Lecture 7:
Boolean Testing Using Fault Models

Instructor: M. Tahoori

This Lecture

- Specific Fault Objective — Target Fault
  - Boolean
    - Algebraic and Boolean Difference
  - Path Tracing
    - D Algorithm, PODEM, Fan
Fault Model-based Test Sets

- Good or Fault-Free Circuit
  - Circuit with No Faults Present
- Faulty Circuit
  - Circuit with Fault Present
- Detection vs Diagnosis

Specific-Fault Oriented Test Generation

- Two fundamental test generation steps
  - **ACTIVATE**, Excite, Provoke or Setup the Fault
    - Make Fault OBSERVABLE, Fault Sensitization
    - Find Primary Input Values that Cause
      - Error Signal in Faulty Circuit
    - For Single-Stuck-at-v Fault
      - Place v’ at Fault Site
  - **PROPAGATE** the Resulting Error to a Primary Output
    - Path Sensitization
    - Find Primary Input Values that Sensitize
      - Error Signal to Primary Output
Specific-Fault Oriented Test Generation

- Example: Test for c/0 is \( w,x,y = 0,1,1 \)
  - ACTIVATE Fault c/0
    - Set \( x = y = 1 \) to make \( c=1 \)
    - in Fault-free Circuit
  - PROPAGATE Value on c to f
    - Set \( w = 0 \) to sensitize c to f

Line Justification

- Find Input Assignment to Place Value v on Line g
- Algebraic Approach
  - Find Boolean Function Realized on line \( g = G(X) \)
  - Use Prime Implicant of \( G(X) \) to Place 1 on g
  - Use Prime Implicate of \( G(X) \) to Place 0 on g

- PROPAGATE Error (Fault Effect)
  - Algebraic Approach
    - Use Boolean Difference
Boolean Difference

- Shannon expansion
  - A Boolean function \( f(X_1, X_2, \ldots, X_n) \) can be expanded about any variable \( X_i \)
  - \( f(X_1, X_2, \ldots, X_n) = X_i f(X_1, \ldots, X_i = 0, \ldots X_n) + X_i f(X_1, \ldots, X_i = 1, \ldots X_n) \)
- Boolean Difference of \( f(X_1, X_2, \ldots, X_n) \) with respect to \( X_i \)
- Symbol is (partial derivation)
  \[
  \frac{df}{dX_i} = f(X_1, X_i, \ldots, X_n) \]
- Definition is:
  \[
  \frac{df}{dX_i} = f(X_1, \ldots, X_i = 0, \ldots X_n) \oplus f(X_1, \ldots, X_i = 1, \ldots X_n)
  \]
Boolean Difference

Example

\[ f = w + xy, \]

\[ f_y' = w \]

\[ f_y = w + x \]

\[ \frac{df}{dy} = (w) \oplus (w + x) = w'x \]

\[ df(x, y, w) \]

\[ \frac{df(x, y, w)}{dy} = 1 \]

for values of \( w \) and \( x \) for which \( f \) depends on \( y \)

\[ df(x, y, w) \]

\[ \frac{df(x, y, w)}{dy} = 0 \]

for values of \( w \) and \( x \) for which \( f \) is independent of \( y \)

\[ df(w + xy) \]

\[ \frac{df(w + xy)}{dy} = w'x \]

\[ w'x = 1, \text{ for } w=0, x=1 \]

When \( w = 0, x = 1, w + xy = y \)
**Boolean Difference**

- **Example**
  - $df/dy = fy' \oplus fy$
  - $= (w + x) \oplus z$
  - $= wz' + xz' + w'x'z$

- **Test pattern generation**
  - $df/dx = d(xy + yz)/dx = yz \oplus (y + yz) = yz'$
  - Test for a/0 is $xyz = (110)$
    - Set $x = 1$ to *Provoke* Fault
    - Set $y = 1, z = 0$ to *Sensitize* Fault Site to Output
Boolean Difference

Test pattern generation
- C/1
- \( \frac{df}{dc} = \frac{d(cx+yz)}{dc} = yz \oplus (x + yz) = x(y' + z) = x(y' + z') \)
- To Propagate Fault, Set \( x = 1, y \) or \( z = 0 \)
- \( c = v' + w' \)
- For C/1, must set \( c = 0 \), so \( v = w = 1 \)

