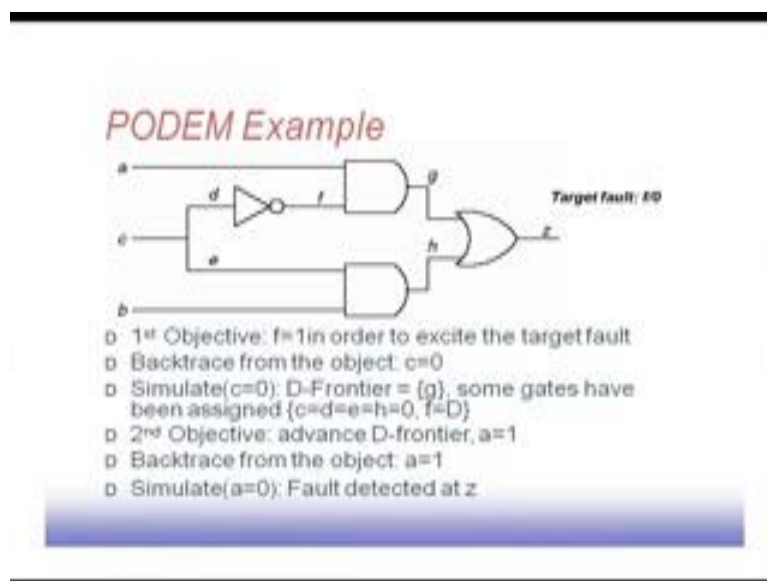


Digital VLSI Testing
Prof. Santanu Chattopadhyay
Department of Electronics and EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 20
Test Generation (Contd.)

In PODEM algorithm, the basic idea is that instead of this G frontier of this D algorithm, we take it to the primary input.

(Refer Slide Time: 00:23)



For a particular point to be justified, we figure out the primary inputs that are responsible that can control the value at that point and then try to set the primary input to some appropriate value. So, for example, in this particular circuit, the target fault is f stuck at 0, those, the first objective will be to set f to be equal to 1 so that to achieve the target fault, the f should be the line, f should be set to equal to 1. So, if we back trace from this point, we come to the point c. So, it generates the back trace routine. So, it generates another objective which is c to be set to be equal to 0. So, since it is a primary input. So, it can be easily set to 0.

Now, we simulate the case c equal to 0. So, when c equal to 0 is simulated then this D frontier becomes this line g, the D frontier can proceed to g and some gates have been


assigned like as you have proceed, as you are simulating c to 0. So, c, d, e, h, all these lines; they get the value 0 and the line f it gets the value d.

Now, we have to advance this D frontier further that is the second objective that comes up you have to advance D frontier further. So, if you want to progress this D frontier from f to g, I have to set the corresponding objective that comes up is setting a equal to 1.

Now, we back trace from the object. So, a equal to 1, this is nothing to be back traced, we just simulate for a equal to 0. So, simulate a equal to 1, it simulate for a equal to 0 and this will make it this line to for progress sorry, this should be a equal to 1, simulate for a equal to 1 and then that will propagate this f to g and that that g to z. So, that way fault will be get fault will get detected at z.

(Refer Slide Time: 02:29)

Another PODEM Example



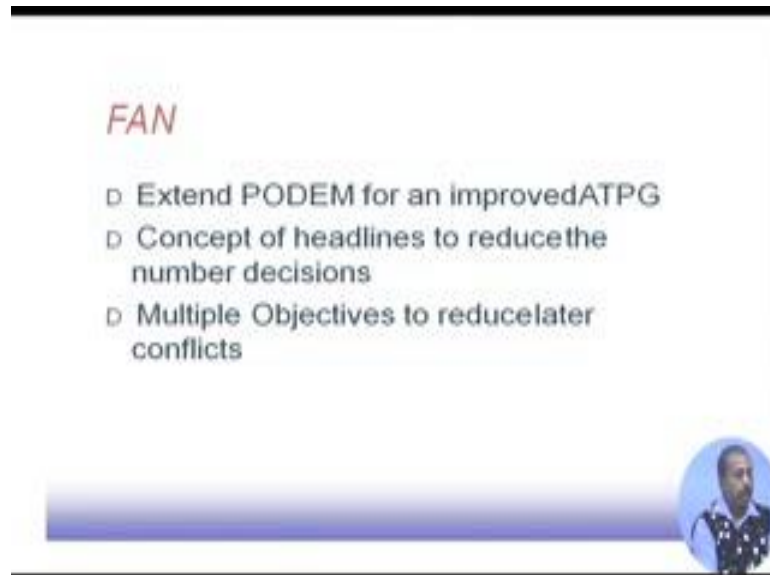
Target fault: b/0

- o 1st Objective: excite fault: b=1
- o Backtrace from objective: a=0
- o Simulate(a=0): b=D, c=0, d=0: empty D-frontier. Must backtrack
- o Change decision to a=1
- o Simulate(a=1): b=0, c=1, d=1, D-frontier still empty
- o Backtrack, no more decisions. Fault untestable.

