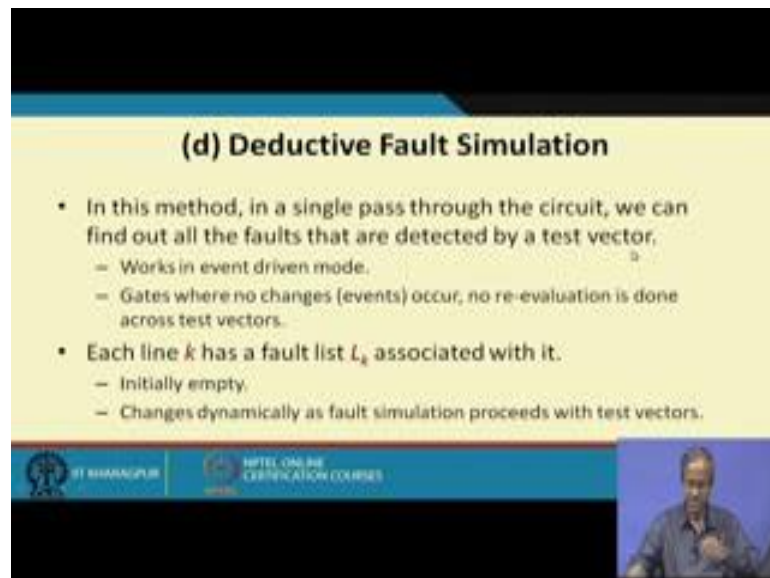


VLSI Physical Design
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 52
Fault Simulation (Part 2)

So, in this class we shall be discussing 2 more fault simulation algorithms called deductive and concurrent. Now conceptually they are similar in this sense that these methods try to simulate all the faults at the same time together, with test vectors applied one after the other. But the way they work they handle the faults are distinctly different.

(Refer Slide Time: 00:57)



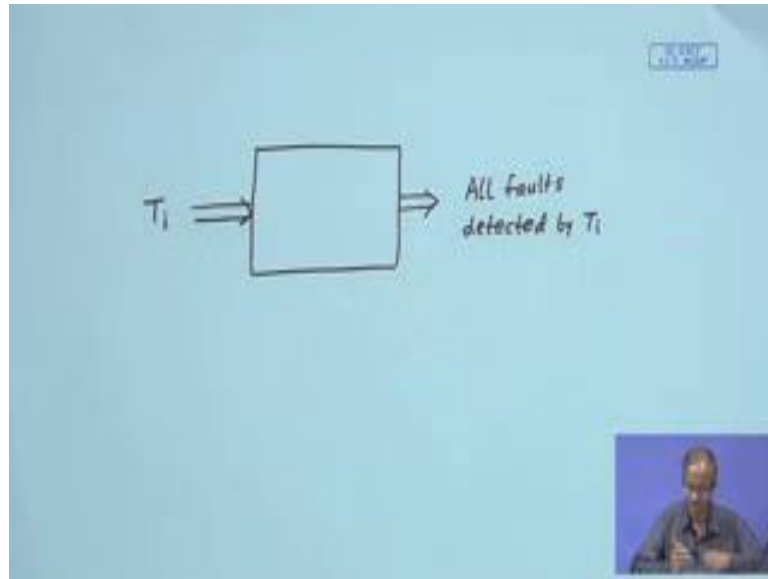
(d) Deductive Fault Simulation

- In this method, in a single pass through the circuit, we can find out all the faults that are detected by a test vector.
 - Works in event driven mode.
 - Gates where no changes (events) occur, no re-evaluation is done across test vectors.
- Each line k has a fault list L_k associated with it.
 - Initially empty.
 - Changes dynamically as fault simulation proceeds with test vectors.

The slide is part of a video lecture. At the bottom, there is a logo for IIT Kharagpur and text for 'NPTEL ONLINE CERTIFICATION COURSES'. A small inset video shows the professor speaking.

So, let us look into these methods one by one. So, the first method that we discussed is called deductive fault simulation. Now see these methods have very interesting characteristics, this says that we make a single pass through the circuit and in that single pass we find out all the faults that are detected by a test vector, which means it is something like this.

(Refer Slide Time: 01:23)



Let us say I have given circuit, I apply a particular test vector let say T_i . What I am saying is that, I scan the gates in the circuit just one pass, I evaluate the output and I can immediately determine all faults detected by T_i this is a very interesting thing. So, I am somehow implicitly simulating all the faults together. So, if there are say 100 test vectors at to do these process 100 times right? So, there is another interesting point is that this method works in even driven mode; like there is a concept of a event. So, for a particular gate if there is no event occurring, we do not evaluate or process the gate at all.

