Automatic Test Pattern Generation - III

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 – 13 (Jan 31, 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 \(\Rightarrow\) a LJ Problem
- The Fault Propagation problem \(\Rightarrow\)
  1. Select a FP path to PO \(\Rightarrow\) Decision
  2. Once the path is selected \(\Rightarrow\) a set of LJ problems
- The LJ Problems \(\Rightarrow\) Decisions or Implications

\[
\begin{align*}
\text{To justify } c &= 1 \Rightarrow a = 1, b = 1 \text{ (Implication)} \\
\text{To justify } c &= 0 \Rightarrow a = 0 \text{ or } b = 0 \text{ (Decision)}
\end{align*}
\]

- Incorrect decision \(\Rightarrow\) Backtrack \(\Rightarrow\) Another decision
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
PODEM

• 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

1. Start
2. Assign binary value to an unassigned PI
3. Determine implications of all PIs
4. Test generated: Yes
   - Is there a D or D’ on any PO?
     - If No: No, Test Possible with additional Assigned PIs?
       - If No: No test exists
       - If Yes: Set untried combination of values on assigned PIs
     - If Yes: Test Possible with additional Assigned PIs?
       - If No: Test Possible with additional Assigned PIs?
       - If Yes: Set untried combination of values on assigned PIs

May be
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>g</td>
<td></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</td>
<td>m</td>
<td>x-path check fail !!</td>
</tr>
<tr>
<td></td>
<td>e=1</td>
<td>e=0</td>
<td>m,n</td>
<td>reversal</td>
</tr>
<tr>
<td>l=1</td>
<td>f=1</td>
<td>f=0</td>
<td>m,n</td>
<td></td>
</tr>
</tbody>
</table>
PODEM : Decision Tree
PODEM doesn’t need

- Consistency check – conflict can never occur
- J-frontier – there are no values that require justification
- Backward implication – values are propagated only in forward directions
Example

- Select path $s - Y$ for fault propagation

(a,b) means that the line has $CC0 = a$ and $CC1 = b$
Initial objective: Set \( r \) to 1 to sensitize fault

\[ s_{a1} \]

Example -- Step 2

\((a,b)\) means that the line has \( CC0 = a \) and \( CC1 = b \)
Example -- Step 3 s sa1

- Backtrace from r

(a,b) means that the line has CC0 = a and CC1 = b
Example -- Step 4 s sa1

- Set $A = 0$ in implication stack

(a,b) means that the line has $CC0 = a$ and $CC1 = b$
Example -- Step 5 s sa1

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

(a,b) means that the line has CC0 = a and CC1 = b
Example -- Step 6 s sa1

- Initial objective: set $r$ to 1

(a,b) means that the line has $CC0 = a$ and $CC1 = b$
Example -- Step 7 s sa1

- Backtrace from r again

(a,b) means that the line has CC0 = a and CC1 = b
Example -- Step 8 s sa1

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

(a,b) means that the line has CC0 = a and CC1 = b
Example -- 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 \( s \rightarrow Y \) and \( s \rightarrow u \rightarrow v \rightarrow Z \) blocked (**D-frontier** disappeared)

\[(a,b)\) means that the line has \( CC0 = a \) and \( CC1 = b \]
Step 11 -- s sa1

- Set $B = 0$ (alternate assignment)

(a,b) means that the line has $CC0 = a$ and $CC1 = b$
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.

(a,b) means that the line has $CC0 = a$ and $CC1 = b$
Step 13 -- s sa1

- Set $A = 1$ (alternate assignment)

$(a,b)$ means that the line has $CC0 = a$ and $CC1 = b$
Step 14 -- s sa1

- Backtrace from r again
Step 15 -- s sa1

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

(a,b) means that the line has CC0 = a and CC1 = b
Backtrack -- s sa1

- Forward implications: $d = 0$, $X = 1$, $m = 1$, $r = 0$. **Conflict:** fault not sensitized. Backtrack

(a,b) means that the line has CC0 = a and CC1 = b
Step 17 -- s sa1

- Set \( B = 1 \) (alternate assignment)
Fault Tested - Step 18 s sa1

- Forward implications: \( d = 1, m = 1, r = 1, q = 0, s = D, v = D, X = 0, Y = D \)

\[(a,b)\] means that the line has \( CC0 = a \) and \( CC1 = b \)
Backtrace \((s, v_s)\) Pseudo-Code

\[ v = v_s; \]

while \((s\) is a gate output) \[
\text{if } (s\text{ is NAND or INVERTER or NOR}) v = \overline{v}; \\
\text{if (objective requires setting all inputs)} \\
\quad \text{select unassigned input } a \text{ of } s \text{ with hardest controllability to value } v; \\
\text{else} \\
\quad \text{select unassigned input } a \text{ of } s \text{ with easiest controllability to value } v; \\
\quad s = a; \\
\] return \((s, v) \) /* Gate and value to be assigned */;
Objective Selection Code

if (gate \( g \) is unassigned) return \((g, \overline{v})\);
select a gate \( P \) from the D-frontier;
select an unassigned input \( l \) of \( P \);
if (gate \( g \) has controlling value)
    \( c = \) controlling input value of \( g \);
else if (0 value easier to get at input of XOR/EQUIV gate)
    \( c = 1 \);
else \( c = 0 \);
return \((l, \overline{c})\);
PODEM Algorithm

while (no fault effect at POs)
    if (xpathcheck (D-frontier))
        \([l, v_l] = \text{Objective} (\text{fault}, v_{\text{fault}})\);
        \((p_i, v_{p_i}) = \text{Backtrace} (l, v_l)\);
        \(\text{Imply} (p_i, v_{p_i})\);
        if (PODEM (\text{fault}, v_{\text{fault}}) == SUCCESS) return (SUCCESS);
        \((p_i, v_{p_i}) = \text{Backtrack} ()\);
        \(\text{Imply} (p_i, v_{p_i})\);
        if (PODEM (\text{fault}, v_{\text{fault}}) == SUCCESS) return (SUCCESS);
        \(\text{Imply} (p_i, “X”)\);
        return (FAILURE);
    else if (implication stack exhausted)
        return (FAILURE);
    else \text{Backtrack} ();
return (SUCCESS);
FANout oriented test generation

FAN

(Fujiwara and Shimono, 1983)
TG Algorithms

Objective

- TG time reduction
  - Reduce number of backtracks
  - Find out the non-existence of solution as soon as possible
  - Branch and bound
FAN Algorithm

- **New concepts:**
  - Immediate assignment of *uniquely-determined signals*
  - *Unique sensitization*
  - Stop Backtrace at *head lines*
  - *Multiple Backtrace*
Thank You