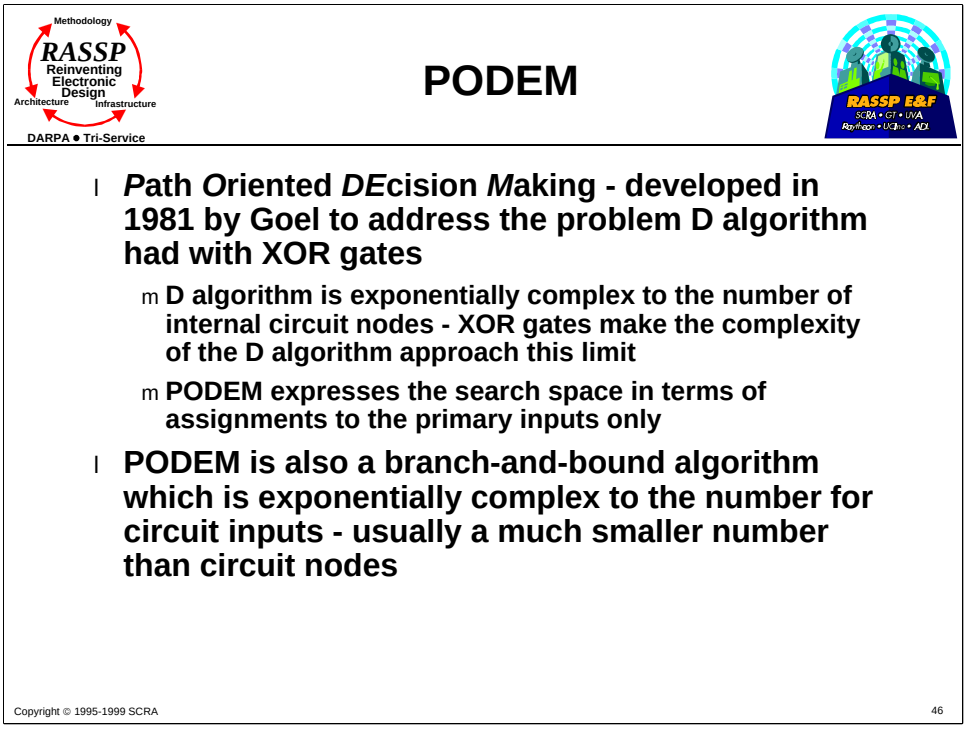


This slide is an example of the application of the D algorithm on a target fault, namely, J s-a-1. The process begins by picking all of the possible PDCFs for J s-a-1. This choice, like all in algorithmic ATPG, can be arbitrarily made or made with the assistance of some heuristic (testability measures). 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.

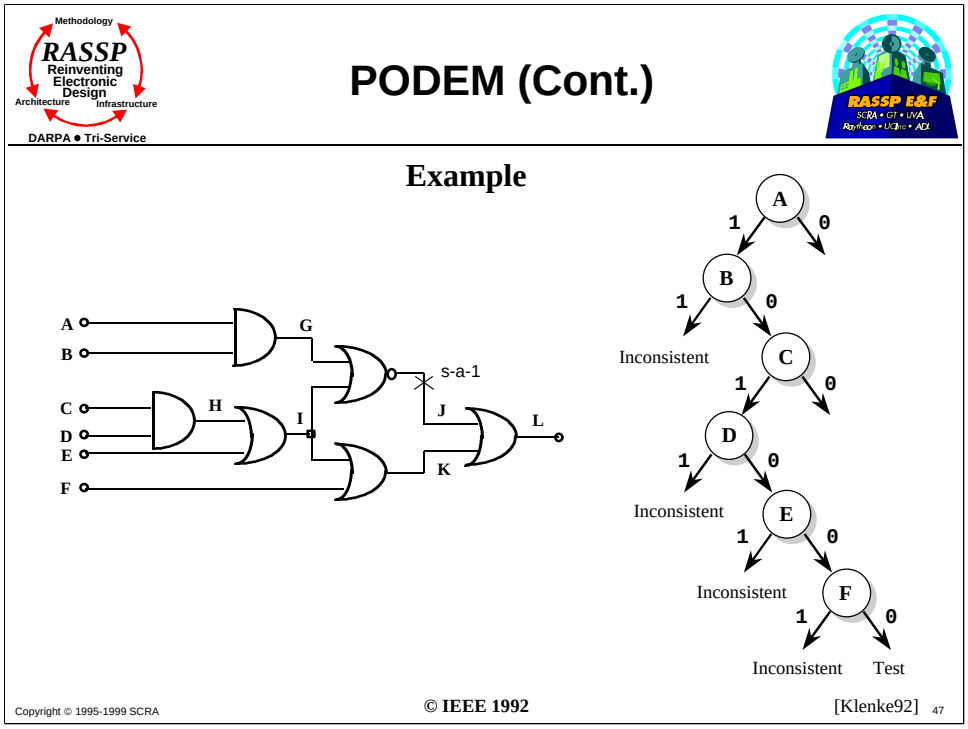
The second example shows what would happen if the alternate PDCF for J s-a-1 is selected. No test would be possible with this selection, and another solution (namely the first) would have to be tried.

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 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.

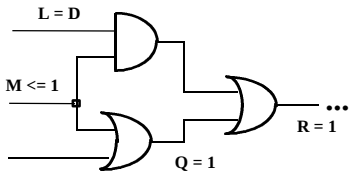
This example uses the simple heuristic of input/value selection of taking the inputs in order and always trying the "1" value first. PODEM in fact has some fairly complex heuristics and procedures that use circuit topology and current state information during the test generation process to select the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

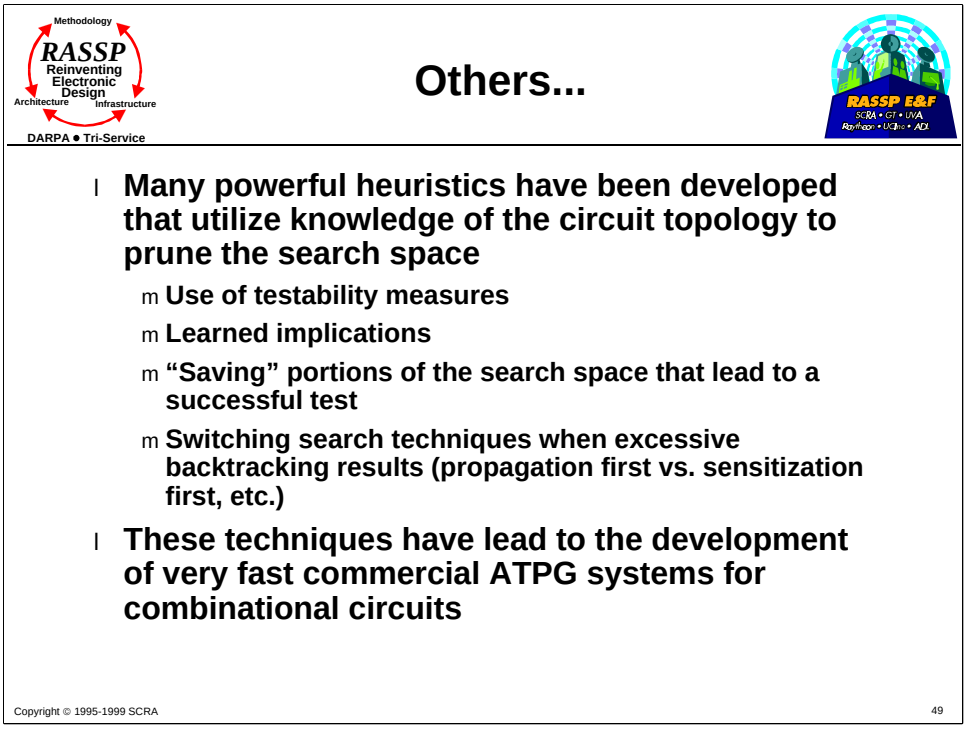
FAN and Related Algorithms

- **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:**

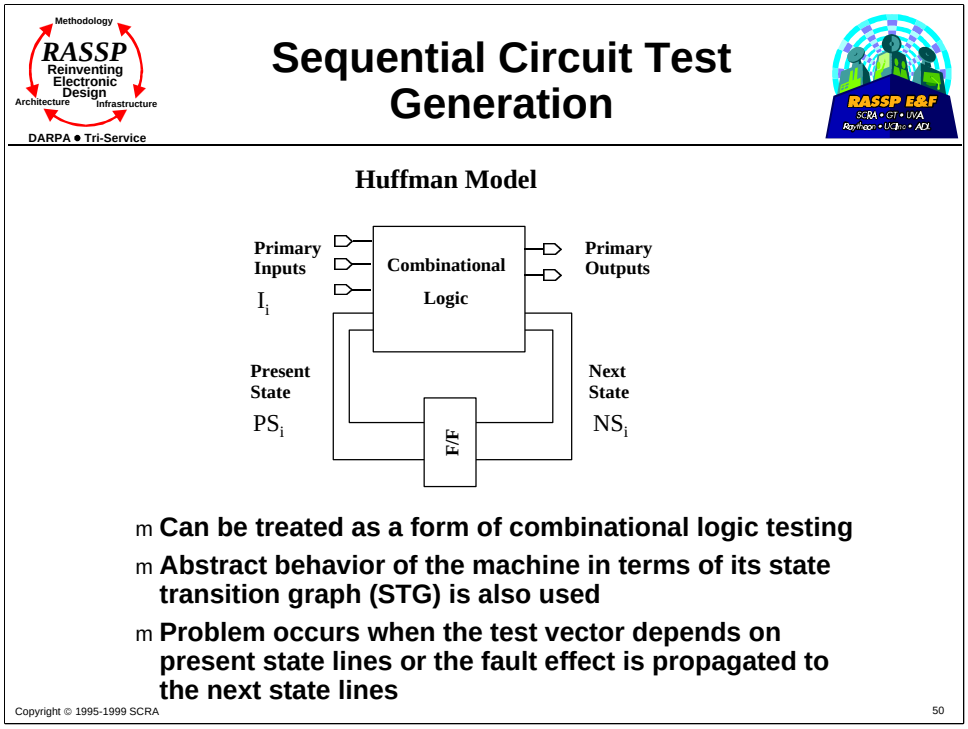


The requirement for M <= 1 to propagate the D value through the AND gate causes R to be set to 1 which terminates the propagation path

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.

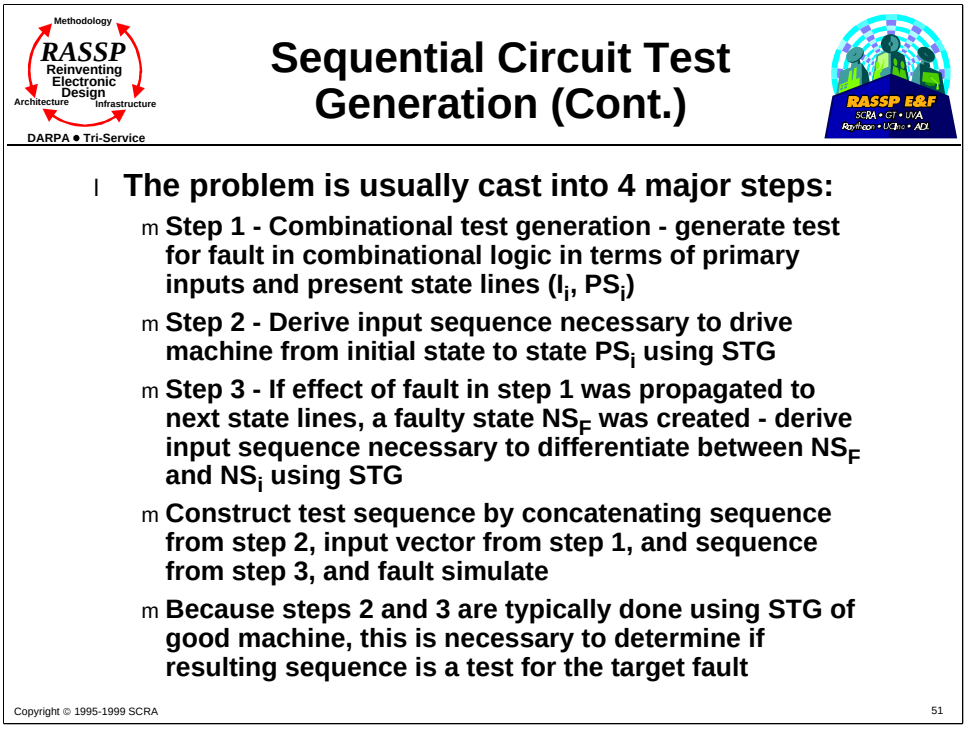


As stated previously, most of the follow-on work the PODEM-FAN has been in the area of improved heuristics and early detection of conflicts during test generation. The bottom line is that very fast test generators for combinational circuits are now available as part of commercial CAD packages (Mentor,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



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.

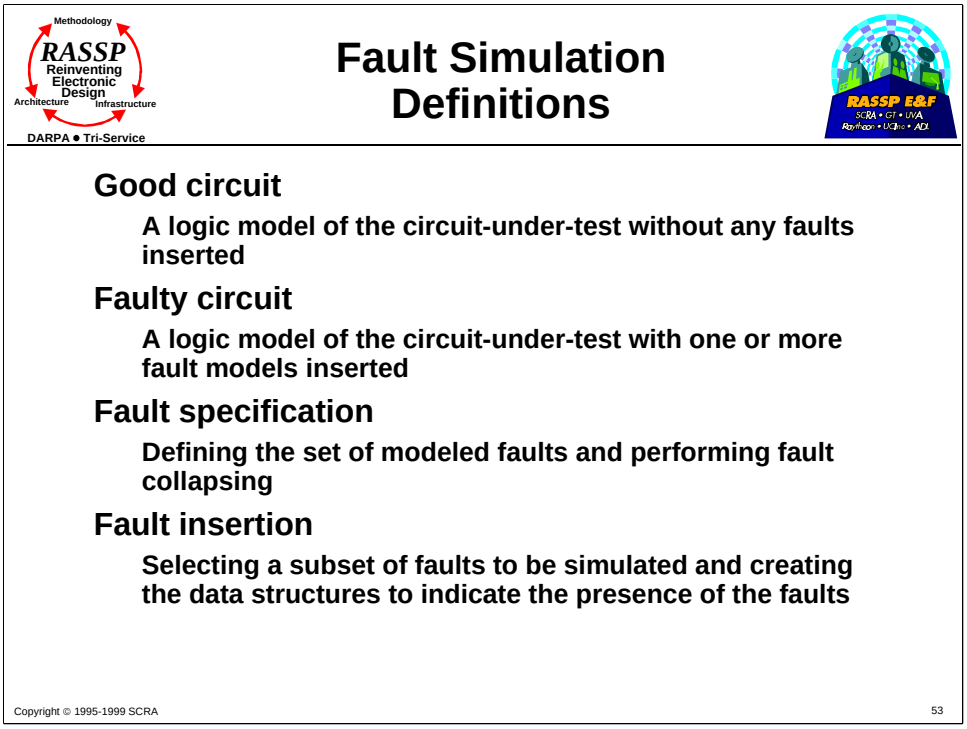
The problem with simply using combinational ATPG techniques on the combinational logic block is that a test may require a specific input combination on the Present State lines and the effect of the fault may be only propagated to the next state lines. In both cases, state machine analysis must be performed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



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



No additional notes on the definitions are required.

Ref: [Abramovici90]

Fault Simulation Definitions (Cont.)

Equivalent fault

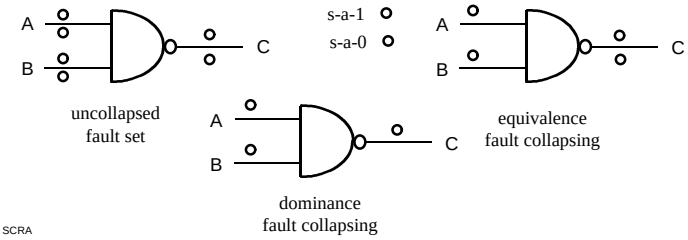
Two faults f_i and f_j are equivalent if there is no test that will distinguish between them

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



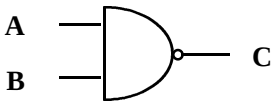
54

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, As-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



Good/Faulty Circuit Response

	f_0 = 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

Fault Table

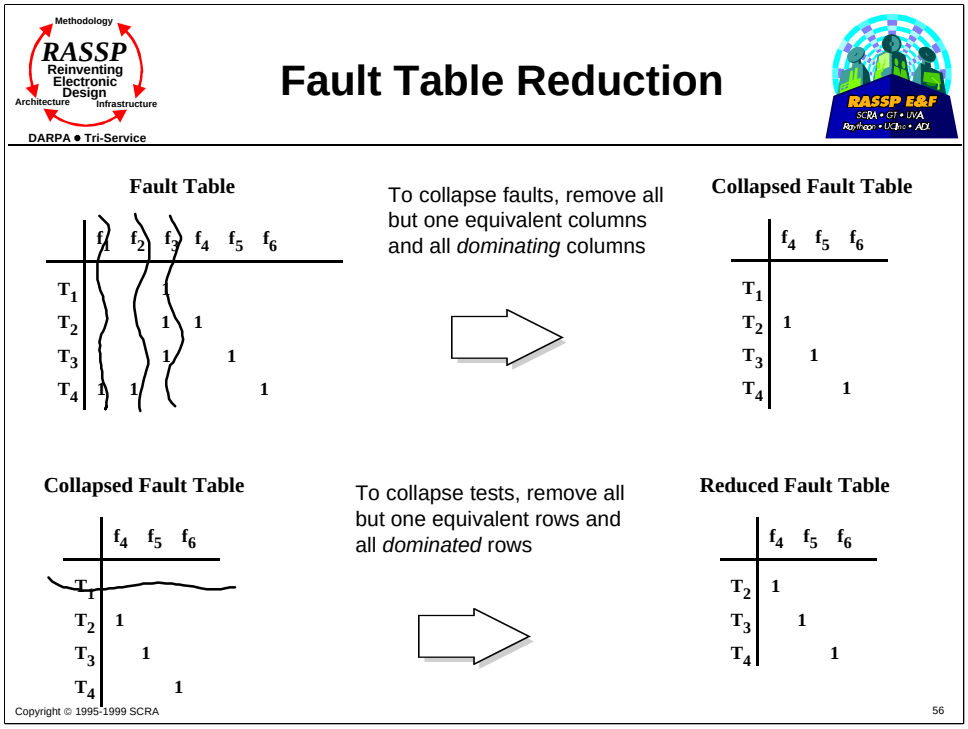
	f_1	f_2	f_3	f_4	f_5	f_6
T_1			1			
T_2			1	1		
T_3			1		1	
T_4	1	1				1

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

55

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

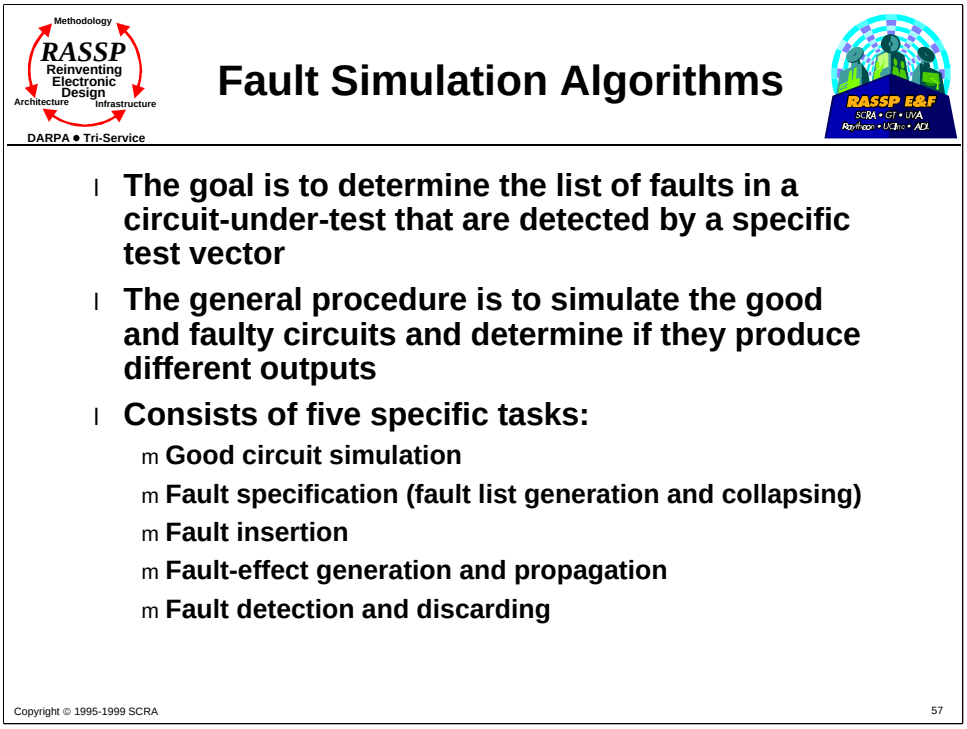
Notice that the equivalent and dominance relationship of the faults can be seen from the table. For example, f1, f2, and f6 are equivalent. Faults f4 and f5 dominate fault f3.



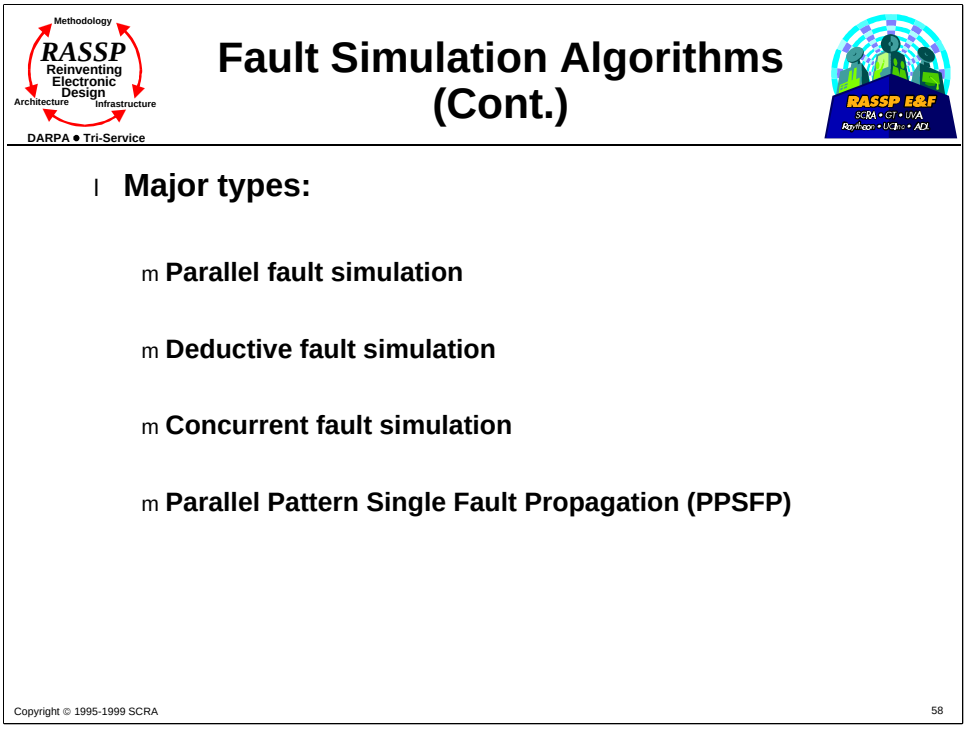
This slide shows the process of fault table collapsing.

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.

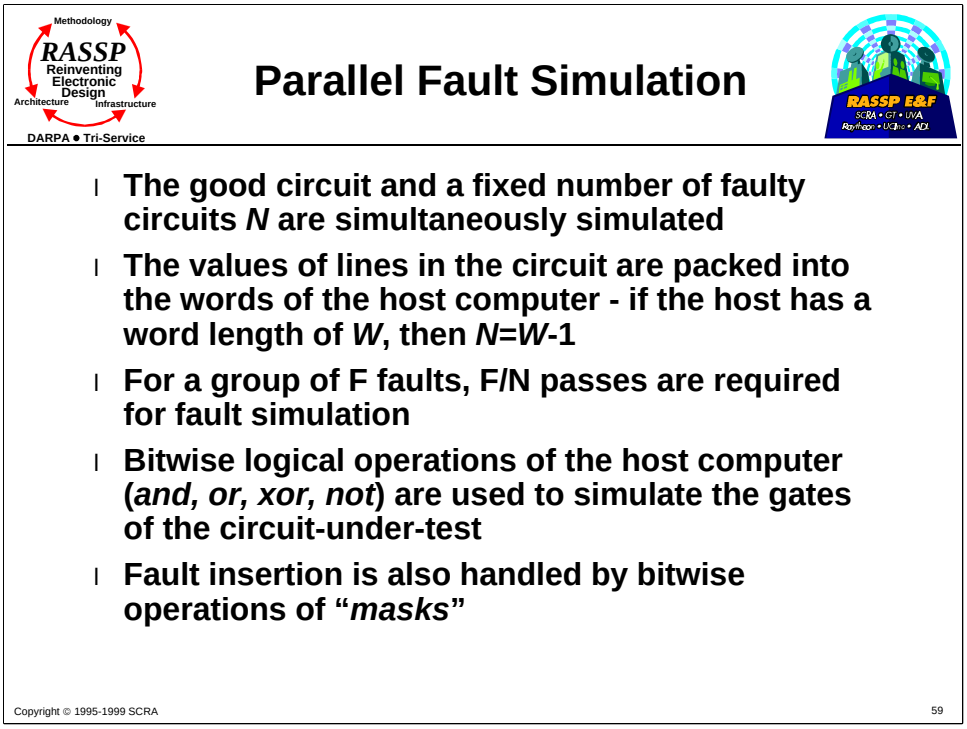
To collapse tests, remove all but one equivalent rows and remove all *dominated* rows.



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



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 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.)

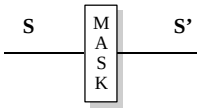
Define masks for signal *S* and fault in position *i* of word:

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

$fvalue(S)_i$ = 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:

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



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 ith 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 ith bit of S is "1" and the fvalue for S is “0”, then:

$$S' = \overline{"1" \cdot "1"} + "0" \cdot "1" = \text{“0”}$$

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

$$S' = \overline{"0" \cdot "1"} + "1" \cdot "1" = \text{“1”}$$

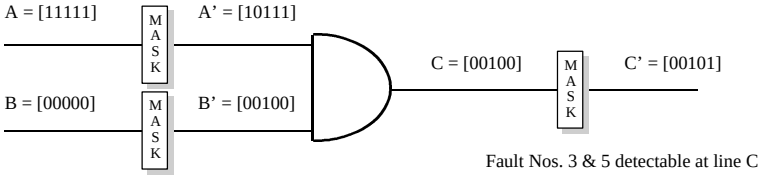
Ref: [Fujiwara85]

Parallel Fault Simulation (Cont.)

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]

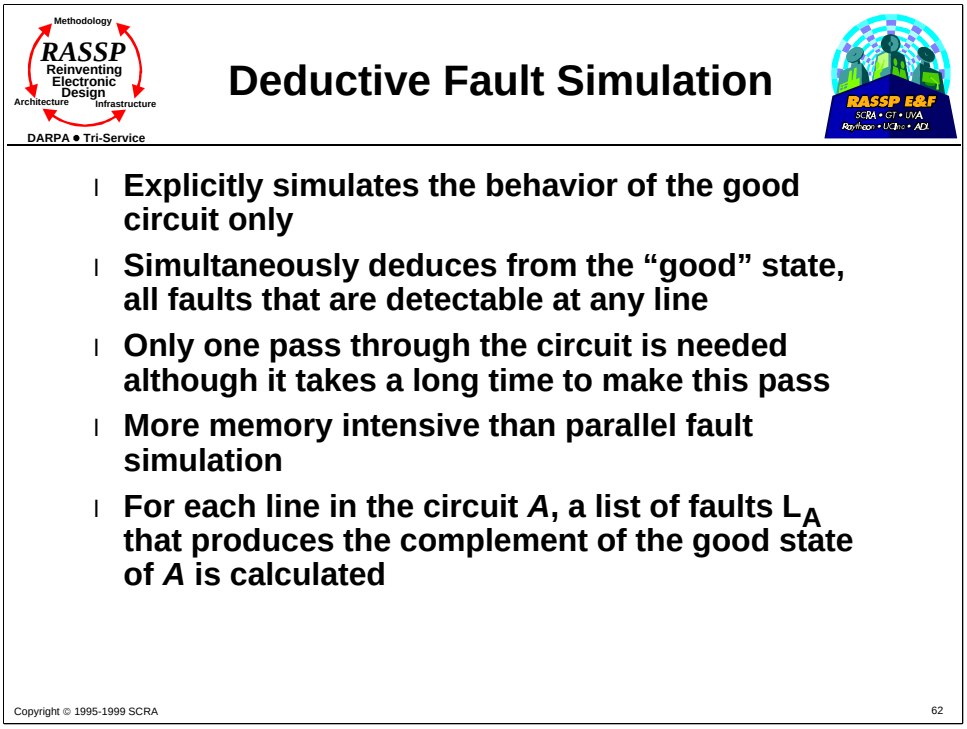


Fault Nos. 3 & 5 detectable at line C

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]

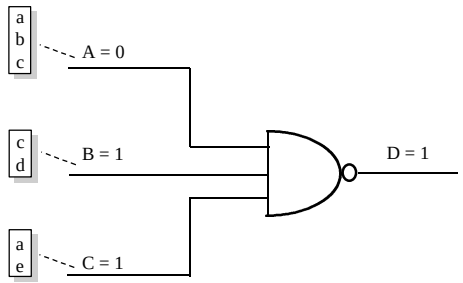


Deductive fault simulation is another algorithm that appears to be more efficient than parallel fault simulation in terms of run time, but is much less efficient than parallel in terms of memory usage.

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]

Deductive Fault Simulation (Cont.)