Let us say this is an across test vectors, what I mean to say is that suppose I have a applied the first test vector T_1 , I have processed my circuit I have find out the faults detected. Now I apply the second test vector T_2 , with respect to the last one I do not destroy the data structure that you have already constructed with T_1 that remains. Now I apply the second test vector, and I make changes to those data structure only when required. I need not have to re evaluate all the gates, I find out in which gates the events are occurring I only make some modification in those gates for the portion of my circuit where no events are occurring, I do not make any changes the same data structure as compared to the last test vector T_1 remains; this continues with one test vector, after the other after T_2 comes T_3 then T_4 and so on. So, on the average this runs much faster.

So, the concept is that for every line in the circuit, we associate a fault list. So, we shall be defining the fault list in the next slide, initially the fault list for all the lines are empty

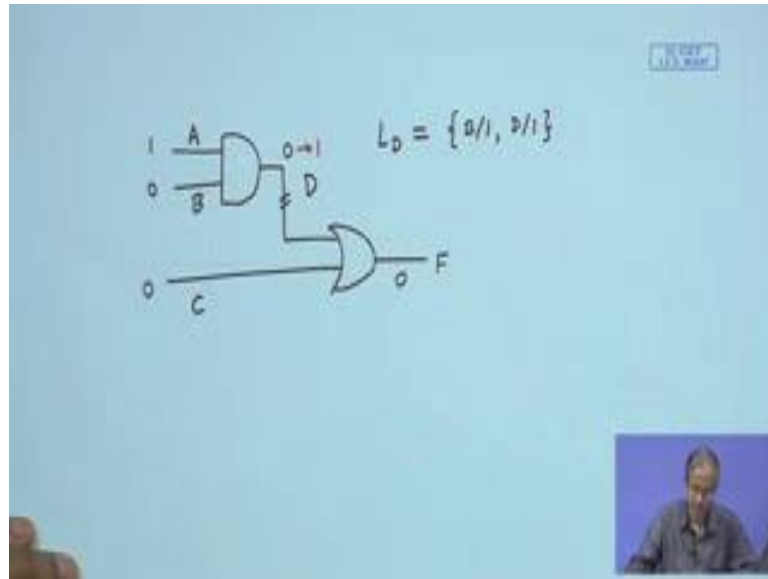
as simulation proceeds this fault lists get populated by some information, and they change dynamically as we simulate with test vectors one after the other. And as I said we do not destroy the information over a one process of simulation, we keep it apply the next test vector and make modifications only where necessary, this is one way we can speed up the process; let us see what is the fault list now.

(Refer Slide Time: 04:26)

- What is fault list?
 - The fault list L_k associated with line k is the set of all faults $\{f\}$ that causes the values of line k in the fault-free circuit N and the faulty circuit N_f to differ at the current simulation time.
 - The fault list for the primary output gives the detectable faults.
- Two interleaved steps:
 - True value (fault-free) simulation is carried out for the given vector.
 - The value implied by the vector at every line in every faulty circuit is deduced (using simple set theoretic rules).

Fault list what it actually means the fault list L_k that is associated with the particular line k is the list of fault. So, the set of faults that changes the logic value of line k , what it says is that that causes the value of line k in the fault free circuit. Let us N denote and the fault free circuit and the faulty circuit N_f to differ; this is the idea let us take an example small example to illustrate this.

(Refer Slide Time: 05:08)

