Advanced VLSI Design

Prof. Virendra K. Singh
Department of Electrical Engineering
Indian Institute of Technology – Bombay

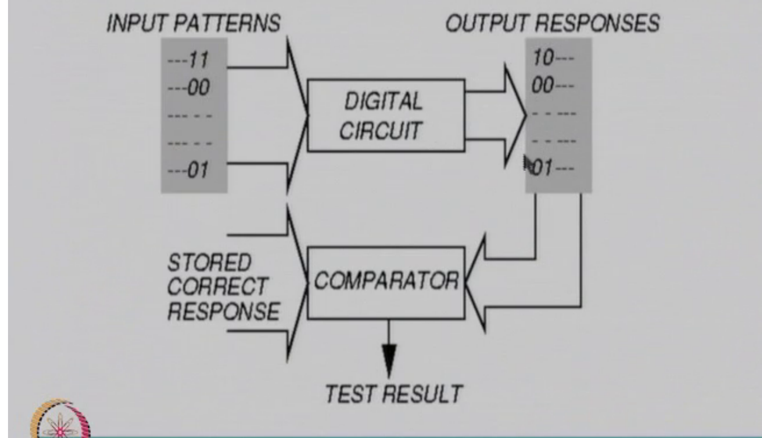
Lecture - 34
VLSI Test Basics – I

Welcome to the lecture series on Advanced VLSI course, in my previous lecture I discussed about the various issues in VLSI test, why VLSI test is important? What are the cost implications? And we have analyzed that what is the typical cost it incurs in terms of money when you test a chip on an expensive tester and we figured out that it is typically somewhere from 2 to 5 cents per second.

And if so that means here if you test your device for about a minute then it may come to 5 come to say 3 dollars that is roughly 150 rupees, so that means here the reasonable time we can take used to test a device could be somewhere from few seconds to minutes and we have also seen that in order to apply all possible test vectors it may take billions of centuries which is impractical.

So our task is to bring down this time from billions of centuries to few seconds to few minutes to make it cost effective and practical okay, so as we discussed in last lecture that the typical test process starts in the following manner.

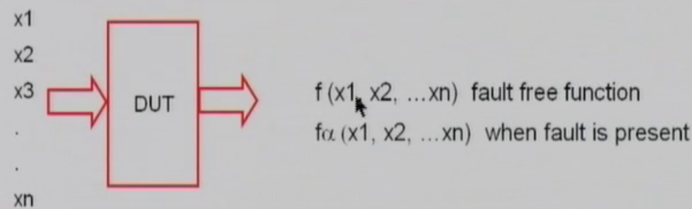
(Refer Slide Time: 02:06)



You have a device or circuit under test you have automatic test equipment you apply test stimuli from automatic test equipment to the circuit under test, collect the response from circuit under test and then compare with other golden response that you have stored on your tester and now here if this matches with the collected response then device is good otherwise device is bad and hence we have to result, so only we have to ship out good device.

In last lecture we also discussed that what are the various requirements what kind of fault coverage we are looking at, so and we discussed that it is governed by the defective parts per million parts we are looking at, and so therefore it where is from application to application like in Automotive application this defective parts per million that we commonly called as PPM is somewhere or it should be very close to 0, though here we cannot assume 0 so it should be close enough to 0.

(Refer Slide Time: 03:31)



Now so what we demand from the test that we have a device under test we apply some stimuli, we collect the response so now here response of a good circuit would given by this function when you apply input at x1, x2 and xn, so this is your fault free function and that you designed, in case there is a fault here function evaluates to f alpha or in case of defect it function evaluates to f alpha.

And now here what we want that here we want to find out a stimuli that can give me two different kind of behaviour of f and f alpha, so that these are distinguishable and we can say that my device is good or bad, so that means here the task is to find out a stimuli that can distinguish faulty and fault free behaviour and that stimuli is called as test for a fault or defect, so but here.

You keep in mind always that whenever we refer a test until unless it is said explicitly it is the detection so that means here we are interested in detection whether chip is fault free or chip is faulty not in diagnosing or locating the fault that means here what kind of fault it is and where exactly it is in the chip, so diagnostic test will talk in separate lecture later.

(Refer Slide Time: 05:25)

Test Basics

For an n input circuit, there are 2^n input combinations.

Ideally we must test for all possible faulty functions.

This will require an exhaustive test with 2^n inputs

x1	x2	x3	f
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Since we cannot apply the exhaustive test set our best bet is to target likely faults!

So now here if you come to look at a say 3 input gate NAND gate, now here this is the truth table of 3 input NAND gate and so now here if you apply say all 3 0's output should be 1 and then here this it sorry, this is the 3 I mean say a function truth table of 3 input function, so now here what we want that here in order to test it we have to apply all possible 8 combinations, but as we have seen in the last lecture that we cannot apply all exhaustive test set.

And hence then we have to find out or we have to select a few test vectors that can target the kind of a faults or defects we are looking at, so when we cannot apply the exhaustive test set then our best bet is to target likely faults. So now here we define couple of definitions faults and errors.

(Refer Slide Time: 06:45)

Test Basics

Defects Faults and Errors

A **Defect** is a physical flaw in the device, i.e. a shorted transistor or an open interconnect

A **Fault** is the logic level manifestation of the Defect, i.e. a line permanently stuck at a low logic level

An **Error** occurs when a fault causes an incorrect logic value at a functional output

So actually we are looking at the defects and defect is defined as physical flaw in the device in other word like a shorted transistor or an open interconnect, this defect has logical level manifestation that means here and that is defined as fault that means here line maybe like permanently stuck or connected to logic level 0 or it can connect to logic level 1, now the so when it manifests as in logic level.

So that means here it causes an incorrect logical output value of the function so that means here there is a defect, defect manifest itself as a fault and then fault manifest itself as a incorrect output or incorrect evaluation of the logical function, so now here we so now the question is as we know that there are numerous defects those are possible and we cannot target all the defects.

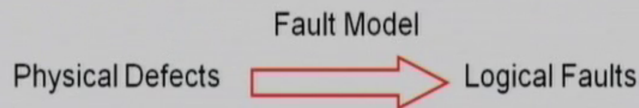
(Refer Slide Time: 07:53)

Test Basics

Likely defects

- Depend on the circuit, layout, process control
- Difficult to obtain

Simplify the problem by targeting only Logical Faults



So now they and these defects they depend on the circuit, layout and process control and difficult to obtain all possible defects some of the defects may not be known from the newer technology, so now we have to simplify the problem by targeting only the logical faults and logical faults are the manifestation of these defects, so you have a physical defect and then you that manifest itself as a fault you model those fault and that appears as a logical faults.

