UNIT - IV
Fault Modeling & Test Pattern Generation

P. VIDYA SAGAR (ASSOCIATE PROFESSOR)

https://potharajuvidyasagar.wordpress.com
 CONTENT


Fault Diagnosis of Digital Systems

- Digital systems, even when designed with highly reliable components, do not operate forever without developing some faults. When a system ultimately does develop a fault it has to be detected and located so that its effect can be removed.

- **Fault detection** means the discovery of something wrong in a digital system or circuit.

- **Fault location** means the identification of the faults with components, functional modules or subsystems, depending on the requirements.

- **Fault diagnosis** includes both fault detection and fault location.
Fault detection

- Fault detection in a logic circuit is carried out by applying a sequence of test inputs and observing the resulting outputs, the cost of testing includes the generation of test sequences and their application.

- One of the main objectives in testing is to minimize the length of the test sequence.

- For a combinational circuit, the total number of input combination needed for testing is given by $2^n$.

- For a sequential circuit with $n$ inputs and $m$ flip-flops the total number of input combinations necessary to test the circuit exhaustively is $2^n * 2^m = 2^{m+n}$

$n$ = number of input wires

$m$ = number of flip flops
Test Generation for Combinational Logic Circuits

➢ There are several methods available for deriving tests for combinational circuits

➢ All these methods are based on the assumption that the circuit under test is non-redundant (A circuit is said to be non-redundant if the function realized by the circuit is not the same as the function realized by the circuit in the presence of a fault) and only a single stuck-at fault is present at any time.

➢ Some of the methods available for combinational circuits testing include: Two ATPG categories:

➢ One dimensional path sensitization(**Path-based methods)

➢ Boolean difference(*Boolean-based methods)
Motivating Problem

➢ The basic principle involved in “path sensitizing” is to choose some path from the origin of the failure to the circuit output.

➢ The path is said to be “sensitized” if the inputs to the gates along the path are assigned values so as to propagate the fault along the chosen path to the output.

➢ We do not need to know Boolean expression.

➢ We can just find a test from circuit netlist.
• **Fault activation**: Assign gate inputs to generate appropriate value at fault site \((H)\)
  - \(A=0\)
• **Sensitization**: Assign side-inputs to non-controlling value to propagate fault effect forward
  - \(J=0\)
• **Justification**: Assign primary inputs to achieve desired values
  - \(B=1\)
Single Path Sensitization

• Single path sensitization (SPS) Algorithm:

1. **Fault activation (aka. Fault excitation):** Assign gate inputs to generate value at fault site; Desired value opposite to the faulty value (e.g. 0 for SA1)

2. **Fault effect propagation:** Select one single path from fault site to an output; Assign side inputs to sensitize fault effect along the path.

3. **Justification:** Assign primary inputs to justify desired values assigned in 1 & 2

   ➢ If justification fails, backtrack.

   ❖ Pros:
     1. Easy to implement
     2. No Boolean equation needed

   ❖ Cons: Q1: Too many paths to choose, which one is correct?
     Q2: Single-path sensitization not enough to detect all faults
Example

- Consider stuck-at-1 fault
- Fault activation: \( a = b = c = 1 \)
- Fault effect propagation: two propagation paths
  - Choose path \{G5\}. Want \( G_2 = 1 \)
  - \( a = d = 0 \) justification fails!
  - Backtrack! Choose another path \{G6\}. Want \( G_4 = 1 \) at \( c = 1, e = 0 \) justification succeeds
- Test pattern: \( abce' \) generated

![Circuit Diagram]
Q1: Generate a test pattern for $E$ SA1 fault. Choose path $ELHK$.
A:

Q2: (Cont’d) Backtrack to another path $EFJK$.
A:
Single Path Not Enough

Example:

1. Fault activation: \(a = b = 1\)
2. Fault propagation: Choose path \(\{G3-G6\}\). want \(G2 = G4 = G5 = 1\)
3. \(G4 = 1\) \(e = 0\) \(G5 = 0\) justification fails
4. Choose another path \(\{G4-G6\}\). Justification also fails.

SPS algorithm fails
Multiple Path Sensitization

- This fault requires *multiple path sensitization*
- Both two paths \{G3-G6\} and \{G4-G6\} are sensitized
  - Error is propagated along both paths simultaneously
  - \( G2 = G5 = 1 \), \( c = e = 1 \)
  - Test generated successfully.

