

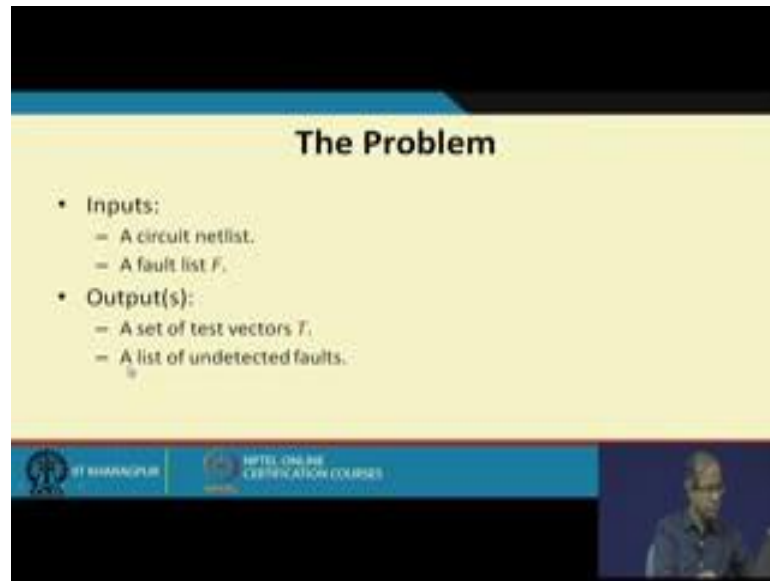
VLSI Physical Design
Prof. Indranil Sengupta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 53
Test Pattern Generation

So, here we continue with our discussion on VLSI testing. So, we recall during the last few lectures we actually talked about the basic motivation behind testing why do we need it. Then we talked about the very important steps of fault modeling, then fault simulation and the different methods which exist there in. So, today we briefly look at the test generation problem; so the topic of this lecture is test pattern generation. Now let me tell you that the problem of test pattern generation is difficult; even for combinational circuits it can take quite significant amount of time to carry out test generation for certain faults.

There are some faults which are said to be difficult to detect; so the test generation software need to spend a lot of time and effort in trying to find out a test for that particular fault. So, I mentioned earlier that in comparison fault simulation is much faster therefore, what people do usually, they generate a test for a particular fault then immediately they carry out fault simulation, to find out what other faults are also getting detected. So, they will simply remove those other faults from their target for list, and try to generate fault the tests for the remaining faults.

(Refer Slide Time: 01:54)



The slide is titled "The Problem" and is presented on a yellow background. It lists the following:

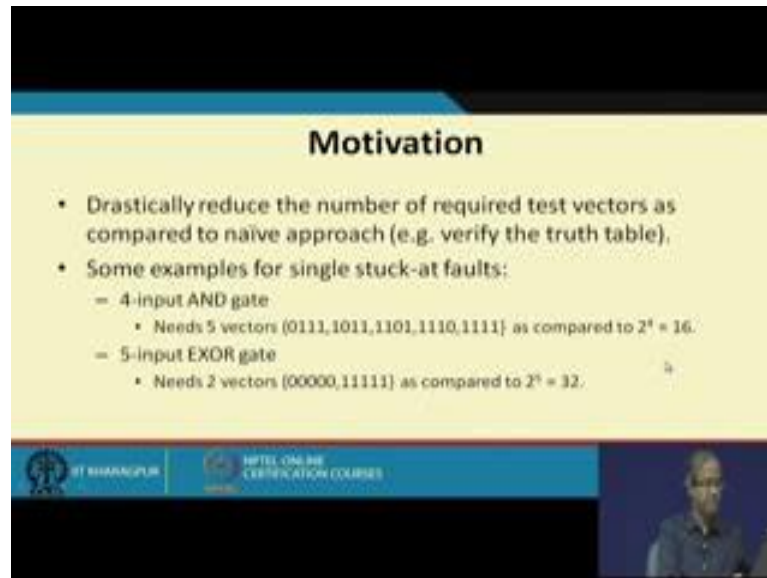
- Inputs:
 - A circuit netlist.
 - A fault list F .
- Output(s):
 - A set of test vectors T .
 - A list of undetected faults.

At the bottom of the slide, there are logos for "ST BHANAGUR" and "INTEL ONLINE CERTIFICATION COURSES". A small video inset in the bottom right corner shows a man speaking.

So, let us look into the basic problem of test pattern generation. So, in this problem the input is circuit net list of course, and fault list.

So, here we are concentrating ourselves to fault list that consists of single stuck at faults. What this tool will give you as output? It will give you a set of test vectors T , that tries to detect faults from this list F and also it will give you list of the faults for which it was not able to find test. Now usually the way this process goes on there is nothing like I mean parallel processing here as what was being done during fault simulation, the detective and concur fault simulation. Here the process is sequential, you take one fault from the fault list try to generate a test, then take the seven fault from the fault list generate a test and so on and in between you can carry out fault simulation to speed up the process by removing some of the faults, which are also getting detected right.

