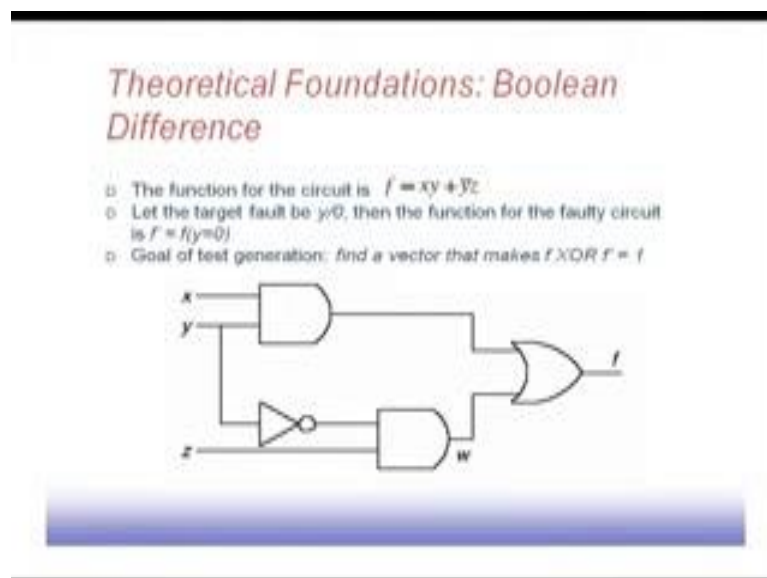


Digital VLSI Testing
Prof. Santanu Chattopadhyay
Department of Electronics and EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 18
Test Generation (Contd.)

Next we will discuss on this test pattern generation technique, but before going to the details of this test generation algorithms, we need to discuss about some of the theoretical foundations of this testing process.

(Refer Slide Time: 00:34)

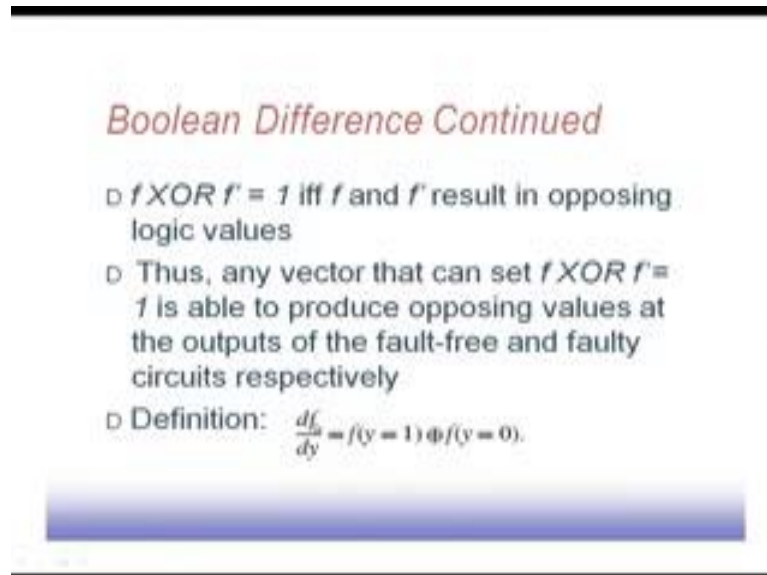


They are based on one Boolean operation which is known as Boolean difference operation. So, this circuit is implementing realising the function f equal to $x y$ plus y bar z .

Now, suppose we are targeting the fault y stuck at 0. So, then the fault are function for the faulty circuit is f dash with which is a modified version of f with y equal to 0. So, if this modified circuit is f dash then goal of the test generation is that under this faulty condition this $f \oplus f$ dash. So, this output should be equal to 1 that is we have got this circuit. So, if we have, if we create another circuit which is f dash and run those 2 circuits simultaneously with this particular vector that we have found and then there is there is hypothetical XOR gate at the output of we which combines the outputs of these 2 copies of the function f and f dash and this output is 1. So, we have to find a vector such

that is a $f \text{ XOR } f'$ equal to 1. So, this is determined by this Boolean difference operator.

(Refer Slide Time: 01:47)



Boolean Difference Continued

- $f \text{ XOR } f' = 1$ iff f and f' result in opposing logic values
- Thus, any vector that can set $f \text{ XOR } f' = 1$ is able to produce opposing values at the outputs of the fault-free and faulty circuits respectively
- Definition: $\frac{df}{dy} = f(y=1) \oplus f(y=0)$.

Naturally $f \text{ XOR } f'$ will be equal to 1 if and only if f and f' they are in opposing logic values that is that is true then only the XOR gate value will be 1. So, any vector that can has the set $f \text{ XOR } f'$ equal to 1 is it can put opposite values to the f and f' circuits as a result we can have this output detected the XOR gate will be equal to XOR gate output will be equal to 1 and we can say that the fault is getting detected.