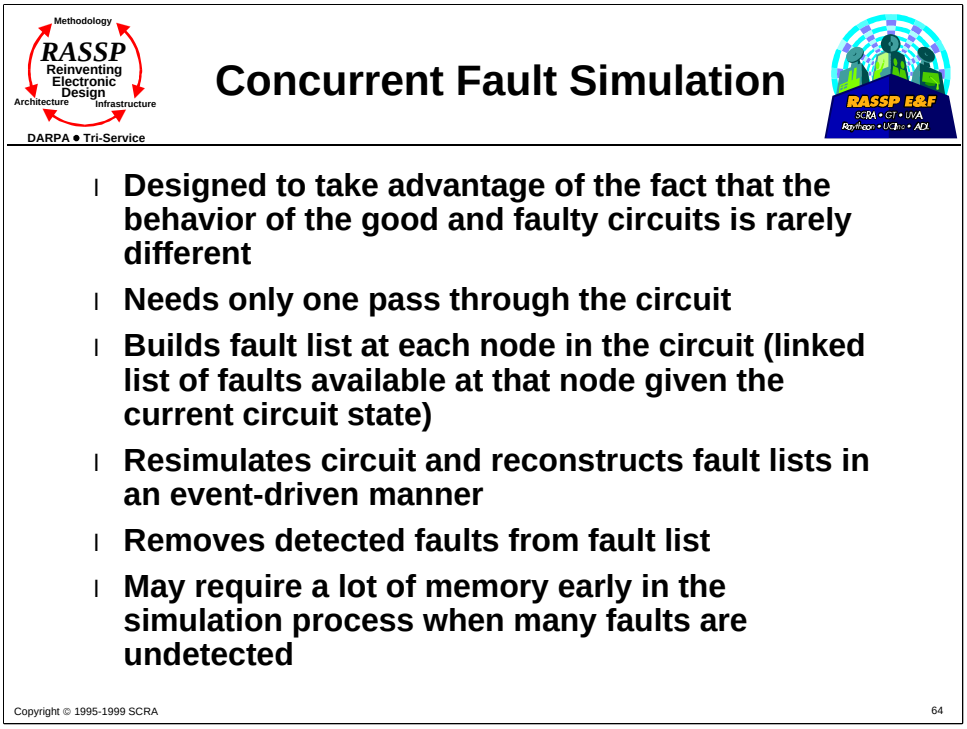


$$L_D = L_A \cap \overline{(L_B \cup L_C)} \cup \{D/0\} = \{b, D/0\}$$

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.

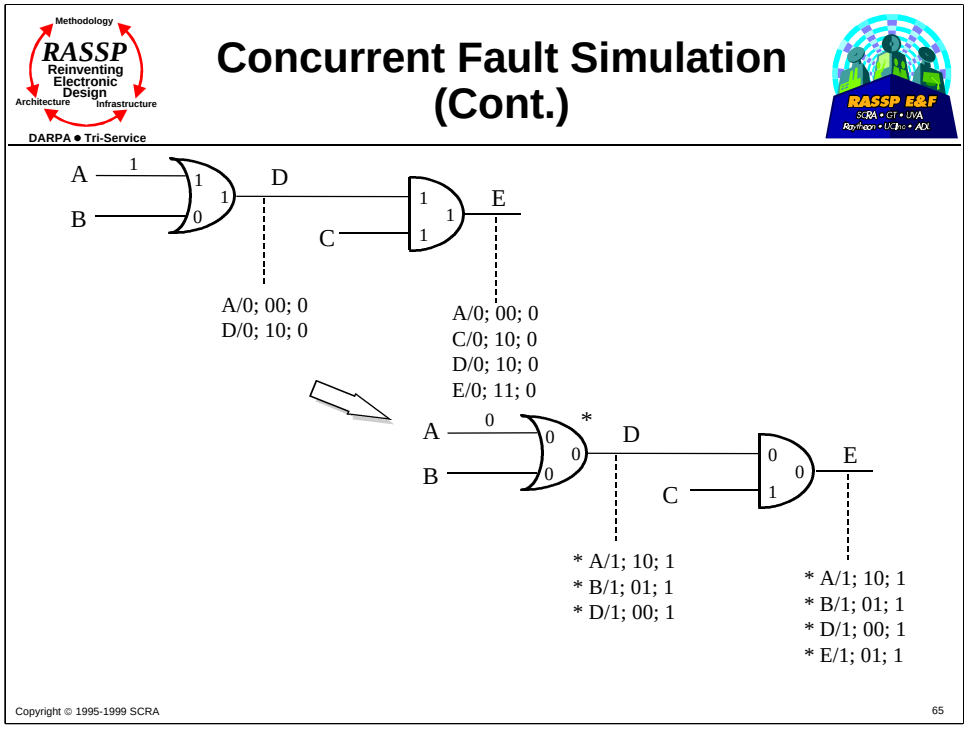
In this case, only faults that cause the values on input A to change (to “1”) without causing the values of inputs B or C to change (to “0”), will cause the output to change. Also, the faults within the logic element itself that cause the output to change must be considered. Thus, the list of faults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

Ref: [Fujiwara85]



Concurrent fault simulation 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.

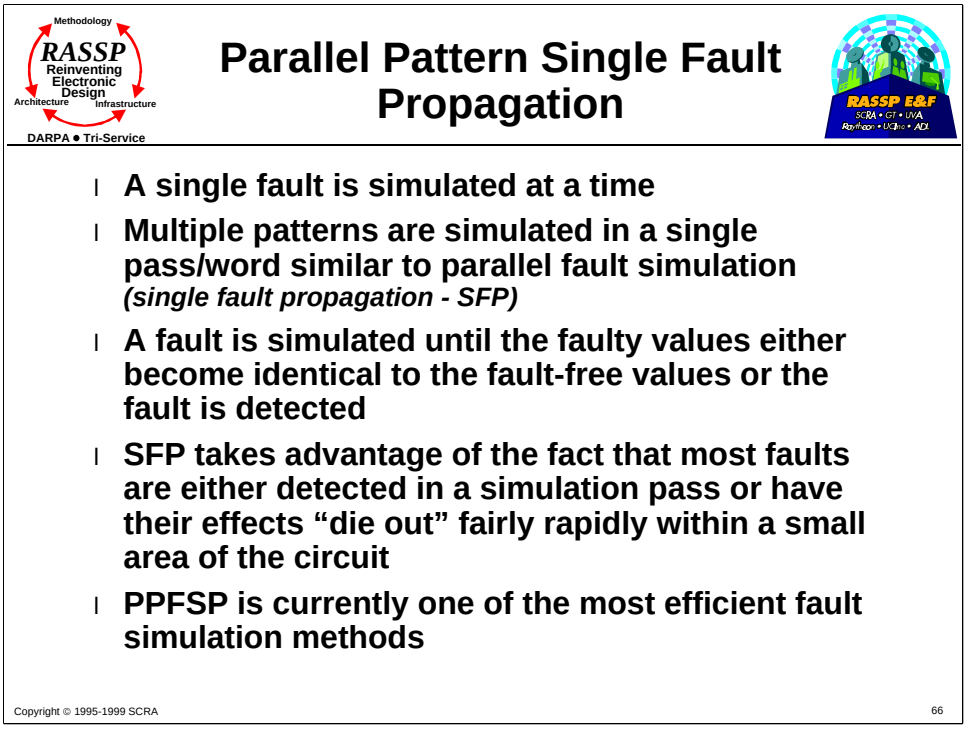
Concurrent fault simulation maintains a linked list of the “faulty circuits” at each line in the circuit. Only the circuits that do not agree, in terms of their input and output relationships, are explicitly simulated.



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.

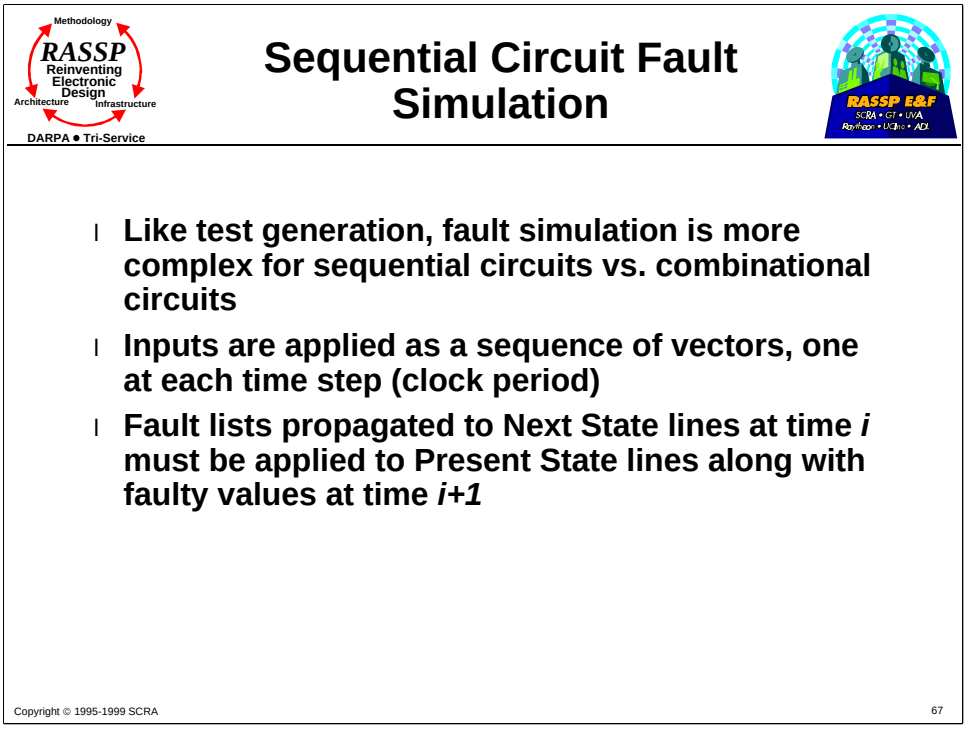
Ref [Fujiwara85]



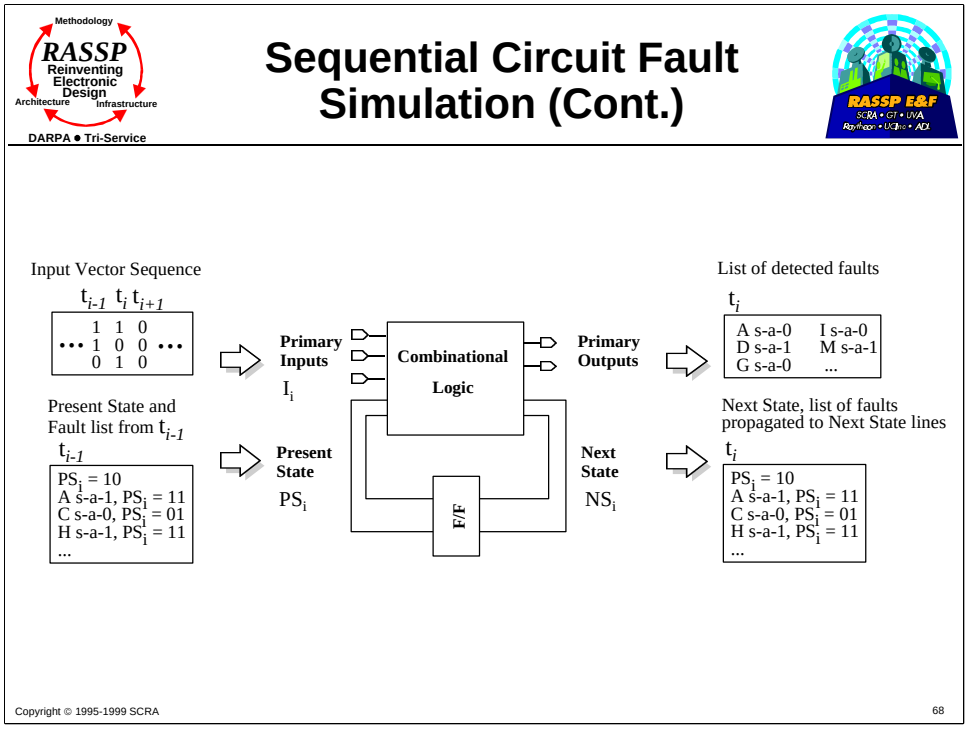
Parallel Pattern Single Fault Propagation is a fault simulation technique that combines two innovations. First, instead of simulating faults in parallel across a word as in parallel fault simulation, PPSFP simulates N patterns (when possible) across a computer word, on one fault at a time.

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]



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

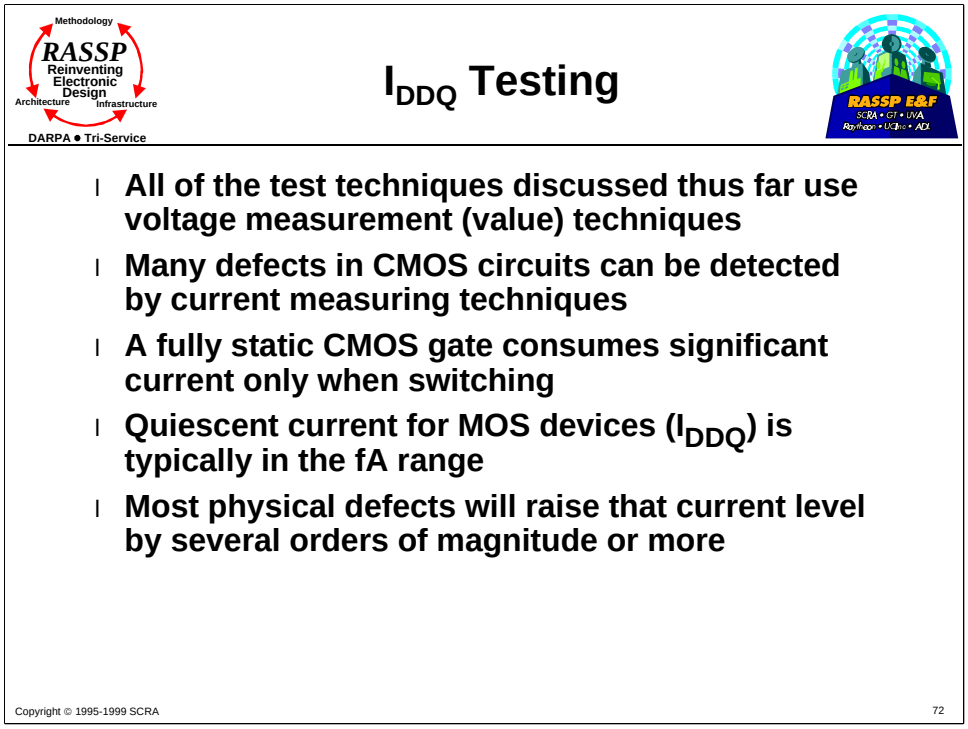
- Introduction (Part 1)
- Fault Modeling
- Test Generation
- Automatic Test Pattern Generation Algorithms
- Fault Simulation Algorithms
- **Introduction (Part 2)**
- I_{DDQ} Testing
- Design for Testability Techniques
- Built-In Self Test
- Hierarchical Design for Test
- Synthesis for Test
- DFT Standards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

Introduction

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

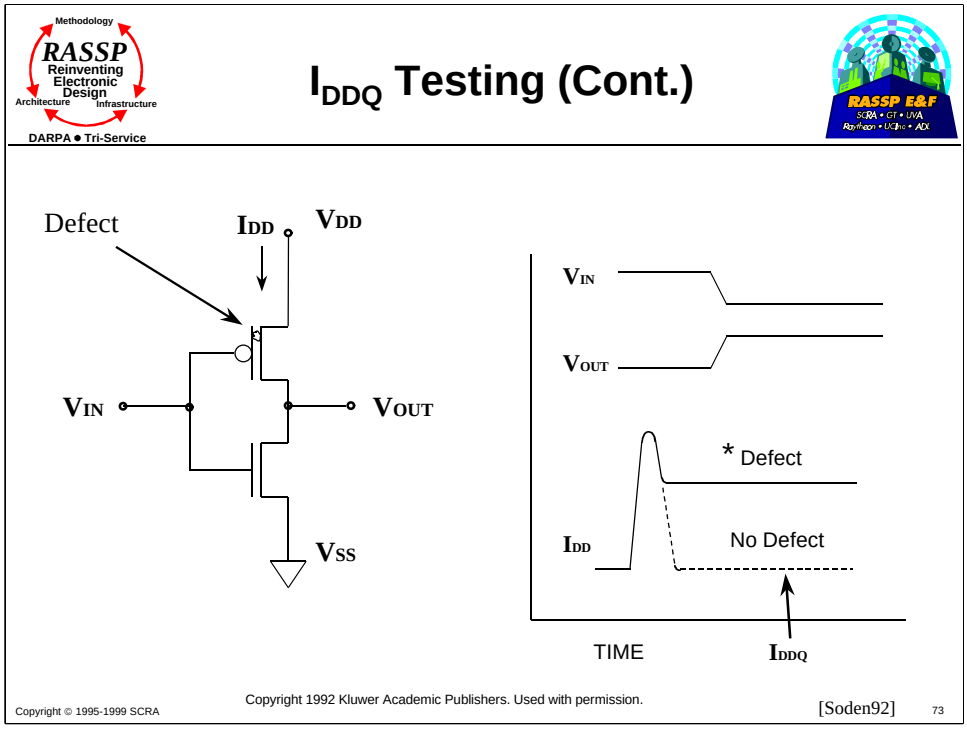
Module Outline

- 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



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.

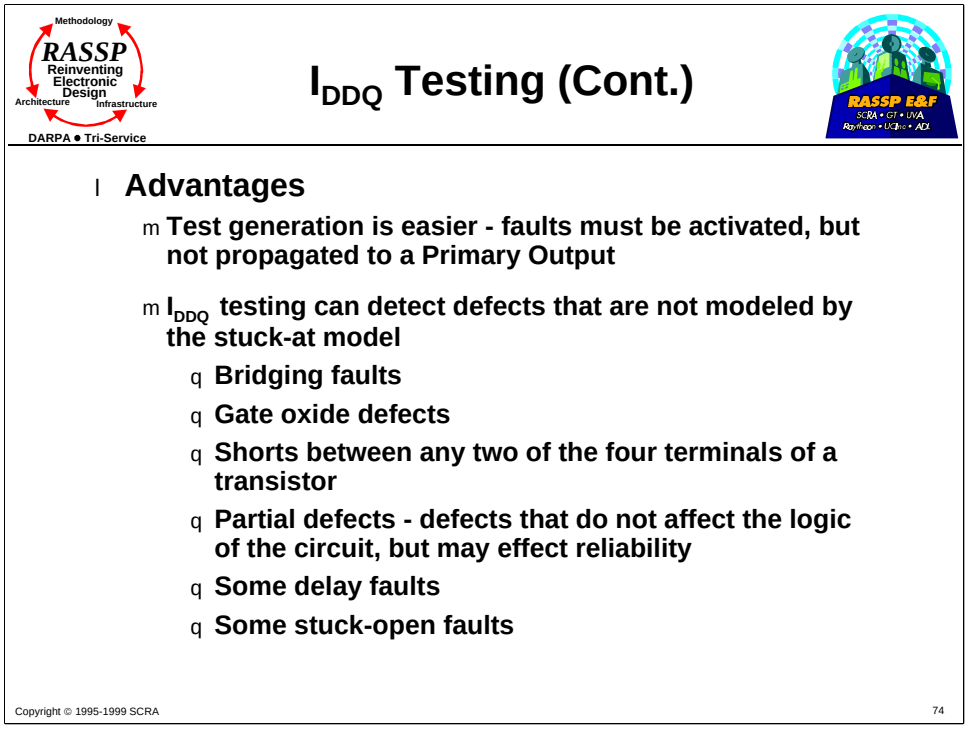
Ref: [Soden92]



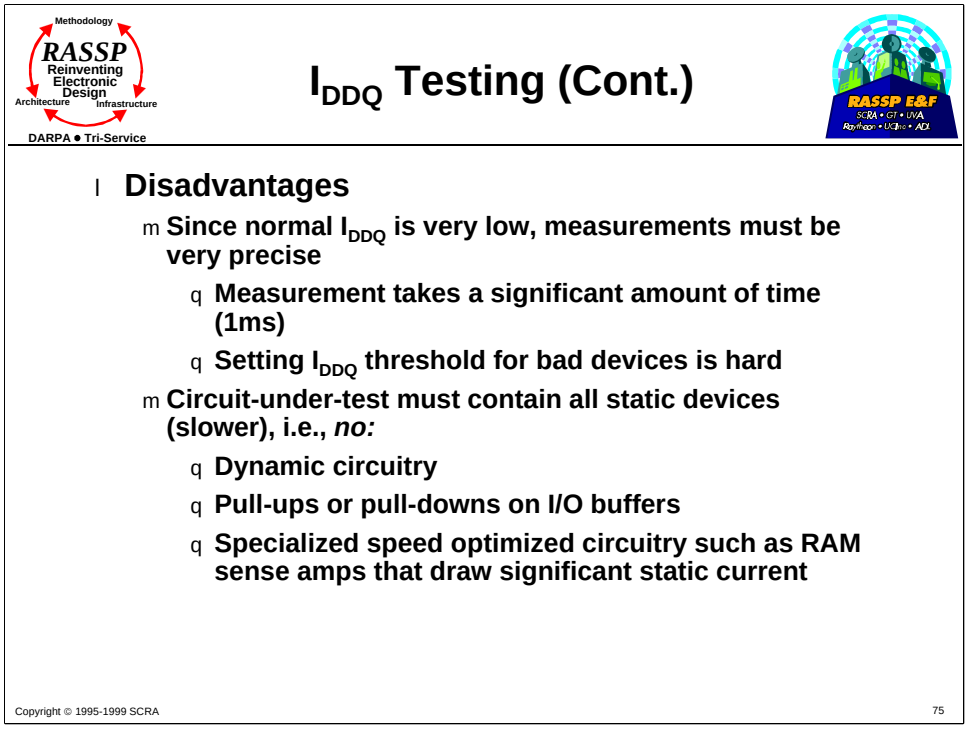
This figure shows the effect of a defect in terms of a gate-source short in the P transistor of an inverter. When the input voltage goes low such that this transistor turns on, significant current flows between the source and gate such that IDDQ increases dramatically. However, because the gate output 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.

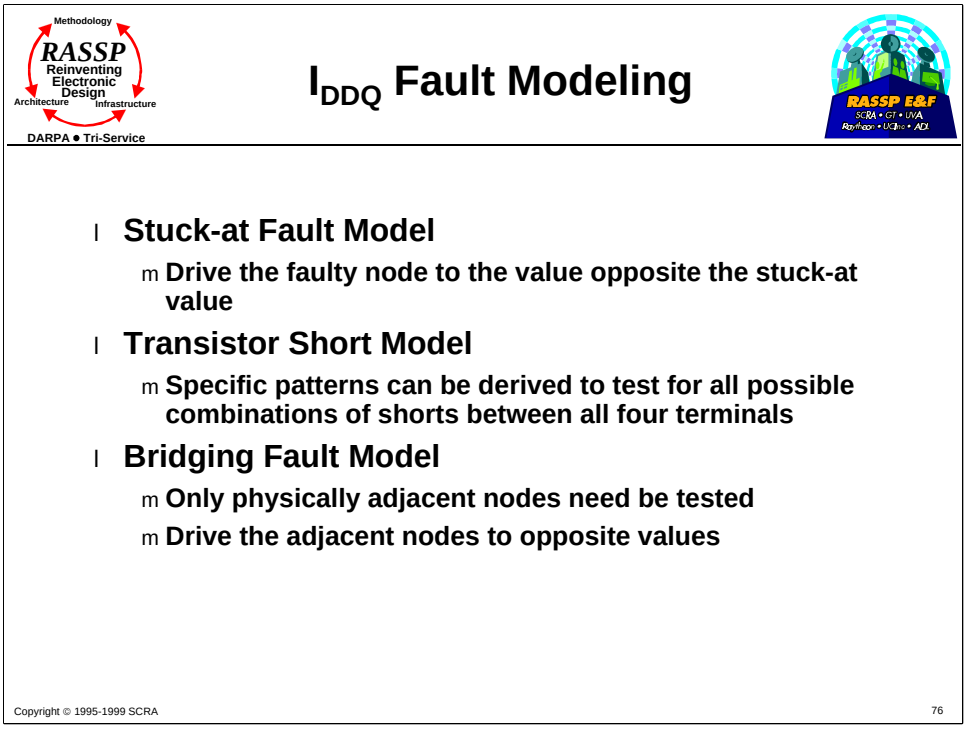
Ref: [Soden92]



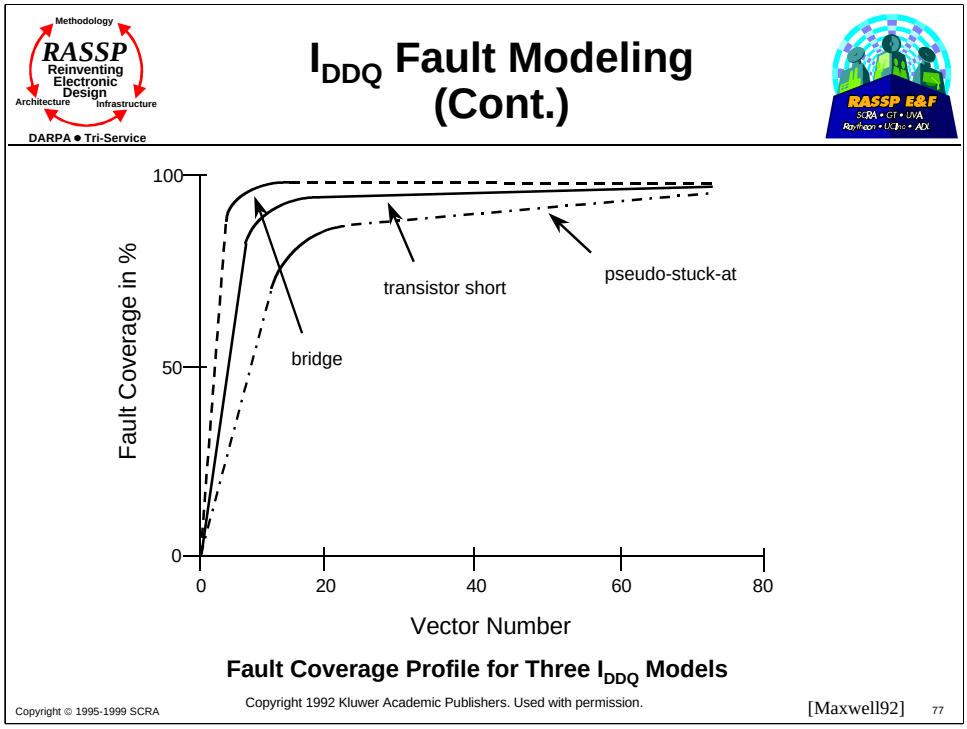
The major advantages of IDDQ testing include the fact that it can potentially detect faults that are undetectable by other models. Also, test generation can be easier because the fault only has to be activated (sensitized) and it will be detected. The fault effect (value) doesn't need to be propagated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



The major disadvantage of IDDQ 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 IDDQ testing, certain restrictions must be placed on the design.

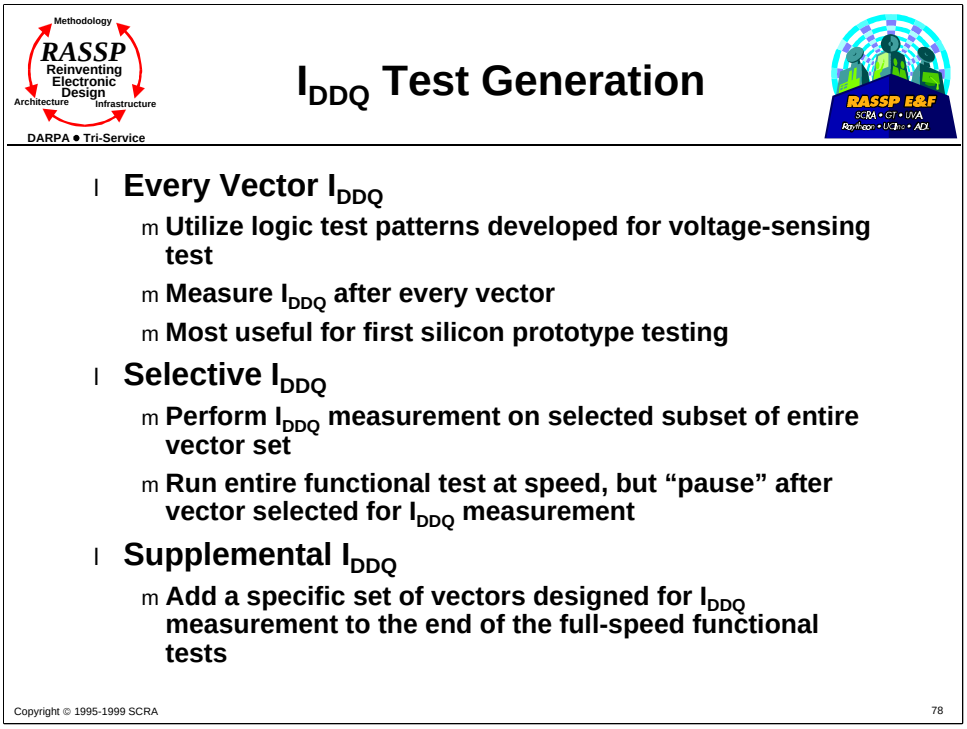


There are several fault models that can be used for IDDQ test generation. All of them operate at the logic or transistor level.

