Testability Driven Statistical Path Selection

Jaeyong Chung, Jinjun Xiong*,
Vladimir Zolotov*, and Jacob A. Abraham

Computer Engineering Research Center, University of Texas at Austin
*IBM T.J. Watson Research Center, Yorktown Heights
## Delay Test

<table>
<thead>
<tr>
<th></th>
<th>Silicon Debug</th>
<th>Manufacturing Test</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Objective</strong></td>
<td>Find speed-limiting Vectors</td>
<td>Check if DUT can run at a desired freq.</td>
</tr>
<tr>
<td><strong>Where it is done</strong></td>
<td>Microprocessor design</td>
<td>ASIC design/Micro. design</td>
</tr>
<tr>
<td><strong>Need to hit Fmax</strong></td>
<td>Necessary</td>
<td>Good correlation is ok</td>
</tr>
<tr>
<td><strong>Effects of interest</strong></td>
<td>Unmodelled effects, model/silicon mismatch (e.g., MIS, crosstalk, power droop)</td>
<td>Effects that differentiate each chips (e.g., process variation, delay defects)</td>
</tr>
<tr>
<td><strong>Important step</strong></td>
<td>Test vector generation</td>
<td>Path selection</td>
</tr>
<tr>
<td><strong>Test type</strong></td>
<td>Transition tests may work better than path tests</td>
<td>Path delay tests are appropriate</td>
</tr>
</tbody>
</table>
SSTA vs. STA in path selection

- SSTA can compute the fault probability for each path explicitly
  - Even if nominal delays are the same, the sensitivity to each source of variations can be different
- STA does not know correlations
  - May keep selecting paths in a chain of gates or in a small region
  - May select paths only sensitive to M1, etc
Path selection using SSTA

- Covering-based algorithms
  - The *process space coverage metric* (PCM) is used as the test quality metric
  - The objective is to maximize PCM when the number of paths to be selected is limited
  - The maximum coverage problem
  - Good paths for a given test clock period

- Criticality-based algorithms
  - Select paths with high criticality
  - Good paths irrespective of the test clock period
Problem Formulation

- Path criticality (e.g., the probability that the path becomes critical) takes the fault probability and the correlations into account.

Selection of paths with high criticality

Paths → ATPG → Test vectors → Scan delay test

Coverage loss due to untestable paths

[J. Chung et al ASPDAC 2011]
An example
An example

<table>
<thead>
<tr>
<th>SSTA (criticality)</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
</tbody>
</table>
An example

Untestable path

Slack

<table>
<thead>
<tr>
<th>SSTA (criticality)</th>
<th>STA</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A</td>
</tr>
<tr>
<td>C</td>
<td>B</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>B</td>
<td>D</td>
</tr>
</tbody>
</table>
Proposed Approach

- Consider the testability in the first place
  - Inspect only testable paths by integrating a SAT solver
- Define the *testable path coverage metric* (TCM)
  - The probability that a given path set contains the least slack testable path
- Our objective is to maximize TCM
TCM vs. other metrics

<table>
<thead>
<tr>
<th>Metric</th>
<th>Dep. Clock</th>
<th>Coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>PCM</td>
<td>Yes</td>
<td>Not 100%</td>
</tr>
<tr>
<td>Sum of criticality</td>
<td>No</td>
<td>Not 100%</td>
</tr>
<tr>
<td>TCM</td>
<td>No</td>
<td>100%</td>
</tr>
</tbody>
</table>

False path

Tested

Slack
Maximizing TCM

- Let $\lambda_{\Omega}$ be the probability that a given path is critical among all paths in the circuit (the conventional path criticality).

- Let $\lambda_{\Omega_t}$ be the probability that a given path is critical among all testable paths in the circuit (criticality among testable paths).