![Diagram of a circuit with gates and inputs a, b, c, e, and G1, G2, G3, G4, G5, and G6, with error propagation and test generation.]
Boolean Difference

- The basic principle of the Boolean difference is to derive two Boolean expressions – one of which represents normal fault-free behavior of the circuit and the other represents the logical behavior under an assumed single s-a-1 or s-a-0 fault condition.

- These two expressions are then exclusive ORed; if the result is 1 a fault is indicated.

- Let \( F(x) = F(X_1, \ldots, X_n) \) be a logic function of \( n \) variables.

- If one of the inputs to the logic function, e.g. input \( X_i \), is faulty, then the output would be \( F(X_1 \ldots X_i \ldots X_n) \).

- The Boolean difference of \( F(X) \) with respect to \( X_i \) is defined as:

\[
\frac{dF(x_1, \ldots, x_i, \ldots, x_n)}{dx_i} = \frac{dF(X)}{dx_i} = F(x_1, \ldots, x_i, \ldots, x_n) \oplus F(x_1, \ldots, \bar{x_i}, \ldots, x_n)
\]
Decomposition

- Consider a circuit that realizes the function $f(a, b, c, ...)$
- Fixed $a$ to one:
  - positive cofactor of $f$ with respect to $a$
  - $f_{a=1} = f(a = 1, b, c, ...)$
- Fixed $a$ to zero:
  - negative cofactor of $f$ with respect to $a$
  - $f_{a=0} = f(a = 0, b, c, ...)$
- Shannon’s Expansion w.r.t. input $a$ [Shannon 1948]

$$f = a \ f_{a=1} + a' \ f_{a=0}$$

![AND gate diagram]
Shannon’s Expansion for output K w.r.t. input B
Boolean Difference

• To detect a stuck-at zero fault
  ➢ Good output (f) and faulty output (fa=0) are different: \( f \oplus f_{a=0} = 1 \)
  After Shannon Expansion:
  \[
  [af_{a=1} + a'f_{a=0}] \oplus f_{a=0} = a[f_{a=1} \oplus f_{a=0}] + a'[f_{a=0} \oplus f_{a=0}] = a[f_{a=1} \oplus f_{a=0}] + 0 = 1
  \]
  ➢ Thus: \( a[f_{a=0} \oplus f_{a=1}] = 1 \)

• To detect a stuck-at one fault
  ➢ Good output (f) and faulty output (fa=0) are different: \( f \oplus f_{a=1} = 1 \)
  After Shannon Expansion:
  \[
  [af_{a=1} + a'f_{a=0}] \oplus f_{a=1} = 1
  \]
  ➢ Thus: \( a'[f_{a=0} \oplus f_{a=1}] = 1 \)

• Boolean difference of f w.r.t. a
  \[
  \frac{df}{da} = [f_{a=0} \oplus f_{a=1}]
  \]
Test Generation Example

- \( f = (a + b)c + c'd \)
- Set of all tests for \( c \) stuck-at-0 is
  \[
  \frac{df}{dc} = 1
  \]

- Set of all tests = \{1x10, x110, 0011\} (x = don’t care inputs)
- One fully specified test pattern: e.g. 1110
- One partially specified test pattern (aka. test cube): 1x10
EXAMPLE

- We already known \( K = A'B'C + A'B \)
- Q1: Boolean difference \( \frac{dK}{dA} = ? \)
- Q2: Use BD to find all test patterns for A stuck-at one fault.
Internal Faults

- Same approach also used for stuck-at faults internal faults
- Let $g$ be internal signal of Boolean function $f$

$$f(x_1, x_2, \ldots, y_1, y_2, \ldots) = W(g, y_1, y_2, \ldots)$$

Test sets for faults

- $g_{SA0}$: $\frac{dW}{dg} = 1$
- $g_{SA1}$: $\frac{dW}{dg} = 1$
EXAMPLE

Tests for $g_{sa0}$
Tests for $g_{sa1}$
EXAMPLE

\[ f = (a + b)c + c'd \]

\[ g = a + b \quad W = gc + c'd \]

\[
\frac{dW}{dg} = f_{g=0} \oplus f_{g=1} = cd' \oplus (c + c'd) = c
\]

