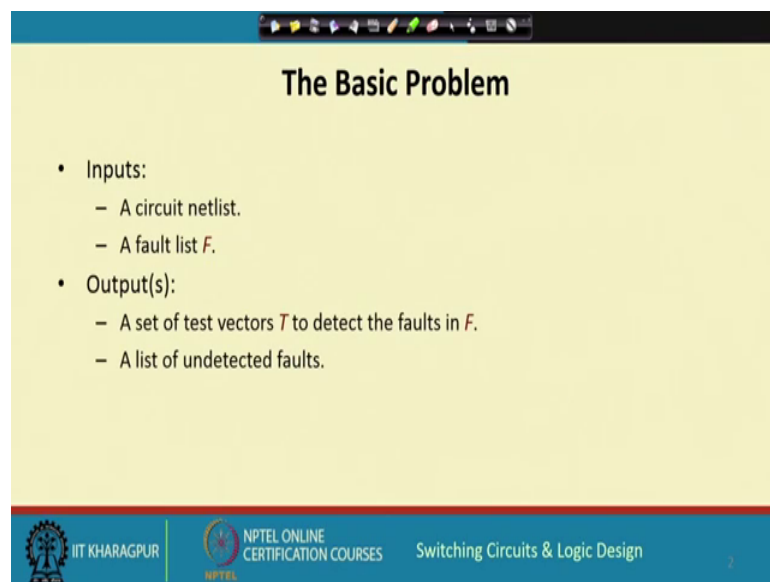


**Switching Circuits and Logic Design**  
**Prof. Indranil Sengupta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 57**  
**Test Generation Pattern**



We now talk about the problem of Test Pattern Generation. Earlier, we have already seen that given a circuit, gate level circuit is example you talked about. We can define a fault model. Stuck at fault model website is the most widely used fault model and in all the examples we shall be showing now we shall be considering this single stuck at fault model. So, the title of the lecture is Test Pattern Generation.

(Refer Slide Time: 00:50)



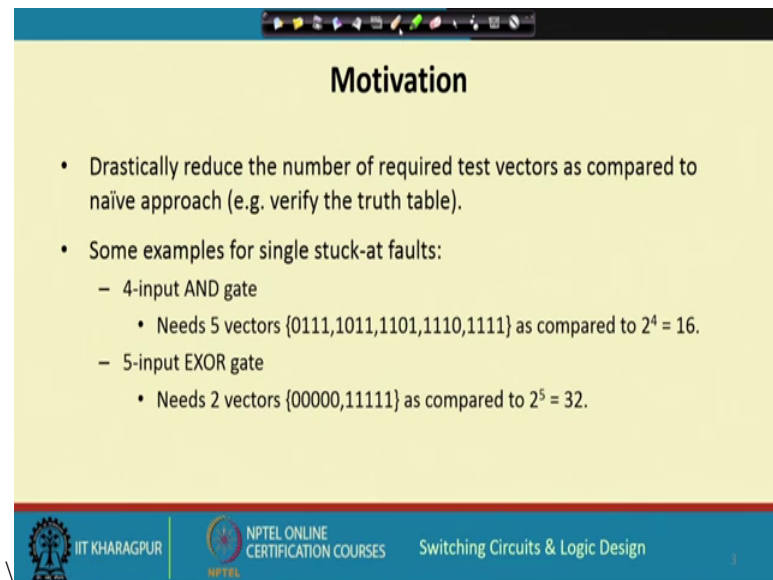
**The Basic Problem**

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

 IIT KHARAGPUR |  NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Design

So, let us see what is the scope of test pattern generation. So, we have a circuit we have a circuit  $C$ . We have a list of faults  $F$ ; we call it a fault list. Then we have a set of test vectors that we want to generate. This will be our output, our input is the circuit and the list of faults. The objective of test pattern generation is to generate a set of test vectors to detect all the faults in  $F$ . Like for example; we took examples earlier. If I have a three input XOR gate, my fault list will concern; see there are 4 lines. So, my fault list the size of the fault list will be 8 there will be 8 faults, but my test set will consist of only 2 patterns; 000 and 111. So, given my circuit this should be my output ok, this is what we want to solve here ok.

(Refer Slide Time: 02:09)



**Motivation**

- 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  $2^5 = 32$ .

