General and efficient SAT-based ATPG framework for multiple various faults its application to logic synthesis

Masahiro Fujita
University of Tokyo

(joint research with Alan and Bob of UCB)

Here only considers combinational circuits
Long history with Ed

• First met Ed at a conference (CHDL) in May, 1983
  – Presented a paper on sort of model checking hardware with Prolog implementation
  – Following Japanese fifth generation computer project
• Ed approached me and said, “We are working on somehow similar problem, but our approach is better”
• Since then, we have been collaborating
  – Have several jointly authored papers
• I am not a student, pos-doc, long-time visitor. I am just a frequent short-time visitor
Testing manufacturing faults

• Make sure that manufactured chips behaves as described in the design descriptions

• Introduce fault models for efficient processing
  – Ways for HW to fail can be pre-determined

• Suppose there are $m$ possibly faulty locations and there are $p$ ways of faults for each location
  – Single fault assumption: total number of fault combinations is $m*p$
  – Multiple fault assumption: total number of fault combinations is $p^{m-1}$

• Generation of complete test vectors for multiple faults was (is) believed to be very difficult
How to represent faults “implicitly as part of SAT problem ?

• Introduce circuits and variables that represent faults into each possibly faulty location

• For stuck-at fault:

\[ x \land y \]

\( C = ((ab) \ast \neg X) + Y \)

Circuit to model stuck-at 1 and 0 faults
y=1: stuck-at 1
x=1: stuck-at 0
Multiple faults

• For each possibly faulty location, insert the circuit to represent stuck-at faults

• If all of $x_i$ and $y_i$ are 0, no fault in the circuit
  – Can represent all fault combinations: $3^m - 1$

• Circuit transformation can be defined in the same way: AND -> (AND, OR, NAND, NOR, EOR, ...)
  – Exponentially many transformations are considered
ATPG with SAT

\( X: \) Faults

\( In: \) Inputs

- Under some inputs, some faults can be detected

\[ \exists In. X. \text{Faulty} (In, X) \neq \text{NoFault} (In) \]

\[ \Rightarrow \text{SAT solution is} \ (in_1, x_1) \]

- There are many techniques based on circuit analysis for much more efficient SAT-based ATPG

  – But here we use this very simple one...
How to eliminate already detected faults

• Under some inputs, some faults can be detected

\[ \exists In.X . Faulty (In, X) \neq NoFault (In) \]

\[ \implies \text{SAT solution is } (in_1, x_1) \]

• Faults that cannot be detected by \( in_1 \)

\[ \exists X . Faulty (in_1, X) = NoFault (in_1) \]

• When generating the next test vector, add the above constraints
  
  – Then we are targeting only remaining faults !
  
  – Should continue until the resulting SAT becomes UNSAT
The problem is essentially an incremental SAT

• \((in_1, in_2, \ldots, in_n)\) are complete test vectors for multiple stuck-at faults

\[ \exists In.X.\text{Faulty} (In, X) \neq \text{NoFault} (In) \Rightarrow \text{SAT}, \text{ solution is } (in_1, x_1) \]

\[ \exists In.X.\text{Faulty} (In, X) \neq \text{NoFault} (In) \land \text{Faulty} (in_1, X) = \text{NoFault} (in_1) \Rightarrow \text{SAT, solution is } (in_2, x_2) \]

\[ \exists In.X.\text{Faulty} (In, X) \neq \text{NoFault} (In) \land \text{Faulty} (in_1, X) = \text{NoFault} (in_1) \land \text{Faulty} (in_2, X) = \text{NoFault} (in_2) \Rightarrow \text{SAT, solution is } (in_3, x_3) \]

\[ \ldots \]

\[ \exists In.X.\text{Faulty} (In, X) \neq \text{NoFault} (In) \land \text{Faulty} (in_1, X) = \text{NoFault} (in_1) \land \text{Faulty} (in_2, X) = \text{NoFault} (in_2) \land \ldots \land \text{Faulty} (in_{n-1}, X) = \text{NoFault} (in_{n-1}) \Rightarrow \text{SAT, solution is } (in_n, x_n) \]

\[ \exists In.X.\text{Faulty} (In, X) \neq \text{NoFault} (In) \land \text{Faulty} (in_1, X) = \text{NoFault} (in_1) \land \text{Faulty} (in_2, X) = \text{NoFault} (in_2) \land \ldots \land \text{Faulty} (in_{n-1}, X) = \text{NoFault} (in_{n-1}) \land \text{Faulty} (in_n, X) = \text{NoFault} (in_n) \Rightarrow \text{UNSAT} \]
Recent findings

• Numbers of complete test pattern for single and multiple faults are **not much different**
  – Need a little bit more test patterns for multiple faults
• ATPG (automatic test pattern generation) is not so much inefficient
  – Entire process of ATPG for multiple faults can be formulated as **single incremental SAT problem**
• Test patterns generated **guarantee 100% correctness**
  – Very small numbers of test patterns are sufficient for typical fault models (always less than 5,000 !?)
  – **Why ?** The ways for HW to fail is prefixed (but exponentially many ways)
Formal analysis with \((in_1, in_2, \ldots, in_n)\)

• If the circuit is correct with \((in_1, in_2, \ldots, in_n)\), it is guaranteed to be correct for all input patterns

• Why?
  – The ways for circuits to be buggy/faulty are controlled by \(X\) variables (parameter variables)
  – Circuits cannot change themselves freely
  – Instead must follow the possible values of \(X\)
  – *This dramatically reduced the ways to fail*
  – But multiple bugs are take care
  – *The ways to fail are exponentially many*
A little bit surprise

- If we start ATPG for multiple faults with the sets of test vectors for single faults, we do not need many more test vectors!

<table>
<thead>
<tr>
<th>Name</th>
<th>Test SSA</th>
<th>Tests Multiple SA (reading tests ssa)</th>
<th>Additional Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Vars</td>
<td>Clauses</td>
</tr>
<tr>
<td>s1423</td>
<td>25</td>
<td>36689</td>
<td>57739</td>
</tr>
<tr>
<td>s1196</td>
<td>117</td>
<td>150946</td>
<td>246265</td>
</tr>
<tr>
<td>s1238</td>
<td>130</td>
<td>186570</td>
<td>389819</td>
</tr>
<tr>
<td>s1488</td>
<td>108</td>
<td>171621</td>
<td>270965</td>
</tr>
<tr>
<td>s1494</td>
<td>110</td>
<td>173752</td>
<td>280337</td>
</tr>
<tr>
<td>s5378</td>
<td>102</td>
<td>428024</td>
<td>729438</td>
</tr>
<tr>
<td>s38417</td>
<td>120</td>
<td>3724712</td>
<td>5492811</td>
</tr>
<tr>
<td>s35932</td>
<td>30</td>
<td>1473112</td>
<td>2175030</td>
</tr>
</tbody>
</table>
Asg with SAT for logic synthesis

$X$: Circuit transformation (Faults)

$In$: Inputs

• Under some inputs, some faults can be detected

$\exists In. X. Faulty (In, X) \neq NewSpec (In)$

=> SAT solution is $(in_1, x_1)$

• Then how can we come up with transformations by which we can realize the spec?

  – Key observation: Redundant faults
By the way

• International Test Conference (ITC) has been organized for more than 30 years
• It has been dealing with “hardware” testing in general
• But like to extend the scope to include “software” testing as well
• Please consider submitting papers to ITC 2015, which will be Disneyland (Los Angeles) Hotel in September