ECE-6140 PROJECT
Part I: LOGIC SIMULATOR

DATA STRUCTURES USED

The data structures used to develop this logic simulator consists of **matrices**. The main functions assigned to the matrices are as follows:

- To store the imported netlist file.
- To store the imported input vector file.
- To store the computed values of nodes in netlist file.
- To store the resulting output vectors

ALGORITHM & PSEUDO CODE

The working of the Logic Simulator is based on computing the gates from the netlist sequentially, starting with the primary inputs.

The algorithm used here can be described in following steps:

- Read the netlist file into a matrix, mat.
- Duplicate the matrix, this is done so that the updated nodes can be mapped on the duplicated matrix
- Read the input vector file.
- For 1st test vector, assign all the primary input values in duplicated matrix, mat2. And propagate them to corresponding node values in mat2 by mapping the node numbers with mat.
- For gate computation, check sequentially by changing the row number of matrix if the 2nd column has input value initialized as 0 or 1
- If that is true then check for corresponding gate, if it is a 2 input gate then check if 3rd column (2nd input) is already initialized.
- If true then, check if 4th column (output) is already computed as 0 or 1. If this is false, then compute the output and store it in 4th column of the corresponding row.
- If it is an 1 input gate then check the 3rd column if output is already initialized. If false then store the computed output in 3rd column of corresponding node.
- Then update that node value at all the same node numbers in mat2.
- Repeat this process till all gates are evaluated.

PSEUDO CODE

On the next page a pseudo code for gate evaluation and node update is presented for both 1 and 2 input gates

```
%%%%%%%%%%%%%%%%%%%%GATE EVALUATION%%%%%%%%%%%%%%%%%%%%%%%%
count=0
while count~=size(mat,1)-2
    for m=1:size(mat,1)-2

        if ((mat2{m,2})==0 || (mat2{m,2})==1)    %%%CHECKING IF 1ST INPUT IS
ASSIGNED


            if (strcmp(mat2(m,1),'AND')==1) %%%CHECKING AND GATE
                if ((mat2{m,3})==0 || (mat2{m,3})==1) %%%CHECKING IF 2ND INPUT
IS ASSIGNED
                    if ((mat2{m,4})~=0 && (mat2{m,4})~=1) %%%CHECKING IF OUTPUT
IS ALREADY EVALUATED
                        mat2{m,4}=and((mat2{m,2}),(mat2{m,3}));%%%EVALUATING
OUTPUT
                        count=count+1
                        % %
                        for j2=1:size(mat,1)-2%   NODE UPDATE
                            for j1=2:4%
                                if strcmp(mat(m,4),mat(j2,j1))==1
                                    mat2{j2,j1}= (mat2{m,4});
                                end
                            end
                        end
                        % %
                    end
                end

            elseif (strcmp(mat2(m,1),'INV')==1)%%%CHECKING NOT GATE
                if ((mat2{m,3})~=0 && (mat2{m,3})~=1)%%%CHECKING IF OUTPUT IS
ALREADY EVALUATED
                    mat2{m,3}=~((mat2{m,2})) ;                    %%%EVALUATING
OUTPUT
                    count=count+1
                    for j2=1:size(mat,1)-2%      NODE UPDATE
                        for j1=2:4%
                            if strcmp(mat(m,3),mat(j2,j1))==1
                                mat2{j2,j1}= (mat2{m,3});
                            end
                        end
                    end

                end
        end

    end

end
```

SIMULATION DATA

CIRCUIT 1: s27.chat

INPUTS	OUTPUTS
1110101	1001
0001010	0110
1010101	1001
0110111	0001
1010001	1001

CIRCUIT 2: s298f_2.chat

INPUTS	OUTPUTS
10101010101010101	00000010101000111000
01011110000000111	00000000011000001000
11111000001111000	00000000001111010010
11100001110001100	00000000100100100101
01111011110000000	11111011110000101101

CIRCUIT 3: s344f_2.chat

INPUTS	OUTPUTS
101010101010101011111111	101010101010101010101101
010111100000001110000000	000111100000001000001111100
111110000011110001111111	000111000001110110001111010
111000011100011000000000	000011011110011111111000010
011110111100000001111111	100111011110000001001000100

CIRCUIT 4: s349f_2.chat

INPUTS	OUTPUTS
101010101010101011111111	101010101010101101010101
010111100000001110000000	000111100000001010111110000
111110000011110001111111	000111000001110100011111100
111000011100011000000000	000011011110011100100001111
011110111100000001111111	100111011110000001010000100