Another example that we can look into is that we target that b stuck at 0. So, at to a to excite the fault we have to have b equal to 1 that is the first objective. So, if we back trace, we get a equal to 0 to be the objective. Now you simulate for the event a equal to 0, now we see that b becomes equal to d because that is the point to be fault has to be traced and then since a equal to 0, c becomes equal to 0 and d becomes equal to 0. So, D frontier becomes empty. So, you have you have to backtrack, we backtrack now we have tried with a equal to 0 decision we backtrack and try to consider a equal to 1 decision, now if you simulate for a equal to 1 then also b becomes equal to 0, c equal to 1, d equal to 1 and the D frontier still empty D frontier vanishes. In fact so that see that there is no

more backtracking that can be done. So, it is a this fault become untestable. So, that is the operation of the PODEM algorithm.

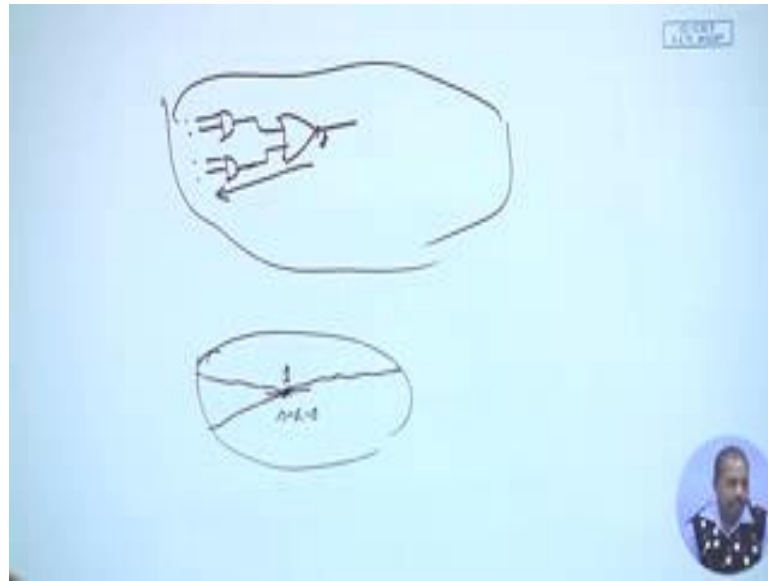
(Refer Slide Time: 03:29)



And improvement of PODEM algorithm is the fan algorithm. So, what it does is that it extends the PODEM algorithm. So, there is a concept of headline. So, headline is that whenever we come to the point where the; we come to the fanout free cone. So, that is called the head line.

That is 1 advancement we do because if we have got a prior combinational cone; that means, all those the output can easily be satisfied like if I have got say a cone like this a cone like this.

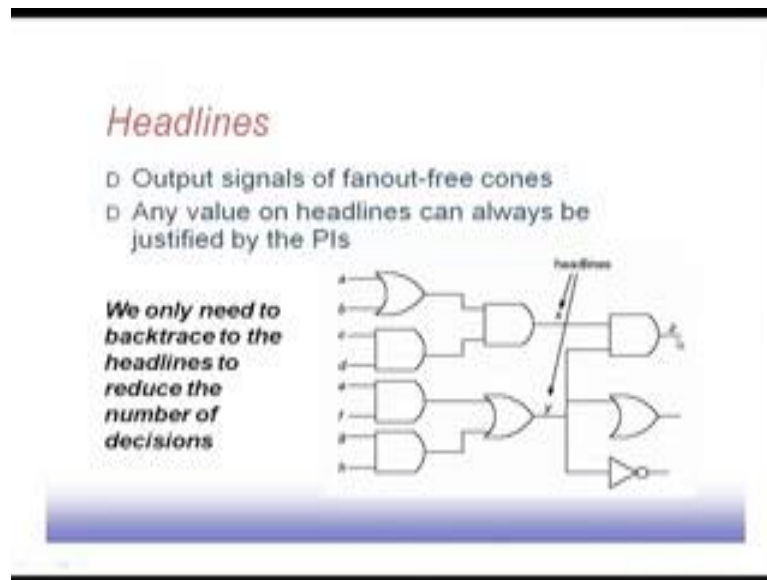
(Refer Slide Time: 04:06)



Then to get a value at this point since it is a combinational logic. So, it is very much possible that I will be able to set this primary inputs to some values for that either 0 or 1 whatever we are looking for can be set at this point so; that means, if we are careful if you have reached this particular point in this entire circuit when we are back tracing. So, if we reach in this particular point. So, there is no necessity to trace back and we are try to justify at the primary input knowing fully well that this part can easily be done and there exist a solution for this part.

That is, this once we reach this headline we do not need to backtrack further another thing is that multiple objectives maybe for multiple objectives will be pursued simultaneously because what happens is that if we pursue with one objective. So, it sets some of the primary inputs some to some value later on we have to pursue some other objective and that may give rise to conflict. So, if we can pursue more than one objective simultaneously then possibilities of conflicts are less. So, these 2 are the extensions in the fan algorithm compared to PODEM.