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Design

So, this motivation we have already taken some examples earlier, but let us again repeat this. Test generation can drastically reduce the number of required test vectors; like the example of an Exclusive OR gate that we have taken, that any odd input XOR gate will require only 2 test vectors for testing all single stuck at faults. You take an AND gate with 4 inputs and 1 output. Normally, speaking the number of input combinations are 16, so, for truth table verification you would be requiring 16 test patterns, but we have seen that only 5 test patterns are sufficient to detect all signal stuck at faults.

So, there can be a drastic reduction in the number right. So, these are some of the examples that I am showing here. For a 4 input AND gate these are the 5 test vectors you need. For any odd input XOR gates you need only 2 test vectors. So, from 16 you come to 5, from 32 you come to 2; that is a drastic reduction.

(Refer Slide Time: 03:40)

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

$T = \{0011, 0101, 0110, 0111, 1011\}$

So, let us take an example net list with 3 gates. You see there are 2, 4, 5, 6, 7; 7 lines. So, there are 14 single stuck at faults.

So, I am just showing, this is not the process how we are generating. Let us see the test vector 1011, what are the faults it can detect? 1011, 10 means here I get 1, here I get 1, here I get 1. So, you see a fault will be detected if the output G in the absence of fault Exclusive OR, the output in the presence of fault is 1, which means they differ. This is the necessary condition for detection of a fault.

Now, 1011 since the output G is 1, so, obviously, G stuck at 0 will be detected and E F are both 1. If either E is 0 or F is 0 then G will be 0, G will change. So, this fault will be there of course, E stuck at 0 is missing here, E stuck at 0 will also be there, E stuck at 0 will also be there and similarly here let us say either C or D, if either C or D becomes 0 then F will be 0 intern G will also be 0 will change. So, C 0 D 0 will also be there. On the other side, if A is 0 output of this OR gate will become 0 and hence G will be 0. So, A 0 will also be there.

So, like this if you check you will find that these stage vectors are able to detect these faults and you will find that these five test vectors are sufficient to detect all the 14 faults in the circuit it includes all the 14 faults just an example to illustrate. So, this I suggest you verify all the other test patterns and check whether this faults are getting detected ok, fine.

(Refer Slide Time: 06:15)

### Generating Test from Truth Table

- How to generate test for the stuck-at-0 fault  $\alpha$ ?

Condition:  $f \oplus f_\alpha = 1$

Inputs (a b c)	f	$f_\alpha$
0 0 0	0	0
0 0 1	0	0
0 1 0	0	0
0 1 1	0	0
1 0 0	0	0
1 0 1	1	1
1 1 0	1	0
1 1 1	1	1

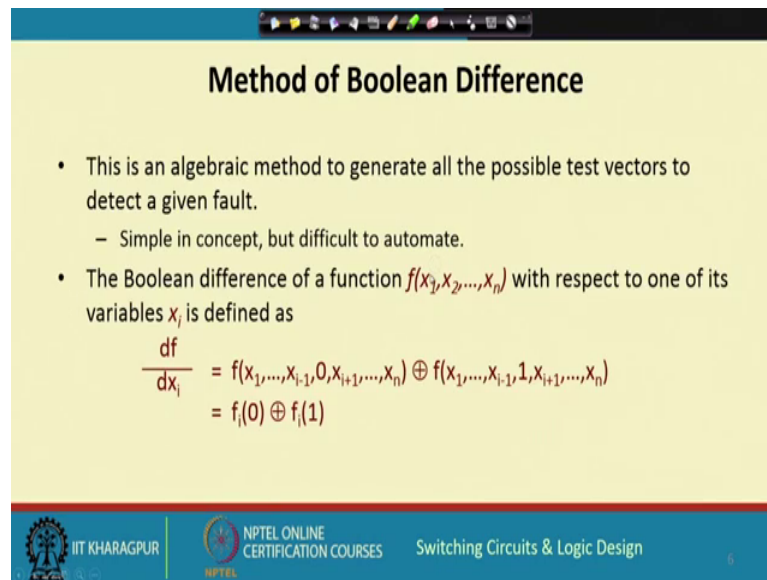
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Design