Tests for \( g \text{ sa0} \) : \((a + b)c = ac + bc\)
Tests for \( g \text{ sa1} \) : \((a + b)'c = a'b'c\)
Chain Rule

- **Chain rule:**
  \[
  \frac{df}{dx_i} = \frac{df}{dg} \frac{dg}{dx_i}
  \]

- **Example**
  \[g = ab\]
  \[W = g + cd\]
  \[
  \frac{dW}{dg} = 1 \oplus cd = c' + d'
  \]
  \[
  \frac{df}{da} = \frac{dW}{dg} \frac{dg}{da} = (c' + d')b
  \]
  \[
  \frac{dg}{da} = b
  \]
EXAMPLE
Some useful properties of the Boolean difference are:

1. \( \frac{d \overline{F(X)}}{dx_i} = - \frac{dF(X)}{dx_i} \) if \( F(X) \) denotes the complement of \( F(X) \)
2. \( \frac{dF(X)}{dx_i} = \frac{dF(X)}{dx_i} \)
3. \( \frac{d}{dx_i} \cdot \frac{dF(X)}{dx_j} = \frac{d}{dx_j} \cdot \frac{dF(X)}{dx_i} \)
4. \( \frac{d[F(X) \cdot G(X)]}{dx_i} = F(X) \frac{dG(X)}{dx_i} \oplus G(X) \frac{dF(X)}{dx_i} \oplus \frac{dF(X)}{dx_i} \cdot \frac{dG(X)}{dx_i} \)
5. \( \frac{d[F(X) + G(X)]}{dx_i} = \frac{dF(X)}{dx_i} \oplus \frac{dG(X)}{dx_i} \oplus \frac{dF(X)}{dx_i} \cdot \frac{dG(X)}{dx_i} \)

A Boolean function \( F(X) \) is said to be independent of \( x_i \) if and only if \( F(X) \) is logically invariant under complementation of \( x_i \), i.e. if

\[
F(x_1, ..., x_i, ..., x_n) = F(x_1, ..., \overline{x_i}, ..., x_n)
\]

This implies that a fault in \( x_i \) will not affect the final output \( F(X) \) and \( \frac{dF(X)}{dx_i} = 0 \). Some additional properties can now be added to the original set:

6. \( \frac{dF(X)}{dx_i} = 0 \) if \( F(X) \) is independent of \( x_i \)
7. \( \frac{dF(X)}{dx_i} = 1 \) if \( F(X) \) is depends only on \( x_i \)
8. \( \frac{d[F(X) \cdot G(X)]}{dx_i} = F(X) \frac{dG(X)}{dx_i} \) if \( F(X) \) is independent of \( x_i \)
9. \( \frac{d[F(X) + G(X)]}{dx_i} = \frac{dF(X)}{dx_i} \cdot \frac{dG(X)}{dx_i} \) if \( F(X) \) is independent of \( x_i \)
The D-algorithm

- The D-algorithm is a modification of the path sensitization method. Unlike the latter it is guaranteed to find a test vector for a fault if one exists.
- The D-algorithm has been specified very formally, and is suitable for computer implementation. It is the most widely used test vector generator.
- The primary difference between the D-algorithm and the path sensitization approach is that the D-algorithm always attempts to sensitize every possible path to the primary outputs.
D-algorithm

➢ D-algorithm uses symbols D and D’ to represent errors. If we use D to denote a 0/1 error (0 is the expected value and 1 is the observed value), then D’ denotes a 1/0 error (1 is the expected value and 0 is the observed value). The meanings of D and D’ can be exchanged as long as their uses are consistent throughout a chip-undertest.

➢ D-algorithm uses a cubical algebra for automatic generation of tests. Three types of cubes are considered:

1. Singular cube;
2. Primitive D-cube of a fault;
Singular Cover & Singular Cube

- **Singular Cover**: The Singular Cover of a logic gate is a compact representation of its truth table. i.e., 2-input AND gate.

- Each row of the singular cover is called a CUBE. Singular Cube is a compact representation of the truth table of a component.