ECE-6140 PROJECT
Part II: DEDUCTIVE FAULT SIMULATOR

DATA STRUCTURES USED

The data structures used to develop this logic simulator consists of **matrices**. The main functions assigned to the matrices are as follows:

- To store the imported netlist file.
- To store the imported input vector file.
- To store the computed values of nodes in netlist file.
- To store the resulting output vectors

ALGORITHM & PSEUDO CODE

The working of the Deductive Fault Simulator is based on computing the fault lists at all nodes and propagate the list to the primary outputs. So in this project we just added the fault list generation algorithm and propagation algorithm to the existing logic simulator made in the previous project.

The algorithm used here can be described in following steps:

- Create an empty cell named ‘list’ of dimensions 1x(#nodes) to store fault lists of all the nodes.
- While assigning test vectors to primary inputs in mat2, add the faults in the lists of primary inputs to be stuck at the complimented value of the test vector we are inputting at the corresponding node.
- Then after each gate evaluation stage done similar to the previous project, list of the output of the gate can be evaluated using the list generation algorithm mentioned as a pseudo code.
- Repeat this process till lists for all the primary outputs is evaluated.
- Taking union of all these lists of primary outputs gives us the final detected fault list.
- To calculate the fault coverage and detected faults for random test vectors, the whole above operation can be repeated under a for loop, checking for each test vector and accumulating the cumulative detected fault list.
- The number of random test vectors to be generated will be a user input, and are generated using ‘rand’ function of matlab.
- For calculating fault coverage, number of faults detected by a test vector is divided by the total number of faults possible (#nodesx2).
- Finally we plot a graph for fault coverage vs number of random tests.

PSEUDO CODE

Below the pseudo codes for list generation, calculation of detected fault list and fault coverage is mentioned.

```
%%%LIST GENERATION FOR AND GATE%%%
if (mat2{m,2}==0 && mat2{m,3}==0)
list{mat4{m,4}}=union((intersect(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 1'));
elseif (mat2{m,2}==0 && mat2{m,3}==1)
list{mat4{m,4}}=union((setdiff(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 1'));
elseif (mat2{m,2}==1 && mat2{m,3}==0)
list{mat4{m,4}}=union((setdiff(list{mat4{m,3}},list{mat4{m,2}})),strcat(mat{m,4},' stuck at 1'));
elseif (mat2{m,2}==1 && mat2{m,3}==1)
list{mat4{m,4}}=union((union(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 0'));
end
%%%%LIST GENERATION FOR OR GATE%%%
if (mat2{m,2}==0 && mat2{m,3}==0)
list{mat4{m,4}}=union((union(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 1'));
elseif (mat2{m,2}==0 && mat2{m,3}==1)
list{mat4{m,4}}=union((setdiff(list{mat4{m,3}},list{mat4{m,2}})),strcat(mat{m,4},' stuck at 0'));
elseif (mat2{m,2}==1 && mat2{m,3}==0)
list{mat4{m,4}}=union((setdiff(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 0'));
elseif (mat2{m,2}==1 && mat2{m,3}==1)
list{mat4{m,4}}=union((intersect(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 0'));
end
%%%%LIST GENERATION FOR INVERTER%%%%
if (mat2{m,2}==0)
list{mat4{m,3}}=union(list{mat4{m,2}},strcat(mat{m,3},' stuck at 0'));
elseif (mat2{m,2}==1)
list{mat4{m,3}}=union(list{mat4{m,2}},strcat(mat{m,3},' stuck at 1'));
end
%%%%LIST GENERATION FOR NAND GATE%%%%
if (mat2{m,2}==0 && mat2{m,3}==0)
list{mat4{m,4}}=union((intersect(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 0'));
elseif (mat2{m,2}==0 && mat2{m,3}==1)
list{mat4{m,4}}=union((setdiff(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 0'));
elseif (mat2{m,2}==1 && mat2{m,3}==0)
list{mat4{m,4}}=union((setdiff(list{mat4{m,3}},list{mat4{m,2}})),strcat(mat{m,4},' stuck at 0'));
elseif (mat2{m,2}==1 && mat2{m,3}==1)
list{mat4{m,4}}=union((union(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 1'));
end
%%%%LIST GENERATION FOR NOR GATE%%%%
if (mat2{m,2}==0 && mat2{m,3}==0)
list{mat4{m,4}}=union((union(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 0'));
elseif (mat2{m,2}==0 && mat2{m,3}==1)
list{mat4{m,4}}=union((setdiff(list{mat4{m,3}},list{mat4{m,2}})),strcat(mat{m,4},' stuck at 1'));
elseif (mat2{m,2}==1 && mat2{m,3}==0)
list{mat4{m,4}}=union((setdiff(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 1'));
elseif (mat2{m,2}==1 && mat2{m,3}==1)
list{mat4{m,4}}=union((intersect(list{mat4{m,2}},list{mat4{m,3}})),strcat(mat{m,4},' stuck at 1'));
end
%%%%LIST GENERATION FOR BUFFER%%%%
if (mat2{m,2}==0)
list{mat4{m,3}}=union(list{mat4{m,2}},strcat(mat{m,3},' stuck at 1'));
elseif (mat2{m,2}==1)
list{mat4{m,3}}=union(list{mat4{m,2}},strcat(mat{m,3},' stuck at 0'));
end
%%%%%%%CALCULATING DETECTED FAULT LIST%%%%%%%%%%%%%%%%
DF=[];
for k=2:out_length-1
    DF=(union(DF,list{str2double(mat4{size(mat2,1),k})})),DF1);
End
%%%%%%%CALCULATING FAULT COVERAGE%%%%%%%%%%%%%%%%
fault_coverage=max(size(DF1))/size(UF,1)*100;
```