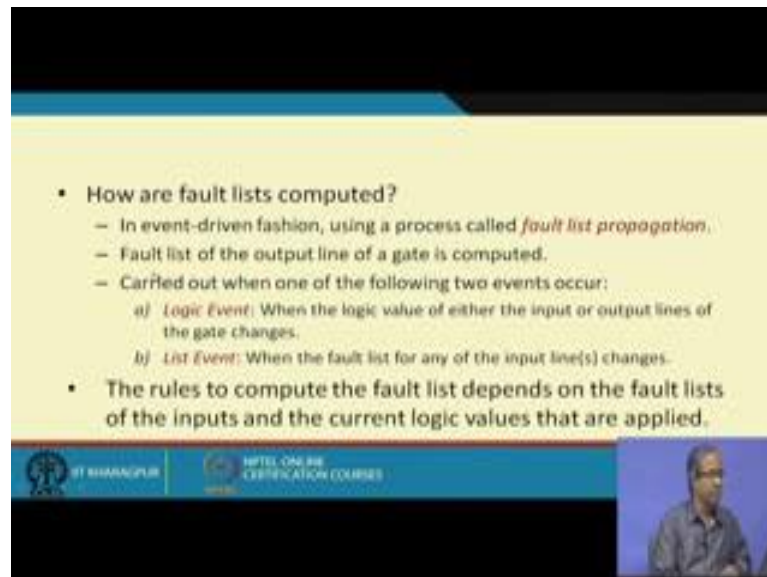


Let us say I have a gate here and gate let us say followed by an OR gate. Suppose we are carrying out simulation with a particular input vector let us say 1 0 and 0. Let us say the inputs are A B and C this is D. So, we are interested in line D, I shall be showing a compute example later. So, when I talk about the fault list of line D see D this is 1 0. So, the fault free value of D 0, and fault free value of the output is also let us say F is also 0. Now L D will contain the set of all faults, which will change the value of the line D; that means, it was 0 the value will change to 1. So, in this example let us see what are the faults which can change the values to 1; see 1 fault can be B stuck at 1 in the previous curve, if B is 1 then input is 1 and only this line will be 1, so one such fault will be B stuck at 1.

And other obvious fault will be D stuck at 1. So, these are the 2 faults. So, fault list will look like this, fault list will consists of all the faults that will change the value of this particular line D, with respect to the fault free version. Fault free value is 0, so what are the faults that will change it to 1 right. So, there is a process I shall show illustrate, we can systematically compute the fault list for all the lines. So, once we have completed the process, what will the fault list for the primary output denote? Suppose you have computed the fault list of the primary output, it will tell you what are the faults that will change the value of the primary output; which actually means what are the faults which are getting detected by the current vector, that will only change the output right with respect to the fault free 1.

So, there are 2 interleaved steps involved; one is the true value or fault free simulation with a given vector and we are concurrently computing the fault lists. So, there are some simple set theoretic rules I will be illustrating, using which you can compute the fault lists.

(Refer Slide Time: 07:56)



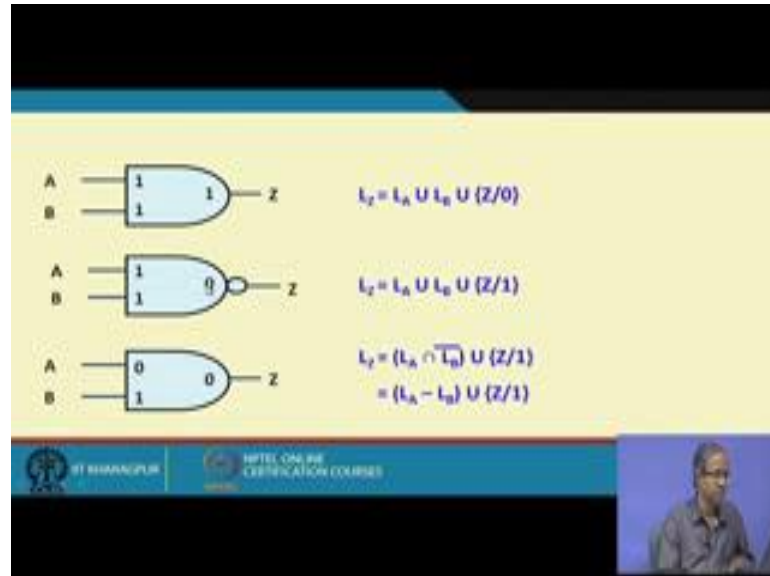
- How are fault lists computed?
 - In event-driven fashion, using a process called *fault list propagation*.
 - Fault list of the output line of a gate is computed.
 - Carried out when one of the following two events occur:
 - a) *Logic Event*: When the logic value of either the input or output lines of the gate changes.
 - b) *List Event*: When the fault list for any of the input line(s) changes.
- The rules to compute the fault list depends on the fault lists of the inputs and the current logic values that are applied.

Let us take some examples later after we talk about these events. Now fault lists are computed in an event driven fashion as I said; fault list propagation the example that I had showed just now here there was one fault B stuck at to 1 which has propagated from the input side to the output side, but D was a local fault right. So, there is a concept of fault list propagation as I move from the input to the output side, fault lists are systematically propagated ok.

So, for every gate we compute the fault list of the output line, and we evaluate this gate only when one of the following this events occur. What are the events the first event says that one of the input lines of the gate has changed its logic state; this is termed as logic event. So, I am applying the test vectors one after the other let us say in T 1 the inputs where 1 0, in T 2 it has become 1 1. So, I say that there is a logic event. List event means may be the input values have not changed, but one of the input fault lists have been modified. When the fault list for one or more of the input lines change; depending on this if there is an event we calculate or compute the fault list of the output if there is no event

we can simply skip that gate. The rules for computation will depend on the input values like I shall show with some examples.

