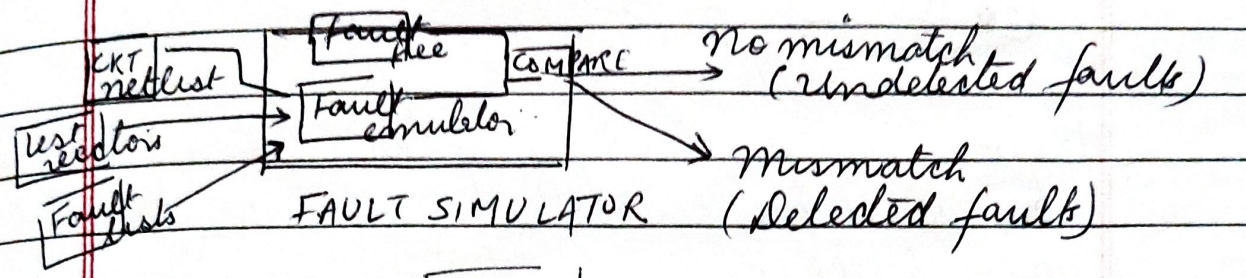
n of random test pattern

Date
Page

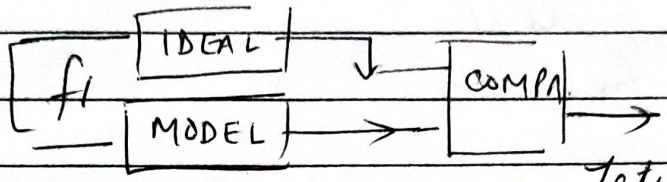
ALGORITHMS FOR FAULT SIMULATION

→ F.C.

→ Fault dropping



e.g.



vector I_1, I_2, I_3

Lets say I_1 detects f_1 then I_2 & I_3 are dropped

This is called fault dropping

This is required to reduce simulation ~~speed~~ time.

Ex: if n faults are there

→ n simulations reqd.

To reduce this simulation time, diff approaches have been proposed.

1) SERIAL FAULT SIMULATION.

- simplest, but time is high
- based on use of true-value simulator repeatedly
- first fault free o/p is recorded & next for each fault model is simulated & compared.

able to handle all type of fault model

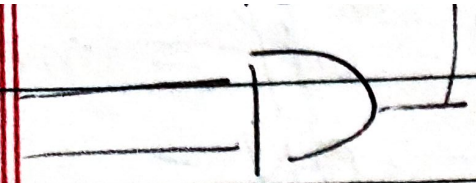
or $\sum_{i=1}^{n} t \times n$ sec. (BEST)

- capable of detecting all faults.

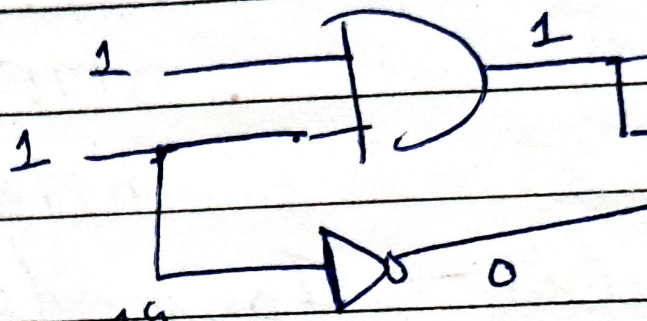
$T = n \times t \times n_{vectors}$ (WORST) no fault drop

2) PARALLEL FAULT SIMULATION.

- meant for uniform condition
- exploits bit-parallelism of logical operation
- to speed up.