We define this operator differential operator $\frac{df}{dy}$ is equal to $f(y=1) \text{ XOR } f(y=0)$. So, if we apply, this is the function, this is the definition of these $\frac{df}{dy}$. So, it is differentiation of function f with respect to variable y . So, it is the equal to the function f with y set to 1 XOR with the function f with y set to 0.

(Refer Slide Time: 02:47)

Boolean Difference Example

□ To excite the fault $y/0, y=1$

□ Thus, $y \cdot f(y=1) \oplus f(y=0) = y \cdot (x \oplus z)$

$$= y \cdot (xz + \bar{x}\bar{z})$$

$$= xy\bar{z} + \bar{x}yz$$

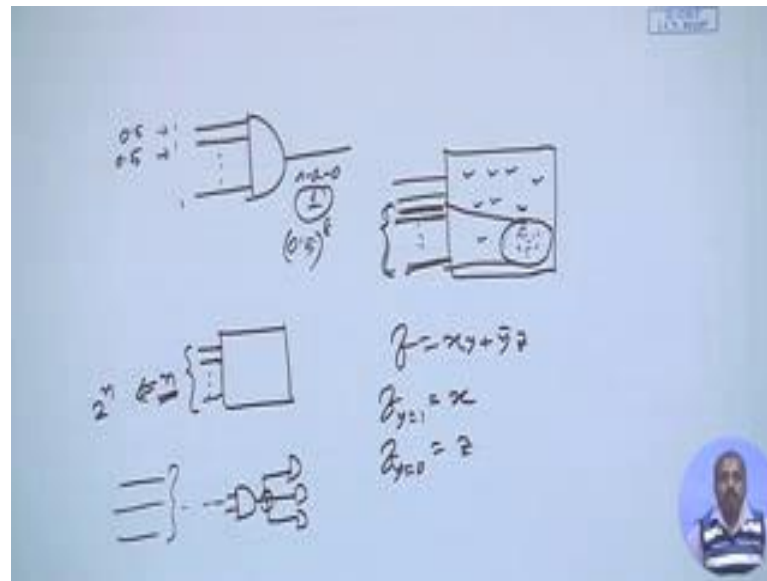
xyz= 110 or 011 can detect the fault

In this particular case, if we do this thing. So, to excite y equal to excite the fault y stuck at 0, 1, one thing that is needed is first of all this line y should be equal to 1 and the remaining part will ensure that f y equal to 1 and f y equal to 0. So, these 2 functions x their XOR being 1 will ensure that the fault actually propagates. So, as you have seen in the test generation process, there are 2 parts 1 part is excite the fault and the second part is to propagate the fault.

For exciting the fault, we need the line to the particular point to be set to equal to the opposite of the fault that we are going to detects. So, you since in this case, we are going to detect y stuck at 0. So, we would like to put y to be equal to 1. So, this is one condition to be satisfied, the y must be equal to 1 second condition to be satisfied is the XOR of faulty and fault free circuits they should be equal to 1. So, that is given by the differentiation of f with respect to y . So, f y equal to 1 XOR with f y equal to 0.

Now, the function that we had was $x y$ plus $y \text{ bar } z$ the function f , we had is $x y$ or $y \text{ bar } z$.

(Refer Slide Time: 04:10)



Now, if we take these 2 cases that is f with y equal to 1 f with y equal to 0. So, this function is x and f with y equal to 0 this function is z . So, these 2 are XOR. So, you see in this expression that this y part is for putting this y line to 1 and then the remaining part is for the propagation this XOR of these 2. So, it is y and x XOR z . So, if you expand it ultimately we get $x y z$ bar and x bar $y z$.

These are the 2 different test vectors that can be applied to check whether that particular fault has occurred or not. So, you see for a quick check we can see that in both the cases, y line has been set to 1. So, we are trying to take check y equal to 0 y stuck at 0. So, it is putting y to be equal to 1. So, for the excitation part is satisfied, now this x is 1 and this z is 0. So, since x and y both are 1. So, this fault free output is or this fault free output is 1 and if the fault is present then also this output will be 1, on the other hand this y is coming here. So, why this line is 0 and z is 0. So, this z is 0. So, you are getting a 1 here, but if you are having the fault that is y is stuck at 0 then this and gate output will be 0. So, as a result this will be this you will get a 0 here and this z line is 0. So, z line this line will be 0. So, we will get is 0 at this point. So, this way this 0 that we had is propagating to the output.