(Refer Slide Time: 08:39)

The Stuck-at Fault Model

Assumes defects cause a signal line to be permanently stuck high or stuck low

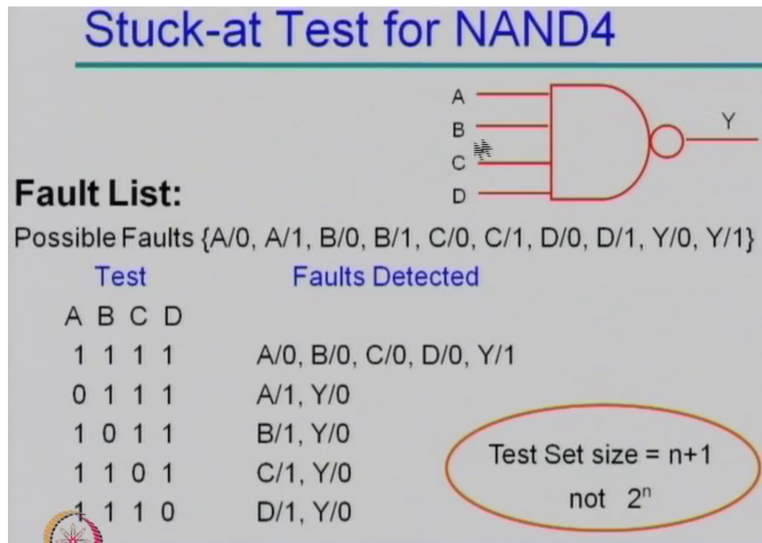
- s-a-0 Stuck-at 0
- s-a-1 Stuck-at 1

- How good is this model?
- What does it buy us?

So now here we model these logical faults and hence we work around those, so one of the simplest model could be we can assume that a defect and cause a signal line either it connects to permanently to logic high or permanently to logic low, so this is the implication of the defect and hence we can model that s stuck-at 0 fault or stuck-at 1 fault that we express s-a-0, s-a-1.

So now here we there are two questions. First thing is how good this model itself is so that means how close it is to model the real defects and then here what does it buy us, we have to evaluate based on that.

(Refer Slide Time: 09:34)



So now here if you look at the say 4 input NAND gate, this 4 input NAND gate can have 4 inputs so 4 input lines and 1 output line, so now if you say that defect may manifest in such a way that any line can either permanently connect to logic 0 or permanently connect to logic 1, so hence there are five lines they can either connect to logic 0 or can connected to logic 1 hence we may have 10 different faults.

Those we represent as A0 that means here line A is stuck to logic 0, A1 that means here line A is stuck to logic 1, B0 means line B stuck to logic 0, B 1 means line B stuck to logic 1, now here if you look at if I apply say all 1's as input, so here if I apply all 1's as input in that case here what we expect from this gate if there is no fault in that case here output should evaluate to 0 right, so now if A stuck to logic 0 then what will happen the output Y will be 1.

Hence I can produce the distinguishable fault free and faulty behaviour hence this vector all 1's can detect this particular fault A0, look at B if B is stuck to logic 0 in that case again here the output it will produce output as 1 when I apply 1 1 1 1 as input and now here again we are

getting distinguishable faulty and fault free behaviour hence this fault is also detected by this one, in the same way C is stuck to 0, D is stuck to 0 can also be detected.

Look at the output if output is if I apply 1 1 1 1 here then my fault free output should be 0, now if the output Y is defective it is stuck to logic 1 then once I apply here 1 1 1 1 output would be 1, hence I will have distinguishable fault free and faulty behaviour, so this particular vector can detect fault A is stuck to 0, B is stuck to 0, C is stuck to 0, D is stuck to 0 and Y is stuck to 1, in the same way if I apply a vector 0 1 1 1 the output of this gate would be 1.

If this A is stuck to logic 1 in that case here before this line is permanently stuck to logic 1 so that means it does not have any effect of what I apply at primary input A and B, C, D are already 1, 1 so now here output would be 0, so I have distinguishable faulty and fault free behaviour hence the A is stuck-at 1 is detective, now look here at the output if I apply this 0 1 1 1 output is 1 if Y is stuck to logic 0 always it will produce output 0 right.

So that means here this gives me the faulty distinguishable faulty and fault free behaviour hence this also detect Y stuck-at 0 fault at line Y, so now here if I look at this so in the same way a vector 1 0 1 1 can detect stuck-at 1 to at B and stuck-at 0 at Y, so now here if I look at these four vectors sorry these 5 vectors they can cover all these 8, 10 faults right, so now here if I look at this number ideally if you remember if I want to test it exhaustively I may need 16 inputs to test this gate.

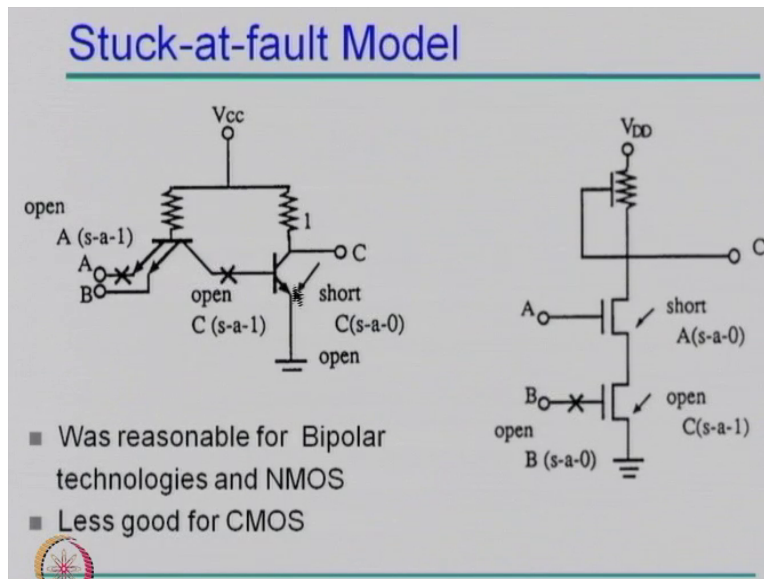
Now if you look at this number 5, 5 is the total number of lines you have so that means here the total number of inputs + 1, so now here what it gives you what it buys out to you is that here now you convert your exponential complexity into a linear complexity, so that means here you need test vector which are linearly proportional to the number of inputs you have, so this is for 1 gate.

If you look at a complete circuit things are pretty much like this not exactly n + 1 this may this would be n + 1 for single output circuit single output fan-out free circuit if it has fan-out in that case number will be slightly more that means here you have to add the number of fan-out points,

so that means here if you can model your defect using stuck-at fault you can convert the test complexity from exponential to the linear and that gives as big relief.

So that means even if you have millions of gates you may have millions of faults that means you may need millions of vectors to test it and millions of vectors may be applied in reasonable time, so and that is why we look at we model the defects as a faults which are the implication of these defects.

(Refer Slide Time: 16:10)



So now what the means in this fault model is what this can model if you look at this circuit which is the TTL logic and the Stuck-at fault model was given way back in early means late 50s or in 60s and at that time technology was Bipolar technology and this worked very well for Bipolar transistors as well as NMOS this is still it is relevant for CMOS technology except few additional faults that may occur in CMOS.