(Refer Slide Time: 03:07)



The slide is titled "Motivation" and contains the following text:

- Drastically reduce the number of required test vectors as compared to naive approach (e.g. verify the truth table),
- Some examples for single stuck-at faults:
 - 4-input AND gate
 - Needs 5 vectors {0111,1011,1101,1110,1111} as compared to $2^4 = 16$.
 - 5-input EXOR gate
 - Needs 2 vectors {00000,11111} as compared to $2^5 = 32$.

At the bottom of the slide, there are logos for "IT MANAGER" and "INTEL ONLINE CERTIFICATION COURSES". A small video inset in the bottom right corner shows a person speaking.

So, I give some examples earlier also so let me repeat here. The basic motivation behind doing test pattern generation is to drastically reduce the number of test factors that are required. Because our target is to test certain set of faults; now possible the best way to do that is to compare the truth table of a given function, but as you can understand this size of the truth table can grow very fast so it is not a very feasible approach. So, assuming that I have target for model for example, the single stuck at fault, I want to find out what is the good I means I will not so optimum because finding the optimum is again very computationally difficult, find a good set of test vectors that will try to find out or detect all the faults from the given set. So, some examples are shown here, I had mentioned this earlier also.

Let us say a 4 input and gate. So, 4 input and gate will require 5 test vectors and not 2 to the power 4 for a normal truth table comparison. Normally you would require 16 vectors, but if you target this stuck at faults only single stuck at faults you need only 5 test vectors. The first 4 test vectors will be detecting these stuck at 1 faults on the inputs, the first one stuck at 1, second one stuck at 1 and so on; and also it will detect stuck at 1 fault on the output. While the last does vector the all 1 pattern this will detect this stuck at 0 faults on the input, as well as this stuck at 0 faults on the output. But for an exclusive or gate with odd number of inputs let us say 5 we need only 2 test vectors; 0 0 0 0 0 and 1 1 1 1 1.

(Refer Slide Time: 05:35)

The slide displays a logic circuit with four inputs (A, B, C, D), two intermediate outputs (E, F), and one final output (G). The circuit consists of three gates: an OR gate (A, B → E), an AND gate (C, D → F), and a final AND gate (E, F → G). A table lists 14 faults and their corresponding test vectors:

Fault	Test Vector
A/0, C/0, D/0, F/0, G/0	1011
A/1, B/1, E/1, G/1	0011
B/0, C/0, D/0, E/0, F/0, G/0	0111
C/1, F/1, G/1	0101
D/1, F/1, G/1	0110

Below the table, the test vectors are listed as $T = (0011, 0101, 0110, 0111, 1011)$.

So, this also very discussed earlier as compared to 32 test vectors if you have done naïve truth table comparison. So, this is the motivation behind test pattern generation, that if we have a target set of faults we can reduce the number of test factors required drastically it was small number. So, let us take some examples. So, here we have a 2 level and our circuit, there are 3 gates as you can see, you can see how many lines are there 1 2 3 4 5 6 7. So, there are total of 14 faults. Now this table shows you the result of a test pattern generation program. So, if you run test pattern generation program telling the tool that I want to detect all single stuck at faults here, then the tool will give you this 5 test vectors; this 5 is not unique, some of the test vectors can be different also, but you can easily check this 1 0 1 1 pattern 1 0 1 1.

So, if you apply 1 0 this E will be 1, if you apply C D as 1 1 F will also be 1. So, 1 1 means G will also be 1. So, you can easily check that this pattern can detect all these faults because. If A is stuck at 0 then both inputs will be 0 0, E will be 0 and hence G will be 0; so if G will change. So, whenever the final output changes we can say that we have detected default. Similarly C is stuck at 0, C is stuck at 0 will make F 0, these stuck at 0 same F stuck at 0, if this is 0 also G will become 0 and of course, G stuck at 0. There is another fault I missed that will also be their G stuck at 0, G stuck at 0 should also be here. Similarly here 0 0 1 1 and this is another pattern this can detect some of this stuck at 1 faults.

Stuck at A stuck at 1, B stuck at 1 or E stuck at and of course, the final output stuck at 1 because normally here the output will be 0. Similarly I show the other 3 test patterns and what are the faults that a getting detected. So, as I can say that my test set that is being generated consist of this 5 test patterns.

(Refer Slide Time: 07:42)

Generating Test from Truth Table

- How to generate test for the stuck-at-0 fault α ?