Normal
$O_1 = 1$
$O_2 = 0$



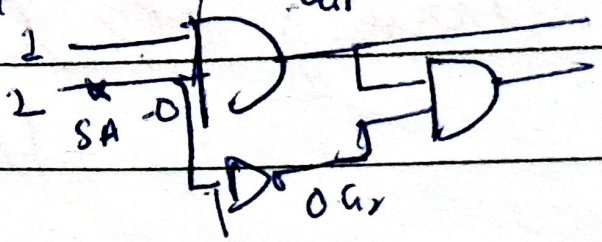
$t=0$, $A=1$, $B=1$,

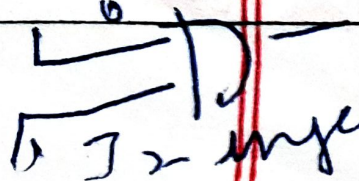
$G_1 = '$

G_2 O_{G_1}

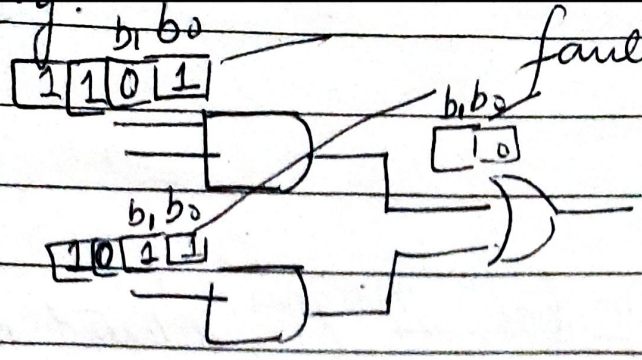
G_3

injecting fault





J = injecting fault



Assume same delay of gates.

b_0 - fault free

b_1 - fault f1

b_2 - fault f2.

→ p - no of test vectors (Ntvec)

f - n " faults

n - logic gates (ckt size)

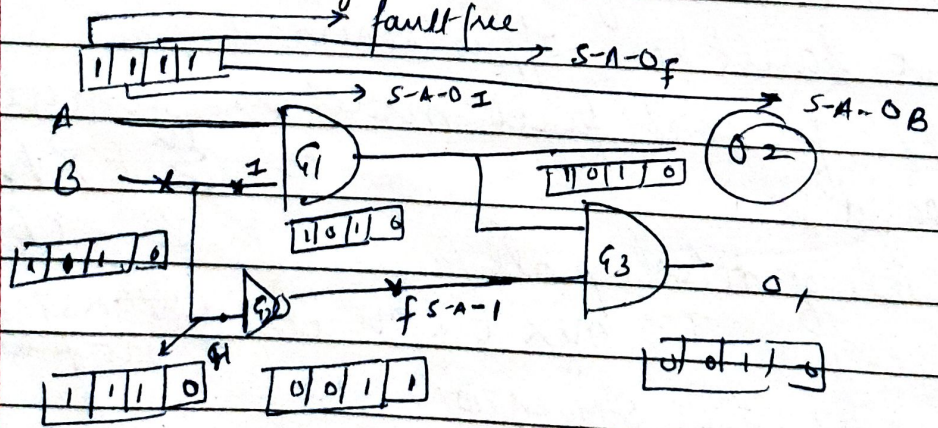
$f \propto n$

Complexity $p \times f \times n$

$\Rightarrow O(pn^2)$

uses m/c word size

→ W is word size W-1 // el simulation use logic.