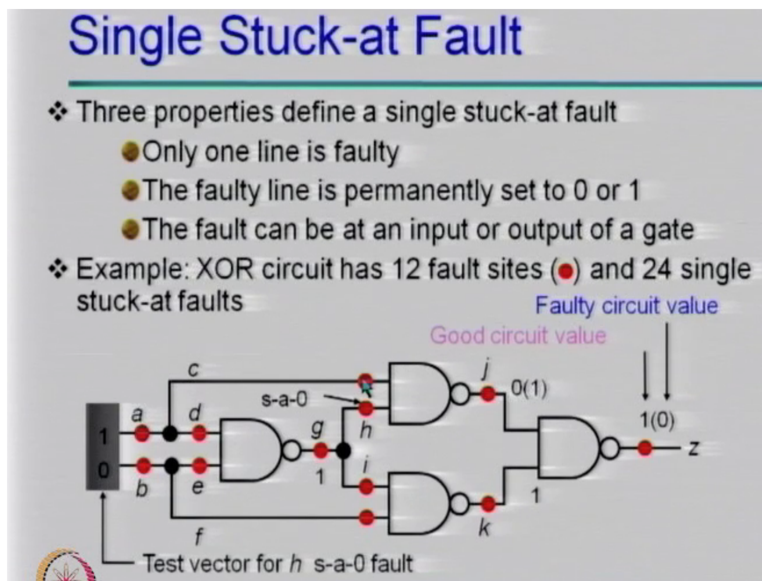
So now here how I justify that say there may be a defects so that means here that this transistor shorts if this transistor shorts that is the defect but the implication of that defect would be like here this output C would be always give you the logic 0 hence we can say that the implication of the defect that this transistor is short is stuck-at 0 fault at logic at output C.

In the same way if this point is open in that case here that the implication of that would be so this transistor will not conduct and then here you will get high potential and high potential means here the C is stuck-at 0, so now here defect is here this is open but here output is stuck to 1, in the same way if you look at the MOS logic if this transistor shorts in that case implication of that is like A is stuck to logic 0.

If this transistor is open in that case here due to this open always this is in the pull up mode and then C acts as always is stuck to logic 1, so now here the defect is this transistor is open but here this implicate in terms of in the form of stuck-at 1 fault at output C, so this tells you that a defect multiple defect can have the same implication so that means here multiple defects can be modelled as a single fault.

So that means here there is no uniqueness or relationship between the defect and fault and this model worked very well for Bipolar and for NMOS but here and it works reasonably well for CMOS except a few more fault, that we have to target separately.

(Refer Slide Time: 19:17)

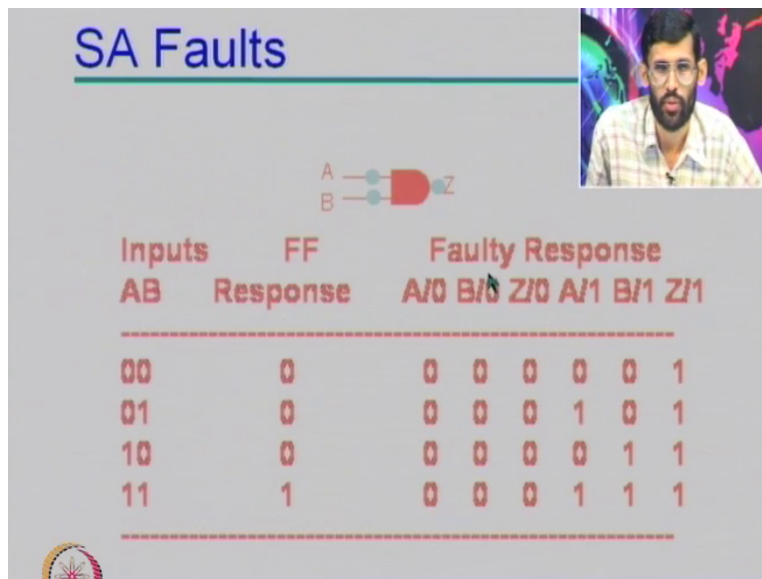


So now here if I can model fault like this then I do not need to look at what kind of defects can occur in the during the manufacturing I need to test only for the modelled fault and modelled faults are proportional to the number of lines we have or number of signal lines we have like for example in this circuit we have total 12 lines so like a, b, d, e, g, h, i, j, k and z, so these are 12

different lines and any line can either is stuck to logic 0 or can stuck to logic 1 hence there can be two faults.

So now here at 12 different fault sites there can be 24 total faults total single stuck-at faults, single stuck-at faults means here we assume that at a time there can be only one fault there cannot be multiple faults that is our assumption, so by from this we infer that here now the number of faults are proportional to the number of signal lines we have keep in mind that fault at the stem of fan-out branch and at different fan-out branches are different they have different implications, so you have to treat them different faults sites.

(Refer Slide Time: 20:48)



So if you look at this like here for example in this AND gate there are two inputs and one output this is the truth table, and now if you look at the truth table you have both input 0 0 output is 0 then here if you have both input as 1 1 output would be 1, now look at how it evaluates when A is stuck to logic 0, B is stuck to logic 0, Z is stuck to logic 0, A is stuck to logic 1, B is stuck to logic 1 and Z is stuck to logic 1, these are the responses under these conditions.

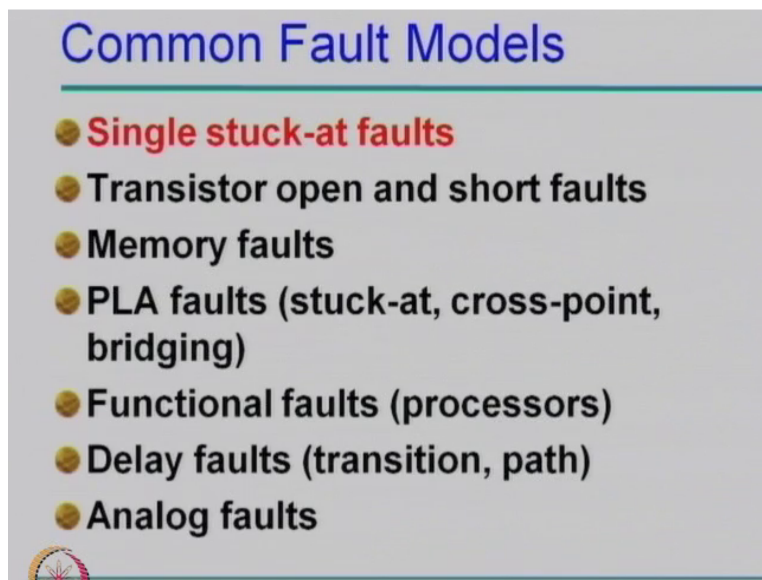
Now look at all faulty and fault free behaviour so fault free behaviour here is 0 and for all these fault free I mean faulty behaviour is also 0 that means this is not distinguishable so that means this input cannot distinguished, cannot detect these faults only this can detect this fault where we

have the different fault free and faulty behaviour, this vector can detect these two faults whereas this vector can detect this two faults, this vector can deduct all these three faults.