Algebraic Technique to Determine
- Path Sensitization from Fault Site to Output, or
- Fault Observability Conditions
- Used Mainly for Theoretical Studies
Path Tracing

Test Generation Using Path Tracing

- **Notation**
  - $D$ Signal Value
    - 1 in Fault-free Circuit, 0 in Faulty Circuit
  - $D'$ or $\bar{D}$ Signal Value
    - 0 in Fault-free Circuit, 1 in Faulty Circuit
  - $X$
    - Signal Value is Unspecified
Notation

Truth Table for AND

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

Path Sensitization Method

- Fault Sensitization
  - Force tested node to opposite of fault value
- Fault Propagation (path sensitization)
  - Propagate the effect to one or more POs
- Line Justification
  - Justify internal signal assignments made to activate and sensitize fault
- These three steps may result in conflict
  - Different values are assigned to the same signal
  - Require backtracking
Path Sensitization Method

- Example (B stuck-at 0)
- Fault activation
  - Requires $B = 1$, $f = D$, $g = D$
- Fault propagation
  - Three scenarios are possible
    - paths $f - h - k - L$, $g - i - j - k - L$, or both

Try path $f - h - k - L$
- Requires $A = 1$, $j = 0$, $E = 1$
- Blocked at $j$
  - Since there is no way to justify 1 on $i$
**Path Sensitization Method**

- Try simultaneous paths $f \rightarrow h \rightarrow k \rightarrow L$ and $g \rightarrow i \rightarrow j \rightarrow k \rightarrow L$
- Blocked at $k$ because D-frontier (chain of $D$ or $\overline{D}$) disappears

![Diagram](image1)

**Path Sensitization Method**

- Final try: path $g \rightarrow i \rightarrow j \rightarrow k \rightarrow L$
- Test found!
Search Space Abstraction

- Binary Decision Tree (BDT)
  - The leaves represent the output of the good machine

(a) Circuit.  

(b) Binary decision tree.

Algorithm Completeness

- All ATPG programs implicitly search BDT
- Definition:
  - Algorithm is complete if it ultimately can search entire binary decision tree, as needed, to generate a test
- Untestable fault
  - No test for it even after entire tree searched
- Combinational circuits only
  - Untestable faults are redundant, showing the presence of unnecessary hardware
ATPG Problem

- Ibarra and Sahni in 1975 showed that ATPG is NP.Complete
  - No polynomial-time algorithm is known
  - Presumed to be exponential
- These ATPG algorithms employ heuristics that
  - Find all necessary signal assignments for a test
    - As early as possible
  - Search as little of the decision space as possible

Forward Implication

- Results in logic gate inputs that are significantly labeled so that output can be uniquely determined
- Example
  - AND gate forward implication table:
Backward Implication

- Unique determination of all gate inputs when the gate output and some of the inputs are given
- Backward implication is implemented procedurally
  - Since tables are cumbersome for gates with more than 2 inputs

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
- Example
  - PIs were set in order A, C, E, and B
  - B was set to 1 but failed
Implication Stack after Backtrack

<table>
<thead>
<tr>
<th>Signal</th>
<th>Value</th>
<th>Alternative tried</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td>1</td>
<td>NO</td>
</tr>
<tr>
<td>B</td>
<td>0</td>
<td>YES</td>
</tr>
<tr>
<td>F</td>
<td>0</td>
<td>YES</td>
</tr>
</tbody>
</table>

Objectives and Backtracing of ATPG

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

![Diagram](Image)
Branch-and-Bound Search

- An efficiently search method of binary search tree
- Branching
  - At each tree level, selects which input variable to set to what value (0 or 1)
- Bounding
  - Avoids exploring large tree portions by restricting search decision choices
  - Complete exploration is impractical
  - Decision about bounding made with limited information
    - Uses heuristics

Specific-Fault Oriented Test Generation

- Three Approaches
  - **D Algorithm**: Internal Line Values Assigned (Roth-1966)
    - D-cubes
    - Bridging faults
    - Logic gate function change faults
  - **PODEM**: Input Values Assigned (Goel – 1981)
    - X-Path-Check
    - Backtracing
  - **FAN**: Input and Internal Values Assigned (1983)