Lecture 9
Combinational Automatic Test-Pattern Generation (ATPG) Basics

- Algorithms and representations
- Structural vs. functional test
- Definitions
- Search spaces
- Completeness
- Algebras
- Types of Algorithms

Original slides copyright by Mike Bushnell and Vishwani Agrawal
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
Functional vs. Structural ATPG
Carry Circuit
Functional vs. Structural (Continued)

- 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+ %
Definition of Automatic Test-Pattern Generator

- Operations on digital hardware:
  - Inject fault into circuit modeled in computer
  - Use various ways to activate and propagate fault effect through hardware to circuit output
  - Output flips from expected to faulty signal

- Electron-beam (E-beam) test observes internal signals – “picture” of nodes charged to 0 and 1 in different colors
  - Too expensive

- Scan design – add test hardware to all flip-flops to make them a giant shift register in test mode
  - Can shift state in, scan state out
  - Widely used – makes sequential test combinational
  - Costs: 5 to 20% chip area, circuit delay, extra pin, longer test sequence
Circuit and Binary Decision Tree

(a) Circuit.

(b) Binary decision tree.
Binary Decision Diagram

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

- 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.
## Algebras: Roth’s 5-Valued and Muth’s 9-Valued

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Good Machine</th>
<th>Failing Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>1/0</td>
<td>1</td>
<td>0</td>
</tr>
<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>G0</td>
<td>0/X</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>G1</td>
<td>1/X</td>
<td>1</td>
<td>X</td>
</tr>
<tr>
<td>F0</td>
<td>X/0</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>F1</td>
<td>X/1</td>
<td>X</td>
<td>1</td>
</tr>
</tbody>
</table>

- Roth’s Algebra
- Muth’s Additions
Roth’s and Muth’s Higher-Order Algebras

- Represent two machines, which are simulated simultaneously by a computer program:
  - Good circuit machine (1\textsuperscript{st} value)
  - Bad circuit machine (2\textsuperscript{nd} 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
Exhaustive Algorithm

- For \( n \)-input circuit, generate all \( 2^n \) input patterns
- Infeasible, unless circuit is partitioned into cones of logic, with \( \leq 15 \) inputs
  - Perform exhaustive ATPG for each cone
  - Misses faults that require specific activation patterns for multiple cones to be tested
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

Flow chart:

1. Start
2. Set Input Probabilities
   - Initially $p(0) = 1/2$, $p(1) = 1/2$
3. Generate a Random Vector
4. Simulate Faults
5. Check Coverage
   - Inadequate
   - Adequate
5.1. Inadequate Coverage:
   - Change Probabilities
6. Adequate Coverage:
   - Stop

No new faults tested (discard vector)
Boolean Difference Symbolic Method (Sellers et al.)

\[ g = G(\ X_1, \ X_2, \ldots, \ X_n) \quad \text{for the fault site} \]

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

\[ 1 \leq j \leq m \]

\[ X_i = 0 \text{ or } 1 \quad \text{for } 1 \leq i \leq n \]
Shannon’s Expansion Theorem: 
\[ F (X_1, X_2, ..., X_n) = X_2 \cdot F (X_1, 1, ..., X_n) + X_2 \cdot F (X_1, 0, ..., X_n) \]

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

Fault Detection Requirements:
\[ G (X_1, X_2, ..., X_n) = 1 \]
\[ \frac{\partial F_j}{\partial g} = F_j (1, X_1, X_2, ..., X_n) \oplus F_j (0, X_1, ..., X_n) = 1 \]
Path Sensitization Method
Circuit Example

1 Fault Sensitization
2 Fault Propagation
3 Line Justification
Path Sensitization Method

Circuit Example

- Try path f – h – k – L blocked at j, since there is no way to justify the 1 on i
Path Sensitization Method

Circuit Example

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

- Final try: path \( g \rightarrow i \rightarrow j \rightarrow k \rightarrow L \) - test found!
**Boolean Satisfiability**

- **2SAT**: $x_i \cdot x_j + x_j \cdot x_k + x_1 \cdot x_m \cdots = 0$
  
  ...$x_p \cdot x_y + x_r \cdot x_s + x_t \cdot x_u \cdots = 0$

- **3SAT**: $x_i \cdot x_j \cdot x_k + x_j \cdot x_k \cdot x_l + x_1 \cdot x_m \cdot x_n \cdots = 0$
  
  ...$x_p \cdot x_y + x_r \cdot x_s \cdot x_t + x_t \cdot x_u \cdot x_v \cdots = 0$
Satisfiability Example for AND Gate

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

AND gate signal relationships:

- If \( a = 0 \), then \( z = 0 \)
- If \( b = 0 \), then \( z = 0 \)
- If \( z = 1 \), then \( a = 1 \) AND \( b = 1 \)
- If \( a = 1 \) AND \( b = 1 \), then \( z = 1 \)

Sum to get: \( a z + b z + a b z = 0 \)

(third relationship is redundant with 1\(^{st}\) two)
Pseudo-Boolean and Boolean False Functions

- Pseudo-Boolean function: use ordinary + -- integer arithmetic operators
  
  Complementation of $x$ represented by $1 - x$
  
  $$F_{\text{pseudo-Bool}} = 2z + ab - az - bz - abz = 0$$

- Energy function representation: let any variable be in the range $(0, 1)$ in pseudo-Boolean function

- Boolean false expression:
  
  $$f_{\text{AND}}(a, b, z) = z \oplus (ab) = \overline{a}z + \overline{b}z + abz$$
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 $\wedge$ used for 3SAT relations
Computational Complexity

- Ibarra and Sahni analysis – NP-Complete
  (no polynomial expression found for compute time, presumed to be exponential)

- Worst case:
  \( \text{no}_{\text{pi}} \) inputs, \( 2^{\text{no}_{\text{pi}}} \) input combinations
  \( \text{no}_{\text{ff}} \) flip-flops, \( 4^{\text{no}_{\text{ff}}} \) initial flip-flop states
  
  (good machine 0 or 1 \( \times \) bad machine 0 or 1)

  work to forward or reverse simulate \( n \) logic gates \( \propto n \)

- Complexity: \( O(n \times 2^{\text{no}_{\text{pi}}} \times 4^{\text{no}_{\text{ff}}}) \)
## 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>FAN</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 †</td>
<td>1988</td>
</tr>
<tr>
<td>Waicuakauski et al.</td>
<td>2189 †</td>
<td>1990</td>
</tr>
<tr>
<td>EST</td>
<td>8765 †</td>
<td>1991</td>
</tr>
<tr>
<td>TRAN</td>
<td>3005 †</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>

†: Indicates that the algorithm is part of an ATPG System.
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)