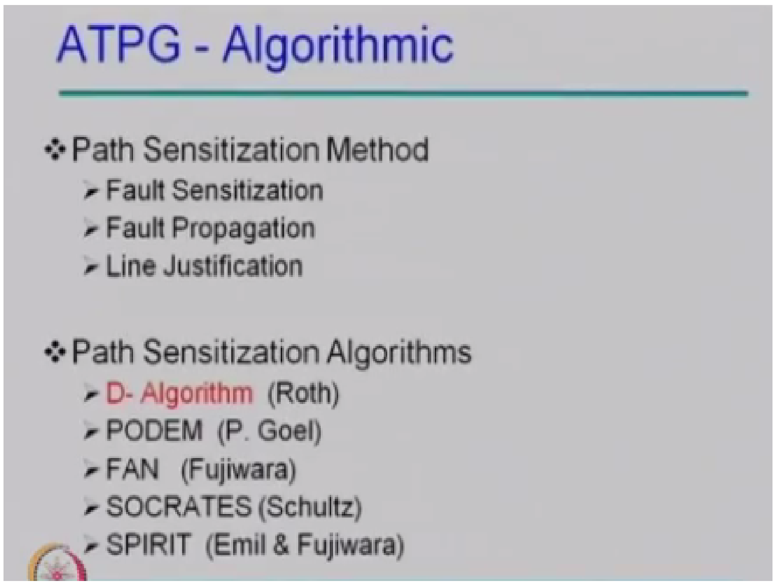
Advanced VLSI Design
Prof. Virendra K. Singh
Department of Electrical Engineering
Indian Institute of Technology – Bombay

Lecture – 36
VLSI Testing: Automatic Test Pattern Generation

Welcome to lecture series on advanced VLSI design. So, continuing with my last lecture on VLSI test, where we discussed about the fault modelling, test generation using algebraic method and we discussed that algebraic method is a complete method. It gives you the test vector if it exists, otherwise it tells you that fault is resistant fault and complexity of both of the process remains same.

(Refer Slide Time: 01:26)



ATPG - Algorithmic

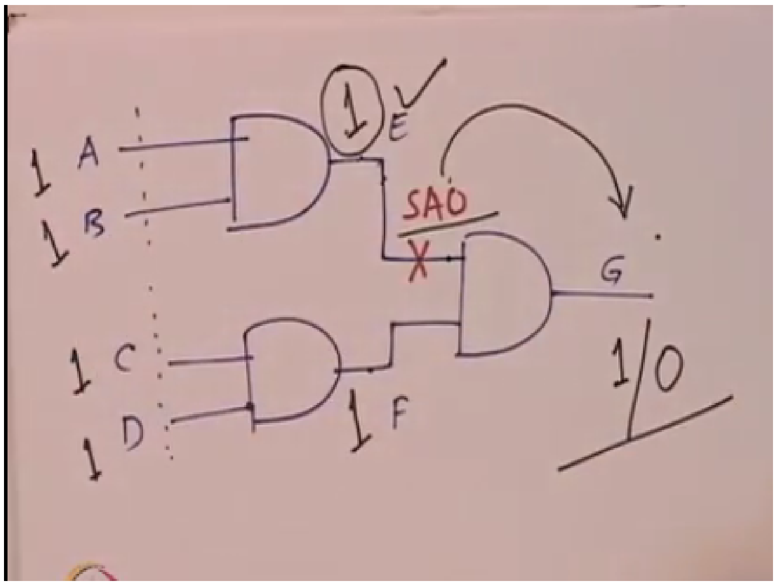
- Path Sensitization Method
  - Fault Sensitization
  - Fault Propagation
  - Line Justification
- Path Sensitization Algorithms
  - D- Algorithm (Roth)
  - PODEM (P. Goel)
  - FAN (Fujiwara)
  - SOCRATES (Schultz)
  - SPIRIT (Emil & Fujiwara)

The problem, that we discussed about algebraic method first, that it is not a scalable, so that means if you have design of multimillion gates, you may not be able to generate test using algebraic method. So, then we have to look for the alternate method, which are scalable and so the algorithm method is an alternate solution. Algorithmic methods are based on path sensitization, which follows 3 steps; fault sensitization or activation, fault propagation and line justification.

