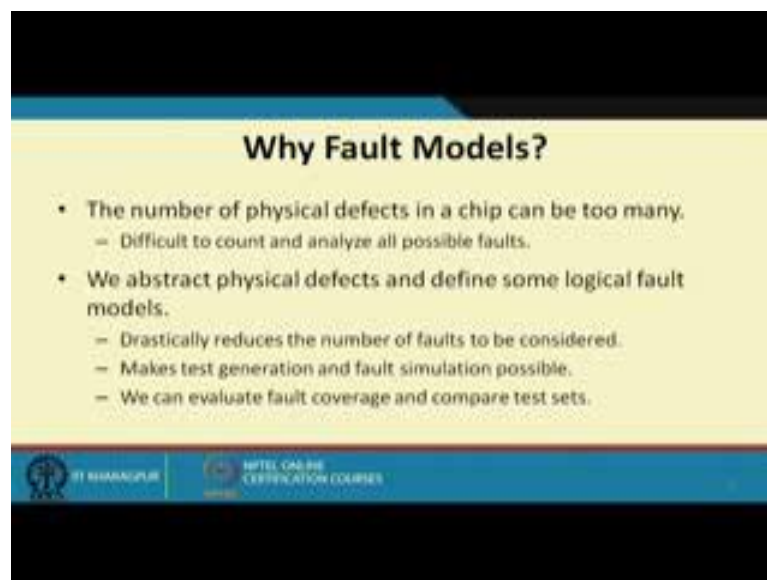


VLSI Physical Design
Prof. Indranil Sengupta
Department of computer Science and Engineering
Indian Institute Technology, Kharagpur

Lecture - 49
Fault Modelling (Part 1)

So, in this lecture we start with a discussion on fault modelling as I said. So, fault modelling as I said this is the first step and a very important step in the total testing process.

(Refer Slide Time: 00:36)



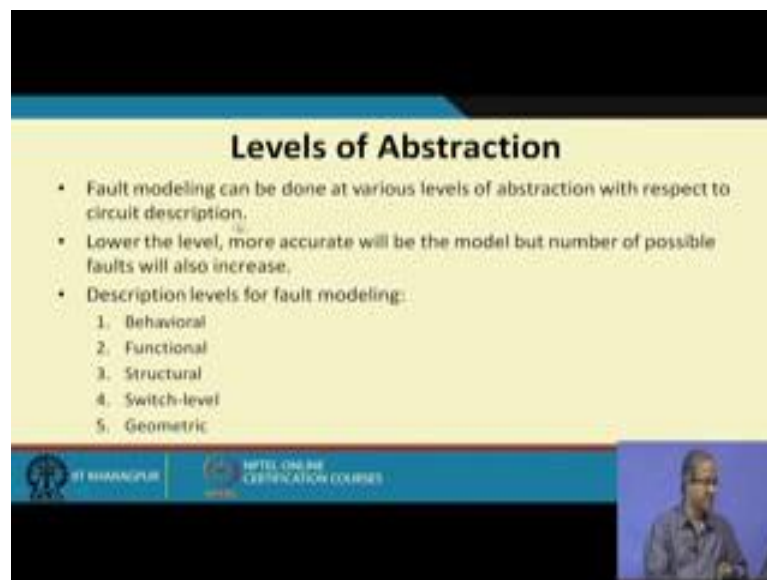
So, let us first try to see once more that why we need a fault model. So, you know last lecture we mentioned this briefly, see we mentioned that the number of physical defects in a chip can be too many. In fact, it can be infinitely large. So, it is difficult; difficult is a very you can say loose term, you should say it is impossible; it is impossible to count and analyse all possible faults, because the number of possible defects can be infinite.

So, what we do we abstract this physical defects and we define some fault models which are sometimes called logical fault models, because these are not faults which are not actually happening in the chip, we are logically assuming or thinking that the chip is behaving as if this kind of fault has occurred in presence of that defect or error right. So, by doing this what are the advantages we gain? Number of fault to be considered can be reduced drastically, by doing that the processes of test generation and fault simulation

can be simplified and it can be made possible, because if we do not provide with a list of faults, what will the test generator do? Test generator does not know that which faults you are trying to detect, right number of faults can be too many.

So, and also you can evaluate fault coverage and you can compare alternate sets of test vectors like you have say 2 test cells t 1 and t 2, you want to find out which one of them is better. You can do some experiment and see that t 1 can detect let say 85 percent of the faults, and t 2 can detect 90 percent of the faults; so t 2 will be better. So, unless you have a fault model you cannot do this kind of quantitative analysis right.

(Refer Slide Time: 02:45)



The slide is titled "Levels of Abstraction" and contains the following text:

- Fault modeling can be done at various levels of abstraction with respect to circuit description.
- Lower the level, more accurate will be the model but number of possible faults will also increase.
- Description levels for fault modeling:
 1. Behavioral
 2. Functional
 3. Structural
 4. Switch-level
 5. Geometric

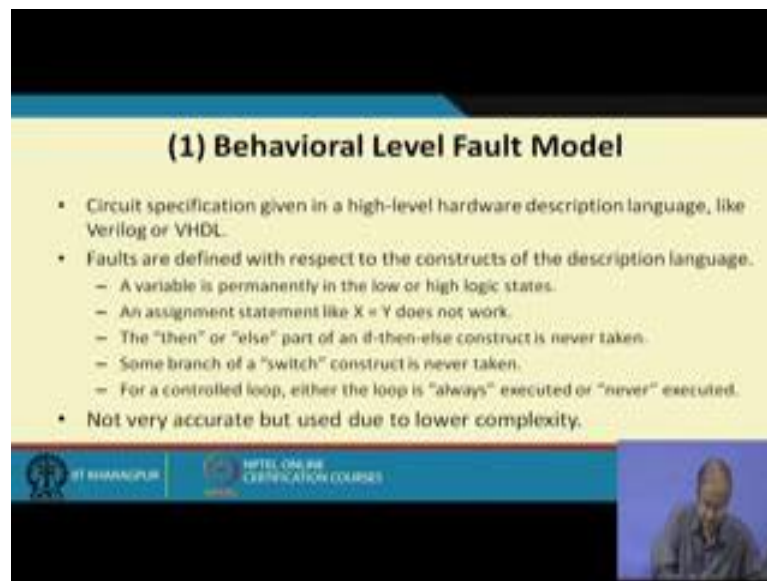
The slide also features the NPTEL logo and the text "NPTEL ONLINE CERTIFICATION COURSES" at the bottom left, and a small video inset of a speaker at the bottom right.