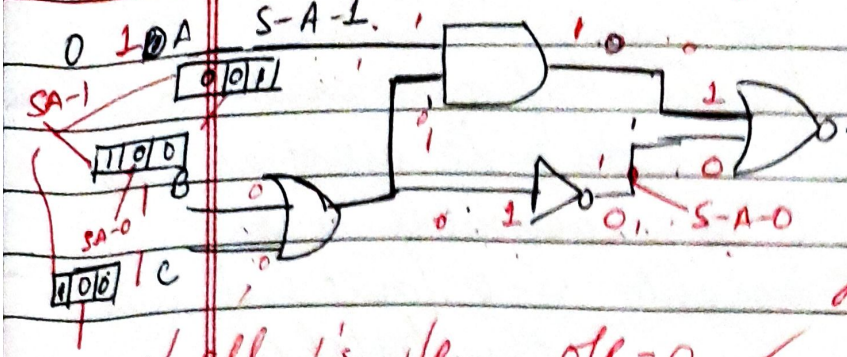


if we observe (O2) it detects faults. S-A-0 I ≡ S-A-0 B

011 – S-A-1 o/p = 1/0 Exp

000 – S-A-0 o/p = 0/1

111 – good response



all 1

if all 1's i/p o/p = 0

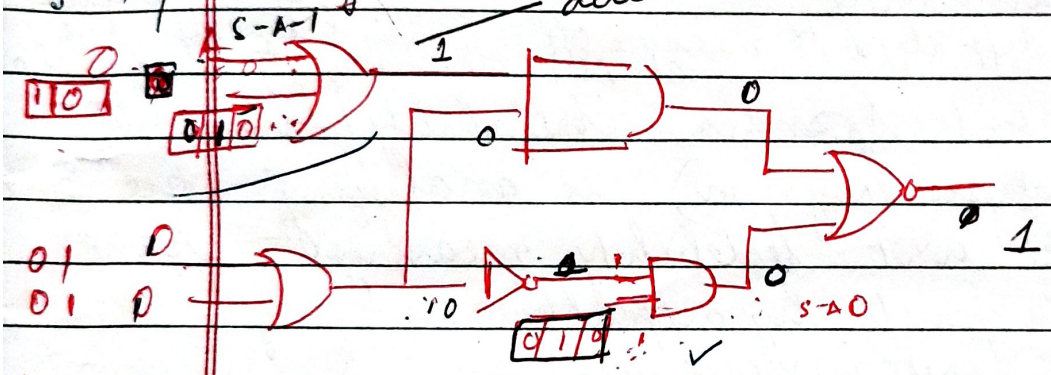
if A is S-A-1 then o/p

Considering S-A-0 fault at 'j' o/p = depends on other i/p

what is i/p at 'A' ??

This requires fault injection

∴ fault injection



fault

A B C

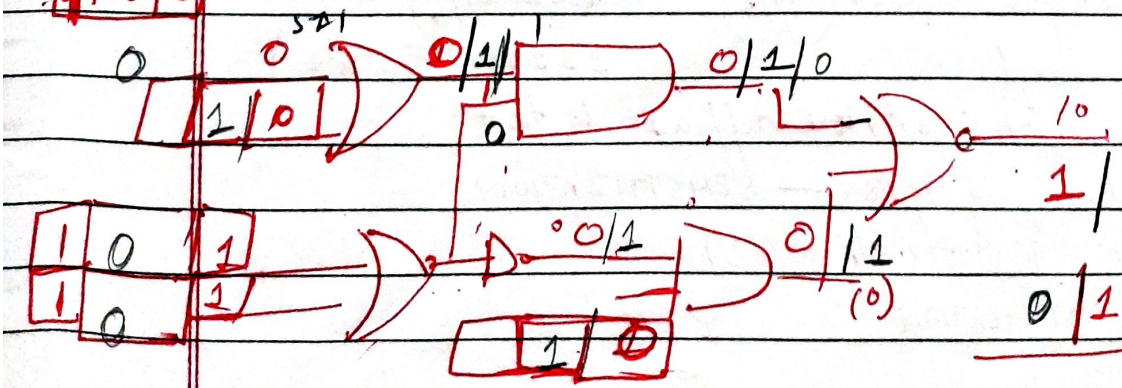
1 0 0

fault free

gives unexpected.

actual o/p Expected o/p

0 1 1



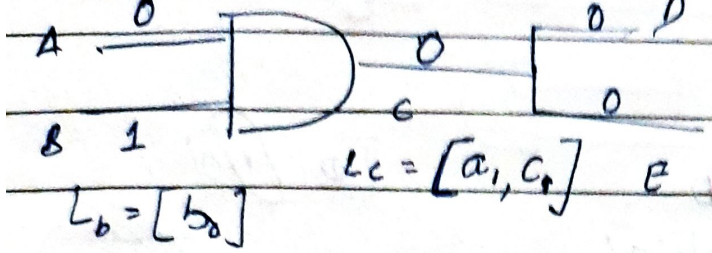
– works both for compiled code & event driven.

Deductive fault Simulation:

deduced from fault free ckt.