There are couple of algorithms given in the literature and some of them I have listed here like D. Algorithm, which was the first complete algorithm given by Roth in 1966 from IBM. Then, there was further improvement on that in 1981 by Goel from IBM and which is called as PODEM. Then, again some new concepts were introduced by Professor Fujiwara from Osaka University and in 1984 and that is known as FAN.

And then, so almost; in all the automatic test, certain generators, they use either Podem or Fan as the base test generation engine. Then, over the years, people started to put some more concepts, which are more or less based on the learning from the circuits and 2 of the examples I have listed here; SOCRATES given by Schultz and SPIRIT given by Emil. So, in this course, I will briefly describe about D. Algorithm and then you can follow Podem and Fan.

(Refer Slide Time: 03:10)



So, as I mention that all these algorithms are based on path sensitization. What does it mean? What is path sensitization? So, let us look at this circuit, which has 4 inputs and one output is pretty simple circuit and now let us say, we have struck at 0 faults at line E. Let so, I have to have find out a input vector that can give me the distinguishable fault t and fault free output at g, right.

So, if I assign 0 at E, in that case it can never give me the distinguishable behaviour, so that means here, I have to assign opposite value to the faulty value at fault site. This is known as fault site. So, now here, I have to assign opposite value and that is known as fault sensitization or fault activation. Now, I need to assign value 1 at some intermediate point here at the fault site, right.

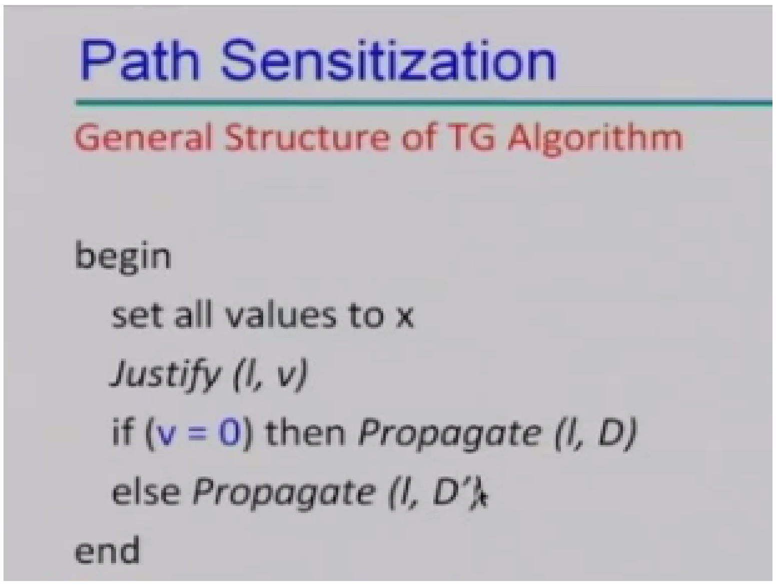
So, now and I do not have any control over here. So, what I have to do is; I have to go back to the primary input and try to find out some input, which can justify value1 here at E, right and we know that here if I have 1 and 1 at A and B, I can get e =1, so this can activate the fault. So, that means here, I; in order to activate, I need to have opposite value here and in order to do that, I have to go back all the way to the primary input and assign these value at the primary input and this is known as line justification.

Because, you are adjustifying that at the primary input line. Now, the next thing is; I cannot observe the fault effect right at this point. Because, I do not have any observability from this point. So, now what I have to do? I have to transfer this fault effect to the primary outputs. So now in order to do that, what I need to do is; I have to assign the known controlling value to the other input of this gate, right and known controlling value of this gate is 1.

So, that is the requirement for the fault propagation. Now, here again I cannot justify, I cannot assign any value at the intermediate point so, in order to do that, I have to go all the way to the primary input and I have to assign 1 1 at C and D. Now, here if I assign 1 1 1 1 at primary input in that case here, this will give me 1 as, if circuit is fault, free and it will give me 0, if circuit has strict, add 0 fault at fault site E.

So, now here it has 3 components; one is the fault activation and another is the fault propagation to the primary output and then I have to justify some of the unjustified intermediate values and for that, I have to go all the way to the primary input. So that means primary input means here, if you level your circuit, you go all the way to the input line that is level 0.