Now, this fault models can be defined at several levels of abstraction; just like a circuit can be described at several levels of abstraction. So, at every such levels you can define a set of faults and do some abstraction.

Now, clearly so as you go towards the lower levels more detail description, your model will become more accurate, but the number of faults can also go on increasing. The typical description levels are as follows. So, the highest level you can think of the behaviour specification, where a circuit is described in terms of its behaviour in a high level language like VHDL verilog, or other languages are there also. So, you can think what kind of faults can take place at that level of discription, then your functional level where you are looking a the functional blocks some of the funtional blocks at the rtl level typically, then structural level typically this works at the gate level.

So, at the level of gate level netlist, what kind of faults can be there. Switch level means transistor level, and geometric level means at the level of the layouts. So, as move down your fault models will become more and more realistic, but the trouble is that the number of circuit components is also going up very rapidly. Number of functional blocks and number of transistors, so you cannot really compare them, number of transistors will be much much higher. So, number of possible faults will also be much higher, but of course, they will be more accurate if you can model the faults at that level.

(Refer Slide Time: 04:32)



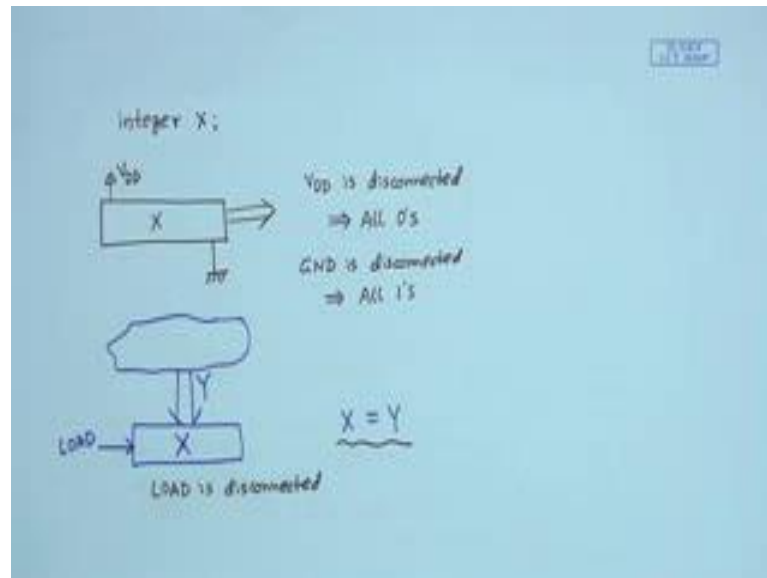
(1) Behavioral Level Fault Model

- Circuit specification given in a high-level hardware description language, like Verilog or VHDL.
- Faults are defined with respect to the constructs of the description language.
 - A variable is permanently in the low or high logic states.
 - An assignment statement like $X = Y$ does not work.
 - The "then" or "else" part of an if-then-else construct is never taken.
 - Some branch of a "switch" construct is never taken.
 - For a controlled loop, either the loop is "always" executed or "never" executed.
- Not very accurate but used due to lower complexity.

IT BHAVANIPUR | NPTEL ONLINE CERTIFICATION COURSES

So, let us quickly look at the various fault models at this levels. So, the highest level behaviour level fault model; here as I had said the circuit specification is specified in some high level design discription language like verilog ar VHDL, and we define some faults which are with respect to the constructs of this language like I am explaining a few of them, like you say some variable is permanently in the low or high level states like let us try to just explain it.

(Refer Slide Time: 05:19)



In verilog you can define a variable right. So, you can define the integer, you can define it as a bit. Let say whenever you define as a let say I define an integer X let say, but in terms of the hardware if I look this X will be mapped into a register, let us say its a 32 width, X will be mapped here. Now in every register in terms of the circuit connections there will be a connection to power supply, there will be another connection to ground; lets say the outputs are coming here. Now lets think of some faults at this level, if V DD is disconnected due to some reason. So, V DD is disconnected. So, if V DD is disconnected if you read out the contents of the flip flops, all of the bits will be looking at 0.

So, you will get all 0's. Similarly if ground is disconnected, and you try to read out the outputs you will see that all the outputs are having V DD that mean all 1's. So, whatever fault model you are assuming there is a correspondence from the correspond means low level hardware realization of that right, then the second such fault which is mentioned here it says that an assignment statement X equal to Y does not work; which means you are giving this assignments, but the value of the expression Y is not getting assigned to X, X remains at it own value. Well again in terms of the hardware what can be the reason let say.

