Self-Adjusting Constrained Random Stimulus Generation Using Splitting Evenness Evaluation and XOR Constraints

Shujun Deng, Zhiqiu Kong, Jinian Bian, Yanni Zhao

Department of Computer Science and Technology, Tsinghua University, Beijing, China

bianjn@tsinghua.edu.cn
Introduction

- Functional Verification: Bottleneck
- Formal Verification
  - Completeness ✗ Scalability
- Simulation
  - Completeness ✓ Scalability
  - Constrained random stimulus generation
    - An enhancement to traditional simulation
    - To find stimuli that would result in different responses for the good and the erroneous circuits
Introduction

- Constrained random stimulus generation
  - Weighted BDD sampling + random walks
    - SystemC Verification Library

- Scalability

- SAT-based method
  - Pre-assignment
    - Assign values to randomly selected variables
  - XOR constraints
    - Adding XOR constraints for randomly selected variables
Introduction

- Our solution

  - Dynamic: self-adjusting
  - Irrelative with the detailed design: no coverage feedback
Background

- Definitions
  - Problem: Distribution of $K$ solutions selected from $N$-sized space
  - Least Even Distribution (LED)
    - Solutions are the same
  - Most Even Distribution (MED)
    - Solutions distributed evenly

![Diagram showing Most Even Distribution (LED) and Not Most Even Distribution](image-url)
Background

- XOR Constraints
  - A SAT problem with \( N > 1 \) solutions can be reduced to only one solution (U-SAT) through randomly adding some XOR constraints with success probability \( p \geq \frac{1}{4} \)
  - By adding a random XOR constraint into a SAT problem, there is a high probability the solution space can be reduced into half.
## Background

- **XOR Constraints**
  - **Original CNF**
    - \((a + b)(b + \neg c + d)(\neg a + c + \neg d)(c + d)\)
    - Initial solution space \{0101, 0110, 0111, 1011, 1110, 1111\}
  - The results for added XOR constraints

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(a \oplus b)</td>
<td>{0101, 0110, 0111, 1011}</td>
<td>4 : 2</td>
</tr>
<tr>
<td>(a \oplus c)</td>
<td>{0110, 0111}</td>
<td>2 : 4</td>
</tr>
<tr>
<td>(a \oplus d)</td>
<td>{0101, 0111, 1110}</td>
<td>3 : 3</td>
</tr>
<tr>
<td>(b \oplus c)</td>
<td>{0101, 1011}</td>
<td>2 : 4</td>
</tr>
<tr>
<td>(b \oplus d)</td>
<td>{0110, 1011, 1110}</td>
<td>3 : 3</td>
</tr>
<tr>
<td>(c \oplus d)</td>
<td>{0101, 0110, 1110}</td>
<td>3 : 3</td>
</tr>
<tr>
<td>(a \oplus b \oplus c)</td>
<td>{0101, 1110, 1111}</td>
<td>3 : 3</td>
</tr>
<tr>
<td>(a \oplus b \oplus d)</td>
<td>{0110, 1111}</td>
<td>2 : 4</td>
</tr>
<tr>
<td>(b \oplus c \oplus d)</td>
<td>{0111, 1111}</td>
<td>2 : 4</td>
</tr>
<tr>
<td>(a \oplus b \oplus c \oplus d)</td>
<td>{0111, 1011, 1110}</td>
<td>3 : 3</td>
</tr>
</tbody>
</table>
Even Distribution Evaluation
How to Evaluate the Evenness?

- Evaluating methods
  - Method based on Discrete Fourier Transfer (DFT)
  - Our methods
    - Weighted Min-Distance-Sum
    - Simplified Min-Distance-Sum
Background

**MED in terms of Discrete Fourier Transfer (DFT)**

\[
\sum_{m=1}^{N-1} \frac{|F(m)|^2 (m - \frac{N}{2})^2}{N-1} = \sum_{m=1}^{N-1} \left[ \sum_{i=0}^{K-1} e^{-j2\pi m(S_i+1)/N} \right]^2 (m - \frac{N}{2})^2
\]

\[
= \sum_{m=1}^{N-1} \left[ \sum_{i=0}^{K-1} \left( \cos \left( \frac{2\pi m(S_i+1)}{N} \right) - j \sin \left( \frac{2\pi m(S_i+1)}{N} \right) \right) \right]^2 (m - \frac{N}{2})^2
\]