(Refer Slide Time: 06:18)

Another Example

Let target fault be $w/0$

$w \cdot \frac{df}{dw} = 1$

$\Rightarrow w \cdot (f(w=1) \oplus f(w=0)) = 1$

$\Rightarrow w \cdot (1 \oplus xy) = 1$

$\Rightarrow w \cdot (\overline{xy}) = 1$

$\Rightarrow w \cdot (\overline{x} + \overline{y}) = 1$

$\Rightarrow w\overline{x} + w\overline{y} = 1$

But: $w = \overline{y} \cdot z$

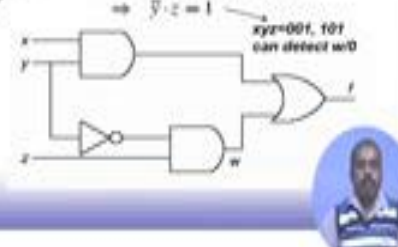
$w \cdot \overline{x} + w \cdot \overline{y} = 1$

$\Rightarrow \overline{y} \cdot z \cdot \overline{x} + \overline{y} \cdot z \cdot \overline{y} = 1$

$\Rightarrow \overline{x} \cdot \overline{y} \cdot z + \overline{y} \cdot z = 1$

$\Rightarrow \overline{y} \cdot z = 1$

$xyz=001, 101$
can detect $w/0$



That way it can detect this particular fault. So, it can detect that y stuck at 0 fault by the first test pattern. So, if you just say try for the next one also you will see that it is able to detect this particular fault y stuck at 0. So, there is a more detailed example here. So, in this particular circuit suppose we are trying to check for this particular fault w stuck at 0.

Previously y that we have taken, y was a primary input in this case w is a internal line of the circuit. So, for this internal line of the circuit again we need to satisfy this condition we need to excite that fault by setting w to 1 because we are trying to check for w stuck at 0. So, we need to put w to 1 and then this propagation which is df/dw should be equal to 1. So, therefore, $w \cdot df/dw$ this whole thing should be equal to 1.

If we expand this, we need to do this thing, df/dw , we need to find out. So, df/dw if you want to find out then it is df/dw is f with w equal to 1 XOR with f with w equal to 0. So, if w equal to 1 then naturally this gate output is 1. So, it is 1 and w equal to 0, w equal to 0. So, this puts the value to be equal to this $x \cdot y$. So, this is coming here. So, it is $1 \text{ XOR } x \cdot y$. So, it is $x \cdot y$ bar. So, w and $x \cdot y$ bar should be equal to 1. So, we just expand it by de Morgan's law. So, we get $w \cdot x$ bar plus $w \cdot y$ bar equal to 1. So, this is the condition

Now, w is given by y bar z . So, if you substitute it here. So, y bar $z \cdot x$ bar plus y bar $z \cdot y$ bar equal to 1. So, x bar y bar z and y plus or y bar z should be equal to 1. So, if you just for that simplify it we get y bar z equal to 1. So, y bar z equal to 1. So, x becomes a do not care. So, we do not need that test generation process need not assign any value to x.

So, value can be 0 or 1, but y should be 0 and z should be 1. So, this is the test these are the 2 test patterns should to be applied for checking w stuck at 0.

In this way for any circuit any combinational circuit you want to detect a particular fault. So, we can do this thing we can take that particular fault at that line excite that particular fault by giving proper value to it and then propagate it by taking the differential of the function with respect to that particular line.

(Refer Slide Time: 08:48)

A Third Example

□ Fault: $z/0$

$$z \cdot \frac{df}{dz} = 1$$

$$\Rightarrow z \cdot (f(z=1) \oplus f(z=0)) = 1$$

$$\Rightarrow z \cdot (xy \oplus xy) = 1$$

$$\Rightarrow z \cdot 0 = 1$$

$$\Rightarrow \text{UNSATISFIABLE}$$

This fault is untestable!


Another example, suppose we want to check z stuck at 0. So, again the same $z \cdot \frac{df}{dz}$ should be equal to 1. So, $\frac{df}{dz}$ if you go so that is given by z and f z equal to 1 f of z equal to 1 and f of z equal to 0 f of z equal to 1 if z is stuck at 1. So, you see that this AND gate becomes a 2 input and gates x y and this z if z is 0 then it is definitely 0. So, z equal to 1 XOR z equal to 0 this part becomes 0. So, this is x y, this x y XOR x y equal to 1, x y XOR x 0. So, we see that we want z and 0 to be what is not satisfiable. So, if it is not satisfiable naturally I cannot have any test pattern that can detect this particular fault that z stuck at 0. So, this becomes an untestable fault.

The fault is untestable fault. So, if we can detect all as we have discussed previously. So, if we can detect all the untestable faults in the circuit then we can remove them from our consideration in the test generation process. So, this is the theoretical way of finding a fault to be untestable.

