Overview
• Structural vs. functional test
• Definitions
• Completeness
• Conditions for finding a test
• Algebras
• Types of Algorithms – classical
• Complexity
• Summary
• Appendices

Combinational ATPG Basics

Functional vs. Structural ATPG
• Functional ATPG – generate complete set of tests for circuit input-output combinations
  – 129 inputs, 65 outputs:
  – $2^{129} = 680,564,733,841,876,926,926,749,214,863,536,422,912$ patterns
  – Using 1 GHz ATE, would take $2.15 \times 10^{22}$ years
• Structural test:
  – No redundant adder hardware, 64 bit slices
  – Each with 27 faults (using fault equivalence)
  – At most $64 \times 27 = 1728$ faults (tests)
  – Takes $0.000001728$ s on 1 GHz ATE
• Designer gives small set of functional tests – augment with structural tests to boost coverage to 98%
Algorithm Completeness

- Definition: Algorithm is complete if it ultimately can search entire binary (decision) space, as needed, to generate a test
- Unstable fault or Undetectable fault – no test for it even after entire space is searched
- Combinational circuits only – unstable faults are redundant, showing the presence of unnecessary hardware

Notation

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Good Circuit</th>
<th>Failing Circuit</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D$</td>
<td>0/1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0/0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$X$</td>
<td>X/X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>$G_0$</td>
<td>0/X</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>$G_1$</td>
<td>1/X</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>$F_0$</td>
<td>X/0</td>
<td>X</td>
<td>0</td>
</tr>
<tr>
<td>$F_1$</td>
<td>X/1</td>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

Roth’s and Muth’s Higher-Order Algebras

- Represent two machines, which are simulated simultaneously by a computer program:
  - Good circuit machine (1st value)
  - Bad circuit machine (2nd value)
- Better to represent both in the algebra:
  - Need only 1 pass of ATPG to solve both
  - Good machine values that preclude bad machine values become obvious sooner & vice versa
- Needed for complete ATPG:
  - Combinational: Multi-path sensitization, Roth Algebra
  - Sequential: Muth Algebra – good and bad machines may have different initial values due to fault

Conditions for Finding a Test

- Fault excitation – the signal value at the fault site must be different from the value of the stuck-at fault (thus fault site must contain a D or a D)
- The fault effect must be propagated to a primary output (A D or a D must appear at the output)
- Some simple observations
  - There must be at least a D or a D on some circuit nets
  - D’s must form a chain to some output

Random-Pattern Generation

- Flow chart for method
- Use to get tests for 60-80% of faults, then switch to D-algorithm or other ATPG for rest
**Boolean Difference Symbolic Method**

(Sellers et al.)

\[ g = G(X_1, X_2, \ldots, X_n) \]

for the fault site

\[ f_j = F_j(g, X_1, X_2, \ldots, X_n) \]

\[ 1 \leq j \leq m \]

\[ X_i = 0 \text{ or } 1 \text{ for } 1 \leq i \leq n \]

---

**Boolean Difference** (Sellers, Hsiao, Bearnson)

- Shannon's Expansion Theorem:
  \[ F(X_1, X_2, \ldots, X_n) = X_2 F(X_1, 1, \ldots, X_n) + \overline{X}_2 F(X_1, 0, \ldots, X_n) \]

- Boolean Difference (partial derivative):
  \[ \frac{\partial F}{\partial g} = F_j(1, X_1, X_2, \ldots, X_n) \oplus F_j(0, X_1, \ldots, X_n) \]

- Fault Detection Requirements for \( g \) stuck-at 0:
  \[ G(X_1, X_2, \ldots, X_n) = 1 \]

\[ \frac{\partial F}{\partial g} = F_j(1, X_1, X_2, \ldots, X_n) \oplus F_j(0, X_1, \ldots, X_n) = 1 \]

---

**Path Sensitization Method - Example**

1. Fault Sensitization
2. Fault Propagation
3. Line Justification

---

**Path Sensitization Method Circuit Example**

- Try path \( f - h - k - L \). This path is blocked at \( j \) since there is no way to justify the 1 on \( i \)

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

- Final try: path \( g - i - j - k - L \) - test found!
Computational Complexity

- Ibarra and Sahni analysis – NP-Complete (no polynomial expression found for compute time, presumed to be exponential)
- Worst case:
  - no_pi inputs, \(2^{\text{no_pi}}\) input combinations
  - no_ff flip-flops, \(4^{\text{no_ff}}\) initial flip-flop states
  - (good machine 0 or 1 \times \text{bad machine 0 or 1})
  - work to forward or reverse simulate \(n\) logic gates \(\propto n\)
- Complexity: \(O(n \times 2^{\text{no_pi}} \times 4^{\text{no_ff}})\)

Summary

- Basic definitions explained
- Developed notation and required algebra that will be used for test generation and fault simulation
- Basics of test generation developed
- Complexity of test generation addressed
- Appendix contains historical reference to the stuck-at fault model, an example of BDD, an instantiation of SAT problem.