(Refer Slide Time: 09:49)



First example we taken and gate; to input and gate with the inputs as 1 1, the output is also 0 1 output also 1. So, let us try to say with respect to this gate this A B need not be primary input, A is coming from some other sub circuit; B is coming from some other sub circuit. So, L A will indicate all faults which change the value of A change means 2 0 and L B will denote all faults that change the value of B it means 2 0; these are and gate. So, under what conditions will the output change, both inputs are 1 1. So, if at least one of the inputs becomes 0 or both then the output will change. So, we denote it as L A union L B said theoretic operations.

So, either A changes or B changes then output will be 0; union of course, output stuck at 0 will be an obvious fault this will also be there been L Z. It take a NAND gate if the output is 0 here same thing, it will be L A union L B union Z stuck at 1, because now the fault free value is 0. This is a slightly more complex case you take a 2 input and gate with the inputs as 0 1, and the output is 0. Now I am trying to find out under what conditions can the output value changes to 1? I can tell output will be 1 will here I mean; obviously, just stuck at one condition or this a changes from 0 to 1, and B does not change B should remain at 1; that means, both is 1 1. This I denote like this L A means a changes and intersection, L B bar B does not change. Now in set (Refer Time: 12:02)

notation if you draw the when diagram, you can easily prove that $L A$ intersection $L B$ bar is nothing, but $L A$ minus $L B$ said difference you take out all the faults from $L A$ that I common in $L B$.

So, if you have a condition like this $L A$ is the line where your trying to make the change, minus $L B$ the line where this no change. So, if there is another line C , C is also at 1 then it should be $L A$ minus $L B$ minus $L C$; that means; B and C both are not changing. So, depending on the inputs your rules will vary because the same and gate, but rules are different right, it will depend on the inputs if it is $A 1$ and $B 0$ that will be $L B$ minus $L A$ right ok.

(Refer Slide Time: 12:52)

The slide contains the following content:

- Advantages:**
 - All faults are handled together in a single pass.
 - Faster than parallel fault simulation.
- Disadvantages:**
 - Memory requirements are unpredictable.
 - Set theoretic rules for computing fault lists are difficult to derive for complex blocks.

Below the text is a diagram of a MUX block with two inputs, $A0$ and $A1$, and a select line S . The output is F .

To the right of the diagram is the rule for $S=0, A0=1, A1=1, F=1$:

$$L_f = (L_{A0} \cup (L_{A0} \cap L_{A1})) \cup (F/0)$$

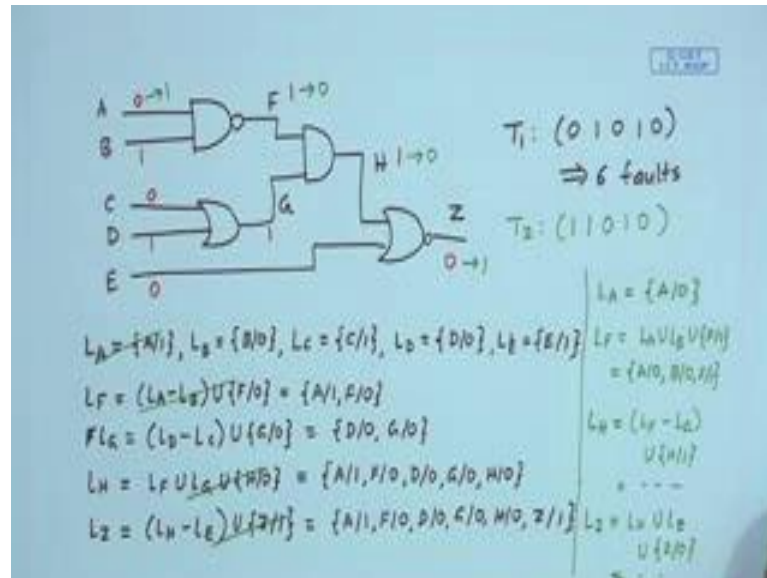
At the bottom of the slide, there is a logo for 'IT MANAGER' and 'INTEL ONLINE CERTIFICATION COURSES'.

So, the advantage of this method is that all faults are handled together in a single pass and therefore, it is faster than parallel fault simulation. But disadvantage is that the fault lists can grow unpredictably. So, memory requirements is typically much higher, and also the set theoretic rules in terms of computing is a little difficult.