as 'a' is '0' so only s-a-1 fault possible for the i/p

$L_a = [a_1]$



$L_b = [b_0]$ $\quad L_c = [a_1, c_0]$ & $L_e = [a_1, c_1, e_1]$

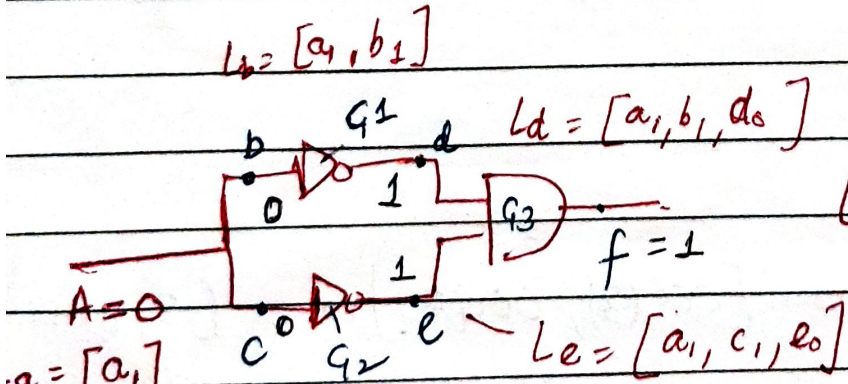
AND

a	b	c	L_c	
0	0	0	$[L_a \cap L_b] \cup C_1$	faults.
0	1	0	$[L_a \cap \bar{L}_b] \cup C_1$	
1	0	0	$[\bar{L}_a \cap L_b] \cup C_1$	
1	1	1	$[L_a \cup L_b] \cup C_0$	

C_1 - C - SA1

C_0 - C - SA0

Total fault list = $[a_1, b_0, c_1]$

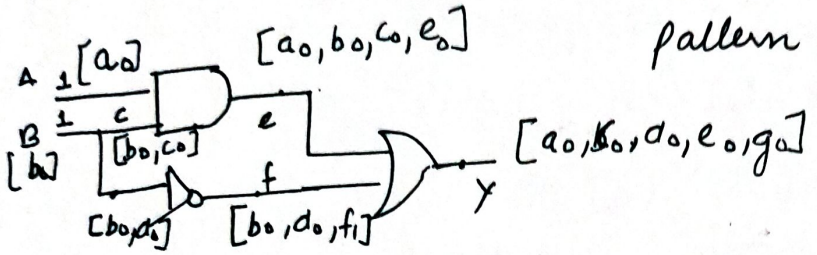


black (fault free)

$[a_1, b_1, c_1, d_0, e_0, f_0]$

DEDUCTIVE FAULT SIMULATION:

- based on deducing the possible fault list from a fault free ckt. using a random pattern.
- based on structure of the ckt. which is same for all faults & hence diff. faults deduction can be carried out simultaneously.



pattern applied 11 Expected o/p Y=1

$L_a = [a_0]$ $L_b = [b_0]$

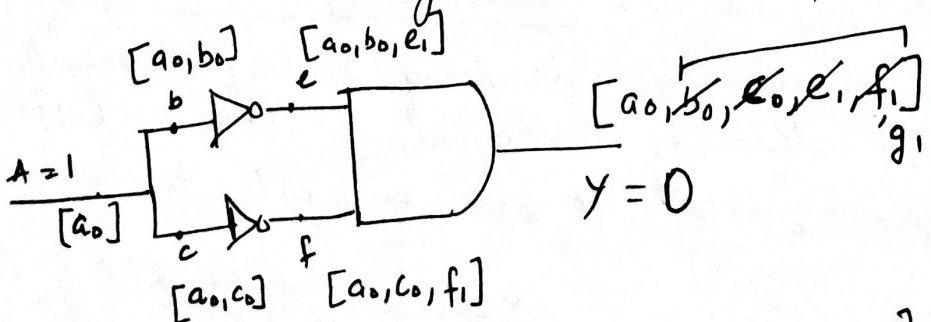
$L_e = [a_0, b_0, c_0, e_0]$ $\left[\because L_e = (L_a \cup L_c) \cup e_0 \right]$

$L_f = [b_0, d_0] \cup f_1$

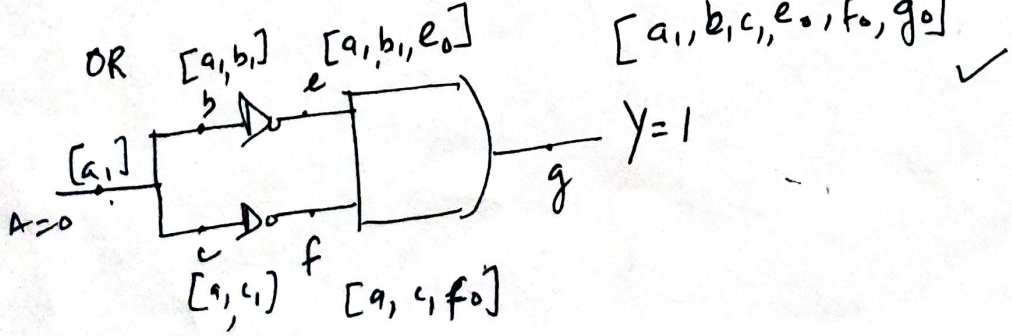
$L_g = [L_e \cap \bar{L}_f] \cup g_0$

↓

b_0 dropped as it will invert at one I/P & give expected o/p



$\therefore [a_0, g_1]$



$[a_1, b_1, c_1, e_0, f_0, g_0]$ ✓

∴ in general

AND

A	B	C	FAULT LIST
0	0	0	$(L_a \cap L_b) \cup C_1$
0	1	0	$(L_a \cap L_b) \cup C_1$
1	0	0	$(L_a \cap L_b) \cup C_1$
1	1	1	$(L_a \cup L_b) \cup C_0$

