ECE 586 – Fault Detection in Digital Circuits
Lecture 13 Fault Simulation II

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

February 25, 2015
Reading Assignment

▶ This lecture: 5.2
▶ Next lecture: 5.3
Outline

Deductive Fault Simulation

Concurrent Fault Simulation
Deductive Fault Simulation

- Inputs/Outputs: same as parallel fault simulation.
  - The good circuit $N$, a set of $F$ faults to be simulated, A test vector.
  - Output 1-bit per fault indicating whether the test vector detects it.
- Idea: for each line, keep a list of faults for which the value is different from the correct one.
  - Can be treated as a compression of the word per line used by parallel fault simulation – only the bits different from the correct one (bit 0) are stored in the fault-list.
  - The lists at the primary outputs contain all the detectable faults with respect to the test vector.
- Challenges: how to maintain the list if it's large?
Example

Faults: \( \{a_0, \bar{a}_1, b_1, c_0, c_1, d_1, e_0, g_0, h_0, h_1\} \)

\[
L_a = \{a_1\}, \quad L_b = \{b_1\}, \quad L_c = \{c_0\}, \quad L_d = \emptyset, \quad L_e = \emptyset
\]

\[
L_f = L_a \cap L_b = \emptyset, \quad L_g = L_c \cup \{g_0\} = \{c_0, g_0\}, \quad L_h = L_c \cup \{h_0\} = \{c_0, h_0\}
\]

\[
L_j = L_g - L_f = \{c_0, g_0\}, \quad L_i = L_d \cup L_h = \{c_0, h_0\}
\]

\[
L_k = L_i - L_e = \{c_0, h_0\}
\]

\[
L_m = L_k - L_j = \{h_0\}
\]

(Abramovici et al., 1990)

▶ Only \( h_0 \) is detected by 00110.
Example

Faults: \( \{a_0, \bar{a}_1, b_1, c_0, c_1, d_1, e_0, g_0, h_0, h_1\} \)

\( L_a = \{a_1\} \), \( L_b = \{b_1\} \), \( L_c = \{c_0\} \), \( L_d = \emptyset \), \( L_e = \emptyset \)

\( L_f = L_a \cap L_b = \emptyset \), \( L_g = L_c \cup \{g_0\} = \{c_0, g_0\} \), \( L_h = L_c \cup \{h_0\} = \{c_0, h_0\} \)

\( L_j = L_g - L_f = \{c_0, g_0\} \), \( L_i = L_d \cup L_h = \{c_0, h_0\} \)

\( L_k = L_i - L_e = \{c_0, h_0\} \)

\( L_m = L_k - L_j = \{h_0\} \)

(Abramovici et al., 1990)

- Only \( h_0 \) is detected by 00110.
We may remove $h_0$ as it is already detected.

A new test vector $11110$ can be simulated using event-driven simulation based on previous results.

For both signal value and fault-list, only the lines that may potentially be changed need to be considered in the worst-case: $a, b, f, j, m$
Event-Driven Fault-List Propagation

We may remove $h_0$ as it is already detected.

A new test vector 11110 can be simulated using event-driven simulation based on previous results.

For both signal value and fault-list, only the lines that may potentially be changed need to be considered in the worst-case: $a$, $b$, $f$, $j$, $m$

Faults: $\{a_0, \tilde{a}_1, b_1, c_0, c_1, d_1, e_0, g_0, h_0, h_1\}$

$L_a = \{a_0\}$, $L_b = \emptyset$,

$L_a = \{a_1\}$, $L_b = \{b_1\}$, $L_c = \{c_0\}$, $L_d = \emptyset$, $L_e = \emptyset$

$L_f = L_a \cap L_b = \emptyset$, $L_g = L_c \cup \{g_0\} = \{c_0, g_0\}$, $L_h = L_c \cup \{h_0\} = \{c_0, h_0\}$

$L_f = \{a_0\}$, $L_j = L_f \cap L_g = \emptyset$.

$L_j = L_g$, $L_f = \{e_0, g_0\}$, $L_i = L_d \cup L_h = \{c_0, h_0\}$

$L_k = L_i - L_e = \{c_0, h_0\}$

$L_m = L_k - \{h_0\}$, $L_m = \tilde{L}_k - \tilde{L}_j = \{c_0\}$.