And the set theoretic rules are possible for many block let us say let us take another block like a multiplexor, I shall say as you show that how it is more difficult. So, you can also frame set theoretic rules for simple blocks like a MUX. Let us consider situation where you select line is 0, first input and the second input both are 1 1 and therefore, the output is 1; this is 0 a 0 is selected it is 1. So, under what condition this F will be 0? First condition is there is an event on line $L G A 0$; that means, $A 0$ was 1 it becomes 0; or

there is another condition there is an event in L S means S becomes 1, which means A 1 get selected and there is also an event in A 1, A 1 is also becoming 0, then this 0 will also propagate and of course, union up stuck at 0.

(Refer Slide Time: 14:37)



So, you can see that as the blocks become more complex you can have a rule, but the rules can become more and more complicated, but let us work out an example right let us see. Let us take an example as follows, let us say we have an NAND gate, we have an OR gate, AND gate and another NOR gate. Let us say there are 5 inputs A B C D E; let us give some names to the lines F G H, let us say output is Z. Let us assume that we apply some input vector let us say 0 1 0 1 and 0. So, where it is 0 1 what will be the output of the NAND gate? It is 1 or gate also 1 and gate 1 1 is 1, 1 0 NOR gate is 0. So, with this values let us try to compute the fault list for this first test vector or first test vector T 1 is 0 1 0 1 0.

So, we start from the input side see L A is a primary in A is a primary input, so L A will contain only those faults with change the value of A, which means only A stuck at 1 right? Similarly L B will be B is at 1. So, B stuck at 0, L C will be C stuck at 1, and L D will be D stuck at 0 and L E will be E stuck at 1 these are the primary input faults. Now let us use those rules to deduce the fault list of other lines, let us look at line F. You see F is a NAND gate, inputs a 0 1 output is 1. So, how the output can become 0? This A has to change to 1, but B should not change; which means L A minus L B of course, union F

stuck at 0. So, if you do $L A \text{ minus } L B$ this is nothing common only A stuck at 1 remains. So, this said becomes A stuck at 1 and F stuck at 0. Similarly let us do $L G$ sorry $L G$, this is an OR gate with the input 0 1 output is 1. So, here again output will be 0 if D changes, but C does not change. So, it will be $L D \text{ minus } L C \text{ union } G$ stuck at 0.

So, $L D \text{ minus } L C$ is how much? There is nothing common again only D stuck at 0, and G stuck at 0; then comes H; LH we look at this and gate both inputs are 1 and 1. So, the output will change if any one of them changes. So, it will be $L F, \text{ union } L G, \text{ union } \text{fault}$ in the output H stuck at 0 this is $L F$ and this is $L G$. So, you take the union this is nothing common. So, A 1, F 0, D 0, G 0 and H 0. This will be $L H$ and finally, you compute $L Z, L Z$ this is a NOR gate 1 0. So, it will make the output 1, H has to change, but E should not change. So, it will be $L H \text{ minus } L E \text{ union } Z$ stuck at 1. So, $L H \text{ minus } L E$ there is nothing common. So, this remains and this 1 and stuck at 1.

So, we have obtained the fault list for the output line Z, you see there 6 faults here. So, what does this mean? It means that for all this 6 faults the output value Z will change from 0 to 1, which means this test vector T 1 can detect 6 faults and these are the 6 faults right. Now let us suppose we apply a second test vector where we make the change in one of the bits, so in let us say next is this 1 1 0 1 0 only a changes. So, I am showing the relevant changes in diagram only, this changes to 1, this is a NAND gate it becomes 1 1. So, F also changes to 0 and gate 0, and 1 this also becomes 0, 0 0 this also becomes 1.

So, you see for this particular gate or the line C D B C D E G there is no event; in neither a logic event nor a list event. So, I need not reevaluate these again, but we have to reevaluate a few of the lines, I am just showing those lines. So, I am just showing it here in the side. So, I need to reevaluate L A. Now a is 1, so L A will be a stuck at 0; this is change; so the old list gets removed. Now this NAND gate is having 1 1 as input. So, the condition for the output changing will be will be $L A \text{ union } L B$. So, L F is recomputed there is has been a list event. List event L A has been modify the also there is a logic event the input has change state. So, F has to be recomputed it. So, L F will be $L A \text{ union } L B \text{ union } F$ stuck at 1; you see L B you take the old value the previous value. So, is L A is now a stuck at 0, L B you take the last previous value and F stuck at 1.

