PODEM Algorithm

- 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

- PODEM: Path-Oriented DEcision Making (Goel 1981)
- Like D Algorithm, PODEM is circuit-based, fault-oriented ATPG algorithm
- Signal values are explicitly assigned at the primary inputs only; other values are computed by implication
- Justification is not needed
- Backtracking means reassigning primary inputs when a contradiction occurs; “implicit enumeration” technique
- A simple “backtrace” heuristic is used to select the next primary input line and the value to assign to it
- 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 Flow Chart

1. Assign binary value to unassigned PI
2. Determine implications of all assigned PIs
   - Test generated? If so, done.
   - Test possible with more assigned PIs? If yes, go to Step 1
   - Is there untried combination of values on assigned PIs? If not, exit: untestable fault
   - Set untried combination of values on assigned PIs using objectives and backtrace.
   - Then, go to Step 2

Find the Test vector that tests SA1 fault at G1.
D-Drive: G2 && {G3 || G5} && G7.
PODEM Example

<table>
<thead>
<tr>
<th>Iteration</th>
<th>Objective</th>
<th>Backtrace Implication</th>
<th>D-Front</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>G₂,₁ = 0</td>
<td>K = 1</td>
<td>(G₂)</td>
</tr>
<tr>
<td>2</td>
<td>G₂,₂ = 1</td>
<td>M = 1</td>
<td>G₂,₃ = D (G₂, G₃)</td>
</tr>
<tr>
<td>3</td>
<td>G₅,₁ = 1</td>
<td>N = 1</td>
<td>G₅,₃ = D' (G₅, G₇)</td>
</tr>
<tr>
<td>5a</td>
<td>G₅,₃ = 1</td>
<td>L = 0</td>
<td>G₅,₅ = D (G₅, G₇)</td>
</tr>
<tr>
<td>5b</td>
<td>G₇,₅ = 1</td>
<td>None</td>
<td></td>
</tr>
</tbody>
</table>

Test: (JKNL)=(1111)

How PODEM Works

1. Initial objective (I = 1)
2. Backtrace (b, 1)
3. PI assignment (b = 1)
4. Imply
5. Backtrace (c, 1)
6. PI assignment (a = 0)
7. Imply (d = 0)

Test: (JKNL)=(1111)

Comparison Between D-ALG & PODEM

D-Algorithm

<table>
<thead>
<tr>
<th>Decision</th>
<th>Implication</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>B₀ D₀</td>
<td>Activate fault</td>
</tr>
<tr>
<td>2</td>
<td>c₀</td>
<td>Backtrace, Imply</td>
</tr>
<tr>
<td>3</td>
<td>F₀ D₀</td>
<td>End of D-Drive</td>
</tr>
<tr>
<td>4</td>
<td>H₀ c₀</td>
<td>Justify F</td>
</tr>
<tr>
<td>5</td>
<td>d₀</td>
<td></td>
</tr>
</tbody>
</table>

PODEM

<table>
<thead>
<tr>
<th>Decision</th>
<th>Implication</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Initial objective (I, 0)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A₀, B₀ D₀</td>
<td>Backtrace, Imply</td>
</tr>
<tr>
<td>3</td>
<td>A₀ B₀ D₀</td>
<td>Backtrace, Imply</td>
</tr>
<tr>
<td>4</td>
<td>H₀, F₀ D₀</td>
<td>Objective (F,R), Imply; Success</td>
</tr>
</tbody>
</table>

Test = (a, b, c, e)=(0100)

Order of PI assignment: a, b, c, e.

Test = (a, b, c, e)=(0100)

Order of PI assignment: a, b, c, e.
D-ALG v.s. PODEM

- Emphasis on D-propagation
- Less backtracking
- Too much backtracking (undoing a previous decision and making a different choice) in some circuits. All possible choices may have to be tried in the worst case
- Some faults require multiple path sensitization

<table>
<thead>
<tr>
<th># of Cells</th>
<th>Run Time</th>
<th>Fault Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>D-ALG</td>
<td>PODEM</td>
</tr>
<tr>
<td>ch 1</td>
<td>828</td>
<td>34.5 100.0</td>
</tr>
<tr>
<td>ch 2</td>
<td>385</td>
<td>16.0 100.0</td>
</tr>
<tr>
<td>ch 3</td>
<td>351</td>
<td>2.2 90.5</td>
</tr>
<tr>
<td>ch 4</td>
<td>1,566</td>
<td>3.1 97.4</td>
</tr>
<tr>
<td>ch 5</td>
<td>1,042</td>
<td>3.2 96.6</td>
</tr>
</tbody>
</table>

Example – Fault s sa1

- Primitive D-cube of Failure

Example 7.3 – Backtrack

- Need alternate propagation D-cube for \( y \)

Example 7.3 – Step 2 s sa1

- Initial objective: Set \( r \) to 1 to sensitize fault
Example 7.3 -- Step 3 s a1

- Backtrace from r

FAN: Fan-out Oriented ATPG Algorithm

- FAN: Fan-out Oriented ATPG Algorithm (Fujiwara & Shimono, 1983)
- Representative of PODEM-based ATPG algorithms that add various heuristic speed-up features
- FAN drops or alters some of basic features of PODEM:
  - It halts backtracking at certain internal lines
  - It tries to satisfy multiple objectives at once (multiple backtrace)
  - It allows backward as well as forward implications
  - It makes quick and easy assignments directly

FAN -- Fujiwara and Shimono (1983)

- New concepts:
  - Immediate assignment of uniquely-determined signals
  - Unique sensitization
  - Stop backtrace at head lines
  - Multiple backtrace

PODEM Fails to Determine Unique Signals

- Backtracing operation fails to set all 3 inputs of gate L to 1
  - Causes unnecessary search
FAN -- Early Determination of Unique Signals

- Determine all unique signals implied by current decisions immediately
  - Avoids unnecessary search

PODEM Makes Unwise Signal Assignments

- Blocks fault propagation due to assignment $J = 0$

Unique Sensitization of FAN with No Search

- FAN immediately sets necessary signals to propagate fault

Headlines

- Headlines $H$ and $J$ separate circuit into 3 parts, for which test generation can be done independently
Contrasting Decision Trees

- **PODEM decision tree**
- **FAN decision tree**

### Multiple Backtrace

- **PODEM**:
  - Depth-first passes: 6 times
- **FAN**:
  - Breadth-first passes: 1 time

### Comparison Between PODEM & FAN

<table>
<thead>
<tr>
<th>Circuit Characteristic</th>
<th>Computing Time</th>
<th>Average # of Backtracks</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **ECC**: 716/1971, SPS 0.2, FAN 1.0, PODEM 31.2, FAN 4.8, FAN 1.2
- **ALU1**: 1003/205, SPS 4.5, FAN 1.0, PODEM 51.7, FAN 42.3, FAN 15.2
- **ALU2**: 1405/29, SPS 14.5, FAN 1.0, PODEM 160.2, FAN 61.8, FAN 0.6
- **ALU3**: 2052/70, SPS 3.1, FAN 1.0, PODEM 15, FAN 5.0, FAN 0.2
- **ALU4**: 2982/105, SPS 8.4, FAN 1.0, PODEM 30.1, FAN 53.0, FAN 23.2

### Random Test Generation

- **L** = length of random test
- **σ** = fault probability
- **Qn** = detection quality

- **HTDF**: Hard-to-detect fault
- **MDTDF**: Most difficult to detect fault
## 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>Waicukauski 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>