Origins of Stuck-Faults

- Eldred (1959) – First use of structural testing for the Honeywell Datamatic 1000 computer
- Galey, Norby, Roth (1961) – First publication of stuck-at-0 and stuck-at-1 faults
- Seshu & Freeman (1962) – Use of stuck-faults for parallel fault simulation
- Poage (1963) – Theoretical analysis of stuck-at faults

Appendices

- Basics of test generation developed
- Complexity of test generation addressed
- Appendix contains historical reference to the stuck-at fault model, an example of BDD, an instantiation of SAT problem.

Circuit and Binary Decision Tree

- BDD – Follow path from source to sink node – product of literals along path gives Boolean value at sink
- Rightmost path: \(A B \bar{C} = 1\)
- Problem: Size varies greatly with variable order
History of Algorithm Speedups

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Est. speedup over D-ALG (normalized to D-ALG time)</th>
<th>Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>D-ALG</td>
<td>1</td>
<td>1966</td>
</tr>
<tr>
<td>PODEM</td>
<td>7</td>
<td>1981</td>
</tr>
<tr>
<td>PAN</td>
<td>23</td>
<td>1983</td>
</tr>
<tr>
<td>TOPS</td>
<td>292</td>
<td>1987</td>
</tr>
<tr>
<td>SOCRATES</td>
<td>1574† Upgraded to ATPG System</td>
<td>1988</td>
</tr>
<tr>
<td>Waicukauski et al.</td>
<td>2189† Upgraded to ATPG System</td>
<td>1990</td>
</tr>
<tr>
<td>EST</td>
<td>8765† Upgraded to ATPG System</td>
<td>1991</td>
</tr>
<tr>
<td>TRAN</td>
<td>3005† Upgraded to ATPG System</td>
<td>1993</td>
</tr>
<tr>
<td>Recursive learning</td>
<td>485</td>
<td>1995</td>
</tr>
<tr>
<td>Tafertshofer et al.</td>
<td>25057</td>
<td>1997</td>
</tr>
</tbody>
</table>

Analog Fault Modeling Impractical for Logic ATPG

- Huge # of different possible analog faults in digital circuit
- Exponential complexity of ATPG algorithm – a 20 flip-flop circuit can take days of computing
  - Cannot afford to go to a lower-level model
- Most test-pattern generators for digital circuits cannot even model at the transistor switch level (see textbook for 5 examples of switch-level ATPG)

Boolean Satisfiability

- 2SAT: \( x_1 \overline{x}_1 + x_2 \overline{x}_2 + x_3 x_3 + \ldots = 0 \)
- 3SAT: \( x_1 x_1 x_1 x_1 + x_2 x_2 x_2 x_2 + \ldots = 0 \)

Satisfiability Example for AND Gate

\[ \sum a_k b_k c_k = 0 \quad \text{ (non-tautology) or} \]
\[ \Pi (a_k + b_k + c_k) = 1 \quad \text{ (satisfiability)} \]

Pseudo-Boolean and Boolean False Functions

- **Pseudo-Boolean function**: use ordinary + -- integer arithmetic operators
  - Complementation of \( x \) represented by \( 1 - x \)
  - \( F_{\text{pseudo-Bool}} = 2 z + a b - a z - b z - a b z = 0 \)
- **Energy function representation**: let any variable be in the range \((0, 1)\) in pseudo-Boolean function
- **Boolean false expression**: \( f_{\text{AND}} (a, b, z) = z \oplus (ab) = a z + b z + a b z \)

AND Gate Implication Graph

- Really efficient
- Each variable has 2 nodes, one for each literal
- If ... then clause represented by edge from if literal to then literal
- Transform into **transitive closure graph**
  - When node true, all reachable states are true
- **ANDing operator** \( \land \) used for 3SAT relations