(Refer Slide Time: 05:15)




As I was telling that headlines output signals of fanout free cones like in this particular circuit. So, this point x is a headline point y is also a headline because they are the output of this particular fanout cones. So, x fanout cone is consisting of these nodes a, b, c and d primary inputs and y's fanout fan in cone consist of e, f, g and h. So, this all the fanout free cone, this is a fanout free cone. So, there we have got the headline any value on headlines can always be justified by the primary input since it is a combinational logic. So, it must be justifiable, otherwise this is only the reverse or the non justification can occur only if the circuit is not design properly, but assuming that we when we are going to the testing phase the circuit has been circuit is well design. So, naturally we can always find some primary input combination which will make the headline point to be true or false as per our requirement.


If you are doing some fault test pattern generation then we can for example, if we are trying to generate some test pattern for z. So, once you come to the point x and y, we know that rest of the thing can be done easily. So, we can reduce the number of back traces to the headlines only we do not need to go further.

(Refer Slide Time: 06:42)

Multiple Objectives

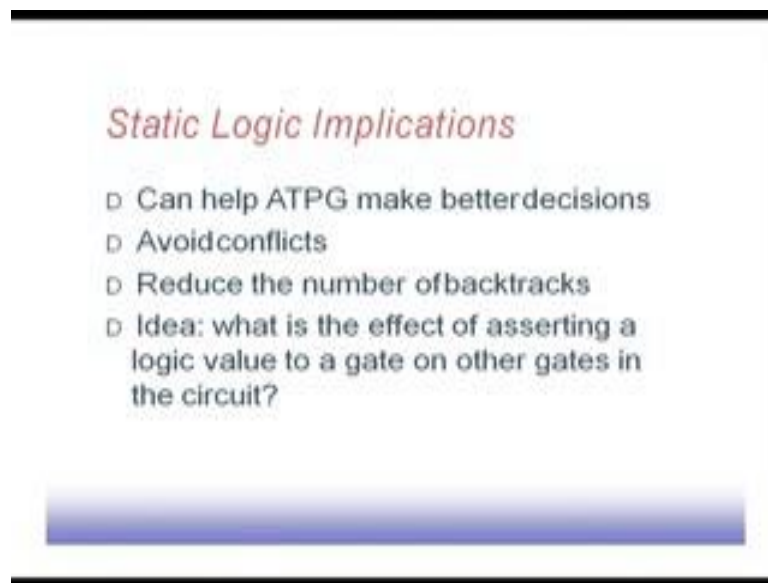


- Objectives: $\{k=0, m=1\}$
- Backtrace from $k=0$ may favor $b=0$, but simulate($b=0$) would violate the second objective $m=1$!
- Makes backtrace more intelligent to avoid future conflicts



Multiple objectives like in this particular case suppose we have got 2 objectives, k is to be set equal to 0 and m to be set equal to 1. So, these are the 2 objective. So, if we proceed with 1 objective at a time for example, if we try with k equal to 0, this objective has to be satisfied. So, if we come, maybe we will choose b to be equal to 0 though there are other options by going through the inputs of this or gate and all that, but suppose the algorithm picks up the b input first and it justifies k equal to 0 by setting b equal to 0, but when is simulate b equal to 0 we see that this m becomes equal to 0. So, there they give rise to a conflict. So, it conflicts violates the second objective m equal to 1. So, we have to make the back trace algorithm more intelligent. So, that this future conflicts can be avoided.

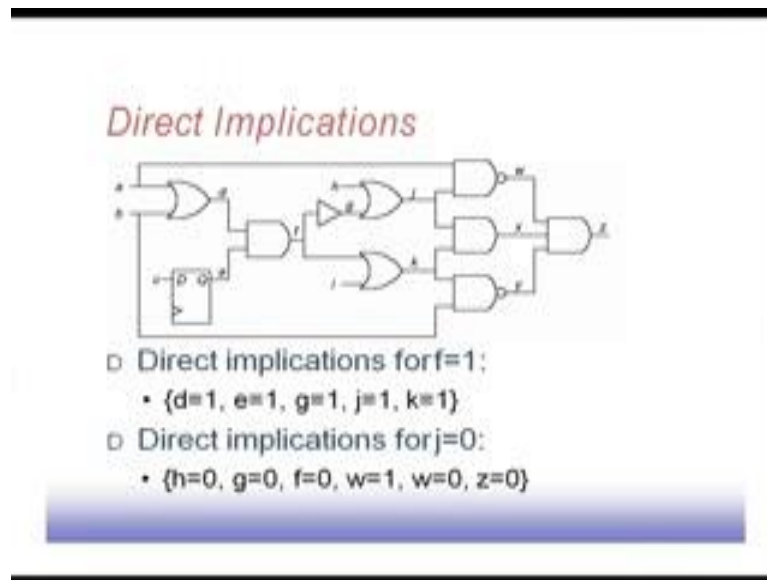
(Refer Slide Time: 07:39)



This fan algorithm is actually doing that. So, it is a having; it is a progressing with multiple objective simultaneously. So, that it can avoid those future conflicts to a great extent. So, the detail of fan is same as the other algorithm the basic cracks of the algorithm we have discussed. Now apart from these algorithms say test generation algorithms there are certain procedures which helps in the test generation process there are certain techniques that have been introduced into this a test generation process. So, that this ATPG algorithms they can be made faster.