α : stuck-at-0

Condition: $f @ f_\alpha = 1$

Inputs (a b c)	f	f_α
000	0	0
001	0	0
010	0	0
011	0	0
100	0	0
101	1	1
110	1	0
111	1	1

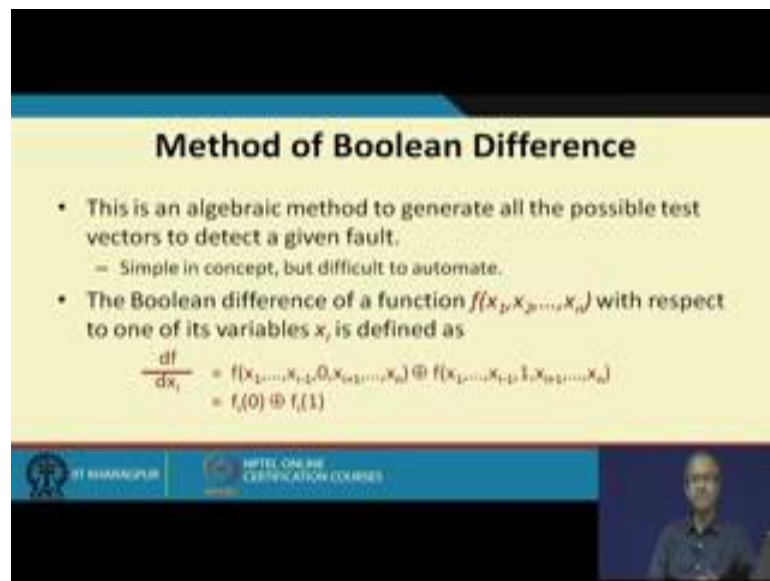
Let us start with the very naive approach; let us try to find out test from the truth table. We take as a problem a circuit like this, so I mean this is again 2 level hand over circuit. So, here we are targeting the detection of stuck at 0 fault on this line, the output of the first and gate. So, we call it alpha. Alpha is the fault which is a stuck at 0 fault on this line. So, what you do we just write down the truth table the inputs are a b c. So, the original function in absence of the fault this realizes a b or a c. So, this is the corresponding truth table.

This is the output column for the fault free function, it is a b or a c; you see when a b is 1 or a c is 1 then the output is 1. Now if there is a stuck at 0 fault here, so this first input of this or gate will always be 0, so now the faulty function becomes only a c. So, I have also shown the faulty function f alpha where you see only a c if it is true, then only the output is 1. So, the so called truth table based approach says that if I have the truth table with me, and if I have side by side the fault free function and the faulty function outputs, then I try to find out row where the outputs are different; like here the output is different in this row, so the fault free function is 1 and the faulty function is 0. So, as I said earlier

also the condition for detection is that for a test vector, the exclusive or of f and f_{α} should be 1, XOR means either 0 1 or 1 0 then the condition says then only you can detect the fault.

So, here it is 1 zero. So, the XOR is 1, so this satisfies. So, from here you can say that for this particular fault there is a single test vector 1 1 0.

(Refer Slide Time: 10:18)



Method of Boolean Difference

- This is an algebraic method to generate all the possible test vectors to detect a given fault.
 - Simple in concept, but difficult to automate.
- The Boolean difference of a function $f(x_1, x_2, \dots, x_n)$ with respect to one of its variables x_i is defined as

$$\frac{df}{dx_i} = f(x_1, \dots, x_{i-1}, 0, x_{i+1}, \dots, x_n) \oplus f(x_1, \dots, x_{i-1}, 1, x_{i+1}, \dots, x_n)$$
$$= f_i(0) \oplus f_i(1)$$

The slide also features logos for IIT SHANGHAI and NPTEL ONLINE CERTIFICATION COURSES, and a small video inset of a speaker in the bottom right corner.

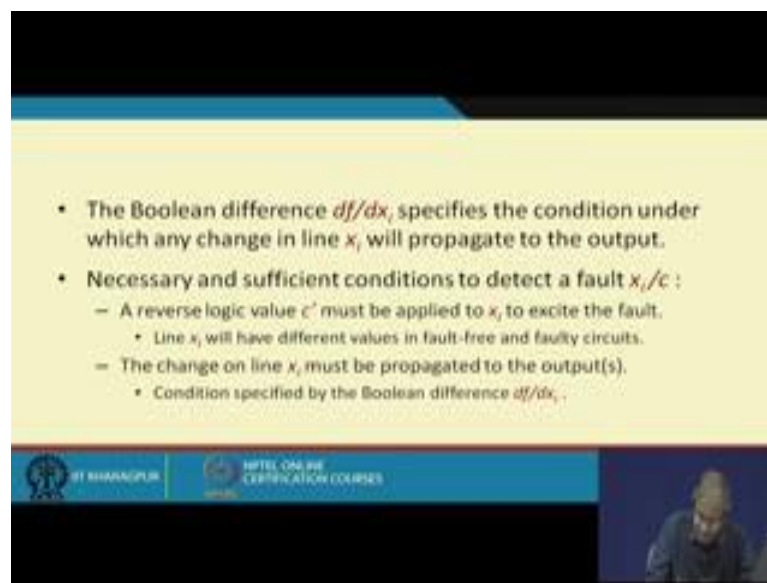
But if you had seen that there are more than one rows where this condition holds then several test patterns are possible for detection right? But in this case this is unique 1 1 0 fine. Now let us look at the method of Boolean difference, we have seen the method of Boolean difference earlier in a different context where you are looking at the faults path the timing analysis there, but here you will see how the same concept of Boolean difference of a function can be used for generating all possible test vectors for a given function, without constructing the truth table explicitly. Let us see the basic concept. So, the first point to notice that this is an algebraic method, this algebraic method means we depend on some algebraic manipulation.