(Refer Slide Time: 06:41)



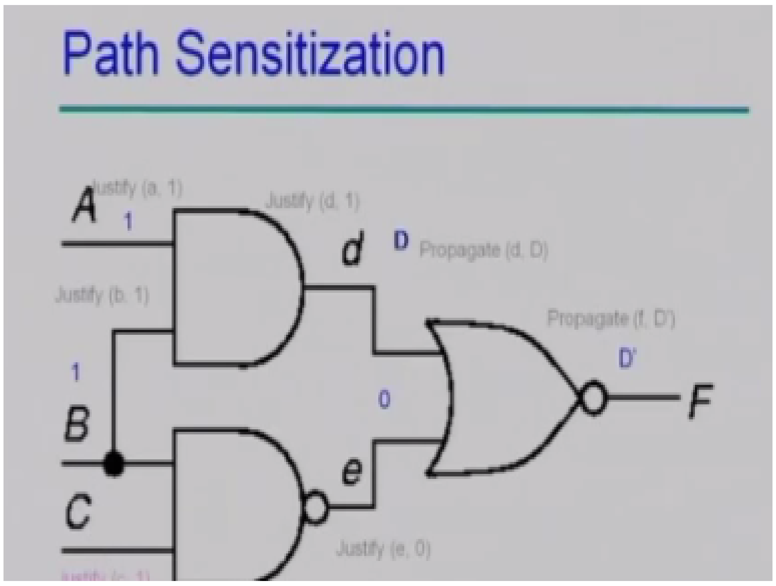
Path Sensitization

General Structure of TG Algorithm

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

So, and that is referred as line justification. So, these are the 3 steps, all the ATPG algorithms have to follow. Fault sensitization, fault propagation and line justification. Now, here so then basic structure of the; any of the ATPG tool, would be like this, you have to set all the values at primary input as x, initially. Then, you justify some value say, v at the fault site, which is, l and then, so this will activate the fault and then after that you have to propagate the fault effect to the primary; one of the primary output.

(Refer Slide Time: 07:31)

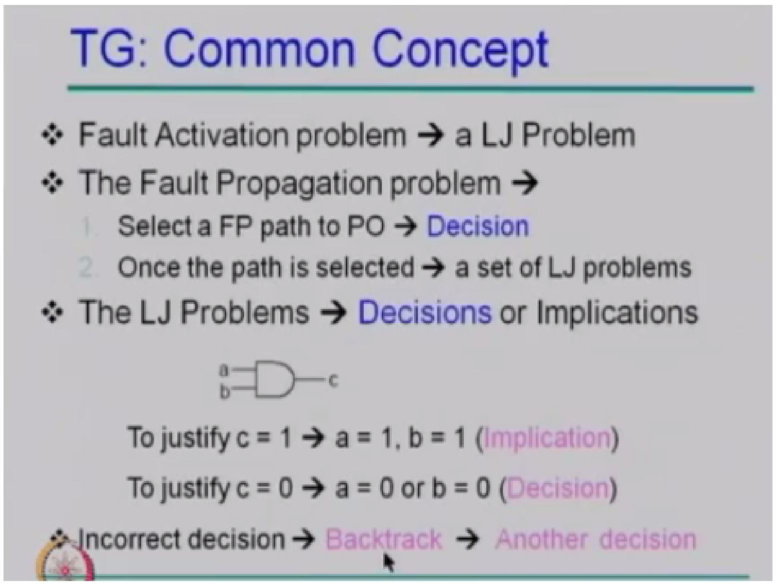


So now here, if your v0 in that case, you propagate means, l, D; these are symbols. I will discuss in minute what D and D bar mean? So you propagate this one, otherwise if v is 1 in that case, you propagate D bar and so this is the basic skeleton. For example, here, in this circuit, here say your d is struck at 0, so in that case here, I assign a symbol d here and now here in order to do that, what I need? I need to have opposite value, assign to d.

So, that means, I have to create a function justify d, d line has one, so in order to do that, I cannot do anything here, so I have to go all the way to the primary inputs and that means here I have to justify A as 1 and I have to justify B as 1 and since these 2 are the primary input, so always you can justify these values. So, your fault is activated. So, now after that you have to propagate this fault to the one of the primary outputs, here we have only one primary output.

