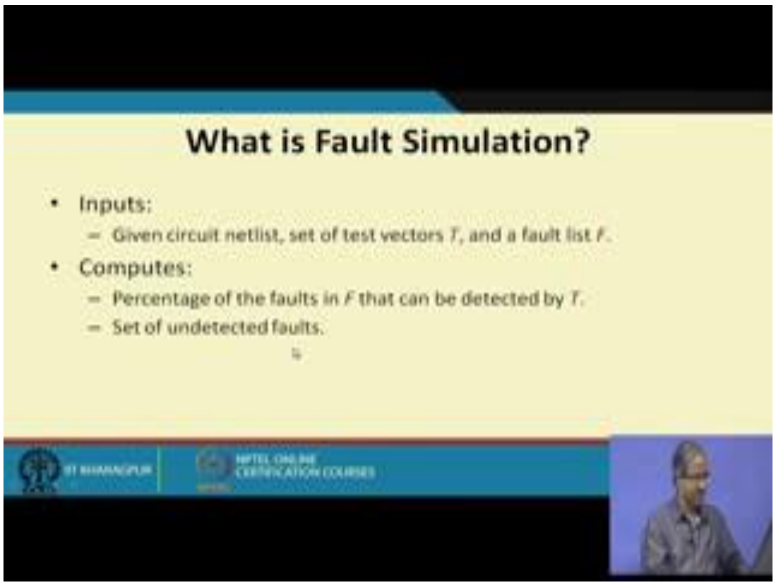
VLSI Physical Design

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

Lecture – 51
Fault Simulation (Part 1)

So, we know start with the discussion on Fault Simulation. Fault Simulation is an very important tool which is used in various stages during the you can say test flow. It is used in conjunction with text generation; it can be used much later during the test flow cycle and so on. So, this is more like a tool which is used to get or analyze the quality of the test vectors and depending on the result of the analysis you can take some remedial actions. So, let us see what fault simulation is.

(Refer Slide Time: 01:01)

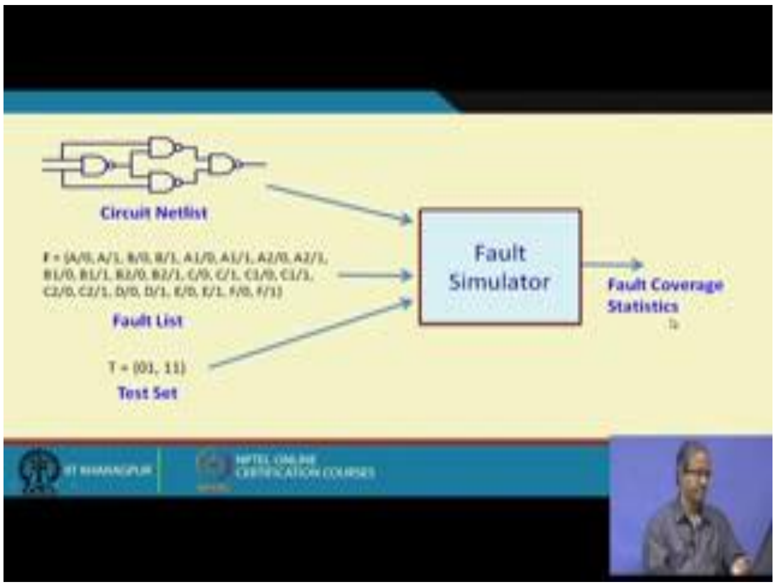


What is Fault Simulation?

- Inputs:
  - Given circuit netlist, set of test vectors *T*, and a fault list *F*.
- Computes:
  - Percentage of the faults in *F* that can be detected by *T*.
  - Set of undetected faults.

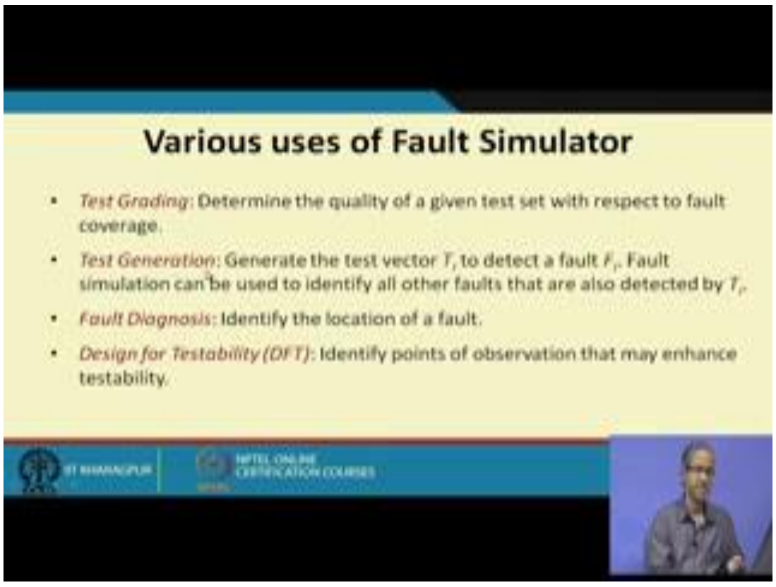
So, the input to a fault simulator are of course, the given circuit in the form of a netlist set of test vectors, and a set of faults, this is called the fault list. So, what the fault simulator computes, it tries to find out what are the faults that are getting detected out of this fault list by this test vectors; so, percentage of the faults in F that can be detected by T. So, addition it can also gives us some valuable information as to what are the yet undetected faults, the faults which are still not detected.

