Fault Simulation

Problem and Motivation

• Fault Simulation Problem:
  – Given
    ▪ A circuit
    ▪ A sequence of test vectors
    ▪ A fault model
  – Determine
    ▪ Fault coverage
      • Fraction (or percentage) of modeled faults detected by test vectors
    ▪ Set of undetected faults
• **Motivation:**
  
  - Determine the quality of a given set of test vectors (test quality).
  - Find undetected faults with respect to a given test set.
Usages of Fault Simulators

a) Test grading
   – Determine the quality of a given test set.

b) Test Generation
   – Generate set of vectors to detect a set of faults.

c) Fault diagnosis
   – Identify the location of a fault.

d) Design for test (DFT)
   – Identification of points that may help improve test quality.

A typical use

1. Select Target Fault Set F
2. Generate Test for some fault in F
3. Fault Simulation
4. Remove Detected Faults from F
5. If F is empty, STOP
How to simulate faults?

- A simple fault simulation algorithm can be obtained by repeated use of any logic simulation algorithm.
  - For each vector, simulation of the fault-free version of the circuit can be followed by simulation of each of the faulty versions of the circuit.

If \( n \rightarrow \) number of input vectors
\( k \rightarrow \) number of faults
Then, number of simulation runs = \( n \times k \)

Some Basic Concepts

- **Fault Masking:**
  - A single-fault test can fail to detect the target fault if another fault is also present.

The fault y/1 prevents the fault x/0 from being detected by the vector 0000.
• **Redundant Faults:**
  
  - For some faults, no tests may exist.
    
    • Such faults are untestable.
  
  - Arise due to some redundancy present in the circuit.

![Diagram](image)

  Untestable

• **Fault Collapsing:**
  
  - The number of faults that need to be simulated can be decreased by exploiting two relations between a pair of faults $f_i$ and $f_j$:
    
    a) Fault equivalence
    
    b) Fault dominance
  
  - Used to reduce the fault simulation time.
• **Fault Dropping:**
  - It is the practice in which faults detected by a vector are deleted from the fault list prior to the simulation of any subsequent vector.
    - Decreases complexity of fault simulation.
    - Cannot be used for all fault simulation algorithms.

---

**Fault Equivalence**

- **Definition:**
  - Two faults $f_i$ and $f_j$ in a circuit are said to be equivalent if the corresponding faulty versions of the circuit are identical.
    - All tests that detect $f_i$ also detect $f_j$.
  - **Example:**
    - An input s-a-0 and output s-a-0 in an AND gate.

- **A point to note:**
  - Number of fault sites in a gate-level circuit
    $$= \#PI + \#gates + \#(\text{fanout branches})$$
Equivalence Rules

AND

OR

NAND

NOR

WIRE

NOT

FANOUT

Equivalence Example

 Collapse ratio = \( \frac{20}{32} = 0.625 \)
Number of lines = 16
Number of single stuck-at faults = 32
Number of faults removed by equivalence collapsing = 12

Thus, collapse ratio = 20 / 32 = 62.5 %

Contd.

• *Fault collapsing using equivalence relation*: 
  – All single faults of a logic circuit can be divided into disjoint equivalence subsets, where all faults in a subset are mutually equivalent.
  – A collapsed fault set contains one fault from each equivalence subset.
Fault Dominance

• **Definition:**
  - If all tests for some fault $f_i$ also detect another fault $f_j$, then $f_j$ is said to dominate $f_i$.
  - If two faults $f_i$ and $f_j$ dominate each other, then they are equivalent.

• **Example:**
  - A 3-input AND gate.
    - $f_i \rightarrow$ s-a-1 fault in one input of the gate
    - $f_j \rightarrow$ s-a-1 in the gate output
  - Tests for $f_i \rightarrow 011$
  - Tests for $f_j \rightarrow 000, 001, 010, 011, 100, 101, 110$
  - Thus, $f_j$ dominates $f_i$.
• **Fault collapsing using dominance relation:**
  
  – If a fault $f_j$ dominates another fault $f_i$, then $f_j$ can be removed from the fault list.
  
  – In a tree circuit (without fanouts), the primary input faults form a dominance collapsed fault set.
    • Proof → straightforward

---

**Observation**

• In general, the complexity of identifying all fault dominance and equivalence relations is high.

• Hence, in practice, the equivalence and dominance relations identified between single stuck-at faults associated with inputs and outputs of gates are used in an iterative fashion to achieve significant fault collapsing.
• An example:

```
A
B
```

```
\[ f_1 : s-a-1 \]
\[ f_2 : s-a-0 \]
\[ f \]
```

\( f_1 \) dominates \( f_2 \) though not obvious from the characteristic of the gate alone

---

**Checkpoints**

• **Definition:**
  - Primary inputs and fanout branches of a combinational circuit are called * checkpoints.