So that means here one vector can detect multiple faults like here this vector can detect two faults, this vector can detect three faults or one fault can be detected by multiple vectors, so now here the question is that how many test vectors are need to detect all the faults so that means we have to choose a smallest set of test vectors that can detect all modelled fault keep in mind modelled fault.

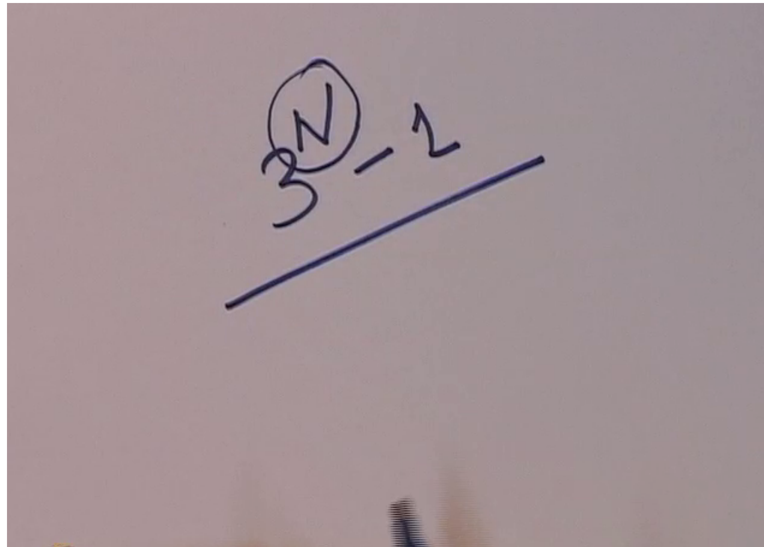
So from this one we have to choose a smaller set that can detect whole possible faults like we so in earlier example that 4 input NAND gate we need 5 test vectors, so now you have modelled fault and you have a relationship between the fault and test vectors and we the objective is to find out smallest set of vectors that can detect all possible modelled fault here in this case all 6 faults okay, so this is one of the fault model which is called as single stuck-at fault.

(Refer Slide Time: 23:28)



Now there are other fault models in this single stuck-at fault we are assuming that there can be one fault at a time in the circuit that may not be very reasonable assumption but people figured out by experiments that if you detect all single stuck-at fault large number of multiple stuck-at faults are can also be detected and because of the nature of multiple stuck-at fault we cannot target at all the possible multiple stuck-at fault.

(Refer Slide Time: 24:17)



And number of possible stuck multiple stuck-at faults are to 3 raise to the power n - 1 that is this 3 raise to the power n - 1 and now here n is the number of fault sites single stuck-at fault sites you have or number of signal lines you have hence now this is exponential again here we are going from linear complexity problem to exponential this is not possible.

So now the experimental studies say that if you detect almost all single stuck-at fault you are likely to detect 80 85 % multiple stuck-at fault that may be good enough, there are other faults like here a transistor can be open or that can be short and these are detected by some different methodologies that may we may discuss sometimes later. Then other kind of circuit we have is memory.

Memory is very special type of circuit and it has a special functionality or a special function to perform that is it stores a value that is logic 0 or 1 and it retain that until it changed. So that means here you have a complex or dense array of memory cells they are storing some information and they retain that until you write it back.

So now because of the limited functionality in place of modelling every fault at every signal line is stuck-at 0 or is stuck-at 1 we go for functional fault model and that functional fault model needs to check whether all the cells are able to detect, able to store value 0, able to store value 1, able to make transition from 0 to 1 or 1 to 0 and the memory address decoder works fine that

means here address decoder should address to single cell and proper cell which you want to address.

These are the various different fault models for PLA kind of Programmable Logic Array kind of circuits here we may have some more faults like stuck-at fault is already there, there may be cross-point so or there can be a bridging, the other kind of circuits that we have in practice are like microprocessor, microcontroller or that kind of device which again here is supposed to perform some kind of operation which is very limited in nature like a microprocessor executes couple of instructions in some specific way.

So it is not very random and hence here we can make use of functional fault model, then as we discussed earlier there are some faults which may or may not be there during the manufacturing time they may implicate while it is operating or your system may have may not have any kind of logical fault but it may not meet your timing requirement and those are performance related faults and those are modelled as delay faults.

Then the analog circuits or weak signal circuits we have and for weak signal or analog circuit we do not have a specific fault model, generally we do a specification based testing so they have different kind of fault again here the modelling of faults in an analogue circuit is open challenge, okay,

So now here we discussed how we can model a fault and how close that is your defect implication and what it buys us, what it buys us is it can convert the exponential problem into a linear problem, so that means here now I do not need to apply to the raise to the power n pattern I may need to apply only n + 1 pattern which is linear and that is a great relief, so now the question how I generate those test patterns that is very important problem.

As we discussed earlier that now here says this pattern 1 1 1 this can detect stuck-at 0 here, stuck-at 0 at B, stuck-at 0 at C, stuck-at 0 at D and stuck-at 1 at Y right, now from this I apply this one and I see what is my faulty and fault free behaviour if I do that simulation so that means

here one true value simulation and then here simulation for each and every fault and see where we have distinguishable faulty and fault free behaviour.

And then based on that we say that this vector can detect these faults right, now if I ask question other way round how if I know that there may be A stuck-at 0 fault here what is the test vector which can detect this stuck-at 0 fault so that means we have to device a mechanism to find out these test vectors which target one particular fault. There are various ways and one of the way is Algebraic method.

(Refer Slide Time: 30:46)

Algebraic: Boolean Difference

- Shannon's Expansion Theorem:
  $F(X_1, X_2, \ldots, X_n) = X_1 \cdot F(1, X_1, \ldots, X_n) + \overline{X_1} \cdot F(0, X_1, \ldots, X_n)$
- Boolean Difference (partial derivative):
  $\frac{\partial F_j}{\partial x_1} = F_j(1, X_2, X_3, \ldots, X_n) \oplus F_j(0, X_2, \ldots, X_n)$
- Fault Detection Requirements:
  $X_1 = 1$ (SA0) or $X_1 = 0$ (SA1)
  $\frac{\partial F_j}{\partial g} = F_j(1, X_1, X_2, \ldots, X_n) \oplus F_j(0, X_1, \ldots, X_n) = 1$

Algebraic method is based on Boolean Algebra, so if you have function F that has n input X1 to Xn then now here the function evaluates to F of X1 to Xn and it evaluates based on the value of X1 to Xn, I can decompose this function into two functions here using the Shannon is expansion theorem say with respect to each and every variable, so with respect to X1 I can decompose as X1 dot F of if I replace X1 by 1 then X2 sorry this is X2 , X2 to Xn and + X1 bar F of 0 X2 to Xn sorry this is X2.