Now, let us let us talk about some of the methods of generating test in a systematic way. Let us consider that I have a circuit which implements the function  $f$  equal to  $a b$  or  $b c$ . Here I am giving method where I am showing a truth table. For the inputs  $a b c$  these are the  $f$  the output values  $a b$  or  $b c$  a  $b$  or  $b c$  sorry a  $b$  or a  $c$  not  $b c$  a  $b$  or a  $c$  a  $b$  a  $c$ . Now, suppose we consider a fault on this line stuck at 0, we call it  $\alpha$ . If this line is stuck at 0 that mean this is always 0. So, effectively the function in the presence of fault will be only  $a c$  because if this is 0 the term  $a b$  will disappear this will become 0. So, the function  $f_\alpha$  in the presence of fault will be only  $a c$ .

So, whenever  $a$  and  $c$  are 1 then only they will be 1. So, from the truth table the idea is as follows, let us just omitted. You look at all the rows and you find out the rows where  $f$  1  $f_\alpha$  are different; you see that they are differ here. So, 110 will be your required test vector. If we apply 110, then in the absence of fault output will be 1 and in the presence of the fault output will be 0. So, you can detect. So, this will be your required test vector. Well, but the problem here is that well you see you have to start with the truth table. And constructing truth table for a large function is not easy.

Suppose I tell you that I have a 30 variable function. 2 to the power 30 means how much? It is about 1 billion ok. So, it is huge. So, you really cannot construct a truth table of that size and compare the fault free and faulty output and see why they differ. So, this method all that is good for understanding this cannot be use for large functions.

(Refer Slide Time: 09:17)



**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

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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Design

So, let us talk about a systematic method called the method of Boolean differences this is an algebraic method meaning that you have to carry out some algebraic calculation in order to find out what test vectors can detect the fault. Let us try to understand the basic principle behind the method first.

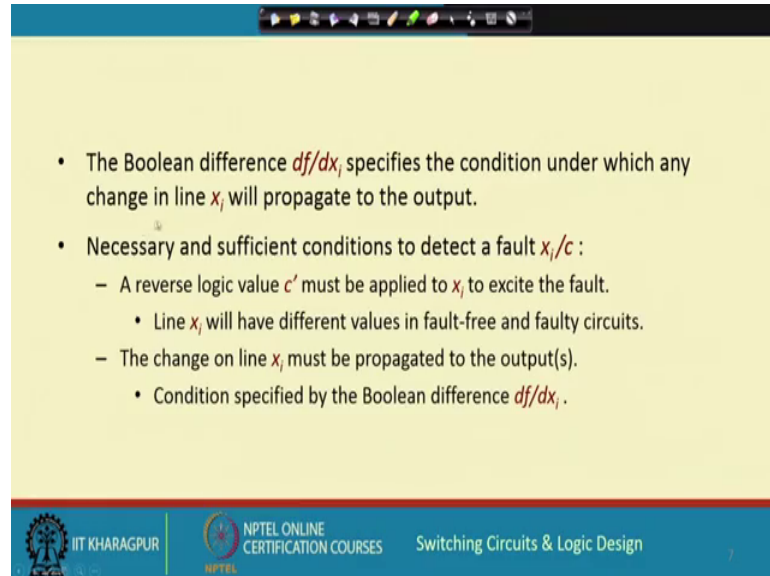
Now, we define something called Boolean difference. Consider an invertible function  $f$ ,  $f$  is a function of  $n$  variables  $X_1$  to  $X_n$ . So, when we define the Boolean difference it is with respect to some variable  $X_i$  let us say  $X_i$ . The idea is very simple. I have the function. I am talking about a variable  $X_i$ , first I set  $X_i$  equal to 0, let us look at what the function is. I set  $X_i$  equal to 1, let us say what the function is.

Then we take the Exclusive OR of these two functions that is what is defined as the Boolean difference. Let us see Boolean difference notationally this is expressed in the derivative notation  $\frac{df}{dx_i}$  and as I had said first in the function you set  $X_i$  equal to 0 then in the function you set  $X_i$  equal to 1 and take the XOR. The function with  $X_i$  equal to 0 in short there is a notation  $f_i(0)$ ;  $f_i(0)$  means, the function with  $X_i$  equal to 0 and  $f_i(1)$  means, the function with  $X_i$  equal to 1 ok.