Here I have a register which is suppose to hold the value of let say X. So, I have a statement X equal to Y. So, there is a circuit which is calculating the value of Y and it is

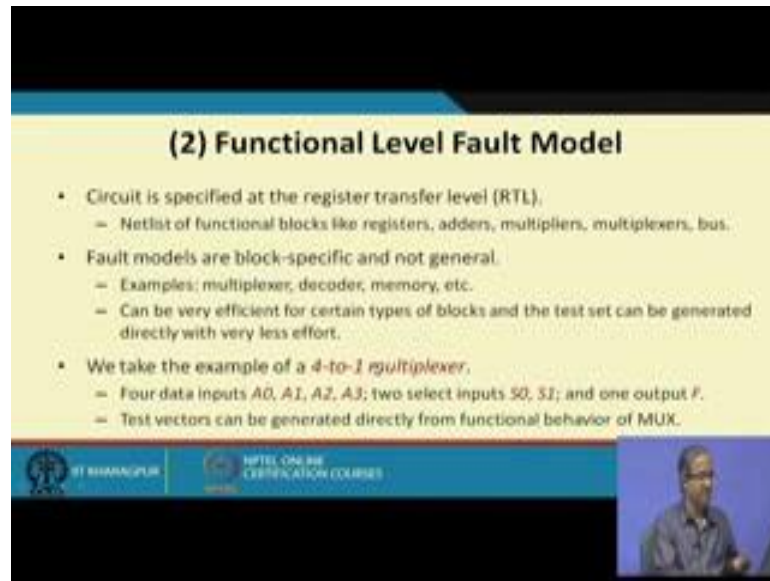
feeding here, and naturally with each register there has to be a some kind of a load control input. Now you think of a fault where this load is disconnected; somehow this load connection is not getting connected. So, what will happen? This Y will never get loaded. So, in terms of the assignment this assignment will never occur right? Then let us come to the next one, lets say for an if then else kind of a construct, this says that either the then or the else path is never taken.

Similar is a case of a multi Y if then else like a switch construct, some branch is never taken. So, you see here also the problem is some what similar; like when we have a multi way branch. So, your are going either here or there or there, there can be some kind of a demultiplexer, which will be selecting which path to follow. Now because of some fault in the demultiplexer, some of the lines may be always getting selected or some the lines are getting never selected right? Due to some problems in the control circuit, this kind of things may happen.

So, here we are abstracting these faults as a fault in the if then eslse construct or as a switch construct. Similarly for a control loop like a let say a loop which is suppose to execute certain numbers of times like a fall loop. So, loop is either always executed or never executed. So, again due to some error in the control circuit which controls the loop right? Normally the loop counter will be decremented and there will a checking for 0, may be the checking for 0 circuit is not working. So, it is always saying that it is not 0. So, the loop will go on and on and on; and if due to a fault if it says that that it is always equal to 0 then the loop will never go it will just immediately come out right.

So, such things are modelled in a behavioral level fault model at a high level, clearly these are not very accurate in terms of the final circuit realization, but some industry use these methods because they have much lower complexity of course, in congestion with the other fault models also fine. Next let us come to functional fault model where you assume that our circuit is specified at the register transfer level, where we have functional blocks like registers, adder, multipliers, multiplexers, bus and so on.

(Refer Slide Time: 10:18)



(2) Functional Level Fault Model

- Circuit is specified at the register transfer level (RTL).
 - Netlist of functional blocks like registers, adders, multipliers, multiplexers, bus.
- Fault models are block-specific and not general.
 - Examples: multiplexer, decoder, memory, etc.
 - Can be very efficient for certain types of blocks and the test set can be generated directly with very less effort.
- We take the example of a 4-to-1 multiplexer.
 - Four data inputs A_0, A_1, A_2, A_3 ; two select inputs S_0, S_1 ; and one output F .
 - Test vectors can be generated directly from functional behavior of MUX.

IT BHARANGPUR | INTEL ONE-ONE CERTIFICATION COURSE

(A small video inset shows a man in a blue shirt speaking.)

Now, one problem with the functional level fault model is that, it is not general means some fault model that can be applied to a multiplexer; you cannot apply to a decoder, you cannot apply to a memory. So, there will be some fault model that is specific to a multiplexer, there is a fault model that is specific to a memory and so on.

But once you can define these fault models they can be very efficient, and the test set can be generated absolutely directly. I will give an example for a multiplexer that how it works. So, we take an example of a 4-to-1 multiplexer. So, for a 4-to-1 multiplexer let say that the 4 data inputs are A_0, A_1, A_2 and A_3 there are getting selected using the select input values S_0 and S_1 and there is an output F . So, we shall show now that how you can generate the test vectors directly, by looking at the functional behaviour of a multiplexer.

(Refer Slide Time: 11:42)

A0	A1	A2	A3	S0	S1	F
0	1	1	1	0	0	0
1	0	0	0	0	0	1
1	0	1	1	1	0	0
0	1	0	0	1	0	1
1	1	0	1	0	1	0
0	0	1	0	0	1	1
1	1	1	0	1	1	0
0	0	0	1	1	1	1

- For a 2^n -to-1 MUX, 2^{n+1} test patterns are required.
- All functional faults in the MUX can be detected.
 - An input line is incorrectly selected.
 - Some line is fixed at 0 or 1.

So, it is shown in this slide; see here we see our multiplexer these are the 4 inputs the select lines and the output. Now here I am directly writing down the test vectors that I require. So, let's try to understand so how you are doing it. First thing is that we are using different values of S0, S1 combinations. So, first 2 rows have select line 00, second 2 rows are 01, next 2 rows are 10 and last rows are 11. So, the first 2 rows are suppose to select A0, next 2 rows are suppose to select A1, next 2 rows suppose to select A2, and the last 2 rows suppose to select A3 now let us see. So, I apply 00 to S0 and S1.