SIMULATION DATA

PART (a):-

CIRCUIT 1: s27.chat

Faults detected for TV: 1110101

1 stuck at 0
3 stuck at 0
5 stuck at 0
7 stuck at 0
9 stuck at 1
11 stuck at 1
12 stuck at 0
13 stuck at 0

CIRCUIT 2: s298f_2.chat

Faults detected for TV: 10101010101010101

3 stuck at 0
5 stuck at 0
6 stuck at 1
7 stuck at 0
8 stuck at 1
9 stuck at 0
10 stuck at 1
11 stuck at 0
12 stuck at 1
15 stuck at 0
18 stuck at 1
19 stuck at 1
20 stuck at 1
21 stuck at 1
22 stuck at 1
23 stuck at 1
24 stuck at 0
25 stuck at 1
26 stuck at 0
27 stuck at 1
28 stuck at 0
29 stuck at 1
30 stuck at 1
31 stuck at 1
32 stuck at 0

33 stuck at 0
34 stuck at 0
35 stuck at 1
36 stuck at 1
37 stuck at 1
39 stuck at 1
41 stuck at 1
45 stuck at 1
48 stuck at 0
49 stuck at 0
51 stuck at 1
52 stuck at 1
53 stuck at 1
54 stuck at 0
56 stuck at 1
57 stuck at 1
58 stuck at 0
64 stuck at 1
66 stuck at 0
67 stuck at 1
68 stuck at 1
95 stuck at 0
102 stuck at 0
103 stuck at 1
105 stuck at 1
106 stuck at 1
107 stuck at 1
108 stuck at 1
109 stuck at 1
110 stuck at 1
115 stuck at 1
116 stuck at 0
117 stuck at 0
118 stuck at 1
119 stuck at 1
120 stuck at 1
122 stuck at 0
132 stuck at 0
133 stuck at 0
135 stuck at 0
138 stuck at 0
141 stuck at 0
142 stuck at 0
143 stuck at 0

145 stuck at 0
146 stuck at 0
163 stuck at 1
164 stuck at 1
166 stuck at 0
168 stuck at 0
169 stuck at 1
170 stuck at 1
173 stuck at 0
182 stuck at 1
183 stuck at 1
186 stuck at 1
188 stuck at 0

CIRCUIT 3: s344f_2.chat

Faults detected for TV: 10101010101010101011111111

1 stuck at 0
2 stuck at 1
3 stuck at 0
4 stuck at 1
5 stuck at 0
6 stuck at 1
7 stuck at 0
8 stuck at 1
9 stuck at 0
10 stuck at 1
11 stuck at 0
12 stuck at 1
13 stuck at 0
14 stuck at 1
15 stuck at 0
16 stuck at 1
25 stuck at 0
26 stuck at 1
27 stuck at 0
28 stuck at 1
29 stuck at 0
30 stuck at 1
31 stuck at 0
32 stuck at 1
33 stuck at 0
34 stuck at 1
35 stuck at 0

