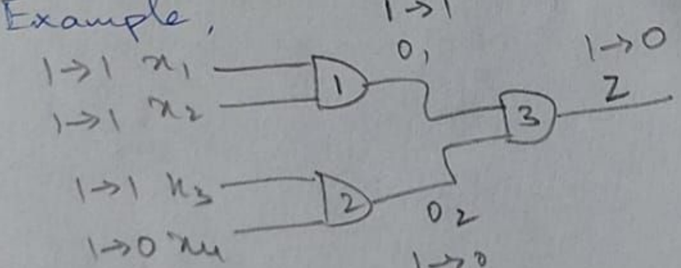
N. KAUSIK
CS6330 End Sem Part 2 CS21M037

6. Event driven sim is better than compiled sim as,
In compiled sim when we apply one input vector after another, we need to compute all the gate outputs again and again. i.e. If there are 10 gates, we need to computee 10 outputs for all input vectors.
However in event driven sim, we only compute the outputs of the gates whose inputs have changed due to changing the input vector and propogate.
In very huge circuits this can reduce the number of gates we compute by a large amount as by changing the input vectors, only few bits in the inputs are changed and hence only few gates are affected.

Example,



changing input from 1111 to 1110,

Only gates 2 ad 3 are affected and hence no need to recompute gate 1.

Compile sim computes all 3 gates
Event driven sim only computes 2 gates.
This reduction is amplified in huge circuits.

7. In circuits we can have fanouts of a wire reconverging at some later level. In such cases, D-Algo takes large number of time (iterations) as these reconvergent fanouts cause a lot of contradictions when we try to justify one input, the other input fails to get justified.
As PODEM uses cross-path analysis it avoids too many such contradictions and finishes faster.
Also, when a contradiction happens we need to revert the status of the circuit to before the last decision was made in D Algo. This is done by storing the status of the circuit before each decision and accessing it again in case contradiction occurs.
Hence these operations require a lot of memory access.
Since PODEM does much less of these operations than DAlgo as it faces less contradictions, it is better if memory access time is significant.
Example, if memory access time is m and D Algo faces 10 contradictions and PODEM only faces 3 contradictions,
D Algo time = 10M, PODEM time = 3m.

8. Since while performing heuristic analysis on digital circuits, we often take some decisions, then find the implications of that decision and in case of any contradictory implications we need to traceback to the failed decision state and take some other decision and continue.

Recursive Algorithms facilitate this kind of behaviour since when we take a decision, we can call a recursion. If it is success, that decision is correct. If it fails, we can traceback to the current recursion and take another decision.

Thus recursive algos are suited for heuristic algorithms.

9. Engineers dont want CAD tools to replace x with 0 and 1 as,

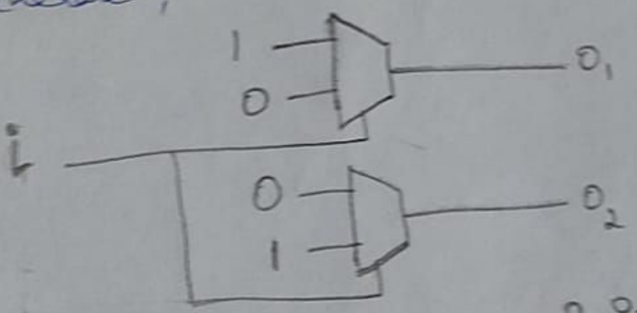
In any circuit there might be a large num of wires and hence large # of x.

Since for each x, by subs it as 0 and 1, 2 diff circuit states are created, a exponential # of circuit states will be created if we want to replace all x with 0 and 1.

10. Given a Sequence of 'm' n-bit vectors, we generate a sequential circuit consisting of n Multiplexers which can take input as the current position in the sequence ($\lceil \log_2 m \rceil$ bits) and will give the required n bit vector as output from the n multiplexers.

Example to get the sequence (10, 01).
$n=2$
$m=2$
$\lceil \log_2 m \rceil = 1$

We can generate,



input = (0, 1) for get position 1 and 2 vectors.

$$\begin{array}{ll} & O_1\ O_2 \\ \text{Output received} \Rightarrow i=0 \Rightarrow \text{out} = & (1\ 0) \\ i=1 \Rightarrow \text{out} = & (0\ 1) \end{array}$$

When we pass i by incrementing 1, we get the sequence as needed.

