SYNTHESES OF MAPPING LOGIC FOR
GENERATING TRANSFORMED
PSEUDO-RANDOM PATTERNS FOR BIST

Nur A. Touba and Edward J. McCluskey

Center for Reliable Computing
Stanford University
ON-CHIP TEST PATTERN GENERATION

- On-Chip Test Pattern Generation
  - Requirement for BIST
  - Provide High Fault Coverage
  - Reasonable Test Length

- Linear Feedback Shift Register (LFSR)
  - Simple Structure $\Rightarrow$ Small Area
  - Dual Purpose: Test Pattern Generator Output Response Analyzer
  - May Not Give High Enough Fault Coverage
PROBLEM

Given: LFSR Doesn’t Provide High Enough Fault Coverage

- Choose Different Seed or Characteristic Polynomial
  - Less than Factor 10 Reduction in Test Length

- Improve Detection Probabilities
  - Insert Test Points or Redesign Circuit

- Augment LFSR with Additional Logic
  - Improve Patterns Generated
PREVIOUS APPROACHES

- Mixed-Mode
  - Use Deterministic Patterns to Detect Missed Faults
  - Deterministic Patterns On-Chip $\Rightarrow$ Large Overhead

- Multiple Seeds and/or Reconfigurable LFSR
  - Logic to Reseed or Change Polynomial of LFSR
  - Seeds/Polynomials Must Be Stored On-Chip

- Weighted Patterns
  - Bias Patterns to those that Detect Hard Faults
  - Multiple Weight Sets Often Required
TRANSFORMED PSEUDO-RANDOM PATTERNS

Transform Output of Pseudo-Random Pattern Generator
- Mapping Logic
- Combinational or Sequential
- Want to Minimize Overhead
MAPPING LOGIC

- Combinational Mapping Logic - Fixed Transformation
  - Single Weight Set
  - Combinational Units, Chatterjee & Pradhan, 1995.
  - LFSROM, Dufaza et al., 1995.

- Sequential Mapping Logic - Transformation Changes after some Num. of Patterns
  - Multiple Weight Sets
CHOOSING MAPPING FUNCTION

- Many Functions Satisfy Fault Coverage Requirement
- Amount of Logic to Implement Each Function Varies
- Problem: Choose Function that Minimizes Area

![Diagram of test pattern flow](image-url)
SPECIFYING MAPPING FUNCTION

GIVEN: Pattern Generating Circuit (LFSR Configuration)
Test Length and Fault Coverage Requirement

1) Simulate Pattern Generating Circuit for Test Length
   • Determine Original Pattern Set

2) Perform Fault Simulation
   • Keep Track of Patterns that Drop Faults

3) Do ATPG for Undetected Faults
   • Leave Unspecified Inputs as ‘X’ to Form Test Cubes
 SPECIFYING MAPPING FUNCTION

<table>
<thead>
<tr>
<th>Patterns That Drop Faults</th>
<th>Original Patterns $x_1x_2x_3x_4x_5x_6$</th>
<th>Transformed Patterns $x_1x_2x_3x_4x_5x_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>010011</td>
<td>010011</td>
<td>010011</td>
</tr>
<tr>
<td>011000</td>
<td>011000</td>
<td>011000</td>
</tr>
<tr>
<td>101101</td>
<td>101101</td>
<td>101101</td>
</tr>
<tr>
<td>010110</td>
<td>010110</td>
<td>010110</td>
</tr>
<tr>
<td>101101</td>
<td>101101</td>
<td>101101</td>
</tr>
<tr>
<td>Patterns Assigned to Test Cubes</td>
<td>???? ???? ???? ????</td>
<td>0010X1</td>
</tr>
<tr>
<td></td>
<td>???? ???? ???? ????</td>
<td>X01100</td>
</tr>
<tr>
<td></td>
<td>???? ???? ???? ????</td>
<td>1XX01X</td>
</tr>
<tr>
<td>Unassigned Patterns</td>
<td>100110</td>
<td>XXXXXX</td>
</tr>
<tr>
<td></td>
<td>001101</td>
<td>XXXXXX</td>
</tr>
<tr>
<td></td>
<td>010111</td>
<td>XXXXXX</td>
</tr>
<tr>
<td></td>
<td>110100</td>
<td>XXXXXX</td>
</tr>
<tr>
<td></td>
<td>100001</td>
<td>XXXXXX</td>
</tr>
<tr>
<td></td>
<td>111001</td>
<td>XXXXXX</td>
</tr>
<tr>
<td></td>
<td>001010</td>
<td>XXXXXX</td>
</tr>
</tbody>
</table>
**BIT-FIXING**

