Crosstalk Glitch Fault ATPG with Test Compaction

Shehzad Hasan, Ajoy K. Palit, Walter Anheier
{hasan, palit, anheier}@item.uni-bremen.de
ITEM, University of Bremen, Otto-Hahn-Allee 1, 28359 Bremen, Germany

Abstract
This work proposes a TPG method for producing maximal crosstalk glitch effect on victim nets. Thereafter, the test set is compacted using different implementations of fault-list chaining algorithm.

1. Introduction
A crosstalk induced glitch effect is produced in a coupled line when aggressors (or affecting lines) switch from one logic state to another (e.g. logic level 0 to 1) and victim (or affected line) remains static (e.g. logic level 0). The magnitude of glitch depends on several factors like coupling influence of aggressors, the alignment of aggressors’ transitions, the direction of the transitions, etc. The two-vector set approach is used for testing of crosstalk glitch faults, the patterns are applied one after the another to produce the required aggressor transition [1, 2]. However, for maximal crosstalk glitch effect one must also consider the spatial, temporal and functional properties of the circuits while determining the test vectors. Testing of both positive and negative glitch faults on each interconnect results in large number of test vectors, which need to be reduced in order to meet the test storage and test application time requirements. This can be achieved through pruning as many as possible crosstalk insensitive sites as well as through application of different test compaction techniques after the test set is generated. In [3] the fault-list chaining algorithm is proposed to reduce the number of patterns after the application of static compaction through merging. In this work different implementations of this algorithm are compared with the previous approach of [3].

2. Crosstalk Glitch ATPG
The test generation for crosstalk glitch faults consists of two phases, namely before transition (BT) and after transition (AT), where transition refers to the change of logic state of aggressor. In order to test for positive glitch faults induced on the victim net, victim has to be set to logic-0 state in both the phases but with aggressors changing states from 0 to 1. This can be achieved by assuming a stuck-at-1 fault on the victim interconnect and for aggressors a stuck-at-1 fault in before transition phase and stuck-at-0 fault in after transition phase. Combinational ATPG programs can then be used for generating such test patterns. A modification is required here to remove the forward propagation phase while testing aggressors, as only victim’s noise effect is required to be propagated to the output. This is proposed by [1, 2] but none of them considered the timing aspects of the circuit. Timing analysis is necessary for removing false crosstalk noise effect. Additionally for multiple aggressors both the timing and direction of aggressor transitions is also important. Fig. 1 shows that for obtaining maximum crosstalk glitch effect on victim line the aggressor signals must be aligned i.e. simultaneous transition and in the same direction.

In our approach a FAN based ATPG is used and modified by removing the forward propagation step while determining stuck at faults on aggressor wires. Each interconnect is assigned a particular weight with respect to the victim wire considering capacitive coupling effects between the wires. Since layout information was not available we used random weights. Spatial pruning is performed by removing aggressors having weight less than a specified threshold, which is because either the aggressor is too far away from the victim net or its effect is shielded by some other interconnect. Temporal pruning is done by grouping aggressors into time compatible sets. An aggressor is time compatible with another if their switching windows overlap. Switching window can be defined as the time duration within which some active transitions can propagate through the wire [3]. Switching windows are calculated by performing static timing analysis of the circuit assigning random delays to interconnects and gates. Functional pruning is performed by removing static nets from the time compatible aggressor sets. This is done by applying test vector combinations of victim net and by observing the states of aggressor nets for a change of logic value. For this purpose 10 test vector combinations of victim net were used (selected randomly).

Thereafter, the maximum weighted set is calculated. Each aggressor within the set is selected as a main aggressor. An aggressor from the remaining aggressors of the set assumes a positive weight if it can be excited to the same logic value as the main aggressor, otherwise a negative weight is assigned. This is done by comparing aggressor nets for a change of logic value. For this purpose 10 test vector combinations of victim net and by observing the states of aggressor nets for a change of logic value. For this purpose 10 test vector combinations of victim net were used (selected randomly).