(Refer Slide Time: 01:46)


Circuit Netlist
F = {A/0, A/1, B/0, B/1, A1/0, A1/1, A2/0, A2/1, B1/0, B1/1, B2/0, B2/1, C/0, C/1, C1/0, C1/1, C2/0, C2/1, D/0, D/1, E/0, E/1, F/0, F/1}
Fault List
T = {01, 11}
Test Set
Fault Simulator
Fault Coverage Statistics

Diagrammatically so we just show it like this; we have a fault simulator, so as 1 of the inputs we give the circuit netlist. As the second input we give the set of all faults well, this can be the original set of faults after fault collapsing whatever, and thirdly we have a set of test vectors. So, what you get at the output is some kind of statistics with respect to fault coverage; which are the faults that are detected, which are the faults which are not getting detected and so on.

(Refer Slide Time: 02:26)



Various uses of Fault Simulator

- *Test Grading*: Determine the quality of a given test set with respect to fault coverage.
- *Test Generation*: Generate the test vector T_i to detect a fault F_i. Fault simulation can be used to identify all other faults that are also detected by T_i.
- *Fault Diagnosis*: Identify the location of a fault.
- *Design for Testability (DFT)*: Identify points of observation that may enhance testability.

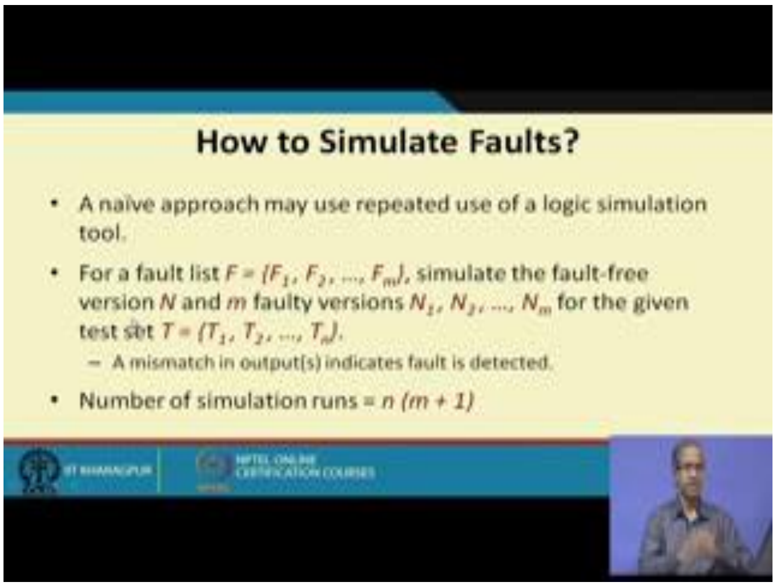
So, as I said we can use the fault simulation tool for various purposes. So, here I shown a few test grading concerns the determination of the quality of a given set of test vectors. Like suppose there are alternate ways of generating some test vectors, I use 2 alternate methods to generate the test vectors for the same circuit; let us say T 1 and T 2. Now I want to compare an access which 1 is better or not, typically I use a fault simulator for these purposes. I run my circuit and the list of faults with this set T 1, I again run with T 2 and I compare the fault coverage I see which 1 is better right.

Second use is in the test generation process; here we generate a test vector T i to detect a fault F i. Now here where fault simulation is used is. So, as we are generating a new test vector T i, we immediately carry out fault simulation to find out what all other faults are also getting detected by this same test vector. Now see here you may be little (Refer Time: 03:53) that why you are using fault simulation along with test generation, I could have this test generation again. Now the thing is that the complexity of fault simulation is much less as compared to test generation.

Test generation takes much longer time as compared to fault simulation. So, we want to reduce how many times we need to run the test generation algorithm or the tool. Suppose I have 100 faults, should I run the test generation tool 100 times for generating test for these faults, or I generated test for the first fault, I immediately run fault simulation which I know is much faster and fault simulator tells me that this test vector also detects 5 other faults. So, I remove those 5 faults from the fault list.

So, my initial list of 100 faults, quickly gets reduced during the process and my overall time becomes much less. Fault diagnosis sometimes we try to identify the location of the fault through simulation by simulating faults, and of course in design for testability means approaches, where we can identify some points where some additional controllability and observability mechanism can be provided to improve testability, this can be done through simulation again.

(Refer Slide Time: 05:29)



How to Simulate Faults?

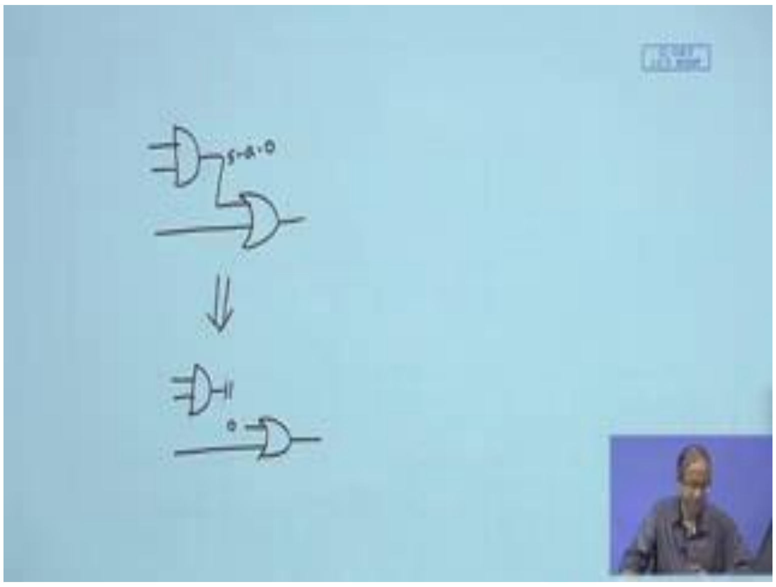
- A naive approach may use repeated use of a logic simulation tool.
- For a fault list $F = \{F_1, F_2, \ldots, F_m\}$, simulate the fault-free version N and m faulty versions $N_1, N_2, \ldots, N_m$ for the given test set $T = \{T_1, T_2, \ldots, T_n\}$.
  - A mismatch in output(s) indicates fault is detected.
- Number of simulation runs = $n\ (m + 1)$

Now, let us see how to simulate faults? We start with the naive approach; here we are saying that we shall be repeatedly using a simple logic simulation tool, what is a logic simulation tool? Logic simulation tool is software, that takes a circuit netlist as input and our test vector as a output, it will compute the logic values at the lines. So, it does not concern with faults, just the logic values that is why this is called logic simulation.

