Automatic Test Pattern Generation - II

Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Dept. of Electrical Engineering
Indian Institute of Technology Bombay
viren@ee.iitb.ac.in

EE 709: Testing & Verification of VLSI Circuits
Lecture – 12 (Jan 30, 2012)
ATPG - Algorithmic

- Path Sensitization Method
  - Fault Sensitization
  - Fault Propagation
  - Line Justification

- Path Sensitization Algorithms
  - D- Algorithm (Roth)
  - PODEM (P. Goel)
  - FAN (Fujiwara)
  - SOCRATES (Schultz)
  - SPIRIT (Emil & Fujiwara)
Common Concept

- Fault Activation problem ➔ a LJ Problem
- The Fault Propagation problem ➔
  1. Select a FP path to PO ➔ Decision
  2. Once the path is selected ➔ a set of LJ problems
- The LJ Problems ➔ Decisions or Implications

To justify $c = 1$ ➔ $a = 1$, $b = 1$ (Implication)

To justify $c = 0$ ➔ $a = 0$ or $b = 0$ (Decision)

- Incorrect decision ➔ Backtrack ➔ Another decision
D-Algorithm

Roth (IBM) - 1966

- Fundamental concepts invented:
  - First complete ATPG algorithm
  - D-Calculus (5 valued logic)
  - Implications – forward and backward
  - Implication stack
  - Backtrack
  - Test Search Space
Singular Cover Example

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

\[
\begin{array}{c}
A & \rightarrow & d \\
B & \rightarrow & e \\
C & \rightarrow & d \\
\end{array}
\]

\[
\begin{array}{c}
A \\
B \\
C \\
\end{array}
\]

\[
\begin{array}{c}
d \\
e \\
d \\
\end{array}
\]

\[
\begin{array}{c}
F \\
\end{array}
\]

<table>
<thead>
<tr>
<th>Gate</th>
<th>Inputs</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>2</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>3</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>1</td>
<td>1</td>
<td>1</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>
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>A</th>
<th>B</th>
<th>d</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>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>1</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>D</td>
</tr>
</tbody>
</table>
D-Cube Operation of D-Intersection

- $\psi$ – undefined (same as $\phi$)
- $\mu$ or $\lambda$ – requires inversion of $D$ and $\overline{D}$

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

- **D-containment** –
  Cube $a$ contains Cube $b$ if $b$ is a subset of $a$

\[
\begin{array}{c|cccccc}
\cap & 0 & 1 & X & D & \overline{D} \\
\hline
0 & 0 & \phi & 0 & \psi & \psi \\
1 & \phi & 1 & 1 & \psi & \psi \\
X & 0 & 1 & X & D & \overline{D} \\
\overline{D} & \psi & \psi & \overline{D} & \mu & \lambda \\
D & \psi & \psi & D & \lambda & \mu
\end{array}
\]
Primitive D-Cube of Failure

- Models circuit faults:
  - *Stuck-at-0*
  - *Stuck-at-1*
  - *Bridging fault* (short circuit)
  - Arbitrary change in logic function

- **AND Output sa0:** "1 1 D"
- **AND Output sa1:** "0 X D"
  "X 0 D"
- **Wire sa0:** "D"

- *Propagation D-cube* – models conditions under which fault effect propagates through gate
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
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 – \textit{D-drive}

while (untried fault effects on \textit{D-frontier})

select next untried \textit{D-frontier} gate for propagation;

while (untried fault effect fanouts exist)

select next untried fault effect fanout;

generate next untried propagation \textit{D-cube};

\textit{D-intersect} selected cube with test cube;

if (intersection fails or is undefined) continue;

if (all propagation \textit{D-cubes} tried & failed) break;

if (intersection succeeded)

add propagation \textit{D-cube} to test cube -- recreate \textit{D-frontier};

Find all forward & backward implications of assignment;

save \textit{D-frontier}, algorithm state, test cube, fanouts, fault;

break;

else if (intersection fails & \textit{D and D} in test cube) \textbf{Backtrack} ();

else if (intersection fails) break;

if (all fault effects unpropagatable) \textbf{Backtrack} ();
D-Algorithm - **Consistency**

\[ g = \text{coordinates of test cube with } 1\text{'s & } 0\text{'s;} \]
\[ \text{if (} g \text{ is only PIs) fault testable & stop;} \]
\[ \text{for (each unjustified signal in } g) \]
\[ \quad \text{Select highest } \# \text{ unjustified signal } z \text{ in } g, \text{ not a PI;} \]
\[ \quad \text{if (inputs to gate } z \text{ are both } D \text{ and } D) \text{ break;} \]
\[ \quad \text{while (untried singular covers of gate } z) \]
\[ \quad \quad \text{select next untried singular cover;} \]
\[ \quad \quad \text{if (no more singular covers)} \]
\[ \quad \quad \quad \text{If (no more stack choices) fault untestable & stop;} \]
\[ \quad \quad \quad \text{else if (untried alternatives in } Consistency) \]
\[ \quad \quad \quad \quad \text{pop implication stack -- try alternate assignment;} \]
\[ \quad \quad \quad \text{else} \]
\[ \quad \quad \quad \quad \text{"Backtrack ();} \]
\[ \quad \quad \quad \quad \text{D-drive ();} \]
\[ \quad \quad \quad \text{If (singular cover D-intersects with } z) \text{ delete } z \text{ from } g, \text{ add inputs to singular cover to } g, \text{ find all forward and backward implications of new assignment, and break;} \]
\[ \text{If (intersection fails) mark singular cover as failed;} \]
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;
Start

Initialize test cube (tc)

Select a primitive D-cube of fault as C

D-intersect C with previous test cube (tc) and perform implication

Is there a D or D' on any PO?