In our approach a FAN based ATPG is used and modified by removing the forward propagation step while determining stuck at faults on aggressor wires. Each interconnect is assigned a particular weight with respect to the victim wire considering capacitive coupling effects between the wires. Since layout information was not available we used random weights. Spatial pruning is performed by removing aggressors having weight less than a specified threshold, which is because either the aggressor is too far away from the victim net or its effect is shielded by some other interconnect. Temporal pruning is done by grouping aggressors into time compatible sets. An aggressor is time compatible with another if their switching windows overlap. Switching window can be defined as the time duration within which some active transitions can propagate through the wire [3]. Switching windows are calculated by performing static timing analysis of the circuit assigning random delays to interconnects and gates. Functional pruning is performed by removing static nets from the time compatible aggressor sets. This is done by applying test vector combinations of victim net and by observing the states of aggressor nets for a change of logic value. For this purpose 10 test vector combinations of victim net were used (selected randomly).

Thereafter, the maximum weighted set is calculated. Each aggressor within the set is selected as a main aggressor. An aggressor from the remaining aggressors of the set assumes a positive weight if it can be excited to the same logic value as the main aggressor, otherwise a negative weight is assigned. This is done by comparing the test patterns of aggressor with patterns of victim net. The test patterns obtained for the maximum weighted sets are then recorded. In order to compare maximum 50 test vector combinations can be stored for each fault. Fig. 2 provides an algorithm for obtaining test vectors for positive glitch faults.

Fig. 1 (a) Aggressors A1 & A2 aligned, max peak on victim V’s output, (b) A2 delayed, (c) A1 falling transition

Fig. 2 Algorithm for generating positive glitch fault
3. Test Set Compaction

The most commonly used technique for initial compaction removes equivalent test patterns and repeatedly merges compatible test vector pairs i.e. combining the bits of the vector pairs that do not conflict in their specified values (0, 1) and where one is unspecified (don’t-care bit) the other is taken. Further reduction of the test set is achieved through ordering the faults and so reusing the test patterns through fault-list chaining. The fault-list chaining algorithm has been discussed in [3]. For reducing the test set as a whole, the total number of fault chains formed is more important than finding the longest chain and hence, it is hard to implement as an optimal algorithm. In this work we look into two different approaches for chain formation and compare them with the previous approach.

First Node First Algorithm (FNF)

In FNF algorithm after selecting a starting node the traversal starts by selecting the first node (child) in the forward path, which has not been traversed before and continuing from it in a similar fashion until a leaf is reached. Further traversal from the starting node is made by selecting the first node (parent) in the backward path, which has not been traversed until a root is reached. The nodes from the path are then removed from the graph and the process is continued until all nodes have been traversed. The nodes within the path are thus randomly selected without concerning the path lengths. The advantage of this approach is its simplicity of implementation.

Considering Fig. 3 and selecting node-A as the starting node and then node-G, may result in any of the following graph traversals:

1. (G, F, E, B, C, D, A, H, I)
2. (H, A, B, C, D) — (G, F, E) — (I)
3. (H, A, B, E) — (G, F) — (C, D) — (I), etc.

Longest Path Algorithm (LP)

In this algorithm once a starting node is selected the longest path is chosen by going through all of its child and parent nodes. The path is then removed from the graph and the process is continued until all nodes have been traversed. This algorithm works on the greedy principle. The total number of paths obtained is greatly dependent on the selection of the starting node. Therefore, initially all the root and leaf nodes are selected as starting nodes and then any arbitrary node can be selected. The heuristic used in this algorithm is to find the longest sequence of matching patterns, since longer chains imply that more faults are covered, which also results in reduction of the number of chains.

Considering Fig. 3 and selecting node-A and then node-G as the starting nodes will result in the following graph traversal:

(H, A, B, C, D) — (G, F, E) — (I)

But if the root or leaf nodes are considered e.g. node-I then the following path is obtained: (G, F, E, B, C, D, A, H, I)

4. Results and Comparison

The Crosstalk ATPG program and the compaction routines were written in C language and were tested on several ISCAS’85 benchmark circuits and were compared for the three approaches, LPC (Longest Path considering Cycles) implemented in [3], and the proposed FNF and LP algorithms. Table 1 shows their comparison.

