**Major Combinational Automatic Test-Pattern Generation Algorithms**

- Definitions
- D-Algorithm (Roth) – 1966
  - D-cubes
  - Bridging faults
  - Logic gate function change faults
- PODEM (Goel) – 1981
  - X-Path-Check
  - Backtracing
- Summary

---

**Forward Implication**

- Results in logic gate inputs that are significantly labeled so that output is uniquely determined
- AND gate forward implication table:

```
  D 1 X D
  0 0 0 0 0
  1 0 1 X D
  X 0 X X X
  D 0 D X D
  0 D D X 0
```

---

**Backward Implication**

- Unique determination of all gate inputs when the gate output and some of the inputs are given

---

**Implication Stack**

- Push-down stack. Records:
  - Each signal set in circuit by ATPG
  - Whether alternate signal value already tried
  - Portion of binary search tree already searched

<table>
<thead>
<tr>
<th>Signal</th>
<th>Value</th>
<th>Alternative tried</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>NO</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>NO</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>NO</td>
</tr>
<tr>
<td>R</td>
<td>0</td>
<td>YES</td>
</tr>
</tbody>
</table>
Objectives and Backtracing of ATPG Algorithm

- **Objective** – desired signal value goal for ATPG
  - Guides it away from infeasible/hard solutions
- **Backtrace** – Determines which primary input and value to set to achieve objective
  - Use testability measures

D-Algorithm – Roth IBM (1966)

- Fundamental concepts invented:
  - First complete ATPG algorithm
  - D-Cube
  - D-Calculus
  - Implications – forward and backward
  - Implication stack
  - Backtrack
  - Test Search Space

Branch-and-Bound Search

- Efficiently searches binary search tree
- **Branching** – At each tree level, selects which input variable to set to what value
- **Bounding** – Avoids exploring large tree portions by artificially restricting search decision choices
  - Complete exploration is impractical
  - Uses heuristics

Implication Stack after Backtrack

<table>
<thead>
<tr>
<th>Stack ptr</th>
<th>B</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signal</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Value</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Alternative Input</td>
<td>E</td>
<td>F</td>
</tr>
</tbody>
</table>

Unexplored
Present Assignment
Searched and Infeasible

![Implication Stack Diagram](image_url)
**Singular Cover Example**

- Minimal set of logic signal assignments to show essential prime implicants of Karnaugh map

![Logic Circuit Diagram](image)

<table>
<thead>
<tr>
<th>Gate</th>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>A  B  d</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0    X</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>X    0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1    1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Gate</th>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOR</td>
<td>d  e  F</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>1    X</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>X    1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0    0</td>
<td>1</td>
</tr>
</tbody>
</table>

**D-Cube**

- Collapsed truth table entry to characterize logic
- Use Roth's 5-valued algebra
- Can change all D's to D's and D's to D's (do both)
- AND gate:

<table>
<thead>
<tr>
<th>Gate</th>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>A  B  d</td>
<td>X</td>
</tr>
<tr>
<td>1</td>
<td>0    1</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X    0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1    1</td>
<td>0</td>
</tr>
</tbody>
</table>

**D-Cube Operation of D-Intersection**

- ψ – undefined (same as 0)
- µ or λ – requires inversion of D and D
- D-intersection: \(0 \lor 0 \lor X = X \lor 0 = 0\)
  \[1 \lor 1 = 1 \lor X = X \lor 1 = 1\]
  \[X \lor X = X\]

<table>
<thead>
<tr>
<th>D-containment - Cube a contains Cube b if b is a subset of a</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cube a contains cube b if b is a subset of a</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D-Cube of Failure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Models circuit faults:</td>
</tr>
<tr>
<td>Stuck-at-0</td>
</tr>
<tr>
<td>Stuck-at-1</td>
</tr>
<tr>
<td>Bridging fault (short circuit)</td>
</tr>
<tr>
<td>Arbitrary change in logic function</td>
</tr>
<tr>
<td>AND Output sa0: 0 1 D</td>
</tr>
<tr>
<td>AND Output sa1: 0 X D</td>
</tr>
<tr>
<td>Wire sa0: D</td>
</tr>
<tr>
<td>Propagation D-cube - models conditions under which fault effect propagates through gate</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D-Cube</th>
<th>A</th>
<th>B</th>
<th>d</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>D</td>
<td>0</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Implication Procedure