So, here suppose I have a fault list consisting of M number of faults F 1 F 2 to F m, what we do? We simulate the fault free version of the circuit let us call it capital N, and the faulty versions in presence of this faults let us call them N 1, N 2 to N m with respect to a given set of test vector let us say there are n test vectors. So, for any test vector with any fault if there is a mismatch in the output, it will indicate that fault is detected.

So, yet we since here what you are doing here we at a time we are inserting a fault in the circuit, then we were doing true value logic simulation. So, what I mean is something like this.

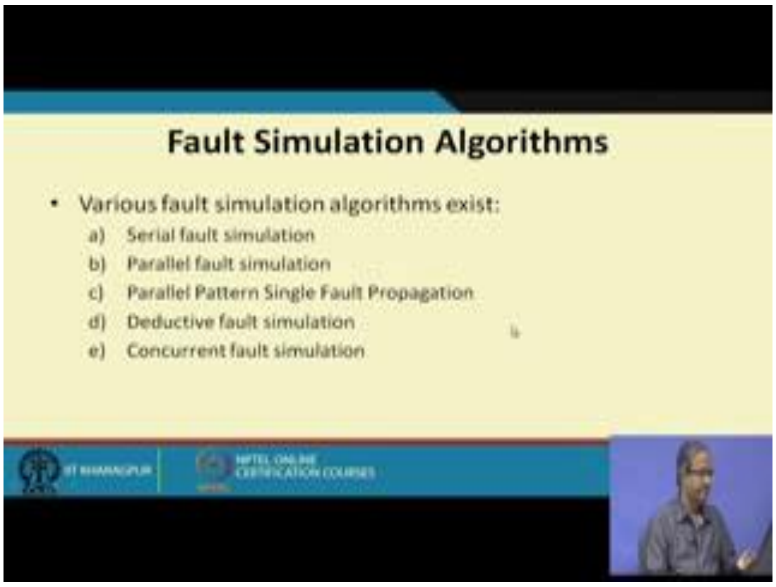
(Refer Slide Time: 06:59)



Suppose I have a simple circuit like this, now I want to introduce a stuck at 0 fault on the line, what I do? I make a small circuit modification, I apply a constant logic value here and then I simulate this is my circuit modification. So, this line is going no way and here I am applying a constant 0 to simulate the effect of stuck at 0.

So, after doing this I can use a normal true value simulation, I apply a logic value and see what my outputs values are right. So, for every test vector I have 2 simulate for the m faults and 1 time for the fault free version, and there are n test vector. So, n multiplied by m plus 1, so many runs of the logic simulation algorithm. So, it is of the order of n multiplied by m; number of test vectors multiplied by number of faults which is pretty large.

(Refer Slide Time: 08:12)

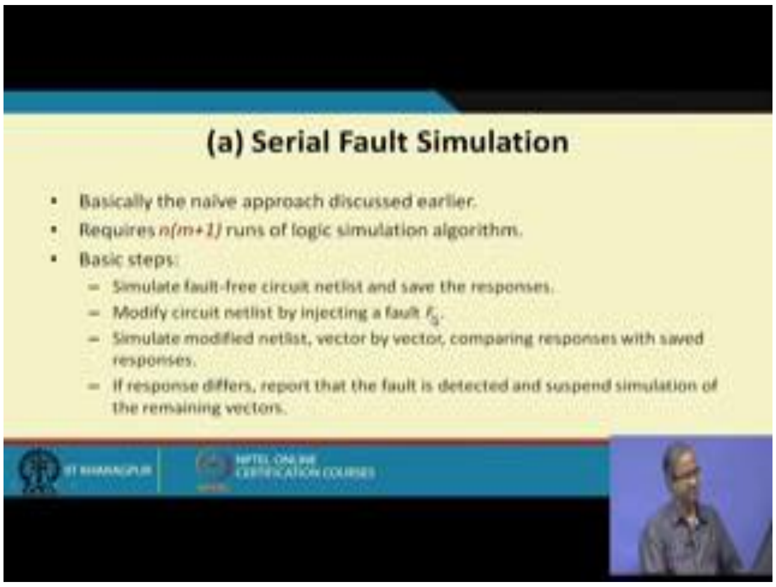


Fault Simulation Algorithms

- Various fault simulation algorithms exist:
  a) Serial fault simulation
  b) Parallel fault simulation
  c) Parallel Pattern Single Fault Propagation
  d) Deductive fault simulation
  e) Concurrent fault simulation

Now, the various fault simulations algorithms can be classified as follow, let the first 1 is the 1 that the naive approach we just now saw, the rest 1 are some kind of faster versions. So, we shall see how they work.