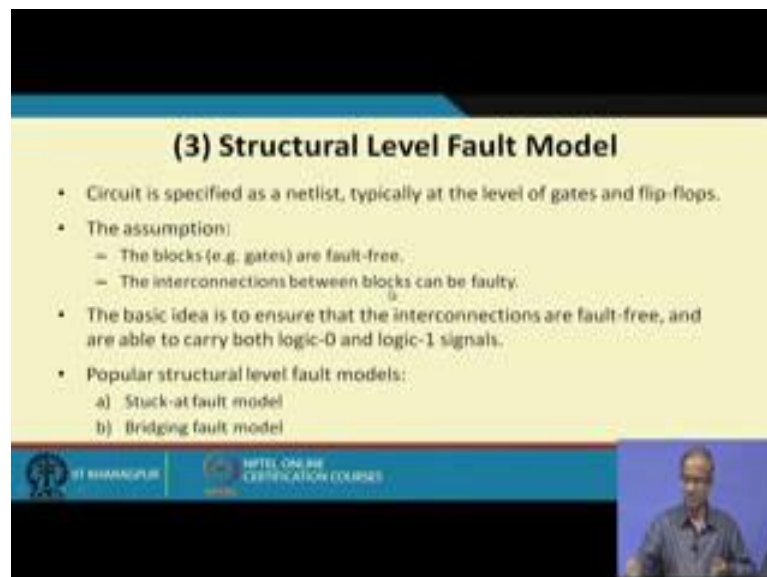
So, now A0 is selected. So, what I can say that I apply A0 equal to 0 and check whether the output is 0 or not. I apply A0 equal to 1 and check whether the output is 1 or not right, but why I am applying the reverse logic value to the other 3 inputs? Because you can argue that when I am selecting A0, this A1, A2, A3 should be known to us right, but I am applying reverse value because if there is a fault like this, that I am trying to select line A0, but due to the fault either A1 or A2 or A3 is getting selected. So, there there will be a miss match in the output. So, normally A0 is suppose to be selected I am getting 0, but if by mistake A1, A2, A3 is getting selected I will get 1 so I can detect the fault.

Similarly for this case normally it is 1 and if some other inputs are getting selected, I will get a 0. So, always I means we detect a fault if we can detect a miss match in the output. So, same thing is happening with the other pairs of rows, in the next pair we are selecting

A 1 by applying S 1 0 S and S 0 1. So, you see in one case I am applying A 0, A 1 equal to 0 other case A 1 equal to 1, the others are reverse; 1 1 1 0 0 similarly here we select A 2, 0 1 here we select A 3, 0 1. So, you see we can directly we have written down this 8 test vectors, that can detect these functional faults in the multiplexer. You see this is exactly what you want for a multiplexer to work correctly you do not need anything more than this, and another thing you see the original multiplexer had 6 inputs right, so 2^6 was 64.

So, instead of 64 test patterns I did only 8 test patterns, this is one of the major objectives of having a test generation method, where the number of test to be applied can be drastically reduced. So, to generalise if we have a 2^n to 1 multiplexer, we will need 2^{n+1} test patterns, and all functional faults as I had said can be detected like some input line is incorrectly selected, some lines are constantly fixed at 0 or 1 all these faults can be detected by this simple set of tests. But you can see this strategy you cannot extend for a decoder also any other functional block, for that you will again have to do some kind of similar functional behaviour analysis and you will have to find out the test vectors, but once you are able to do it is very easy right.

(Refer Slide Time: 15:33)



(3) Structural Level Fault Model

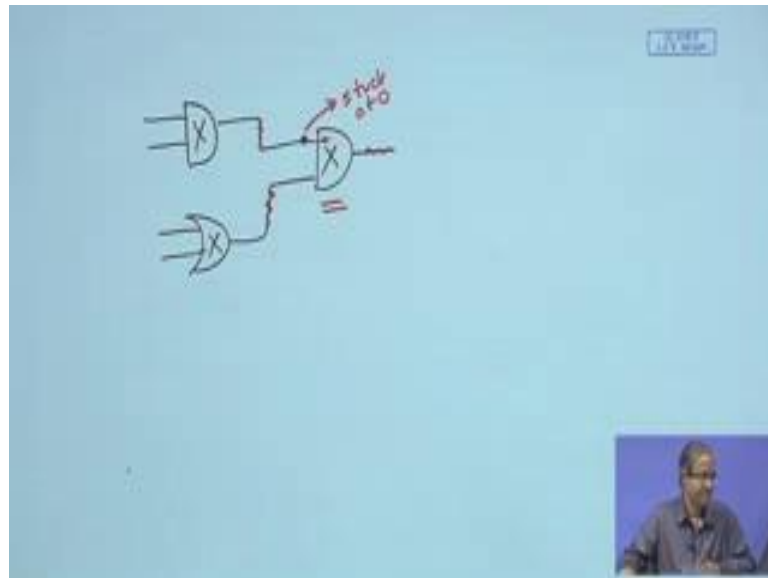
- Circuit is specified as a netlist, typically at the level of gates and flip-flops.
- The assumption:
 - The blocks (e.g. gates) are fault-free.
 - The interconnections between blocks can be faulty.
- The basic idea is to ensure that the interconnections are fault-free, and are able to carry both logic-0 and logic-1 signals.
- Popular structural level fault models:
 - a) Stuck-at fault model
 - b) Bridging fault model

The slide also features a video inset of a presenter in the bottom right corner and logos for IIT BHARANGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom.

Now, let's come to the structural level fault model, which is perhaps the most widely used and common. Here the circuit is specified as a netlist means an interconnection of some blocks, typically at the level of gates and flip flops. Because most of the structural fault

models that have been proposed and discussed, they consider the circuit as a netlist of gates or for sequential circuits also flip flops right. Now here the interesting thing is that we assume that the functional blocks does not contain any faults, the gates are fault free, the interconnection between the blocks can be faulty. So, what we mean is this?

(Refer Slide Time: 16:25)



Let say we have a circuit, lets take a 3 gates circuits simple. So, according to this fault model we are saying that there can be no faults inside these gates, these gates are perfectly alright. Faults can only happen at the interconnections; some failure can happen in this interconnection or this interconnection or this interconnection. You see because of this thing what you can say is that may be for this gate let say if we look at this gate, we can say that one of the line of this gate is permanently stuck at 0 let say. Stuck at 0 means it is always at 0, this can be regarded as a fault due to this interconnection, the interconnection line accidentally there is a failure, there by this line is always at 0 or similarly here, but again you see I repeat these are all manifestation of physical faults, I am saying that there is a stuck at 0 fault here, but it does not mean that physically this wire is faulty; physically it might happen that that there is some fault inside this gate, but it is behaving as if this line is always at 0 fine.