OR

A	B	C	
0	0	0	$(L_a \cup L_b) \cup C_1$
0	1	1	$(L_a \cap L_b) \cup C_0$
1	0	1	$(L_a \cap L_b) \cup C_0$
1	1	1	$(L_a \cap L_b) \cup C_0$

NOT

A	C	
0	1	$L_a \cup C_0$
1	0	$L_a \cup C_1$

→ fast but need to identify unknown initial state which makes it complex. ∴ (1, 0, X)

Secondly for seq. ckts., race conditions with 'X' may be identified as potentially detectable fault.

This can be solved by inserting delay elements. until total delay > delay of combinational logic before being propagated to sequential I/P. (This applies to asynchronous ckts)

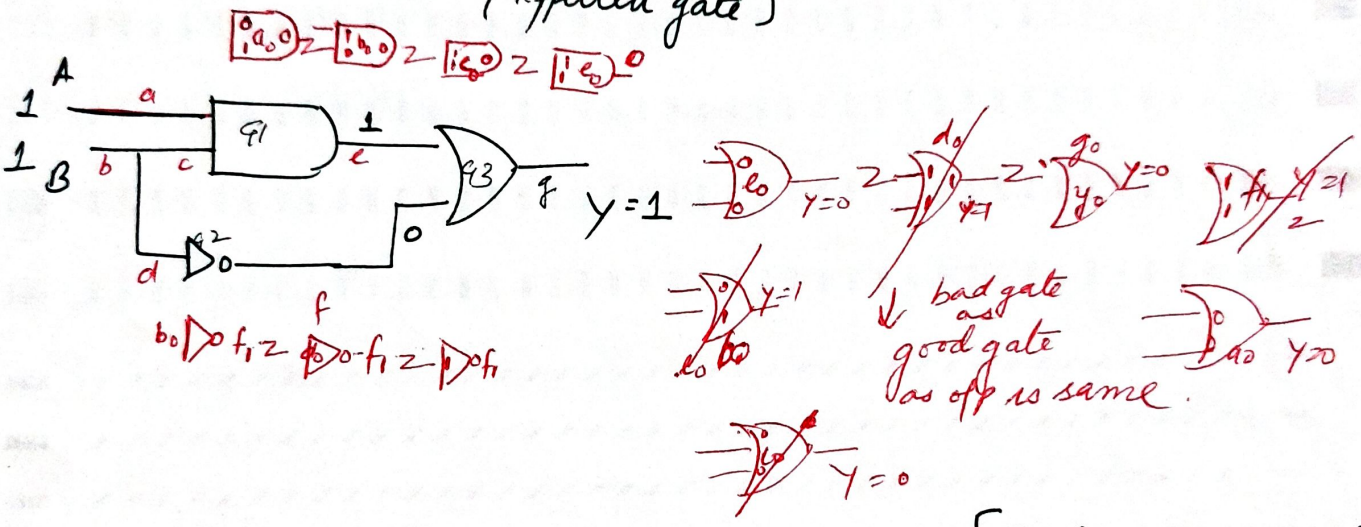
- can be compiled code / event driven ⇒ only gate whose I/P changes entire ckt needs to be responds simulated when I/P pattern changes.

CONCURRENT FAULT SIMULATION :

Similar to deductive fault simulation but retains info when moving from one pattern to another.
- biased on event-driven simulation, not compile code simulation.

EVENT → good event
→ fault event

STRUCTURE : ckt is flattened
→ GOOD GATE
→ FAULT LIST
→ BAD GATE (affected gate)



$\therefore [a_0, c_0, e_0, b_0, f_1, g_0, d_0]$

In actual 4 gates multi-ening o/p but no fault dropping.

Now consider 1→0 of a
i.e. event at 'a'

∴ we need not proceed with entire simulation but only affected gates.