So this gives you the two different factors these are defined as cofactors so that means this function evaluates to this function if X1 is 1 this function evaluates to this function if X1 is 0 right, so that means here if I want to detect a fault at X1 what I want that the behaviour that I can

obtained by having two values at X1, 0 and 1 should be distinguishable otherwise I cannot detect that fault right.

So that means here this function and this function should be orthogonal to each other, so that means here this the XOR operation of this function and XOR operation of this function must be different, so that means XOR operation must be must evaluate to 1 and that is defined as Boolean difference so that means Boolean difference with respect to a given variable where your faults site lies should evaluate to 1.

So if you do not have fault it evaluates to some values if your fault in that case here it should evaluate to different value, now then the what should be the test vector?

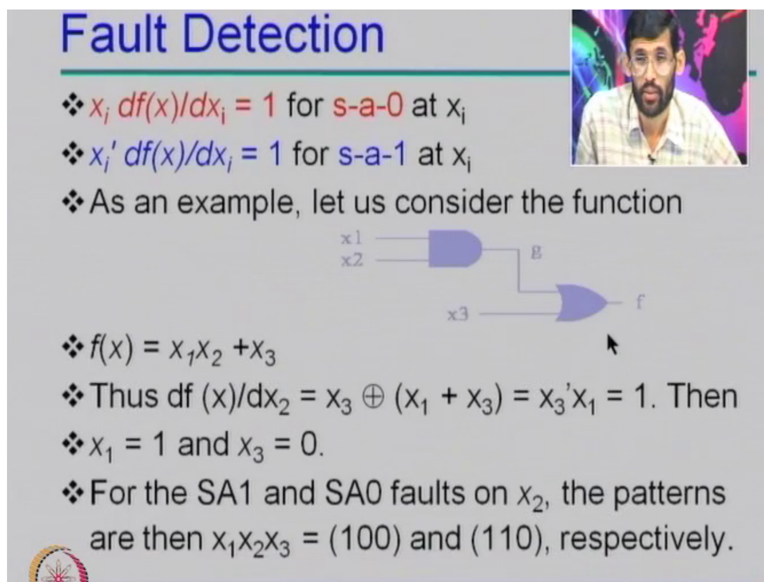
(Refer Slide Time: 33:13)

So we know that this Xi if fault is at Xi then your Boolean difference with respect to Xi is del f upon del xi, so assume that fault is stuck-at 0, if it is stuck-at 0 in that case here that fault should be excited, excited means here that you have to have distinguishable the different value should be applied at that fault site. So that means this Xi must be = 1 and this Boolean difference must also evaluate to 1 this says Xi dot del f upon del Xi this should evaluate to 1 this gives you the test vector for stuck-at 0 fault at xi, at xi.

(Refer Slide Time: 34:14)

If fault is stuck-at 1 at Xi then because here if you apply 1 in that case here you are not going to change anything so you have to have opposite value so that means here Xi must be 0 and that fault effect should be propagated to the output so that means here del f upon del Xi this should evaluate to 1 that means here this is Xi bar dot del f upon del Xi this should evaluate to 1, this is the, these gives you the test vector for stuck-at 1 fault at Xi.

(Refer Slide Time: 34:59)



Now let us take a small example so these are the two conditions take an example here this function I can write as X1 X2 + X3 assume that there may be stuck-at 0 and stuck-at 1 fault at X2 right, that means I have to find out whether the Boolean difference of this is 1 or not if

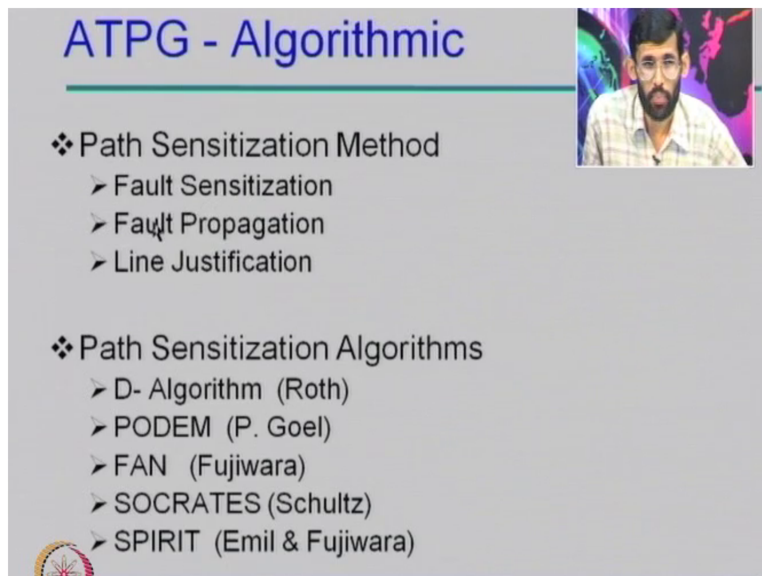
Boolean difference is not 1 that means X2 has no role to play in making a decision or in evaluation of this function right, that means X2 is a redundant input here.

So now look at and how I can obtain this Boolean difference if I put X2 = 1 then it will evaluate to X1 + X3, if I put X2 = 0 in that case here it will evaluate to X3, so now here Boolean difference would be X3 XOR with X1 + X3 that is X3 bar X1 that should be 1, so what it results into if always this Boolean difference must be 1 that means here X1 should be 1 and X3 should be 0 if you want to detect this fault.

Now here if you want to detect we stuck-at 0 fault that means X2 must be 1 so you have to have just opposite value and if you want to detect X2 stuck-at 1 then here X2 must be 0 right, so now the test vector for stuck-at 1 would be 1 0 0 and test vector for stuck-at 0 would B 1 1 0 this way I can mathematically or using Boolean Algebra can generate test vector for a given fault, so now here I know that this circuit has 1, 2, 3, 4 and 5 fault sites there can be obtain different faults

And I can detect test for individual fault using this Boolean Algebra but manipulation of Boolean expressions for a fairly big circuit or support is not so easy, so then we need to find out a way a method which is scalable for large enough circuit.

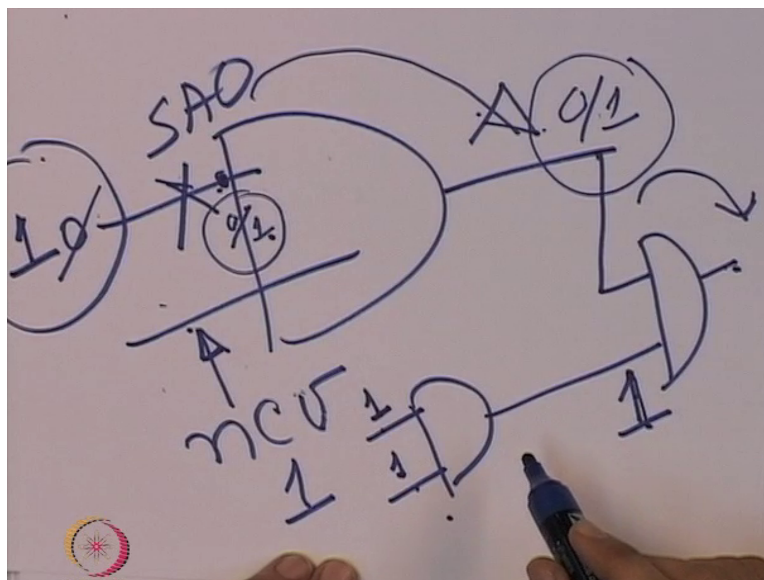
(Refer Slide Time: 37:31)



And then the way is algorithm 1 way rather than algebraic way, so and there are couple of algorithmic methods presented in the literature like first came in 1968 that is known as D-Algorithm which is was given by Roth from IBM, then 1981 other algorithm came that is known as PODEM which was again given by Goel from IBM, then in 1984 the another better algorithm came which was given by Fujiwara.

