**PODEM algorithm for ATPG**

Problem Statement:

PODEM (Path-Oriented Decision Making) is an Automatic Test Pattern Generation (ATPG) algorithm. The aim of this project is to implement the PODEM algorithm to generate test vectors for a given fault. Given a circuit and a fault location this algorithm will tell us whether the fault can be detected and a combination of input values which can detect the fault.

The output is a test if one exists, or the indication that the fault is untestable.

Assumptions:

1. Single stuck-at fault.

2. Written code assuming only 4 types of gates in netlist but can be easily extended to other type of gates.

Input output descriptions:

netlist.txt is the input file having description of nodes in spice netlist format. The netlist we used in worked out example:

not1 n1 in1 and2\_1 n2 in2 in3 not2 n3 in4 nand2\_1 n4 in2 n1 nand2\_2 n5 in1 n2 nand2\_3 n6 n2 in4 nand2\_4 n7 n3 in3

nand4\_1 out1 n4 n5 n6 n7

in1,in2,in3 and in4 are the primary inputs. out1 is the primary output.**Our code can run for any such spice netlist (for any number of input and any number of output)**

Fault Location is the node where fault is to be checked ,values should be taken from the netlist file e.g. n1,n4. Fault Value is the complementary value of the stuck at fault.

“Fault is detected, Input vectors are:” will be the output in case given fault can be sensitized and propagated. “Fault is not detectable” will the output if either fault cannot be sensitized or cannot be propagated or both.

NCV = [1,0,1,0,1,-5,-5,1] matrix for non-controlling value for each gate type

Algorithm:

PODEM (Path-Oriented Decision Making) is an Automatic Test Pattern Generation

(ATPG) algorithm.

The algorithm works in two major step :

1. Sensitize the fault

2. Fault propagation

In the first step, appropriate PI (primary inputs) test are evaluated which excites the fault, that is, causes the complementary value to appear at the fault site

If this is found, then the circuit should be able to propagate the fault, for it to be detectable.

In the second step, nodes that are connected to the output gate are assigned non controlling values.

A sample circuit is chosen for verification purposes. Various subroutines of the PODEM

algorithm are individually verified and explained in the below. Generic steps followed in algorithm:

1. Apply fault excitation conditions.

2. Perform implications of last assignment

3. If the fault symptoms have reached atleast one primary output, justify the remaining unjustified lines. If justification fails, backtrace and go to Step 2.

4 Perform the resulting implications and go to step 2.

6.Once bactracing is completed, to propogate the fault, assign non-controlling values to nodes at output gate (by backtracing and assigning PIs appropriately)

In step 6, we always select a primary input (specifically for PODEM Algorithm). The process whereby we identify a suitable primary input, and, the value to assign to it,is called backtrace. The backtrace procedure is given an objective- an internal line and a desired value for it- and it traces a path backwards in the circuit until an input is found.

PODEM proves to be more efficient as compared to a D-ALG because it limits its search space only to Primary Inputs (PIs) of the circuits. D-ALG on the other hand has a search space comprising of all the internal nodes of the circuit along with the PIs.

The major modules of the MATLAB code are :

 podem

 main\_podem

 read\_netlist

 convert\_netlist

 create\_devicemap

 cnctd\_devices

 objective

 backtrace

 imply

 type

 type\_imply

 gates

A brief functionality of each module:

1. podem :

Global variable declaration. Input assignment and calling function main\_podem

2. main\_podem :

 Input netlist by calling “read\_netlist” and “convert\_netlist”

 Sensitize the fault

1. There are three major functions used here - objective, backtrace and imply.

Given a FaultLocation, FaultValue pair, it backtraces, and evaluates and updates values at each

node

 Fault propagation

1. Non-controlling value (NCV) is found for each node connected to output node.

2. objective is called to set each of these nodes to this value. If a set of PI s is

found which give NCV values at each of these nodes, „Fault is detected‟ is

displayed along with the set of PI s

3. read\_netlist and convert\_netlist

These functions read the spice netlist, store the input and output nodes in format: [ Device\_ID, Gate Type, Output node, Input Nodes ]

Device\_ID is an unique number assigned to each gate using node\_map function.

Gate Type is a number assigned to a particular type of gate as follows :

|  |  |  |  |
| --- | --- | --- | --- |
| **Type of gate** | **Gate type** | **Type of gate** | **Gate type** |
| NOT | 1 | NAND 2-input | 5 |
| OR 2-input | 2 | XOR 2-input | 6 |
| AND 2-input | 3 | XNOR 2-input | 7 |
| NOR 2-input | 4 | NAND 4- input | 8 |

\*Have used only 4 gates –not, and,nand,nand4 in the code.

4. objective:

Input to this function is a particular nG, nV where nV is the required value at the node nG. It calls backtrace function and backtraces this pair of (nG, nV) till a primary input is achieved and isPI flag is asserted. After each backtrace operation, value of nG, nV is updated to the next value in stack\_G.

5. backtrace:

Function is called inside objective. Takes an input pair (nG, nV). In every row of netlist it checks where this nG is an output node of a gate. For this row, according to type

of gate and nV, it evaluates the values of inputs of that gate using “type” function. These

values of inputs to the gate are stored in a stack\_G and stack\_V variable.