The first of the first one of them in that category is the static logic implication. So, it can help ATPG make better decisions because it will have the ATPG algorithm will be fad with some information. So, that it can take quick decision on the on the selecting the inputs to be justified, it avoid conflicts reduced number of backtracks. So, these are the 3 outcome, if I can do some sort of logic implications calculated. So, we will see what is the logic implication? So, the idea is that what is the effect of asserting a logic value to a gate on other gates in the circuit? So, we randomly pick up 1 gate and try to see that if I want to set this particular gate point to be equal to 1 then what is the implication on other gates that are there in the circuit? So, other gates in the circuit means some maybe if it is if this output is feeding an OR gate then irrespective of other inputs to the OR gate. So, OR gate output will become equal to 1.

(Refer Slide Time: 09:34)

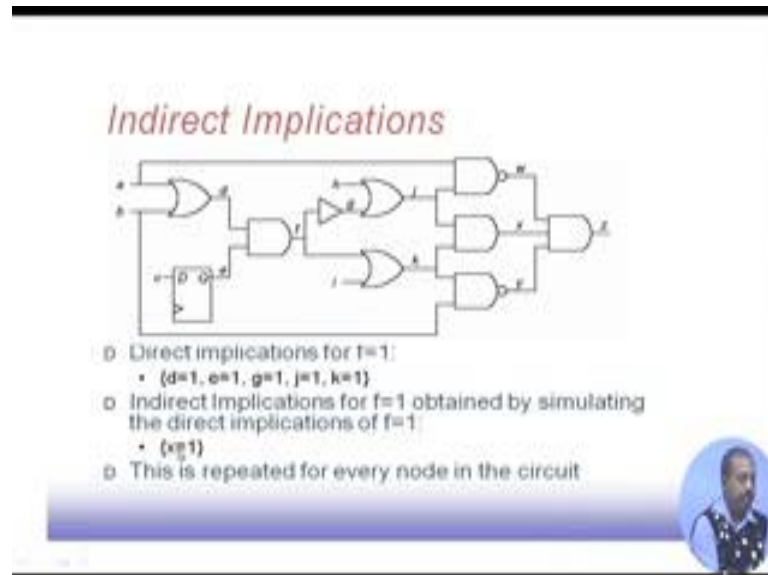


That way we can figure out the certain values for the logic gate outputs whenever a particular gate input or some other gate output is set to be some proper value. So, first we will consider the direct implication. So, direct implication, this is an example circuit consisting of many inputs flip flops, etcetera, now you see that if we are, if we force this line f to be equal to 1 then what happens if we force this f line, f to be equal to 1 then we immediately get d must be 1, e must be 1, g must be 1, j must be 1 and k must be 1. So, if an ATP, this competition is done before and that is if I said f to be equal to 1, these are the things that are going to happen, these are the different line status that are going to happen. So, how does it help? So, if in my ATPG process, if it is some at some point of time we need to set f to be equal to 1, we immediately know some of the other signal lines what are their value? So, whether it gives rise to conflict or not? So, that can be checked very easily. So, we do not need to do a simulation for f equal to 1 or both in the forward direction and backward direction.

They are these are called the direct implication because by doing 1 step we can get the implication of it direct, similarly if I say j equal to 0 then its direct implications are h equal to 0, g equal to 0, f equal to 0, w equal to 1, w then w should be equal to 1, sorry, this should be x not w , x is equal to 0 and z equal to 0. So, that will be the implication. So, direct implications for given any signal, if we give it some value then these are the various are the signal values that will occur. So, you can come to a direct implication for

all those signals. So, you do not need to calculate them again at the time of generating the test pattern set.

(Refer Slide Time: 11:25)



Now, there are some indirect implications as well. So, direct implication. So, it is one step. So, basically if we are coming back to the previous example like when we are computing direct implication for f equal to 1, we find that these are the points, now indirect implications of f equal to 1 or obtained by simulating the direct implication of f equal to 1.

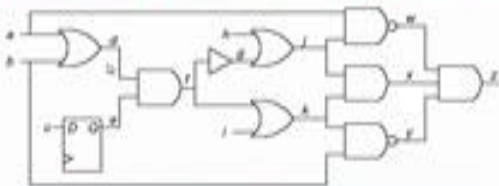
So, if you simulate these cases d equal to 1, if e equal to 1, g equal to 1, j equal to 1 and k equal to 1 then what will happen? I will get x equal to 1. So, x equal to 1 is the indirect implication. So, which is obtained by when we are simulating this direct implication because j and g both are j and k both are equal to 1. So, x will be equal to 1. So, this is an indirect implication of f equal to 1. So, indirect implication computation is slightly more complex because now we have to take all the direct implications and advanced those direct implications 1 step further. So, that is the indirect implication.