- Singular cover for two-input AND gate:

<table>
<thead>
<tr>
<th>Truth Table</th>
<th>Singular Cover</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

https://potharaj vidyasagar.wordpress.com
### AND Gate

#### Truth Table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Singular Cover

<table>
<thead>
<tr>
<th>Singular covers</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC-1</td>
<td>0</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>SC-2</td>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SC-3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### NOR Gate

#### Truth Table

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Singular Cover

<table>
<thead>
<tr>
<th>Singular covers</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>SC-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>SC-2</td>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>SC-3</td>
<td>1</td>
<td>X</td>
<td>0</td>
</tr>
</tbody>
</table>
Primitive D-Cubes of Failure (P.D.C.F.s)

- A Primitive D-Cube of Failure for a fault in a circuit is a set of inputs to the circuit which bring the fault to the circuit output.

- To generate the P.D.C.F. for a fault
  - Generate singular covers for the circuit in both its faulted and fault-free states.
  - Intersect the P0 cubes of the fault free cover with the F1 cubes of the faulted cover and intersect the P1 cubes with the F0 cubes.

- F1 and F0 play analogous roles in the faulted cover to P1 and P0 in the fault-free cover.

- Intersection is defined by intersecting each element of the cubes according to the following intersection rules:
  \[
  0 \cap 0 = 0 \quad 0 \cap X = X \quad X \cap 0 = 0 \\
  1 \cap 1 = 1 \quad 1 \cap X = X \quad X \cap 1 = 1 \\
  X \cap X = X \\
  1 \cap 0 = D \\
  0 \cap 1 = D' 
  \]
AND Gate (Sa-0 Fault on o/p)

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y(NF)} \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y(F)} \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

No Fault

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y} \\
0 & X & 0 \\
X & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

Sa-0 Fault

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y} \\
X & X & 0 \\
\end{array}
\]

PDCF

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y} \\
1 & 1 & D' \\
\end{array}
\]

\[
\begin{align*}
0 \cap 0 &= 0 \\
1 \cap 1 &= 1 \\
X \cap X &= X \\
1 \cap 0 &= D \\
0 \cap 1 &= D'
\end{align*}
\]

AND Gate (Sa-1 Fault on o/p)

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y(NF)} \\
0 & 0 & 0 \\
0 & 1 & 0 \\
1 & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y(F)} \\
0 & 0 & 1 \\
0 & 1 & 1 \\
1 & 0 & 1 \\
1 & 1 & 1 \\
\end{array}
\]

No Fault

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y} \\
0 & X & 0 \\
X & 0 & 0 \\
1 & 1 & 1 \\
\end{array}
\]

Sa-1 Fault

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y} \\
X & X & 1 \\
\end{array}
\]

PDCF

\[
\begin{array}{ccc}
\text{A} & \text{B} & \text{Y} \\
0 & X & D \\
X & 0 & D \\
\end{array}
\]

\[
\begin{align*}
1 \cap 0 &= D \\
0 \cap 1 &= D'
\end{align*}
\]
### NAND Gate (Sa-0 Fault on o/p)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y(NF)</th>
<th>Y(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

- No Fault
- Sa-0 Fault
- PDCF

### NAND Gate (Sa-1 Fault on o/p)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y(NF)</th>
<th>Y(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- No Fault
- Sa-1 Fault
- PDCF

\[
0 \land 0 = 0 \land X = X \land 0 = 0 \\
1 \land 1 = 1 \land X = X \land 1 = 1 \\
X \land X = X \\
1 \land 0 = D \\
0 \land 1 = D'
\]
NOR Gate (Sa-0 Fault on o/p)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y(NF)</th>
<th>Y(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

No Fault

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sa-0 Fault

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PDCF

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[0 \land 0 = 0 \land X = X \land 0 = 0\]
\[1 \land 1 = 1 \land X = X \land 1 = 1\]
\[X \land X = X\]
\[1 \land 0 = D\]
\[0 \land 1 = D'\]

NOR Gate (Sa-1 Fault on o/p)

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y(NF)</th>
<th>Y(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

No Fault

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Sa-1 Fault

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PDCF

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

\[0 \land 0 = 0 \land X = X \land 0 = 0\]
\[1 \land 1 = 1 \land X = X \land 1 = 1\]
\[X \land X = X\]
\[1 \land 0 = D\]
\[0 \land 1 = D'\]
Develop PDCFs

Consider the output of the two-input AND gate SA1

![Logic Diagram](image-url)
Propagation D-Cube

➢ The propagation D-Cubes of a gate are those which cause the output of a gate to depend solely on one or more of its inputs (usually one). This allows a fault on this input to be propagated through the gate.