(Refer Slide Time: 08:34)



(a) Serial Fault Simulation

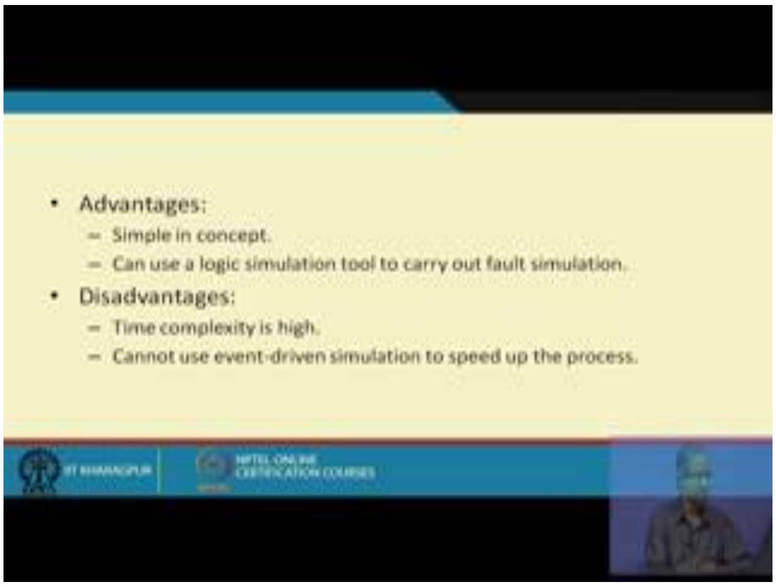
- Basically the naive approach discussed earlier.
- Requires *n(m+1)* runs of logic simulation algorithm.
- Basic steps:
  - Simulate fault-free circuit netlist and save the responses.
  - Modify circuit netlist by injecting a fault F_s.
  - Simulate modified netlist, vector by vector, comparing responses with saved responses.
  - If response differs, report that the fault is detected and suspend simulation of the remaining vectors.

So, serial fault simulation we have already seen this is the naive approach. So, we have seen that it requires so many runs of the logic simulation algorithm, where n is the number of test vectors, and m is the number of faults. So, they will be m number of faulty circuits and 1 faulty and 1 fault free circuit, so many simulations we have to carry out for every test vector.

So, the basic steps can be summarized like this we first simulate fault free circuit for all the test vectors, and save the responses in a file; then at a time we inject 1 fault, we modify the circuit netlist just in the way I should I shown you just we can make some changes in your circuit netlist, so as to incorporate the effect of the fault, then you simulate this modified netlist for every vector and you compare responses with the 1 which you have saved, to check whether the fault is getting detected or not. So, if you see that the fault is getting detected, then you can report that the fault is detected and you can suspend simulation of the remaining vectors because you know that it is already detected, you need not simulate with the remaining vectors.

So, actually the total time will less than n into m plus 1 this is the maximum. So, as soon as you find that a fault is getting detected, you need not simulate to the remaining vectors we move on to the next fault.

(Refer Slide Time: 10:09)

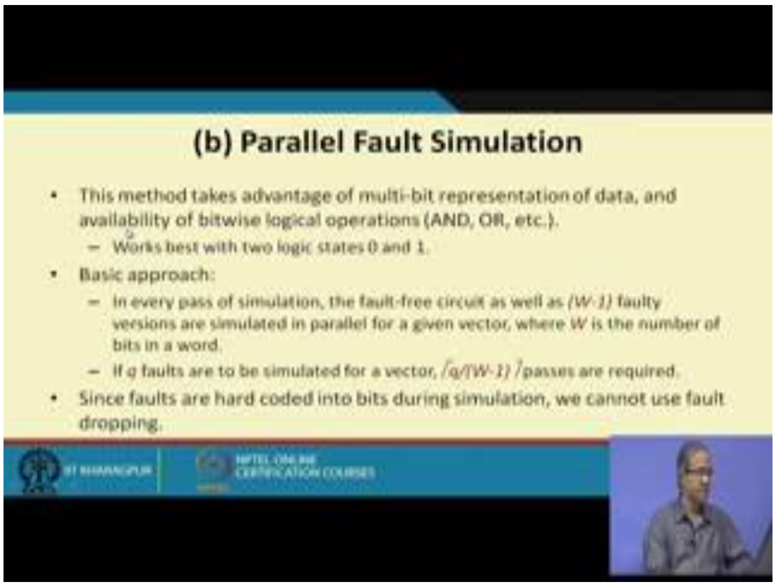


- Advantages:
  - Simple in concept.
  - Can use a logic simulation tool to carry out fault simulation.
- Disadvantages:
  - Time complexity is high.
  - Cannot use event-driven simulation to speed up the process.

So, advantage as this is very simple. Second you do not need as special software for this just a logic simulation tool is enough, drawback is that the time complexity is of the order of m into n which is pretty high, and also you cannot use it in the event driven mode.

We event driven mode you recall is a method of simulation, where I do not simulate the whole of the circuits, I simulate only that part of the circuits where some changes are taking place. So, unnecessary I will not simulate the whole thing, but this method of serial method unfortunately cannot be used in the event driven mode so let us now come with the improvements.

