On-chip Test Generation
Using Linear Subspaces

Ramashis Das, Igor Markov, John P. Hayes
University of Michigan
Ann Arbor, MI, USA
Outline

- Introduction
- Theoretical Framework
- Proposed Design
- Experimental Results
- Conclusions
Generic BIST Circuit

- Basic blocks:
  - Test Pattern Generator (TPG)
  - Test controller
  - Response Analyzer

- TPG: Feeds inputs to the circuit under test

- RA: Compares outputs with fault-free responses
Test Pattern Generator (TPG)

- Performance of BIST determined by
  - Efficiency of TPG in producing good test vectors

- Desired features:
  - Low hardware overhead
  - High fault coverage
  - Short testing time
Generic TPG Structure

- **Basic blocks:**
  - State Controller
  - ROM
  - Mapping Logic

- **SC:** Holds current state of TPG in SR

- **ML:** Decodes state into test inputs for CUT

- **M:** Stores predetermined test data

![Diagram of TPG Structure](image-url)
Some Existing TPG Designs

- Pre-stored tests
- Linear feedback shift registers (LFSRs)
- Linear transformation circuits [Akers, *ITC* 1989]
  - Obtain test set $T$ by running an ATPG program
  - Embed $T$ in $k$ n-bit vectors
  - SC: $k$-bit binary counter
  - ML: XOR array

![Diagram showing a binary counter, XOR array, and their connection to the circuit under test (CUT)]
Vector Spaces

- **Space**: Largest set that satisfies closure property on a field $F$
- For test vectors, vector space $V$ is defined over bit vectors and the field $F_2 = \{0, 1\}$
  - $\oplus$ – bit-wise XOR
  - $\bullet$ – bit-wise AND
- For $n = 3$, the vector space is $V_3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
Clusters (Subspaces)

- Cluster (subspace): subset of vector space
  - closed under bit-wise XORs

- Some clusters of $V_3$:
  - \{000\}
  - \{000, 001\}
  - \{000, 001, 010, 011\}

- Any cluster includes \{000...0\}
### Bases

- **Basis (of a cluster): set of vectors**
  - Must produce the entire cluster by bitwise XOR operations (linear combinations)
  - Smallest such set (not unique)

<table>
<thead>
<tr>
<th>Cluster $V'_3$</th>
<th>Basis $B'_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>${000, 001}$</td>
<td>${001}$</td>
</tr>
<tr>
<td>${000}$</td>
<td>${}$</td>
</tr>
<tr>
<td>${000, 001, 010, 011}$</td>
<td>${001, 010}$ or ${001, 011}$, or ${010, 011}$</td>
</tr>
</tbody>
</table>

$011 = 001 \oplus 010$
Example: Test set Compression
ISCAS-85 c499 benchmark circuit

Compression ratio of 6.6!
Time to test: $2^8 = 256$
Size Reduction

Entire Space
(2^n vectors)

Basis
(n vectors)
Size Reduction

- Basis of size $k$ captures a cluster of size $2^k$
- Compression ratio: $2^k/k$

Storage overhead: $k$
Testing time: $2^k$
Single vs. Multiple Clusters

Storage overhead: $k$
Testing time: $2^k$
**Single vs. Multiple Clusters**

Storage overhead: $k_1 + k_2 + k_3 (> k, \sim k)$

Testing time: $2^{k_1} + 2^{k_2} + 2^{k_3} (<< 2^k)$
Our Design Flow

Our Design Flow

Circuit netlist → ATALANTA

Subspace selection

Clusters

Basic TPG design

Design improvement

Final TPG
Basic Idea of our TPG Design

1. Controller
2. Basis
3. Hardware to generate cluster from basis vectors
4. State controller SC
5. State register SR
6. ROM M
7. Mapping logic ML
8. To circuit under test (CUT)

To circuit under test (CUT)
Generating a Cluster

- Enumerate all possible sums of basis vectors
  - Use a $k$-bit binary counter $b=0..2^{k-1}$
  - $j^{th}$ bit of count $b$ includes/excludes $v_j$ (basis vector)

\[
X_b = \sum_{j=0}^{k} b_j v_j, 0 \leq b \leq 2^{k-1}
\]

- Thus, $b$ determines a sum of basis vectors $X_b$
- All possible $X_b$ = all vectors in the subspace
  - Enumerated with a binary counter (or another counter!)
Example: Cluster Generation

Basis: \( \{ \nu_0 = 0001, \nu_1 = 0011, \nu_2 = 0101 \} \)
Size = 3 \( \rightarrow \) need a 3-bit counter (state \( b = b_2b_1b_0 \))

\( \nu_0 = 0001 \)
\( \nu_1 = 0011 \)
\( \nu_2 = 0101 \)

Bit-wise XOR of the selected vectors

\( b = 000 \)
\( b = 001 \)
\( b = 110 \)
\( b = 111 \)

\( X = 0000 \)
\( X = 0001 \)
\( X = 0110 \)
\( (0101 \oplus 0011) \)
\( X = 0111 \)
\( (0101 \oplus 0011 \oplus 0001) \)
Pros and Cons of Binary Enumeration

- **Pros**
  - Basis selection is flexible
  - Allows one to ensure full fault coverage

- **Cons**
  - Explicit XOR of all vectors → 2-D array of XORs
  - Large delays and area overhead due to XORs

- Similar to test embedding [Akers, *ITC’89*]
Use Gray Codes Instead!