Similarly, you compute H same way you continue. So, you compute the value of L H now the input has 0 and 1 and gate, output will change if F changes, but G does not

change. So, it will be $L F \text{ minus } L g \text{ union } H \text{ stuck at } 1$. So, you can similarly calculate and finally, when you come to Z , $L Z$ will be $0 0$ in or. So, means if any one of the changes output will change. So, it will be $L H \text{ union, } L E \text{ union, } Z \text{ stuck at } 0$. So, this you can calculate similarly. So, you see that ensure you are just re computing some of the fault list. A you are re computing, $L F$ you are re computing then $L H$ you are re computing, $L Z$ you are re computing what some of them you are not touching they remain the same as the previous one.

So, this illustrates even remain simulation right. So, you are not simulating the whole circuit, but we are simulating only those parts which are required.

(Refer Slide Time: 23:32)

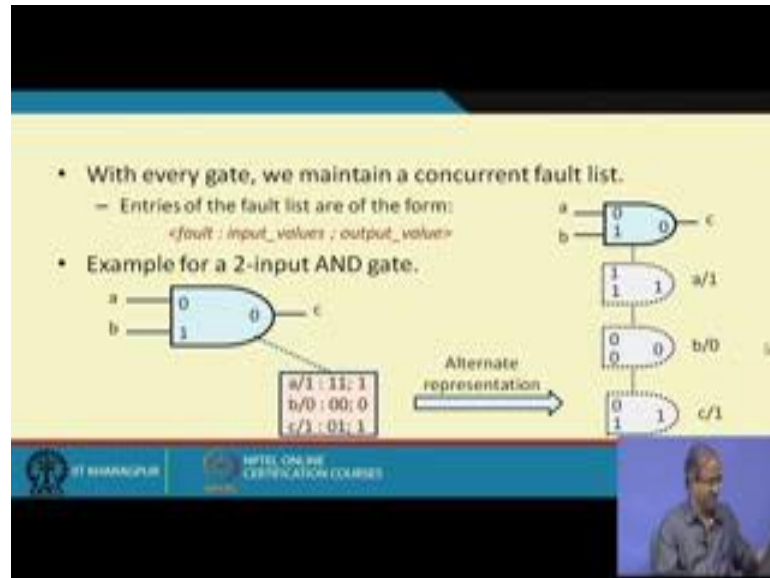
(e) Concurrent Fault Simulation

- An observation:
 - Most of the time during simulation, most of the values in most of the faulty circuits agree with the corresponding values in the good circuit.
- Basic idea:
 - Like in deductive fault simulation, all faults are considered together.
 - The fault-free version of the circuit, and each of its faulty versions, are concurrently simulated for a given vector.
 - Simulate the good circuit N .
 - For every faulty circuit N_i , simulate only those elements in N_i that differ from the corresponding ones in N .

So, let us move to the last method that we should be discussing one fault simulation is the concurrent fault simulation. Now concurrent fault simulation is based on the premise that during simulation most of the values in most of the faulty circuits do not change; that on the average only a few lines small percentage of the lines they change in the logic values, this is the observation. So, here again just like in the previous method we consider all the faults, but we shall see we maintain much more information here, we maintain not only information about the faults, but also the input and output values of the gates. So, essentially we simulate the fault free version of the circuit and each of the faulty version concurrently; that is why it is called concurrent false simulation. I shall be explained in this with example.

So, we simulate the good circuit, and for every faulty circuit you simulate only those for which at least one of the input or output values are changing with respect to N. So, I shall be illustrating this with the help of an example.

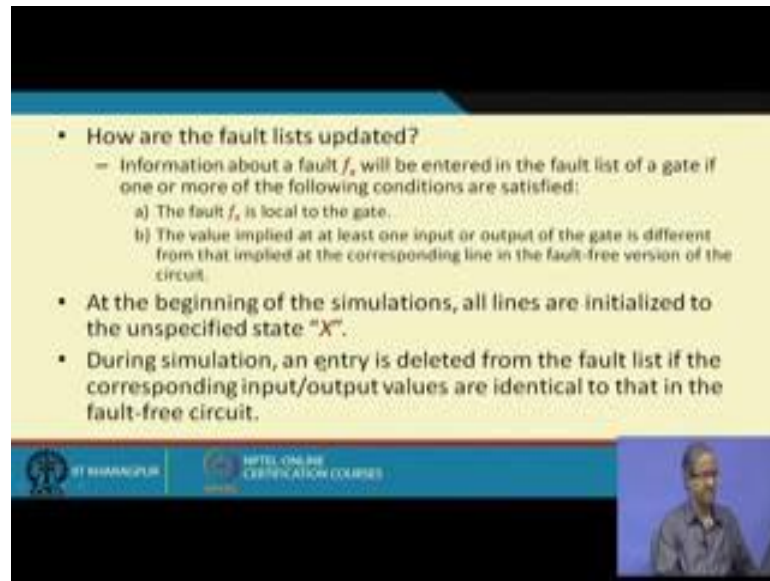
(Refer Slide Time: 25:00)