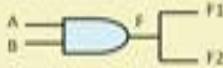
So, the assumptions are like this, the blocks are fault free and interconnections can be faulty. So, the basic idea is to ensure that there is no fault in the interconnections and they can be able to carry both logic 0 and logic 1 signals. There are 2 popular structural

fault models which are used; the first one is a most popular, this is called stuck at fault model and the second one is called bridging fault model.

(Refer Slide Time: 18:20)

(a) Stuck-at Fault Model

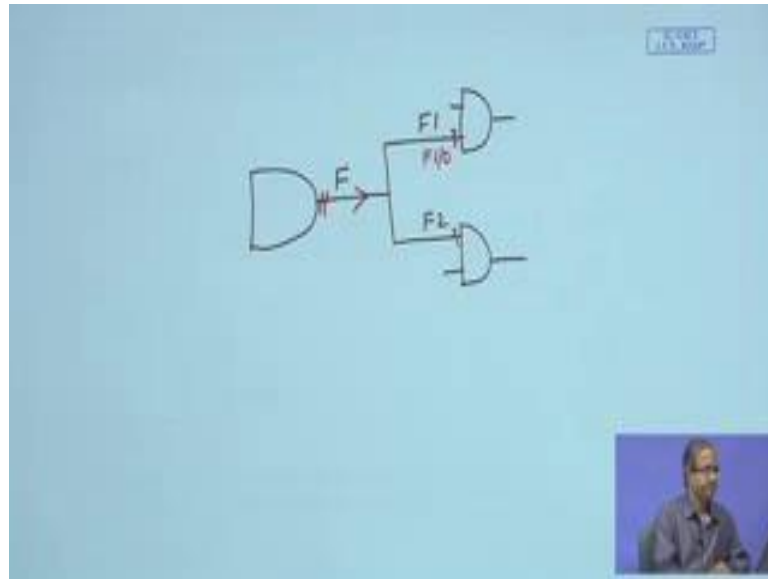
- Here we assume that some of the circuit lines are permanently fixed at logic-0 or logic-1 due to some failures.
- Very popular fault model, as it can model many realistic physical failures.
 - Technology independent.
 - Faults on a line A are denoted as: A s-a-0 or $A/0$, and A s-a-1 or $A/1$.
 - Fanout stems and fanout branches are considered as separate lines. Why?



IT BHARATPUR NPTEL ONLINE CERTIFICATION COURSES

Let see stuck at fault model is conceptually very simple, it says that some of the circuits line are permanently fixed at logic 0 or logic 1 means we called them stuck at 0 or stuck at 1. This is a very popular fault model and it has been analysed and found that it can actually model many realistic physical failures and of course, this fault does not assume that whether you are using CMOS or TTL or any other technology, this is technology independent and notationally for a particular line lets say A faults are denoted as either like this A stuck at 0 or you simply write A slash 0 or A stuck at 1 or A slash 1. Now one point you note is that in this fault model the fanout stems and fanout branches are considered as separate lines, like in this diagrams F is a fanout stem, and F 1, F 2 are fanout branches, they are consider a separate lines let see why, lets try to explain.

(Refer Slide Time: 19:41)



So, we have this gate this output is F , and this output is going to 2 fanout branches F_1 and F_2 ; and this fanout branches are possibly going to the input of some other gates right this is one gate, there another gate there. Now let see say mean electrically, electrically speaking F , F_1 , F_2 are identical.

So, whatever voltage should be there same voltage should come in F_1 and F_2 . But you see there can be some faults which are here may be lets says there is a disconnection here, this wire which is suppose to be connected to the this inout of the this and gate there is a disconnection here, because of which this gate input is behaving as if it is at 0. So, I denote it as F_1 stuck at 0 fault, but clearly this fault will not affect F_2 or it will not effect F . This will only effect the input of this gate; that means, F_1 similarly there can be a fault at F_2 this will be independent of F_1 or F , but; however, if there is a fault at F , that fault will be going to both F_1 and F_2 ; because the direction of signal transmission is this alright. So, because of this the faults of the fanout stem and the branches they are considered as 3 different faults. So, let us come back.

(Refer Slide Time: 21:18)

A 2-input XOR function realized using 2-input NAND gates

- 12 lines in the circuit
- 24 stuck-at faults:
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

BT BHARANGPUR INTEL ONLINE CERTIFICATION COURSES

Lets take an example; this is a nand realization of a 2 input xor function. So, there are 4 2 input nand gates. So, as you can see there is a fanout connection from here, one from here and another from here. So, I have named the different lines accordingly this is A and the 2 fanout branches are called A 1, A 2. This is B, B 1 B 2 and this is C and this are C 1 C 2. So, if you count that how many lines are there in a circuit, you can see 1 2 3 4 5 6 7 8 9 10 11 and 12. So, there are total 12 lines in the cicuit. So, now, if you want to count that how many stuck at faults are there total. So, each of the lines can be stuck at 0, or stuck at 1 like A can be 0 or 1, B can be 0 or 1, A 1 can be 0 or 1. So, I have made the fanout branches and stems separate A 2 can be 0 or 1.

(Refer Slide Time: 22:44)

• **Single stuck-at fault**

- Only one line of the circuit has a stuck-at fault at any given time.
- Most widely used fault model in the industry.
- For a circuit with k lines, total number of single stuck-at faults is $2k$.