• **Checkpoint theorem:**
  - A test set that detects all single (multiple) stuck-at faults on all checkpoints of a combinational circuit, also detects all single (multiple) stuck-at faults in that circuit.
    - *A SIMPLE HEURISTIC FOR FAULT COLLAPSING.*
Alternatives to Fault Simulation

• Prototyping with fault injection capabilities
  – Costly
  – Limited fault injection capability
  – Design changes hard to implement
  – Long lead time

• Hardware emulators
  – Costly
  – Require special hardware

Fault Simulator in a VLSI Design Flow
Fault Simulation Algorithms

• Serial
• Parallel
• Deductive
• Concurrent
• Others
  – Critical path tracing
  – Parallel pattern, etc.

[A] Serial Algorithm

• Algorithm:
  – Simulate fault-free circuit and save responses.
  Repeat following steps for each fault in the fault list:
    • Modify netlist by injecting one fault.
    • Simulate modified netlist, vector by vector, comparing responses with saved responses.
    • If response differs, report fault detection and suspend simulation of remaining vectors.
• **Advantages:**
  - Easy to implement; needs only a true-value simulator, less memory.
  - Most faults, including analog faults, can be simulated.

• **Disadvantage:**
  - Much repeated computation; CPU time prohibitive for VLSI circuits.

---

**Serial Algorithm (Cont.)**

- **Alternative:** Simulate many faults together.

```
Test vectors  --- Fault-free circuit  --- Comparator  \ f1 detected?
    |                      /            |
    |                      / Comparator |
    v                      v Comparator
Circuit with fault f_1  \ f2 detected?  \ fn detected?
    |          |          |          |
    |          |          |          |
    |          |          |          |
Circuit with fault f_2  \               \ fn detected?
    |          |          |    |          |
    |          |          |    |          |
    |          |          |    |          |
Circuit with fault f_n
```
[B] Parallel Fault Simulation

- Takes advantage of multi-bit representation of data and availability of bitwise operations.
  - Compiled-code method.
  - Works best with two-states (0,1).
- Extends the basic concept of parallel logic simulation.

Basic mechanism:
- In each pass of simulation, the fault-free circuit as well as (W-1) faulty versions are simulated in parallel for a given vector, where ‘W’ is the machine word length.
  - If ‘q’ faults are to be simulated for a vector, \(\lceil q/(W-1) \rceil\) passes are required.
  - Fault dropping cannot be used.
• How to insert faults?
  – For each fault, an appropriate fault mask is used to insert the effect of the fault at its site.
  – For each line, the fault mask is comprised of two W-bit integers, $M_Z$ and $M_O$.

<table>
<thead>
<tr>
<th>$i^{th}$ bit of mask</th>
<th>$M_Z$</th>
<th>$M_O$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault-free version of circuit</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Faulty circuit with $C_i / 0$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Faulty circuit with $C_i / 1$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Faulty circuit with a fault not located at $C_i$</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

If $Z$ denotes the logic value (vector) computed at $C_i$, it is corrected using the expression:

$$Z' = (Z \text{ AND } M_Z) \text{ OR } M_O$$
## Parallel Fault Simulation Example

![Diagram of a circuit](image)

- **Fault Masks:**

<table>
<thead>
<tr>
<th></th>
<th>M&lt;sub&gt;z&lt;/sub&gt;</th>
<th>M&lt;sub&gt;O&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>11111</td>
<td>00000</td>
</tr>
<tr>
<td>b</td>
<td>11111</td>
<td>00000</td>
</tr>
<tr>
<td>c</td>
<td>10111</td>
<td>00000</td>
</tr>
<tr>
<td>d</td>
<td>11111</td>
<td>00000</td>
</tr>
<tr>
<td>e</td>
<td>11101</td>
<td>00001</td>
</tr>
<tr>
<td>f</td>
<td>11111</td>
<td>00100</td>
</tr>
<tr>
<td>g</td>
<td>11111</td>
<td>00000</td>
</tr>
</tbody>
</table>
**[C] Parallel-Pattern Single-Fault Propagation (PPSFP)**

- **Basic idea:**
  - A batch of vectors are simulated in parallel.
  - If the fault list contains ‘q’ faults during the simulation of a batch of ‘W’ vectors, then their simulation is carried out in a total of (q+1) passes.
    - In each pass after the first, one fault is inserted into the circuit.

- Faster than parallel fault simulation.
  - Computation of logic values is faster at all lines except at the fault site.