(Refer Slide Time: 10:56)



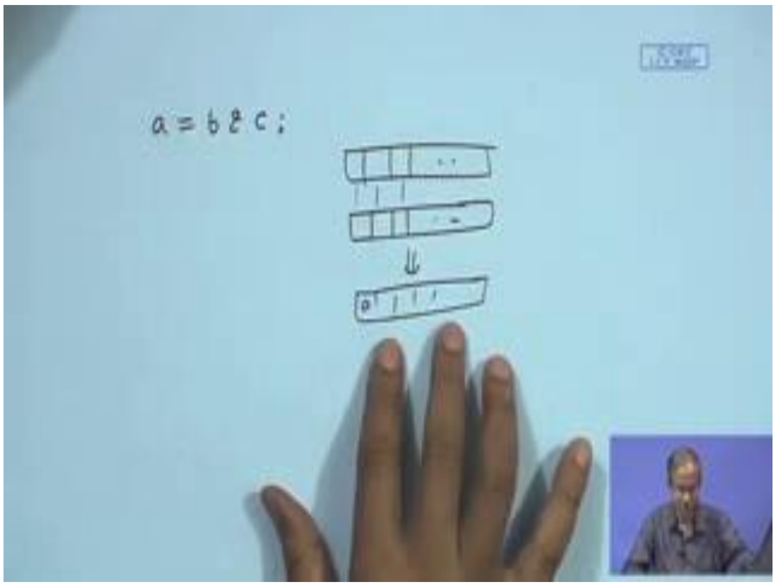
(b) Parallel Fault Simulation

- This method takes advantage of multi-bit representation of data, and availability of bitwise logical operations (AND, OR, etc.).
  - Works best with two logic states 0 and 1.
- Basic approach:
  - In every pass of simulation, the fault-free circuit as well as *(W-1)* faulty versions are simulated in parallel for a given vector, where *W* is the number of bits in a word.
  - If *q* faults are to be simulated for a vector, $\lceil q/(W-1) \rceil$ passes are required.
- Since faults are hard coded into bits during simulation, we cannot use fault dropping.

The first method is called parallel fault simulation. So, here the main motivation is that compute a words are a contained multiple bits, typically 32 or 64 also in some modern machines.

So, here we take advantage of multi bit representation of data, and also we know that we have and, or this kind of logical operation which are available in the instruction set while even in the even in the language like c, we have the logical and ampersand logical or the bar or not this kind of operations which you can carry out on an entire word; like say in c if I write a equal to b and c let us say, so b will be let us say a 32 bit word, c is a 30 bit word so what is carried out is bit by bit ending and you get the result A, this is A.

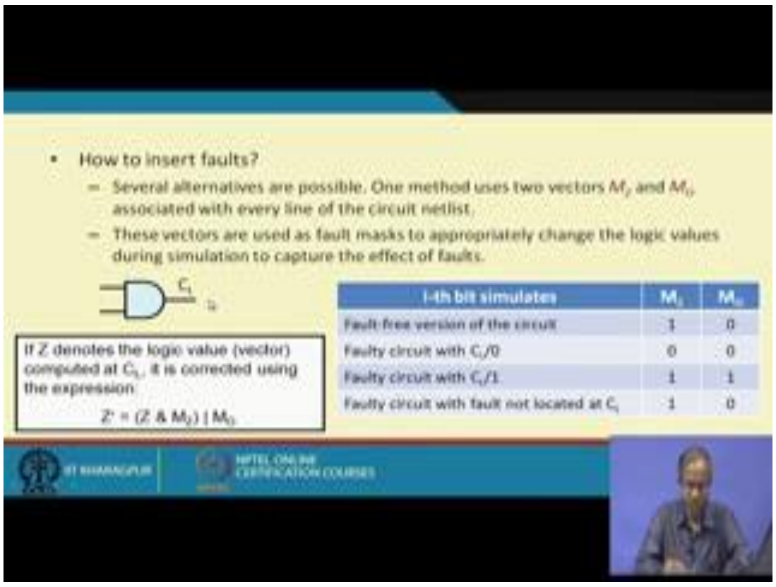
(Refer Slide Time: 11:51)



Now the thing is that in the single instruction you are actually carrying out 32 and operations. There are 32 bits, this and is carried out bit wise for each of the 32 bits. So, there is a parallelism involved here, this is exploited in this method. So, what we do? Here in every pass of the simulation we simulate the fault free circuits as well as W minus 1 of the faulty version. I shall we illustrating with an example what I am saying is that in my computer word I have W bits, I reserved 1 of the bit let us say the first bit for the fault free circuit, and remaining W minus 1 bit I reserve for representing the faulty behavior in presence of 1 of the faults.

So, at a time I can represent the behavior for W minus 1 faults. So, if there are more number of faults, I have to repeat this multiple times. So, this what we do in 1 pass, if there are total of q faults, so at any single pass I can simulate with W minus 1 faults, so I need q divide by W minus 1 ceiling of that so many passes. And here as I said in this W minus 1 bits where hard coding the faults into the bits, that is why we cannot use fault dropping we cannot remove a fault just like that. So, here also we have to simulate with all the faults.

(Refer Slide Time: 14:02)



- How to insert faults?
  - Several alternatives are possible. One method uses two vectors M_Z and M_O associated with every line of the circuit netlist.
  - These vectors are used as fault masks to appropriately change the logic values during simulation to capture the effect of faults.

If Z denotes the logic value (vector) computed at C_L, it is corrected using the expression