Even Distribution Evaluation

- All the possible solutions are located on a circle
- Problem transfer:
  - Evenness →
    - Distance of each adjoining solutions
    - Weighted minimum distance sum
Our Evaluation Methods

- **Weighted Min-Distance-Sum**

\[
\sum_{u=1}^{K-1} \left( \left( \sum_{i=0}^{K-1} \left| \frac{N}{K} - \frac{\Delta_{i,u}}{u} \right| \right)^2 (u - \frac{N}{2})^2 \right) \frac{N^2}{K^2} \sum_{u=1}^{K-1} ((K-u)^2 (2u-N)^2)
\]

\[
\Delta_{i,u} = \begin{cases} 
S_i - S_{i-u} & \text{if } i \geq u \\
S_i + N - S_{i+K-u} & \text{otherwise}
\end{cases}
\]
Our Evaluation Methods (Cont’)

- **Simplified Min-Distance-Sum**

\[
D = \frac{\sum_{i=0}^{k-1} \frac{2^n}{k} - \Delta_i}{k - 1} 2^{n+1} / k
\]

\[
\Delta_i = \begin{cases} 
S_i - S_{i-1}, & i = 1, 2, ..., k - 1 \\
S_0 + 2^n - S_{k-1}, & i = 0 
\end{cases}
\]

- \(D\) ranges from 0 to 1
- Less \(D\) means more even distributed
Difference

- **Weighted Min-Distance-Sum**
  - 1-step to \((K-1)\)-step distances
  - *Complexity: \(O(K^2)\)*
  - Can distinguish \(\{5,5,6,5,5,6\}\) from \(\{5,5,5,5,6,6\}\)

- **Simplified Min-Distance-Sum**
  - Only 1-step distance
  - *Complexity: \(O(K)\)*
  - Can not distinguish \(\{5,5,6,5,5,6\}\) from \(\{5,5,5,5,6,6\}\)

- That is all right for random stimulus generation – no need to find the optimal stimulus set

---

(a) (b)
Experiments for Evaluation

- Simplified Min-Distance-Sum (Simp-MDS)
  - Time: (1) 0.003s (2) 0.003s

- Weighted Min-Distance-Sum (MDS)
  - Time: (1) 0.290s (2) 0.300s

- MED in terms of Discrete Fourier Transfer (DFT)
  - Time: (1) 3.585s (2) 4.598s

- The trends are similar for different test-cases

**Conclusion:** Simp-MDS is more efficient than other methods, and it is adequate for random stimulus generation
Self-Adjust Framework based on Simplified Min-Distance-Sum
Limitation of Evenness Evaluation

- This stimulus’s evenness score is good, but it is not good for practical simulation
- Splitting strategy is needed

\[
K = \begin{cases} 
000000000000 \\
010000000000 \\
100000000000 \\
110000000000 
\end{cases}
\]
Splitting Strategy

- $K$ is the expected number of test vectors
- Split each test vector into groups with $\log_2 K$ width

\[
K = \begin{cases} 
01...10 & 01...11 \ldots \text{00...01...} \\
11...11 & 00...10 \ldots \text{10...11...} \\
\ldots & \ldots \\
00...01 & 11...10 \ldots \text{01...10...} \\
\end{cases}
\]

\[
\begin{array}{ccc}
\log_2 K & \log_2 K & \log_2 K \\
\end{array}
\]

$n$
Main Framework

Algorithm:
1. \( \text{cur_sti} \leftarrow 0; \) /* current stimuli */
2. \( \text{inc_sti} \leftarrow \left\lfloor \frac{K}{t} \right\rfloor; \) /* stimuli generated each time */
3. while ( \( \text{cur_sti} < K \) ) {
4.   if ( \( \text{inc_sti} \leq \left\lfloor K \right\rfloor \) ) { /* the last time */
5.     /* generate all the left stimuli */
6.     \( \text{inc_sti} = K - \text{cur_sti}; \)
7.   }
8.   if ( \( \text{inc_sti} \neq \left\lfloor \frac{K}{t} \right\rfloor \) ) { /* not the first time */
9.     \( \text{split_evaluate}(); \)
10. }
11. else {
12.     /* generate inc_sti stimuli */
13.     for ( \( i = 0; i < \text{inc_sti}; i++ \) ) {
14.       Add random XOR constraints for \( \text{Ins} \);
15.       Generate one stimulus using Minisat;
16.     }
17.   }
18. \( \text{cur_sti} += \text{inc_sti}; \)
19. /* the number of stimuli generated next time*/
20. \( \text{inc_sti} = \left\lfloor \text{inc_sti} \times \frac{t-1}{t} \right\rfloor; \)
21. }
22. return \( \text{stimuli} \);
Split-Evaluation Function