6. type:

Used in backtracing. Takes a row of format decive\_ID , gate type, output nodes values, input nodes values. Checks the type of gate and calls appropriate gate function (say nand\_out for gate type = 5 for nand gate) and returns the row with updated input values.

7. imply:

This function is used to perform a logic simulation of the circuit depending on the values of the Primary Inputs. Its inputs are (PI, PI\_Value) obtained from the objective function.

It searches for the gates in netlist where PI node is an input and stores them in device\_connected array. Then it calls “imply\_device” function which operates as: It calls “type\_imply” function which evaluates the output node values.

Also updates the node\_values array

This runs in a while loop which runs till all primary outputs are implied.

8. type\_imply:

Similar to type function but it is used in imply. According to the type, corresponding gates are called. Difference here is that inputs are known and output is to be implied. The updated row is

output from the function.

9. Gates:

 nand\_out:

Function for 2 input NAND gate. Inputs to this function are the two inputs and output of the row which is to be updated and g. g will be 0 for backtracing and g=1 for implying.

 and\_out:

Similar to nand\_out.

 nand\_4input:

Similar to nand\_out.

 not\_out:

Similar to nand\_out.

Worked out examples:

We have worked out the algorithm on the following circuit. Codes in MATLAB and python are in the code-and-demo folder.

Example:

netlist: Alphabets are shown here just to facilitate understanding of below diagram not1 n1(E) in1(A)

and2\_1 n2 (G) in2(B) in3(C)

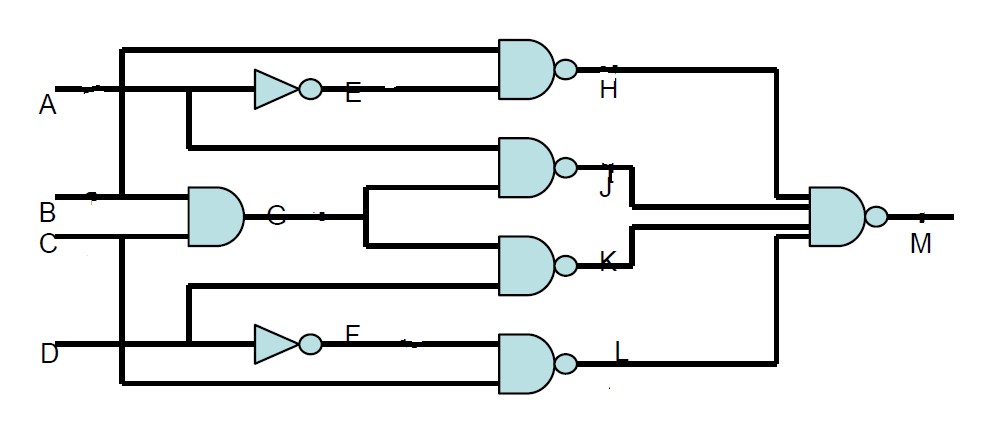
not2 n3(F) in4(D) nand2\_1 n4(H) in2 n1 nand2\_2 n5 (J) n1 n2

nand2\_3 n6 (K) n2 in4

nand2\_4 n7 (L) n3 in3 nand4\_1 out1(M) n4 n5 n6 n7

using node map function all nodes of netlist will be assigned a unique device\_iD as:

|  |  |  |
| --- | --- | --- |
| Device\_ID` | Nodes from Figure | Node of netlist |
| 1 | E | n1 |
| 2 | A | in1 |
| 3 | G | n2 |
| 4 | B | in2 |
| 5 | C | in3 |
| 6 | F | n3 |
| 7 | D | in4 |
| 8 | H | n4 |
| 9 | J | n5 |
| 10 | K | n6 |
| 11 | L | n7 |
| 12 | M | out1 |



1. Fault at node K ie stuck-at-1=> V =0:

Step 1: Set objective (K, 0) and backtrace(K, 0) which gives (G,1) and (D,1). D = 1 found by backtracing (till a primary input). Now call imply(D, 1)

D = 1 implies F = 0 implies L = 1 while other nodes are still unassigned

Step 2: Now check for value at fault location. It is still unassigned (value = -5), so continue with using the second option of backtrace (K, 0). i.e (G,1)(as stored in stack earlier) . backtrace (G,1) gives (B,1) and (C,1).

Imply with (B, 1) gives G = -5, K = -5

Step 3: Now imply with C = 1. => G =1, K=0. So fault is sensitized. Step 4: For propagation, backtrace(J, 1) => A =0, E =1, H= 0.

Step 5: Since H=0, fault cannot be propagated. So no test exists. And fault is not detectable

2 Fault at node K, stuck-at- 0 => V =1

Step 1: Set objective(K,1). backtrace(K,1) => D=0. imply(D,0) will give F=1, K=1.Fault at K is sensitized

Step 2: To propagate the fault, backtrace (H,1) => (E, 0) => (A,1) , (B,X)

imply(A,1) => no changes in node\_values. Step3: bactrace(J,1) => (G, 0) => (C ,0)

imply(C,0) => (J,1),(L,1)

Step 4: Since J, L and H, all are set to Non-controlling value of Nand4. Fault can be propagated.

Input Vector is (A, B, C, D) = (1, X, 0, 0)