So, what does the Boolean difference really mean? You say I am taking the Exclusive OR of two functions. Now when will the exclusive or of two function be equal to 1? This will be 1 if either their values are 0 1 or 1 0 ok, which means this condition says that whenever  $X_i$  changes the function also changes this is at indirect

way of saying it. Boolean difference equal to 1 means, when the variable  $X_i$  changes the output also changes that condition ok fine. This is what Boolean difference means.

(Refer Slide Time: 12:35)



- The Boolean difference  $df/dx_i$  specifies the condition under which any change in line  $x_i$  will propagate to the output.
- Necessary and sufficient conditions to detect a fault  $x_i/c$  :
  - A reverse logic value  $c'$  must be applied to  $x_i$  to excite the fault.
    - Line  $x_i$  will have different values in fault-free and faulty circuits.
  - The change on line  $x_i$  must be propagated to the output(s).
    - Condition specified by the Boolean difference  $df/dx_i$ .

Now, this is what I just now told. Boolean difference specifies the condition under which change in line  $X_i$  will make the output change, we say will propagate to the output, the change will propagate. Now, talking about a fault on line  $x_i$  stuck at  $c$ ,  $c$  can be either 0 or 1; we are talking about either  $x_i$  stuck at 0 or  $x_i$  stuck at 1. So, to detect a fault there are two things to be satisfied. Let us see let us take a very simple example to illustrate what I am saying. Suppose this is my line  $x_i$  and I am trying to detect a fault  $x_i$  stuck at 0.

To detect a fault  $x_i$  stuck at 0, the first thing I have to satisfy is I have to apply a reverse logic value at the line where I am trying to detect the fault because if I apply a 0 then stuck at 0 also it will be 0 and 0 is also 0; so, I cannot see any change. So, first condition is the reverse logic value  $c'$  must be applied to  $x_i$ , we call here that we are exciting the fault. Then the change on this line  $i$ , there is a change we are forcing on this line  $i$ . Normally, it will be 1 if there is a fault it will be 0. So, this change must be propagated to the output. How is the change we are expressing?

We are expression by the Boolean difference. So, these two things must be combined together.

(Refer Slide Time: 14:17)

- 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  $x_i'$        $df(X)/dx_i$

IIT KHARAGPUR      NPTEL ONLINE CERTIFICATION COURSES      Switching Circuits & Logic Des

So, what we are saying that the two things combined together we are saying that if we want to detect  $X_i$  stuck at 0 we have to set  $X_i$  to 1 first, which we denote by  $X_i$  and the change must be propagated to the output, which we are expressing by the Boolean difference. This and this must be 1 which means,  $X_i$  should be 1 and also Boolean difference should be 1. Similarly, for  $X_i$  stuck at 1 I have to set  $X_i$  equal to 0. So, that here I am writing  $X_i$  bar. So,  $X_i$  bar means  $X_i$  has to be 0 and Boolean difference has to be 1.

That means I am exciting the fault and I am propagating the change to the output both the conditions must be true simultaneously. This is the basic idea behind the method of Boolean difference. Let us take an example. This is what I had said excite the fault either  $X_i$  or  $X_i$  bar and propagate the fault is a Boolean difference ok.

(Refer Slide Time: 15:34)

$F = (A+B)C' + CD$   
 $dF/dC = (A+B) \oplus D$   
 $= AD' + BD' + A'B'D$

For C/0 ::  $C \cdot dF/dC = 1$   
 $ACD' + BCD' + A'B'CD = 1$

For C/1 ::  $C' \cdot dF/dC = 1$   
 $AC'D' + BC'D' + A'B'C'D = 1$

Generate tests for C/0 and C/1

1	X	1	0
X	1	1	0
0	0	1	1

1	X	0	0
X	1	0	0
0	0	0	1

Let us take a slightly complex circuit like this. Here there is a circuit with 5 gates. So, if you compute the function realise you see function is this A plus B C bar and or C D. So, if you compute the Boolean difference let us say we want to find out some faults on line C.