- In Gray codes (000, 001, 011, 010, 110, ...)
  - Two consecutive counts differ by a single bit
- We replace a binary counter with a Gray-code counter in
  \[ X_g = \sum_{j=0}^{k} g_j v_j, 0 \leq g \leq 2^{k-1} \]
- \( X_{g+1} \) differs from \( X_g \) by basis vector \( v_j \), such that \( j=h \) is the flipped bit
  \[ X_{g+1} = X_g \oplus v_h \]

Fewer XOR gates required!
Example: Using Gray Codes

Basis: \{v_0 = 0001, v_1 = 0011, v_2 = 0101\}
Size = 3 → need a 3-bit counter (state \( g = g_2g_1g_0 \))

<table>
<thead>
<tr>
<th>Gray code ((g))</th>
<th>Flipping bit index ((h))</th>
<th>One-hot Gray code</th>
<th>(X_g)</th>
<th>(\oplus v_h)</th>
<th>(= X_{g+1})</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>-</td>
<td>-</td>
<td>0000</td>
<td>-</td>
<td>0000</td>
</tr>
<tr>
<td>001</td>
<td>0</td>
<td>001</td>
<td>0000</td>
<td>0001</td>
<td>0001</td>
</tr>
<tr>
<td>011</td>
<td>1</td>
<td>010</td>
<td>0001</td>
<td>0011</td>
<td>0010</td>
</tr>
<tr>
<td>010</td>
<td>0</td>
<td>001</td>
<td>0010</td>
<td>0001</td>
<td>0011</td>
</tr>
<tr>
<td>110</td>
<td>2</td>
<td>100</td>
<td>0011</td>
<td>0101</td>
<td>0110</td>
</tr>
<tr>
<td>111</td>
<td>0</td>
<td>001</td>
<td>0110</td>
<td>0001</td>
<td>0111</td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td>010</td>
<td>0111</td>
<td>0011</td>
<td>0100</td>
</tr>
<tr>
<td>100</td>
<td>0</td>
<td>001</td>
<td>0100</td>
<td>0001</td>
<td>0101</td>
</tr>
</tbody>
</table>
Cluster Generation in H/W

- One-hot Gray code counter
- ROM Containing Basis vectors
- Mapping logic

One-hot encoded address of the vector $v_h$

Circuit that computes bit-wise XOR of $v_h$ and previous test vector $X_g$
One-hot Gray-code counter

k-bit binary counter

ROM

Contains set of basis vectors

Mapping logic

To circuit under test (CUT)
Design Improvement

Multiple levels of logic merged & simplified.

One-hot Gray-code counter

k-bit binary counter

Layer of XOR gates

SC
k-bit binary counter
ML
Layer of XOR gates

To circuit under test (CUT)
Our Design Flow - Revisited

- Circuit netlist
- ATALANTA
- Subspace selection
- Fault simulator
- Clusters
- Basic TPG design
- Design improvement
- Final TPG
Subspace/Cluster Selection

- Find clusters & bases for a circuit
  - To achieve full fault coverage (by subspaces)
- Optimize performance metrics:
  - Area overhead: # basis vectors
  - Testing time: size of largest cluster

1. Start with \{000...0\} vector in cluster S
2. Find faults not detected by S (fault simulation)
3. Run ATPG to obtain test vectors T
4. Find \( t \in T \) that best increases fault coverage of S
5. Add \( t \) it to S
6. Unless full fault coverage reached, go to step 2
## Results: Testset Compression

<table>
<thead>
<tr>
<th>Benchmark circuits</th>
<th>No of inputs</th>
<th>ATALANATA</th>
<th>Proposed Method</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>No. of test patterns (n_1)</td>
<td>Max cluster size</td>
</tr>
<tr>
<td>c432</td>
<td>36</td>
<td>51</td>
<td>8</td>
</tr>
<tr>
<td>c499</td>
<td>41</td>
<td>53</td>
<td>8</td>
</tr>
<tr>
<td>c880</td>
<td>60</td>
<td>58</td>
<td>9</td>
</tr>
<tr>
<td>c1355</td>
<td>41</td>
<td>86</td>
<td>10</td>
</tr>
<tr>
<td>c1908</td>
<td>33</td>
<td>115</td>
<td>11</td>
</tr>
<tr>
<td>c2670</td>
<td>233</td>
<td>101</td>
<td>11</td>
</tr>
<tr>
<td>c3540</td>
<td>50</td>
<td>144</td>
<td>12</td>
</tr>
<tr>
<td>c5315</td>
<td>178</td>
<td>116</td>
<td>11</td>
</tr>
<tr>
<td>c6288</td>
<td>32</td>
<td>31</td>
<td>7</td>
</tr>
<tr>
<td>c7552</td>
<td>207</td>
<td>212</td>
<td>13</td>
</tr>
</tbody>
</table>
### Results: Testset Size