So, because it is algebraic manipulation it is little difficult to implement, this method is not very convenient to implement as a computer program. So, this says in concept to the simple, but with respect to automation or writing a program for it, it is a little difficult. So, you recalled the definition of a Boolean difference which we discussed earlier also, the Boolean difference of an in variable function let say f , where the input variables are

$X_1 X_2$ to X_n . The Boolean difference with respect to one of the variables let say X_i is defined as follows; we denoted as the differential notation df/dx_i . Now you think of the derivative the differential notation, when you studied derivatives in calculus what does dy/dx denote? dy/dx denote that if we change X by an unit amount, how much is the difference in y ? That is the basic idea behind the (Refer Time: 12:09) right.

So, here also it is something similar; it says if I change X_i what is the change in f ? So, the way this is measured is well I said X_i to 0, this is a sub function where the other variables remain same only X_i is 0. I said X_i to 1 that means, I am changing the value of X_i and I am seeing whether the value of f is changing or not; that means, I am doing an exclusive or. If I see this part is 0, this part is 1 or 1 and 0 then I will say that in f also there is a difference. So, in short we denote it as $f_{i=0} \text{ XOR } f_{i=1}$; $f_{i=0}$ means the i th variable is said to 0, the i th variable is said to 1; this is the definition of Boolean difference; now let say how this can be used.

(Refer Slide Time: 13:09)

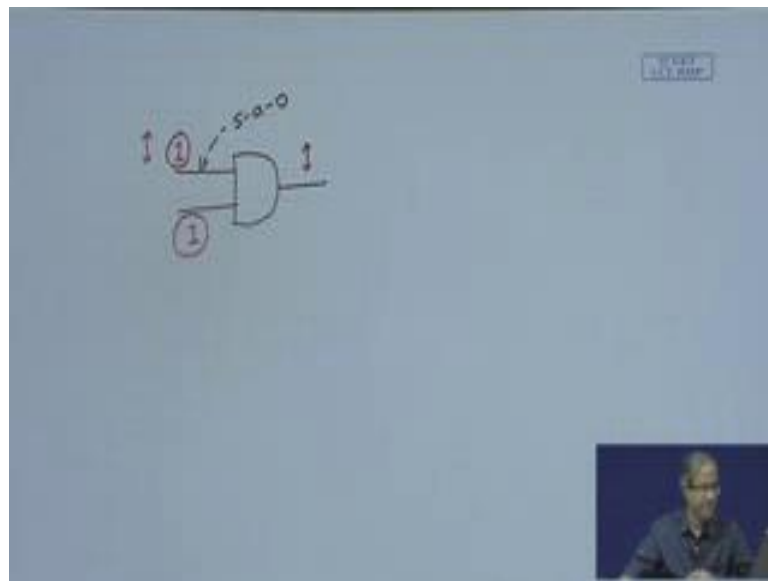


So, Boolean difference basically as I said it specifies the condition under which if you change X_i the value of f will also change; that means, the change will propagate to the output.

This is the concept of Boolean difference; let me again go back it says if I change the value of X_i then this condition specifies that f is also changing if df/dx_i is equal to one let say, which means this is 0 1 or 1 0 which means if I change X_i , f is also changing; or

which in other word says that any change in line X_i is propagating to the output in terms of a change; that means, if X_i changes f also changes. So, based on this concept we can determine the necessary and sufficient conditions to detect a particular fault let us say X_i stuck at C where C can be either 0 or 1. So, there are 2 conditions for detection of a fault so this is mentioned here. The first condition says here we need to apply reverse logic value I denoted as C' . So, if it is a stuck at 0 fault, then I have to apply a 1 to this line to excite the fault.