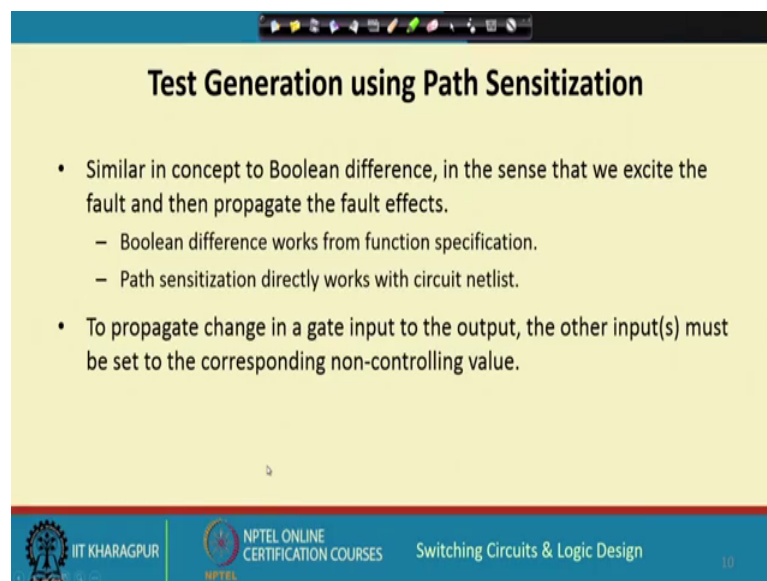
So, we are computing  $dF/dC$ . First we are setting C equal to 0, if in this F you set C equal to 0 the second term will disappear and C bar will be 1. So, only A plus B then C equal to 1, C equal to 1 means C bar is 0 this will disappear only D. So, XOR D. So, if you expand this I am not showing the steps, the final expression will be this ok. So, when you are trying to detect C stuck at 0, the condition will be C and Boolean difference equal to 1. Boolean difference I have already calculated C and this. So, it will be  $ACD' + BCD' + A'B'CD$ .

From here directly you can know what are the test vectors will be generated. See  $ACD' + BCD' + A'B'CD$  bar means A is 1 there is no B. So, B is do not care, C is 1, D is 0. The second term there is no A. So, A is don't care B C D is 0 here 0 0 1 1. Similarly, for C stuck at 1 you have a C bar here. So, the expression will be like this. So,  $AC'D' + BC'D' + A'B'C'D$  bar like this. So, you see the good thing is that for this fault you are getting all test vectors which can detect it; don't care means it can be either 0 or 1. So, it can be 1 0 1 0 or 1 1 1 0 and this 0 1 1 1 and 1 1 1 0 we have already included and 0 0 0 1.



So, there can be 2 3 4 possible test vectors here ok, similarly here. So, this is the method of Boolean difference where with respect to the input if you take the Boolean difference then you can directly generate all the possible test vectors that can detect either stuck at 0 or stuck at 1 faults on those lines ok. This is an algebraic method, it generates all faults, but because it is algebraic it is difficult to automate it is difficult to write a program ok. So, practical test generation tools are normally not written using this Boolean difference method because of this problem.

(Refer Slide Time: 18:54)



**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.
- To propagate change in a gate input to the output, the other input(s) must be set to the corresponding non-controlling value.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Design | 10

So, I am giving an idea what kind of methods are used there. There is something called path sensitization, I will just give you the basic idea behind this. So, you will have an idea. See for testing we still follow the same principle as Boolean difference. We have to excite the fault means, reverse logic value I have to apply and propagate the fault effect to the output. The difference between Boolean difference is that for Boolean difference we started with the function expression and did algebraic manipulation, but in the method of path sensitization we start with the gate level circuit and whatever we do we work at the gate level circuit only ok, this is the difference. So, let us see what will happen.

And another thing is that when you are talking about propagating let us say let us take an example. Suppose, I have a gate here the output is going to some other gate, let us say it is a NAND gate; this is the output and I am talking about a fault on this line, this is

alpha. So, some change on this line must be propagated to the output. So, what we are saying is that whenever some gates are encountered in between like it is a NAND gate what should be the other input because if I apply a 0 on the other input the output will be permanently 1, the change will not be propagated.

So, I have to apply 1 on the other input. This is called non controlling value. Similarly, for AND gate also it will be 1, for an OR gate it should be 0, for a NOR gate it should also be 0 ok, then only the change will propagate. This is what you mean by non controlling value. So, we will take an example.

