Concurrent Fault Simulator

Aakarsh A
Department of Electronics and Electrical Communication Engineering
Indian Institute of Technology, Kharagpur

October 17, 2025

Overview

1. What and Why?
2. Architecture and Description
3. Simulation Results
4. Future Work

Problem Statement at a glance

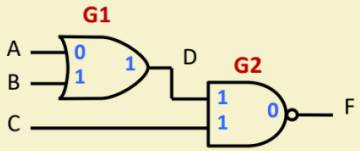
Implement a concurrent fault simulator that will generate the statistics regarding the faults detected by a given set of test vectors.

1. The gate-level circuit netlist must be read from a structural Verilog file.
2. You must prepare a collapsed list of single stuck-at faults.
3. Read a file containing the list of test vectors (choose any suitable format).
4. The simulator must generate the statistics in a properly formatted output file.

Concurrent Fault simulation

Why? To simulate and identify the single stuck at faults. A method is faster when redundant computations are avoided.

What?

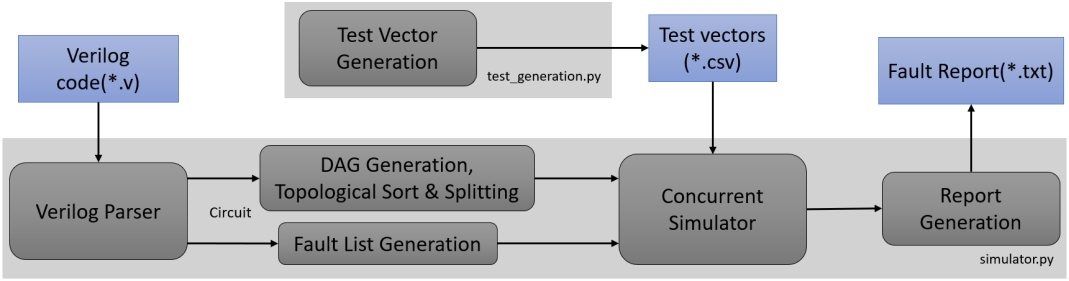


Test Vector 1:
A = 0, B = 1, C = 1

A/1 : 11; 1 B/0 : 00; 0 D/0 : 01; 0	G1
D/0 : 01; 1 C/0 : 10; 1 F/1 : 11; 1	G2
B/0 : 01; 1	

Faults D/0, C/0, F/1, B/0 are detected.

Block Diagram



Core Data Structures

```python
class Circuit:
    def __init__(self, name: str):
        self.name = name
        self.PIs: List[str] = []
        self.POs: List[str] = []
        self.wires: Set[str] = set()
        self.gates: List[Gate] = []
        self.drivers: Dict[str, Gate] = {}
        self.fanout: Dict[str, List[str]] = collections.defaultdict(list)
        self.state_nodes: List[str] = []
        self.dff_defs: List[Gate] = []
```

```python
class Gate(NamedTuple):
    gtype: str
    out: str
    ins: Tuple[str, ...]
```

Description

1. **Verilog Parser:** Produces a class called **Circuit** containing all the PIs, POs, wires and gates by text parsing.
2. **DAG Generation and Splitting:** Applies **Kahn's Algorithm** for topological sort and splits the graphs at Flipflops.
3. **Fault List Generation:** Generates exhaustive fault list for all nets by taking the circuit object as the input.
4. **Concurrent Simulator:** Creates injection mask (pair of vectors for each gate) for storing the fault lists in format ab;z for PI, similar format for nets and it is iterated through every test vector. Fault list is updated after every iteration. After all iterations, it gives us the list of all detected faults and finally we have the required report generation giving the following statistics: detected/undetected faults, coverage and first vector to detect a particular stuck-at-fault.

Fault Coverage reports

```
Fault Simulation Report:

Top module       : mux2to1
Primary Inputs   : A, B, Sel
Primary Outputs  : Y
Vectors simulated: 6
Faults (total)   : 14
Detected         : 14
Coverage         : 100.00%

Detected Faults (net, sa, first_detected_at_vector):
  id=   0 : A s-a-0 @ v3
  id=   1 : B s-a-0 @ v5
  id=   2 : Sel s-a-0 @ v5
  id=   3 : Y s-a-0 @ v3
  id=   4 : nSel s-a-0 @ v3
  id=   5 : w1 s-a-0 @ v3
  id=   6 : w2 s-a-0 @ v5
  id=   7 : A s-a-1 @ v1
  id=   8 : B s-a-1 @ v6
  id=   9 : Sel s-a-1 @ v2
  id=  10 : Y s-a-1 @ v1
  id=  11 : nSel s-a-1 @ v6
  id=  12 : w1 s-a-1 @ v1
  id=  13 : w2 s-a-1 @ v1

Undetected Faults:
```

```
Fault Simulation Report

Top module       : seq_ckt
Primary Inputs   : A, B, clk
Primary Outputs  : Q
State Nodes (Q)  : Q, Q
Vectors simulated: 10
Faults (total)   : 10
Detected         : 8
Coverage         : 80.00%

Detected Faults (net, sa, first_detected_at_vector):
  id=   0 : A s-a-0 @ v6
  id=   1 : B s-a-0 @ v6
  id=   2 : Q s-a-0 @ v6
  id=   4 : w1 s-a-0 @ v6
  id=   5 : A s-a-1 @ v8
  id=   6 : B s-a-1 @ v4
  id=   7 : Q s-a-1 @ v1
  id=   9 : w1 s-a-1 @ v2

Undetected Faults:
  id=   3 : clk s-a-0
  id=   8 : clk s-a-1
```

Future Work

- To implement Fault collapsing and Fault dominance into the flow.
- To expand the code to **event-driven simulation** instead of updating injection mask of every gate for every test vector iteration.
- (Possibility - if time permits) Integrating with **HDL toolchain** (**Yosys** - Open source tool) for generating a **JSON netlist** for any Verilog code (structural/behavioural) and make the changes in the python script to read through the netlist. This **generalises** the simulator for any given synthesizable HDL code.
- Verifying fault coverage for larger and complex combinational and sequential circuits, given test vectors.

The End