Overview
Major ATPG algorithms

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

Combinational ATPG

Forward Implication

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

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

Backward Implication

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

Implication Stack, Decision Tree, and Backtrack

Objectives and Backtracing in ATPG

- **Objective** — desired signal value goal for ATPG
  - Guides it away from infeasible/hard solutions
  - Uses heuristics
- **Backtrace** — Determines which primary input and value to set to achieve objective
  - Use heuristics such as nearest PI
- **Forward trace** — Determines gate through which the fault effect should be sensitized
  - Use heuristics such as output that is closest to the present fault effect
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
- Backtracking – Search fails, therefore undo some of the work completed and start searching from a location where search options still exist

D-Algorithm – Roth (1966)

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

Singular Cover Example

- Minimal set of logic signal assignments to represent a function
  – show prime implicants and prime implicates of Karnaugh map

<table>
<thead>
<tr>
<th>Gate</th>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>A</td>
<td>d</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td></td>
</tr>
<tr>
<td></td>
<td>d</td>
<td></td>
</tr>
<tr>
<td>NOR</td>
<td>d</td>
<td>e</td>
</tr>
<tr>
<td></td>
<td>F</td>
<td></td>
</tr>
</tbody>
</table>

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

Primitive D-Cube of Failure

- Models circuit faults:
  – Stuck-at-0
  – Stuck-at-1
  – Other faults, such as Bridging fault (short circuit)
  – Arbitrarily change in logic function
  – AND Output sa0: “1 D”
  – AND Output sa1: “0 X D”
  – Wire sa0: “D”
  – Propagation D-cube – models conditions under which fault effect propagates through gate

Construction of Primitive D-Cubes of Failure

1. Make cube set α1 when good machine output is 1 and set α0 when good machine output is 0
2. Make cube set β1 when failing machine output is 1 and β0 when it is 0
3. Change α1 outputs to 0 and D-intersect each cube with every β0. If intersection works, change output of cube to D
4. Change α0 outputs to 1 and D-intersect each cube with every β1. If intersection works, change output of cube to D

Gate Function Change D-Cube of Failure

Cube-set α b c Cube-set α b c
α0 0 X 0 PDFs for AND changing to OR 0 1 0 0
X 0 0 α1 1 1 1 D
β0 0 0 0 1 0 0 0 D
β1 1 X 1 X 1 1 1 D

Wire sa0: “D”
Propagation D-Cube

• Collapsed truth table entry to characterize logic
• Use Roth’s S-valued algebra
• AND gate: use the rules given earlier using α and β but in this case work with good circuit only

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>D</td>
</tr>
</tbody>
</table>

Write all primitive Cubes of AND gate and then create propagation cubes

<table>
<thead>
<tr>
<th>Cube a contains</th>
<th>Cube b if b is a subset of a</th>
</tr>
</thead>
<tbody>
<tr>
<td>$0 \land 0$</td>
<td>$0 = 0$</td>
</tr>
<tr>
<td>$0 \land 1$</td>
<td>$X = X$</td>
</tr>
<tr>
<td>$1 \land X$</td>
<td>$X = X$</td>
</tr>
</tbody>
</table>

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

Implication Procedure

1. Model fault with appropriate primitive D-cube of failure
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

D-Cube Operation of D-Intersection

\[
\psi = \text{undefined (same as } \phi) \\
\mu \text{ or } \lambda \text{ requires inversion of } D \text{ and } \overline{D}
\]

- **D-intersection**: $0 \land 0 = 0 \land X = X \land 0 = 0$
- $1 \land 1 = 1 \land X = X \land 1 = 1$
- $X \land X = X$

D-containment

- **Cube a contains**: $0 \land 0 = 0 \land X = X \land 0 = 0$
- **Cube b if b is a subset of a**

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 (D) 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 (all propagation D-cubes tried & failed) 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 D in test cube) Backtrack ();
      else if (intersection fails) Backtrack ();
      else (call fault effects untestable) Backtrack ();

D-Algorithm -- Consistency

$g =$ coordinates of test cube with 1’s & 0’s;
if ($g$ is only PIs) fault testable & stop;
for (each unjustified signal in $g$)
   select next untried singular cover;
   if (no more singular covers)
      If (no more stack choices) fault untestable & stop;
      else if (untried alternatives in Consistency)
         pop implication stack -- try alternate assignment;
      else
         Backtrack ();
   if (singular cover D-intersects with $z$) delete $z$ from $g$,
      add inputs to singular cover to $g$,
      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 0</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>