(Refer Slide Time: 10:04)

Wrap Up on Boolean Difference

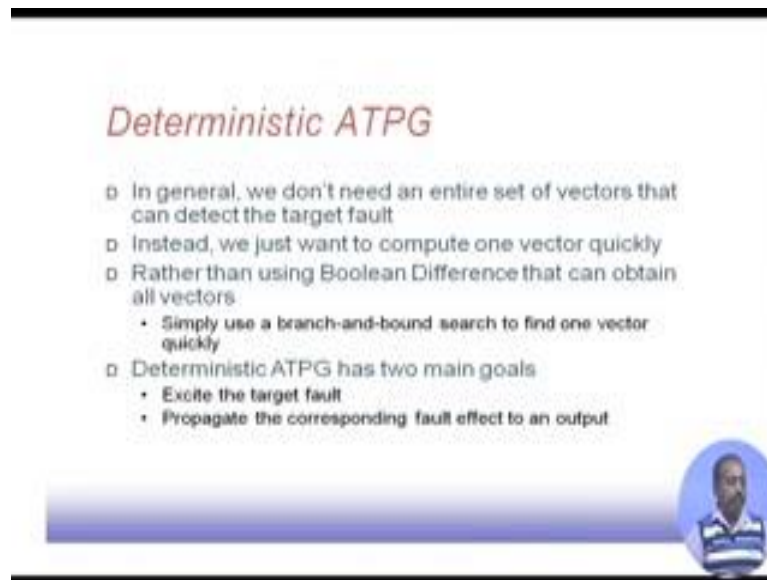
- ▷ Given a circuit with output f and fault a/v .
- ▷ The set of vectors that can detect this fault includes all vectors that satisfy

$$(a = \bar{v}) \cdot \frac{df}{da} = 1$$


In general we can say that given a circuit with output f and the fault to be detected α stuck at some value v . So, α is the signal line and v is the value stuck at 0 or stuck at one like that the set of vectors that can detect this fault includes all vectors that satisfy this particular condition α equal to \bar{v} the opposite of the fault that we have to excite.

α must be equal to the opposite of the fault α equal to \bar{v} and $\frac{df}{d\alpha}$ should be equal to 1. So, this is the formula to be applied for generating the test patterns for the particular fault α stuck at v .

(Refer Slide Time: 10:49)



Deterministic ATPG

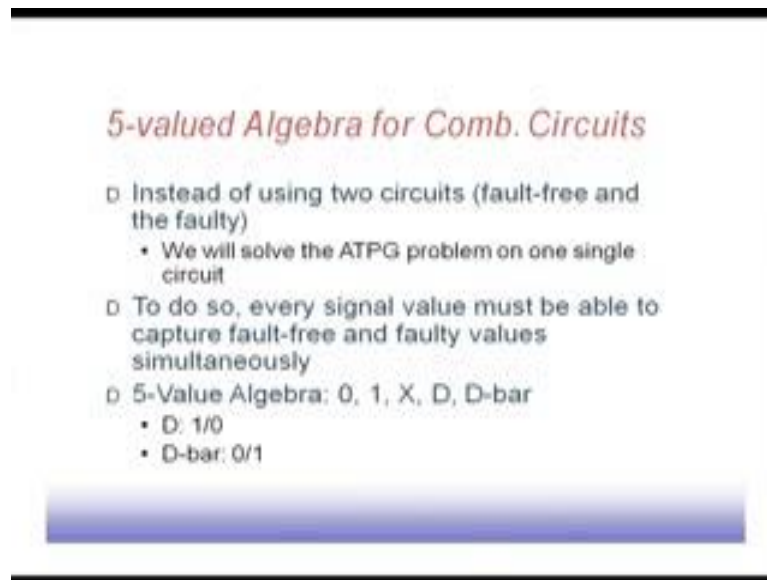
- In general, we don't need an entire set of vectors that can detect the target fault
- Instead, we just want to compute one vector quickly
- Rather than using Boolean Difference that can obtain all vectors
 - Simply use a branch-and-bound search to find one vector quickly
- Deterministic ATPG has two main goals
 - Excite the target fault
 - Propagate the corresponding fault effect to an output

In general, we do not need a entire set of vectors that the Boolean difference method that we have seen the advantages that it gives you all possible test patterns that can detect a particular fault, but it is it is not required in many cases we may be satisfied with having a single test pattern for a fault or maybe a few test patterns for a particular fault. So, that depends on the application that you are cutting very very few cases. So, we will require all patterns that can detect a particular fault. So, that is generally not needed.

rather than using Boolean difference that can obtain all vectors, we can use some sort of branch and bound search technique to find one vector quickly and then we can then we can do this fault simulation to see what this way to what more which more faults this vector can detect and then try out with the remaining faults dropping all the known all the detected faults from the set.

So, this deterministic ATPG it has got 2 goals excite the target fault and then propagate the corresponding fault effect to an output. So, any fault generation algorithm will look into. So, they basically do this thing somehow they need to excite the fault. So, we have to put the opposite of the stuck at value at that point. So, that is the fault excitation and we need to propagate the faulty response of that point to some primary output. So, these are the 2 things to be done by any ATPG algorithm.