Example: 0010 mapped to 1000
- Fix First Bit to a 1
- Fix Third Bit to a 0

<table>
<thead>
<tr>
<th>Original Patterns</th>
<th>Test Cubes</th>
<th>B-Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \ x_2 \ x_3 \ x_4$</td>
<td>$x_1 \ x_2 \ x_3 \ x_4$</td>
<td>$x_1' \ x_2' \ x_3' \ x_4' \ x_1 \ x_2 \ x_3 \ x_4$</td>
</tr>
<tr>
<td>1 1 1 1</td>
<td>0 0 1 1</td>
<td>1* 1* 0 0 0 0 1 1</td>
</tr>
<tr>
<td>0 0 1 0</td>
<td>X 0 0 0</td>
<td>1 1 1* 1 1 0 0 0</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 X X 0</td>
<td>0 1 1 1* 1* 1 1 0</td>
</tr>
<tr>
<td>0 1 1 1</td>
<td>1 1 0 X</td>
<td>0 0 1* 1 1* 1 0 1</td>
</tr>
</tbody>
</table>

Source Function Active for
(0010,0101,0111)
MINIMIZING MAPPING FUNCTION

Assign Original Patterns to Test Cubes to Minimize Rectangle Cover of B-Matrix

<table>
<thead>
<tr>
<th>Original Patterns</th>
<th>Test Cubes</th>
<th>B-Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \ x_2 \ x_3 \ x_4$</td>
<td>$x_1' \ x_2' \ x_3' \ x_4'$</td>
<td>$x_1 \ x_2 \ x_3 \ x_4$</td>
</tr>
<tr>
<td>1 1 1 1 $\rightarrow$ 0 0 1 1</td>
<td>1* 1* 0 0 0 0 1 1</td>
<td></td>
</tr>
<tr>
<td>0 0 1 0 $\rightarrow$ X 0 0 0</td>
<td>1 1</td>
<td>1* 1 1 0 0 0</td>
</tr>
<tr>
<td>0 1 0 1 $\rightarrow$ 1 X X 0</td>
<td>0 1</td>
<td>1 1* 1* 1 1 0</td>
</tr>
<tr>
<td>1 0 1 1 $\rightarrow$ 1 1 0 X</td>
<td>0 0</td>
<td>1* 1 1 1* 0 1</td>
</tr>
</tbody>
</table>

Replace 1011 with 0111

<table>
<thead>
<tr>
<th>Original Patterns</th>
<th>Test Cubes</th>
<th>B-Matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>$x_1 \ x_2 \ x_3 \ x_4$</td>
<td>$x_1' \ x_2' \ x_3' \ x_4'$</td>
<td>$x_1 \ x_2 \ x_3 \ x_4$</td>
</tr>
<tr>
<td>1 1 1 1 $\rightarrow$ 0 0 1 1</td>
<td>1* 1* 0 0 0 0 1 1</td>
<td></td>
</tr>
<tr>
<td>0 0 1 0 $\rightarrow$ X 0 0 0</td>
<td>1 1</td>
<td>1* 1 1 0 0 0</td>
</tr>
<tr>
<td>0 1 0 1 $\rightarrow$ 1 X X 0</td>
<td>0 1</td>
<td>1 1* 1* 1 1 0</td>
</tr>
<tr>
<td>0 1 1 1 $\rightarrow$ 1 1 0 X</td>
<td>0 0</td>
<td>1* 1 1 1* 0 1</td>
</tr>
</tbody>
</table>
PROCEDURE FOR SELECTING MAPPING FUNCTION

INPUT: Test Cubes for Undetected Faults
       Original Patterns that Don’t Drop Faults

1) Form B-Matrix

2) Initial Assignment of Original Patterns
   • Based on Minimizing Number of Bit Differences

3) Let Each Stared Entry in B-Matrix be in R

4) EXPAND(R)

5) IRREDUNDANT(R)

6) Reassign Original Patterns to Eliminate Rectangles in R

7) REDUCE(R)

8) If Size of R Decreased, Loop Back to Step 4
SYNTHESESIZING BIT-FIXING LOGIC

- Determine Source Function for Each Rectangle in $R$
  - On-Set = Assigned Patterns to Transform
  - Off-Set = Patterns that Drop Faults
    + Other Assigned Patterns

- Use Logic Synthesis Tool to Generate Implementation
EXAMPLE OF IMPLEMENTATION

- Source Function 1: $x_1 \, x_2$  \hspace{1cm} Fix Bits: $x_1', \, x_2'$
- Source Function 2: $x_1' \,(x_2' + x_4)$  \hspace{1cm} Fix Bits: $x_1, \, x_3', \, x_4'$
COMPARING WITH PRIOR METHODS

Three Important Factors for Comparing Methods:
- Test Length
- Fault Coverage
- Hardware Area: Flip-Flops plus Gate Equivalents

Assume Parallel Application (“Test Per Clock”)

Gate Equivalents (GE’s)
- \((0.5)(n)\) GE’s for n-input NAND or NOR
- \((2.5)(n-1)\) GE’s for n-input XOR
- 1.5 GE’s for 2-to-1 MUX
MULTIPLE WEIGHT SETS

- Weight Sets from [Bershteyn, ITC 93] Used
- Assume No Extra Stages Added to Avoid Correlation
- Assume 4 two-input NAND/NOR to Generate Weighted Signals for Each Input
- One 2-to-1 MUX per Input for Each Weight Set