Consistent

Yes

Line Justification

Impossible

done

No

Test generated

Backtrack to the last point a choice existed

Select a gate from D-frontier and propagate D-cube of the Selected gate as C

None exists

No test exists

Inconsistent

None exists
D-Algo (Line Justification)

Begin

Is there any line in tc which are not justified?

Yes

Select an unjustified line and a singular cube C to justify the line

Interset C with previous test cube tc

Inconsistent

Backtrack to the last point a choice existed

None exists

Line Justification impossible

Consistent

No

Test has been generated
Circuit Example 1

A | B | C | F
---|---|---|---
0 | 0 | 0 | 0
0 | 0 | 1 | 0
0 | 1 | 0 | 0
0 | 1 | 1 | 1
1 | 0 | 0 | 0
1 | 0 | 1 | 0
1 | 1 | 0 | 0
1 | 1 | 1 | 0
### Singular Cover & D-Cubes

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>d</th>
<th>e</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>1</td>
<td>0</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
<tr>
<td>D</td>
<td>D</td>
<td>1</td>
<td>D</td>
<td>D</td>
<td>D</td>
<td>D</td>
</tr>
</tbody>
</table>

- **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>$A$</th>
<th>$B$</th>
<th>$C$</th>
<th>$d$</th>
<th>$e$</th>
<th>$F$</th>
<th>Cube type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>D</td>
<td></td>
<td></td>
<td>PDF of AND gate</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td>D</td>
<td>0</td>
<td>D</td>
<td>Prop. D-cube for NOR</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>
</tr>
</tbody>
</table>
9 - V Algorithm (Muth)

- Logic values \{0/0, 1/1, 0/1, 1/0, 0/u, 1/u, u/0, u/1, u/u\}
  - 0/u = \{0, D'\}, 1/u = \{D, 1\}, u/0 = \{0, D\}, u/1 = \{D', 1\}
  - u/u = \{0, 1, D, D'\}

- Reduces amount of search done in multiple path sensitization – D-Algo
9 - V Algorithm
9 - V Algorithm
## 9-V Algorithm: Value Comp

<table>
<thead>
<tr>
<th>Decision</th>
<th>Implication</th>
<th>Comments</th>
<th>l=u/1</th>
<th>j=u/1</th>
<th>Propagate via n</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>h=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>b=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>c=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g=D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=u/1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>k=u/1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m=u/1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d=1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>i=D</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d?0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>n=1/u</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>l=u/1</td>
<td></td>
<td></td>
<td>n=D</td>
<td>f?u/0</td>
<td></td>
</tr>
<tr>
<td>j=u/1</td>
<td></td>
<td></td>
<td>f=1</td>
<td>f?0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>e?u/0</td>
<td>e=1</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>e?0</td>
<td>k=D</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>m=D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
9-V Algorithm: Value Comp

No Backtracking
Path Oriented DEcision Making (PODEM)
P. Goel, IBM, 1981
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
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 (D’) to PO
- Backtracing
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
PODEM-Algorithm

Start

Assign binary value to an unsigned PI

Determine implications of all PIs

Is there a D or D’ on any PO?

No

Test Possible with additional Assigned PIs?

No

Is there an untried combination of Values on assigned PIs?

No

Set untried combination of values on assigned PIs

Test generated

Yes

No

May be

No test exists

Yes
PODEM

Ex: Objective = (F, 1).

The first time of backtracing

The second time of backtracing
D-Algorithm : Example
PODEM : Example

x-path (to PO) check fail => backtracking!!

Objective
## PODEM : Value Comp

<table>
<thead>
<tr>
<th>Objective</th>
<th>PI assignment</th>
<th>Implications</th>
<th>D-frontier</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=0</td>
<td>a=0</td>
<td>h=1</td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>b=1</td>
<td>b=1</td>
<td></td>
<td>g</td>
<td></td>
</tr>
<tr>
<td>c=1</td>
<td>c=1</td>
<td>g=D</td>
<td>i,k,m</td>
<td></td>
</tr>
<tr>
<td>d=1</td>
<td>d=1</td>
<td>d?0</td>
<td>k,m,n</td>
<td></td>
</tr>
<tr>
<td>k=1</td>
<td>e=0</td>
<td>e?1, j=0, k=1, n=1</td>
<td>m</td>
<td>x-path check fail !!</td>
</tr>
<tr>
<td>e=1</td>
<td>e?0, j=1, k=D</td>
<td></td>
<td>m,n</td>
<td>reversal</td>
</tr>
<tr>
<td>l=1</td>
<td>f=1</td>
<td>f?0, l=1, m=D, n=D</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
PODEM : Decision Tree
Thank You