Indirect implication; this direct implication and indirect implication, if we repeat this for all nodes in the circuits, in this case, we have shown only for the line f equal to 1. So, if we repeat this process for all nodes like d, e, j, k, w, x, y and z ; if you repeat it for all the cases then for every node we have got idea about what will happen if I said that particular node value to 1 or if I said that particular node value to 0.

That way it becomes easier in the test generation process.

(Refer Slide Time: 13:04)

Extended Backward Implications

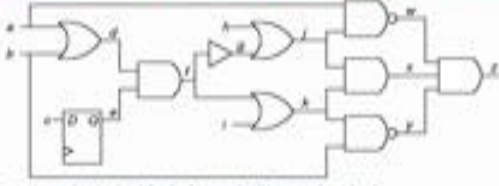


- Direct and indirect implications for $f=1$:
 - $\{d=1, e=1, g=1, j=1, k=1, x=1\}$
- Ext. Back. Implications obtained by enumerating cases for unjustified gates
 - Unjustified gates: $\{d=1\}$

Then there is extended backward implication. So, what it says is we first compute the direct and indirect implications for f equal to 1. So, that is here d equal to 1, e equal to 1 up to x equal to 1. Now extended backward implications are obtained by enumerating cases of unjustified gates like we said that d will be equal to 1, like for setting f equal to 1, one implication is d will be equal to 1, but its input a and b there, till unjustified. So, we can we can say that this is d equal to 1 is an unjustified case.

(Refer Slide Time: 13:47)

Extended Backward Implications



- In order to justify $d=1$, need either $a=1$ or $b=1$
 - $\text{Simulate}(a=1, \text{impl}(f=1)) = S_a$
 - $\text{Simulate}(b=1, \text{impl}(f=1)) = S_b$
- Intersection of S_a and S_b is the the set of ext. back. implications for $f=1$
 - $f=1 \text{ implies } (z=0)$
- This is repeated for every unjustified gate, as well as for every node in the circuit

We need to backward back trace further. So, in order to justify this d equal to 1, we will need either a equal to 1 or b equal to 1. So, either of them are equal. So, we can say that simulate a equal to 1 with the implication of f equal to 1. So, this is the set S_a and similarly if we simulate b the other possibility was b equal to 1. So, if you simulate b equal to 1 for the implication of f equal to 1, we get the set of gates which is S_b .

Now, if we take intersection of these 2 sets S_a and S_b then we get that it is the set of extend extended backward implications for f equal to 1, we will see that if you simulate, this S_a and S_b , say a equal to 1, then b equal to 1, then this f is already equal to 1, then all these values be a becoming equal to 1 ultimately z becomes equal to 0.

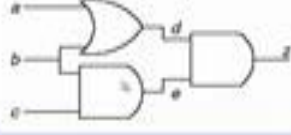
Similarly, if we simulate S_b that is if you simulate this particular case b equal to 1 for implication f equal to 1, we get the set of symbols S_b . So, if you take intersection of S_a and S_b , we will see that z equal to 0, will come into the thing. So, it means that f equal to 1 sets z , z to be equal to 0. So, you see that these, once you do first to do direct implication then indirect time implication then we do an extended backward implications. So, that way the almost entire circuit we will get covered. So, if there are some possibilities that some of the signal values will get some definite value. So, that can be obtained by doing this process.

This process procedure will be repeated for every unjustified gate as well as for every node in the circuit. So, if we do that then we get a fair idea about what is what is going to be happen what is going to happen at various signal nodes various signal nodes in the circuit when we are putting a particular line to a particular value that is this whole thing as happened when we are a trying to push f equal to 1. So, what is the implication of f equal to 1? By doing this process we can figure out.

(Refer Slide Time: 16:09)

Dynamic Logic Implications

- Similar to Static Logic Implications, but has some signals already assigned values
- Suppose $c=1$ has already been assigned
 - Then to obtain $z=0$, b must be 0
 - This is the intersection of having either $d=0$ or $e=0$ in the presence of $c=1$



```
graph LR; a --> OR1(( )); b --> OR1; OR1 -- d --> AND2(( )); c --> AND1(( )); b --> AND1; AND1 -- e --> AND2; AND2 -- z --> out((z))
```

Another one another implication is known as dynamic logic implication. So, it is similar to static logic implication, but has some signals already assigned values. So, in case of static logic implication, we were assuming that only the one particular implication like f equal to 1, that is known and rest of the thing, we were trying to drive, but in this case, we will assume that some signals are also have some value like say suppose c equal to 1 has already been assigned. So, c is already equal to 1 then if you want to get z equal to 0, b must be equal to 0.