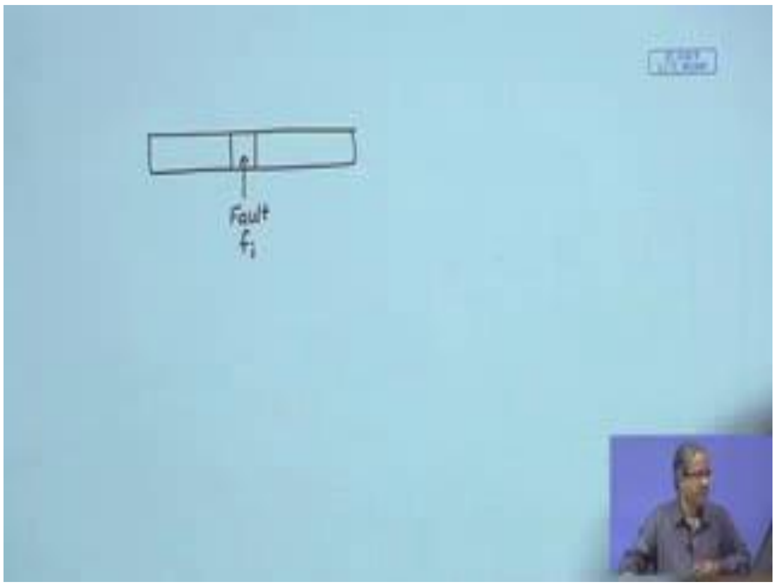
$$Z' = (Z \,\&\, M_Z) \mid M_O$$

i-th bit simulates	M_Z	M_O
Fault-free version of the circuit	1	0
Faulty circuit with C_L/0	0	0
Faulty circuit with C_L/1	1	1
Faulty circuit with fault not located at C_L	1	0

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let us see with the help of an example, but let us see how we insert faults. There are many methods which have been proposed, so here we are explaining 1 of the methods. So, we illustrate with a particular gate; let us say I have a gate in my circuits, whose output line I call it a C l, here what you saying is that with every line we associate 2 vectors M z and M o, these are used to insert the effect of faults and just one thing you remember, so whenever we talk about the vectors, let us say I have a vector like this. So, any particular bit of this vector will indicate the effect of fault f i.

(Refer Slide Time: 14:41)



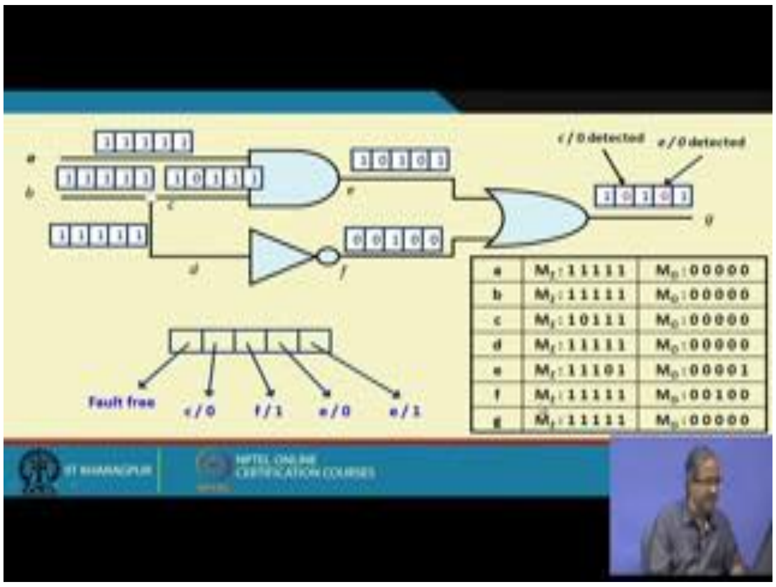
So, fault f i occurs what will be the value logic value on that line? Similarly the others bits will indicate effect of some other faults just remember this. So, here we are saying how these bits are assigned. So, what we are saying is that the rule is something like this, suppose this is an AND gate, so I have the vectors corresponding to my 2 input lines, I first compute and of those 2 vectors let us call it Z, Z denote the logic value computed at C l.

Now, after I compute Z, I make corrections or modifications to the bits to incorporate the effect of the faults here. What I do? I and Z with M Z, then or with M o and this is my modified logic expression or the word at the output l. Let us see how this M Z and M o bits are assigned this is explained here. So, we are talking of the ith bit. So, ith bit indicates the simulated value corresponding to fault f i. So, if it is fault free version; that means, I am talking about the first bit, then I said the bit in M Z to 1, M o to 0. Let us see what happens. So, if I and something with 1, it remains the same thing if I or with 0 it remains the same. So, I do not make any changes Z remains Z, this is for the fault free version. Similarly if the fault occurs somewhere else not in this gate, with fault not located at C l, then also I do not make any change M Z is 1 M o is 0.

So, here we make changes only when there is either a stuck at 0 or stuck at 1 fault on C l. So, if there is a stuck at 0 fault on C l, we make the corresponding bits M Z, M o both 0 0 why? If I do this Z ampersand 0, it become 0 or 0, it becomes 0. So, forcibly I am making that bit 0 right, to simulate stuck at 0. Similarly to simulate stuck at 1 I make both 1 1. C l is the, that is M Z, 1 1 means here this M Z can be 0 also does not matter 0 or 1 1.

So, here suppose it is 1 1. So, I and Z with 1 it becomes Z, I or with 1 so it is forcibly becoming 1. Another or with 1 is 1, so I am simulating a stuck at 1. So, if there is this ith bit corresponds to a stuck at 0 or 1 fault on this particular line, then only we change the corresponding bits like this otherwise we said the bits as 1 0 and 1 0 right. Let us take an example; suppose we have a simple example consisting of 3 gates - AND gate, NOR gate and an OR gate.

(Refer Slide Time: 18:11)



a	M_Z : 11111	M_O : 00000
b	M_Z : 11111	M_O : 00000
c	M_Z : 10111	M_O : 00000
d	M_Z : 11111	M_O : 00000
e	M_Z : 11101	M_O : 00001
f	M_Z : 11111	M_O : 00100
g	M_Z : 11111	M_O : 00000

Just for the sake of example if suppose my work size is 5, first bit indicates fault free, next 4 bit indicate these faults. Suppose we want to simulated these 4 faults c stuck at 0, f stuck at 0, e stuck at 0, e stuck at 1. So, there are so many lines in the circuit a b c d e f and g; in this table we are showing the corresponding M Z and M o values, which we fix up at the beginning of the simulation. Let us see for a there is no fault on line a; so for a you see for all the bits it is 1 0, 1 0, 1 0 or 1 0, 1 0, 1 0. So, M Z is all 1, M o is all 0.