(Refer Slide Time: 12:25)



5-valued Algebra for Comb. Circuits

- Instead of using two circuits (fault-free and the faulty)
 - We will solve the ATPG problem on one single circuit
- To do so, every signal value must be able to capture fault-free and faulty values simultaneously
- 5-Value Algebra: 0, 1, X, D, D-bar
 - D: 1/0
 - D-bar: 0/1

In many of the ATPG algorithm, you will be using a 5 valued algebra instead of a instead of 2 valued logic. So, you will be using a 5 valued logic. So, what it does? What it helps is that we do not need to consider 2 copies of circuits. So, previously we was telling that we have we need to have 2 copies of the circuit a fault free circuit and a faulty circuit and see whether their responses are different or not. So, instead of having 2 different circuits, if we are having if we see that we have we have got only we will work with only 1 circuit then this 5 valued algebra is going to help us for doing the ATPG procedure without having multiple copies of the circuit.

A signal value; we must be able to capture the fault free and faulty value simultaneously. So, this is the requirements. So, 0 or 1, they are deterministic values because they can tell a line to be permanently 0 or is statically 0 or statically 1 on the other hand. So, sometimes we need to tell that if this point is in a fault free circuit, if it is d, if it is some value in the faulty circuit, it will the reverse of that. So, this is captured by introducing 2 new symbols d and d bar. So, 0 1 x they remain as it is. So, 0 is a static, 0 1 is static, 0 1 x is unknown and d and d bar, they d maybe they can pick up the value 1 or 0 if. So, d is picking up the value 1 then wherever in the in the circuit we have got d bar. So, they will they that will be equal to 0 and on the other hand if d value at some point is 0 then all points that at d bar in the circuit that have current logic assignment is d bar they are at value 1. So, they are actually complementary d and d bar are complementary to each

other by having this d and d bar symbols. So, we can make our test generation process simple.

(Refer Slide Time: 14:27)

Boolean Operators on 5-Valued Algebra

AND	0	1	d	\bar{d}	x
0	0	0	0	0	0
1	0	1	d	\bar{d}	x
d	0	d	d	0	x
\bar{d}	0	\bar{d}	0	\bar{d}	x
x	0	x	x	x	x

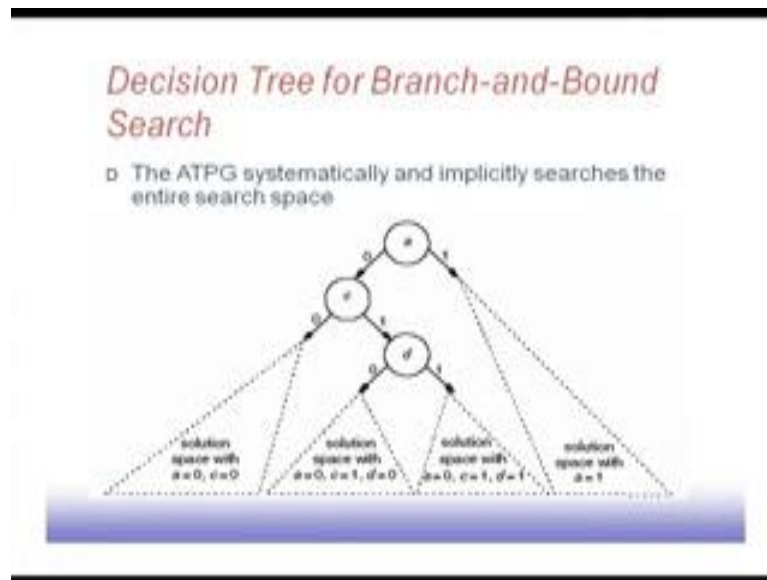
OR	0	1	d	\bar{d}	x
0	0	1	d	\bar{d}	x
1	1	1	1	1	1
d	d	1	d	1	x
\bar{d}	\bar{d}	1	1	\bar{d}	x
x	x	1	x	x	x

NOT	0	1
0	1	0
1	0	1
d	\bar{d}	d
\bar{d}	d	\bar{d}
x	x	x

if you just try to see how this algebra will work, we have got say 1 input is they put on to this row and in another input is put on to this column and operation. So, if 1 of the input in 0 then naturally and gate output is 0. So, there is no problem if one of the input is 1. So, AND gate output is equal to the other value. So, 0 1 d bar d, d bar x if one of the input is d then other input, d and 1 output is d and d bar is definitely 0 because they are complementary to each other. So, the other they are and must be equal to 0.

Similarly, d bar and d that is also equal to 0 then d bar and d bar is d bar. So, like that we can formulate this and table and if one of the values is unknown and other one is not a definite um controlling value then the value will remain unknown like these values they are remaining as unknowns in an or gate. So, we can similarly formulate these logic table and for NOR gate also we can formulate this logic table. So, it is we are showing it for only AND, OR and NOR. So, we can formulate several similar such tables for other logic functions like NAND, NOR, XOR. So, it does not matter. So, for all the logic families, I am knowing that d and d bar are complementary of each other. So, you can formulate this type of tables.

