Combinational D-Algorithm

L Lakshmanan, Ananya Sane, Jaishnav Yarramaneni, Eswara Rohan

Abstract

This report is for the Design for Testability course project, which required our team to design a software that is capable of taking a circuit in netlist format as an input (subject to limitations), and generating test vectors to test specific stuck-at faults by using the D-Algorithm. The software takes the circuit as input and simulates faults at all the edges, and finds test vectors (if they exist) for said faults. Our implementation of the D-Algorithm involves using functions specifically to find d-frontiers and j-frontiers, and perform justifications and backtrack to find the test pattern. The code is attached with the submission, and sample outputs and inputs are given in the report.

1 Introduction

Test pattern generation is one of the most important steps in testing the design in VLSI circuits. This is done with the help of an Automatic Test Pattern Generator (ATPG). The ATPG uses algorithms to generate test patterns and the D-algorithm is one such method of deterministic test pattern generation along with the PODEM and FAN algorithms. D-algorithm defines a construct called the D-algebra, which is used for the test pattern generation. This algorithm uses concepts like fault activation, propagation, justification, and backtracking to find the test vector that will enable us to detect the fault if the fault is testable.

2 Problem Statement

The main problem statement for the project is to construct a software that can take circuits as inputs and give us the corresponding output test vectors for possible faults in the circuit by using the D-Algorithm.

3 Working of D-Algorithm

The D-Algorithm is implemented with the help of the D-Algebra, which consists of a 5-value logic given by 1, 0, D, D', and X. The steps followed in the algorithm are as follows.

- Fault Activation or creating a D-Frontier.
- Fault Effect propagation or driving D-Frontier towards the output.
- Find the suitable inputs justifying the Fault or Justifying J-Frontier.
- Backtracking if any conflict occurs at some node.

3.1 Create D-Frontier or Fault Activation

- A D-Frontier is a set of gates with D or D' at the inputs and X at the output.
- Given a fault we try to generate a Primitive D Cube which is used to specify minimum input conditions required at a gate to propagate D or D' to the output.
- Assign the values of Primitive D Cube to the gate inputs thereby generating a D-Frontier. This is the fault activation step and we get D or D' at the output of the gate after this.

3.2 Drive D-Frontier towards output or Fault Propagation

- After creating a D-Frontier we drive the D-Frontier towards output using Propagation D Cubes (PDCs).
- Propagation D Cube is the minimum gate input assignments required to propagate a D/D' from gate inputs to gate output.
- In case of fan-outs, choose a single path for propagation. If the fault effects cannot be propagated, backtrack to the last decision point and change choice.

3.3 Justify J-Frontier

- J-Frontier is a set of gates whose output value is assigned but the input values are not yet decided (X).
- Justification step involves assigning input values to the gates so that a consistent set of circuit input values that gives the required value at the fault site.

3.4 Backtracking at conflict

- Backtracking involves changing the choice at the last decision point if there is any inconsistency found with the assigned values.

3.5 Flow Chart of D-Algorithm

The flowchart can be seen for a better understanding and visualisation of the flow of the algorithm in Figure 1.

4 Algorithm Implementation

- The programming language used for this project is C++ with the help of STL library. The language was chosen because of it's speed and high level implementation, both of which are benefits and aid our construction.
- The main data structures used in implementing this project include Graphs, Arrays, Pairs and Vectors.

A circuit is built and is given as an input in a specific format.

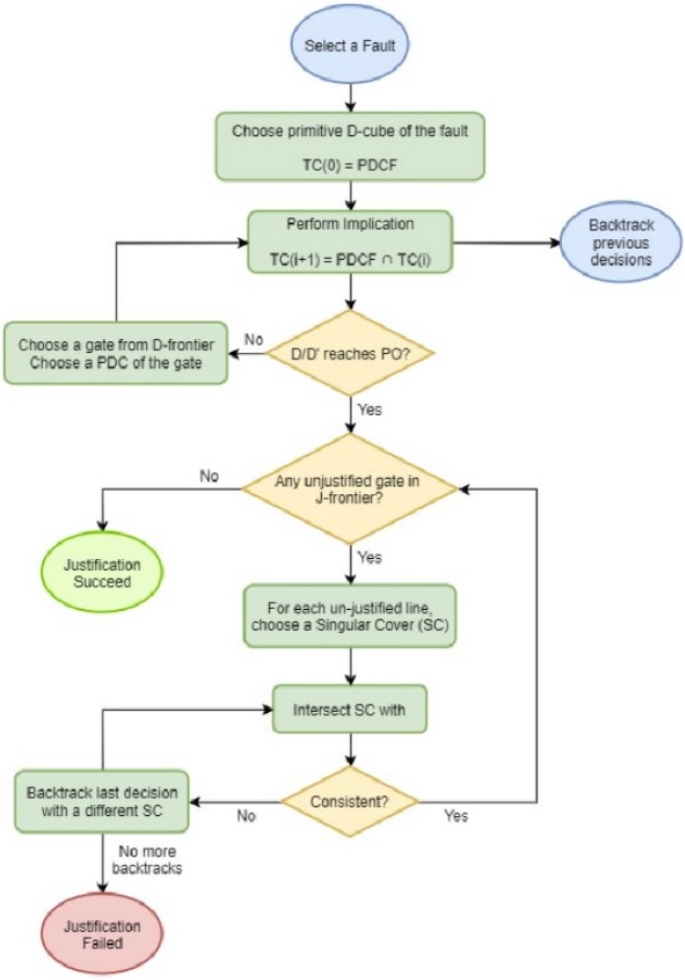

Figure 1: Flowchart describing general flow of D-Algorithm

4.1 Combinational circuit implementation

- The combinational circuit is represented with the help of graphs.
- Each node of the graph represents either a gate, primary inputs, primary output or a branchout.
- The gate, primary inputs, primary outputs and the branches are identified using a specific number which determines their functionality of the type of gate or the stuck-at-fault or fault free. Edges of the graph are wires, and are indexed starting from 50 to prevent conflict in naming.

The circuit is then simulated with all possible single-stuck-at faults and the test pattern is generated for each of them if possible.

4.2 Generating the test vector

The process of test vector generation follows the basic 4 steps described above and each is implemented with the help of the functions being dfrontier, jfunct, justify, D_algo.

- The dfrontier function returns the D frontiers when the input is given.
- The jfunct function which returns J frontiers.
- Then the justification along with back propagation happens in justify function.
- The D_algo function orders the above function in a sequence and returns the final result.

5 Results

6 Conclusions

7 Further Improvements

- Extension of D-Algorithm for sequential circuits.
- Addition of functionality for working on gates having more than 2 inputs.