➢ PDC consists of a table for each circuit element which has entries for propagating faults on any one of its inputs to the output.
  • To generate PDC entry corresponding to any one column, D-intersect any two rows of SC which have opposite values (0 and 1) in that column.
  • There can be multiple rows for one column
**Propagation D cube**

- **AND Gate**
- **SC**:
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

- **NAND Gate**
- **SC**:
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>X</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **NOR Gate**
- **SC**:
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>X</td>
<td>0</td>
</tr>
</tbody>
</table>

- **PDC**
- **SC**:
  
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>D’</td>
<td>1</td>
<td>D’</td>
</tr>
<tr>
<td>1</td>
<td>D’</td>
<td>D’</td>
</tr>
</tbody>
</table>

- **PDC**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>D’</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>D’</td>
<td>D</td>
</tr>
</tbody>
</table>

- **PDC**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>D’</td>
<td>D</td>
</tr>
<tr>
<td>D’</td>
<td>0</td>
<td>D</td>
</tr>
</tbody>
</table>
D-Algorithm Steps

➢ Choose a stuck-at-fault at any of the nodes.
  • Choose a pdcf for generating the fault.
  • Choose an output and a path to the output and propagate the fault to the output by choosing pdc for all circuit elements on the path. (D-Drive)
  • Use the SC of all unassigned circuit elements to arrive at a consistent set of inputs. (back-propagate or consistency check)
D-algorithm steps

The use of D-algorithm to determine test patterns follows the steps shown below.

1) Select a primitive D-cube for the fault of which test vectors are to be determined.

2) Select propagation D-cubes from the logic gates in the path from the faulty node to the output. This allows the difference (D or D’) to be propagated to the output so that it can be observed. This is called the forward trace operation.

3) For all other logic blocks that are not involved with the sensitized path, try to match the cubes in their singular cover with the values determined so far. A consistent set of input values is the valid test vector. If a consistent set of input values cannot be found, no test vector can be found for this fault (e.g., the circuit is redundant).
D-Algorithm: PDCF Example

➢ Choose a fault say g s-a-0. Choose pDCF of gate 2 for generating this fault
➢ \((a \ b \ c \ d \ e \ f \ g \ h \ i) = (X \ X \ X \ 0 \ 0 \ X \ D \ X \ X)\)
D-Algorithm: D-Drive Example

- Propagate the fault to the o/p using pdc of gates 3 & 4

- \text{pdc 3: } (X X X 0 0 1 D D' X)
- \text{pdc 4: } (0 X X 0 0 1 D D' D')
D-Algorithm: Consistency Example

- Perform consistency operation for gate 1

- \( (X X X 0 0 1 D D' X) \)
- \( sc 1 (0 1 1 0 0 1 D D' D') \)
# D-Algorithm: Summary

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
<th>f</th>
<th>g</th>
<th>h</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Initial</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td><strong>pdcf 2</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>D</td>
<td>x</td>
</tr>
<tr>
<td><strong>pdc 3</strong></td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D</td>
<td>D'</td>
<td>x</td>
</tr>
<tr>
<td><strong>pdc 4</strong></td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D</td>
<td>D'</td>
<td>D'</td>
</tr>
<tr>
<td><strong>consis. 1</strong></td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>D</td>
<td>D'</td>
<td>D'</td>
</tr>
</tbody>
</table>
Example
## Example

### Circuit Diagram

![Circuit Diagram](image)

### Truth Table

<table>
<thead>
<tr>
<th></th>
<th>$a$</th>
<th>$b$</th>
<th>$c$</th>
<th>$d$</th>
<th>$e$</th>
<th>$f$</th>
<th>$g$</th>
<th>$h$</th>
<th>$Z$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Primitive D-cube (gate 3)</td>
<td>1</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td>$D$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Propagation D-cube (gate 5)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$D$</td>
<td>1</td>
<td>$\bar{D}$</td>
</tr>
<tr>
<td>Singular cover (gate 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Singular cover (gate 4)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$X$</td>
<td>0</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Singular cover (gate 2)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PODEM: Path-Oriented DECision Making

IDEAS:
1. Only allow assignments to PI only
   - Doesn’t assign internal nodes
   - Greatly reduces search tree
2. Assigned PI are then forward implication
   - No justification needed
3. Flip last PI assignment when two conditions:
   A. Fault not activated
   B. No propagation path to any output
PODEM consists of six steps:

- **Step 1.** Assume all primary inputs are \( x \), which are unassigned. Determine an initial *objective*; an objective is defined by a logic (0 or 1) value referred to as *objective logic level*. The initial objective is to select a logic value so that the fault to be detected is sensitized.