(Refer Slide Time: 15:57)



Now, how a test generation process can work now. So, suppose the ATPG, it systematically and implicitly searches the entire search space. So, this is my search space. So, first it will take up some variable put it to value 0 after putting it to value 0. So, it will, it may find that next input c . So, it puts the value c equal to 0 and in the process, it explodes the search space in this part where a equal to 0 and c equal to 0.

So, this part of search space is these has got all solutions that all of that can be obtained by setting a equal to 0 and c equal to 0, but it may So happen that for a particular fault setting a equal to 0 and c equal to 0 does not lead to any test pattern. So, after a few steps in this tree, we will find that we cannot it all the paths that are going for that. So, they are not going to lead to any test pattern there is no variable that can be set now to lead to the test pattern. So, in that case we can come back, we can backtrack and come with the other (Refer Time: 17:07).

For example suppose we some after sometime we find that setting c equal to 0 is never going to lead to any test patterns set for a particular fault. So, we come back try with c equal to 1. So, this way it can systematically search the entire space solutions space and come up with the test patterns. So, this ATPG algorithms they can systematically search the total solution space.

(Refer Slide Time: 17:31)

Backtracking

- The ATPG searches one branch at a time
- Whenever a conflict (e.g., all D's disappeared) arises, must backtrack on previous decisions

If $d=1$ also causes a conflict, backtrack to $c=0$

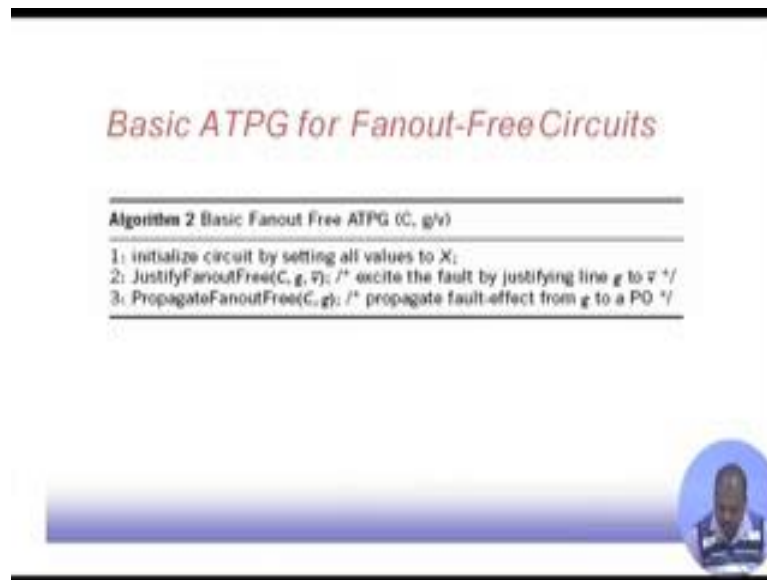
Conflict
backtrack

The diagram shows a circuit with nodes a, c, and d. Node a is connected to c, and c is connected to d. Node d is connected to two other nodes, one of which is labeled 'Conflict'. A dashed arrow labeled 'backtrack' points from the conflict node back to node c. A small inset image of a person is visible in the bottom right corner of the slide.

And backtracking will definitely be required as I was telling that if I find that d equal to plus a to equal to 0, c equal to 1 and d equal to 0, after coming to this point you may find a situation that for a conflict has occurred a conflict has occurred means all there is no more d that is present in the circuit. So, we can set that d 2, 0 or 1 and all d 's have disappeared. So, their conflict has occurred and all the input values they got specified and there are conflicting requirements for different paths for this excitation and propagation of this fault to the output.

There is conflicts, we can say that they this particular path is not going to lead to any test pattern. So, we come back we do a backtracking and then we come back here, now if you find that d equal to 1 also creates the problem then there is also conflict then we will try to explode the path which will come from here with c equal to 0. So, in this way this backtracking and this branch and bound. So, they are going to help us in generating the test pattern set.

(Refer Slide Time: 18:40)