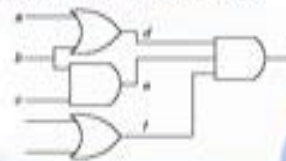
This is the intersection of having either d equal to 0 or e equal to 0, in the presence of c equal to 1. So, in the presence of c equal to 1 or in the presence of this c equal to 1 assignment, we are trying to get z equal to 0. So, what is the implication of that? Implication of that says that that b must be equal to 0. So, that way that is a dynamic implication compared to static, static did not assume any other signal value for any other primary inputs, but in case of dynamic logic implication. So, it assumes that some of the input lines are already been assigned. So, in with respect to that it is trying to simulate try to find the implication of some sig signal lines set to some value.

(Refer Slide Time: 17:33)

Another Dynamic Implications

Example

- Suppose $b=D$
- In order to propagate the fault-effect to z , $f = 1$ is a necessary condition [Akers 76, Fujiwara 83]
- To take this further, the intersection of all the necessary assignments for all fault-effects in the D-frontier can be taken [Hamzaoglu99]

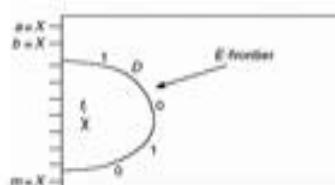


Another example suppose b is equal to d, if you want to propagate this fault effect to z, f must be equal to 1, b equal to b, if this f must be equal to 1 and if you want to take this further, the intersection of all necessary assignments of all fault effects in the D frontier can be taken. So, we can take these D frontier like this is a D frontier then e is a D frontier. So, if you are trying to propagate them then by setting, we if you take intersection of them then this you can find the implication dynamic implication.

(Refer Slide Time: 18:14)

Evaluation Frontiers

- If two faults have the same E-frontier with at least one fault-effect, then the values on the unassigned PIs can be the same [Giraldi 90]

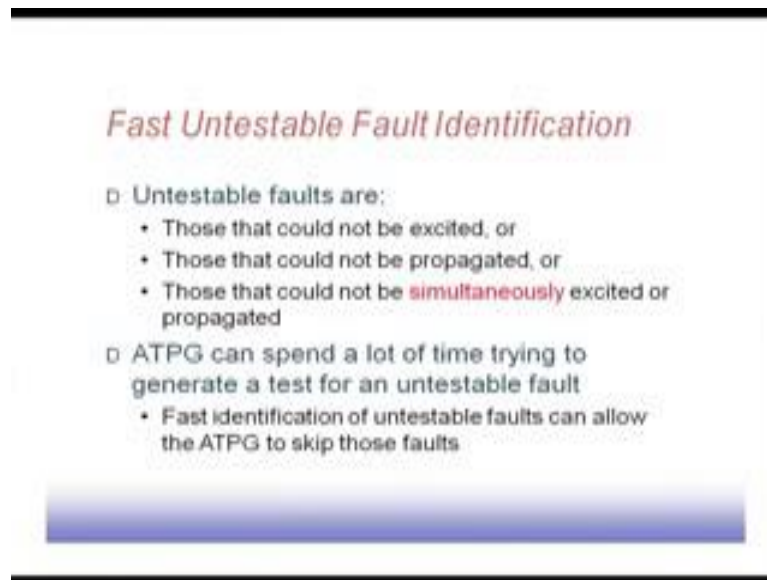


Another way of speeding of this fault test pattern generation process is the evaluation frontiers. So, it says that evaluation frontier is like this. So, for a particular fault F, I suppose at some point of time we find that these are the evolve this is the evaluation frontier that is these lines must be set to these values this point 1, this point d, this point 0, this point 1, this point 0 and for this these are the inputs that have been that needs to be assigned to some value now for. So, this has to be forwarded because this fault effect, these d has to forward to some primary output similarly these 1 0, they have to justified and all that.

Now, if it is found that 2 faults are same E frontier with at least 1 fault effect then the values on the unassigned primary inputs can be the same. So, this is a result from some paper. So, what it says is that if you find there are at there are 2 faults having same E frontier. So, if it is same, if for 2 faults in E frontiers are same; that means, we can assign we do not need to if we have already computed the primary inputs for the first fault we do not need to compute primary inputs for the second fault because for the E frontier. So, same set of assignments can be done.

If we maintain the list of, if you maintain the list of E frontiers for all the fault that we have processed So far, whenever we are trying out with a new fault. So, you and if you see that E frontier is becoming same with some previously computed E frontier then we can directly take the input assignment part from the first test pattern. So, that way the process of this test generation may be made faster.

(Refer Slide Time: 20:04)



Another very important issue in this ATPG algorithm is to identify what which faults are untestable otherwise what will happen is if a fault is untestable and ATPG algorithm will go on running this backtrack routine and all that it will try to explode the search space in more and more detail, but since the fault is untestable. So, it will never be able to find the particular pattern any particular pattern for that. So, this is the wastage on the part of the this test generation faults.