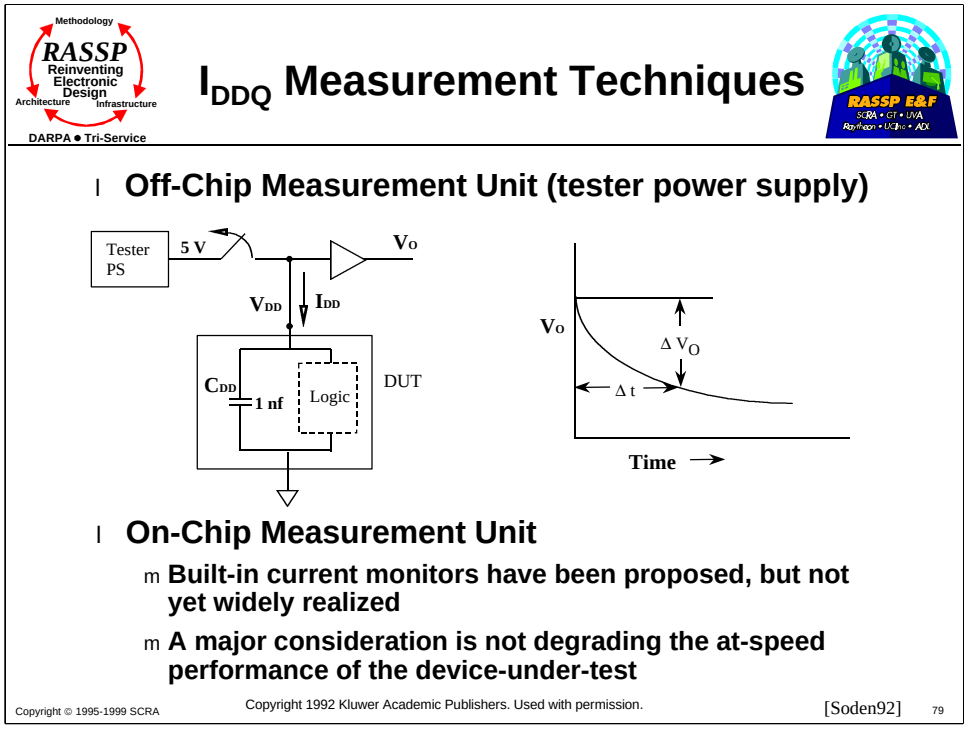


This graph shows the fault coverage vs number of test vectors for three IDDQ models. Note that for bridging faults and transistor shorts, very high fault coverage is obtained for a very few vectors. This doesn't mean that this necessarily applies for defect coverage.

Ref [Maxwell92]



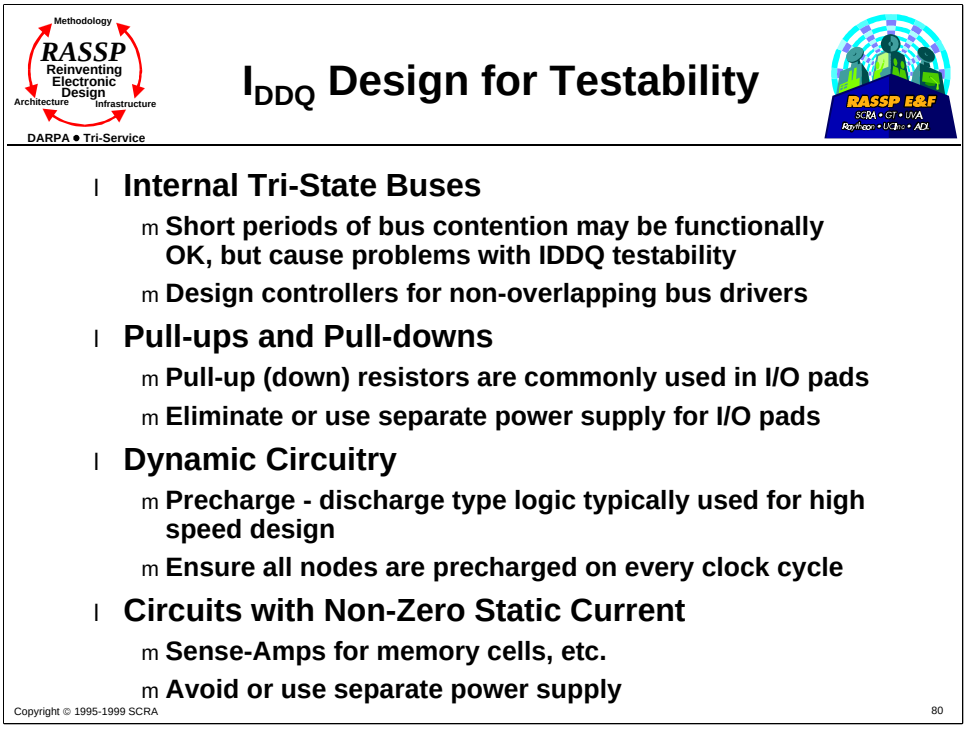
There are several different methods that can be used to develop IDDQ tests. Most are used in conjunction with voltage (value) testing for the best speed/quality tradeoff.



As noted before, IDDQ 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 IDDQ. A device with an IDDQ-detectable fault will discharge faster. This RC time constant measuring method is the main reason IDDQ testing takes so long.

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

Ref: [Soden92]



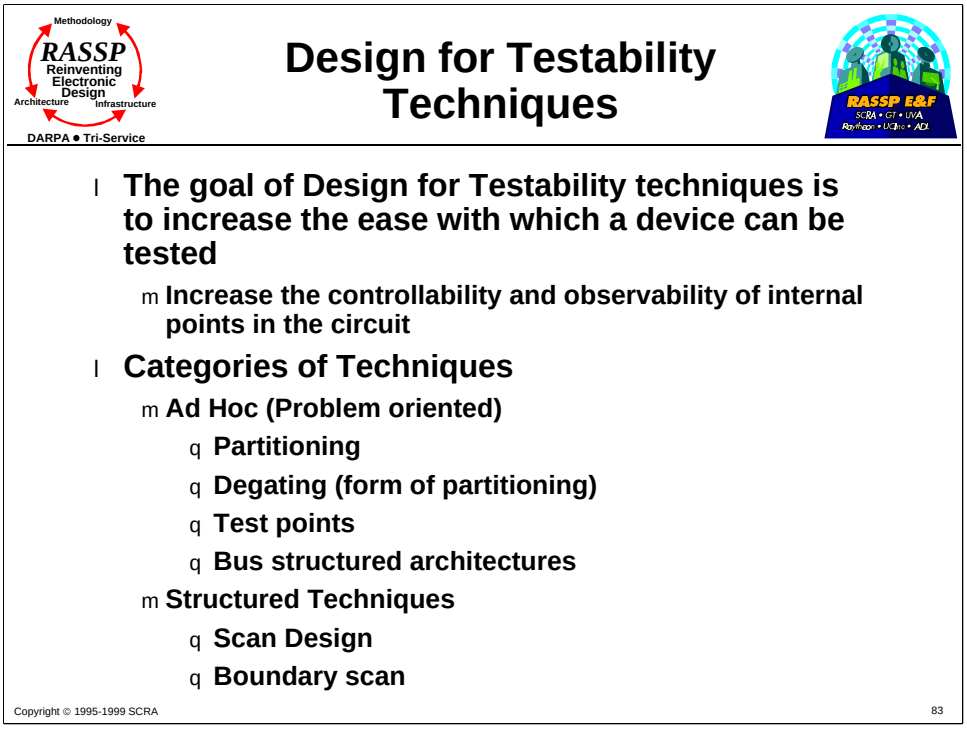
In order to ensure/improve IDDQ testability, several design constraints must be applied to limit good circuit IDDQ.

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

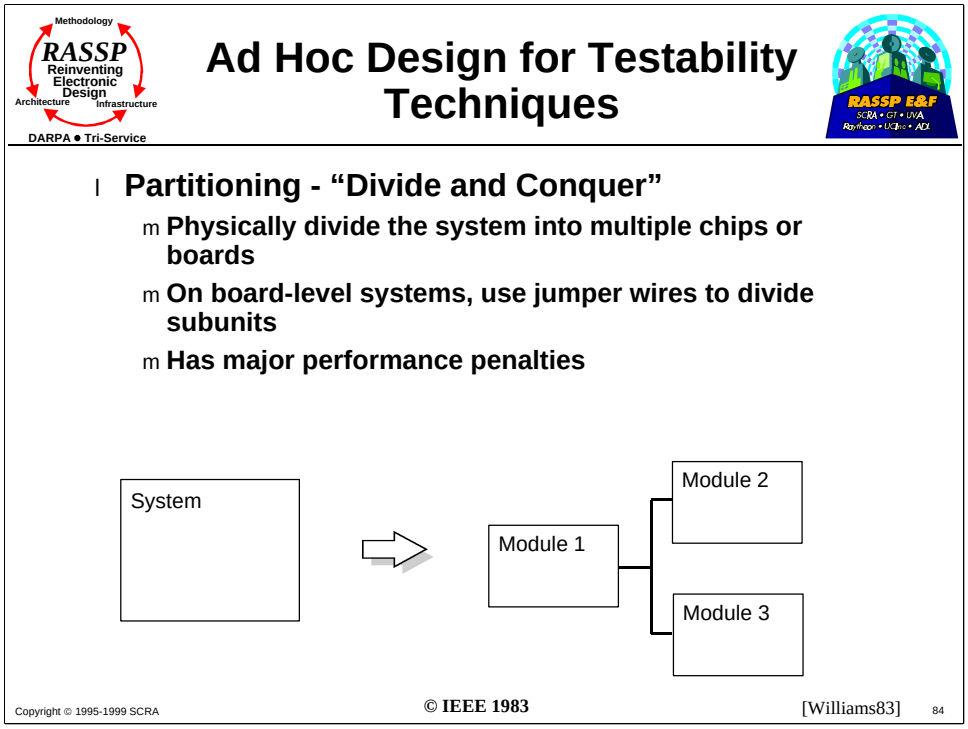
Section Outline

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

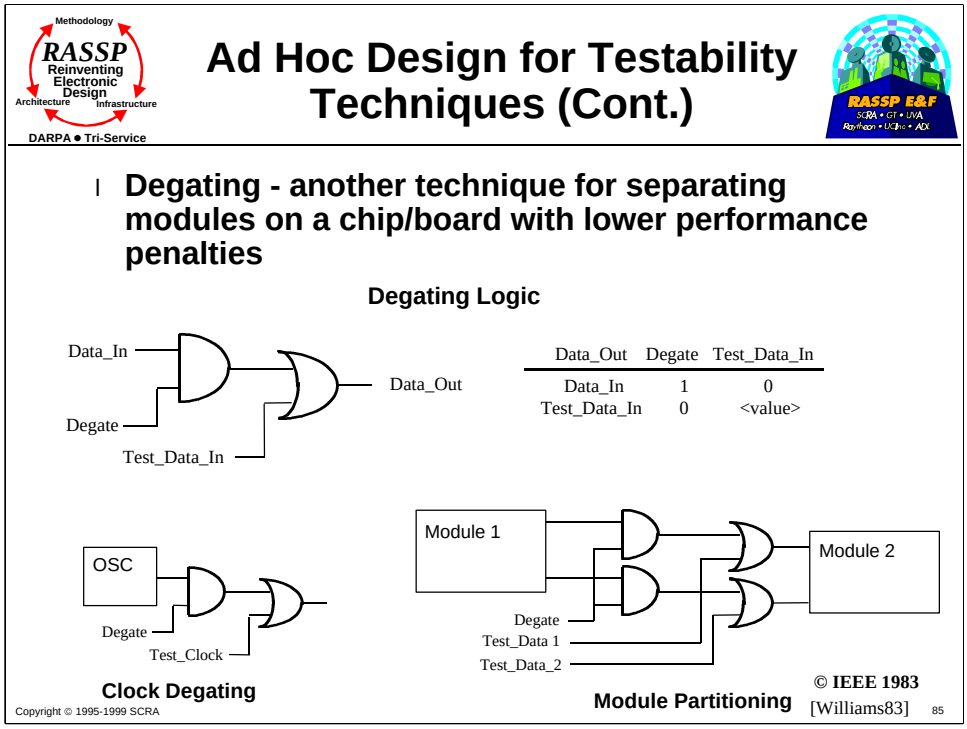


Three major categories of Design for test have been developed. BIST is a category of DFT because, obviously, the most testable chip is one that tests itself. However, it is such a big topic that we will cover it in its own section.

Ref: [Williams83]



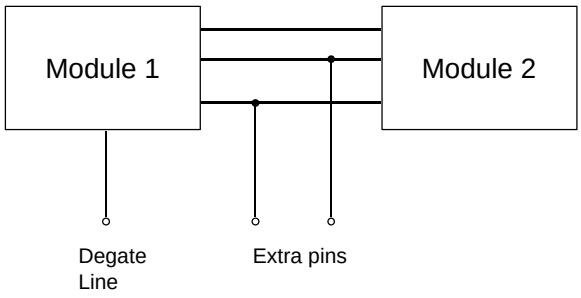
Obviously, anytime you can partition a system into subsystems (physically), the testability is improved. The major problem with this technique is the major performance penalty it incurs.



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.

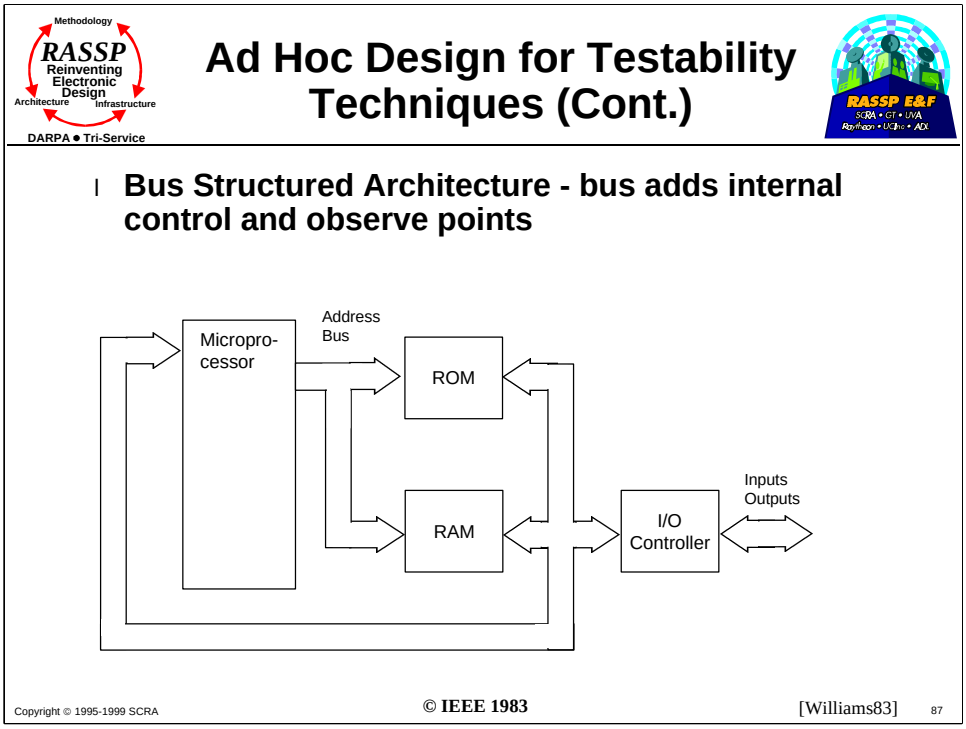
Ad Hoc Design for Testability Techniques (Cont.)

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

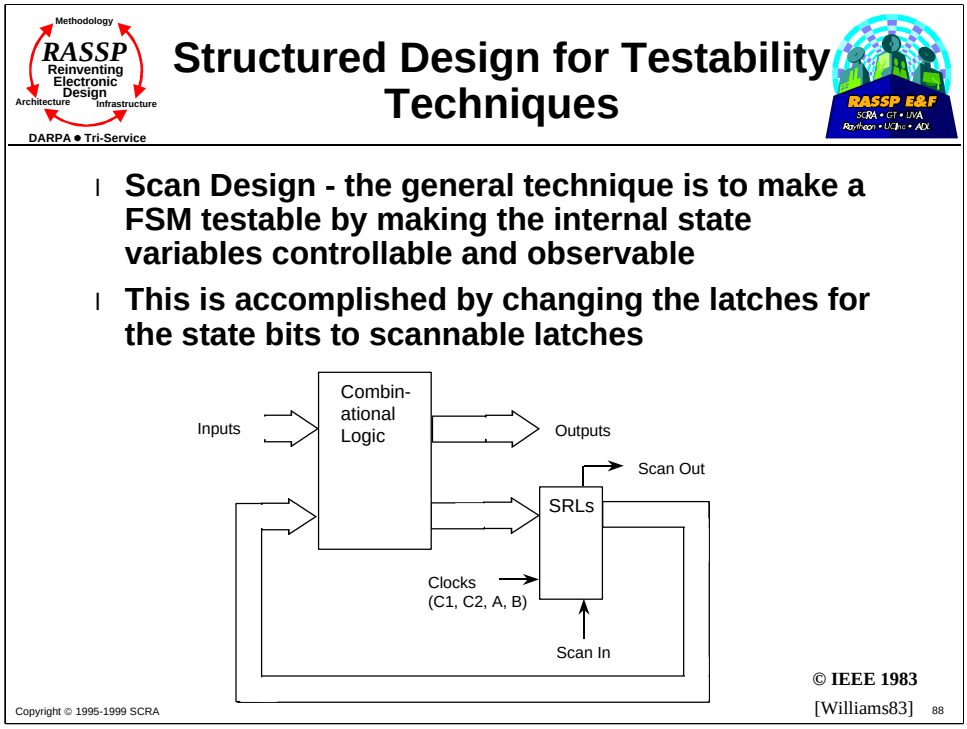


[Williams83]

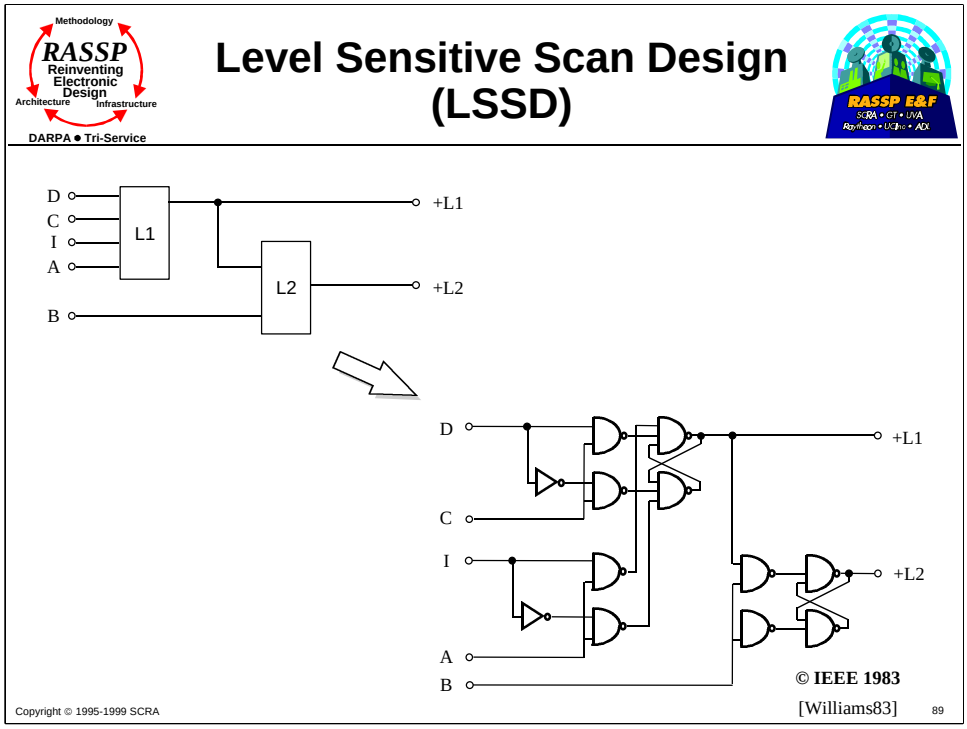
The test points approach is similar to degating except that the external lines are used to both observe (in normal operation) and drive the test nodes. In this case, the degating signal must be able to tri-state the internal nodes (as on an internal bus) so that they can be driven by the external test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



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

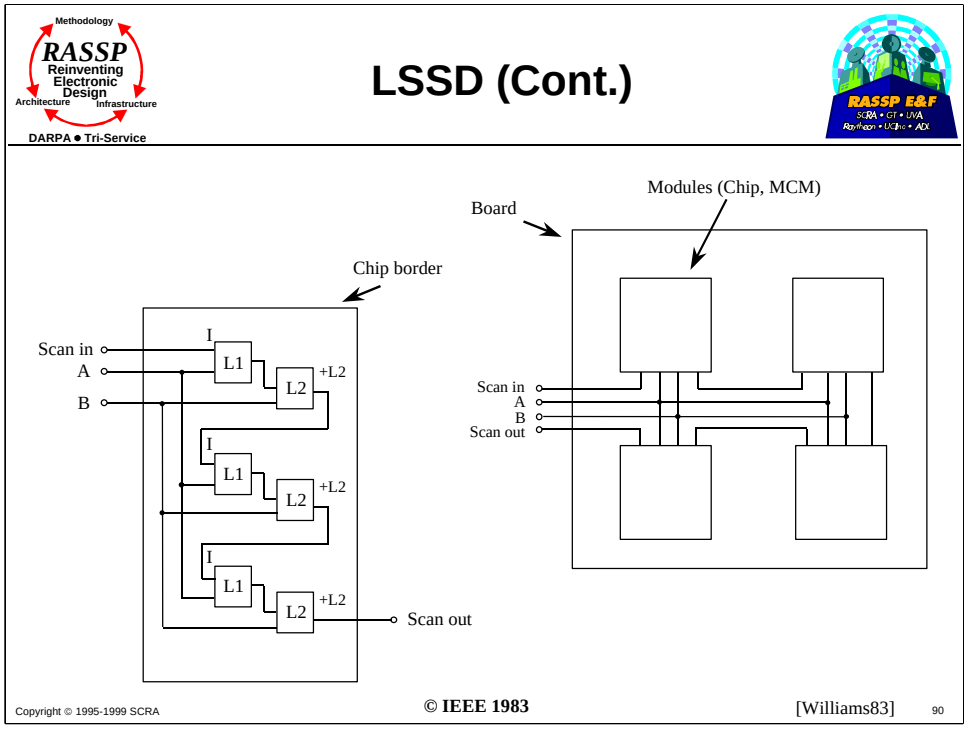


Scan design is the most common structured design for testability technique. In it, all of the latches in the design are made externally controllable and observable. Therefore, the testing problem becomes one of combinational logic testing only.

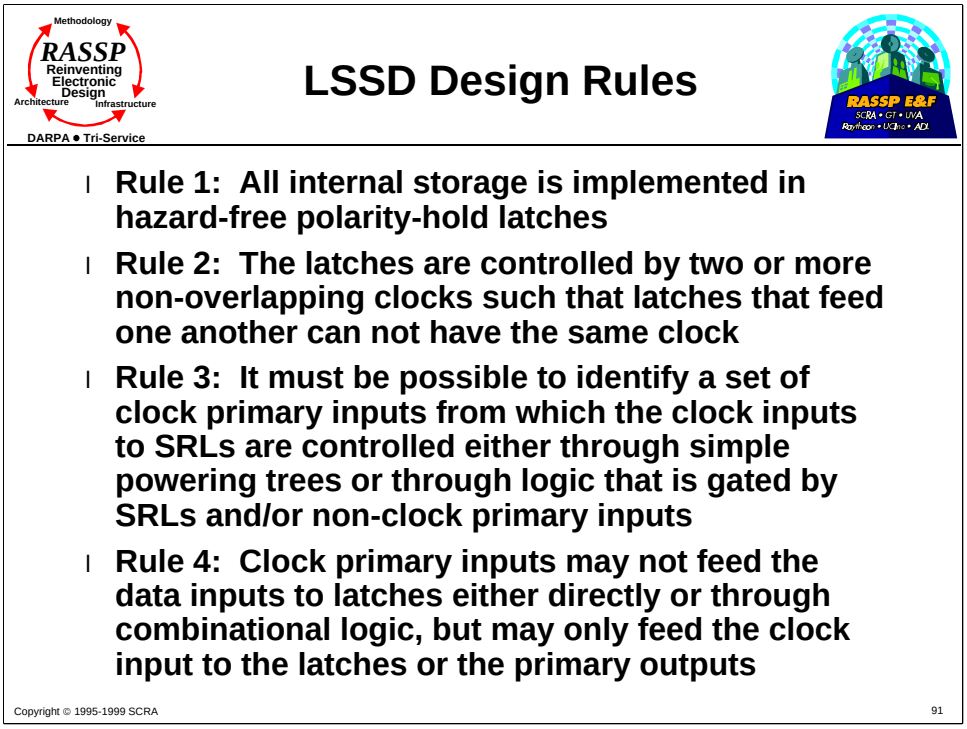


This figure shows the general configuration for of a level sensitive scan design latch as well as a NAND/NOT implementation.

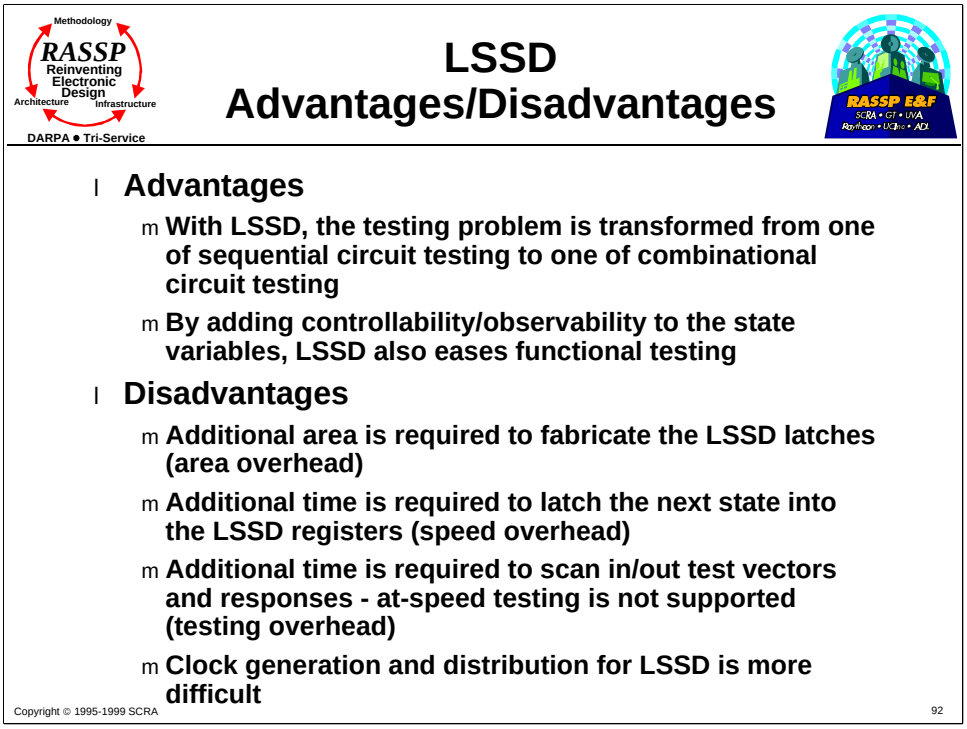
D is the normal data line and C is the normal clock line. Line L1 is the normal output. Lines I, A, B and L2 form the shift portion of the latch. I is the shift data in and L2 is the shift data out. A and B are the two phase, non overlapping shift clocks.



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.

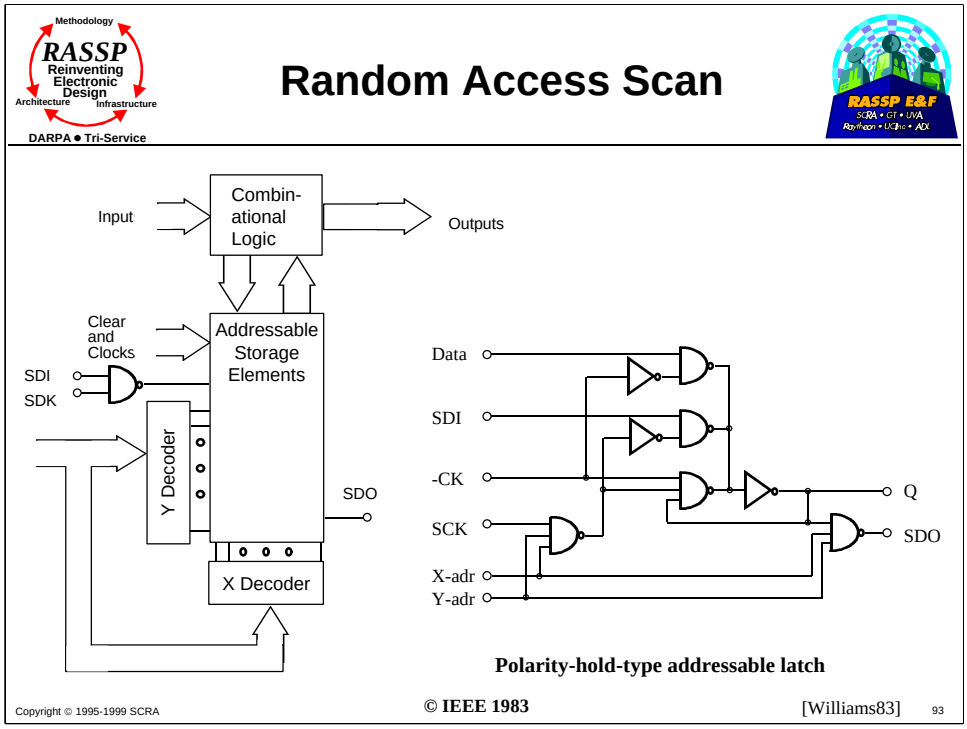


The use of LSSD requires the adherence to several design rules. A chip with only LSSD latches is not truly LSSD unless it adheres to these rules. The rules are designed to ensure that all internal nodes are controllable and observable via the shift registers.

