Speeding Up
Byzantine Fault Diagnosis Using Symbolic Simulation
Problem: Fault Diagnosis

Circuit Under Diagnosis (CUD)

test patterns

expected response

not equal!

faulty response

a chip with defects inside

Question: Where are the fault locations?
Fault Classification

Faults in A logic IC

Single Fault
- Two-net fault
  - bridging fault

Multiple Fault
- One-net fault
  - stuck-at node
  - logical node fault
  - Byzantine node fault
  - open

A Node Type of Fault

The faulty output could be ambiguous.
Byzantine Open Fault

- Definition:
  - A fault that causes an ambiguous voltage level

\[ \alpha \sim 2.5 \text{ V} \]

G1 → G2 → G3

- open fault
- pseudo ‘0’
- pseudo ‘1’

'1' → '0'
Motivation

• Lavo, Larabee, and Chess [1996]
  – Use Stuck-At Fault Simulation

• Venkataraman and Drummonds [2000]
  – Based on exhaustive enumeration
  – Complexity grows exponentially

• Our approach
  – Uses symbolic technique
  – Performs implicit enumeration
  – Exact and Efficient
  – Does not cause memory explosion
Outline

• Introduction
• Our Algorithm
• Experimental Results
• Conclusions
Terminology

- **Circuit Under Diagnosis (CUD):** The Faulty Chip
- **Circuit Model (MODEL):** The Gate-Level Netlist
- **Failing Input Vector:** Causes Mismatches
Methodology: Inject and Evaluate

Grading a signal’s suspicion degree of being a fault:
1. Perform **symbolic injection** at f for each failing vector
2. Count the **curable vectors and curable outputs**
Overall Algorithm

failing input vectors  design model  faulty response

for each failing vector \( v \) {
    for each suspect signal \( f \) {
        Step 1: perform symbolic injection
        Step 2: perform symbolic propagation
        Step 3: curability check
    }
}

Sort the signals by the no. of “curable vectors”,
If tied, sort by the no. of “curable outputs”

Sorting criterion

ranking of each signal’s possibility
of being a fault location
Step 1: Symbolic Injection

A symbolic injection \textit{implicitly enumerates} $2^k$ injections for a signal with $k$ fanout branches.
Step 2: Symbolic Propagation

Speed-Up Techniques:
(1) OBDD, (2) Event-Driven Simulation
Step 3: Curability Check

To cure this failing input vector $v$ by injection at $\alpha$:

$$\exists x_1 \ldots x_k \left\{ \bigcap_{i=1}^{m} \left[ R_i (X, v) \equiv CUD_i (v) \right] \right\} \Rightarrow (x_1 \equiv 0) \cap (x_3 \equiv 0) \cap (\overline{x_1 x_2} \equiv 0)$$
Outline

• Introduction
• Our Algorithm
• Experimental Results
• Conclusions
Experimental Setup

Injection of Byzantine stuck-at faults:
(1) pick one signal at a time randomly
(2) tie each fanout branch to either 0 or 1 randomly
Accuracy of Single Fault Diagnosis

Average first-hit index
(100 faulty circuit for each benchmark)

CPU time < 11.2 seconds

ISCAS-85 benchmark circuits
Accuracy of Double Fault Diagnosis

Average first-hit index
(100 faulty circuit for each benchmark)

CPU time < 211 seconds

ISCAS-85 benchmark circuits
Conclusions

- We proposed a symbolic formulation
  - For Byzantine fault diagnosis
  - On top of the inject-and-evaluate paradigm

- Major computation includes
  - (1) Symbolic injection
  - (2) Symbolic propagation with OBDD
  - (3) Curability check

- We achieved accurate results efficiently!