- **Step 2.** Select a primary input and assign a logic value that has good likelihood of satisfying the initial objective.

- **Step 3.** Propagate forward the value at the selected primary input in conjunction with \( X \)'s at the rest of the primary inputs by using the five-valued logic 0, 1, \( X \), \( D \), and \( D \).
Contd....

➢ **Step 4.** If it is a test, a $D$ or a $\bar{D}$ is propagated to the output of the circuit, exit; otherwise, assign the complement of the previous value to the primary input and determine whether it is a test.

➢ **Step 5.** Assign a 0 or a 1 to one more primary input, and go to step 4 to check whether the resulting combination is a test.

➢ **Step 6.** Continue with steps 4 and 5 until a test is found, or the fault is found to be undetectable.
Flowchart of PODEM

Set objective:
Backtrace to assign one PI to achieve objective

Forward implication

Is D or D' at PO?

Test generated

yes

Is there untried combination of assigned Pls?

yes

Backtrack: set untried combination of values on assigned Pl's.

no

no

Test possible w/ additional PI assignments?

maybe

No test Exists.

yes
PODEM Example
Let us illustrate the application of PODEM by deriving a test for fault \( l \) s-a-1 in the circuit shown in Figure. Since a test for fault \( l \) s-a-1 is to be derived, the initial objective is to set \( l \) to 0.

Either \( B \) or \( C \) can be assigned 1 to satisfy the objective. Assuming we choose \( B \) to be at 1, the result of the forward propagation is:

\[
\begin{align*}
A & B & C & l & m & n & p & F \\
0 & 1 & X & 0 & X & 0 & & X
\end{align*}
\]

The next objective is to propagate \( D \) (or \( D \)) through \( n \) to output \( F \). This can be done by assigning proper logic value to input \( C \). Suppose we set \( C \) to 1, this results in the following:

\[
\begin{align*}
A & B & C & l & m & n & p & F \\
0 & 1 & 1 & 1 & D' & 0 & 0 & X
\end{align*}
\]
This will block the propagation of $D$ because $n$ is forced to 0. However if $C$ is assigned 1, $D$ is propagated through $n$:

- $A \ B \ C \ l \ m \ n \ p \ F$
- $X\ 1\ 0\ D'\ 0\ D\ D'\ X$

The final objective is to propagate $D$ (or $D'$) to output $F$. This can be done by assigning proper logic value i.e. 0 to input $A$.

- $A \ B \ C \ l \ m \ n \ p \ F$
- $0\ 1\ 0\ D'\ 0\ D\ D'\ D$

Thus, $ABC=010$ is the test for $ls\ a\ l$. 

![Logic Diagram](https://potharajuvidyasagar.wordpress.com)
PODEM Example
PODEM Example
2. Initialize all signals to Xs.
3. Set objective: G2 with output at 1[0].
4. Backtrace: Use Table 1.11 and Table 1.13 to set inputs to G2 to 0 and 0.
5. Set PI values \( B = 0 \) and \( C = 0 \).
6. Simulate with \( A = X, B = 0, C = 0, \) and \( D = X \). Current fault tested at output of G2 (output of G2 has 1[0] in simulation above), but not propagated to a PO.
7. Select Z-path from fault to PO Y through G5 and G8.
8. Set objective: G5 output at 0[1].
9. Backtrace: Set the top input to G5 to 0.
10. Set PI value \((A = 0)\).

11. Simulate with \((A = 0, \ B = 0, \ C = 0, \text{ and } \ D = X)\). Current fault tested at output of G5, but propagated to a PO.

12. Set objective: G8 output at 1[0].

13. Backtrace: Note in simulation above: first input to G8 already set to 0, second input set to 0[1], third input to 0[X], the last input to X.