(Refer Slide Time: 21:07)

**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.

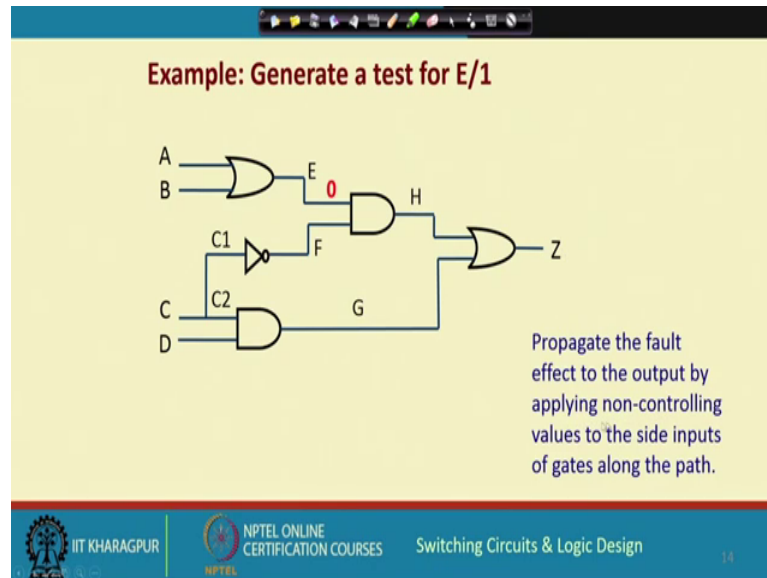
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Des

But the basic principle just works in two phases. So, what you do? I will illustrate these steps with an example do not worry. So, at this site of the fault suppose we want to detect the fault. So, on the line where the fault is there, you assign a logic value that is reverse or complement of the polarity of the fault. Like in a line  $f$ , if you want to detect  $f$  stuck at 0 then you apply  $f$  equal to 1, the reverse value.

Then forward drive phase; from this site of the fault the line where the fault is there you select a path to one of the outputs. You sensitize the path by assigning non controlling value we just now said what is non controlling values. So, that change at the site of the fault will be propagating to the output. After we have done this enough to do a backward trace phase because whatever logic values have assigned you have to backtrack. So, that all the primary inputs are assigned appropriate values.

So, once we have done that you know what are the inputs that have to be applied. This steps, let me explain with the help of an example then it will be clear.

(Refer Slide Time: 22:41)



Take a circuit like this, there are 5 gates and suppose we want to detect a fault on a line E; this E is the output of this OR gate E stuck at 1. So, in the first step what we said? That a reverse logic value must be applied at the site of the fault. So, because we are talking of E stuck at 1 what we do? We apply a 0 at the site of the fault, the reverse logic value right. This is the first step.

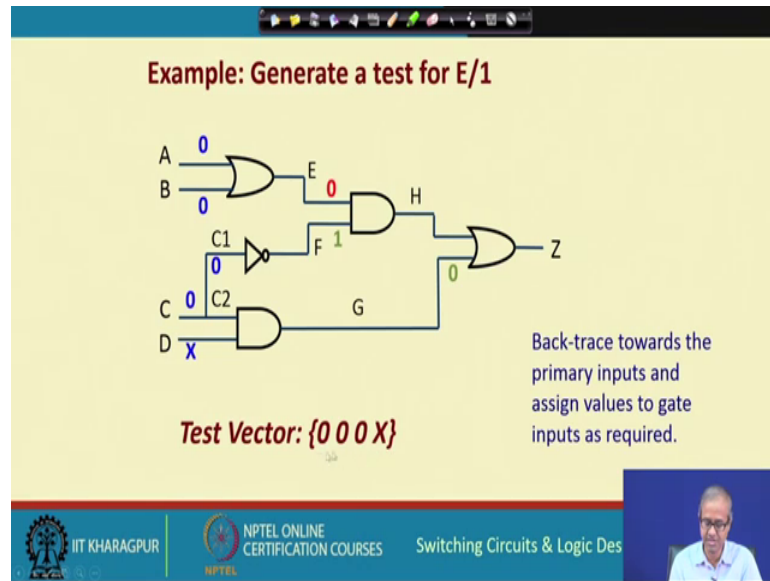
Then from the site of the fault we have to select a path to the output. Here there is a one path only from this H to Z. So, next step we have to propagate the fault effect to the output. So, there is only one path no option and we have to apply non controlling values. See change on line E must propagate as a change on Z. For that you see I have an AND gate here, I have an OR gate here. I told for an AND gate non controlling value is 1, for an OR gate non controlling value is 0. So, what will happen is you will apply a 1 here and you will apply a 0 here.