1. Model fault with appropriate primitive D-cube of failure (PDF)
2. Select propagation D-cubes to propagate fault effect to a circuit output (D-drive procedure)
3. Select singular cover cubes to justify internal circuit signals (Consistency procedure)
   - Put signal assignments in test cube
   - Regrettably, cubes are selected very arbitrarily by D-ALG

Bridging Fault Circuit

Construction of Primitive D-Cubes of Failure

1. Make cube set $\alpha_1$ when good machine output is 1 and set $\alpha_0$ when good machine output is 0
2. Make cube set $\beta_1$ when failing machine output is 1 and $\beta_0$ when it is 0
3. Change $\alpha_1$ outputs to 0 and D-intersect each cube with every $\beta_0$. If intersection works, change output of cube to $\beta_0$
4. Change $\alpha_0$ outputs to 1 and D-intersect each cube with every $\beta_1$. If intersection works, change output of cube to $\beta_1$

Bridging Fault D-Cubes of Failure

<table>
<thead>
<tr>
<th>Cube-set $\alpha$</th>
<th>a</th>
<th>b</th>
<th>$a^*$</th>
<th>$b^*$</th>
<th>Cube-set $\beta$</th>
<th>a</th>
<th>$a^*$</th>
<th>b</th>
<th>$b^*$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha_0$</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0 X 0 X</td>
<td>$\beta_0$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$\alpha_1$</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1 X 1 X</td>
<td>$\beta_1$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

PDFs for Bridging fault:

<table>
<thead>
<tr>
<th>a b</th>
<th>$a^<em>$ b $b^</em>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0</td>
<td>$D$</td>
</tr>
<tr>
<td>0 1</td>
<td>D 1</td>
</tr>
</tbody>
</table>
while (untried fault effects on D-frontier)  
select next untried D-frontier gate for propagation;  
while (untried fault effect fanouts exist)  
select next untried fault effect fanout;  
generate next untried propagation D-cube;  
D-intersect selected cube with test cube;  
if (intersection fails or is undefined) continue;  
if (intersection succeeded)  
if (intersection fails & in test cube) break;  
if (intersection succeeded)  
add propagation D-cube to test cube – recreate D-frontier;  
Find all forward & backward implications of assignment;  
save D-frontier, algorithm state, test cube, fanouts, fault;  
break;  
else if (intersection fails & D and D̅ in test cube) Backtrack ();  
else if (intersection fails) break;  
if (all fault effects unpropagatable) Backtrack ();

D-Algorithm – Top Level

1. Number all circuit lines in increasing level order from PIs to POs;  
2. Select a primitive D-cube of the fault to be the test cube;  
   - Put logic outputs with inputs labeled as D (5) onto the D-frontier;  
3. D-drive ();  
4. Consistency ();  
5. return ();

D-Algorithm – D-drive

while (untried fault effects on D-frontier)  
select next untried D-frontier gate for propagation;  
while (untried fault effect fanouts exist)  
select next untried fault effect fanout;  
generate next untried propagation D-cube;  
D-intersect selected cube with test cube;  
if (intersection fails or is undefined) continue;  
if (intersection succeeded)  
if (intersection fails & in test cube) break;  
if (intersection succeeded)  
add propagation D-cube to test cube – recreate D-frontier;  
Find all forward & backward implications of assignment;  
save D-frontier, algorithm state, test cube, fanouts, fault;  
break;  
else if (intersection fails & D and D̅ in test cube) Backtrack ();  
else if (intersection fails) break;  
if (all fault effects unpropagatable) Backtrack ();