- **Limitations:**
  - Applicable to combinational circuits only.
[D] Deductive Fault Simulation

- This method utilizes a dynamic data structure:
  - One-pass simulation.
  - Each line $k$ contains a list $L_k$ of faults detectable at $k$.
- It comprises of two (interleaved) steps:
  a) Fault-free circuit simulation is performed for the given vector.
  b) The value implied by the vector at every line in each faulty circuit is deduced (using set theoretic rules).
- Originally implemented for 3-valued logic.

Fault List:
- The fault list $L_i$ associated with line ‘i’ is the set of all faults $\{f\}$ that cause the values of ‘i’ in $N$ and $N_f$ to differ at the current simulation time.
  - $N$ is the fault-free circuit, and $N_f$ the circuit in presence of fault ‘f’.
  - If ‘i’ is a primary output (PO), then $L_i$ is the set of detected faults.
• **Computation of fault list:**
  - Uses a method called fault-list propagation.
  - Performed when one of the following occurs:
    - Logic event: change in signal value of an input or output line of the gate.
    - List event: change in the fault list of one or more inputs of the gate.
  
• Some illustrative examples to illustrate computation of fault list is shown next.

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>L_Z = L_A \cup L_B \cup {Z/0}</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>L_Z = L_A \cup L_B \cup {Z/1}</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>L_Z = (L_A \cap L_B) \cup {Z/1} = (L_A - L_B) \cup {Z/1}</td>
</tr>
</tbody>
</table>
• **Limitations:**
  - Set-theoretic rules are difficult to derive for complex gates and higher-level functional blocks.
  - Gate delays are difficult to use.
  - Memory requirement is unpredictable.

An Example
[E] Concurrent Fault Simulation

• **Basic motivating factor:**
  - Most of the time during simulation, most of the values in most of the faulty circuits agree with the corresponding values in the good circuit.

• **Basic concept:**
  - The fault-free version of the circuit, and each of its faulty versions, are concurrently simulated for a given vector.
    • Simulates the good circuit \( N \).
    • For every faulty circuit \( N_f \), simulate only those elements in \( N_f \) that are different from the corresponding ones in \( N \).
• **Data structure used:**
  - Concurrent fault list, in which entries are of the form:
    
    `< fault, input_values, output_value >`

    
    ![Logic gate diagram](image)

    
    
    
    ```plaintext
    a/1 : 11; 1
    b/0 : 00; 0
    c/1 : 01; 1
    ```

• **Alternate way of representing the fault list**

    ![Logic gate diagram](image)
• **Maintaining the fault lists:**
  
  - Information about a fault will be entered in the fault list if one or more of the following conditions are satisfied:
    
    a) The fault ‘£/x’ is local to the gate.
    
    b) The value implied at at least one input or output of the gate is different from that implied at the corresponding line in the fault-free version of the circuit.

  - Initially, the values of all lines are set to the unspecified state ‘X’.
  
  - An entry is removed from the fault list if the corresponding input/output values are identical to that of the fault-free circuit.
• **Advantages and limitations:**
  – Runs faster as compared to deductive fault simulation for most of the circuits.
  – Memory requirement is higher since the sizes of the fault lists are greater.
  – It can easily be extended to cases where the results of fault simulation depend on timing of events.
    • Delays of the gates can be different.

An Example
### [F] Critical Path Tracing

- Differs from the paradigms discussed earlier in two main ways:
  - a) The method targets all faults within certain parts of a circuit.
    - The complexity of fault simulation is independent of the number of faults.
  - b) The method can only be applied to fanout-free circuits, in its strictest form.
    - Handling of fanouts requires explicit simulation, and hence more computational overheads.

### An Example
Fault Sampling

- A randomly selected subset (sample) of faults is simulated.
- Measured coverage in the sample is used to estimate fault coverage in the entire circuit.
- **Advantage:**
  - Saving in computing resources (CPU time and memory.)
- **Disadvantage:**
  - Limited data on undetected faults.

Motivation for Sampling

- Complexity of fault simulation depends on:
  - Number of gates
  - Number of faults
  - Number of vectors

- Complexity of fault simulation with fault sampling depends on:
  - Number of gates
  - Number of vectors
Random Sampling Model

All faults with a fixed but unknown coverage

$N_p =$ total number of faults (population size)

$C =$ fault coverage (unknown)

$N_s =$ sample size

$c =$ sample coverage (a random variable)

Summary

- Fault simulator is an essential tool for test engineers.
- Concurrent fault simulation algorithm offers the best choice.
- For large circuits, the accuracy of random fault sampling only depends on the sample size (1,000 to 2,000 faults) and not on the circuit size.
  - The method has significant advantages in reducing CPU time and memory needs of the simulator.