Steps for Fault \( d \) sa0

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

Example 7.3 – Fault \( u \) sa1

- Primitive D-cube of Failure

Example 7.3 – Step 2 \( u \) sa1

- Propagation D-cube for \( v \)
Example 7.3 – Step 2 \( u \) sa1
• Forward and backward implications

\[
\begin{array}{c}
\text{Inconsistent} \\
\text{• } d = 0 \text{ and } m = 1 \text{ cannot justify } r = 1 \text{ (equivalence)} \\
\quad \text{• Backtrack} \\
\quad \text{• Remove } B = 0 \text{ assignment}
\end{array}
\]

Example 7.3 – Backtrack
• Need alternate propagation D-cube for \( v \)

Example 7.3 – Step 3 \( u \) sa1
• Propagation D-cube for \( v \) and implications

Example 7.3 – Step 4 \( u \) sa1
• Propagation D-cube for \( Z \) and implications
PODEM -- Goel (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(0) 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, exit: 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 excite fault

Example 7.3 -- Step 3 s sa1

- Backtrace from r
Example 7.3 -- Step 4 sa1

- Set $A = 0$ in implication stack

Example 7.3 -- Step 5 sa1

- Forward implications: $d = 0$, $X = 1$

Example 7.3 -- Step 6 sa1

- Initial objective: set $r$ to 1

Example 7.3 -- Step 7 sa1

- Backtrace from $r$ again

Example 7.3 -- Step 8 sa1

- Set $B$ to 1. Implications in stack: $A = 0$, $B = 1$

Example 7.3 -- Step 9 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

Fault Tested -- Step 18 s sa1
- Forward implications: $d = 1$, $m = 1$, $r = 1$, $q = 0$, $s = D$, $v = D$, $X = 0$, $Y = D$.

Objective Selection Code

PODEM Algorithm
while (no fault effect at POs)
  if (senscheck (D-frontier))
    let $(l, v) = Objective (fault, v_{fault})$;
    $(pi, v_{pi}) = Backtrack (l, v)$;
    Imply $(pi, v_{pi})$;
  else if (PODEM (fault, $v_{fault}$) = SUCCESS) return (SUCCESS);
    $(pi, v_{pi}) = Backtrack ()$;
    Imply $(pi, v_{pi})$;
  else if (PODEM (fault, $v_{fault}$) = SUCCESS) return (SUCCESS);
    Imply $(pi, \neg v)$;
  else return (FAILURE);
else if (implication stack exhausted)
  Backtrack ();
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 PIs
  - Use X-PATH-CHECK to see if D-frontier exists
  - Objectives – bring ATPG closer to getting D(DD) to PO
  - Backtracing

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

Objectives and Backtracing in ATPG

- Objective – desired signal value goal for ATPG
  - Guides it away from infeasible/hard solutions
  - Uses heuristics

- Backtrace – Determines which primary input and value to set to achieve objective
  - Use testability measures

Bridging Fault D-Cubes of Failure

<table>
<thead>
<tr>
<th>Cube-set</th>
<th>a</th>
<th>b</th>
<th>a*</th>
<th>b*</th>
</tr>
</thead>
<tbody>
<tr>
<td>α0</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>0</td>
<td>X</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>α1</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>X</td>
<td>1</td>
<td>X</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>β0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>β1</td>
<td>X</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

PDFs for Bridging fault

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>a*</th>
<th>b*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>D</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Example 7.2 Fault \( A_{sa0} \)

- **Step 1 -- D-Drive -- Set \( A = 1 \)**

- **Step 2 -- D-Drive -- Set \( f = 0 \)**

- **Step 3 -- D-Drive -- Set \( k = 1 \)**

- **Step 4 -- Consistency -- Set \( g = 1 \)**

- **Step 5 -- Consistency -- \( f = 0 \) Already set**

- **Step 6 -- Consistency -- Set \( c = 0 \), Set \( e = 0 \)**
D-Chain Dies -- Example 7.2

- Step 7 – Consistency – Set B = 0
- D-Chain dies

Test cube: A, B, C, D, e, f, g, h, k, L

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!