And then there are couple of other algorithms one of the popular algorithm was SOCRATES which uses learning process, then SPIRIT again this was from Fujiwara and his group, so all these algorithmic methods are based on three principles or three steps, one is the Fault Sensitization, Fault Propagation and Line Justification, fault sensitization mean here you have to sensitize the fault.

(Refer Slide Time: 38:52)



That means if you are looking at say this is your AND gate and your fault is here so say this is stuck-at 0 fault, how I can sensitize this if I apply here 0 can I sensitize this fault I cannot because this circuit can never produce different value under the faulty and fault free conditions so hence this is this cannot be, so now here sensitization condition is that you have to have just opposite value to the fault so now here you have to have 1 so this is the fault sensitization.

Now can I observe the fault effect here I cannot observe fault effect here so now you have to propagate this fault effect to one of the observable point and observable point is the primary

output, so you have to transfer the fault effect to the observable point, so here I have distinguishable fault effect if it is faulty then here I have 0, if it is fault free in that case I will have 1, the same thing I want to propagate to the output.

And in order to propagate that to the output here the another input to this gate must be at the non-controlling value, and non-controlling value for the AND gate is 1 so if I put it at one in that case here output would be determined by an another input right, so now here if they if I can have distinguishable faulty and fault free behaviour at this input that would propagate to the output and I can look at the output this is known as fault propagation.

And now here this is the case when this is the primary output say this is not primary output there is one more gate after this now again I have to propagate this to the primary output if I propagate this to the primary output here again I have to put it at the non-controlling value and non-controlling value is 1, assume this is also coming from another gate say and now here what I want so I cannot justify value 1 here.

I can justify value at the primary input and these two are primary input so I can justify this value 1 by justification at the primary input and now here I justified all the values at the primary input there are 4 primary inputs and I justified all the those, and those are in 1 line hence here this is known as line justification so the third process is line justification, so these are the three important factors in this.

So now here an algorithmic method to generate a test is based on path sensitization and path sensitization follow these 3 steps fault sensitization, fault propagation and line justification and as I said that here that these are the couple of algorithms there are many more.

(Refer Slide Time: 41:58)

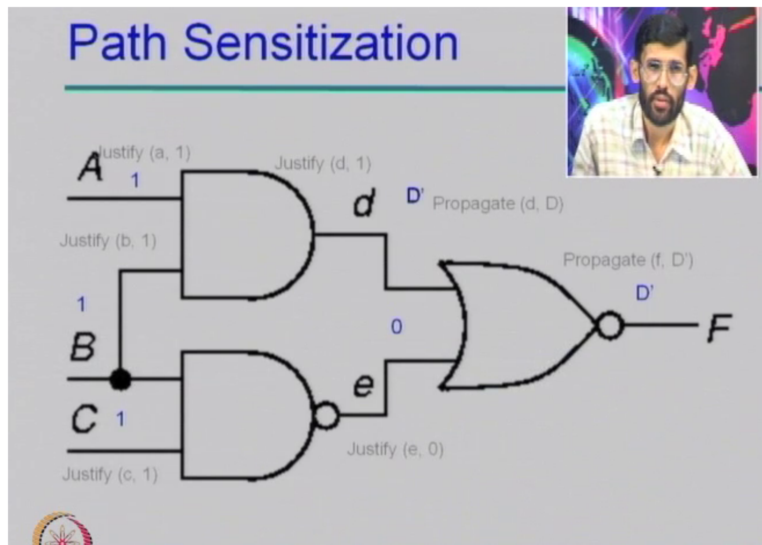
Path Sensitization

General Structure of TG Algorithm

```
begin
    set all values to x
    Justify (l, v)
    if (v = 0) then Propagate (l, D)
    else Propagate (l, D')
end
```

So the general structure is you have to begin from a fault and justify that value at that location, and then you have to propagate that faulty value or distinguishable faulty and fault free value to the one of the primary outputs and then you have to justify that.

(Refer Slide Time: 42:23)



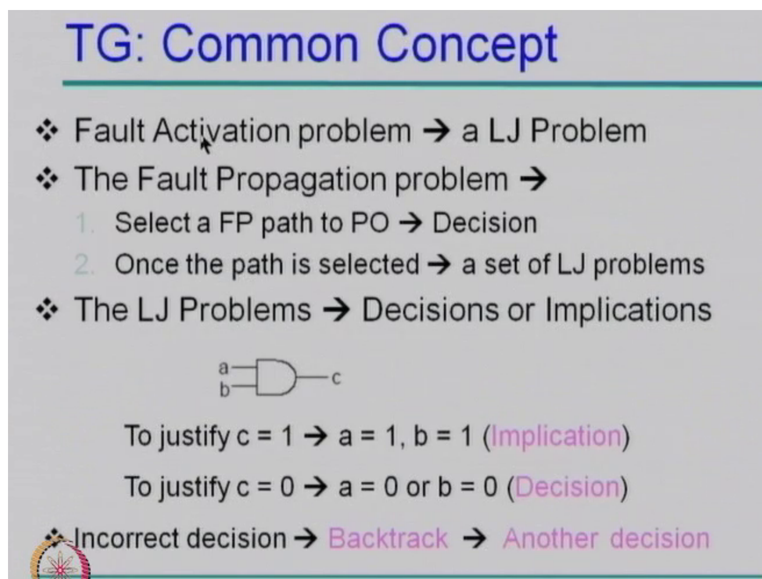
So if like here for example in this circuit say there is a fault art line d and that fault is assumed that it is stuck-at 1, so now here if it is sorry stuck-at 0 so the fault at d is stuck-at is 0 then here what I want in order to sensitize this fault I need to have value 1 so I have to justify value 1 here how I can justify that value 1 in order to justify I have to go back to the primary input, so what gives me 1 here I can have obtained value 1 here only by assigning A = 1 and B = 1.

So I assign the A = 1 and I assign justify value B = 1 and because these two are the primary input it is do evolve, now this site my fault now I have to propagate this fault effect to one of the primary output and it has only one primary output so I have to propagate here in order to propagate here so I have to propagate it here and now because this is inverting gate so now here they see I use a simple D this symbol D is just notional thing say this D has distinguishable faulty and fault free behaviour.