Similarly b, there is no fault in this list; so b is also like that 1 0 1 0 all 1 all 0. But when you come to c you see that the second bit simulates the fault c stuck at 0. So, you see for the second bit we have made it 0 and 0, but the rest are 1 0 1 0 1 0 1 0 d again there is no fault, so d is all 1 0; e in e both faults are there the last 2 bits this is e stuck at 0 this is e stuck at 1. So, you see for e this last but 1 bit we put 0 and 0 to simulate stuck at 0, and last bit 1 and 1 to simulate stuck at 1, let us start 1 0 1 0 1 0.

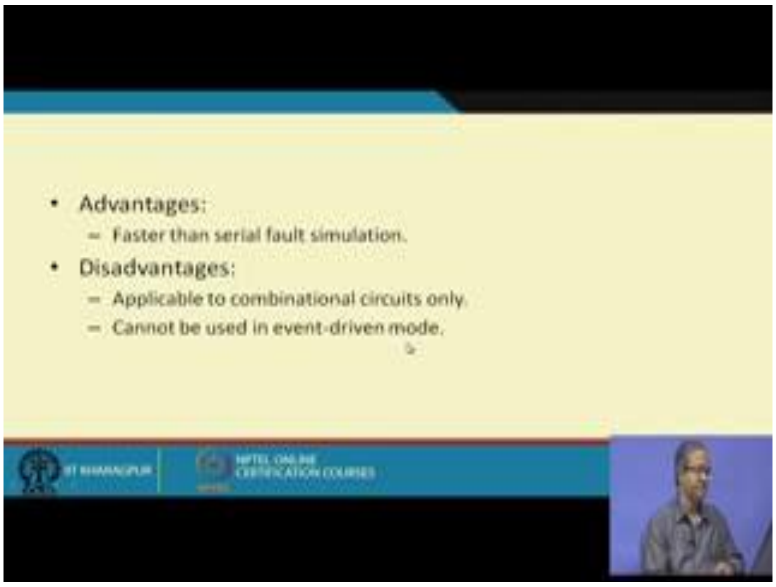
In f the middle bit is stuck at 1, so you see middle bit is 1 and 1; g no fault all 1 0 right. So, we fix these vectors a priori, then suppose we want to simulate with a 1 b 1. So, we start with a vector where a is 1 for all the cases, b is also 1 for all the cases. So, it is 1 for the fault free circuit, also for the faulty circuits we are applying the same inputs, a is 1 all 1, b is all 1. Now you progressively carry out simulation for the c; this is a fanout connections, for d this will be start here and here in d what you do? You just recall the equation and M Z or M o right. So, this 1 1 which is coming, and M Z all 1 same or M o all 0 same, so it does not change it remains all ones.

Let us come to c, this is also fanout branch that the same 1 1 1 1 1 will be Z here, Z and this. So, this bit becomes 0 or this so this second bit becomes 0. Now when you take and you do a bit by bit end of this 2 vectors 1 1 1 1 1 and 1 0 1 11. So, it becomes 1 0 1 1 1 that is Z, then you make correction with e and with this, this bit also become 0 or with that that bit becomes 1 is already 1. So, now, e becomes 1 0 1 0 1 0 1.

Similarly, there is a not gate 1 1 1 1 1 becomes 0 0 0 0 0; then inject the effect of fault and with this and or with this, this middle bit becomes 1, middle bit becomes 1 in finally, or this and this or no fault in g it becomes this. So, in the process when you finally, get the word at the output you see that your fault free output is 1, you check where are the 0 bits you see there are 2 zeros, which means this 0 corresponds to c stuck at 0, this 0 corresponds to e stuck at 0, which means both c stuck at 0 and e stuck at 0 are getting detected by this vector, and all this 4 faults we have simulated together in parallel in this 1 pass.

So, you see if it is a 32 bit word, you can simulate 31 faults together right, this is an advantage. But the over rate is that you have to use this M Z and M o and for evaluating each gate normally you would be requiring only and for an AND gate, but here even after evaluation you need an additional AND, and an additional or operation for this correction this is the over head; so 2 additional operations on every line using M Z and M o.

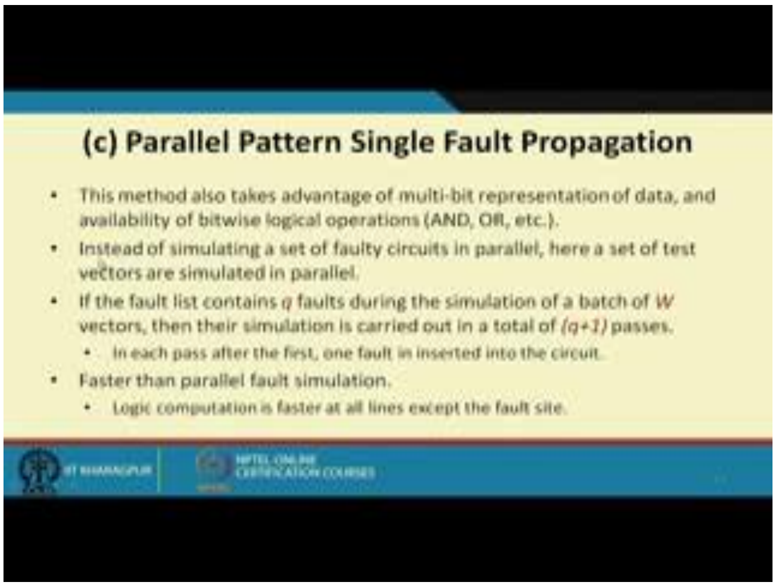
(Refer Slide Time: 23:16)



- Advantages:
  - Faster than serial fault simulation.
- Disadvantages:
  - Applicable to combinational circuits only.
  - Cannot be used in event-driven mode.