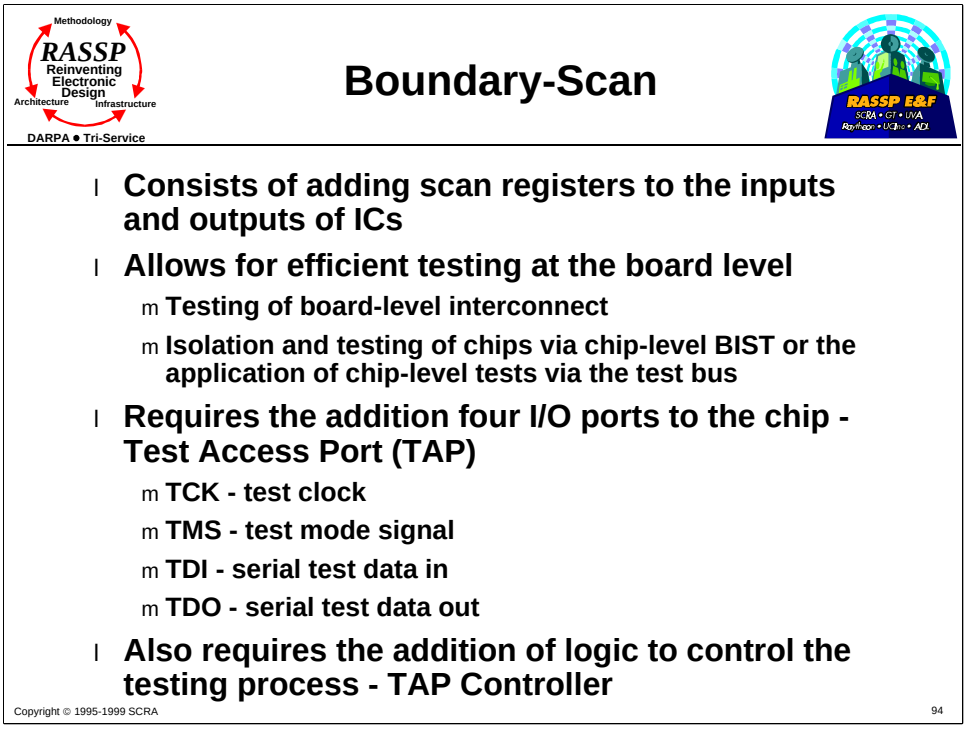


The major advantage of LSSD is that it transforms the testing problem from sequential to combinational testing, which is a much more tractable problem.

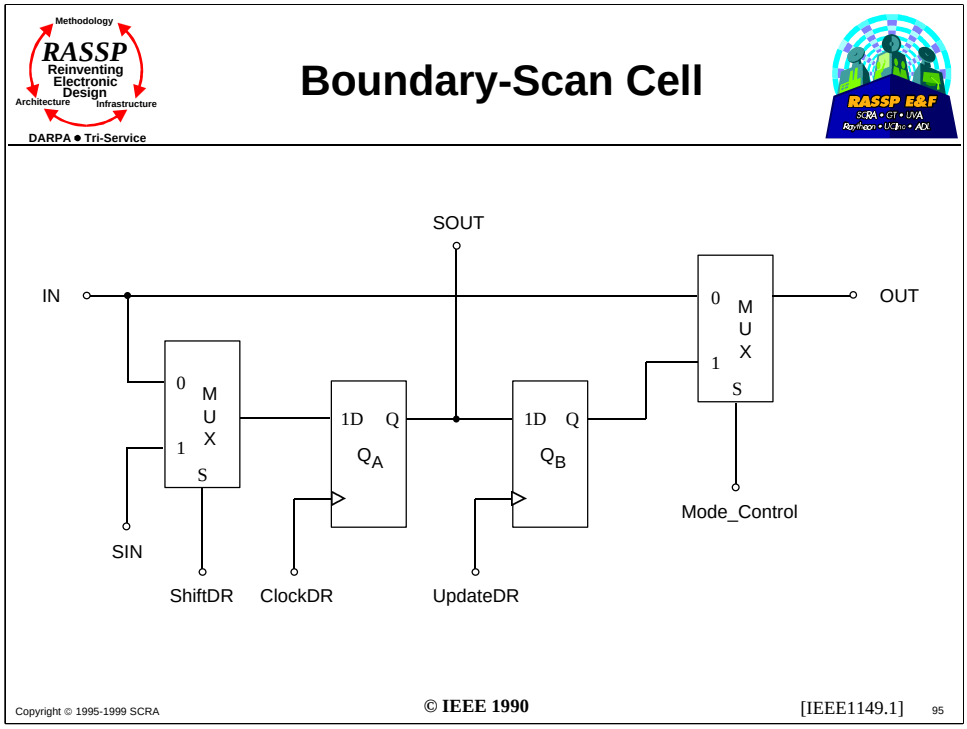
The major disadvantage of LSSD is probably the speed overhead because it adds several gate delays to the critical path of the design. The testing overhead can be a big problem too because some ASIC vendors charge by the clock cycle for test application.



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.

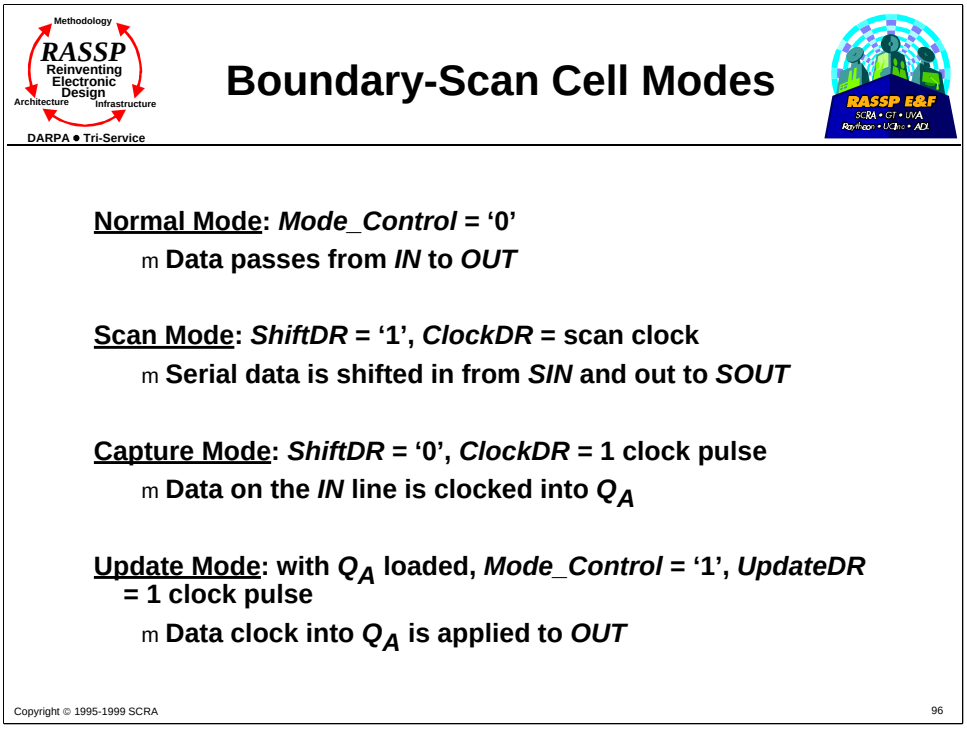


Boundary scan is probably the most widely used DFT technique. It has been adopted by the IEEE as a standard (to be discussed later). It requires the addition of some logic to the chip for control and some additional I/O ports, but the overhead is minimal.



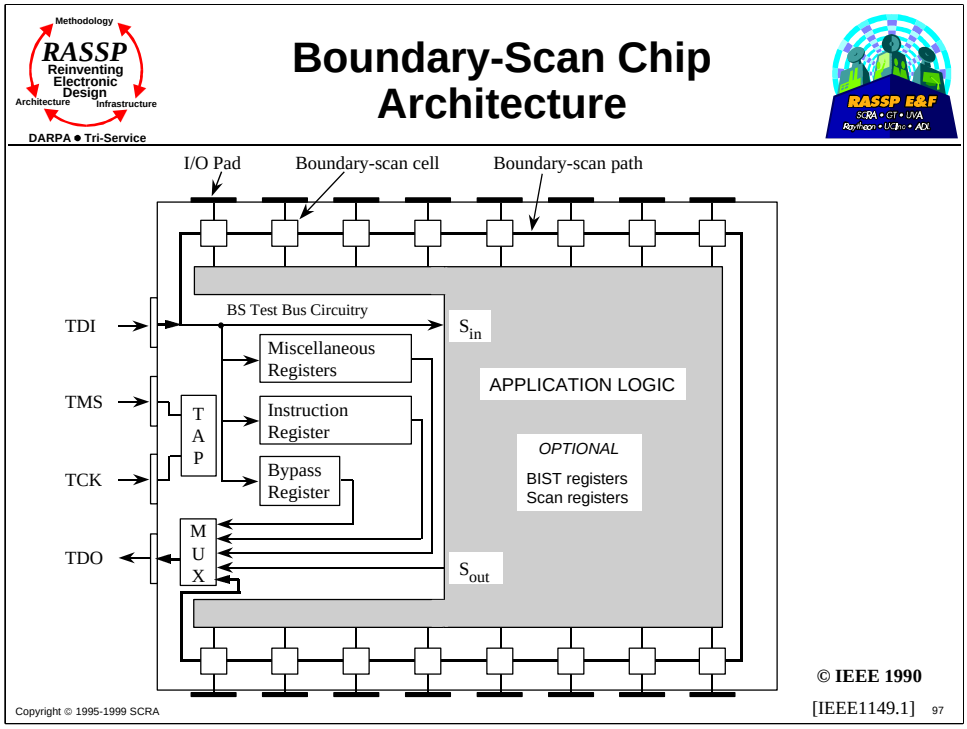
This figure show the basic structure of a boundary-scan cell. One of these cells is added to each I/O port on the chip. One reason that boundary scan may be popular is that the relative cost of adding the scan cells to the I/O pads is low compared to the cost of adding full scan. For example, only a 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]



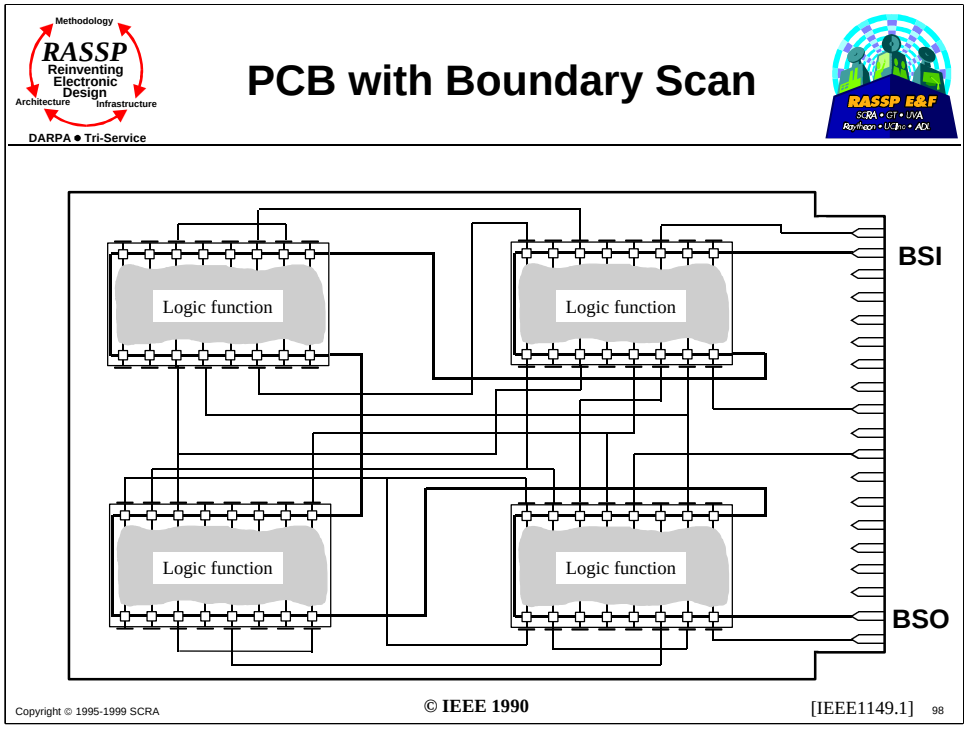
There are four major modes for the boundary-scan cell. The normal mode simply passes inputs to outputs. The scan mode passes data from SIN to Qa and from Qa to Sout. Capture mode loads the value on the input to Qa. Finally, update mode loads values from Qa to the output.

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.



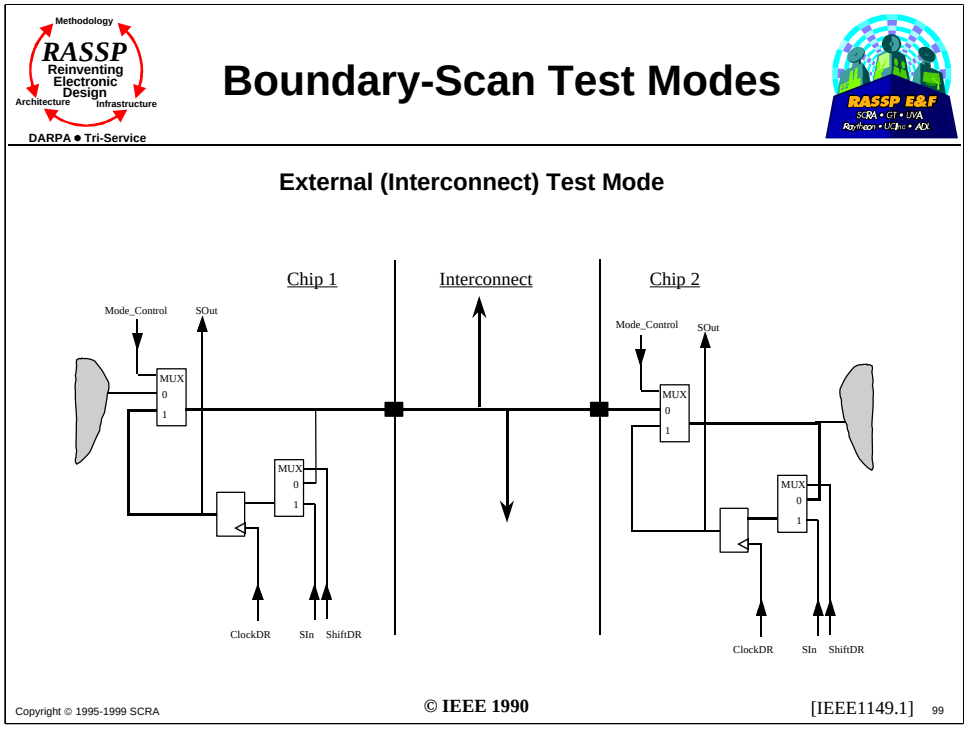
This figure shows the architecture of a boundary-scan-complaint chip. Note that the application logic of the chip itself may include DFT or BIST techniques (scan, BILBO, etc.) and this test logic is controlled by the boundary-scan TAP controller.

Ref: [IEEE1149.1]



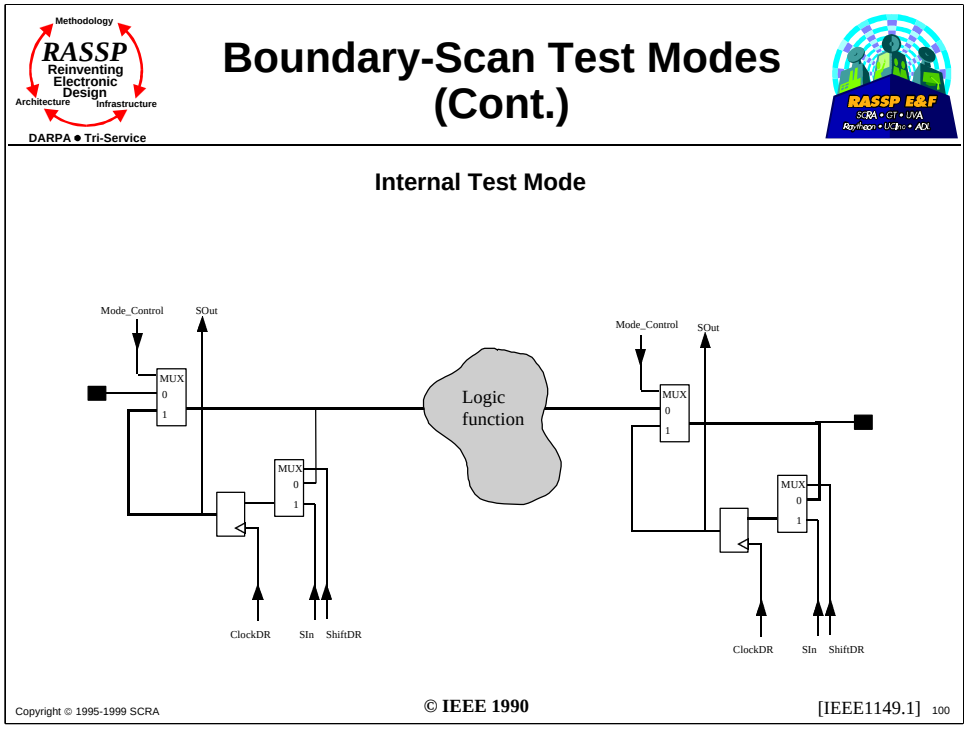
In a PCB with boundary scan, if each chip has scan latches they can be hooked together into one long scan chain. If a board is going to contain some non-scan logic, using scan where possible is still effective because a non-scan chip can be tested via boundary scan if it is surrounded by scan chips.

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.

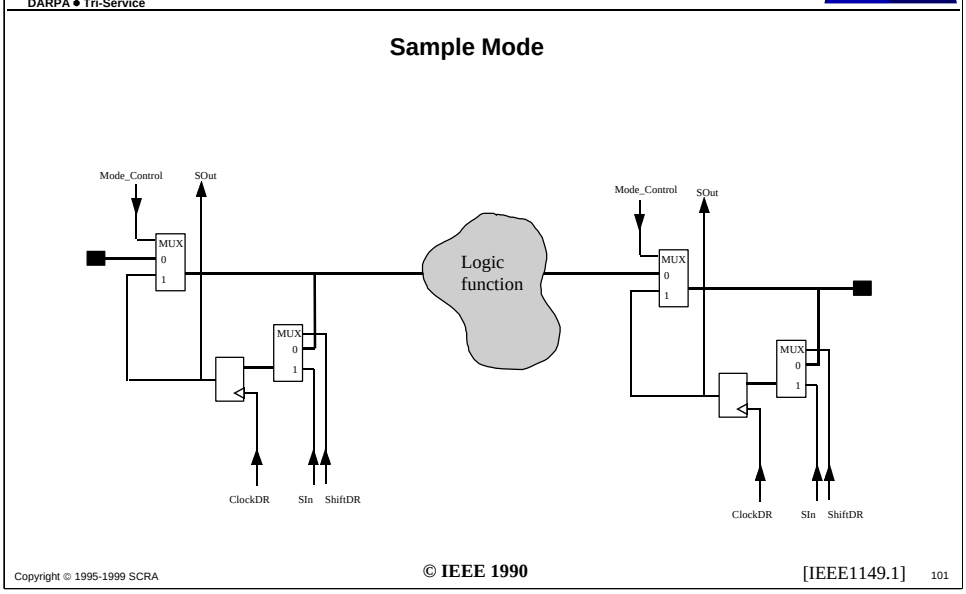
Ref: [IEEE1149.1]



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.

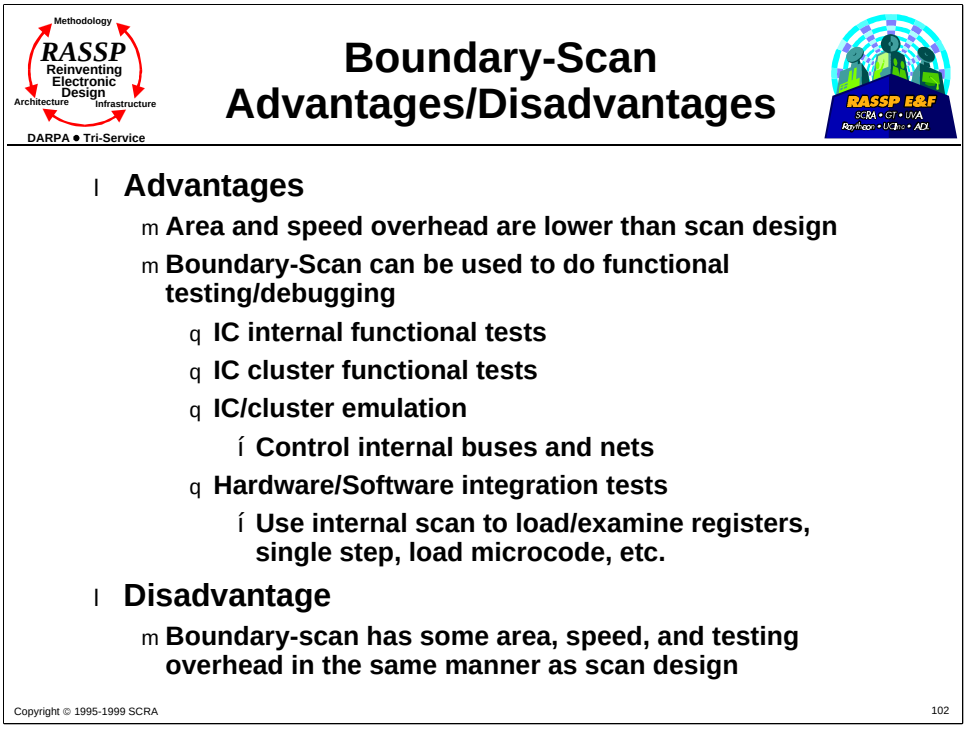
Ref: [IEEE1149.1]

Boundary-Scan Test Modes (Cont.)



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 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.

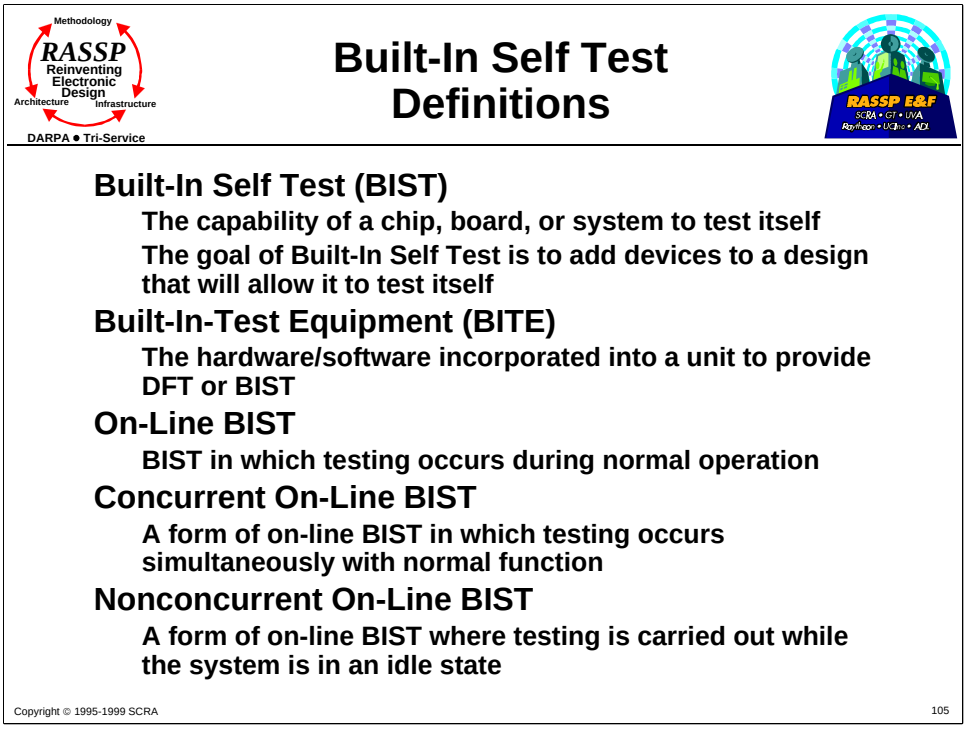
Boundary-scan does have non-zero area, speed, and testing overheads, and that needs to be considered when adding it.

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

Section Outline

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



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

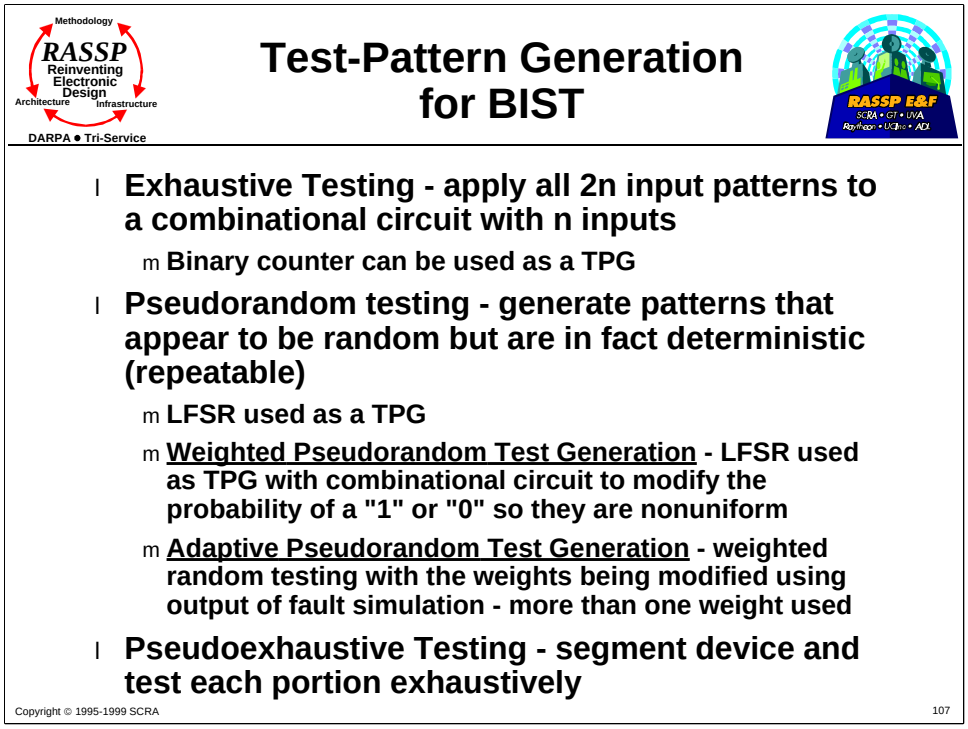
Pseudo Random Pattern Generator (PRPG)

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

Multiple-Input Signature Register (MISR)

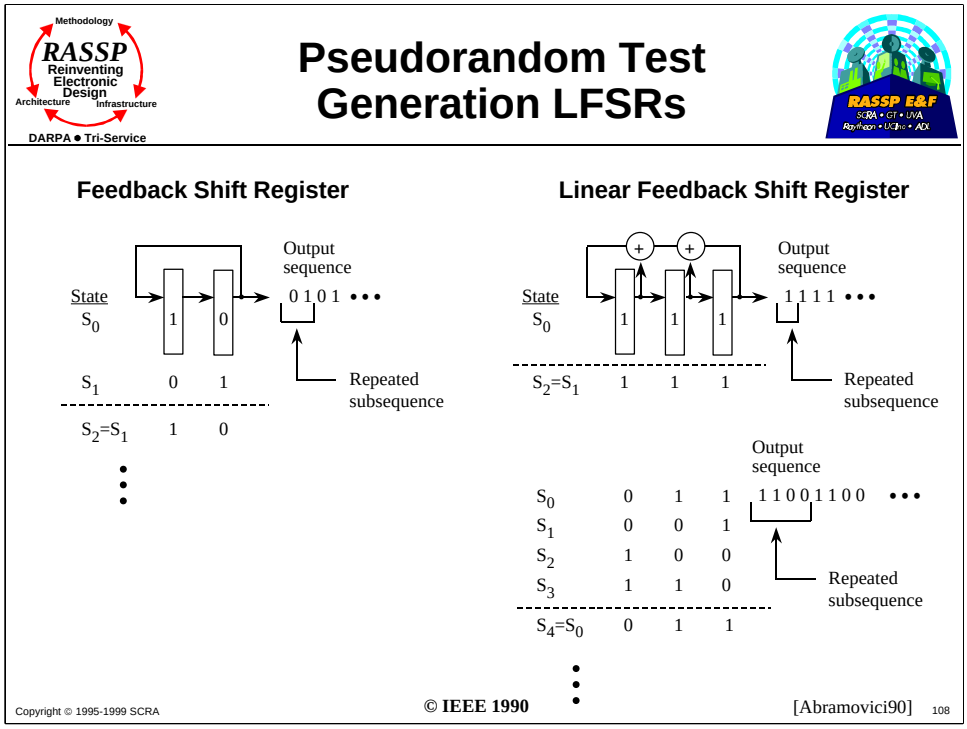
a multi-input device that compresses a series of input patterns into a (pseudo) unique signature

No additional notes are required for the definitions.



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

LFSR will be explained further.



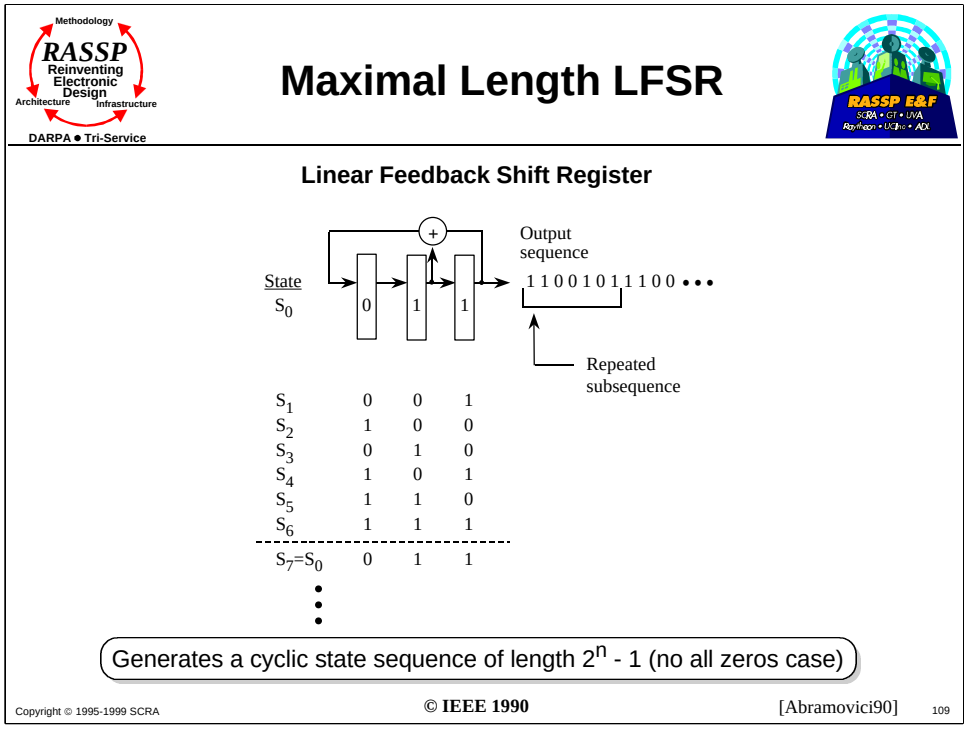
Pseudorandom (likelihood of "1" or "0" is 50%, but patterns are deterministic/repeatable) patterns may be generated by a linear feedback shift register (LFSR).