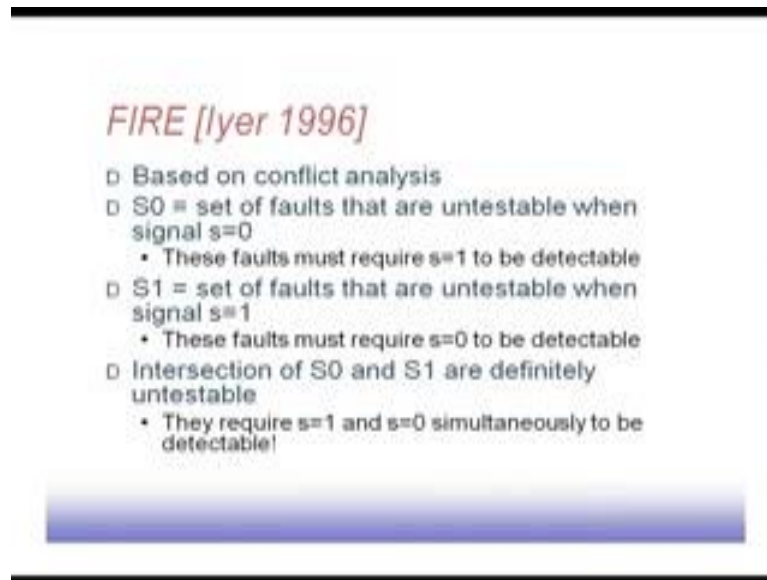
We need to identify the untestable fault. So, if we can do this untestable fault identification fast then we it is going to help us in the test generation process. In fact, any ATPG algorithm you run. So, if the circuit has got some untestable fault it reports these are the untestable faults in the circuits. So, we definitely does some analysis to figure out what are the untestable faults in the circuit.

Untestable faults are those faults that could not be excited. So, this is one possibility. So, the fault could not be excited or the fault could not be propagated. So, basically what happens is if this is the circuit and there is a fault at this point, now one possibilities that the fault is such that I cannot excite this fault. So, this line stuck at 0, I can never put a 1 value at this point. So, due to the logic that it that it is having, it is not possible to put a 1 at this point. So, that is that way it becomes an untestable faults.

Another possibility is that it is not possible to take this output take this observation to some primary output. So, it is not possible to propagate this fault to some primary output.

So, this is another case where the fault become untestable other possibilities that is it cannot be simultaneously excited or propagated. So, those fault that could not be simultaneously excited or propagated. So, naturally they are also untestable fault.

(Refer Slide Time: 22:37)



FIRE [Iyer 1996]

- Based on conflict analysis
- S_0 = set of faults that are untestable when signal $s=0$
 - These faults must require $s=1$ to be detectable
- S_1 = set of faults that are untestable when signal $s=1$
 - These faults must require $s=0$ to be detectable
- Intersection of S_0 and S_1 are definitely untestable
 - They require $s=1$ and $s=0$ simultaneously to be detectable!

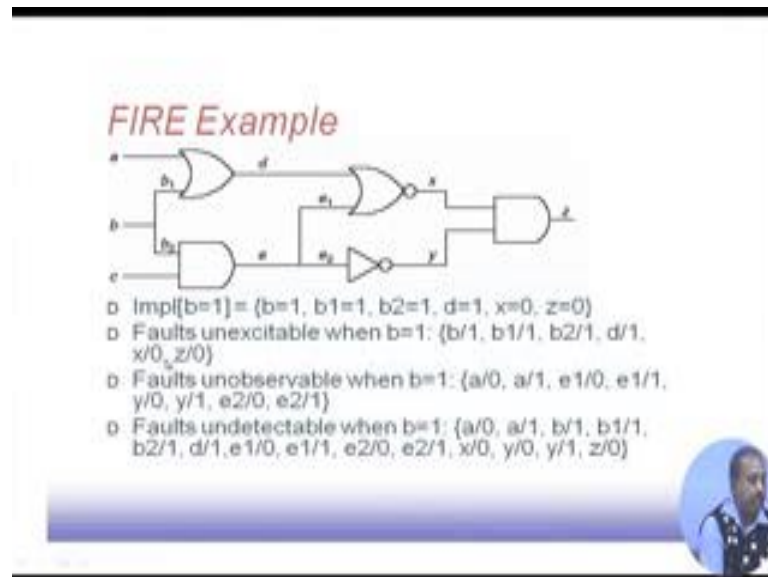
Now, naturally the problem are ATPG will spend lot of time trying to generate a test for an untestable fault. So, if we have got a fast identification of untestable faults. So, it will allow the ATPG to skip those faults. So, it can try it can avoid those faults and try to generate test pattern for the remaining faults.

The work by Iyer 1996; that is the fire is it is a tool that has been reported. So, it is based on conflict analysis. So, what it does let us say, S_0 is the set of faults that are untestable when signal line S is equal to 0 these faults required S equal to 1 to be detectable. So, these, when S equal to 0, I need to detect the fault, what this they will require because we are trying to check for 0, we must we must get a 1 at this line. So, these fault must required S equal to 1 to be detected.