(Refer Slide Time: 14:49)



Like I am giving an example here, let us take a simplest example of a single gate. Suppose we are trying to find out stuck at 0 on this line. So, our first requirement will be to apply a reverse logic value on this line which is 1, the compliment of 0. And the second requirement will be; to apply suitable logic value here such that this change that is happening on this line, fault free was 0 fault is 1, this change should be propagating to the output. For the AND gate I have to apply 1 which is a non controlling value to the other input, so that this change propagates, these are the 2 conditions. So, this is mentioned here; firstly, a reverse logic value must be applied to X_i , which means the particular line X_i will have different logic values in the fault free and faulty circuit configurations and secondly, the change on line X_i must be propagated to at least 1 of the outputs these are the 2 conditions.

Now, the second condition can be specified by Boolean difference you see, what does $\frac{df}{dx_i}$ denote? It denotes that change in X_i is causing a change in f which is exactly the fault propagation condition.

(Refer Slide Time: 16:21)

- The set of test vectors that can detect the fault $x_i/0$ is given by $x_i \frac{df(X)}{dx_i} = 1$
- The set of test vectors that can detect the fault $x_i/1$ is given by $x_i' \frac{df(X)}{dx_i} = 1$

Excite the fault, and propagate the fault

x_i or x_i' $\frac{df(X)}{dx_i}$

So, we can combine these 2 observations and you can determine the condition for the generation of the test vectors, it is summarized here. The set of test vectors that can detect the fault X_i is stuck at 0 will be given by the product of 2 things X_i , which means what? I mean I have trying to detect X_i is stuck at 0 means I have to apply a reverse logic value; I am denoting it by X_i' . Because if this function has to be 1 both X_i and this has to be 1. So, X_i equal to 1 means I am applying a reverse logic value and $\frac{df}{dX_i}$ equal to 1 means the fault is getting propagated, the changes getting propagated. So, X_i and $\frac{df}{dX_i}$ equal to 1 if I solve this equation, this will give me all the possible input combinations for which this condition is satisfied and they will constitute this set up test vectors for this fault.

Similarly, for the fault X_i stuck at 1, the only changes I use X_i' . Because now X_i has to set will reverse value which is 0, so the product of this 2 will be 1. So, X_i' is equal to 1 means $X_i = 0$ and of course, the change will be propagating to the output right. So, there are 2 conditions excite the fault and propagate the fault for exciting the fault I am using either X_i or X_i' depending on the polarity of the fault, and for

propagating the fault I am using the second term the Boolean difference the product of these 2 has to be 1.

(Refer Slide Time: 18:06)

So, let us work out for a simple example; consider gate level net list like this comparison of 5 gates, suppose we want to generate tests for fault C stuck at 0 and C stuck at 1. So, if you just make a Boolean manipulation this is the function A or B, C prime and C prime and the other site C D or C D, this is the function.

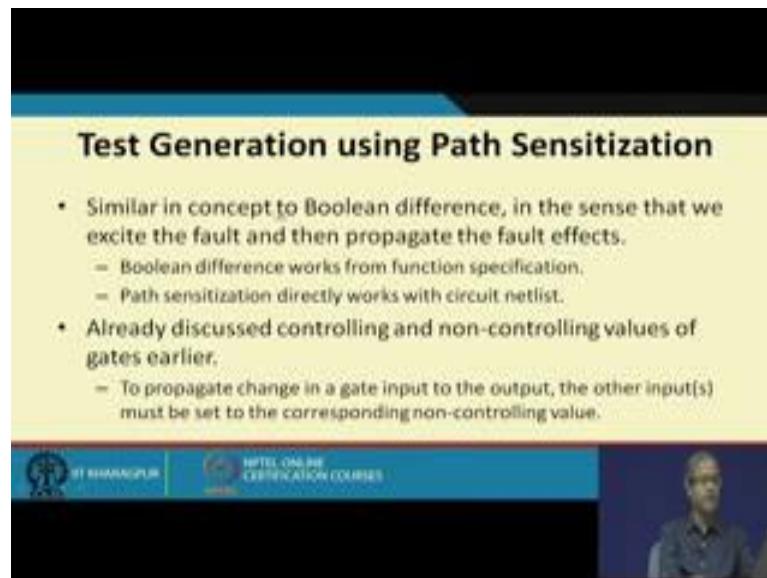
So, since we are targeting to detect faults on line C, we compute the Boolean difference dF/dC . So, what we do? We apply C equal to 0 first, if I apply c equal to 0 the second term vanishes and the C bar becomes 1. So, we are left with A plus B, A plus B and we said C XOR, we said C equal to 1. If I said C equal to 1 the first term will vanish because C prime will be 0 and this term will be only D. So, if you just simplify this you get this expression. Now for C stuck at 0 the condition for detection will be C and dF/dC will be equal to 1. So, you multiply this dF/dC by C, so the condition is ACD' prime, BCD' prime or A' prime B' prime $C'D$ equal to 1. So, from this you can say that you have generated 3 it is vectors, the first vector says that A has to be 1, C has to be 1, D has to be 0; and B is do not guess. So, actually this is 2 test vectors, B can be either 0 or 1 I show it as do not care. Second one says A is do not care $B' C' D$.