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

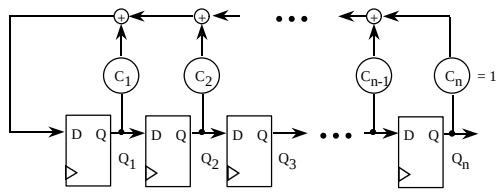


The all zeros case is not possible in this type of LFSR, but notice that the probability of any bit being "1" or "0" is 50% except for that. Therefore, the sequence is pseudorandom in the sense that the probability of a "1" or "0" is approx. 50%, but the sequence is repeatable.

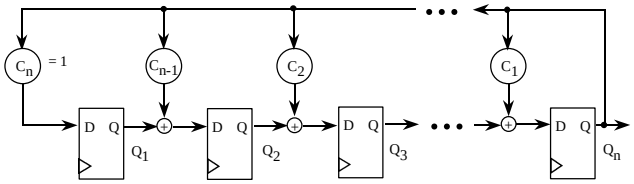
Like a binary counter, all 2^n - 1 states are generated, but in a “random” order that is repeatable.

Rev: [Abramovici90]

LFSR Canonical Forms and Characteristic Polynomials



Type 1 LFSR



Type 2 LFSR

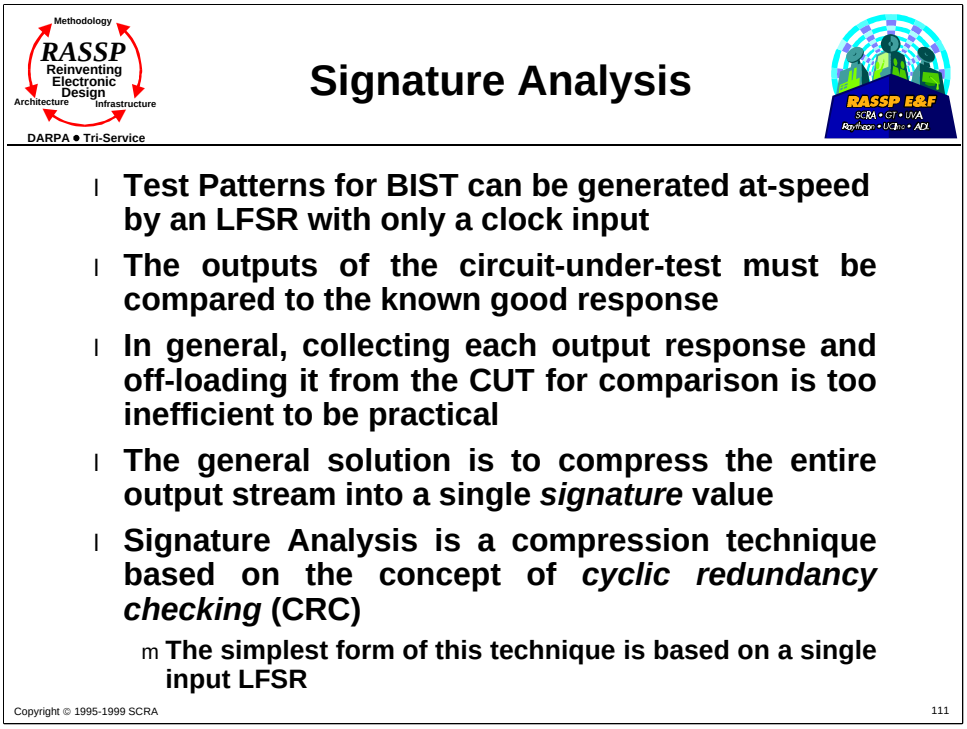
Characteristic Polynomial

$$P(x) = 1 + c_1x + c_2x^2 + ... + c_nx^n$$

[Abramovici90] 110

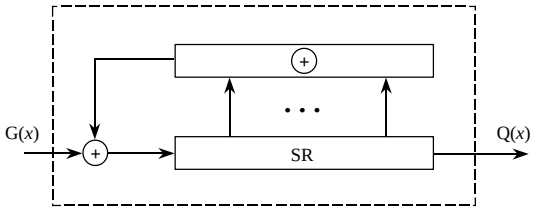
In terms of the mathematical theory, LFSRs have a characteristic polynomial associated with them that can be expressed in terms of the feedback connections. This will be used further in the section on signature analysis.

Ref: [Abramovici90]



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.

Signature Analysis (Cont.)



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

or

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

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

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

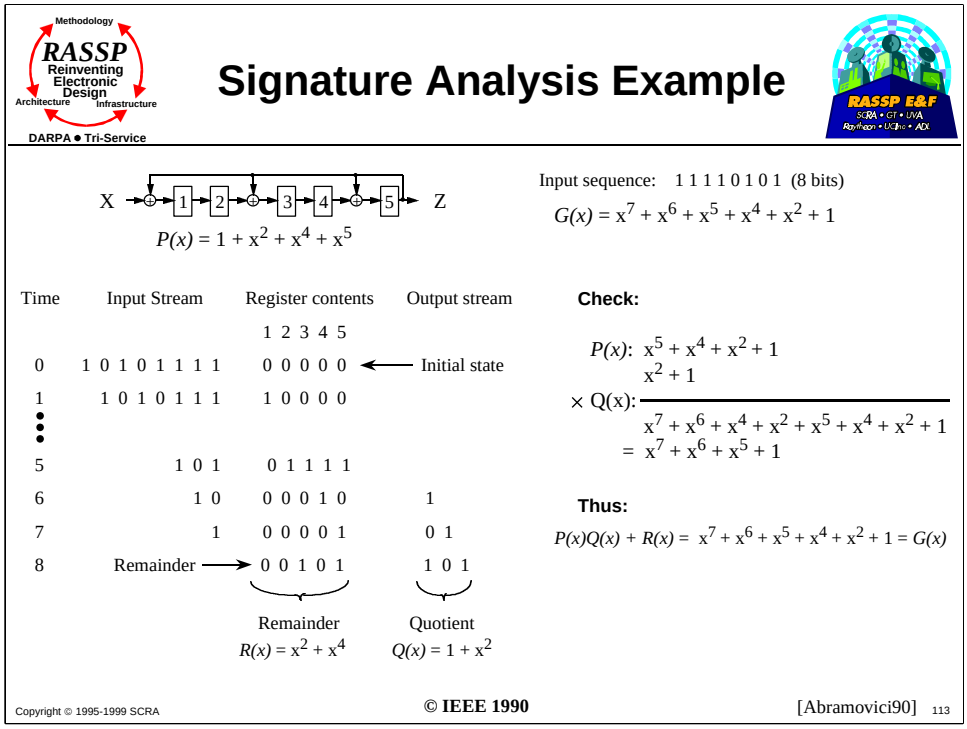
number of erroneous bits streams that produce the same signature as a particular fault-free response: $2^{m-n} - 1$

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

[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.

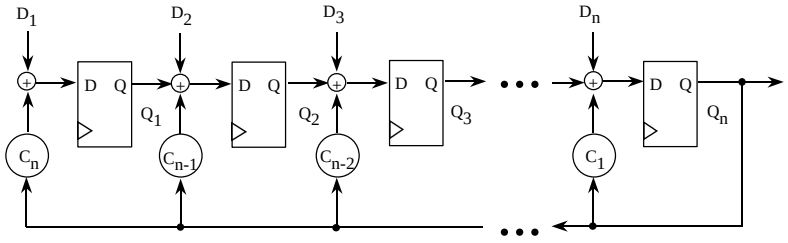
Ref: [Abramovici90]



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.

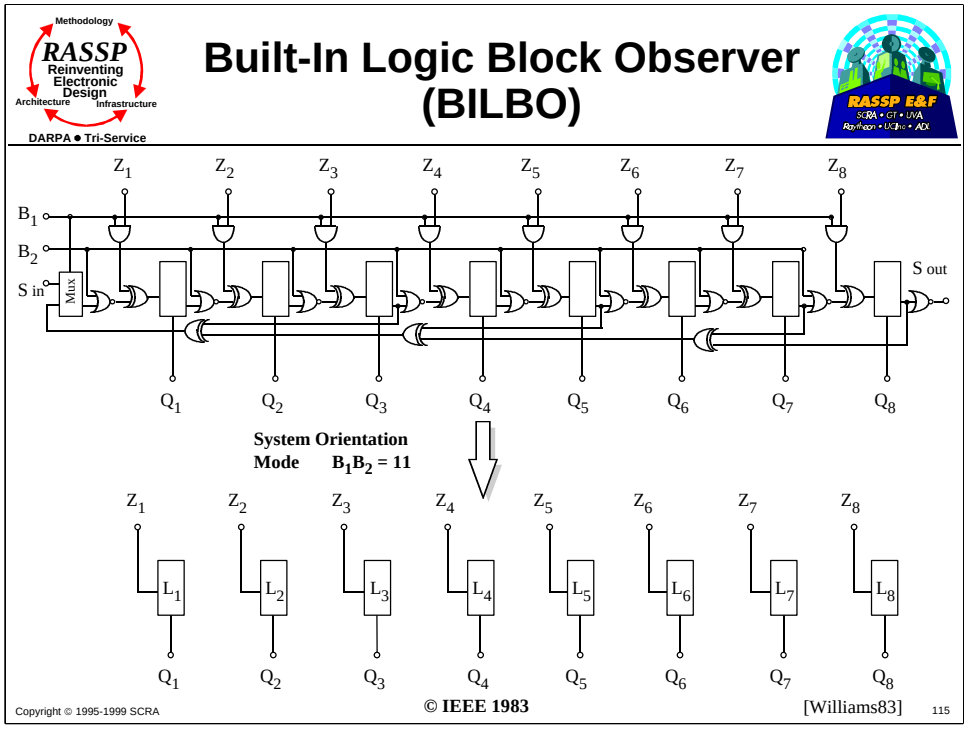
Ref: [Abramovici90]

Multiple Input Signature Register



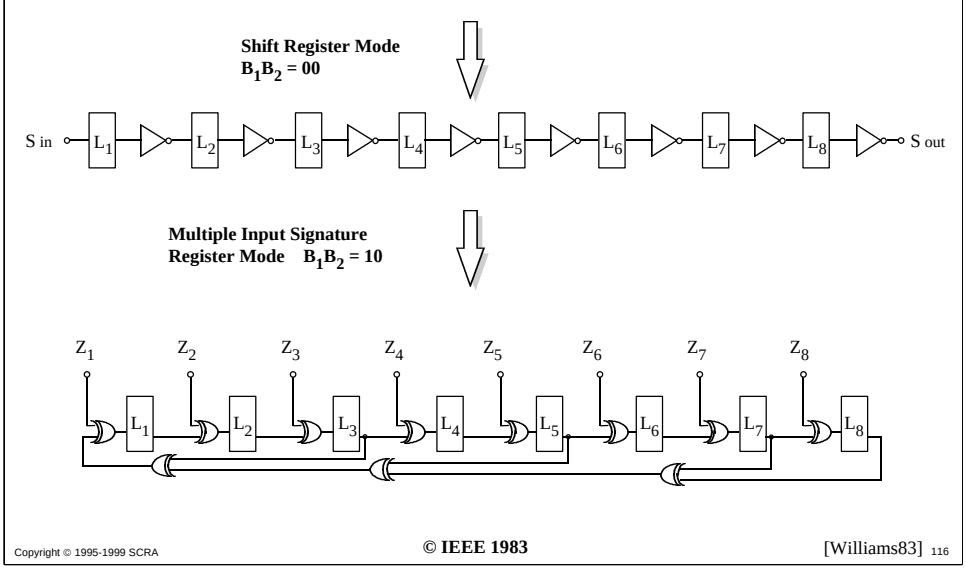
[Abramovici90]

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.

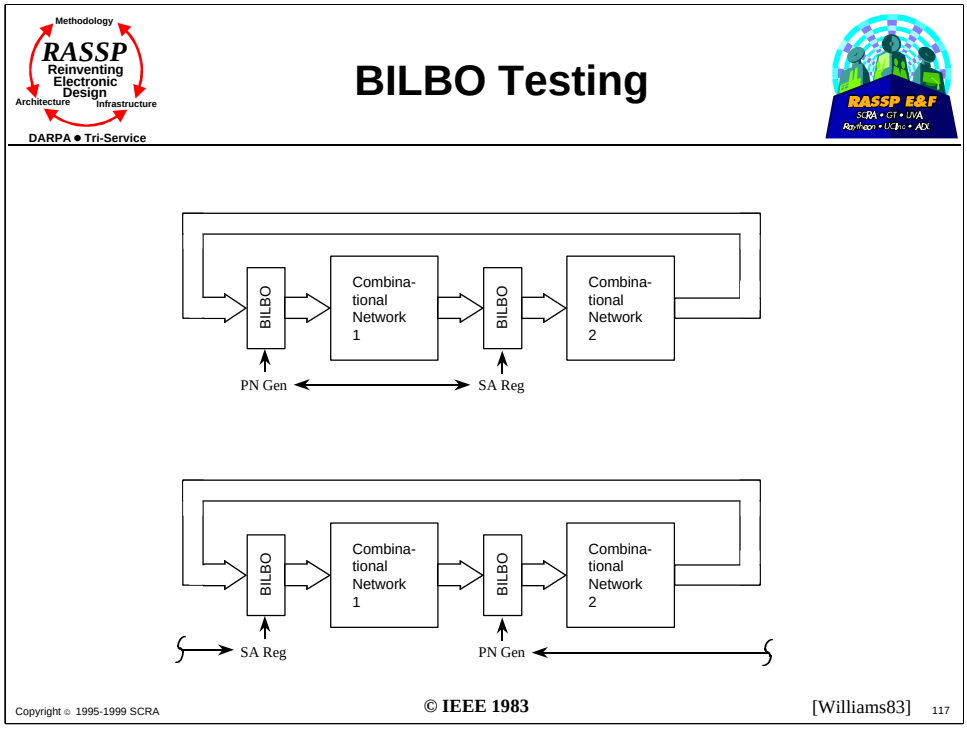
BILBO (Cont.)



© IEEE 1983

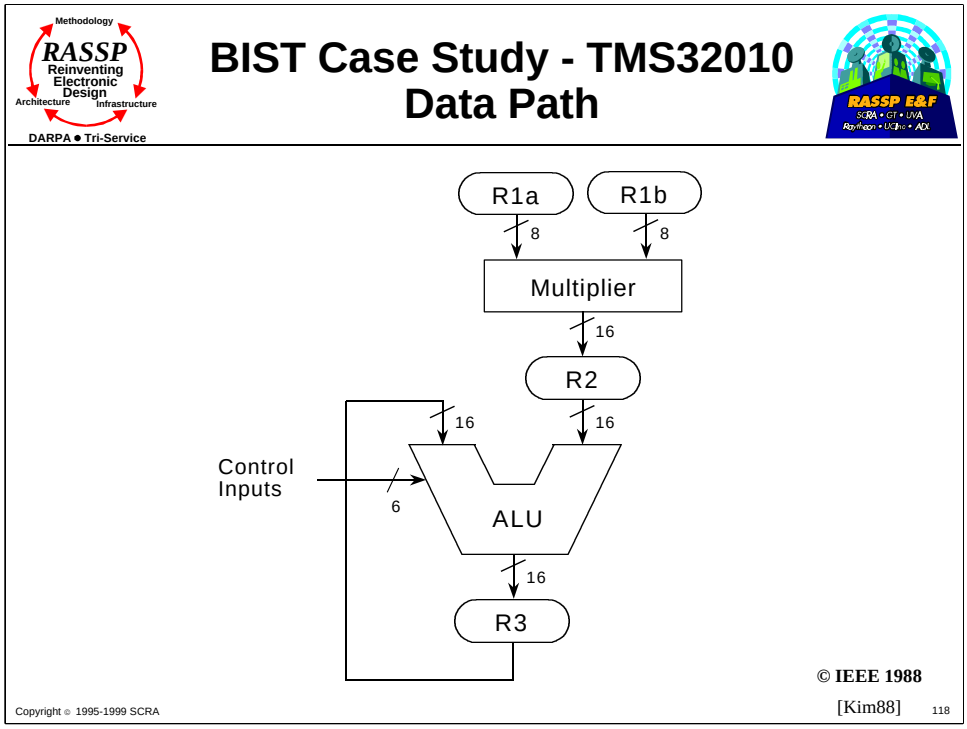
[Williams83]

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.



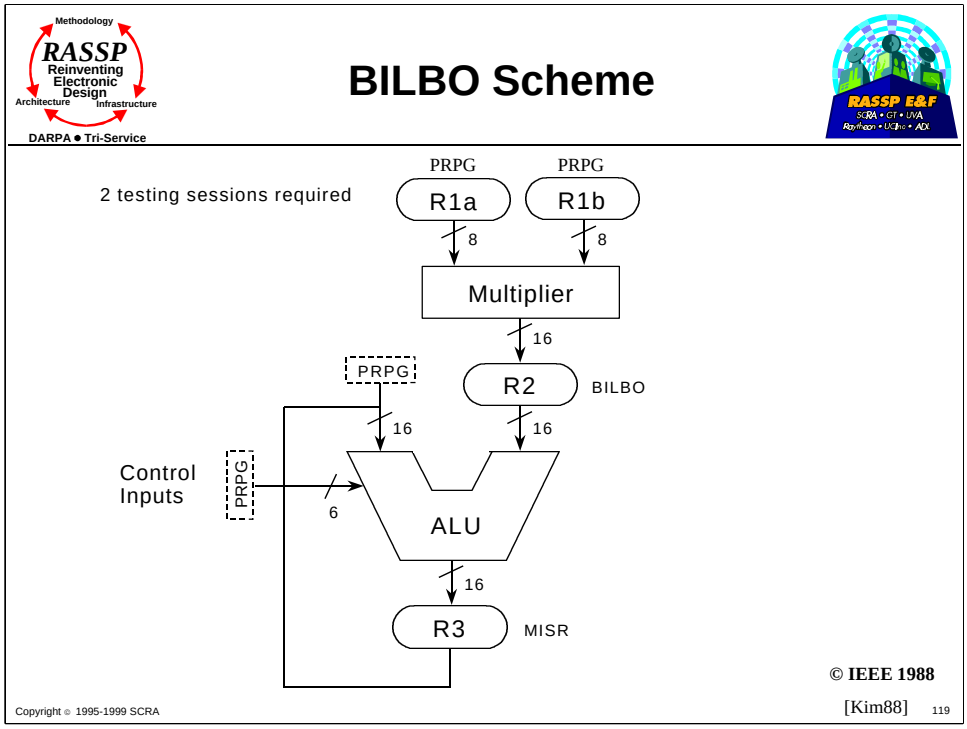
BILBO Testing

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.

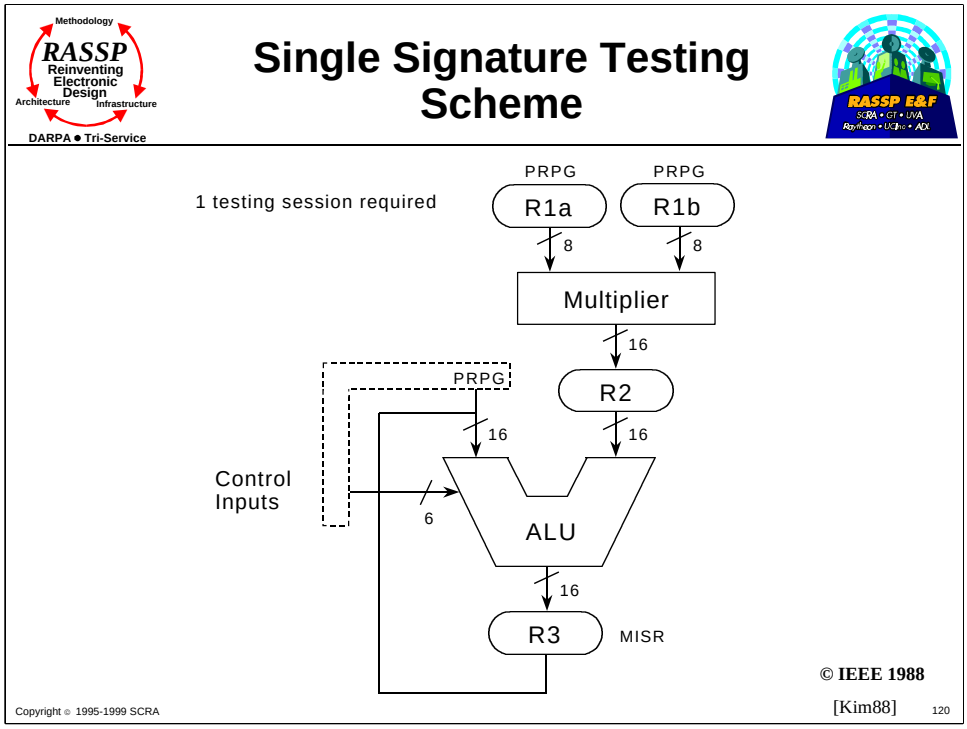


This is a case study in the literature which describes various configuration of BIST for a section of the TMS32010 data path shown here.

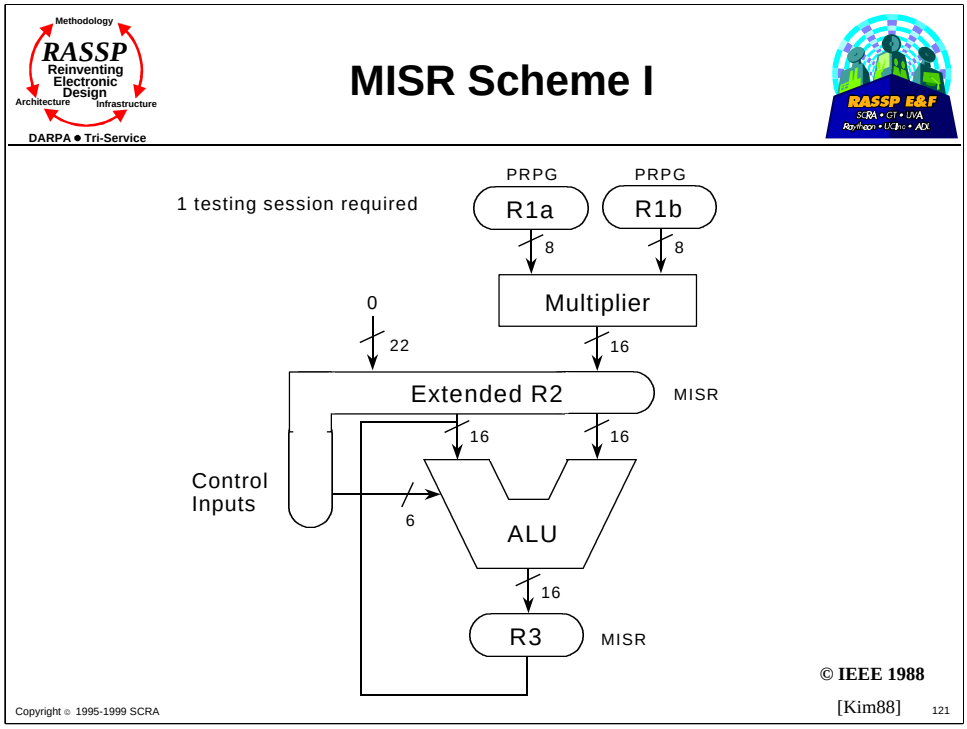
Ref: [Kim88]



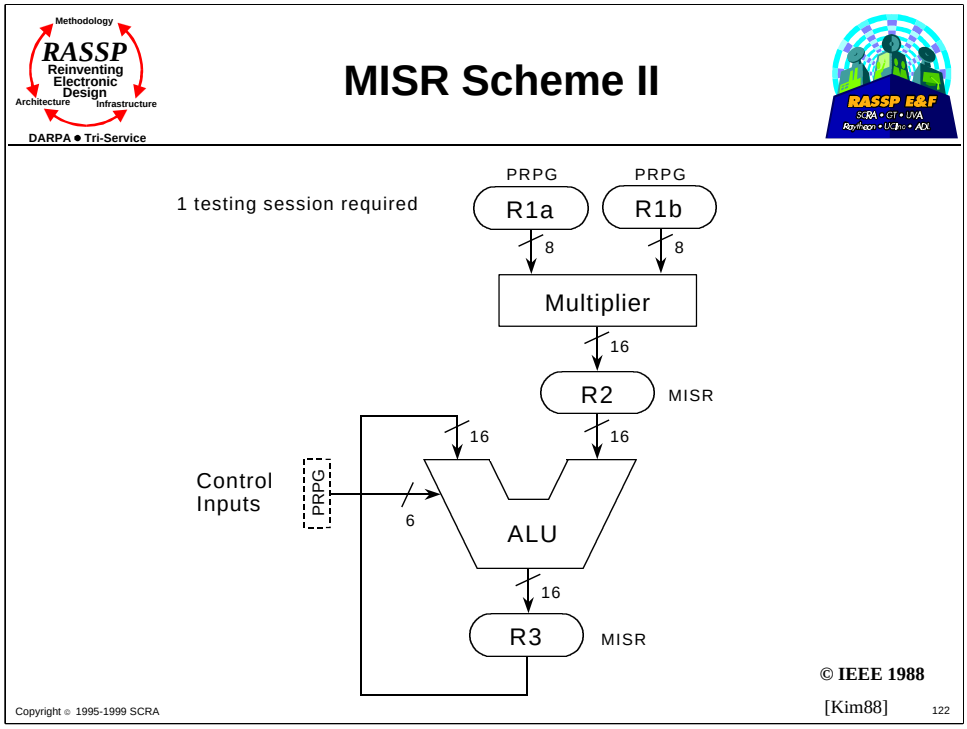
This figure shows the normal BILBO scheme where R1a, R1b, R2, and R3 are replaced with BILBO registers. Also, a 6 bit PRPG is required for the ALU mode bits, and a 16 bit PRPG is required for the leftmost ALU bits. Two testing sessions are required, one for the multiplier and one for the ALU.



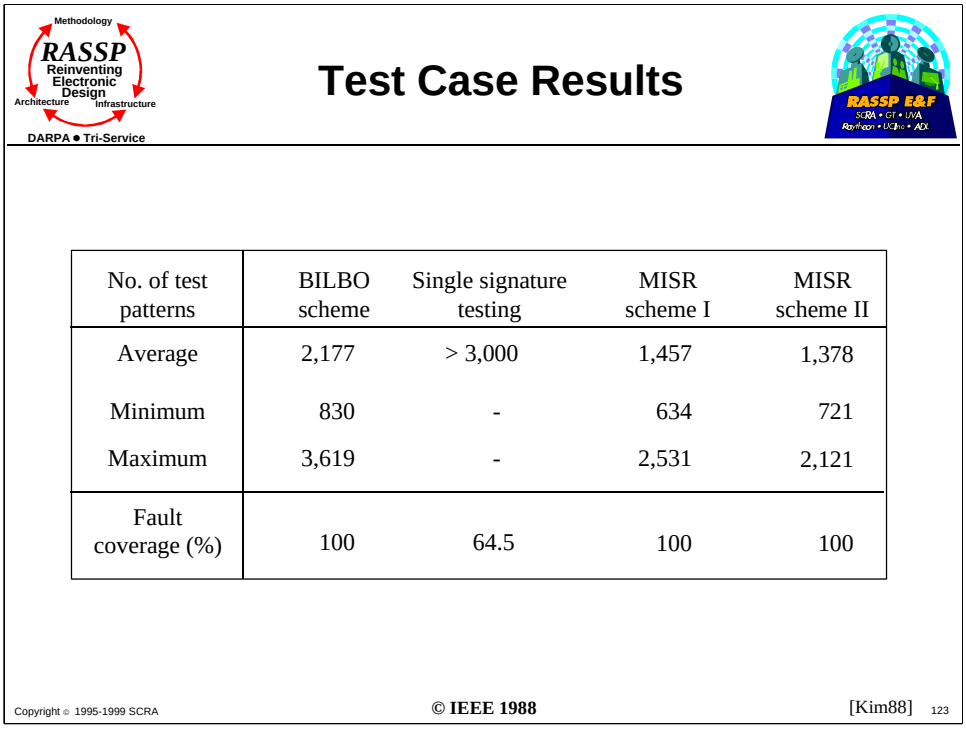
In this scheme, R2 is unmodified and the outputs of the Multiplier during testing are used as partial test inputs to the ALU for testing. Again, a 6 and 16 bit PRPG is required, but R2 is not a BILBO. Only one testing session is required.



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.