$k = 12 \rightarrow$ Number of single stuck-at faults = 24

So, in this way for the 12 lines, we can enumerate 24 possible stuck at faults right. So, for this circuit we can count and tell that there are 24 stuck at faults. Let us look at some specifics; there are 2 variations of stuck at fault model, one is called single stuck at fault which is the most popularly used; because of its simplicity. This fault model says that only one line of the circuit can be faulty at a given time. So, although there are 12 lines here, but at a time 1 line will be faulty; this fault model is most widely used in the industry, and you can clearly you can count that because each of the lines can have 2 faults stuck at 0 or stuck at 1. So, how many single stuck at faults can be there? We have already counted earlier there can be a 24 such faults so that will 24. So, it will 2 into k. So, if there are 12 lines, and if you assume that 1 fault is occurring at a time. So, there will be $2k$ possible faults. So, in this case number of single stuck at faults will be 24.

(Refer Slide Time: 23:56)

• **Multiple stuck-at fault**

- Any number of circuit lines can have stuck-at faults at any given time.
- For a circuit with k lines, total number of multiple stuck-at faults is $3^k - 1$.

$k = 12 \rightarrow$ Number of multiple stuck-at faults = $3^{12} - 1 = 5,31,440$

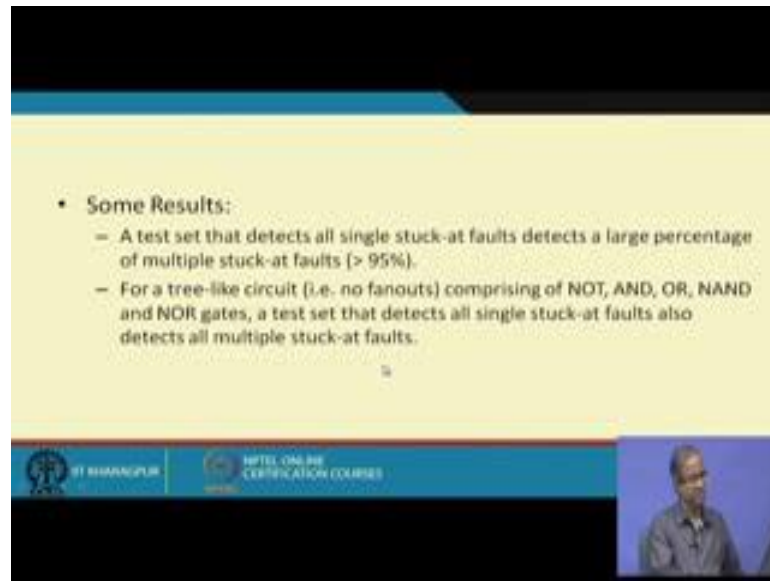
IT MANAGER | NPTEL ONLINE CERTIFICATION COURSES

Now, now if we remove the restriction of single fault occurring, we can have a any arbitrary number of faults happening, this is called multiple stuck at fault model. This says that any number of circuits lines can have such stuck at faults at a time.

So, here we have given expression if a circuit as k lines, total number of multiple stuck at faults will be $3^k - 1$; well how this number is coming? Let see we have a circuit there are k numbers of lines, look at each of the line one at a time, the first line can be in 3 states meaning it can be either good, it can be stuck at 0, it can be stuck at 1. The second line can also be in 3 states good, stuck at 0, stuck at 1. Third line can also be in 3 states good, stuck at 0, stuck at 1. So, there are k such lines. So, the total number of possibilities will be 3 into 3 into 3 k times, 3 to the power k . At out of this combinations one combination will denote all lines are fault free. So, that's why we subtract one from that 3 to the power k minus 1. So, many faulty situations can be there right, 3 to the power k minus 1.

So, you see that even for the small circuit. So, when you have k equal to 12, 3 to the power 12 minus 1 is becoming more than 5.3 lakhs. So, what will happen for the largest circuits, multiple stuck at faults will be virtual infinite right that is why people stick to a single stuck at fault and analyse them.

(Refer Slide Time: 25:51)



- Some Results:
 - A test set that detects all single stuck-at faults detects a large percentage of multiple stuck-at faults (> 95%).
 - For a tree-like circuit (i.e. no fanouts) comprising of NOT, AND, OR, NAND and NOR gates, a test set that detects all single stuck-at faults also detects all multiple stuck-at faults.

But there is an interesting result which can make us feel good. A test set that detects all single stuck at faults, will also detect a large percentage of multiple stuck at faults. It is at least greater than 95 percent this has been proved exponentially and also verified. And there are some results for some specific kinds of circuits like if a circuit does not have any fanout connections, this is called a tree like circuit with no exclusive or exclusive nor gate, and not and or nand nor gates. Then it it can be proved that any set of test vectors that detects all single stuck at faults, can also detect all multiple stuck at faults right. So, these results tell us that single stuck at fault is not that bad, if you can detect all single stuck at faults we can also detect a large percentage of multiple stuck at faults, in some cases all of them.

(Refer Slide Time: 26:54)

The slide is titled "Single Stuck-at Fault Testing Examples". It contains two examples of AND gates and their corresponding test vectors.

Example 1: 2-input AND gate

The diagram shows an AND gate with inputs A and B, and output F. The test vectors are listed as $T = \{01, 10, 11\}$.

- 01 : detects A/1 and F/1
- 10 : detects B/1 and F/1
- 11 : detects A/0, B/0 and F/0

Example 2: 4-input AND gate

The diagram shows an AND gate with inputs A, B, C, and D, and output F. The test vectors are listed as $T = \{0111, 1011, 1101, 1110, 1111\}$.

- 0111 : detects A/1 and F/1
- 1011 : detects B/1 and F/1
- 1101 : detects C/1 and F/1
- 1110 : detects D/1 and F/1
- 1111 : detects A/0, B/0, C/0, D/0 and F/0

At the bottom of the slide, there are logos for "VT SHARANGAPUR" and "INTEL ONLINE CERTIFICATION COURSE". A small video inset shows a person speaking.

Let us look at some examples, here we have a simple 2 input and gate we stick to single stuck at faults now.