So now here so I can propagate it to the output now in order to means get it propagated here we know that here I need non-controlling value here and non-controlling value of NOR gate is 0 hence I have to justify value 0 here I cannot justify value 0 here or say put it other way, I do not have any control here right so I have to go back to the primary input and see how I can justify that.

So now in order to justify 0 what I want both input must be 1, one input is already 1 if I assign another input as 1 in that case I can justify value 0 here and hence I am done my fault effect will propagated to the output and I can get a test vector that is 1 1 1, so this is the basic of all these algorithmic methods to generate a test.

(Refer Slide Time: 44:59)



So if I look at how the what are the difficulties with this here there are as we know there are three steps fault activation or sensitization, fault propagation and line justification or so now here what

are the difficulties with fault activation I have look at the previous approach here if previous figure here if want to have 1 here I have to go back to all the way to the primary input and assign some value here right, so this problem converts into line justification problem.

So fault sensitization problem or fault activation problem is a line justification problem what are the difficulties of line justification problem we will discuss will be later, other step is you have to propagate fault effect to one of the primary output how you can propagate so if it is fan-out free circuit in that case it is easy because here you can always target to one output like you have to follow a path.

But in a circuit where in you have several fan-outs you can propagate fault effect to any of the output through any of these fan out points, so that means you have to make a decision that how with through which part you have to propagate the fault effect to the one of the primary output it is like you from say department to hostel you want to find out a way and that way is the easiest one so you hit to a crossing and then you have to see which way I should go should I take left or should I take right.

And now here if you are not familiar with that location sometimes means your decision may be good and you reach to your hostel, if you are not familiar and your decision may be wrong if it is wrong then here and you may end up in a forest, so you have to what you do you have to come back to the same point right, so now here and then again explore the other path so now this process can be a decision making process or so now there are two things.

One is the propagation is a decision making process, second thing is once you made a decision you have to have assign the non-controlling value to the another primary input of that great right, so and that you mean at intermediate point you cannot justify so you have to go back and justify at primary input, so now that problem again here converts into a line justification problem now look at the line justification problem.

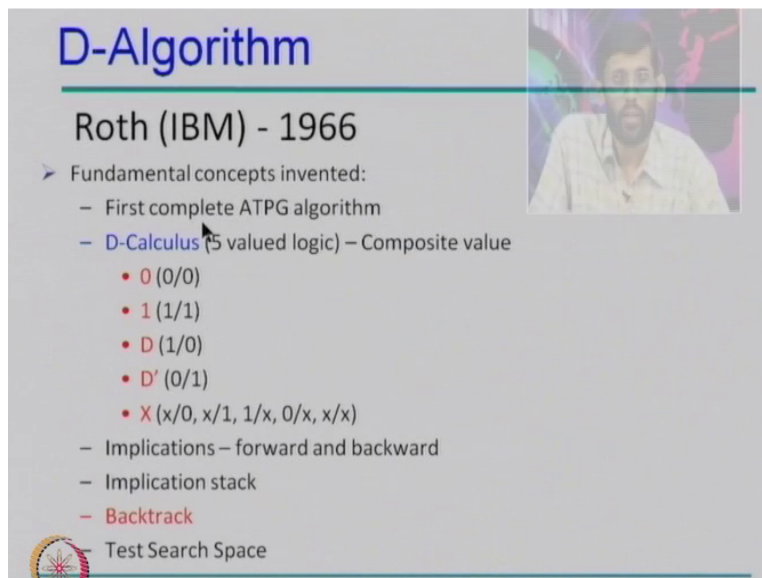
Line justification problem can be either a decision making or that can be a implication say look at this get output is C and now you may need to assign say 1 here if you need to assign 1 here in

that case you have only one choice A = 1 and B = 1 that is just by implication that if you want C = 1 you have to have A = 1 and B = 1 that is simply implication and whereas if you want to assign 0 here there are two ways one is if you, rather I can say three ways.

That A can be 0 then it gives you C = 0 or B can be 0 then you will also get C = 0 right, if both are 0 in that case also C = 0 so now here and this is the decision making process because you how to assign either A = 0 or B = 0 and then this may so then this is the decision and decision may be right or this may be wrong if it is wrong in that case you have to come back to the previous decision making point and explorer the an other one okay.

So this gives you the in nutshell the difficulty of test generation process you are making several decisions and these decisions are may or may not good and if they are not good in that case you have to backtrack to the previous decision making point and explore an other opportunities.

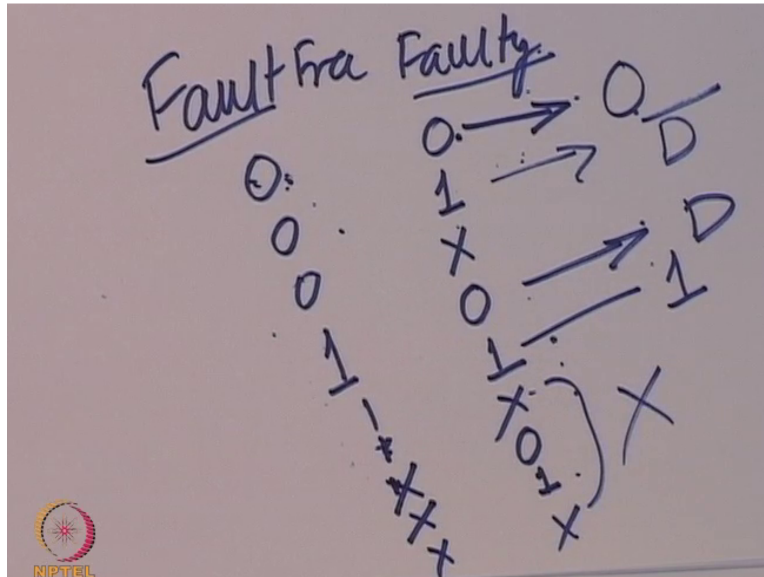
(Refer Slide Time: 49:51)



So now here I briefly will discuss D-Algorithm that was given by Roth in 1968 1966 this was the first complete ATPG algorithm so what does this mean that if test exist for any given fault it gives you the test vector otherwise it will say that there is no test for that particular vector and those faults are redundant faults those are coming from redundancy in the circuit this is based on D-Calculus that uses the five valued algebra.

So and because here we want to propagate a composite value that contains faulty and fault free output, so that is described by 5 symbols.