Similarly, S_1 is the set of faults that are untestable when signal line S is equal to 1 and these faults will require that S equal to 0 value must be coming to for the fault to be detected. Now if there are some faults which are common in both of these sets S_0 and S_1 . So, the faults which are not which become untestable with S equal to 0 maybe testable with S equal to 1 and vice versa the set of faults which are untestable under S is equal to 1 maybe testable under S is equal to 0. So, if there is some faults which is

common in both the sets; that means, it is neither testable with S equal to 0 nor testable with S is equal to 1. So, these faults are simultaneously. So, they will require simultaneously S equal to 1 and S is equal to 0. So, they become untestable faults. So, these are this is the basic logic by which we compute the setup untestable faults.

(Refer Slide Time: 24:14)



Like say this one, if we have this line implication of b equal to 1. So, b equal to 1. So, it will set b equal to 1, $b1$ equal to 1, $b2$ equal to 1, d equal to 1, then x equal to 0 and z equal to 0. So, this is the implication of b equal to 1.

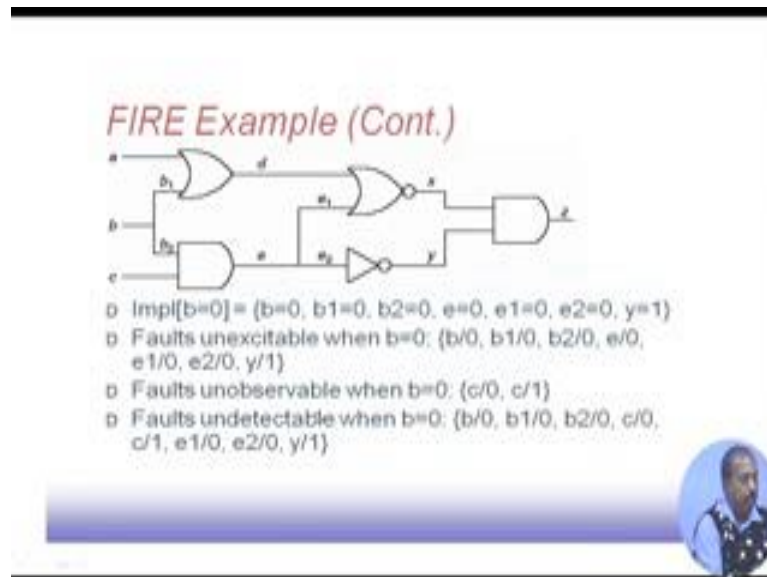
Now, faults which are unexcitable when b equal to 1 you know b equal to 1, the faults we cannot check b stuck at 1 because this will required b line to at 0, similarly we have if we see if we want to check $b1$ stuck at 1 that is also not possible because I need a 0 at this point which is not do not possible similarly $b2$ is stuck at 1 d stuck at 1 x stuck at 0 and z stuck at 0. So, these line this faults cannot be excited by when b is equal to 1.

Now, because b equal to 1, you will always ensure we will always make z equal to 0. So, after computing this implications set after computing this implications set you can say that for b equal to 1, all these all these faults reliance which are on the implication set. So, they become untestable they are unexcitable.

Similarly, we can also find out the faults which are unobservable when b equal to 1. So, that is a equal to 0, a equal to 1, $e1$ equal to 0, $e1$ equal to 1. So, this set is the faults that

are unobservable similarly. So, naturally the faults which are undetectable when b equal to 1 are the union of these 2 sets: faults that are unexcitable and faults that are unobservable. So, that gives us the faults which are undetectable with b equal to 1.

(Refer Slide Time: 26:05)



Now, if we consider the other case that is b equal to 0. So, this is the implication of b equal to 0, b equal to 0, b_1 equal to 0, etcetera and then faults which are unexcitable when b equal to 0 is basically for from this implication set faults that are unobservable when b equal to 0. So, if you propagate through the circuit will find the c equal to 0 and c stuck at 0 and c stuck at 1. So, these are the 2 faults that cannot be observed when b is equal to 0. So, if you take union of these 2 sets. So, you will find out the faults which are undetectable when b equal to 0. So, this is the set.


(Refer Slide Time: 26:50)

FIRE Example (Cont.)

□ Now that the two sets of faults undetectable when $b=0$ and $b=1$ have been computed

□ The intersection of the two sets are those faults the require $b=1$ AND $b=0$ for detection, thus untestable:

- $\{b2/0, c/0, c/1, e/0, e1/0, e2/0, y/1\}$



Now, we have got the set of faults which are undetectable with b equal to 1 and b equal to 0. So, if we take, if we take intersection of these 2 sets. So, we take intersection of these 2 sets,, we get $b2$ stuck at 0 c stuck at 0, c stuck at 1, e stuck at 0, $e1$ stuck at 0, $e2$ stuck at 0 and y stuck at 1. So, these are the faults which are common in both the sets b equal to 1 and b equal to 0. So, these faults are actually untestable faults for these circuits. So, you see for this even for this small circuit there are so many faults which are untestable. So, if you have got a big circuit. So, you can imagine like how many faults maybe untestable. So, if we can detect those untestable faults fast then naturally we will be able to make our ATPG run faster than cases where it will try to generate test for this untestable faults as well.