Let us take 2 input and gate with the inputs 0 1 and 0; here also we maintain fault list, but as I said the nature of the fault list is more elaborate. We keep information of not only the faults, but also the input values and the output value. So, for this particular gate you see *a* is 0 *b* 1 and *c* 0. So, what are the faults which can change at least some thing? One is *a* stuck at 1, this input can change; if this fault happens the input will become 1 1 and the output will be 1, *b* stuck at 0 *b* is 0, input will be 0 0, but the output does not you see here even if the output does not change we keep it in the fault list.

And lastly *c* stuck at 1 where input is 0 1, but the output is 1. Now in many text books this same information which is shown in a tabular fashion like this, it is represented like this as if this is the fault free version of your gate, and these are the 3 faulty versions is shown like this. So, you are simulating them together; you are simulating the fault free version as well as these 3 faulty versions which change at least 1 input or output lines of the gate right this is the idea.

(Refer Slide Time: 26:33)



• **How are the fault lists updated?**

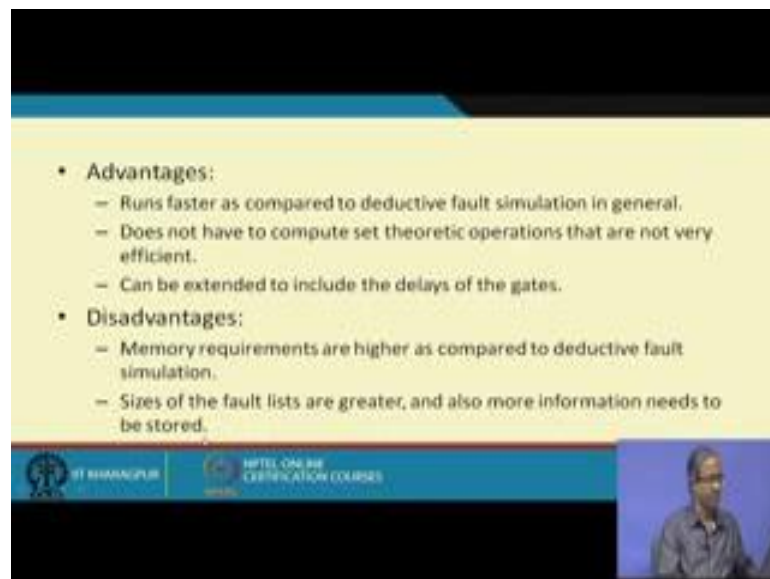
- Information about a fault f_s will be entered in the fault list of a gate if one or more of the following conditions are satisfied:
 - a) The fault f_s is local to the gate.
 - b) The value implied at at least one input or output of the gate is different from that implied at the corresponding line in the fault-free version of the circuit.
- At the beginning of the simulations, all lines are initialized to the unspecified state "X".
- During simulation, an entry is deleted from the fault list if the corresponding input/output values are identical to that in the fault-free circuit.

NPTEL ONLINE CERTIFICATION COURSES

Now, how are the fault list updated here? Fault list will be updated if one and more of the following conditions are satisfied; firstly of course, the fault is local to the gate or the value implied at least one input or the output of the gate is different from that in the corresponding fault free version.

Now, I shall try to explain this with the help of an example. And at the beginning all lines are in an unspecified state X as simulation proceeds, the values get defined and the fault lists get updated.

(Refer Slide Time: 27:19)



• **Advantages:**

- Runs faster as compared to deductive fault simulation in general.
- Does not have to compute set theoretic operations that are not very efficient.
- Can be extended to include the delays of the gates.