But the third 1 says 0 0 1 1. So, we have generated all possible tests that can detect C stuck at 0. Similarly for C stuck at 1 we multiplied dF/dC by C prime, we get an

expression like this, and for this to be 1 these three product terms will have to be 1, this or this or this any 1 at least; so the conditions are $A \bar{C} \bar{D}$ $\bar{A} C \bar{D}$ $\bar{A} \bar{C} D$ $A C D$ is do not care and similarly these. So, you see this is very simple method which allows us to determine all possible tests that can detect a particular fault, but one thing here I have just given an example to illustrate the method, and this example whichever I have shown I have assumed that the fault is appearing in the input, but in general the fault can also appear in a line which is somewhere in between.

So, for those there are number of Boolean I mean said Boolean difference rules that you can apply, which requires significant amount of function manipulations, I am not shown this examples here, but just let me tell you that for detecting internal faults in a circuit, the problem is little more difficult, we have to divide the function rules sub functions compute Boolean difference of the sub functions again combine them it is a little involved process, but it can be done. But the trouble is that as I said the entire process consists of manipulation of expressions, which is a little difficult to do in an automated way by using a tool or a program.

(Refer Slide Time: 21:44)



Test Generation using Path Sensitization

- Similar in concept to Boolean difference, in the sense that we excite the fault and then propagate the fault effects.
 - Boolean difference works from function specification.
 - Path sensitization directly works with circuit netlist.
- Already discussed controlling and non-controlling values of gates earlier.
 - To propagate change in a gate input to the output, the other input(s) must be set to the corresponding non-controlling value.

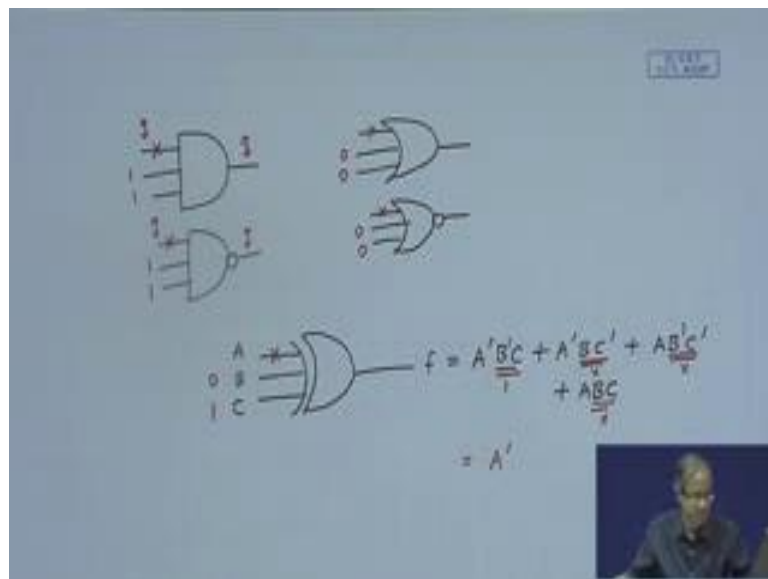
INTEL ONLINE CERTIFICATION COURSES

So, most of the modern tools that are used for test generation, they use some kind of path sanitization approach. But again let me tell you the practical methods which have there, they are for to complex to be discussed in the short period of time here. So, I will just give you the basic concept behind path sensitization, and I will tell you what are the

challenges existing methods and tools they try to address this challenges and use a number of heuristics such that test generation can be faster ok.

So, here the first point is that it is similar in concept to Boolean difference; how? Because here also we are satisfying the 2 conditions for detection, first is exciting the fault second is propagating the effect of the fault. But the main difference lies in this sub bullets. Boolean difference works from function specification like you look at the previous slide, we started with this specification of a function as a Boolean expression from there we did everything right? But in contrast path sensitization does not look at the function at all, it directly looks at the circuit net list gates and there interconnections. So, the method of path sensitization is least bothered about what is the function that is realized by the circuit, but it is more interested in the actual circuit description ok.

(Refer Slide Time: 23:46)



So, earlier you recall we discussed the controlling and non controlling values of the gates. So, to propagate change in a gate input to the output, the other input must be said to the corresponding non controlling values. So, let me just refresh your memories. So, if I have an AND gate, let say it is a 3 input and gate. The same will be true for a nand gate I am showing it side by side. Suppose here I am considering fault one the first line, stuck at 0 stuck at 1. So, if it is stuck at 0 have to apply a 1 if it is stuck at 1 I have a apply a 0, which means I am creating a change on this line. But in order that this change is propagated to the output you recall for an AND gate, the non controlling value as 1,

which means I apply the non controlling values similarly for nand to 1. So, that whatever is the change that will be propagated as signal change in the output.