<table>
<thead>
<tr>
<th>Benchmark circuits</th>
<th>Weighted random pattern</th>
<th>Akers and Jansz</th>
<th>Multiple seeds/polyomials</th>
<th>Use of counters</th>
<th>GLFSR</th>
<th>Our TPG</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>636</td>
<td>1024</td>
<td>320</td>
<td>125</td>
<td>n/a</td>
<td>256</td>
</tr>
<tr>
<td>c499</td>
<td>1125</td>
<td>1024</td>
<td>679</td>
<td>22064</td>
<td>n/a</td>
<td>256</td>
</tr>
<tr>
<td>c880</td>
<td>765</td>
<td>8192</td>
<td>1596</td>
<td>29</td>
<td>640</td>
<td>527</td>
</tr>
<tr>
<td>c1355</td>
<td>3059</td>
<td>4096</td>
<td>1447</td>
<td>122344062</td>
<td>1760</td>
<td>1024</td>
</tr>
<tr>
<td>c1908</td>
<td>3539</td>
<td>8192</td>
<td>3659</td>
<td>1169</td>
<td>4700</td>
<td>2055</td>
</tr>
<tr>
<td>c2670</td>
<td>7689</td>
<td>65536</td>
<td>33000</td>
<td>n/a</td>
<td>6128</td>
<td>6269</td>
</tr>
<tr>
<td>c3540</td>
<td>3351</td>
<td>8192</td>
<td>6592</td>
<td>970</td>
<td>4828</td>
<td>4099</td>
</tr>
<tr>
<td>c5315</td>
<td>2279</td>
<td>8192</td>
<td>1843</td>
<td>62</td>
<td>n/a</td>
<td>2048</td>
</tr>
<tr>
<td>c6288</td>
<td>39</td>
<td>512</td>
<td>43</td>
<td>n/a</td>
<td>n/a</td>
<td>128</td>
</tr>
<tr>
<td>c7552</td>
<td>9276</td>
<td>n/a</td>
<td>32800</td>
<td>n/a</td>
<td>n/a</td>
<td>24577</td>
</tr>
</tbody>
</table>
## Results: Area Overhead

<table>
<thead>
<tr>
<th>Benchmark circuits</th>
<th>Akers and Jansz</th>
<th>GLFSR</th>
<th>CAPS</th>
<th>Our TPG</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>630</td>
<td>827</td>
<td>922</td>
<td>468</td>
</tr>
<tr>
<td>c499</td>
<td>784</td>
<td>926</td>
<td>1044</td>
<td>555</td>
</tr>
<tr>
<td>c880</td>
<td>1208</td>
<td>1365</td>
<td>1531</td>
<td>1099</td>
</tr>
<tr>
<td>c1355</td>
<td>738</td>
<td>926</td>
<td>1044</td>
<td>683</td>
</tr>
<tr>
<td>c1908</td>
<td>725</td>
<td>761</td>
<td>847</td>
<td>715</td>
</tr>
<tr>
<td>c2670</td>
<td>3668</td>
<td>5309</td>
<td>5951</td>
<td>9631</td>
</tr>
<tr>
<td>c3540</td>
<td>1027</td>
<td>1086</td>
<td>1255</td>
<td>1127</td>
</tr>
<tr>
<td>c5315</td>
<td>2708</td>
<td>3984</td>
<td>4517</td>
<td>3216</td>
</tr>
<tr>
<td>c6288</td>
<td>429</td>
<td>745</td>
<td>824</td>
<td>387</td>
</tr>
<tr>
<td>c7552</td>
<td>n/a</td>
<td>4617</td>
<td>5245</td>
<td>9834</td>
</tr>
</tbody>
</table>
Conclusions

- New on-chip test generation technique
  - Uses linear subspaces and bases
  - Uses Gray-code enumeration
  - Multi-level logic optimization (see paper)
  - End result: compact hardware design

- Heuristic for selecting subspaces & bases

- Salient features:
  - Achieves complete fault coverage
  - Relatively low hardware cost
  - Relatively small test set (testing time)
Thank You
Future Work

- Handling incompletely specified test sets (don’t cares)
- Overlap of Clusters
- Trade off between fault coverage and area overhead and/or testing time
- Improvement in heuristic used to select clusters
- Extension to scan-based testing