36 stuck at 1
37 stuck at 0
38 stuck at 1
39 stuck at 0
40 stuck at 1
41 stuck at 0
42 stuck at 1
43 stuck at 0
44 stuck at 1
45 stuck at 0
46 stuck at 1
47 stuck at 0
48 stuck at 0
49 stuck at 1
50 stuck at 0
51 stuck at 1
52 stuck at 0
53 stuck at 0
54 stuck at 0
55 stuck at 1
56 stuck at 0
57 stuck at 1
58 stuck at 0
60 stuck at 0
61 stuck at 1
62 stuck at 1
64 stuck at 0
65 stuck at 1
66 stuck at 1
67 stuck at 0
68 stuck at 1
69 stuck at 0
70 stuck at 1
71 stuck at 1
72 stuck at 1
73 stuck at 1
76 stuck at 1
77 stuck at 1
78 stuck at 1
91 stuck at 0
92 stuck at 1
95 stuck at 1
96 stuck at 1
97 stuck at 0

99 stuck at 1
100 stuck at 1
101 stuck at 0
105 stuck at 1
106 stuck at 0
108 stuck at 0
109 stuck at 1
110 stuck at 1
111 stuck at 1
112 stuck at 0
114 stuck at 0
115 stuck at 1
116 stuck at 1
129 stuck at 1
130 stuck at 1
137 stuck at 0
139 stuck at 0
141 stuck at 1
142 stuck at 1
144 stuck at 0
146 stuck at 0
176 stuck at 0
177 stuck at 0
179 stuck at 1
180 stuck at 1
181 stuck at 0
183 stuck at 0
188 stuck at 1
189 stuck at 1
190 stuck at 1

CIRCUIT 4: s349f_2.chat

Faults detected for TV: 101010101010101011111111

1 stuck at 0
2 stuck at 1
3 stuck at 0
4 stuck at 1
5 stuck at 0
6 stuck at 1
7 stuck at 0
8 stuck at 1
9 stuck at 0
10 stuck at 1
11 stuck at 0

12 stuck at 1
13 stuck at 0
14 stuck at 1
15 stuck at 0
16 stuck at 1
25 stuck at 0
26 stuck at 1
27 stuck at 0
28 stuck at 1
29 stuck at 0
30 stuck at 1
31 stuck at 0
32 stuck at 1
33 stuck at 0
34 stuck at 1
35 stuck at 0
36 stuck at 1
37 stuck at 0
38 stuck at 1
39 stuck at 0
40 stuck at 1
41 stuck at 0
42 stuck at 0
43 stuck at 1
44 stuck at 0
45 stuck at 1
46 stuck at 0
47 stuck at 1
48 stuck at 0
49 stuck at 1
50 stuck at 0
51 stuck at 1
52 stuck at 0
53 stuck at 1
54 stuck at 1
55 stuck at 0
56 stuck at 1
57 stuck at 0
58 stuck at 0
60 stuck at 1
61 stuck at 1
62 stuck at 0
64 stuck at 1
65 stuck at 1
66 stuck at 0
67 stuck at 1
68 stuck at 0
69 stuck at 1

70 stuck at 1
71 stuck at 1
72 stuck at 1
73 stuck at 1
74 stuck at 1
109 stuck at 0
110 stuck at 1
111 stuck at 1
113 stuck at 0
114 stuck at 1
115 stuck at 1
116 stuck at 1
118 stuck at 1
120 stuck at 0
121 stuck at 0
122 stuck at 1
123 stuck at 1
124 stuck at 1
126 stuck at 0
127 stuck at 0
128 stuck at 1
129 stuck at 1
130 stuck at 1
131 stuck at 1
133 stuck at 0
134 stuck at 1
135 stuck at 1
137 stuck at 0
138 stuck at 1
171 stuck at 0
173 stuck at 0
174 stuck at 1
176 stuck at 0
177 stuck at 1
178 stuck at 1
179 stuck at 0
180 stuck at 0
181 stuck at 0
183 stuck at 0
187 stuck at 1
188 stuck at 1
189 stuck at 1

PART (b) :-

CIRCUIT 1: s27.chat

Number of random test vectors applied = 30
Number of vectors needed to achieve more than 75% fault coverage = 6
Number of vectors needed to achieve more than 90% fault coverage = 18