So, this method is clearly faster than serial for simulation, but the disadvantage is that it is applicable to combination circuits only; and it cannot be used event driven mode, because you are injecting different faults in different bits you cannot identify the events like that that I have to simulate this part not that part, but to simulate for all the faults.

(Refer Slide Time: 23:44)



(c) Parallel Pattern Single Fault Propagation

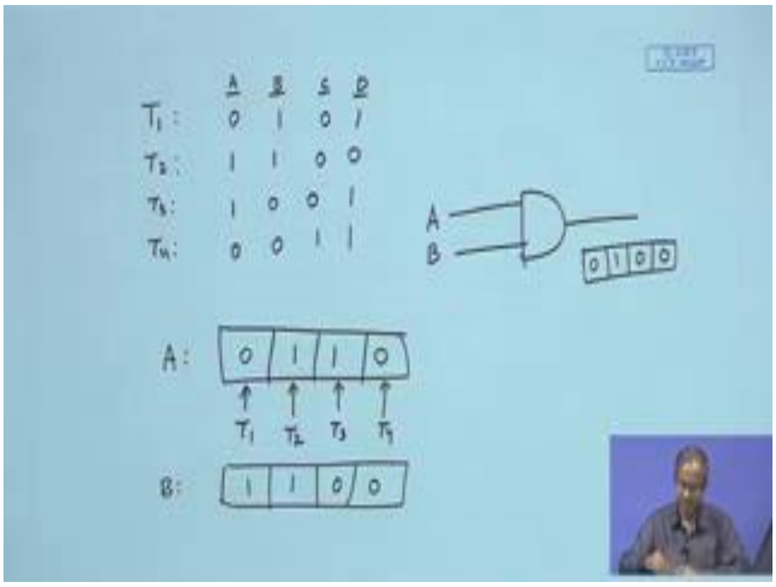
- This method also takes advantage of multi-bit representation of data, and availability of bitwise logical operations (AND, OR, etc.).
- Instead of simulating a set of faulty circuits in parallel, here a set of test vectors are simulated in parallel.
- If the fault list contains *q* faults during the simulation of a batch of *W* vectors, then their simulation is carried out in a total of *(q+1)* passes.
  - In each pass after the first, one fault in inserted into the circuit.
- Faster than parallel fault simulation.
  - Logic computation is faster at all lines except the fault site.

Now, these same methods will if you can modify a little bit you get a third approach, let us try to see. In the parallel faults simulation what we were doing? We were simulating many faults together with one test vector at a time. So, how were simulating many faults together by having M Z, M o in all the lines. So, what was the difficulty or the drawback? And every line computation we needed one additional and one additional or operation during the simulation processing. But suppose we do it the other way round, what is other way round? We inject one fault at a time, one fault at a time means we can modify the netlist just like we mentioned earlier; and we do parallel simulation, but now the different bits will indicate the different test vectors not the different faults.

So, we are simulating with different test vectors together. So, what is the advantage? Now we are doing away with those M Z and M o, we have no M Z M o. Where injecting 1 fault at a time means we are actually modifying the circuits netlist, we are packing the bits with the different vectors and you are simulating them together; result is that this will run faster as compared to the earlier method parallel fault simulation, because we are not requiring to store and process the M Z and M o vectors, right.

So, here as I said instead of simulating a faulty set of faulty circuits in parallel, we simulate a set of test vectors in parallel. So, if we are fault list consist of q faults, so at a time you are simulating 1 fault; so totally you need q plus 1 1 for the fault free simulation, and q for this q faults and in each of the passes you are simulating with all the test vectors. Like how you are simulating with test vectors I am giving you a small example, suppose I have a 4 bit circuits, for input circuits let us say T 1, T 2, T 3, T 4.

(Refer Slide Time: 26:10)



	A	B	C	D
T_1:	0	1	0	1
T_2:	1	1	0	0
T_3:	1	0	0	1
T_4:	0	0	1	1

Let us say the inputs are A B C D, the circuits inputs are A B C D. Suppose the test vectors are as follows T 1 is 0 1 0 1; T 2 is 1 1 0 0 1 0 0 1 0 0 1 1 let us say. So, now, the when we pack out word let us say we have a 4 bit word in this case, the first bit will indicate T 1, second bit will indicate T 2, third bit will indicate T 3, 4 bit will indicate T 4. So, there will be 1 word for lying A, 1 word for lying B and so on. So, for lying A the word will be 0 1 1 0; similarly for line B the word will be 1 1 0 0.

So, in my circuits if I have a scenario where there is a let us say and gate which is driving A and B I can directly take the AND of these 2 without any corrective steps; simply bit by bit and it becomes 0 1 0 0, so the output vector will be 0 1 0 and 0. This will much faster as you can see. So, this is clearly faster than parallel logic simulation, because except the faults sides other parts we have not making any changes, in this way you comes to the end of this particular lecture.

In the next lecture we shall be looking at a couple of you can very interesting and you can say more powerful techniques for fault simulation, where all the faults are simulated together. There is a very well defined technique that we shall be presenting there, where the faults are all handled together and another thing also ensured the process will be event driven, we will not process the parts of the circuits where there are no changes that will make the process even faster. So, this we shall be discussing in our next lecture.

Thank you.