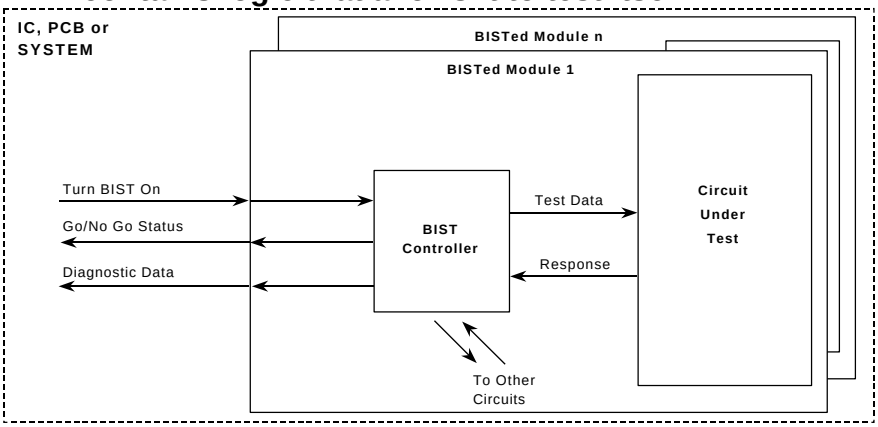


This table show the results of the various schemes. Twenty different runs with different seeds for the PSRGs were done. For designs 1, 3, and 4, the simulation was stopped when fault coverage reached 100%. For design 2, fault coverage saturated at 64.5%.

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

Autonomous Built-In Self-Test (ABIST)

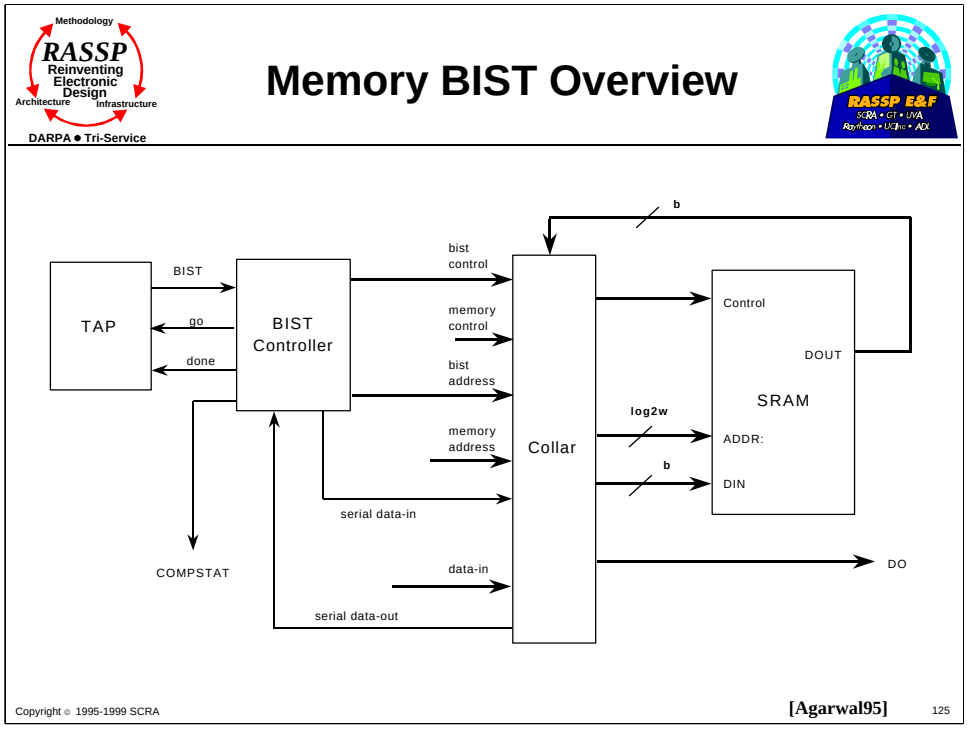
- **An approach to testing in which a module contains logic that allows it to test itself**



[Agarwal95]
124

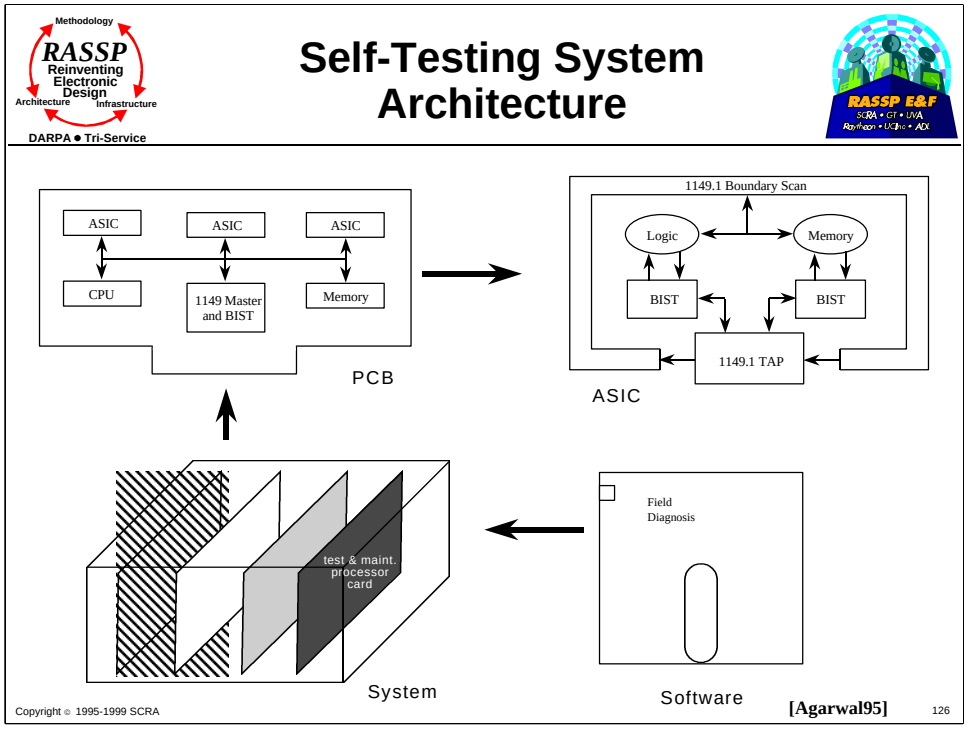
A novel concept being developed for RASSP by LogicVision is Autonomous Built-In Self-Test. In ABIST, normal BIST techniques are built into the circuit-under-test, but a controller is added that will completely control that BIST hardware to configure it as required and run the required test sessions. 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]

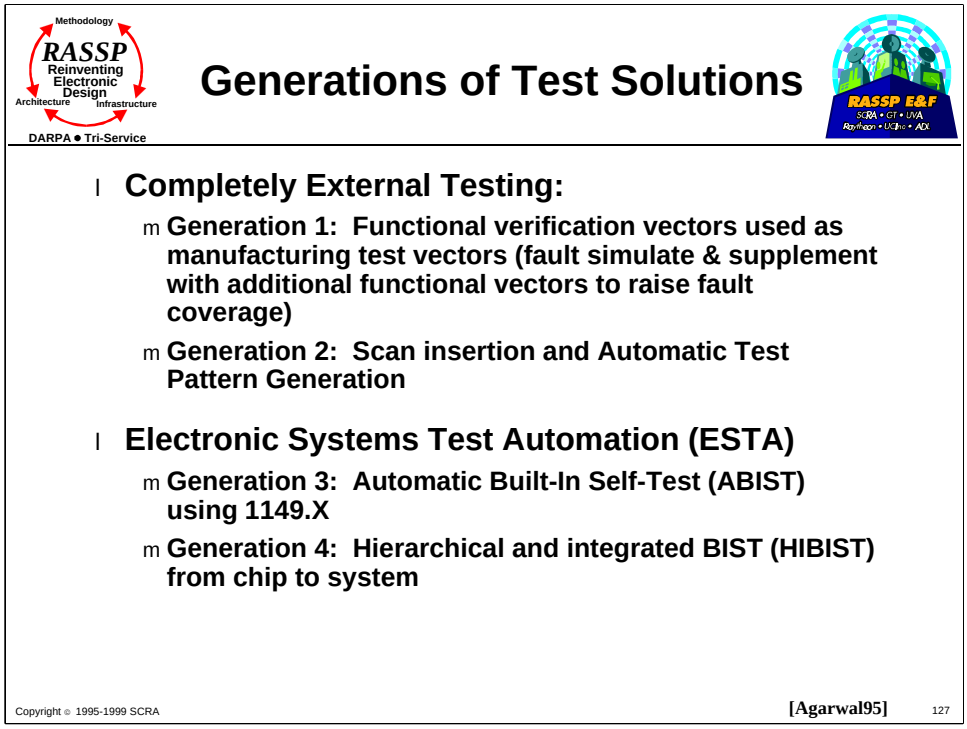


This figure shows the configuration of an ABIST SRAM developed by LogicVision. The collar is a parametarizable part that can be configured for various sizes of SRAMs. It contains the shift registers necessary to apply the test data to the SRAM core.

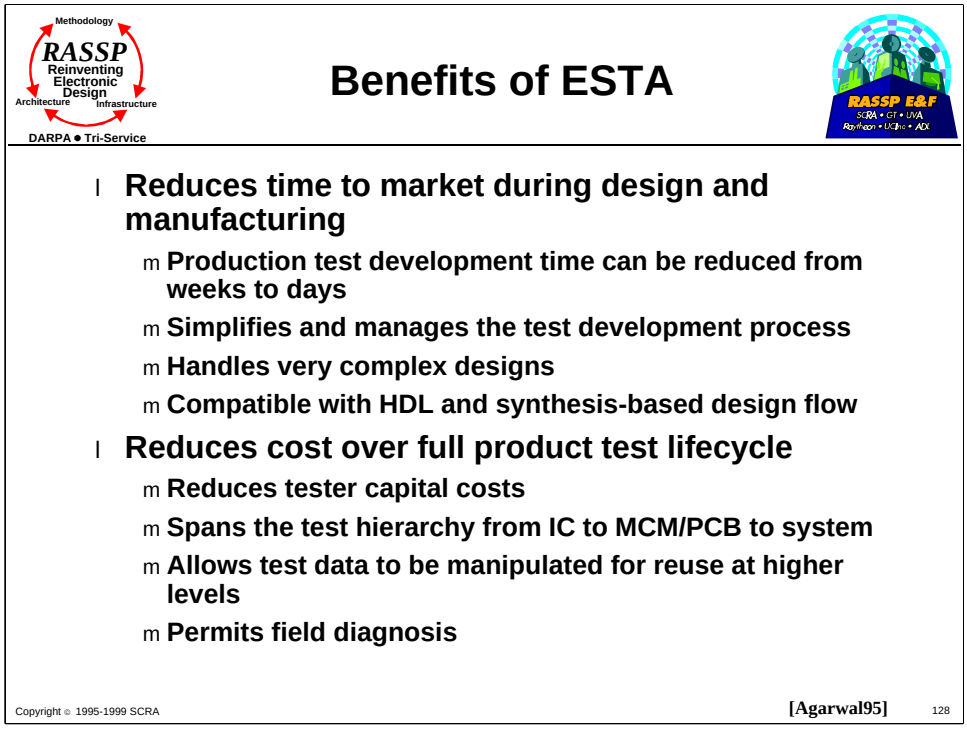
LogicVision has a tool, ICRAMBIST, that automatically generates VHDL or Verilog descriptions of these parts.



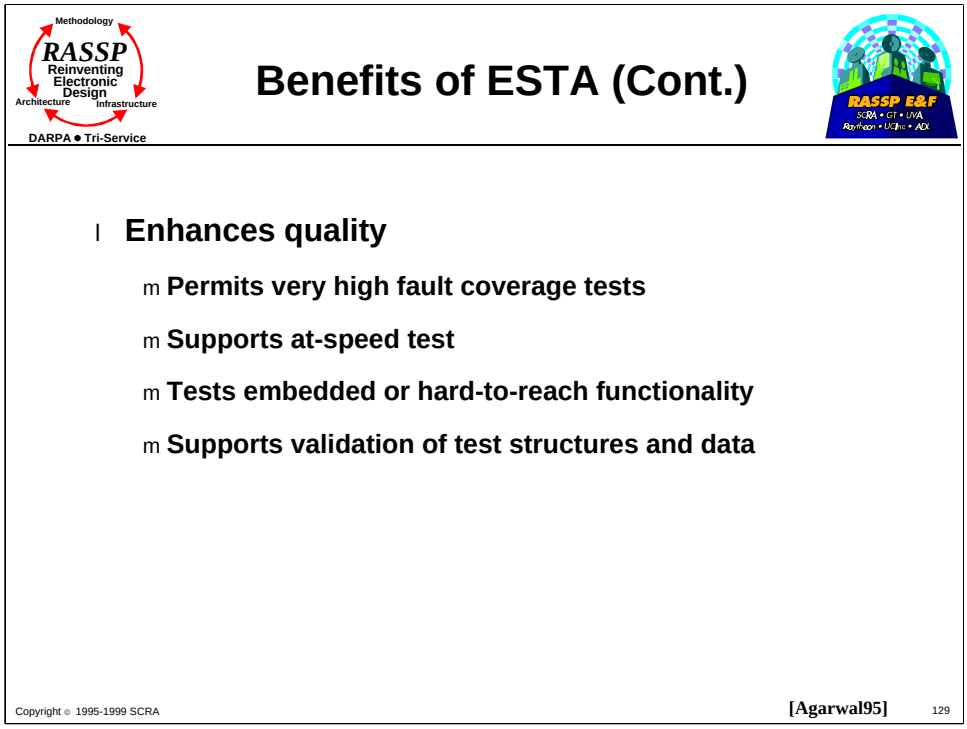
This figure illustrates the necessity of building in testability at all levels, from chip, to board, to system, to diagnostic software.



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.



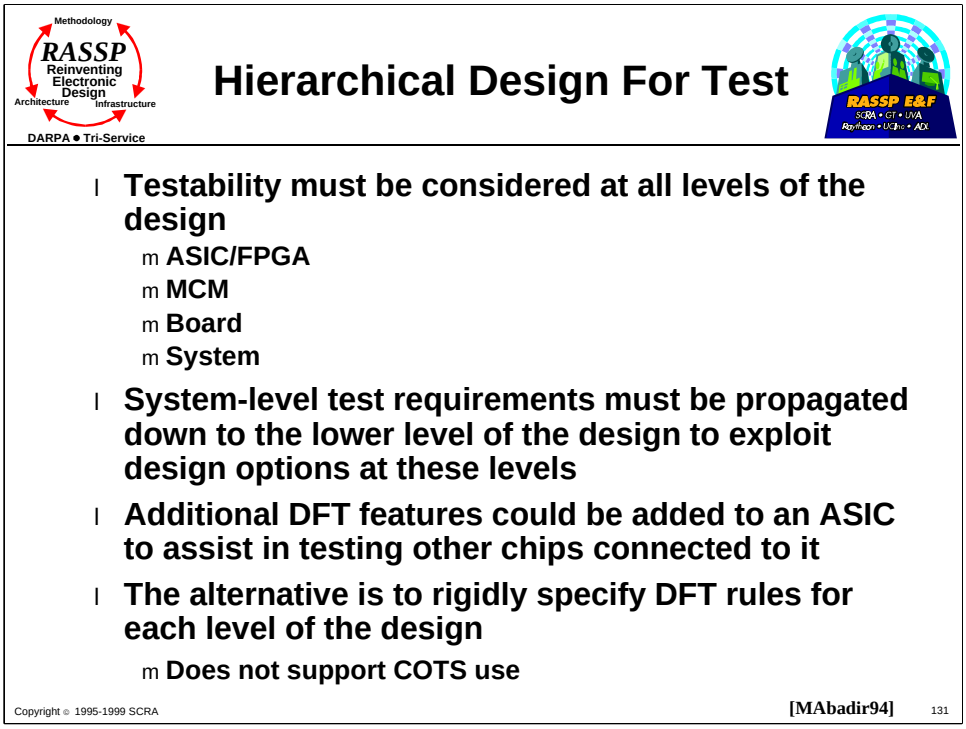
There are major benefits to the incorporation of ESTA, some of which are stated here...



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

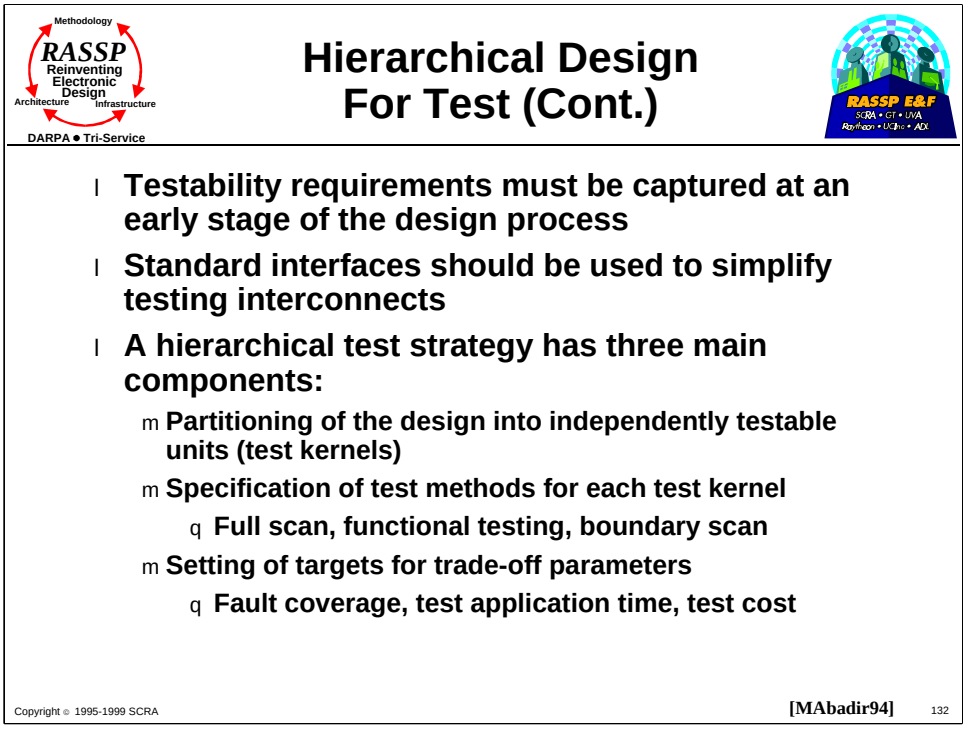


Hierarchical Design for Test requires that testing be considered at all levels of the design process.

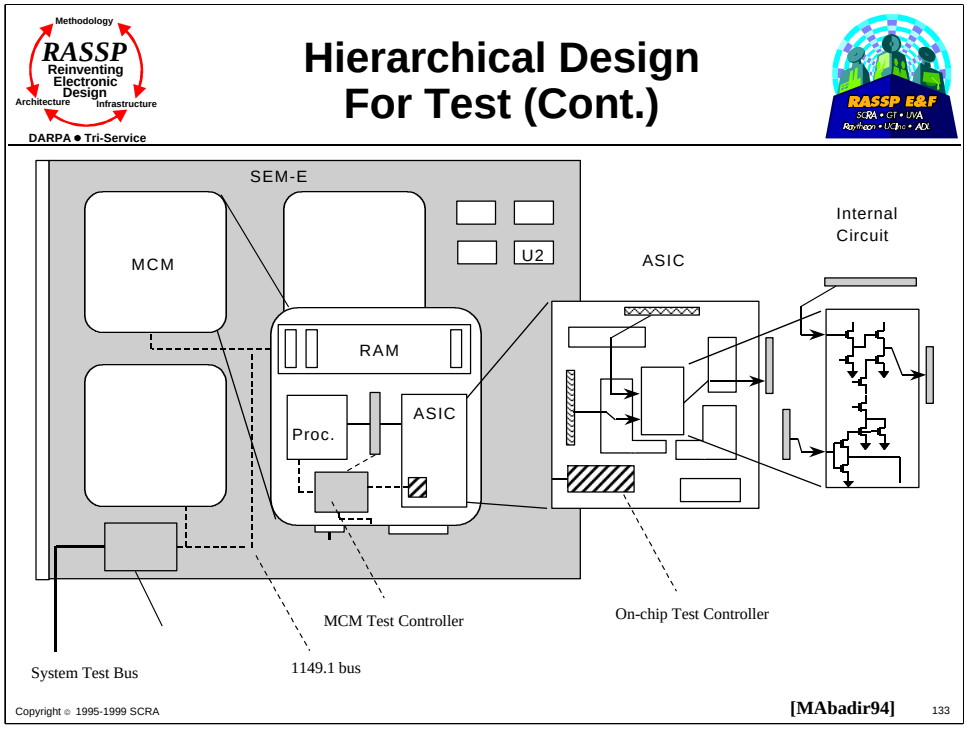
For example, if during system design a COTS processor that doesn't include boundary scan is specified, boundary scan can be added/required on all interface chips around it to effectively make the processor boundary scan testable.

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

Ref: [Abadir94]



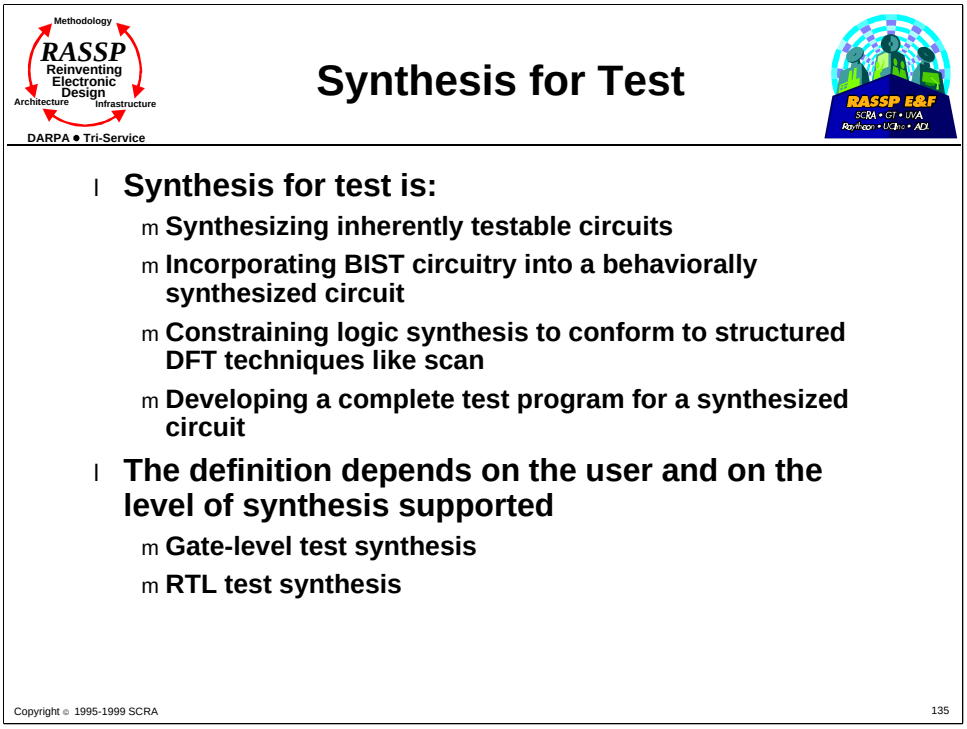
Some further points about HDFT are stated here.



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.

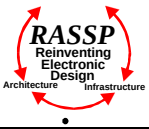
Module Outline

- 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



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.

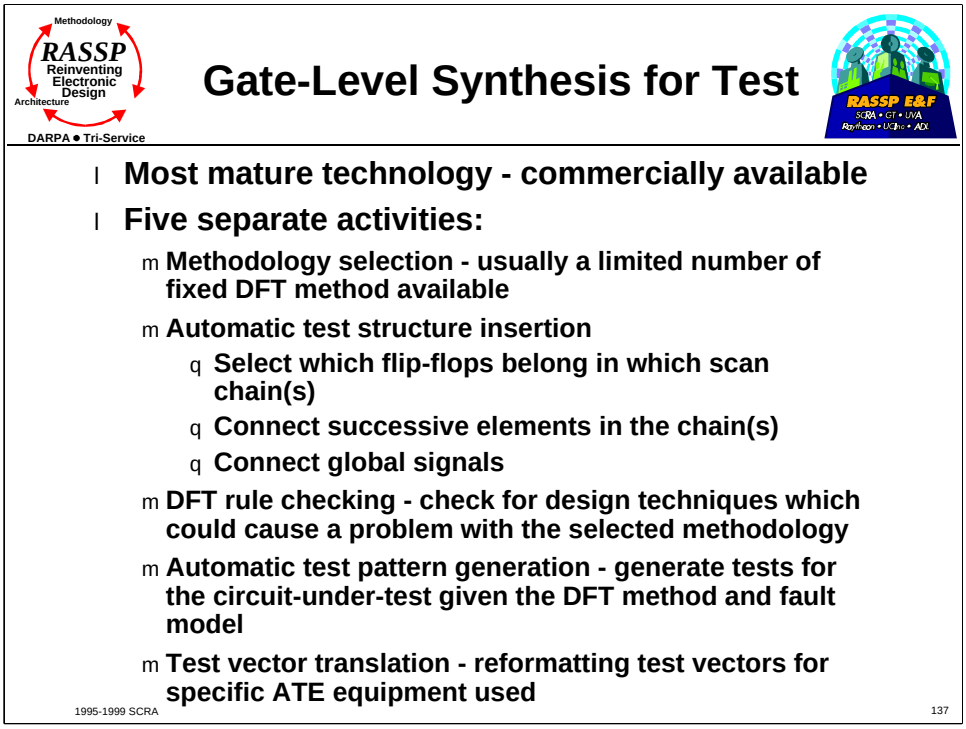
Ref [Aitken95]



Elements of Synthesis for Test

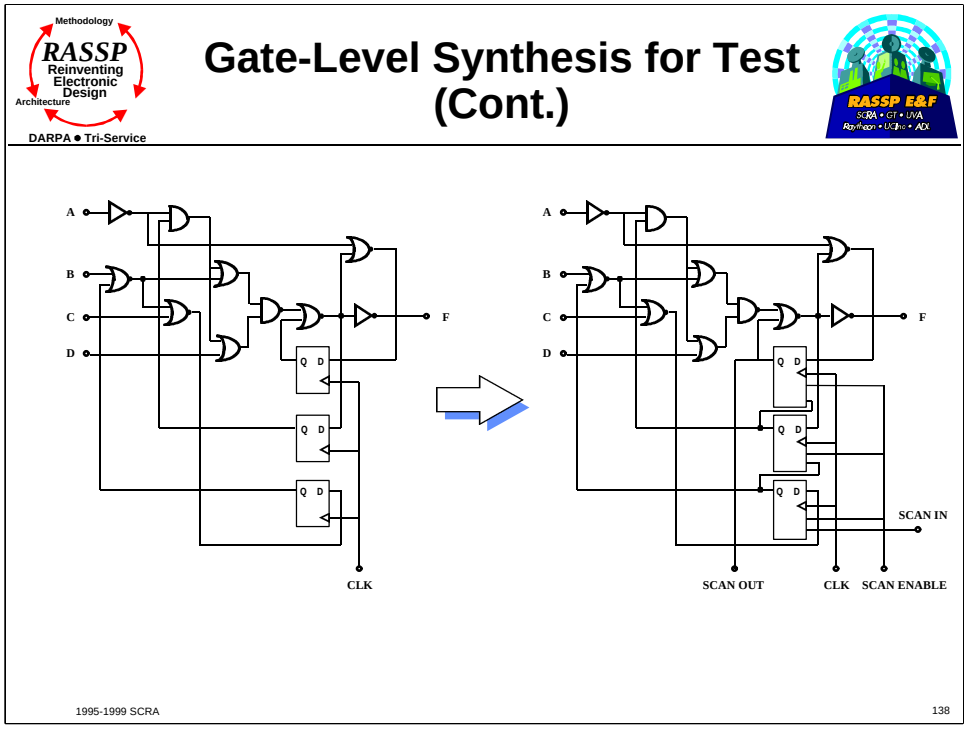
- **Methodology selection - selection of the overall set of rules, structures, and design constraints to be used in including testability in the design process**
- **Test structure insertion - the actual insertion of the selected test structures into the description of the circuit-under-test**
- **Circuit 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.**

Ref [Aitken95]



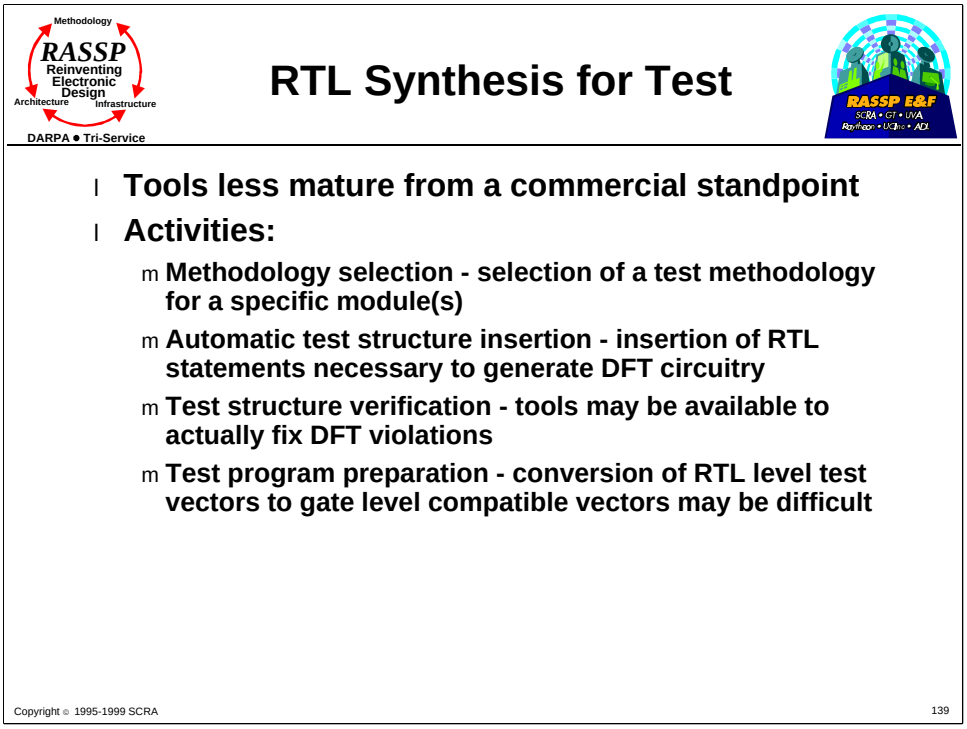
Gate level synthesis for test is fairly straight forward in that it consists of selection of a testability measure (usually some form of scan full or partial), and then automatic insertion and test pattern generation and test program construction (building the scan chains).