So, now you propagate d to F. Now, here because this is inverting gate, so d would be propagated as d bar. So, now if d is propagated as d bar at the output, so in that and in order to do that, you need to have non-controlling value at e, which is 0, right. So, in order to assign 0 here, what you need? So you have to justify value 0 at e and in order to do that, you need to justify B as 1 and C as 1. B is already 1, so now when you justify C as 1 in that case here, you are done.

(Refer Slide Time: 09:03)



TG: Common Concept

- Fault Activation problem → a LJ Problem
- The Fault Propagation problem →
  1. Select a FP path to PO → Decision
  2. Once the path is selected → a set of LJ problems
- The LJ Problems → Decisions or Implications

To justify c = 1 → a = 1, b = 1 (Implication)

To justify c = 0 → a = 0 or b = 0 (Decision)

- Incorrect decision → Backtrack → Another decision

So, now here, 1 1 1 is your test vector. Now, if you look at this process in that case here, as I said that there are 3 steps; one is fault activation, then fault propagation and line justification. Look at the complexity of this fault activation, line justification and fault propagation. So, first is fault activation or fault sensitization. So, now in order to; means what do we need to do in fault activation or sensitization, I have to assign the opposite value at the fault site, right.

So, now here you need to justify one, so and how you can justify? You can justify that by assignment at primary input, right. So, that means here, this problem converts into line justification problem. So, your fault activation problem is like a line justification problem. If you look at the fault propagation, in this example, which we have seen, we have only one primary output; it may be possible that there would be a Fan out point here.

And then, these can propagate; this may potentially be propagated to multiple outputs. In that case, you have to select one, right. So, that means here, you may need to; you need to select a fault propagation path to one of the primary outputs at least and that is the decision making process. So, now here this is a decision making process, it is like you want to go from the institute to the hostel.

And then, if you are not aware about all the ways you start to follow the sign and then you reach to some cross point and then add some cross point, if sign is not there, you have to make a decision which way you should go and then you keep on moving until the next decision point and you make an another decision and keep on going. Hoping, that you would reach to the destination or hostel.

If your decision is wrong, you end up in a forest then you how to come back to the previous decision point and then take the different decision, right. So, means the fault propagation is a decision making process, always you how to make a decision at the Fan out point. Once you make a decision after that say, once you make the decision, here in this case there is no decision, there is only one path.

So, you propagate it and then you need to assign known controlling value to all the, of inputs of that gate, right. So, now here and then you cannot assign any value at the intermediate point, so you have to go all the way again to the primary input and then this problem converts into line justification problem. So, now here faults propagation problem is either decision making, so this is decision making as well as the line justification problem.

If you look at the line justification problem, again in this small example here, when you want to justify say 0 here, you need to have both 1 1, so there is no choice. This may possible that you may need to assign 1 here, in that case here; you can have anyone as 0 that satisfies your requirement. So, now your line justification problem is either decision making process or implication. What is implication?

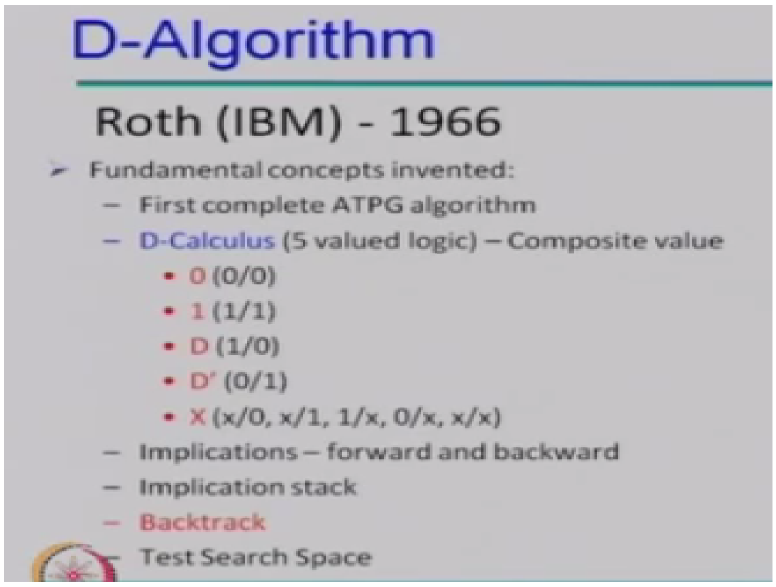
Say, if in this gate here, if you want to justify output C as 1, in that case always you need to have A and B as 1, there is no choice. So, that is implication so, if you want 1 here, in that case here, A and B are 1, this is typically referred as backward implication. If you have, say A as 0 and B can be anything in that case, output C is always 0, that is forward implication, so this is the implication. But, if you want to assign C as 0, in that case you have choice.