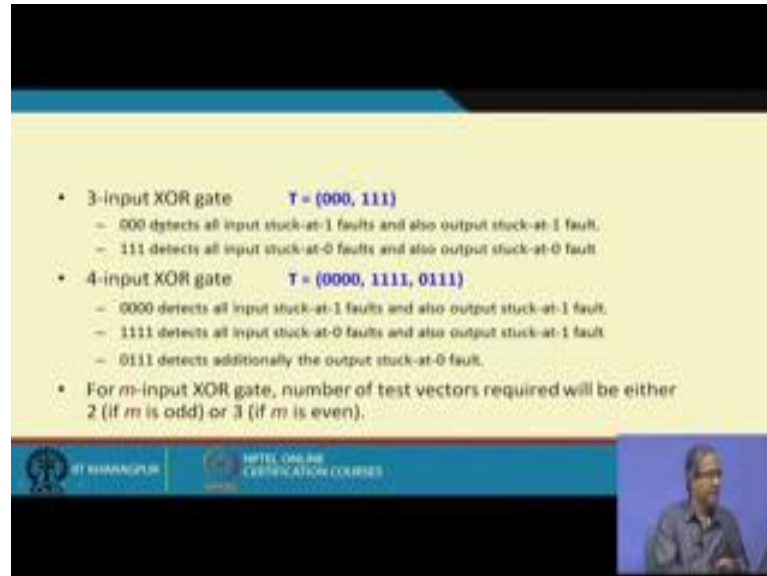
So, I say that to test this and gate I need 3 test vectors 0 1, 1 0 and 1 1; why? If we just look at it 0 1 if I apply 0 1, the output will be 0. So, what are the faults it can detect? It can detect A stuck at 1, because normally the output was 0, but if A is also fixed at 1, both will be 1 1 the output will change. Similarly if the output line is stuck at 1 that also will make the output change, this 1 0 similarly will detect fault on B, it will detect if B is stuck at 1 then the out will change and also F 1.

Well 0 0 test vector is useless for a and gate it will not detect any fault any single fault, because output is 0 in case of just only for F stuck at 1, but for faults in A there will be or B no change, but if I apply 1 1 then I can detect all these 3 faults because normally output will be 1 if any of these 3 fault are there output will become 0 right.

Now, lets go to a 4 input and gate, you see for 4 input and gate we need 5 test vectors, not instead of 16, 2 to the power 4. Here we require 1 less you see 0 1 1 1, 0 1 1 1 this will detect fault A stuck at 0, this can easily check if A stuck at 0 is there the output will change and also F stuck at 1. 1 0 1 1 will detect stuck at 1 on line B, 1 1 0 1 will detect stuck at 1 on line C and 1 1 1 0 will detect stuck at 1 on D; and similarly 1 1 1 1 will detect stuct at 0 on all these lines. So, only 5 test vectors are required. So, in general for

an n input and gate, you need n plus 1, so many test vectors and not 2 to the power n right.

(Refer Slide Time: 29:19)



- 3-input XOR gate $T = \{000, 111\}$
 - 000 detects all input stuck-at-1 faults and also output stuck-at-1 fault.
 - 111 detects all input stuck-at-0 faults and also output stuck-at-0 fault.
- 4-input XOR gate $T = \{0000, 1111, 0111\}$
 - 0000 detects all input stuck-at-1 faults and also output stuck-at-1 fault.
 - 1111 detects all input stuck-at-0 faults and also output stuck-at-1 fault.
 - 0111 detects additionally the output stuck-at-0 fault.
- For m -input XOR gate, number of test vectors required will be either 2 (if m is odd) or 3 (if m is even).

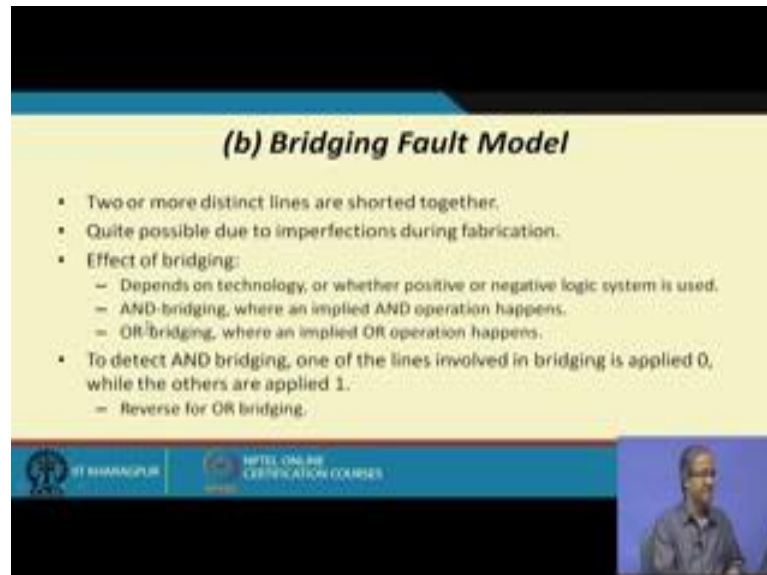
Let us take some examples 3 input exclusive XOR. A 3 input exclusive xor will recall only 2 test vectors, 0 0 0 and 1 1 1. See you think what an exclusive xor gates, an exclusive xor gate will have the output 1, if odd number of inputs are at 1 and the output will be 0 if even numbers of inputs are 0 right. Now here the first test vector is having even number of inputs as 0 as 1 and here is having odd number of input.

So, for first one out is 0, here output is 1. Now if any one of the input is having stuck at 1 fault, so number of 1 will change from even to odd. So, 0 0 0 can detect all input stuck at 1 faults and of course, also output stuck at 1 fault. Similarly 111 there is an odd number of 1, but if there is a stuck at 0 fault in any one of them, number of ones will become even, so output will become 0. So, 1 1 1 can detect all input stuck at 0 faults. Look at your 4 input xor gate well; unfortunately here you need 1 extra vector why? Because this all 0 and all 1 patterns both are having even number of ones.