(Refer Slide Time: 50:50)



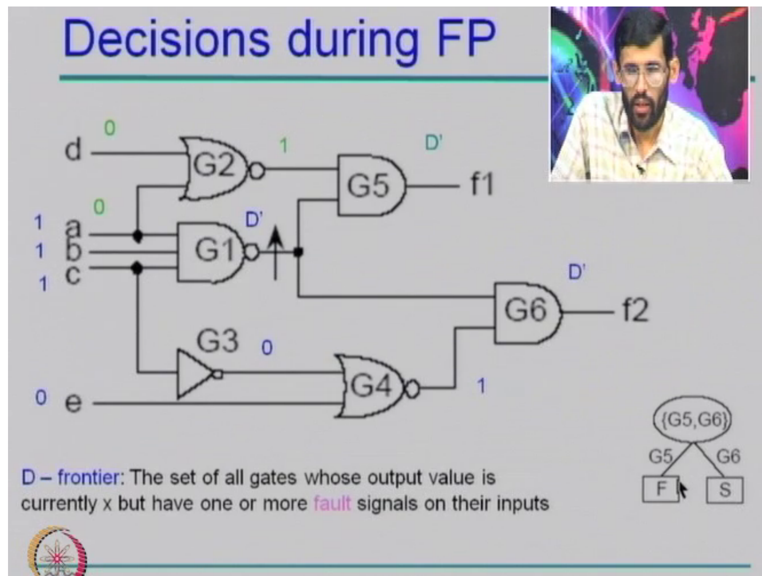
Assume so now you want to list the output has a composite value that is fault free and faulty right, so now here fault free can be 0 then faulty can be 0, faulty can be 1 and faulty can be X if you are using 3 value 0, 1 and X and now here fault free maybe 1,:fault free maybe 0, 1 and X, so 1 1 1 and now here your fault free maybe X and faulty maybe 0, fault free maybe X and this may be 1 and this is X and this is X.

So now here you have both 0 0 that is assigned with a symbol say 0 because this gives you the indistinguishable faulty and fault free behaviour, here also you are getting both 1 1 in that case here assign a symbol 1, here this gives you fault free value as 0 sorry 1 and faulty value as 0 this is assigned with a symbol D this is just opposite to this now here this is assigned with a symbol D bar and then the rest of the values wherein you have at least 1 X are combined in one and make X.

So these are the composite values it uses now here it heavily dependent on implications you tries to use implications as much as possible and implications as I said that here like if you want to have if you have two input of an AND gate as 1 1 then output is always 1 right and if one input is 1 and an other.

So that is the implication and for that implication it maintains a implication is stack and then because at various points you are making a decision and if your decision is wrong so you have to backtrack and for that backtrack here you have to again maintain a stack that can bring you back to the same previous decision point and so now here it tries to search the solution space.

(Refer Slide Time: 53:14)



Now here like here how the decision making process works like for example in this circuit here if I say there is a stuck-at 0 stuck-at 1 fault present here that means I can represent that by a composite value and that composite value is your D bar wherein the fault free value is 1 and faulty behaviour is fault free is 0 and faulty value is 1, so now D bar then first you have to excite it, excitation means here you have to have value 0 here in order to have 0 you have all three values is 1 so that is line justification problem.

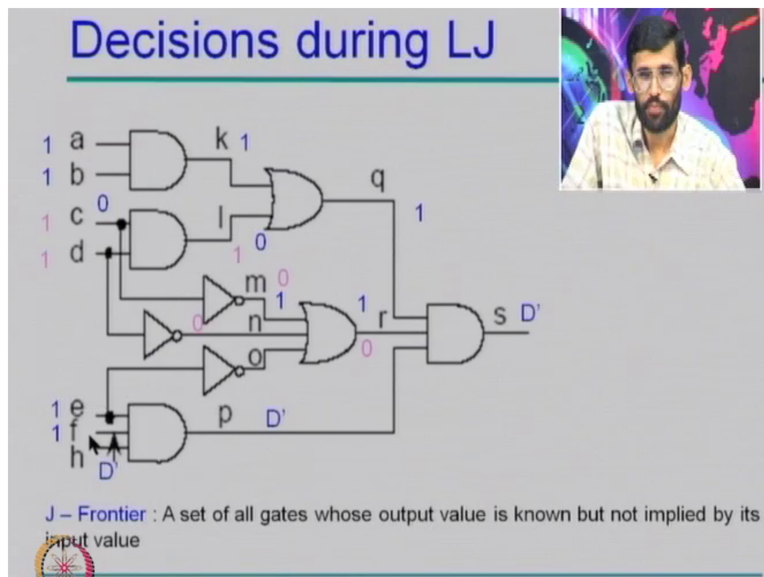
Now at this point there are two fan-outs you can propagate through either through this one or through this output, so but at the same time you have to imply so now here this 1 can imply as a 0 here so again here let us say now you would like to propagate this through gate G5 now if you want to propagate through G5 in that case you need to have value 1 here in order to have 1 what do you want you want both of the values are 0 but there is a conflict here.

Because it has already assigned value 1 right so that means you end up with the conflict and then come back to the previous decision point and your previous decision point was to transfer the fault effect through this path, now you have to explore the alternate path and alternate path is through gate G4 if you look at the exploring through G4 in that case you want 1 value here and now here for in order to have 1 here both of these input must be 0, one input is already 0 and now here this is justified.

Hence you will get input as here d can be anything whether 0 or 1 then a should be 1, b should be 1, c should be 1 and e should be 0 this is the test vector for this particular fault and now if you look at the decision making process there was one decision point wherein you could have transfer this fault effect through either G5 or G6 you decided go through G5 you failed then you return back to the same decision making point.

And now you explorer through G6 and you succeed and that gives you test vector in the same way so now here in order to look at whether my fault effect is present in the circuit or not we maintain a stack that is a list that called as the D - frontier and this D - frontier will have both of the this gate G5 and G6 we explorer one so we take out one and then explore the another one.

(Refer Slide Time: 56:20)



Now here this is an another example of line justification say you have stuck-at 1 fault here at line h and now here in order to excite this I need to have 0 value here and in order to propagate this

you need to have e and f has 1 1 this will propagate here and in order to propagate means through this gate you need to have 1 here and 1 here, let us say you want to justify this 1 first by using the line justification in order to have 1 here you may have either 1 here or 1 here.

So let us say you assign 1 here so now in order to have 1 here you will need to have both 1 here right, now that has implication so m would be 0 and then this n would be 0 and if m and n are 0 and this o is already 0 by this implication, so that means here this r would be 0 and hence your fault effect will be lost, so now so that means your earlier decision was not good so that means here like assigning 1 here was not good so now come back to the previous point.

And now in place of this you assign 1 here if you assign 1 here that you can easily justify by assigning 1 and 1 here and now you are done, if you so now in order to assign then now you are done here, but you have to still justify this 1 here in order to have 1 here say you want to have 1 here in order to have 1 here you may want to 0 here right, if it is 0 then again you are done.

And so now here you have to maintain a stack a list that is called as J – frontier that which are the gates who has assigned output but unassigned input okay, so this way you can generate test using D-Algorithm and so now you need to go through these three steps fault sensitization, fault propagation and line justification.

These algorithm gives you a test vector if it accessed always hence it is a complete algorithm, though in worst possible case it may explore the entire solution space hence the this is and be complete problem, but in the observation is that most of the time within the few backtrack you can generate the test hence here this problem is do evolve. Now we stop here we will carry forward in the next lecture, good day.