D-Algorithm – Consistency

s = coordinates of test cube with 1’s & 0’s;  
if (s is only PIs) Backtrack ();  
for (each unjustified signal in s)  
Select highest # unjustified signal x in s, not a PI;  
if (inputs to gate x are both D and D̅) break;  
while (untried singular covers of gate x)  
select next untried singular cover;  
if (no more singular covers)  
if (no more stack choices) best alternative & break;  
else if (untried alternatives in Consistency)  
pop implication stack – try alternate assignment;  
else  
Backtrack ();  
D-drive ();  
if (singular cover D-intersects with x) delete x from s, add inputs to singular cover in s, find all forward and  
backward implications of new assignment, and break;  
if (intersection fails) mark singular cover as failed;
Backtrack

if (PO exists with fault effect) Consistency ();
else pop prior implication stack setting to try alternate assignment;
if (no untried choices in implication stack)
    fault untestable & stop;
else return;

Circuit Example 7.1 and Truth Table

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

Singular Cover & D-Cubes

- Singular cover – Used for justifying lines
- Propagation D-cubes – Conditions under which difference between good/failing machines propagates

Steps for Fault d sa0

<table>
<thead>
<tr>
<th>Step</th>
<th>A B C d e F</th>
<th>Cube type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1 1 D</td>
<td>PDF of AND gate</td>
</tr>
<tr>
<td>2</td>
<td>D 0 D</td>
<td>Prop. D-cube for NOR</td>
</tr>
<tr>
<td>3</td>
<td>1 1 0</td>
<td>Sing. Cover of NAND</td>
</tr>
</tbody>
</table>
Example 7.2 Fault A sa0

- Step 1 – D-Drive – Set $A = 1$

- Step 2 – Example 7.2
  - Step 2 – D-Drive – Set $f = 0$

- Step 3 – Example 7.2
  - Step 3 – D-Drive – Set $k = 1$

- Step 4 – Example 7.2
  - Step 4 – Consistency – Set $g = 1$
Step 5 – Example 7.2
- Step 5 – Consistency – $f = 0$ Already set

D-Chain Dies – Example 7.2
- Step 7 – Consistency – Set $B = 0$
- D-Chain dies
  
  Test cube: A, B, C, D, $a$, $f$, $g$, $h$, $k$, $L$

Step 6 – Example 7.2
- Step 6 – Consistency – Set $c = 0$, Set $a = 0$

Example 7.3 – Fault s $sa1$
- Primitive D-cube of Failure
Example 7.3 – Step 2 s sa1

- Propagation D-cube for $v$

Example 7.3 – Step 2 s sa1

- Forward & Backward Implications

Example 7.3 – Step 3 s sa1

- Propagation D-cube for $Z$ – test found!

Example 7.3 – Fault u sa1

- Primitive D-cube of Failure
Inconsistent

- $d = 0$ and $m = 1$ cannot justify $r = 1$ (equivalence)
  - Backtrack
  - Remove $B = 0$ assignment
Example 7.3 – Step 3 u sa1

- Propagation D-cube for \( v \)

Example 7.3 – Step 4 u sa1

- Propagation D-cube for \( z \)
- Propagation D-cube for \( Z \) and implications

PODEM – Goel

IBM (1981)

- New concepts introduced:
  - Expand binary decision tree only around primary inputs
  - Use X-PATH-CHECK to test whether D-frontier still there
  - Objectives – bring ATPG closer to propagating D (false) to PO
  - Backtracing
Motivation

- IBM introduced semiconductor DRAM memory into its mainframes – late 1970's
- Memory had error correction and translation circuits – improved reliability
  * D-ALG unable to test these circuits
  * Search too undirected
  * Large XOR-gate trees
  * Must set all external inputs to define output
  * Needed a better ATPG tool

PODEM High-Level Flow

1. Assign binary value to unassigned PI
2. Determine implications of all PIs
3. Test Generated? If so, done.
4. Test possible with more assigned PIs? If maybe, go to Step 1
5. Is there untried combination of values on assigned PIs? If not, with untestable fault
6. Set untried combination of values on assigned PIs using objectives and backtrace. Then, go to Step 2

Example 7.3 Again

- Select path s – Y for fault propagation

Example 7.3 – Step 2 s sa1

- Initial objective: Set r to 1 to sensitize fault
Example 7.3 – Step 3 s sa1
- Backtrace from r