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.



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.

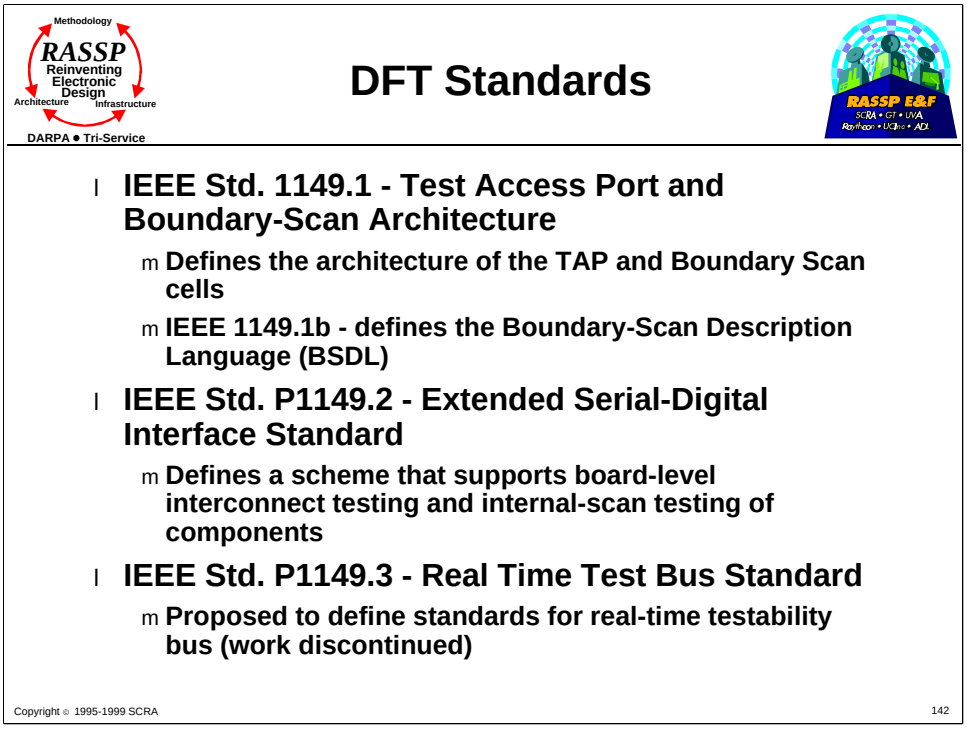
University tools for RTL synthesis for test concentrate on generating testable structures. This involves construction of a register adjacency graph and then changing it to make the structure more testable. This is done by minimizing the sequential depth between input and output registers, maximizing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

Module Outline

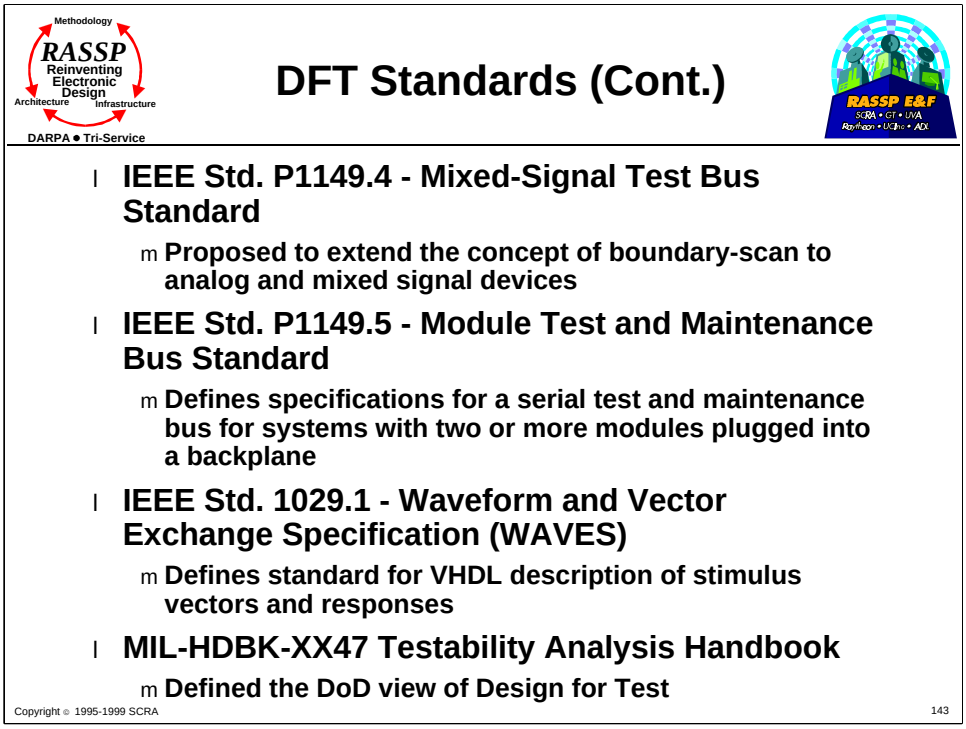
- Introduction (Part 1)
- Fault Modeling
- Test Generation
- Automatic Test Pattern Generation Algorithms
- Fault Simulation Algorithms
- Introduction (Part 2)
- I_{DDQ} Testing
- Design for Testability Techniques
- Built-In Self Test
- Hierarchical Design for Test
- Synthesis for Test
- **DFT Standards**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

Section Outline

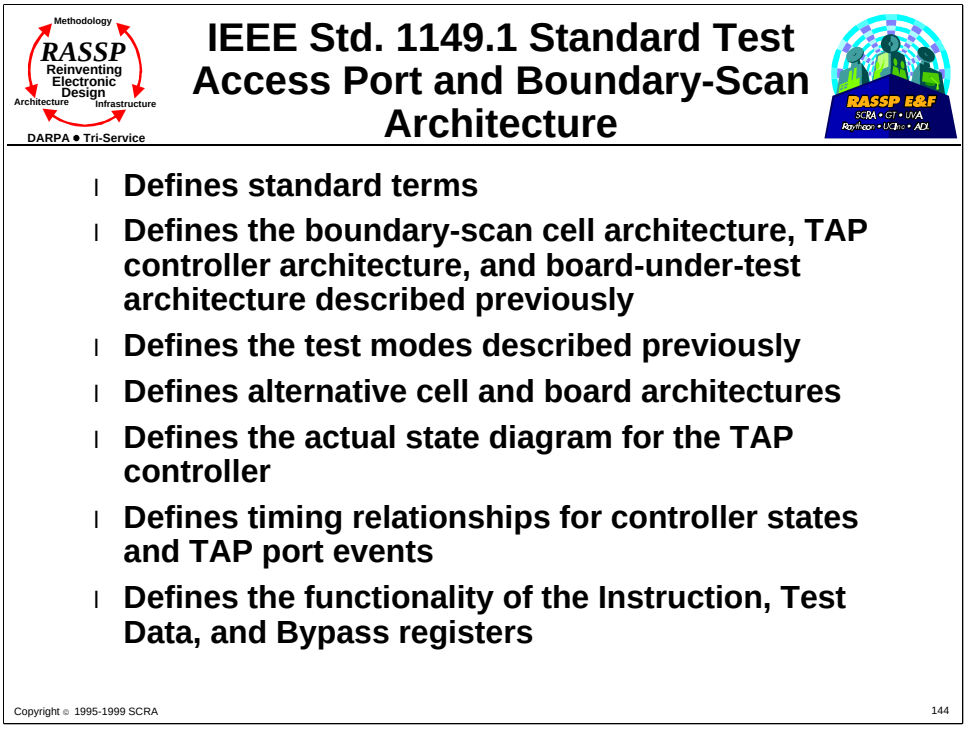
- **DFT Standards**
  - **IEEE Std. 1149.1**
  - **IEEE Std. 1149.1b**
  - **IEEE Std. 1149.5**
  - **IEEE Std. 1029.1**
  - **MIL-HDBK-XX47**



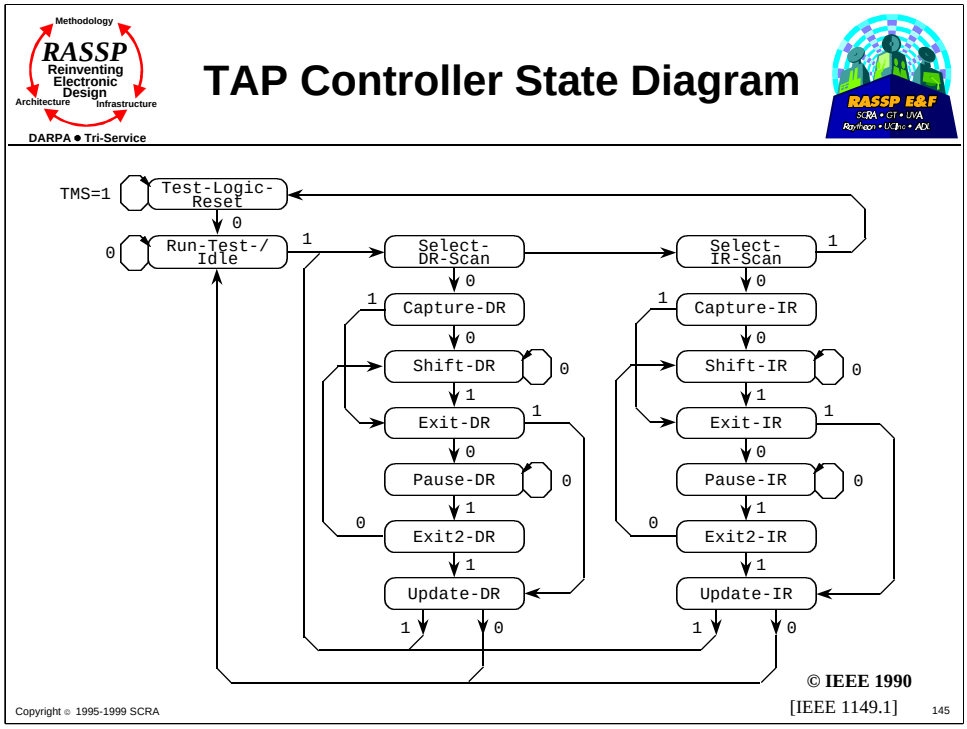
This slide lists some for the approved/proposed IEEE testing standards. Note that P1149.3 is currently inactive.



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.

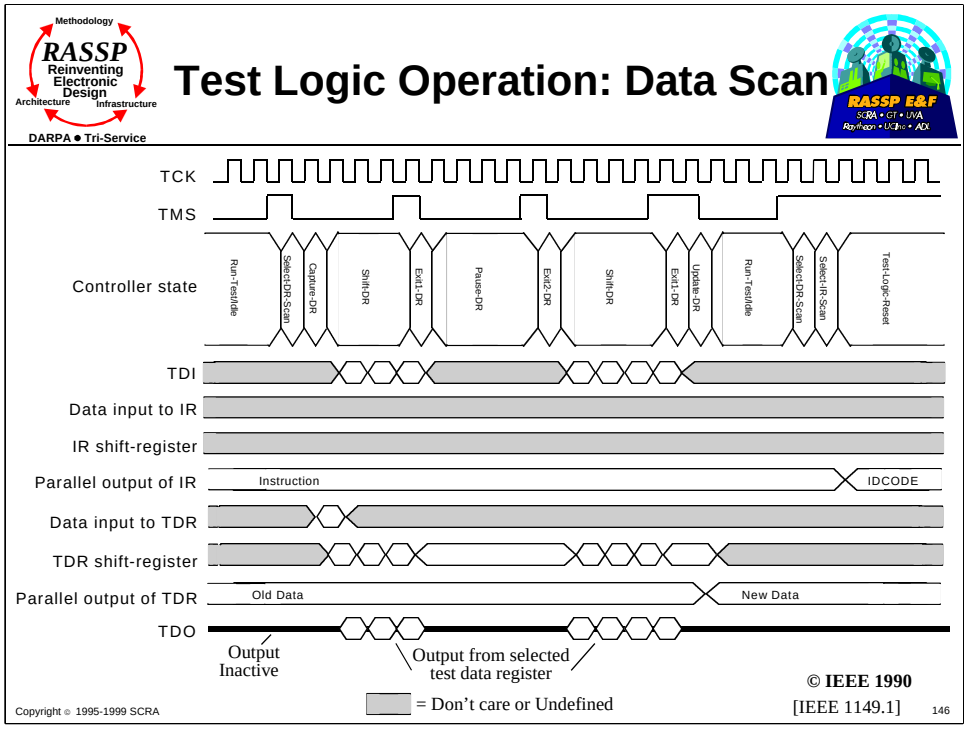


Here are listed some of the major contents of the IEEE 1149.1 standard. The architecture described previously for the boundary-scan cells/architecture are defined in this standard.



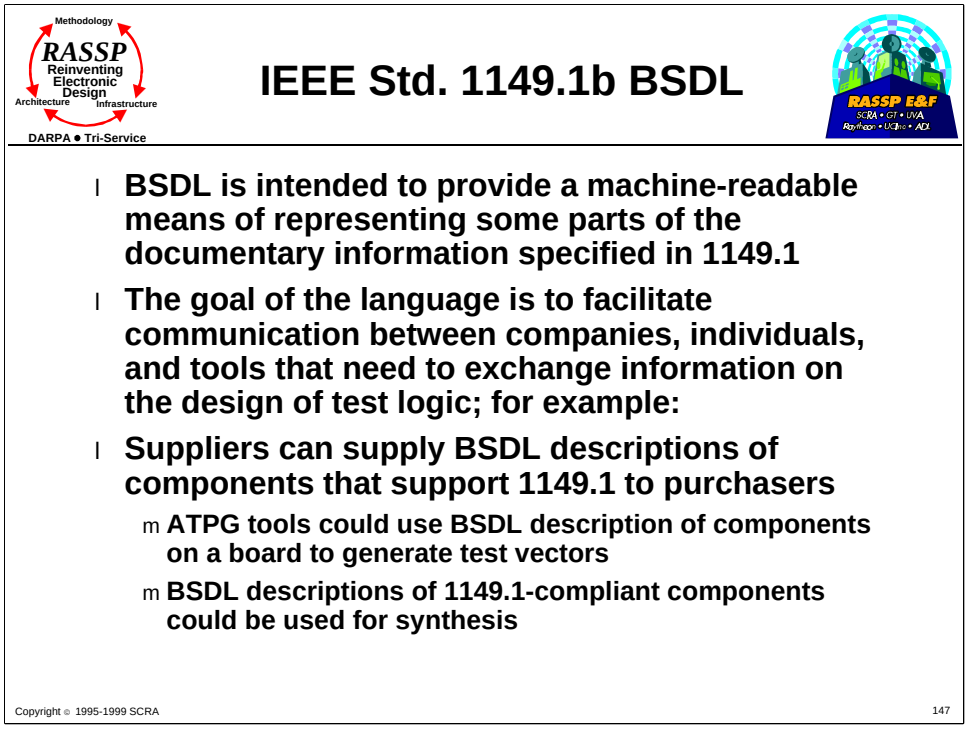
Additional details such as the TAP controller state diagram are defined in the standard to insure interoperability between parts that are designed to the standard.

Ref: [IEEE1149.1]

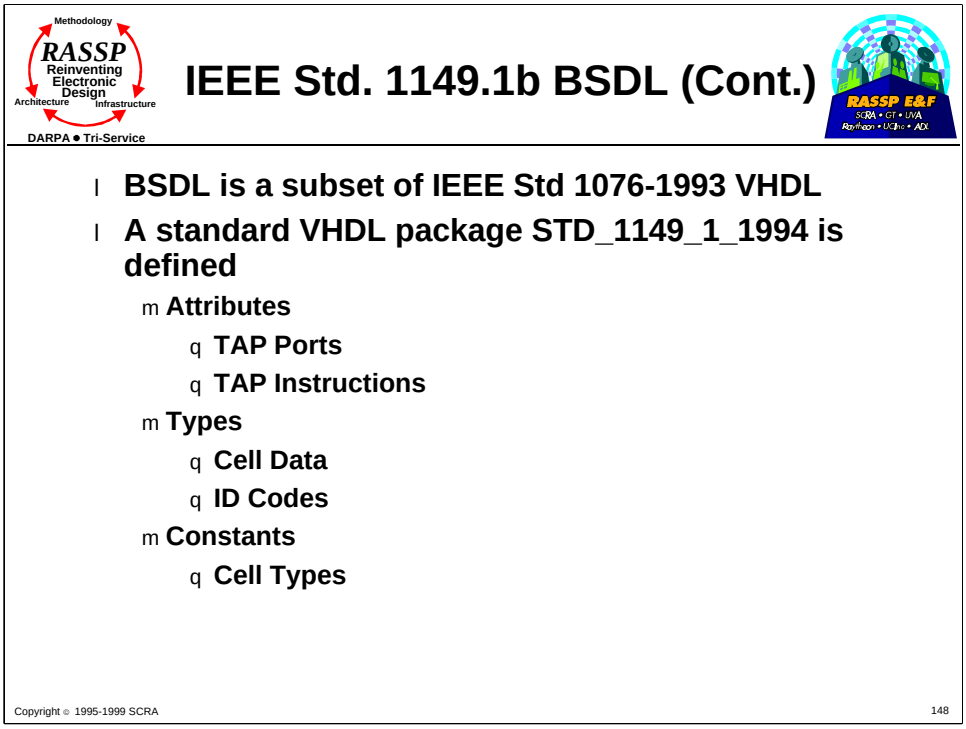


Timing diagrams such as this one are also defined to ensure interoperability.

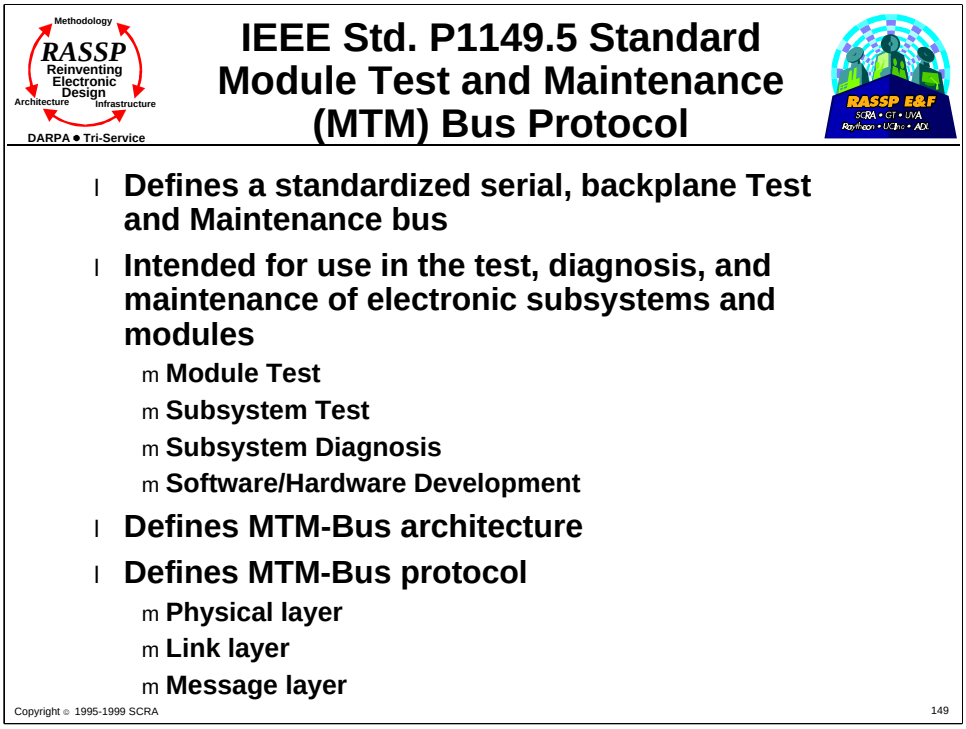
Ref: [IEEE 1149.1]



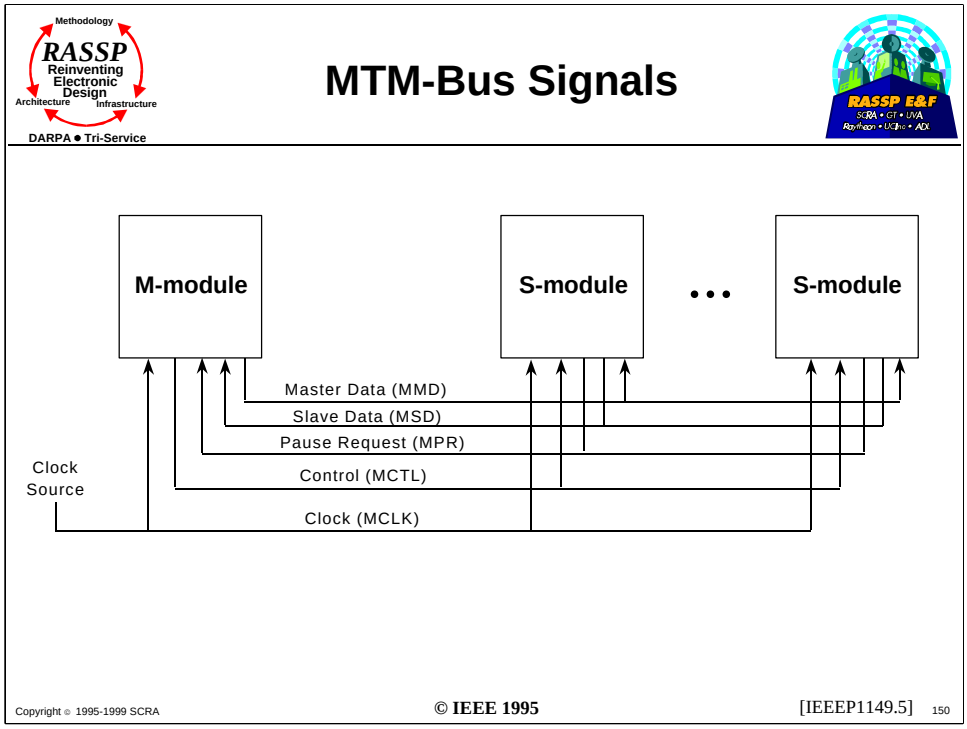
The 1149.1b standard is an appendix to 1149.1 that defines the Boundary-Scan Description Language. BSDL is designed to standardize the way boundary-scan architectures are described.



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.

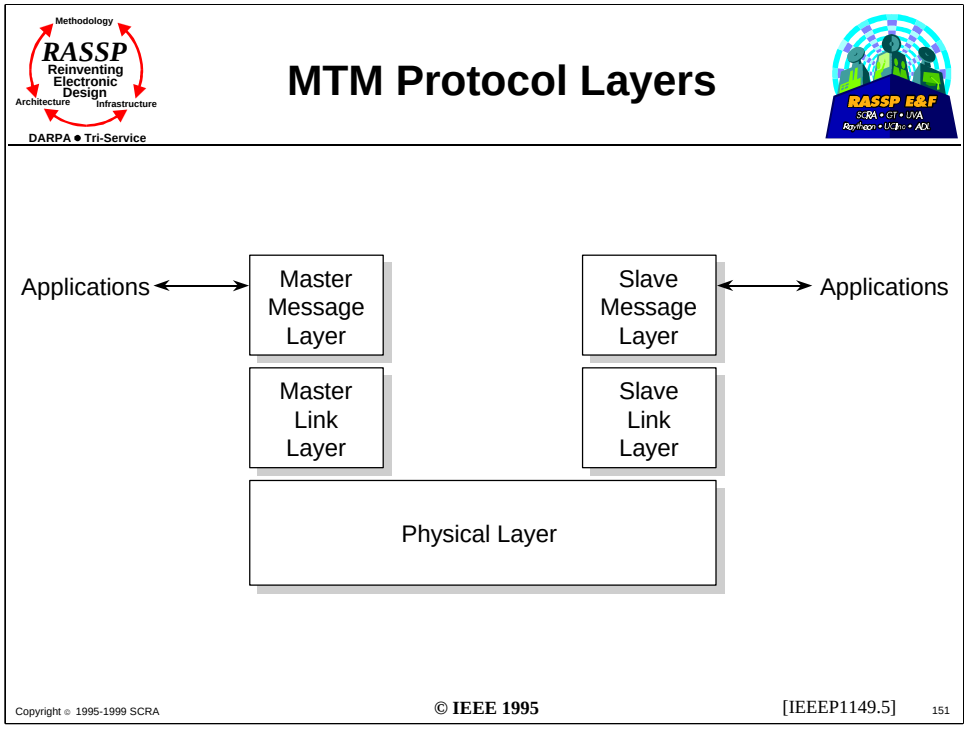


The IEEE P1149.5 MTM standard defines test interoperability at the next higher level from 1149.1.



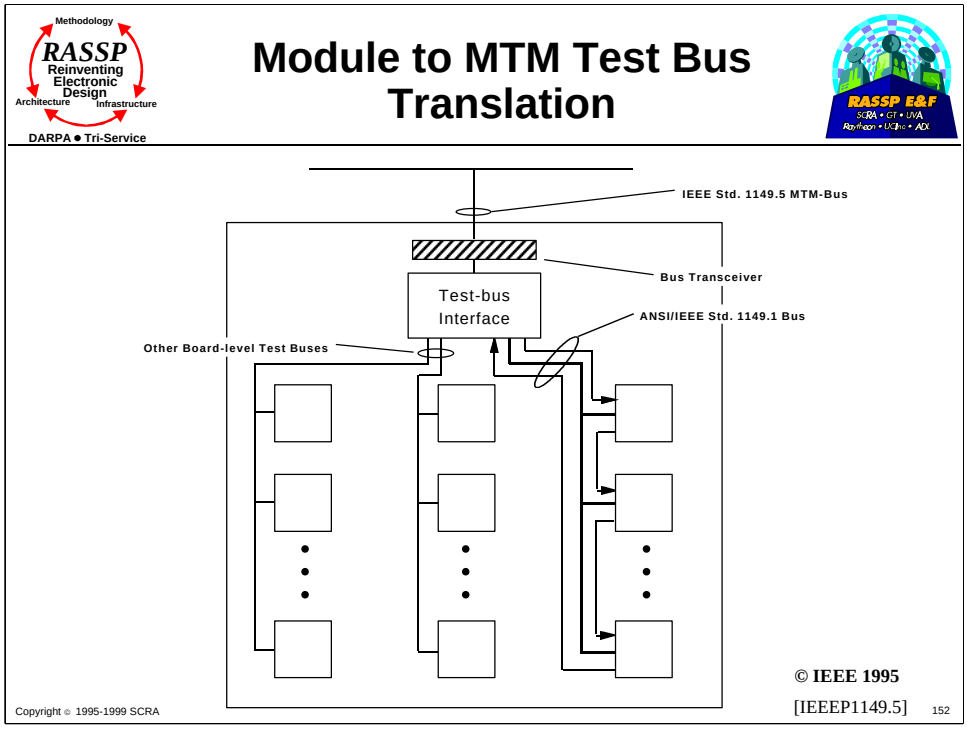
This figure shows the MTM bus and how it is used to connect testable systems together so that they can transmit and receive test data over the same bus.

Ref: [IEEEP1149.5]



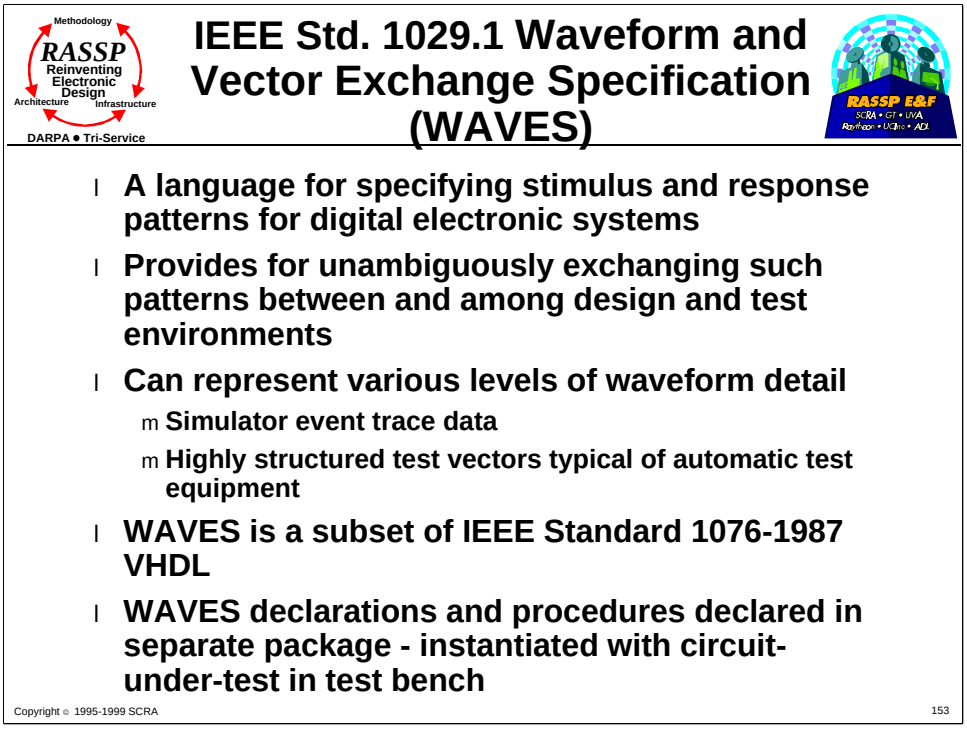
The standard also defines a fairly detailed protocol for transmitting messages over the MTM bus. Different messages lengths and instructions are also defined.

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.

