N. KAUSIK
CS21M037

# CS6330 End Sem Part 1

1) a) Fault Collapsing

It is the process by which given a circuit, we identify the set of faults on wires in the circuit which can be used to detect any fault ~~in the~~ in the circuit. I.e. By detecting just these collapsed faults, all other faults on all wires of the circuit are detected.

Eg. A circuit has 100 possible faults (50 wires). After collapsing we are left with 10 faults. Then we can find test vectors for only these 10 and by detecting these faults, indirectly we can also detect all the 100 faults.

b) Fault Dominance

We can say a fault 'f' dominates another fault 'g' if all the test vectors which detect g also detect f. Hence, if we try to detect g using test vectors, f is also detected.

c) Fault Equivalence

We can say that two faults 'f' and 'g' are equivalent if the possible test vectors that detect f and g are the same. f and g are equivalent if both f dom g and g dom f are true at the same time.

2. Digital Systems Testing is a complex problem to handle due to the possibility of complex circuits which can have many FANOUTs in wires and Reconvergence of those fanouts.

Due to this, most of the algorithms dont fall under 'P'. Most algorithms are NP Complete which means there is no known 'polynomial time' algorithm.

Due to these fanouts and reconvergences there is possiblity of contradictions when running algorithms like D Algo, PODEM etc. Hence we need to keep track and backtrack if necessary which cause algorithms to be slow and also consume more memory to run.

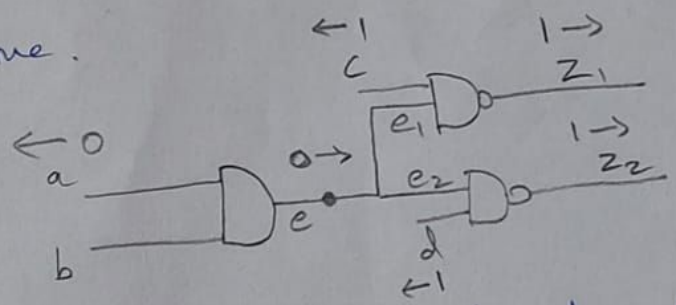
Even in cases where there is no fanout like a n-input AND made using 2-input AND gates, as the number of inputs rises, the number of gates rises exponentially. (num gates here = $2^n - 1$)

3. The ESFF data structure is used for Test Generation as, From the element table, we can directly get the number of inputs, number of outputs and their indices in Signal and Fanin table of any gate/element.

In Test Generation algorithms at each iteration we will need to be able to identify all input wires (Justify) and all the fanout wires (Propogate) of a current gate.

Using ESFF table we can get these directly in very less time.

Example,




to detect e s-a-0, we will try to get a test vector that produces a 0 at e.

Hence we need to Justify a as 0 and Propogate the error by trying to set the $Z_1$ and $Z_2$ as 1.

Further we need to Justify c as 1, d as 1 to do so.

Using ESFF table we can directly find the inputs of element e using element table and fanin table.

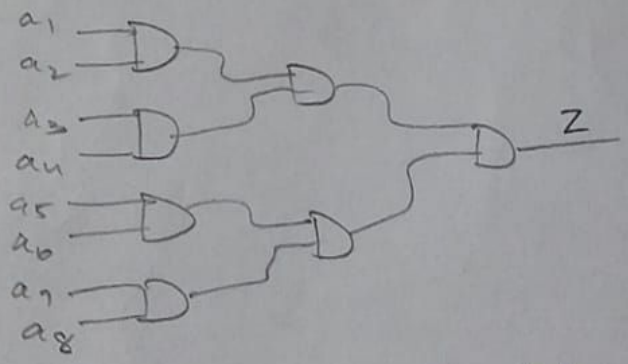
Also we can find out it's fanouts using fanout and signal table.

Hence ESFF is used for test generation.

4. Even though fault independent test generation algos are there, we still need fault oriented test generation to COMPLETELY test all the possible faults in the circuit.

The main disadvantage with fault indep test gen is that it can provide proper test vectors which can detect some faults in the circuit, however they may not give test vectors to detect ALL the possible faults within a reasonable time/iterations. So to detect those faults we need to generate test vectors using fault oriented test gen.

Eg.




To detect Z S-a-0, the only test vector which detects it is (11111111).

If we use fault indep test gen like Random Test Gen, the probability that we will get this vector and detect the fault is $\frac{1}{2^8}$, very small. Hence even after many tries, we may never be able to detect Z S-a-0.

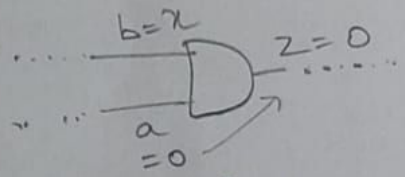
Using Fault Oriented Test Gen we can use this fault as target and generate the test vector directly.

Hence it is still used for Completely detecting all possible faults in circuit.

5. Since we are using a complete and sound algo and it says on a wire 'a' which is input to a AND gate somewhere in a circuit cannot be sensitized for S-a-0.

∴ It means we can never get a '1' on wire a using any test vector to sensitize a S-a-0.

That means for all input test vectors, $a = 0$.

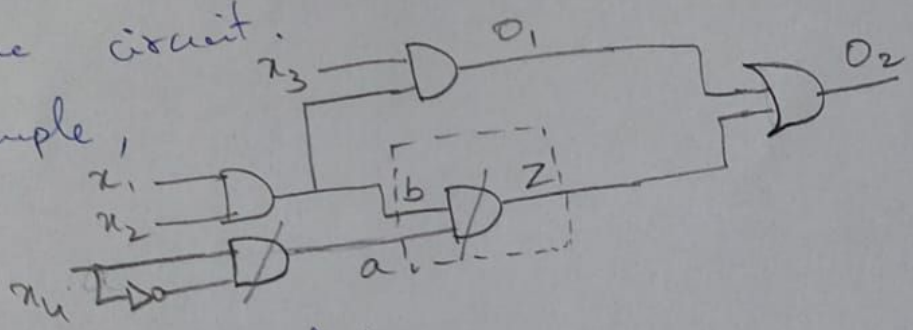



Since gate is AND, the output of this AND gate is always '0' for all inputs.

Hence we dont need to compute upto z and instead can pass a 0 to z directly.

Thus we can remove all the gates ~~for~~ which lie along the path from PIs to a or b which DOES NOT affect any of the other gates in the circuit.

Example,




This is reduced to,