FF’s = \( \log_2(\text{Number of Weight Sets}) \)
GE’s = \([4 + (1.5)(\text{Number of Weight Sets})](\text{Number of Inputs})\)
## COMPARISON WITH WEIGHTED PATTERN TESTING

Weight Sets from [Bershteyn, ITC 93] Used

<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Random TLen</th>
<th>Multiple Weight Sets</th>
<th>Proposed Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>TLen</td>
<td>WS</td>
</tr>
<tr>
<td>s420</td>
<td>1.1M</td>
<td>532</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1.8K</td>
<td>593</td>
<td>3</td>
</tr>
<tr>
<td>s641</td>
<td>1.0M</td>
<td>893</td>
<td>5</td>
</tr>
<tr>
<td>s838</td>
<td>&gt;99M</td>
<td>12K</td>
<td>3</td>
</tr>
<tr>
<td>C2670</td>
<td>4.6M</td>
<td>2K</td>
<td>12</td>
</tr>
<tr>
<td>C7552</td>
<td>&gt;99M</td>
<td>69K</td>
<td>5</td>
</tr>
</tbody>
</table>
3-WEIGHT METHOD


While Random Patterns Applied
  • 3-Gate Modules Fix Certain Inputs
  • Forms “Expanded Tests”

Extra Flip-Flops Control Which Expanded Test Applied

Logic for 3-Gate Modules Depends on Fan-in

\[ FF's = \log_2(\text{Number of Expanded Tests}) \]
\[ GE's = (\text{Number of 3-Gate Modules})(1 + \text{Average Fan-In}) \]
## COMPARISON WITH 3-WEIGHT METHOD


<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Random TLen</th>
<th>3-Weight Method TLen</th>
<th>FF</th>
<th>GE</th>
<th>Proposed Method TLen</th>
<th>FF</th>
<th>GE</th>
</tr>
</thead>
<tbody>
<tr>
<td>C2670</td>
<td>4.6M</td>
<td>19K</td>
<td>5</td>
<td>1507</td>
<td>1K</td>
<td>0</td>
<td>218</td>
</tr>
<tr>
<td></td>
<td></td>
<td>30K</td>
<td>5</td>
<td>1316</td>
<td>7K</td>
<td>0</td>
<td>121</td>
</tr>
<tr>
<td>C7552</td>
<td>&gt;99M</td>
<td>47K</td>
<td>6</td>
<td>3003</td>
<td>10K</td>
<td>0</td>
<td>186</td>
</tr>
<tr>
<td></td>
<td></td>
<td>72K</td>
<td>6</td>
<td>2475</td>
<td>50K</td>
<td>0</td>
<td>139</td>
</tr>
</tbody>
</table>
FIXED-BIASED METHOD


- Use Weighted Bit Stream and Fix Value of Some Bits

- Configuration Sequences Periodically Loaded from ROM
  - Assume ROM is Off-Chip for Comparison Sake

- 17-Stage LFSR used to Generate Weighted Bit Stream

- Each Fixed Bit Requires 1 FF, 4 MUXes, and a NAND

  FF’s = 17 + Number of Fixed Bits
  GE’s = [(4)(1.5) + 1] (Number of Fixed Bits)
COMPARISON WITH FIXED-BIASED METHOD


<table>
<thead>
<tr>
<th>Circuit Name</th>
<th>Random TLen</th>
<th>Fixed-Biased TLen</th>
<th>FF</th>
<th>GE</th>
<th>Proposed Method TLen</th>
<th>FF</th>
<th>GE</th>
</tr>
</thead>
<tbody>
<tr>
<td>s420</td>
<td>1.1M</td>
<td>5K</td>
<td>18</td>
<td>&gt;7</td>
<td>500</td>
<td>0</td>
<td>37</td>
</tr>
<tr>
<td>s641</td>
<td>1.0M</td>
<td>19K</td>
<td>20</td>
<td>&gt;21</td>
<td>500</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>s838</td>
<td>&gt;99M</td>
<td>86K</td>
<td>19</td>
<td>&gt;14</td>
<td>850</td>
<td>0</td>
<td>86</td>
</tr>
<tr>
<td>C2670</td>
<td>4.6M</td>
<td>19K</td>
<td>54</td>
<td>&gt;259</td>
<td>1K</td>
<td>0</td>
<td>218</td>
</tr>
<tr>
<td>C7552</td>
<td>&gt;99M</td>
<td>191K</td>
<td>111</td>
<td>&gt;658</td>
<td>10K</td>
<td>0</td>
<td>186</td>
</tr>
</tbody>
</table>


CONCLUSIONS

- Select Mapping Function
  - Iterative Procedure Involving Global Operations
  - Less Overhead than Other Methods

- Advantages:
  - Easy to Insert into Existing Design
  - Fully Compatible with BILBO Registers
  - Easy to Trade Off: Test Time
    - Fault Coverage
    - Hardware Area
  - No Additional Sequential Logic Required
  - Very Simple Control (Only Test Mode Line)