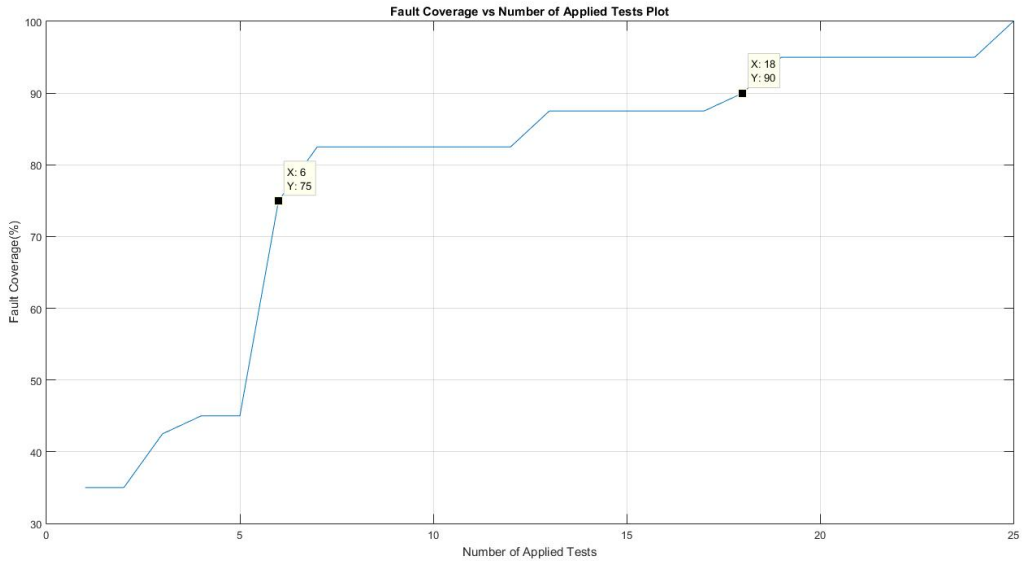

Fig. 1: Fault coverage vs Number of applied tests for s27.chat

CIRCUIT 2: s298f_2.chat

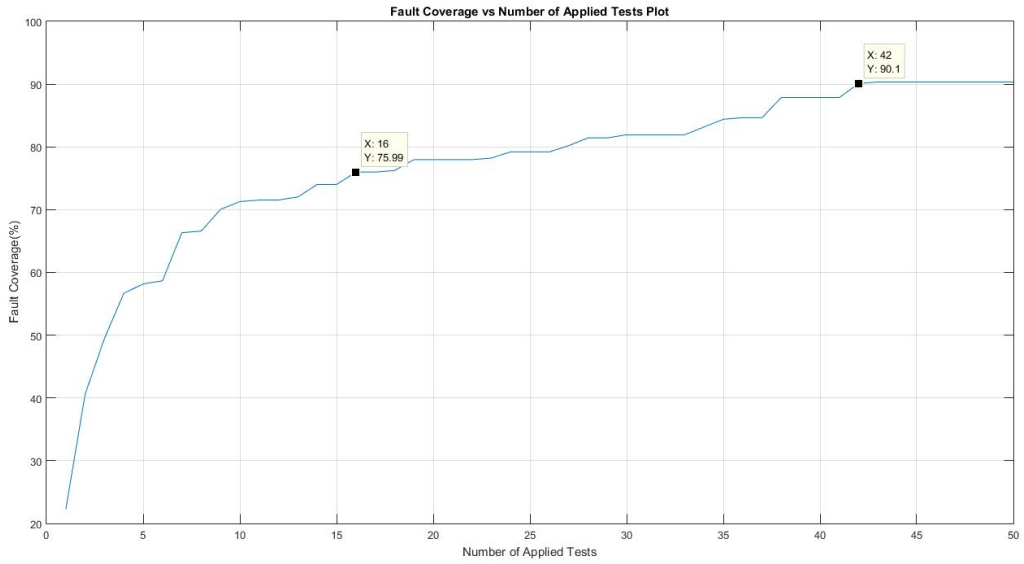

Fig. 2: Fault coverage vs Number of applied tests for s298f_2.chat

Number of random test vectors applied = 50
Number of vectors needed to achieve more than 75% fault coverage = 16
Number of vectors needed to achieve more than 90% fault coverage = 42

CIRCUIT 3: s344f_2.chat

Number of random test vectors applied = 100
Number of vectors needed to achieve more than 75% fault coverage = 7
Number of vectors needed to achieve more than 90% fault coverage = 21