(Abramovici et al., 1990)
Fault Storage

\[ L_a = \{3, 9, 12\} \]

(a)

\[ \begin{array}{c}
\text{a} \\
\text{3} \\
\text{9} \\
\text{12} \\
\end{array} \rightarrow \begin{array}{c}
\text{3} \\
\text{9} \\
\text{12} \\
\end{array} \]

(b)

(c)

\[ W+1 \]

\[ W+2 \]

\[ W \]

(d)

Figure 5.12 Three storage structures for lists (a) Fault list (b) Linked list (c) Sequential table (d) Characteristic vector (Abramovici et al., 1990)

- Note that characteristic vector is equivalent to parallel fault simulation.
Discussions for Deductive Fault Simulation

- Similar to parallel fault simulation, cannot simulate circuits directly when non-Boolean operations are involved and not efficient for multivalued logic.
  - Difficult to define rules for fault-list propagation.
- May benefit from event-driven simulations.
  - However, it is still possible to have more activities than value changes since we need to propagate fault-lists, which could be very costly (need to support $\cup$ and $\cap$ operations).
Similar to parallel fault simulation, cannot simulate circuits directly when non-Boolean operations are involved and not efficient for multivalued logic.

- Difficult to define rules for fault-list propagation.

May benefit from event-driven simulations.

- However, it is still possible to have more activities than value changes since we need to propagate fault-lists, which could be very costly (need to support $\cup$ and $\cap$ operations).
Outline

Deductive Fault Simulation

Concurrent Fault Simulation
Concurrent Fault Simulation

▶ Idea: keep a concurrent fault list for each element.
   ▶ For those faults cause the computation at the element being different from the good circuit.
   ▶ As a comparison, a list of faults is kept for each line in deductive fault simulation.
   ▶ Recall $a_1$ refers to the fault $a$ s-a-1 and so on.

▶ What is the benefit over deductive fault simulation?
Concurrent Fault Simulation

Idea: keep a concurrent fault list for each element.
- For those faults cause the computation at the element being different from the good circuit.
- As a comparison, a list of faults is kept for each line in deductive fault simulation.
- Recall $a_1$ refers to the fault $a$ s-a-1 and so on.

What is the benefit over deductive fault simulation?

Figure 5.15
Concurrent fault list for gate $c$
(a) Pictorial representation
(b) Tabular representation

(Abramovici et al., 1990)
Consider a scenario where the input vector changes a little.

- Change of good signal value may make faults visible.
  - They need to be propagated.
Consider a scenario where the input vector changes a little.
Change of good signal value may make faults visible.
They need to be propagated.
The fault lists are updated by adding and deleting faults.

- Unlike deductive fault simulation where the whole lists need to be reconstructed.

- Note that local faults will never be deleted, e.g. $c_1$.
  - Local faults are faults at inputs/outputs of the element.

- The element may need to be re-evaluated for the faulty circuits, e.g. $\beta$ and $d_1$ where the top signal is good.
Events on Lines

Events on a line due to faults are maintained as a list of fault/faulty value pairs \((f, v'_f)\).

Events in the good circuit can be maintained in the same list using a special fault \(f = 0\).
Faults involving multivalued logic are propagated as it is and can be handled as long as the element can be evaluated.
Handle Reconvergent Fanout

(Fig. 5.22, Abramovici et al., 1990)

▶ Only a single copy of the fault should be preserved if it is presented at multiple inputs.
Discussions for Concurrent Fault Simulation

- Able to simulate circuits with non-Boolean operations and multivalued logic.
  - Details are hidden in the evaluation of elements.
  - Allow to utilize hierarchical simulation frameworks.
- Concurrent fault lists are updated incrementally, saving computations compared to deductive fault simulation.
- Storage of the lists is a concern.
  - Use multiple passes to reduce the sizes of the lists per pass.
Discussions for Concurrent Fault Simulation

- Able to simulate circuits with non-Boolean operations and multivalued logic.
  - Details are hidden in the evaluation of elements.
  - Allow to utilize hierarchical simulation frameworks.
- Concurrent fault lists are updated incrementally, saving computations compared to deductive fault simulation.
- Storage of the lists is a concern.
  - Use multiple passes to reduce the sizes of the lists per pass.
Discussions for Concurrent Fault Simulation

- Able to simulate circuits with non-Boolean operations and multivalued logic.
  - Details are hidden in the evaluation of elements.
  - Allow to utilize hierarchical simulation frameworks.
- Concurrent fault lists are updated incrementally, saving computations compared to deductive fault simulation.
- Storage of the lists is a concern.
  - Use multiple passes to reduce the sizes of the lists per pass.
In most cases, it is possible to improve an existing algorithm by overcoming its short-comings.