- Due to the mutually exclusive nature of the critical events,
  
  $$ TCM(\Pi) = \sum_{p \in \Pi} \lambda_{\Omega_t}(p). $$
Maximizing TCM

- For \( U \subseteq \Omega_t \), \( \lambda_{\Omega_t}(p) \leq \lambda_U(p) \)
- Our algorithm is based on the branch and bound framework proposed in [ICCAD 08, Vladimir et al]
  - Inspect paths in the depth-first manner
  - Maintain \( k \) best paths \( \Pi \)
  - Whenever a new path is found,
    - Replace it with one path in \( \Pi \)
    - Or just throw it away
  - Greedy Algorithm
An example

\[ \Pi \]

\[
\begin{array}{cccc}
A & B & C & D \\
\lambda_{\{A,B,C,D\}} & 0.1 & 0.4 & 0.3 & 0.2 \\
\lambda_{\Omega_i} & 0.08 & 0.3 & 0.1 & 0.1 \\
\end{array}
\]

new path

D    B    C
Maximizing TCM

- Criticality metric allows us the following modification
  - Whenever \( m \) new paths are accumulated,
    - Consider the replacement
    - Compute criticality among \( \prod \) and the \( m \) new paths
    - Discard the worst \( m \) paths
  - Lazy update so enhance the runtime
  - Less greedy manner (i.e., more global view) so improve the QoR
Pruning

- To make the path selection using depth-first search practical
  - If a subpath is not testable, all paths going through the subpath are not testable
  - If all paths going through a branch are worse than the worst path in $\prod$, we can prune the branch
Experimental Setup

- SSTA algorithm used: [Visweswariah DAC 2004]
- Spatial correlation model: A quad tree with 3 levels
  - 4%, 5%, and 6% variation at 1\textsuperscript{st}, 2\textsuperscript{nd}, and 3\textsuperscript{rd} level
  - 21 global sources of variation
- 5% random independent variation
- \textit{Minisat} is integrated
- Proposed method is compared with:
  - STA+ATPG
  - SSTA+ATPG
Experimental Results

eth_45 (82.4%)
Experimental Results

tv80_45 (0.03%)
## Experimental Results

<table>
<thead>
<tr>
<th>Circuit</th>
<th>SSTA+ATPG</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TCM</td>
<td>CPU(s)</td>
</tr>
<tr>
<td>C17</td>
<td>100%</td>
<td>0.000</td>
</tr>
<tr>
<td>C432</td>
<td>37%</td>
<td>0.082</td>
</tr>
<tr>
<td>C499</td>
<td>88%</td>
<td>0.030</td>
</tr>
<tr>
<td>C880</td>
<td>91%</td>
<td>0.010</td>
</tr>
<tr>
<td>C1355</td>
<td>86%</td>
<td>0.028</td>
</tr>
<tr>
<td>C1908</td>
<td>0%</td>
<td>464.9</td>
</tr>
<tr>
<td>C2670</td>
<td>87%</td>
<td>0.029</td>
</tr>
<tr>
<td>C3540</td>
<td>0%</td>
<td>0.747</td>
</tr>
<tr>
<td>C5315</td>
<td>100%</td>
<td>0.061</td>
</tr>
<tr>
<td>C6288</td>
<td>0.6%</td>
<td>18369.2</td>
</tr>
<tr>
<td>C7552</td>
<td>100%</td>
<td>0.507</td>
</tr>
<tr>
<td>Avg.</td>
<td>62%</td>
<td>1712.3</td>
</tr>
</tbody>
</table>
Conclusions

- Proposed method achieves 47% better QoR in TCM and up to 361X speedup compared to SSTA+ATPG
- Our method actually involves the test generation
- The remaining work is to load the test patterns on a tester
  - This allows us to demonstrate the benefits of statistical methodology on a silicon
    - Short turn-around time
    - Apples-to-apples comparison
Novel way of SAT Integration

1. Assuming $x_1 = 1$, check if satisfiable

2. Assuming $x_1 = 1$ and $x_2 = 1$, check if satisfiable

3. Assuming $x_1 = 1$, $x_2 = 1$ and $x_3 = 1$, check if satisfiable
   → the solver returns UNSAT

Conventional flow

Proposed flow

1. Assuming $x_1 = 1$, $x_2 = 1$, $x_3 = 1$, $x_4 = 1$ and $x_5 = 1$, check if satisfiable
   → the solver returns UNSAT with conflict assumptions which can be $x_1 = 1$, $x_3 = 1$