Well for a nand they will be inversion, but still the change will be there. If this line changes from 0 to 1 this will change from 1 to 0, but for end it will change in the same direction. But similarly for an OR gate, if you have an OR the non controlling values are different or an or same. So, if we have a fault on this line and this line, here the non controlling values will be zeros. Because if you apply 1, that is the controlling value that will force the output to be 1, so this change will not propagate right. So, these are the non controlling values, but you think of an exclusive or gate let us take a 3 input example, 1 X XOR gate let me just. So, f is given by the expression like this, A mean odd number of input should be 1.

So, $A \bar{B} \bar{C}$ or $A \bar{B} C \bar{C}$ or $A B \bar{C}$ or $A B C$. So, here the good thing is that if I want to propagate this fault, I can apply arbitrary values on the other input; because for XOR gate there is nothing like a controlling input every input is controlling is every input is non controlling. Let us say suppose I apply B 0 and C 1, let us see what happens. B 0 C 1 means this will become 1, B 0 this will disappear, B 0 C 1 this will also disappears C prime is there, B 0 C 1 this will also disappear. So, you see your expression is A prime. So, the value of A is getting propagated right. So, for XOR gate for any combination of the inputs the fault will get propagated.

(Refer Slide Time: 26:50)

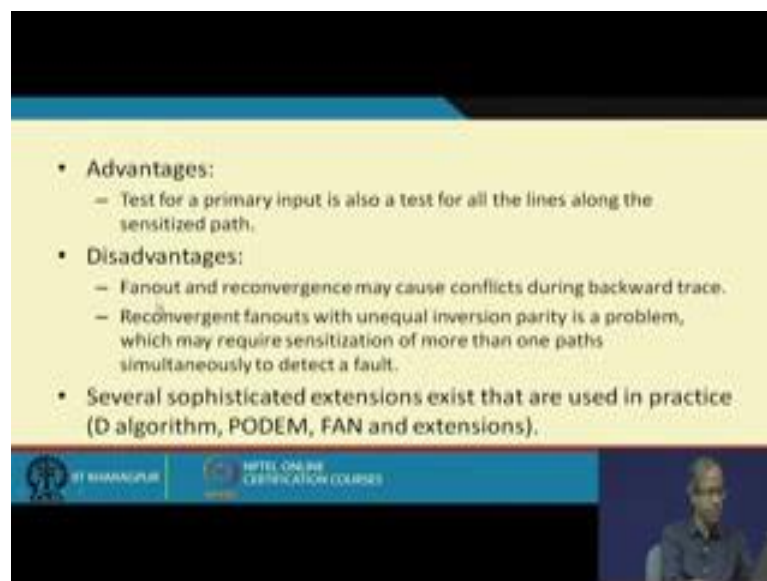
Basic Principle

1. At the site of the fault, assign a logic value that is *complementary* to the polarity of the fault being tested.
2. Forward Drive Phase
 - Select a path from the site of the fault to one of the circuit outputs.
 - Sensitize the path by assigning non-controlling values to the inputs of the gates along the path so as to propagate the effect of the fault.
3. Backward Trace Phase
 - Determine the primary inputs that will produce the necessary signal values as specified in Steps 1 and 2.
 - Requires tracing the signals back towards the primary inputs.

IIT BHARUNPUR NPTEL ONLINE CERTIFICATION COURSES

So, the basic principle of this method of path sensitization is like this. So, at the site of the fault, we assign complimentary logic value just like what you are doing in Boolean difference; then we sequentially carry out 2 phases which we shall illustrate with an example. First is the forward drive phase, we will select a path from the point where the fault is occurring we call it this sight of the fault to one of the circuit outputs. Then we try to sensitize the path by assigning non controlling values just in the way we mentioned, so that the effect of the fault can propagate up to the output. Then we need to do something called a backward tracing, where we have to determine the primary inputs which make all this things possible. Here we have applied some logic values to some of the intermediate lines, but we have to do a backward tracing to determine what primary input values will cause these signal values as we have decided in steps 1 and 2; this requires tracing the signal backwards.