So, the output is 0 in the both the cases. So, how do I detect output stuck at 0 fault? For that I need 1 extra test vector with odd number of ones, the last vector detects output stuck at 0 fault, but the other 2 just as usual all input stuck at 1, stuck at 0 faults, and for both these cases output is 0 that is why I detect stuck at 1 faults. So, in general for m input xor gate, m can be 100 1000 whatever. So, if you able to build that gate, number of

test vectors required will be only either 2 or 3; 2 if m is odd and 3 if m is even this is a very good feature of xor gates.

(Refer Slide Time: 31:31)



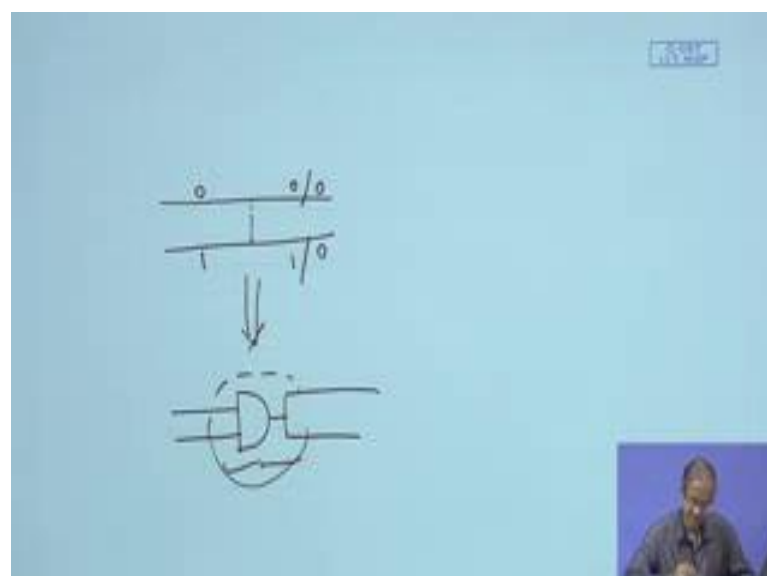
(b) Bridging Fault Model

- Two or more distinct lines are shorted together.
- Quite possible due to imperfections during fabrication.
- Effect of bridging:
 - Depends on technology, or whether positive or negative logic system is used.
 - AND-bridging, where an implied AND operation happens.
 - OR-bridging, where an implied OR operation happens.
- To detect AND bridging, one of the lines involved in bridging is applied 0, while the others are applied 1.
 - Reverse for OR bridging.

And lastly let us look at the bridging fault model, here we say that 2 or more lines are getting shorted together, this is quite possible because lines are running so closely together in the layout, and during fabrication there can be some imperfections.

Because of this bridging something is happening we called it as a AND-bridging or a OR-bridging this and bridging says.

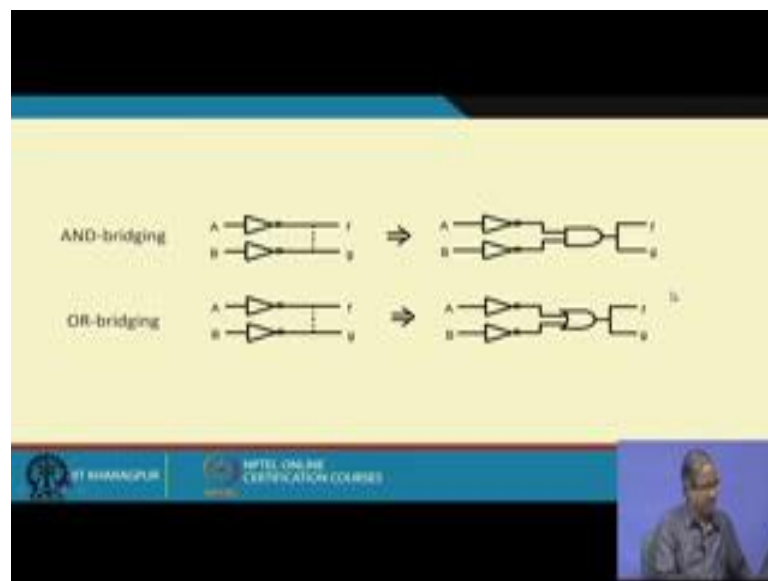
(Refer Slide Time: 32:01)



So, it happens like this, there are 2 lines and there is a short circuit here. Because of AND-bridging what we say is that we assume that as if there is an implied and gate, which has been inserted here, because of this connection and this output of this and gate is going to both places; which means if one of these inputs is 1, other input is 0; because of this and bridging there will be an and of 0 and 1, both of these lines will become 0 and 0 right.

Similarly, OR, so whether it is and or or it depends on which technology you are using for CMOS it is I am bridging typically, and whether we are using positive or negative logic. Positive logic means high voltage is 1 low voltage is 0; negative logic means high voltage is 0 low voltage is 1. Now as the example I have shown to detect and bridging I have to put one of the lines involved in the bridging as 0, while the other lines as one because why? Because if there is no bridging here, the output will be coming at as 0 or 1, it was 0 or 1, but because of this fault, the output will be coming 0 and 0 there will be a change. So, I can detect it right.

(Refer Slide Time: 33:31)



So, this diagram shows that exactly what I told. In the first case as you can see these lines F and G the outputs of 2 gates are running parallel, they are getting shorted bridge. So, affectively there is an and insert and operation inserted the output of which is going to F and G, and if it is or bridging there will a or operation right.

So, with this we come to the end of this lecture, in the next lecture we shall be looking at some more fault models, and some other means operations that we typically carry out to reduce the number of faults.

Thank you.