ECE-470 Digital Design II
Combinational Automatic Test-Pattern Generation (ATPG) Basics

Outline
• ATPG for SSFs in Combinational Circuits
  — Circuits with fanout-free structure
    • Activation/justification
    • Propagation
  — Circuits with fanout
    • Activation/justification
    • Propagation
    • Backtracking

ATPG Model
Fault universe
Tests
Diagnostic data
Test vectors
Vector 1
Vector N
Input patterns
Digital circuit
Output responses

Principle of testing

Notation with Composite Logic Values

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Meaning</th>
<th>Good Machine</th>
<th>Faulty Machine</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>1/0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Ð</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>0</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

Conditions for Finding a Test Vector
• Fault **activation (line-justification)** - 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 Ð)
• The fault effect must be **propagated** to a primary output (D or Ð a must appear at the output)
• Some simple observations
  – There must be at least a D or Ð on some circuit nets)
  – D’s must form a chain to some output
  – This technique describes **path sensitization** algorithm

Example: Justify and Propagate

\( D = 1/0 \)
Example: Justify and Propagate

Justify

```
begin
Set f to val
if f is a PI then return
// f is a gate output
  c = controlling value of f
  i = inversion of f
  inval = val XOR i
  if (inval == c̄)
    then for every input j of f
      Justify(j, inval) // recursive call
  else
    begin
      select one input (j) of f
      Justify (j, inval) // recursive call
    end
end
```

Propagate

```
• Error (D or D̄) propagation problem is transformed into
  a set of line-justification problems

Propagate (f, err)
// err is D or D̄
begin
Set f to err
if f is a PO then return
  k = the fanout of f
  c = controlling value of fanout k
  i = inversion of fanout k
  for every input j of k other than f
    Justify(j, c̄)
  Propagate (k, err XOR i) // recursive call
end
```

Circuits with Fanout (>1): Test Generation

• There are several ways/path of propagating an error
• Line-justification problems are no longer independent →
  conflicts may appear among implications → backtracking
  strategy needed

Computations Table

<table>
<thead>
<tr>
<th>Decisions</th>
<th>Implications</th>
<th>Purpose/ Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>a=1</td>
<td></td>
<td>Activate the fault</td>
</tr>
<tr>
<td>b=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>c=1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>f=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>h=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>g=1</td>
<td>u=0</td>
<td>Propagate through 5</td>
</tr>
<tr>
<td>d=0</td>
<td></td>
<td>Contradiction/conflict</td>
</tr>
<tr>
<td>a=0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Backtrack
Path Sensitization

- Decisions on which path to sensitize
- Backtracking:
  - Choose other path to propagate
  - Choose other gate-input to justify gate-output
- Simultaneous path sensitization may lead to no test found

Example

Try path $f \rightarrow h \rightarrow k \rightarrow L$ blocked at $j$, since there is no way to justify the 1 on $i$

Example

Try simultaneously 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

Example

Final try: path $g \rightarrow i \rightarrow j \rightarrow k \rightarrow L \rightarrow \text{test found!}$

Example

Backtracking: Means going back to the last decision situation and make a different decision

1: Decisions during $\text{Justify}(f, \text{val})$

```
Justify(f, val)
begin
Set f to val
if f is a PI then return
// f is a gate output
v = controlling value of f
i = inversion of f
inval = val XOR i
if (inval == $\overline{c}$)
    for every input j of f
        Justify(j, inval)   // recursive call
else
    begin
        select one input (j) of f
        Justify(j, inval)   // recursive call
    end
end
```

2: Decisions during $\text{Propagate}(f, \text{err})$

```
Propagate(f, err)
if err is D or $\overline{D}$ begin
Set f to err
if f is a PO then return
// f is a gate output
v = controlling value of f
i = inversion of f
for every input j of fanout k
    Justify(j, v XOR i)   // recursive call
end
```

Backtrack = recovery from incorrect decisions
General Outline of a Backtracking TG Algorithm to Generate Test for fs-a-v

```
Solve()
begin
  if imply_and_check() == FAILURE then return FAILURE
  if (error at PO and all lines are justified)
    then return SUCCESS
  if (no error can be propagated to a PO)
    then return FAILURE
  select an unsolved problem // line-justification or err-propagation
  repeat
    begin
      select one untried way to solve it
      if Solve() == SUCCESS then return SUCCESS
    end
  until all ways to solve it have been tried
  return FAILURE
end
```

Common Concepts: **Decision Tree**
- Facilitates implementation (and visualization) of the backtracking algorithm
- Decision node: problem (justification, propagation) that the algorithm is attempting to solve
- Branch: corresponds to a decision (i.e., trying one of the available alternative ways to solve a problem)

![Decision Tree Example](image)

Common Concepts: **D-frontier, J-frontier**
- TG algorithm is exhaustive: if there is a test for a fault, the algorithm will find it: implicit enumeration of all possible solutions
- Worst-case complexity is exponential
- D-frontier: all gates whose output value is currently x but have one or more error signals on their inputs
- J-frontier: all gates whose output values is known but not implied by its input values

Backtracking TG Algorithm
- Algorithm failure cases
  - D-frontier becomes empty: no possibility to propagate a fault-effect (error: D or D̄) through the circuit
  - A signal must be simultaneously set to both 0 and 1 in order to satisfy testing conditions for the test vector, but this is impossible. It is a conflict/contradiction
- 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

Common Concepts: **Implication Process**
- `Imply_and_check()`
  - Compute all values that can be uniquely determined by implication
  - Check for consistency and assign values
  - Maintain D-frontier and J-frontier
- Assignment-queue (similar to event-queue):
  - entry has form (f, v, direction)
  - v is the value to be assigned to signal-line f
  - direction can be forward or backward
  - Example: to generate a test for fault f sa1 the initial two entries in assignment-queue are: (f,0,backward) and (f,D,forward)
Backward Implications

Before

After

Forward Implications: Binary Values

Before

After

Forward Implications: Error Values

Before

After

ATPG Algorithm Types

- Exhaustive
  - for n-input circuit, generate all $2^n$ input patterns
- Random pattern generation
  - generate random test vectors until desired FC
- Boolean difference symbolic
  - not efficient for large circuits
- Path sensitization
  - D-algorithm, 9-V, PODEM, ...
- Other algorithms
  - Fault independent algorithms
  - Combined deterministic/random

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 ATPG System</td>
<td>1988</td>
</tr>
<tr>
<td>Waicukauski et al</td>
<td>2189 ATPG System</td>
<td>1990</td>
</tr>
<tr>
<td>EST</td>
<td>8765 ATPG System</td>
<td>1991</td>
</tr>
<tr>
<td>TRAN</td>
<td>3005 ATPG System</td>
<td>1993</td>
</tr>
<tr>
<td>Recursive learning</td>
<td>485 ATPG System</td>
<td>1995</td>
</tr>
<tr>
<td>Tufertshofer et al</td>
<td>25057 ATPG System</td>
<td>1997</td>
</tr>
</tbody>
</table>