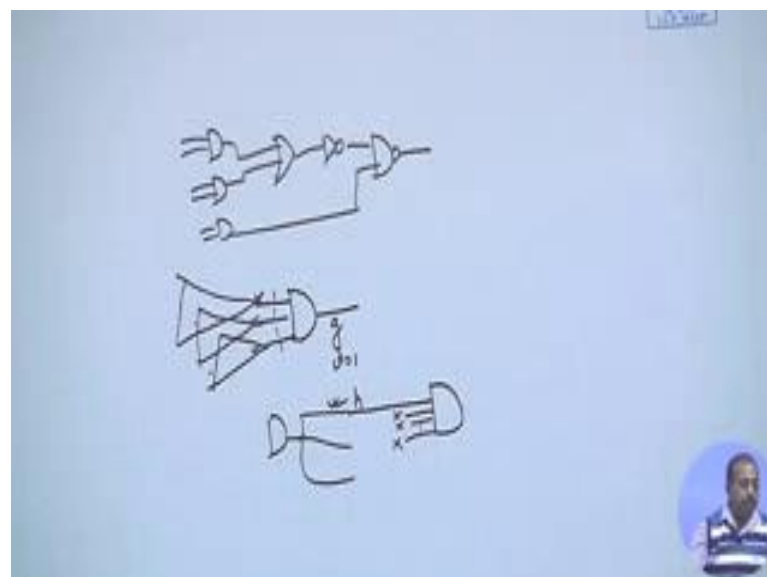
Basic ATPG for Fanout-Free Circuits

Algorithm 2 Basic Fanout Free ATPG (C, g, v)

```
1: initialize circuit by setting all values to X;  
2: JustifyFanoutFree( $C, g, v$ ); /* excite the fault by justifying line  $g$  to  $v$  */  
3: PropagateFanoutFree( $C, g$ ); /* propagate fault-effect from  $g$  to a PO */
```

Now, if you do not have any fanout in your circuit then this can be basic ATPGs. So, fanout means that there is no branching like if I have got some circuit here. So, these are the primary inputs and after sometime there is a gate and this gate output goes to several places. So, we have we say that this is a fanout point. Now if this type of situation is not there then what you get essentially is a tree type of circuit.

(Refer Slide Time: 19:14)




You have got some gates at the first level then these inputs are coming there then you have got some gate at the next level there maybe some other gates after that, but it is

It says that initialize circuit by setting all values to x. So, all input value; all input values all these a signal line values, they are set to x and we are trying to detect this particular fault signal line g stuck at some value v. So, then we call justify fanout free which is c g and v bar. So, this will what it will try to do it will try to excite the fault by justifying the line g to v bar. So, it will try to set the input values of the circuit such that this line g gets the value of g bar and after that it will also the justification has been done now this v bar pa response has to be propagated to the output. So, how can you do this? So, there must be some propagation procedure. So, this propagates fanout free. So, this c g, this will propagate this line g to at least one of the primary outputs.

The Justify Routine

Algorithm 3 JustifyFanoutFree(C, g, v)

```
1:  $g \leftarrow v$ ;  
2: if gate type of  $g$  == primary input then  
3:   return;  
4: else if gate type of  $g$  == AND gate then  
5:   if  $v == 1$  then  
6:     for all inputs  $h$  of  $g$  do  
7:       JustifyFanoutFree( $C, h, 1$ );  
8:   end for  
9:   else ( $v == 0$ )  
10:     $h$  = pick one input of  $g$  whose value ==  $X$ ;  
11:    JustifyFanoutFree( $C, h, 0$ );  
12:   end if  
13: else if gate type of  $g$  == OR gate then  
14:   ...  
15: end if
```



Now, otherwise, if the gate type of g , g is AND gate then if g equal to 1 then we need to justify all the inputs. So, what we essentially mean is that if this is the AND gate, this is the line g I am talking about now. So, if this if you want to put this line to equal to 1 then all these lines are to be put to 1. So, whatever input cone we have here for this for these

inputs. So, all of them, they need to be put to be equal to 1. So, somehow once must appear at all this point. So, that is what is told here that if gate type is used and gate if the gate type is and gate then this if the value that we want to put is 1 then for all inputs h of g we need to justify fanout free c h 1. So, all the inputs have to be justified.

Otherwise if v is equal to 0 then we can very easily we can take only 1 of the inputs and put it to 0. So, we pick up 1 input of g whose value is currently unknown. So, we do not want to put fall into any conflict. So, we pick up some value of g pick up some input of g whose value is currently unknown and try to put a 0 there and how can we put a 0 there. So, we can call this routine justify fanout free c h 0. So, it will put a 0 value at that particular point at the point h. So, this way it can continue, this is a recursive routine and if this is a recursive routine. So, if it is find after sometime that we cannot proceed further then it will pick up some other input and it will try out with that. So, if you are trying with one input it fails. So, it can try with another input and do this thing.

Similarly alpha other gate type if g is or then we have to just reverse this particular check v equal to 1 and v equal to 0, in this way we can figure out like all different types of gates what has to be done? So, once you know that you are circuit is consisting of these types of gates and you for each of them. So, you can write down their corresponding code. So, which will be doing this justification routing?