You can have either A as 0 or you can have B as 0. So, that is decision making process. Now say, you make a decision that A is 0 and then here again A may be intermediate point and then you may want to go back to the primary input and then there may be a conflict in between and your decision may be wrong and then you may need to come back again and take assign the alternate value, so alternate assignment could be like here B as 0 and C can be any value x, right.

So, this is decision making process and so now in the fault propagation, you are making a decision in line justification, you are making a decision, so there are couple of decisions you are making and your decision may be right, may not be right. In other word, your decision may be wrong. If your decision is wrong in that case here, you have to back track like I told you, when you are exploring in your path, if you end up say, in forest you may need to come back and then where you will come back?

You will come back to the previous decision making point and then you how to explore the alternate path. So, that is the back tracking. So, now if you back track multiple times it will take; more time to reach to the destination, so in case of incorrect decision, you back track and then make another decision and then if you reach to the primary output here, you say you are done and then that gives you the test vector.

(Refer Slide Time: 15:05)


D-Algorithm

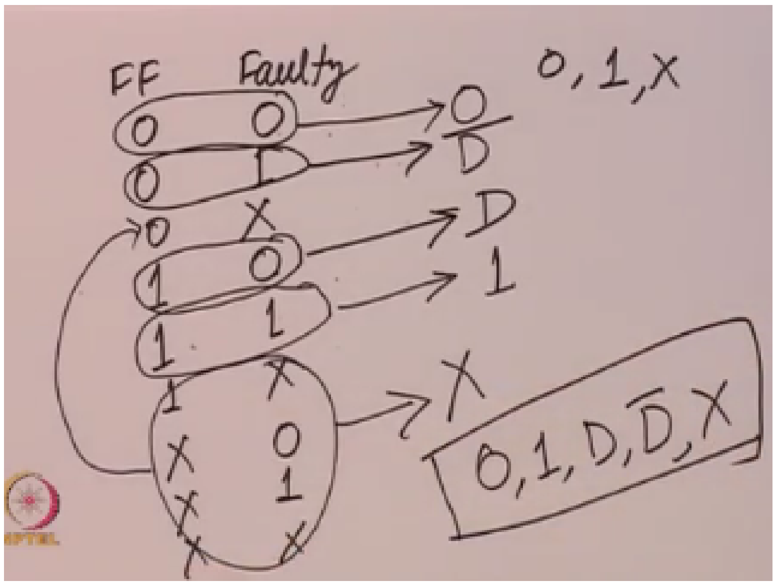
Roth (IBM) - 1966

- Fundamental concepts invented:
  - First complete ATPG algorithm
  - D-Calculus (5 valued logic) – Composite value
    - 0 (0/0)
    - 1 (1/1)
    - D (1/0)
    - D' (0/1)
    - X (x/0, x/1, 1/x, 0/x, x/x)
  - Implications – forward and backward
  - Implication stack
  - Backtrack
  - Test Search Space

So, as I mentioned you earlier that D. Algorithm was the first complete algorithm given by Roth in from IBM in 1966, it was the complete algorithm; complete algorithm means, it gives you the test vector if accessed, otherwise it tells you that particular fault is resistant fault. So, that means here having that fault in the circuit would not result into malfunctioning of the circuit.

This is based on 5 valued logic or 5 valued algebra that 5 value, because here, what you want is; you want to have distinguishable faulty and fault free behaviour. So, that means here, when you propagate your signal, you how to propagate both of the values faulty value as well as fault free value. So, that means here, your logic must have composite value that has fault free value and faulty value.

(Refer Slide Time: 15:56)



And so, now here so we can have 3 kind of values; 0, 1 and x and how many combinations we may have? Your fault free value and faulty value, if you look at fault free value is 0, faulty value may be 0, fault free value is 0, faulty may be 1, this is 0 and this is x. Then, your fault free value may be 1, faulty 0, this is 1, faulty is 1 and 1 and x and your fault free may be x and faulty is 0, x, faulty is 1, x and both are x, x.