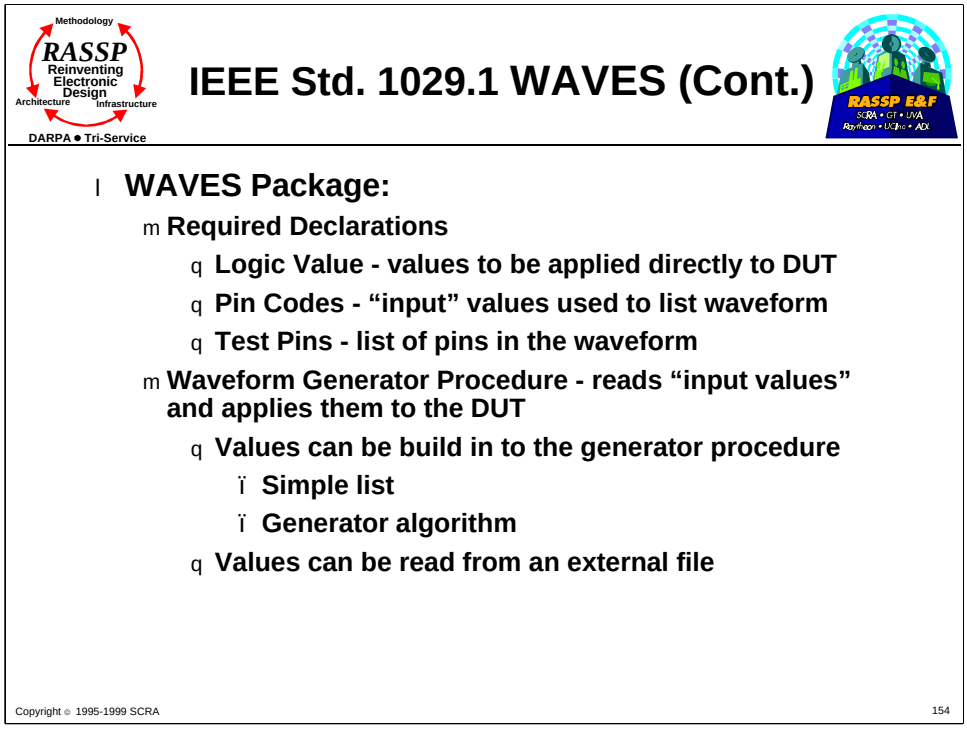
Ref: [IEEEP1149.5]



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.

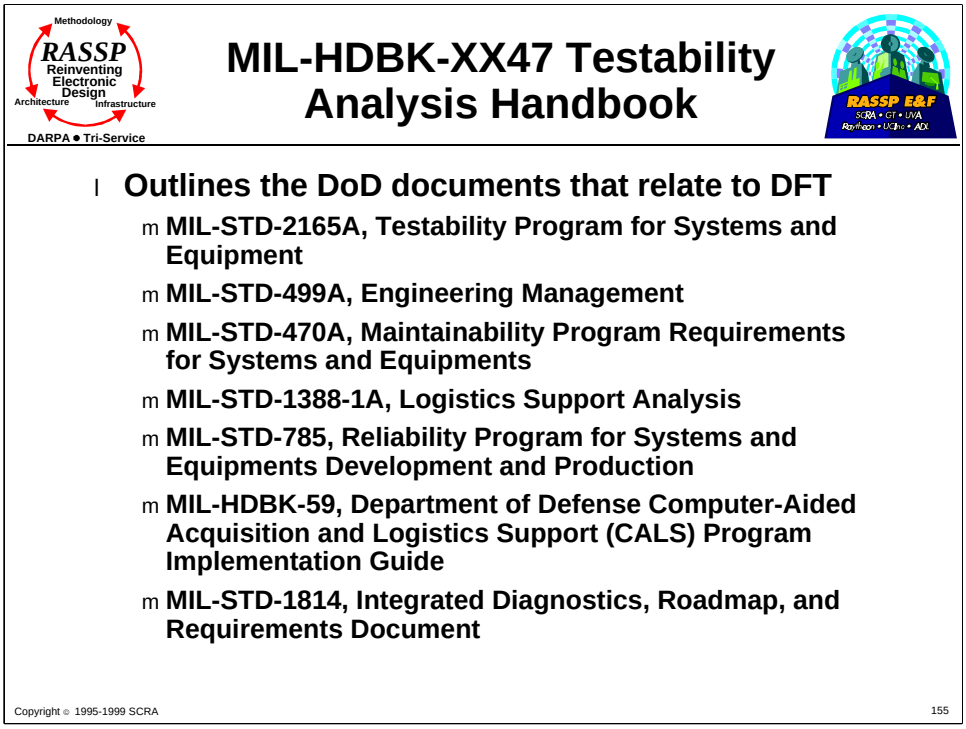
WAVES consists mainly of separate packages that are instantiated in a test bench for a device-under-test.



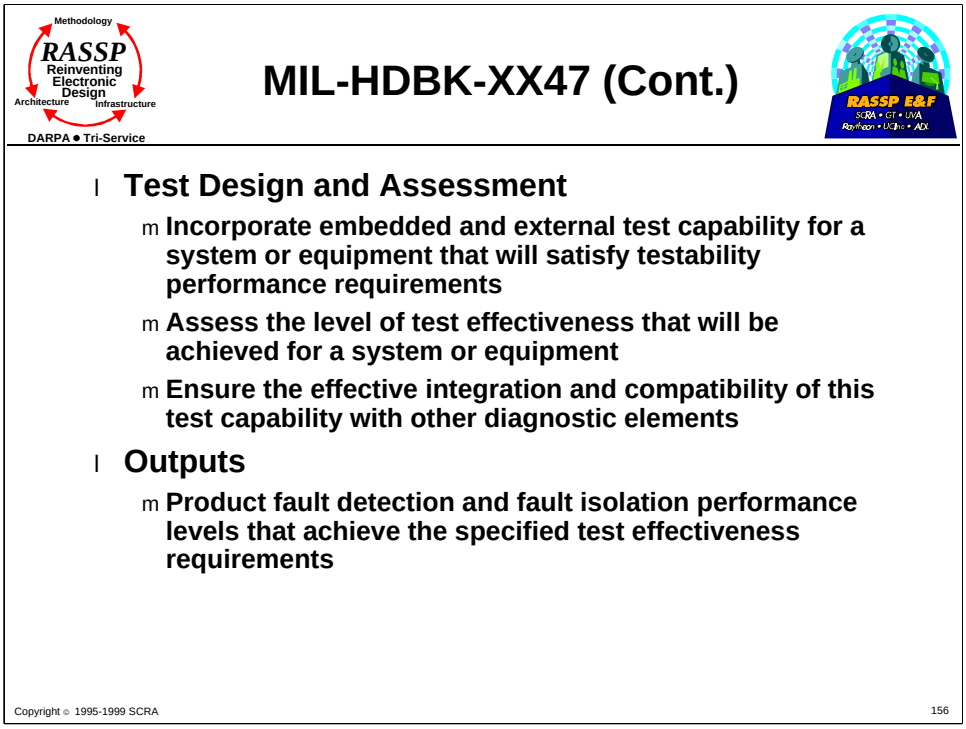
There are standard WAVES packages that define procedures like apply. The user defined WAVES package (which uses the predefined packages) consists of required definitions and the waveform generator procedure.

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).



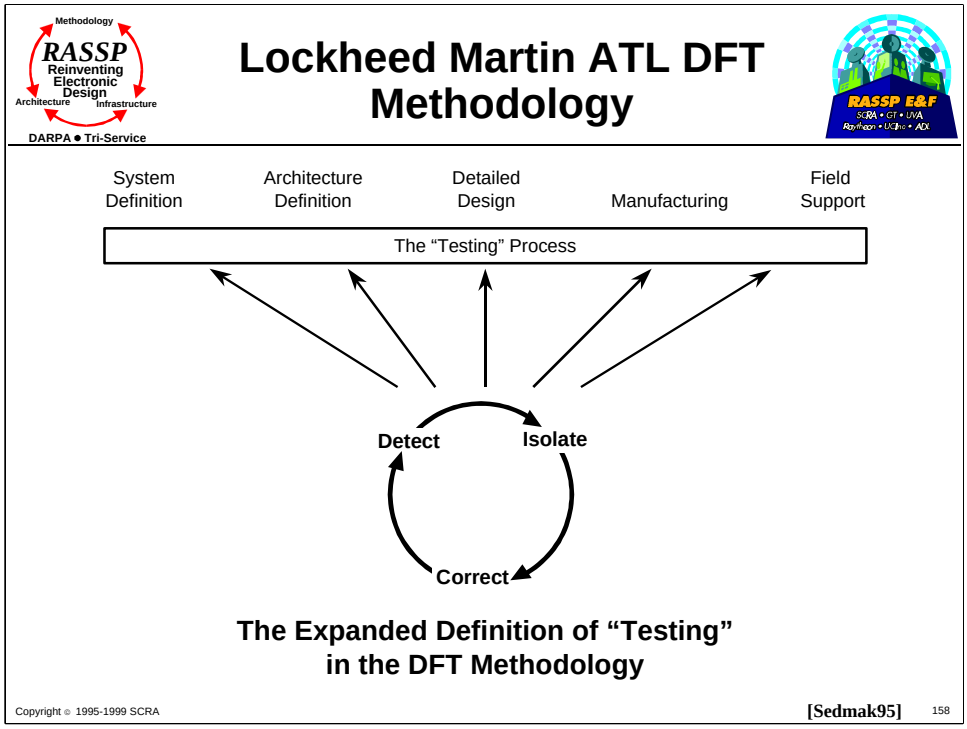
MIL-HDBK-XX47 outlines the DoD documents that outline the process and product of Design for Test for Military systems. Some detail is included to show how these documents relate to the process and each other.



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.

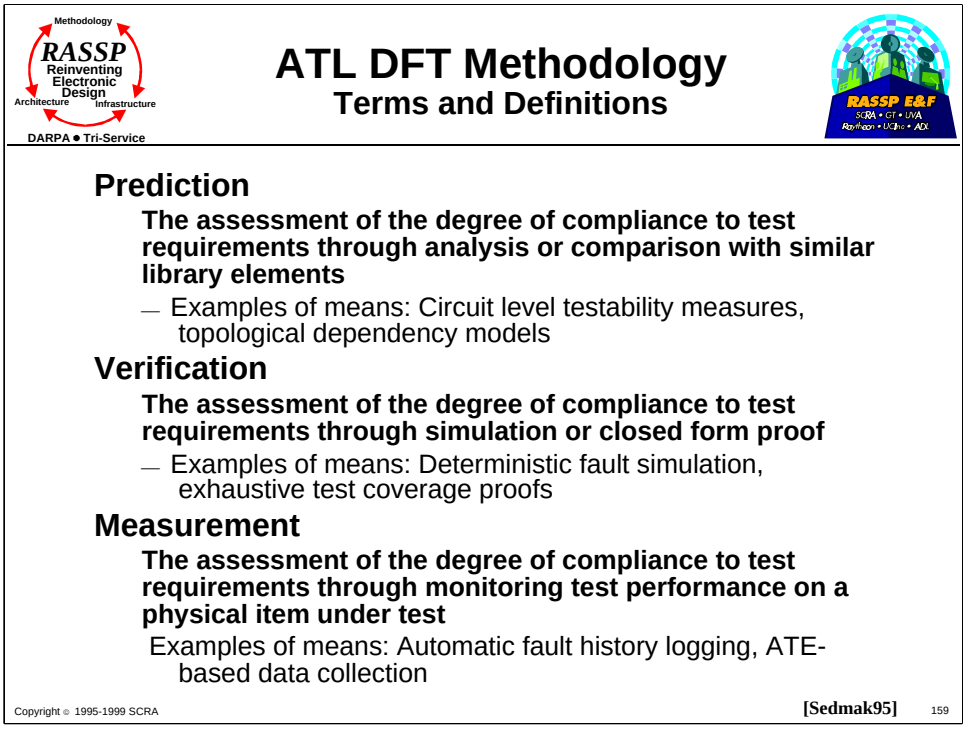
Module Outline

- 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



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.

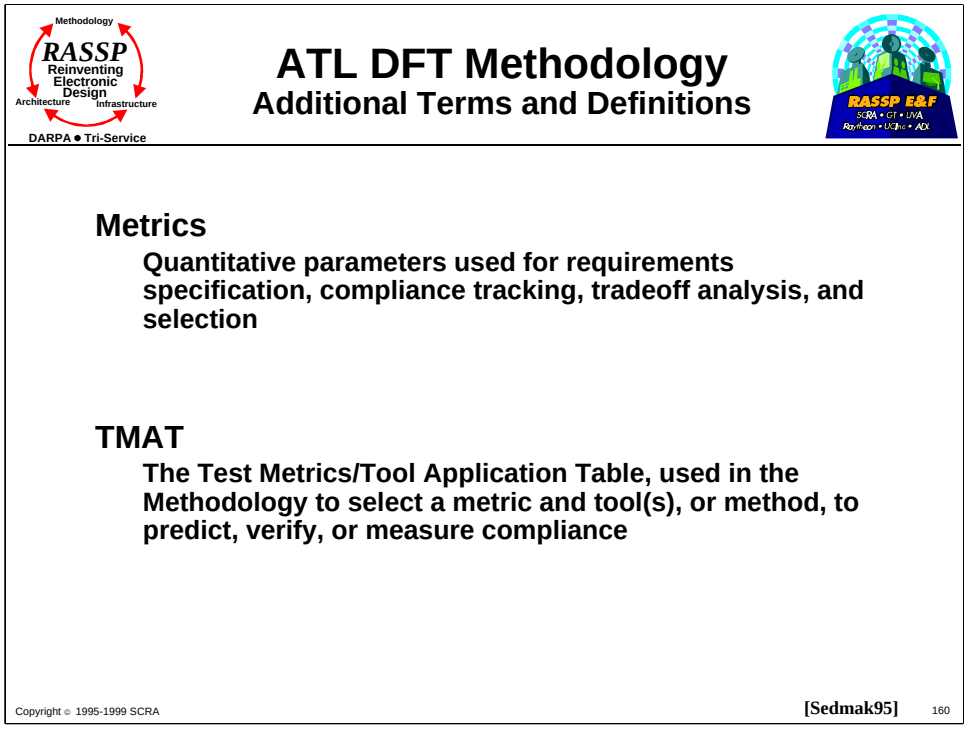
Ref [Sedmak95]



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

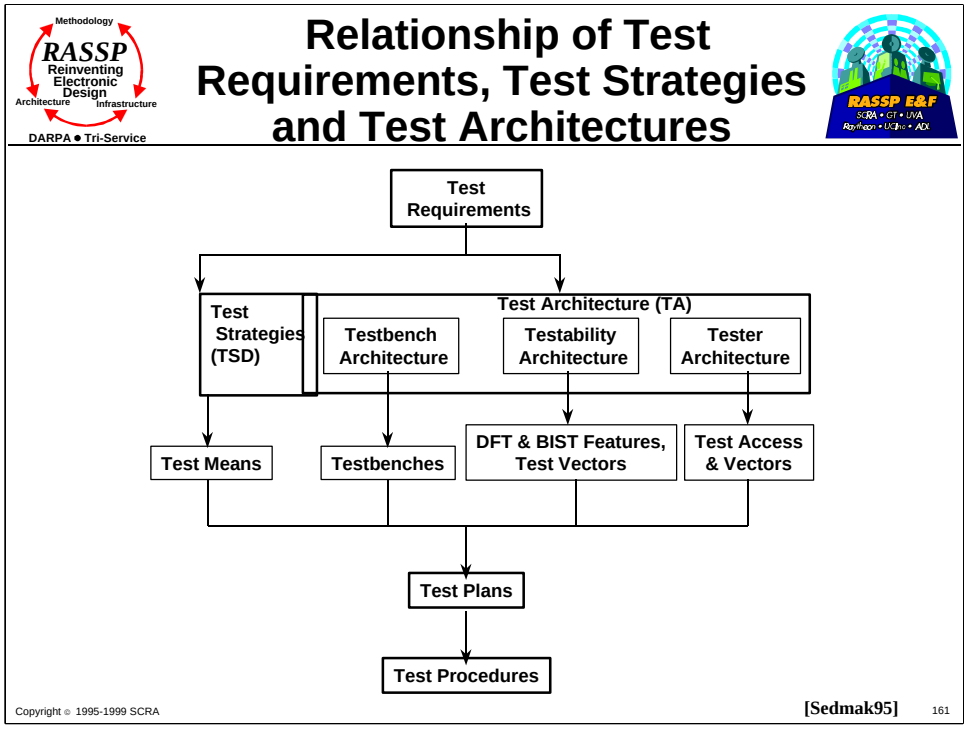
Automatic fault history logging automatically registers which fault have been detected, isolated, or corrected at each step in the process.

Ref [Sedmak95]



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.

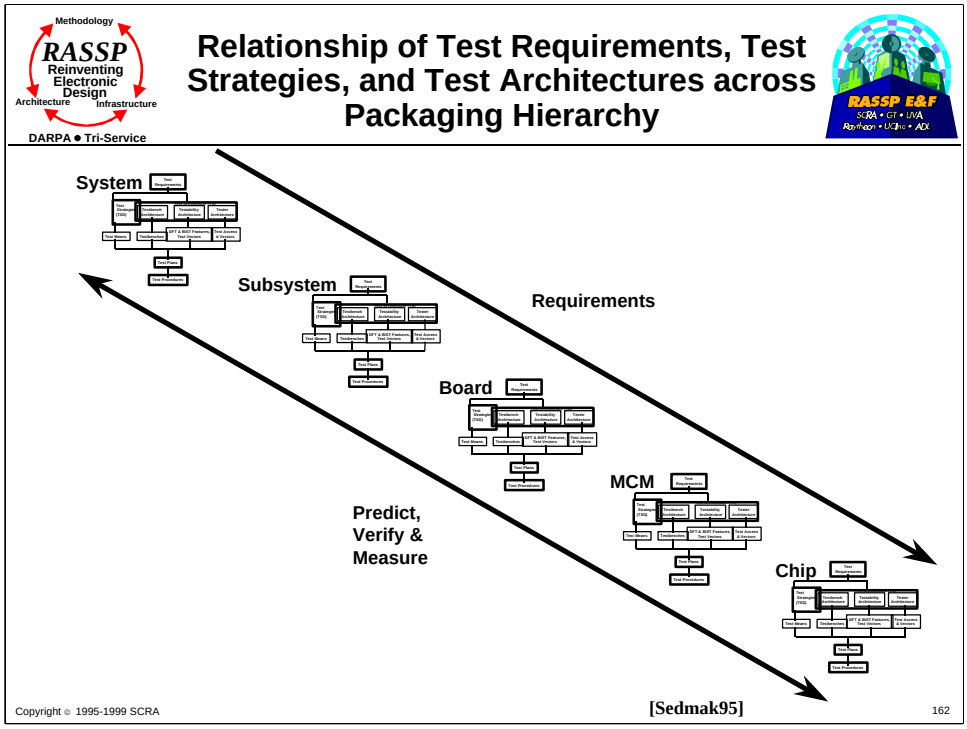
Testbenches define testing during VP stage.

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.

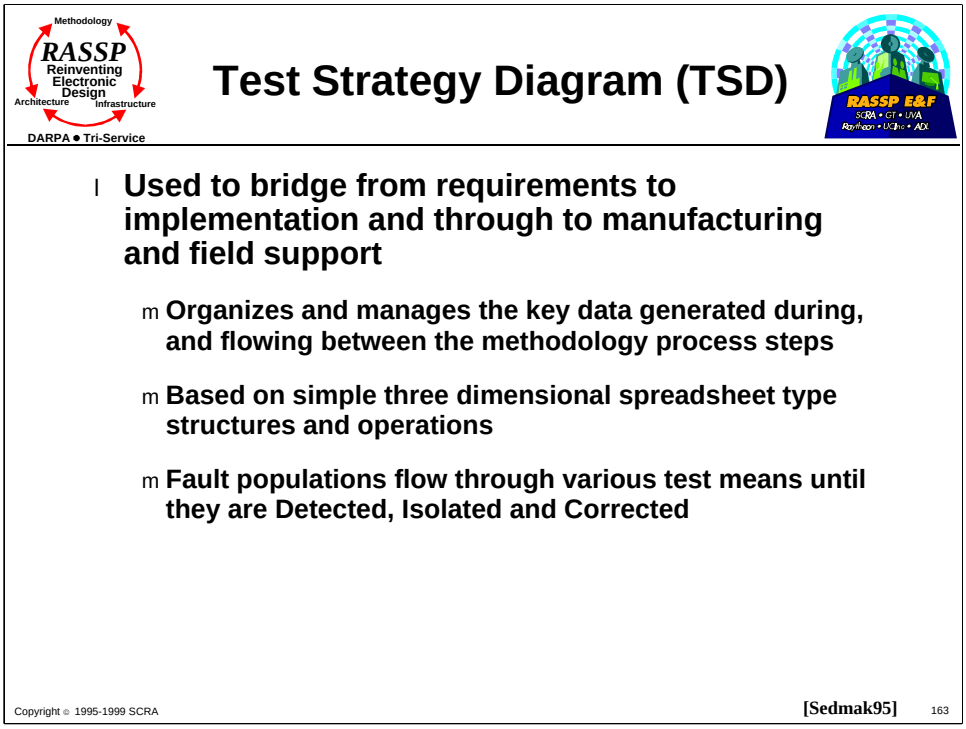
Ref [Sedmak95]



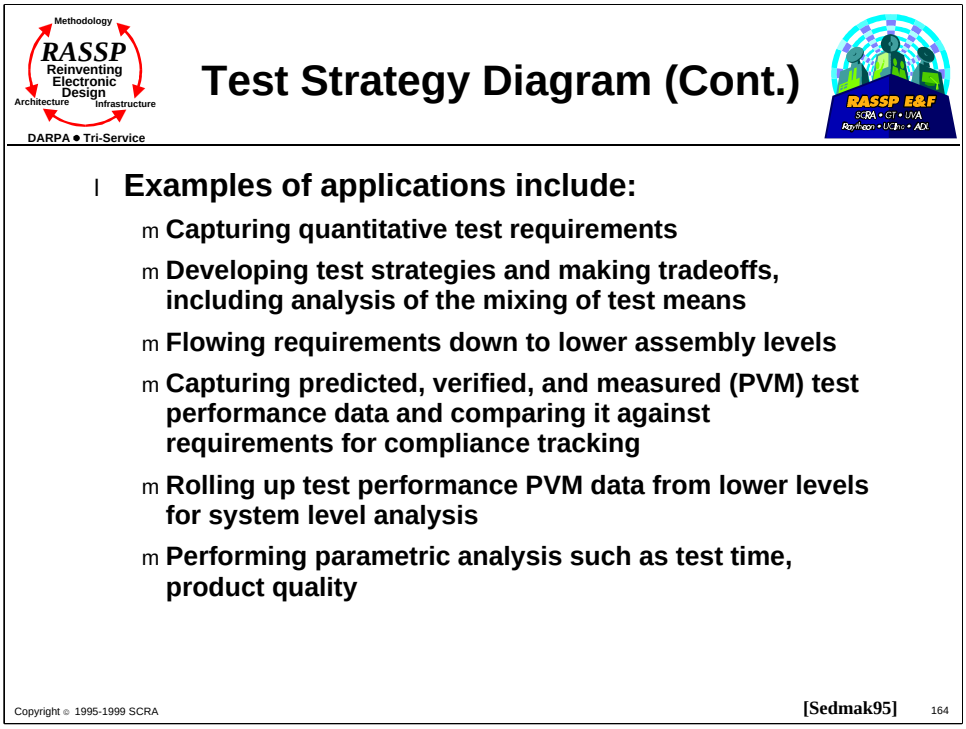
Requirements (TSD, TA) are flowed down which preserves test philosophy.

PVM measures are flowed back up (feedback).

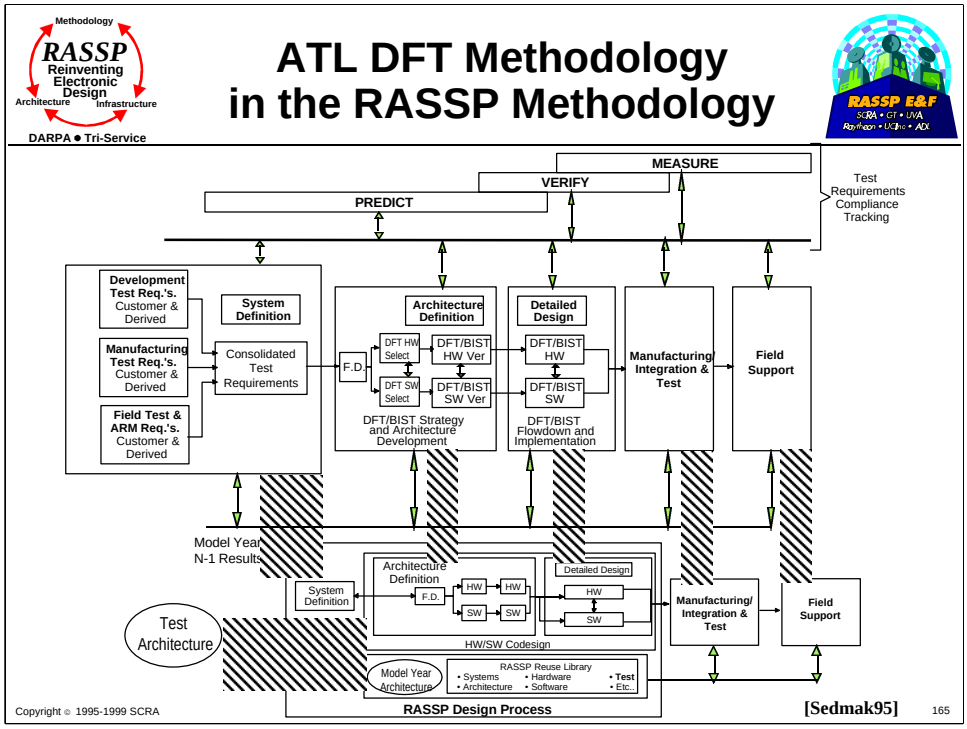
Ref [Sedmak95]



Ref [Sedmak95]



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.

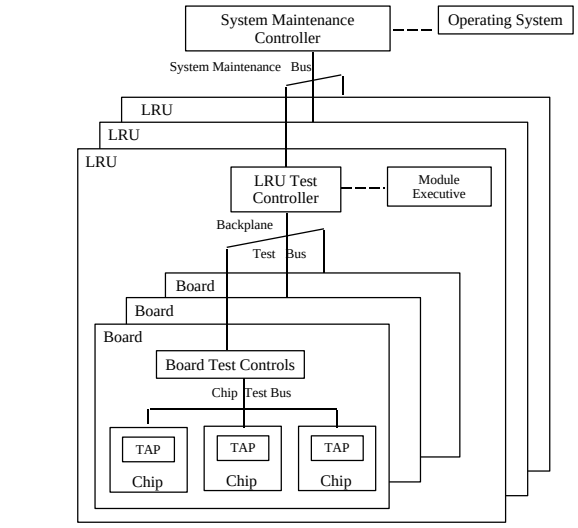
Manufacturing - exploits DFT methodology, gains feedback on actual defects (MYA, reuse!)

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]

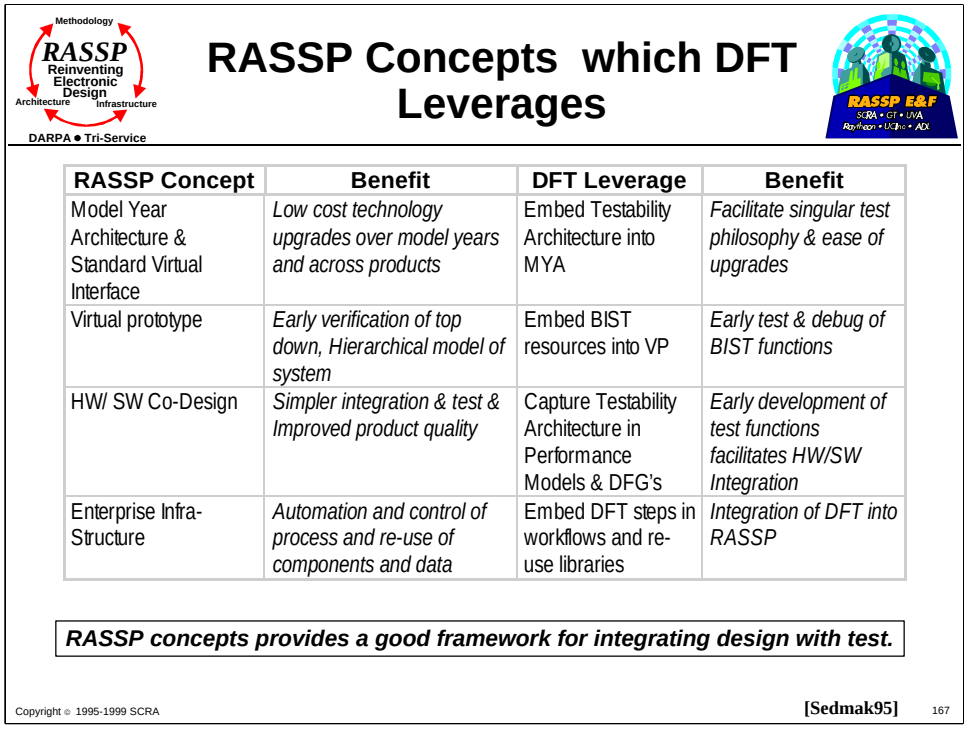
ATL HW Testability Architecture



- **Bias towards Boundary Scan and BIST but accommodate COTS and use ATE when required and/or practical**
- **"Lead - Follow - or Get out of the Way" philosophy for MCM/ board and system test**
- **Hierarchical array of dedicated Test & Maintenance Busses and controllers (IEEE 1149.1, P1149.5, Scan Bridge™, and/or ASP™ as required)**
- **Re-useable (within and between packaging level) BIST controllers, PRPG's and SAR's**
- **Accommodates 'Spoilers'**
  - **COTS**
  - **NDI**
  - **Development Only Contracts**
  - **Expendables**

[Sedmak95] 166

Ref [Sedmak95]



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

[Sedmak95]

Design-For-Test Integration Tasks

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

[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**

Summary

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

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

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., Gadient, 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, "I_{DDQ} 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.

[Waicukauski87] Waicukauski, 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