Here, each multiplexer takes a $\lceil \log_2 m \rceil$ bits select and it's inputs are fixed as the 'm' values of the x^{th} bit for the x^{th} multiplexer.

So, if select is 100, the 5th vector in the Sequence is given by the n multiplexers.

Such an algorithm can be followed to generate a sequential circuit to generate a given sequence of n-bit vectors.

In context of digital circuits,
Using some algorithms we would have determined the set of test input vectors to detect faults in our circuit. We however need to apply these test vectors in a correct order / sequence so that the number of toggles in the input vectors when we change from one vector to the next is minimised overall.
This is done to minimise the power dissipation as it is prop to the # toggles.

eg. Test vectors $= \{ \overset{T_1}{(000)}, \overset{T_2}{(101)}, \overset{T_3}{(010)} \}$

If we apply in sequence $T_1 T_2 T_3$,

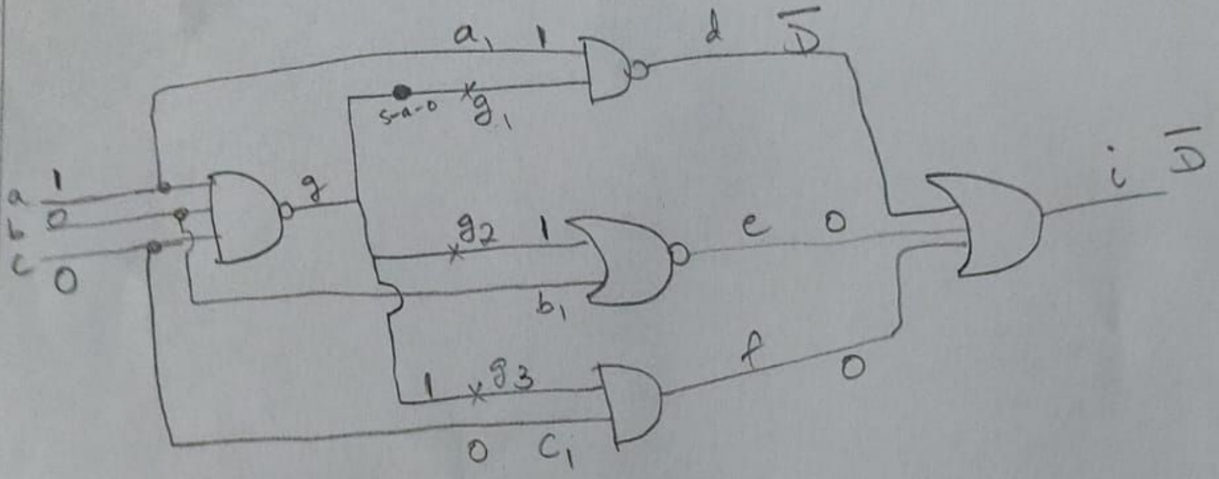
$\#$ toggles $= 2 + 3 = 5$

If we apply as $T_3 T_1 T_2$,

$\#$ toggles $= 1 + 2 = 3 < 5$.

Hence the sequence of test vectors is important while testing to minimise power dissipation.
Hence having an algorithm to produce a seq circuit that produces these vectors in the required sequence is very useful.

1.



detect g_1 : s-a-0,

	Decisions	Implications	D-Frontier
S_1	select i $f = 0$	To drive 1 in g_1, $g = 1$, $g_2 = 1$, $g_3 = 1$ $e = 0$, $d = \overline{D}$, $a_1 = 1$, $a = 1$	$\{i\}$ as i has inputs $\overline{D}$, $\overset{(e)}{0}$, x and output x.
		as $g_3 = 1$, for $f = 0$, $c_1 = 0$, $c = 0$ [c=0 implies g=1 no contradiction] $i = \overline{D}$ (Propogated to PO)	$\{\}$

Since propogated to PO and D Frontiers are empty,
In gate for 'g', if b is 0 or 1, still g and e are unaffected. We can take 0 or 1 for b.
If we take 0,
Test Vector to detect g_1: s-a-0 is, $\begin{pmatrix} a & b & c \\ 1 & 0 & 0 \end{pmatrix}$
when applied, If fault $i = 1$
If no fault $i = 0$