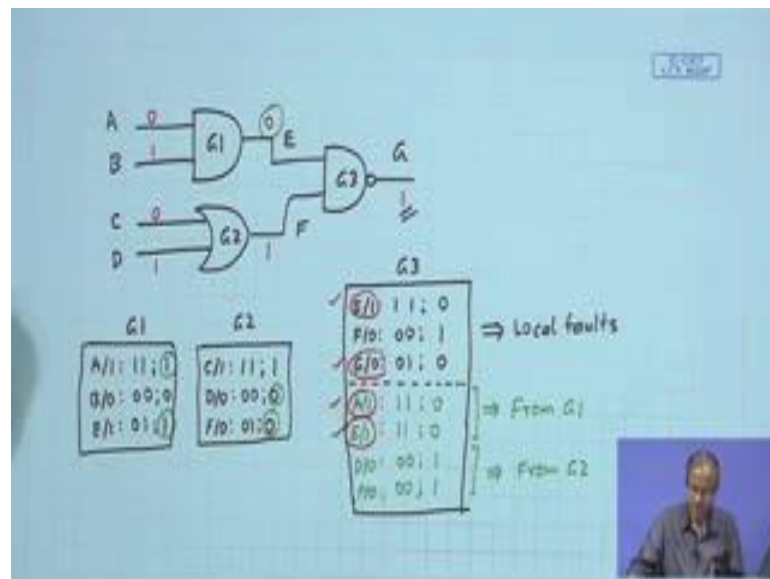
• **Disadvantages:**

- Memory requirements are higher as compared to deductive fault simulation.
- Sizes of the fault lists are greater, and also more information needs to be stored.

NPTEL ONLINE CERTIFICATION COURSES

So, the advantage of this method is you see here there are no complex set theoretic rules. So, on the average it runs faster as compare to deductive fault simulation right. Here we can also include delays of the gates, but the problem is that here the fault lists are much more elaborate. So, memory requirement is higher than compared to deductive fault simulation. So, you need more information to be stored. So, let us illustrate this with the help of an example.

(Refer Slide Time: 27:55)



So, we take a simple example here, let us take circuit consisting of 3 gates; the inputs are A B C and D this is E F and the output is G. So, this is a NAND gate output is G.

So, let us name the gates G 1 G 2 G 3. Now one difference is there with respect to deductive fault simulation, because in the earlier case we defined the fault list corresponding to each of the lines, but here we are defining fault list for the gates not the lines, this is one difference. So, let us try to simulate the circuit with the input value let say 0 1 0 1. So, this value is 0, this value is 1 and this value is 1. I am just showing you for 1 simulation. For this case level where level you process the gates first look at G 1. I am showing the fault list for the different gates for G 1; G 1 the inputs are 0 1 output is 0. So, what will the fault list contain? One fault will be a stuck at 1 for which the inputs will be 1 1, and the output will also become 1. Second fault is B stuck at 0, input is 0 0 output is 0 and third 1 is E stuck at 1.

So, input is 0 1 output will be 1, this is the fault list for gate G 1. Similarly the fault list for gate G 2 will be like this, it is 0 1 1 this is an OR gate. So, there can a fault 1 line C, C is stuck at 1. So, it will be 1 1 no change in the output, D stuck at 0 0 0 output will become 0, or F stuck at 0 input is 0 1, but the output is 0 right. Now let us come to G 3. So, here we shall see how fault lists are propagated. First let us look at the local faults 0 1 and 1. So, the faults will be E stuck at 1, F stuck at 0 and G stuck at 0. So, if you have E stuck at 1 input will be 1 1, NAND gate output will be 0. F stuck at 0 0 output 1 and this 0 1 output 0.

So, I am showing it separately because these are the local faults. Now you come to the propagated faults, you see propagated faults I am showing by different color, you see here there is an event on line E if this 0 changes to 1 right. So, I have considered the local faulty here, but this E might be changing to 1 because of some fault in G 1 also which changes the output to 1. So, look at G 1; C wherever it changes to 1 it is here and here right these 2 faults. So, these 2 faults will get added to this list. So, both A stuck at 1 and E stuck at 1 can make this first input as 1. So, this will can make it 1 1, this E stuck at 1 also can make it 1 1. So, the output will be 0.

So, these 2 faults these have been propagated from G 1. Similarly look at F well we have already inserted F stuck at 0 local fault here, but F might become 0 because of so fault which is there in G 2, which are this and this. So, also add those 2 faults; D stuck at 0 and F stuck at 0. So, this will become 0 0 these 2 faults are from G 2. So, you see that in this way you can proceed you can compute the fault list finally. So, when you reach the fault list of the final gate output, you know that the fault free value is 1, you look at all those faults for which the output is coming as 0 like one is this, one is this, one is this and one` is this. So, you can conclude that this particular test vector detects these 4 faults right. And when the second vectors applied we follow process which is very similar to the deductive fault simulation; same way you proceed, same way you modified the fault list if there is no event do not making the changes same way.

So, we have actually illustrated 2 methods of fault simulation, where you have seen that we can simulate all the faults together it is pretty fast, but memory requirement is higher than the earlier methods we mentioned. So, in our next lecture that will follow we shall be looking at the other processes of testing like test generation design for test ability etcetera.

Thank you.