So, if we do this then it is guaranteed that any change on line E will be propagating as a change on line H and a change on line Z output right. So, your forward drive phase is done. Now you have to do backward propagation. You see backward propagation what is the basic idea that you have set this line E to 0, line F to 1 and G to 0, but what we have not done it is that you do not know what values have to apply to A B C D. So, from E

equal to 0 I have to propagate back, from F equal to 1 again I have to propagate back and from G equal to 0 again I have to propagate back.

This is back this is called back propagation. I have to finally, assign some values to A, B, C and D ok.

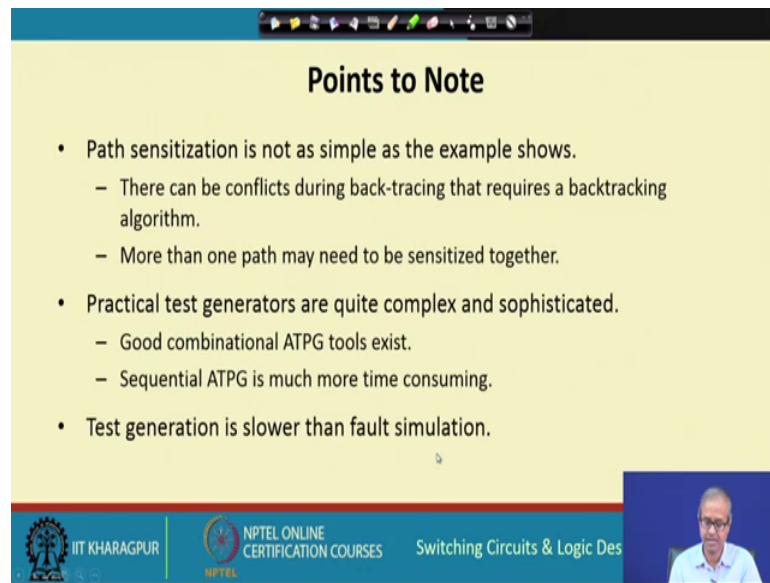
(Refer Slide Time: 25:20)



Back trace towards the primary inputs and assign values to the gate inputs. So, to make the output of this OR gate 0, both the inputs must be 0 0 A 0 B 0 done. Now, F is 1, so, this NOT gate should be 0, which means, C is 0 and if C is 0 the value of G is automatically 0, you do not have to do anything more it is already done; so, D can be don't care. So, you have got a test vector A 0 B 0 C 0 D don't care. You see this is the basic idea behind the method of path sensitization.

This is very simple in concept you see from the gate level netlist, you can simply propagate forward and backward and generator test. So, test vector will be this.

(Refer Slide Time: 26:17)



**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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | Switching Circuits & Logic Des

But the thing is not so simple if you remember something very clearly, that this path sensitization method is not as simple as this example shows because during backtracking or back tracing there can be conflicts. Some path may tell that an input A must be set to 0, but some other path may tell that A must be set to 1, which will not work. So, you will have to go back make some changes and again come back, lot of backtracking may be required in such cases ok or more than one paths may need to be sensitized together, but these are a things I am not discussing here.

But you should remember this because the practical tools are much more complex than what I have shown in the example. Very good automated test pattern generation, it is called ATPG; Automated Test Pattern Generation, such ATPG tools exist and sequential ATPG tools are more time consuming, we shall see later how sequential circuit test pattern generation can be handled and as I had said; we shall not be discussing fault simulation here in this course, but you should remember one thing that the process of test generation because of it is complexity is typically slower than fault simulation.

So, with this we come to the end of this lecture. In the next lecture we shall be talking about some of the techniques that we can follow for generating test for sequential circuits. Namely, generic strategy called design for testability. This we shall be discussing in the next lecture.

Thank you.