全晶片邏輯診斷法
Whole-Chip Diagnosis Strategy

清華大學電機系
黃錫瑜
參與研究生：王泓斌，林祐群，李彥鋒
2003/3/27
**Problem: Fault Diagnosis**

Circuit Under Diagnosis (CUD)

- **expected response**
- **not equal !**
- **faulty response**

**test patterns**

- a chip with defects inside

**Question**: Where are the fault locations?
Project Architecture

Tool environment
1. netlist parser
2. test pattern translator
3. test data log analyzer
4. link to layout database

- strategy for whole chip diagnosis
- delay fault diagnosis
- diagnosis for partial-scan designs

inject-and-evaluate diagnosis paradigm
(類針灸式的邏輯晶片診斷法)
Structural Pruning

Cone Union v.s. Cone Intersection

chip inputs

fault

z1

flip-flop outputs

z2

default

z3

flip-flop inputs

chip outputs
Structural Pruning

Cone-Union-Based Pruning

chip inputs

flip-flop outputs

Z1

Z2

Z3

chip outputs

flip-flop inputs

fault

fault

fault

Z1

Z2

Z3
Structurally Independent Faults

Faults F1 and F2 are independent faults

→ Divide-and-Conquer
Divide-And-Conquer (Good Case)

Two partitions: \{Z_1\} and \{Z_2, Z_3\}
Divide-And-Conquer (Bad Case)

One partition

chip inputs

flip-flop outputs

chip outputs

flip-flop inputs

fault

z1

z2

z3

z1

z2

z3

fault

z1

z2

z3
Outline

• Introduction
• Naive Method
• Our Strategy
• Experimental Results
• Summary
Main Strategy

• Phase 1: Isolate Independent Faults
  – Search for prime candidates
  – Use word-level information

• Phase 2: Locate Dependent Faults As Well
  – Perform partitioning
  – Aim at finding one fault in each block
A signal $f$ is a prime candidate if
(1) All failing input vectors are partially curable by $f$
(2) Curable-Output-Set($f$) is not covered by any other’s
Fake Prime Candidates

- Structurally Independent Faults
  - are often prime candidates
- Fake Prime Candidates
  - are prime candidates that are NOT really faults - aliasing

Ex: Dependent Double Faults F1 & F2
f1 f2 and f3 are fake prime candidates
Word-Level Registers and Outputs

Word-Level Output: \( O_1 \)
Word-Level Registers: \( R_1, R_2, \text{State} \)

```verilog
module design( O1, ...)
    output[31:0] O1;
    reg[31:0] R1, R2;
    reg[5:0] State
    ...
endmodule
```
Word-Level Prime Candidates

Ex: Consider candidate $f_1$

Reachable output \{ $O_1$ \}

Reachable register \{ $R_1$ \}

$\Rightarrow$ All syndromes at $O_1$ and $R_1$ cannot be cured simultaneously

$\Rightarrow$ $f_1$ is regarded as FAKE
Efficiency of Using Word-Level Info.

- Without word-level Information
  - contained 2.4 faults in 72.3 candidates
- With word-level Information
  - contained 1.23 faults out of 3.65 candidates

<table>
<thead>
<tr>
<th># of candidates</th>
<th>Original</th>
<th>After Filtering</th>
<th>Filtering Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prime Candidates</td>
<td>2.375</td>
<td>1.23</td>
<td>48.2 %</td>
</tr>
<tr>
<td>Fake Prime Candidates</td>
<td>69.96</td>
<td>2.42</td>
<td>96.5 %</td>
</tr>
</tbody>
</table>
Overall Flow

- failing input vectors
- design model
- faulty response

Phase 1:
(1) Find **Word-Level Prime Candidates**

Phase 2:
(1) Remove explained outputs and their fanin cones
(2) **Partition the rest model into groups**
(3) Perform **diagnosis** for each group

Rank candidates produced in phases 1 & 2
Grouping Using Dependency Graph

Prime candidates

x a
x b
x c
x c
x e
x y
x z
x f
x g
x h
x i
x j
x k

b
a
c
d
e
y
z
f
i
g
h
j
k
Remove Explained Faulty Outputs

prime candidates

syndromes at y and z are fully explained
Grouping Example
Outline

• Introduction
• Naive Method
• Our Strategy
• Experimental Results
• Summary
## Asymmetric Crypto Processor

### Curability Analysis

<table>
<thead>
<tr>
<th># of Faults</th>
<th>Inspect 5 signals</th>
<th>Inspect 10 signals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CA</td>
<td>OUR</td>
</tr>
<tr>
<td>2</td>
<td>1.13</td>
<td>1.54</td>
</tr>
<tr>
<td>3</td>
<td>1.15</td>
<td>2.17</td>
</tr>
<tr>
<td>4</td>
<td>1.26</td>
<td>2.72</td>
</tr>
<tr>
<td>5</td>
<td>1.27</td>
<td>2.98</td>
</tr>
<tr>
<td>Avg.</td>
<td>1.20</td>
<td>2.35</td>
</tr>
</tbody>
</table>

**CA**: Curability Analysis

**IMP**: Improvement (%)
# (III) ARM-Like Processor

**CA**: Curability Analysis  
**IMP**: Improvement (%)  

<table>
<thead>
<tr>
<th># of Faults</th>
<th>Inspect 5 signals</th>
<th>Inspect 10 signals</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CA</td>
<td>OUR</td>
</tr>
<tr>
<td>2</td>
<td>1.00</td>
<td>1.56</td>
</tr>
<tr>
<td>3</td>
<td>1.26</td>
<td>1.78</td>
</tr>
<tr>
<td>4</td>
<td>1.24</td>
<td>1.89</td>
</tr>
<tr>
<td>5</td>
<td>1.28</td>
<td>1.91</td>
</tr>
<tr>
<td>Avg.</td>
<td>1.20</td>
<td>1.79</td>
</tr>
</tbody>
</table>
Our New method is faster than CA mainly because of

(1) Structural pruning
(2) Removal of explained output and their fanins
Conclusion

• Strategy
  – (1) Search For Word-Level Prime Candidates
  – (2) Identify Independent Faults First
  – (3) Locate Dependent Faults As Well

• Results
  – identify 2.98 faults in 5 signal inspections
  – find 3.8 faults in 10 signal inspections