The first column of Table 1 represents the circuit tested, whereas the second column shows the number of victim associated with this circuit. The third column represents the maximum number of aggressors selected randomly per victim wire. The total number of detected faults and their required number of patterns are mentioned in fourth and fifth column respectively. Sixth column represents the reduced number of patterns obtained after merging. The next three sub-columns of each algorithm show the result of the respective algorithm. The first column shows the number of patterns obtained (%Patt), the second the percent reduction in test set (%Cmp) and the third the percent improvement achieved by chaining over merging (%Imp). The best results obtained are shaded. It can be seen that up to 78% reduction in crosstalk glitch fault test set can be obtained under random weights.

5. Conclusion

In this work initially the test patterns were generated for maximal crosstalk effect by considering the spatial, temporal and logical relationships between interconnects, then the test set was compacted. Initial test compaction was performed through merging vector pairs and further through fault list chaining algorithm. In this work we compared three different implementations of fault-list chaining algorithm. It was observed by several test runs of the program that both FNF and LP are better algorithms than LPC.

6. References


<table>
<thead>
<tr>
<th>Circuit</th>
<th>Vic nets</th>
<th>Max Agg</th>
<th>Xtalk Faults</th>
<th># Xtalk vectors</th>
<th># Vectors after merging</th>
<th>LPC</th>
<th>FNF</th>
<th>LP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td># Patt</td>
<td>% Cmp</td>
<td>% Imp</td>
</tr>
<tr>
<td>c432</td>
<td>343</td>
<td>20</td>
<td>610</td>
<td>1220</td>
<td>688</td>
<td>685</td>
<td>43.85</td>
<td>23.1</td>
</tr>
<tr>
<td>c499</td>
<td>440</td>
<td>20</td>
<td>724</td>
<td>1448</td>
<td>952</td>
<td>790</td>
<td>45.44</td>
<td>17.0</td>
</tr>
<tr>
<td>c880</td>
<td>755</td>
<td>5</td>
<td>1234</td>
<td>2468</td>
<td>760</td>
<td>719</td>
<td>70.87</td>
<td>5.4</td>
</tr>
<tr>
<td>c1355</td>
<td>1096</td>
<td>20</td>
<td>1370</td>
<td>2740</td>
<td>1656</td>
<td>1370</td>
<td>50.00</td>
<td>17.3</td>
</tr>
<tr>
<td>c1908</td>
<td>1523</td>
<td>10</td>
<td>2466</td>
<td>4932</td>
<td>1982</td>
<td>1752</td>
<td>64.48</td>
<td>11.6</td>
</tr>
<tr>
<td>c2670</td>
<td>2216</td>
<td>5</td>
<td>3585</td>
<td>7170</td>
<td>1792</td>
<td>1650</td>
<td>76.99</td>
<td>7.9</td>
</tr>
<tr>
<td>c3540</td>
<td>2961</td>
<td>10</td>
<td>4564</td>
<td>9128</td>
<td>3370</td>
<td>2898</td>
<td>68.25</td>
<td>14.0</td>
</tr>
<tr>
<td>c5315</td>
<td>4509</td>
<td>4</td>
<td>7857</td>
<td>15714</td>
<td>3640</td>
<td>3504</td>
<td>77.70</td>
<td>3.7</td>
</tr>
<tr>
<td>c6288</td>
<td>4832</td>
<td>20</td>
<td>6158</td>
<td>12316</td>
<td>3680</td>
<td>2794</td>
<td>77.31</td>
<td>24.1</td>
</tr>
<tr>
<td>c7552</td>
<td>6252</td>
<td>4</td>
<td>10106</td>
<td>20212</td>
<td>4834</td>
<td>4688</td>
<td>77.82</td>
<td>7.2</td>
</tr>
</tbody>
</table>

Fig. 3 Representation of fault-list as a graph, where the nodes represent crosstalk faults and directed edges matching patterns.