(Refer Slide Time: 28:06)



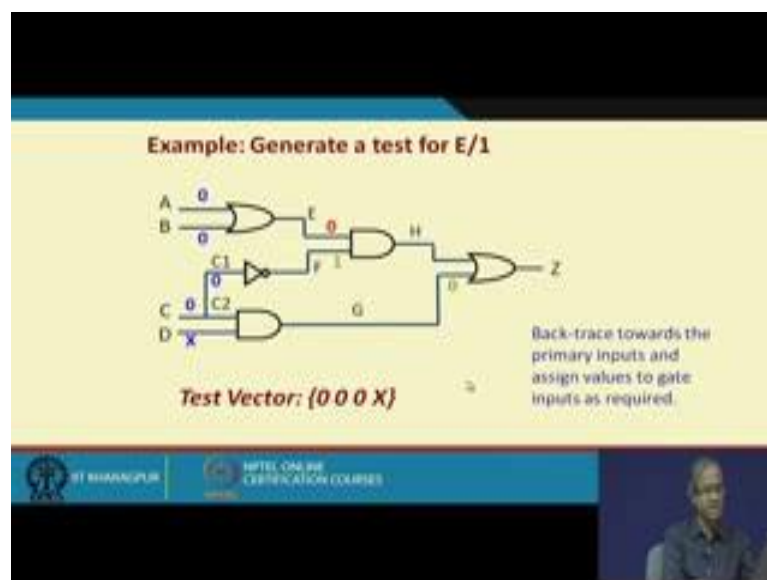
- **Advantages:**
 - Test for a primary input is also a test for all the lines along the sensitized path.
- **Disadvantages:**
 - Fanout and reconvergence may cause conflicts during backward trace.
 - Reconvergent fanouts with unequal inversion parity is a problem, which may require sensitization of more than one paths simultaneously to detect a fault.
- Several sophisticated extensions exist that are used in practice (D algorithm, PODEM, FAN and extensions).

So, we shall be taking an example the advantages are; the test from a for a primary input is also a test for all the lines along the sensitized path, why? You see I have an input there is a long path to the output. So, I apply the inputs to the gates in such a way that any change in the input will be propagating to the output, this same will whole for the faults on the intermediate lines; if this line is selected not only the input any change in this line will also the propagated, any change in this line will also the propagated, so all faults along the path will be detected by the single test vector, that is one advantage. But disadvantage is that while here we are not discussing this details, that if the circuit has

fan outs and after fan out it is again converging later, it may require lot of back tracking because there can be conflicts during backward trace. You are I mean you may fail to find the unique assignment of the input values, so, you may have to go back try an alternate path and repeat the process.

So, this may also require sensitization of more than one path simultaneously; these create problems and complexities as I just said. So, in practice there are many methods which are used, there all variations of the basic algorithms which are D algorithm, PODEM and FAN, but the current versions are improvements of this with some heuristics that help us in running the test generation faster.

(Refer Slide Time: 29:49)



Let us take an example. So, we have a circuit net list here, let us try to generate a test for fault E stuck at 1. So, in our first step what we do? A reverse logic values applied at the site of the fault. So, we apply a 0 here.

Next step we have to propagate the fault effect to the output by applying non controlling values to the other inputs of gate along the path. So, here there is a single output and from the site of the path there is a fault this is a single path. So, what I do? And gate I apply a non controlling value of 1, or gate I apply a non controlling value of 0, so that the fault effect gets propagated. Then this 0 1 1 these are my assignments already met, now I have to back tress. Back tress towards the primary input and assigns values to the gate inputs accordingly; like for example, I have a 0 here. So, what should be the values of A

B which is or? Must also be 0 0 then only I will get 0 here. So, like, if we proceed back
tress 0 0, then I have a 1 here this inverter input should be 0 because this is 0, C will also
be 0.

Now, you see. So, once you made C 0 that is automatically makes D 0. So, D is not
required to be specified. So, D can be do not care. So, you see for this fault we have
obtained a test vector 0 0 0 do not care and so easily without any manipulation anything.
And the point I am saying that the path that is getting sensitized like for example, E stuck
at 1 was my fault, I can also detect see here the H would be 0 1 you can applying. So, H
will be 0, under fault free condition. So, H stuck at 0 will also be there and Z stuck at 0
will also be there. So, all this faults along the path will also be detected by this test
vector.

(Refer Slide Time: 31:58)

Points to Note

- Path sensitization is not as simple as the example shows.
 - There can be conflicts during back-tracing that requires a backtracking algorithm.
 - More than one path may need to be sensitized together.
- Practical test generators are quite complex and sophisticated.
 - Good combinational ATPG tools exist.
 - Sequential ATPG is much more time consuming.
- Test generation is slower than fault simulation.

So, just a few points to note as I said path sensitization is not as simple as this example
shows, there can be lot of conflicts during the back tracing phase we need a so called
back trucking algorithm; or you may need to sensitize multiple paths at the same time.

So, practical test generators are quite complex and sophisticated. So, regarding
combinational ATPG stands for automated test pattern generation; so combinational
ATPG tools are available, they are pretty good. But sequential ATPG tools are in
comparison not that good in terms in the sense that they take much more time, and they
are often not able to detect the test for many of the faults. So, the fault coverage is much

less and there is another observation we made test generation is lower than fault simulation. So, with this we come to the end of this lecture.

Thank you.