Function:
1. Function split_evaluate() {
2. /* split the current stimuli into groups, each with
   \([\log_2 \text{cur}\_\text{sti}]\) size. */
3. split();
4. /* evaluate each group using the formula (5)
   independently, recorded into the array a_evalu. */
5. a_evalu = simp_mds();
6. sort(a_evalu);
7. select_the_worst_group();
8. /* generate inc_sti stimuli */
9. for ( i = 0; i < inc_sti; i ++ ) {
10.   Add random solution for the worst group;
11.   Add random XOR constraints for other Ins;
12.   Generate one stimulus using Minisat;
13. }
14. }

EDA Lab, Dept. of Computer Science & Technology, Tsinghua Univ.
Experimental Results

- **Benchmark:** s27 in ISCAS89 expanded for 50 time-frames
- **RAN:** direct random stimulus generation
- **XOR:** stimulus generation with random XOR constraints
- **SELF-ADJ:** the self-adjusting method

![Graph showing fault coverage over number of test vectors for RAN, XOR, and SELF-ADJ methods.](image-url)
## More Results

<table>
<thead>
<tr>
<th>Test-case</th>
<th>#Faults</th>
<th>$K$</th>
<th>RAN</th>
<th>XOR</th>
<th>ROW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>#RAN.C</td>
<td>RAN.T</td>
<td>#XOR.C</td>
</tr>
<tr>
<td>s298_5</td>
<td>1428</td>
<td>32</td>
<td>45.77</td>
<td>56.03%</td>
<td>16.40</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>155.45</td>
<td>67.51%</td>
<td>44.01</td>
</tr>
<tr>
<td>s382_5</td>
<td>1827</td>
<td>32</td>
<td>55.66</td>
<td>33.59%</td>
<td>12.51</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>121.13</td>
<td>43.59%</td>
<td>32.47</td>
</tr>
<tr>
<td>s386_5</td>
<td>1872</td>
<td>32</td>
<td>41.99</td>
<td>43.00%</td>
<td>17.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>93.31</td>
<td>49.95%</td>
<td>31.87</td>
</tr>
<tr>
<td>s1196_2</td>
<td>2439</td>
<td>32</td>
<td>19.77</td>
<td>45.80%</td>
<td>8.99</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>9.23</td>
<td>55.66%</td>
<td>17.91</td>
</tr>
<tr>
<td>s1238_2</td>
<td>2665</td>
<td>32</td>
<td>17.36</td>
<td>45.04%</td>
<td>9.10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>8.89</td>
<td>54.93%</td>
<td>17.21</td>
</tr>
<tr>
<td>s1488_2</td>
<td>2960</td>
<td>32</td>
<td>229.30</td>
<td>39.06%</td>
<td>240.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>595.87</td>
<td>45.01%</td>
<td>555.77</td>
</tr>
<tr>
<td>s1494_2</td>
<td>3000</td>
<td>32</td>
<td>280.73</td>
<td>39.55%</td>
<td>232.02</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>629.55</td>
<td>42.13%</td>
<td>554.16</td>
</tr>
<tr>
<td>s13207_2</td>
<td>18292</td>
<td>32</td>
<td>1704.01</td>
<td>53.45%</td>
<td>1527.48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>2937.05</td>
<td>59.42%</td>
<td>3428.06</td>
</tr>
<tr>
<td>s15850_2</td>
<td>22256</td>
<td>32</td>
<td>2271.92</td>
<td>49.52%</td>
<td>2094.03</td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>5559.34</td>
<td>56.39%</td>
<td>4912.75</td>
</tr>
<tr>
<td>Average</td>
<td>6304.3</td>
<td>48</td>
<td>820.91</td>
<td>48.87%</td>
<td>764.02</td>
</tr>
</tbody>
</table>
Analysis

- Average coverage and run time comparison
  - SELF-ADJ uses the same time to find the same number of test vectors with 37% higher coverage ratio than RAN
Conclusions and Future works

- Simplified Min-Distance-Sum is efficient and adequate for applications in constrained random stimulus generation.
- When the test-case is difficult, the evaluation time can be ignored (Solving time >> Evaluation time).
- Self-adjusting method can improve the fault coverage ratio considerably.
- Future works:
  - Algorithm optimization
  - Apply to high level functional verification
Thank You!