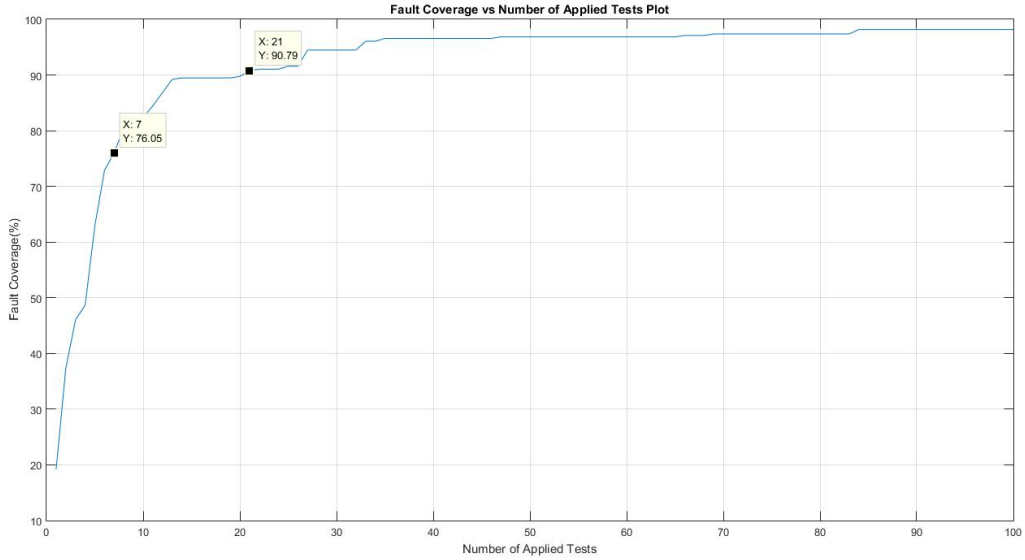

Fig. 3: Fault coverage vs Number of applied tests for s344f_2.chat

CIRCUIT 4: s349f_2.chat

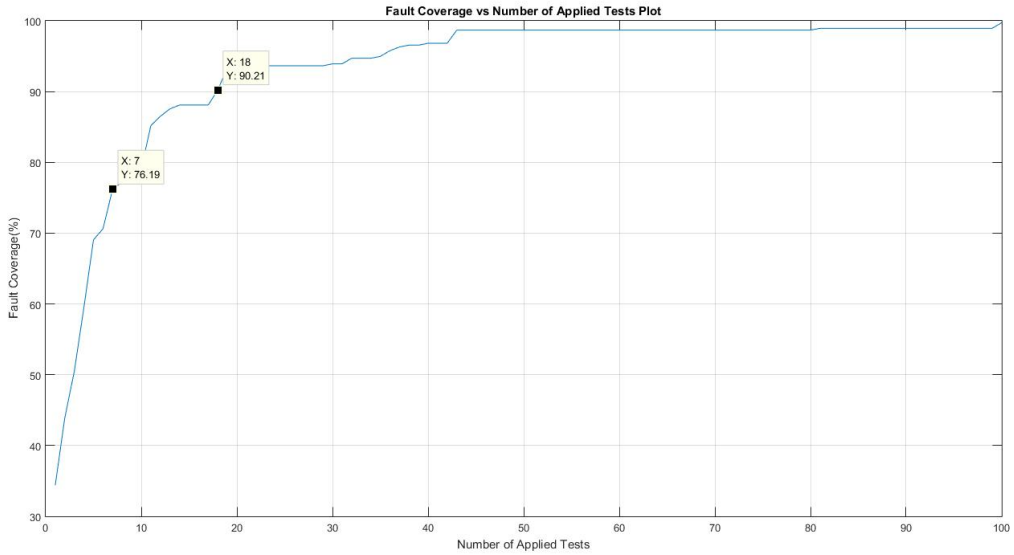

Fig. 4: Fault coverage vs Number of applied tests for s349f_2.chat

Number of random test vectors applied = 100
Number of vectors needed to achieve more than 75% fault coverage = 7
Number of vectors needed to achieve more than 90% fault coverage = 18

ECE-6140 PROJECT
Part III: PODEM ALGORITHM

DATA STRUCTURES USED

The data structures used to develop this logic simulator consists of **matrices**. The main functions assigned to the matrices are as follows:

- To store the imported netlist file.
- To store the imported input vector file.
- To store the computed values of nodes in netlist file.
- To store the resulting output vectors

ALGORITHM & PSEUDO CODE

Podem is an ATPG algorithm that implements a direct search process. The decisions made using Podem consist of only primary inputs (PI) assignments. The main operation of Podem is explained using four important functions mentioned below,

- **Objective:** Its main job is to assign appropriate value to sensitize the fault, and from there on to assign appropriate values to other inputs of gate to propagate the sensitized fault to the output.
- **Backtrace:** This function is called right after the objective function to set appropriate values to the primary inputs to satisfy the objective set earlier.
- **Imply:** This function is called after the backtrcae function to evaluate the implications of setting the values to primary input in the backtrace stage. All the gate evaluations done are 5 valued evaluations i.e. the gate inputs and outputs can be any of x, d, d_bar, 0 or 1. Also D Frontier is constantly updated after every time imply function is called.
- **X-Path Check:** This function is always called after imply and is checked only if the fault is sensitized. The main aim of this function is to make sure that the fault can be propagated to the output by checking if any of the paths from output of any of the gates from D-Frontier to any of the primary outputs contain all the values as x.

Apart from these four functions if x path check fails once, we reverse the decision for the primary input assignment in the backtrace operation and call Podem main function again recursively and assign appropriate value to some other primary input to satisfy the same objective.

If even then x path check fails, we set the primary input to x and call podem again recursively, if this time x path check fails, we can say that the fault is redundant and no test vector is found as all the possible paths and input combinations have been checked for fault propagation.

But if the fault reaches any of the primary outputs at any given stage, we can say that podem is successful and generate the test vector. The generated test vector can be checked using a fault simulator.

Pseudo Code

Below the pseudo codes/ shell codes for the podem main, objective and backtrace functions are mentioned based on which my code was written

```
PODEM()
begin
if (error at PO) then return SUCCESS // error is d or d_bar
if (test not possible) then return FAILURE //x path check failed
(k, v_k) = Objective() // selects gates from D-frontier
(j, v_j) = Backtrace(k, vk) // j is a PI
Imply(j, v_j)
x_check=x_path_check() //set x_check as '1' if x path is available, else '0'
if PODEM() = SUCCESS then return SUCCESS
Imply(j, v̄_j) // reverse decision
x_check=x_path_check()
if PODEM() = SUCCESS then return SUCCESS
Imply(j, x)
return FAILURE
end
```

```
Objective()
begin// target fault is f stuck at v
if (the value of f is x) then return (f, v̄)
select a gate (G) from the D-frontier
select an input (j) of G with value x
c = controlling value of G
return (j, c̄)
end
```

```
Backtrace(k, v_k) // map objective into PI assignment
begin
v = v_k
while (k is a gate output)
begin
i = inversion of k
select an input (j) of k with value x
v = v XOR i
k = j
end
 // if here, then k is a PI
return (k, v)
end
```

SIMULATION DATA

CIRCUIT 1: s27.chat

Target Fault	Test Vector generated by PODEM algorithm
16 stuck at 0	X0X10X0
10 stuck at 1	X00XXX0
12 stuck at 0	1XXX1XX
18 stuck at 1	11X101X

CIRCUIT 2: s298f_2.chat

Target Fault	Test Vector generated by PODEM algorithm
70 stuck at 1	01X1XXXXXXXXX0XX
73 stuck at 0	111XXXXXXXXXX0XX
26 stuck at 1	XX1X1XXX0XXXXXXX
92 stuck at 0	X10101XXXXX0X0XX

CIRCUIT 3: s344f_2.chat

Target Fault	Test Vector generated by PODEM algorithm
166 stuck at 0	01X00XXXX011XX0XXXXXXX
71 stuck at 1	10XXXXXXXXXXXXXXXXXXXX
16 stuck at 0	10XXXXXXXXXXXX1XXXXXXX
91 stuck at 1	111XXXXXXXXXXXXXXXXXX

CIRCUIT 4: s349f_2.chat

Target Fault	Test Vector generated by PODEM algorithm
25 stuck at 1	XXXXXXXXXXXXXX1XXXXXXX
51 stuck at 0	00XXXXXXXXXXXX0XXXXXXX
105 stuck at 1	01X1000XXX01XX10XXXXXXX
7 stuck at 0	XXXXXX1XXXXXXXXXXXXXX

*Fault 179 stuck at 1 is **undetectable** for circuit 4.