(Refer Slide Time: 23:28)

Example

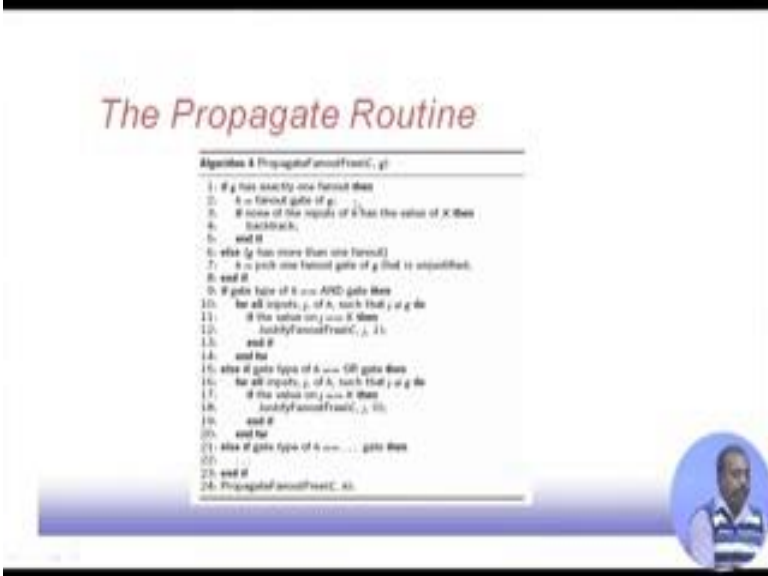
The recursive calls to `JustifyFanoutFree()`:

- call #1: `JustifyFanoutFree(C, g, 1)`
- call #2: `JustifyFanoutFree(C, a, 1)`
- call #3: `JustifyFanoutFree(C, f, 1)`
- call #5: `JustifyFanoutFree(C, c, 0)`

Here suppose in this particular circuit. So, we want to detect this fault g stuck at 0. So, first it will try to call; it will try to justify it will put a value try to put a value of 1 at g , it will justify fanout free function will be called with the circuit c with the line g going to 1. So, if this line g goes to 1 and then naturally we it has to call it has to put this one at both the inputs because this is an AND gate. So, it will call this justify fanout free C a 1. So, since this is a primary input. So, this will be immediately satisfied and a will be put to equal to 1.

Now, it will call this justify fanout free f to be equal to on 1e which since this is an inverter will recursively give a call to justify C stuck at 0 and then it will be doing this thing that it will be pa putting this C to be it is a primary input. So, it will put C to be equal to 0 after that it will try to. So, that is the justification of g . So, this particular line can be set to up to a values. So, g value, g line can be set to 0.

(Refer Slide Time: 24:41)



The Propagate Routine

Algorithm 4 PropagateFault(c, g)

- 1: if g has exactly one fanout then
 - 2: $h \leftarrow$ fanout gate of g
 - 3: if none of the inputs of h has the value of x then
 - 4: backtrack
 - 5: end if
- 6: else (g has more than one fanout)
 - 7: $h \leftarrow$ pick one fanout gate of g that is unjustified
 - 8: end if
 - 9: if gate type of h == AND gate then
 - 10: for all inputs p_i of h , such that $p_i \neq g$ do
 - 11: if the value on p_i == x then
 - 12: justifyFanoutFree(c, p_i)
 - 13: end if
 - 14: end for
 - 15: else if gate type of h == OR gate then
 - 16: for all inputs p_i of h , such that $p_i \neq g$ do
 - 17: if the value on p_i == x then
 - 18: justifyFanoutFree(c, p_i)
 - 19: end if
 - 20: end for
 - 21: else if gate type of h == ... gate then
 - 22: end if
- 23: end if
- 24: PropagateFault(c, h)

Next is the propagate routine. So, in this example we have seen that how this g line can be set to 1. Now we need to propagate this g through this OR gate to this primary output. So, that part will be done by this propagation routine. So, if g has exactly 1 fanout then we say suppose h is the fanout of g if none of the inputs of h has the value x then backtrack. So, this means we are trying to propagate, but there is only if there is only 1 fanout of g that line is h and the inputs of h are all none of the inputs of h are x that is all the inputs of h are already specified. So, we cannot do anything. So, then it will be then

we need to backtrack because they through this line h, we are unable to send the output send this line g to the output.

Otherwise g has more than one output. So, if g has more than out one output fanout 1 more than 1 fanout then it will pick up any of the anyone fanout of g with that is unjustified. So, it is not yet taken care of. So, it is it will tick a pick it up. Now it will try to propagate this g through this h. So, this h is AND gate and then for all inputs j of h such that j is not equal to g if the value on j is x then it will try to propagate. So, it is basically I have got the say this line has to be propagated now we see that this line. So, this is the line h that is that we want to propagate. So, it is coming from this and gate. So, this is line g. So, it has got many fanouts. So, out of that this fanout has being the chosen.

Now, it will try to see that other inputs of h other inputs of this and gate they need to be justified. So, that we can this h propagate. So, at present what is required is all these values must be at x. So, all those, if the value is on the value of j is equal to x then we need to put the value to be equal to 1. So, C j 1 that has to be done similarly if the type of the gate is OR gate, it is just the reverse logic. So, if you we need to put this j equal to where wherever we have got j equal to X, we need to put the j equal to 0, in this way for all the gates, all types of gates we can write down the routine. So, that it will propagate through the circuit that the line g will propagate to the primary output.

We will continue in the next class.