14. Set objective: G7 output to 0.

15. Backtrace: Set first input to G7 to 1 (second input already set to 0 because \(C = 0)\). Set second input to G3 to 0 (first input already set to 0 because \(B = 0)\).

16. Set PI value \((D = 0)\).

17. Simulate with \((A = 0, \ B = 0, \ C = 0, \text{ and } \ D = 0)\). Current fault detected at PO y with 1[0].
Kohavi algorithm:

➢ This technique has two sets of tests, namely a-tests and b-tests and considers the altered Boolean function realization by the circuit due to the presence of a single fault rather than considering the faults itself. For the effective implementation of this algorithm, three restrictions are imposed by Kohavi.

➢ The first one is that, the network must be a two-level AND-OR (or) OR-AND network.

➢ The second restriction is that, each AND gate must realize a prime cube.

➢ The third one is that, the AND-OR network must implement a Boolean function which is a sum of irredundant prime implicant.
Consider a SA0 fault on any of the inputs of an AND gate. The effect of this fault on the output function is the elimination of the prime implicant realized by the AND gate. The set of distinguished minterms that tests each AND gate for SA0 faults is called the set of a-tests. The set of minterms that tests each AND gate for SA1 faults is called the set of b-sets.

The circuit above realizes the function $f = x_1 x'_2 + x_3 x_4$. The prime implicants of the function are $x_1 x'_2 = 1022$ and $x_3 x_4 = 2211$ which can be found from the Karnaugh map.
The set of minterms that test each AND gate for SA0 faults is called the set of a-tests.

- Consider a SA0 fault on any of the inputs of an AND gate. The effect of this fault on the output function is the elimination of the prime implicant realized by the AND gate. This can be checked any one of the distinguished minterms of the prime cube.

Figure (a) AND–OR network realizing the function $f = x_1 x'_2 + x_3 x_4$;
The set of minterms that test each AND gate for SA1 faults is called the set of b-tests.

\[ x_1 \quad x_2' \quad 1022 \]
\[ x_3 \quad x_4 \quad 2211 \]

Which are shown from the K-Map.

Suppose that there is SA1 fault in the input line \( x_2' \). The effect of this fault is change the prime implicant \( x_1 \quad x_2' \) to \( x_1 \). In other words, the cube \( 1022 \) expands to the cube \( 1222 \). Thus the fault can be detected choosing, as test vector, a minterm that is in the subcube \( 1122 \), which is adjacent to the original cube.
b-tests:

• Another consideration is that the minterm is in the offset of the function.

• “To test an SA1 fault in any input of an AND gate, it is necessary to select a test minterm in each of the adjacent subcubes of the prime implicant realized by the AND gate. The set of all such minterms constitutes the set of b-tests.”
The Kohavi Algorithm

\[ f = x_1 x_2 + x_1 x_3' x_4' + x_2 x_4 \]
Pseudorandom Testing with Signature Analysis:

- When pseudorandom testing is combined with signature analysis, certain hardware efficiency can be realized by utilizing special multipurpose LFSR architectures. In addition, when these LFSRs are configured as memory elements of the functional design, even more hardware efficiency is obtained, along with an improved level of controllability, observability, and testability of the built-in self-test circuit.
Transition Count testing:

The transition count is defined as the total number of transitions of $1 \to 0$ and $0 \to 1$ in an output response sequence corresponding to a given input test sequence. For example, if an output response sequence $Z=10011010$, then the transition count $c(Z)=4$. Thus, instead of recording the entire output response sequence, only the transition count is recorded. The transition count is then compared with the expected one, and if they differ, the circuit under test is declared faulty.

![Transition Count Diagram](attachment:transition_count.png)
Signature Analysis:

- Signature analysis technique is pioneered by Hewlett-Packard Ltd. that detects errors in data streams caused by hardware faults. It uses a data compaction technique to reduce long data streams into a unique code called the *signature*. Signatures can be created from the data streams by feeding the data into an $n$-bit LFSR. The feedback mechanism consists of EX-ORing selected taps.

![Signature Analysis Diagram]

- Feedback from the shift register
- To data display
- EX-OR
- $n$-bit shift register
- Clock
- Start
- Stop