Example 7.3 – Step 4 s sa1
- Set A = 0 in implication stack

Example 7.3 – Step 5 s sa1
- Forward implications: d = 0, x = 1

Example 7.3 – Step 6 s sa1
- Initial objective: set r to 1
Example 7.3 – Step 7 s sa1
- Backtrace from r again

Example 7.3 – Step 8 s sa1
- Set B to 1. Implications in stack: A = 0, B = 1

Example 7.3 – Step 9 s sa1
- Forward implications: k = 1, m = 0, r = 1, q = 1, Y = 1, s = D, u = D, v = D, Z = 1

Backtrack – Step 10 s sa1
- X-PATH-CHECK shows paths s -> Y and s -> u -> v -> Z blocked (D-frontier disappeared)
Step 11 -- s sa1

- Set B = 0 (alternate assignment)

Backtrack -- s sa1

- Forward implications: d = 0, X = 1, m = 1, r = 0, s = 1, q = 0, Y = 1, v = 0, Z = 1. Fault not sensitized.

Step 13 -- s sa1

- Set A = 1 (alternate assignment)

Step 14 -- s sa1

- Backtrace from r again
Step 15 – s sa1
- Set B = 0. Implications in stack: A = 1, B = 0

Backtrack – s sa1
- Forward implications: d = 0, X = 1, m = 1, r = 0. Conflict: fault not sensitized. Backtrack

Step 17 – s sa1
- Set B = 1 (alternate assignment)

Fault Tested – Step 18 s sa1
- Forward implications: d = 1, m = 1, r = 1, q = 0, s = B, v = B, X = 0, Y = B
Backtrace \((s, v_s)\) Pseudo-Code

\[
v = v_s;
\]

while \((s \text{ is a gate output})\)

\[
\text{if (s is NAND or INVERTER or NOR) } v = \overline{v}_i
\]

\[
\text{else if (objective requires setting all inputs) select unassigned input } a \text{ of } s \text{ with hardest controllability to value } v;
\]

\[
\text{else select unassigned input } a \text{ of } s \text{ with easiest controllability to value } v;
\]

\[
s = a;
\]

return \((s, v) /\ast \text{ Gate and value to be assigned }^{\ast}/\);  

Objective Selection Code

\[
\text{if (gate } g \text{ is unassigned) return } (g, \overline{v});
\]

\[
\text{select a gate } P \text{ from the D-frontier; select an unassigned input } i \text{ of } P.
\]

\[
\text{if (gate } g \text{ has controlling value }
\]

\[
\text{else if (0 value easier to get at input of XOR/EQUIV gate)}
\]

\[
\text{else } c = 0;
\]

\[
\text{return } (i, c);
\]

PODEM Algorithm

\[
\text{while (no fault effect at POs)}
\]

\[
\text{if (xpathcheck (D-frontier)}
\]

\[
(l, v_l) = \text{Objective (fault, } v_{\text{PO}})\)
\]

\[
(pl, v_{pl}) = \text{Backtrace } (l, v_l);
\]

\[
\text{imply } (pl, v_{pl});
\]

\[
\text{if (PODEM (fault, } v_{\text{fault}) == SUCCESS) return SUCCESS};
\]

\[
(pl, v_{pl}) = \text{Backtrace } (l);
\]

\[
\text{imply } (pl, v_{pl});
\]

\[
\text{if (PODEM (fault, } v_{\text{fault}) == SUCCESS) return SUCCESS};
\]

\[
\text{imply } (pl, \overline{v})
\]

\[
\text{return FAILURE};
\]

\[
\text{else if (implication stack exhausted)}
\]

\[
\text{return FAILURE};
\]

\[
\text{else Backtrace } (l);
\]

\[
\text{return (SUCCESS)}
\]

Summary

- D-ALG – First complete ATPG algorithm
- D-Cube
- D-Calculus
- Implications – forward and backward
- Implication stack
- Backup
- PODEM
- Expand decision tree only around PI
- Use X-PATH-CHECK to see if D-frontier exists
- Objectives – bring ATPG closer to getting D (0) to PO
- Backtracing