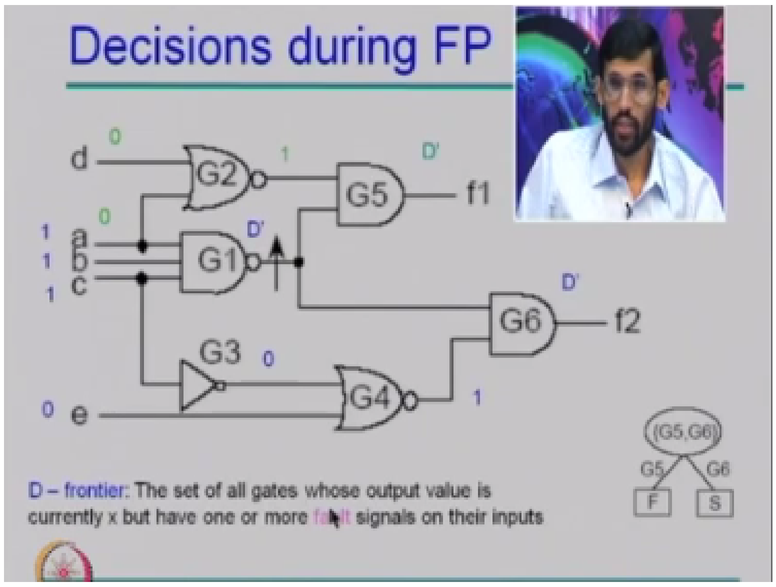
Now, if you look at this, in this case, both values are indistinguishable always it stays to 0, I assign one symbol say that symbol is 0. In this case, both values are one, so these are again in distinguishable and I assign other symbols say that symbol is 1. This condition when fault free value is 1 and faulty value is 0, I assign another symbol and say that symbol is D, what is represent? This represent; you are struck at 0 fault at fault site, right.

Because, in case of fault it gives you 0 values, in case of fault free signal, it is 1, because that you want to assign. This value and fault free is 1 and faulty is 0, this is another, just opposite to this and hence here we assign another symbol D bar and all other values, where in one value is x, so that means here, we cannot determine what would be that value, that may be 1 so in that case, it may be in distinguishable or it may be distinguishable but we cannot reason about that, so we clubbed all these values who has x, es in one symbol, that is x.

So, now here we have 5 values; 0, 1, D, D bar, and x, right. So, now here, this method works on 5 valued algebra, where in the 5 symbols that we use are 0, 1, D, D bar and x and it heavily uses implications in the forward direction as well as in the backward direction because keep in mind, that when you are making a decision there are chances that you make incorrect decision and you need to back track.

And so, now here when you are making back track, you are losing time and you need to search more space. In order to back track here, what we need to do is we have to maintain a stack all the decision, so that here when you back track, you will come back to the previous decision point and then you will freshly start to explore the search space right from that previous decision making point.

(Refer Slide Time: 19:23)


Decisions during FP

D – frontier: The set of all gates whose output value is currently x but have one or more fault signals on their inputs

So, now here let us look at little bit more in this, so as I discussed here, your; it follows all those 3 steps; fault sensitization, fault propagation and line justification. So, when you are propagating the fault effect, you are making a decision and if your decision is wrong, you have to back track, right. So, now here how do you do? Let us assume that you have this circuit and there is a struck at one fault present at this point.

The point, where we have arrow, so now here what I want here? I want to assign value 1, 0 here and in case of fault here this produces 1; always 1 here, right. In case of, so, whether I assign here 0, I assign 1 here always it will produce 1, so now here that I can capture by a symbol D bar. In order to have D bar here after this faulty site here, I need to have 0, in order to have 0, what I need? I need A must be 1, B must be 1, C must be 1 and this is line justification problem.

So, once I have this, i will get it. Once, you have 1 here, in that case here, this will imply 0 here, right and so now here I have choice that so from this point; this is by implication. From this point, now there are 2 Fan out branches; one goes to get G5 and other goes to get G6. So, now I have to make a decision here, so let us say I want to; first I make a decision in favour of G5, so that means here I want to propagate fault effect to f1 through G5.

If I propagate this and this is non-inverting gate, so now here D bar should propagate here as D bar. In order to have propagate these as D bar here, what I want; this should have non-controlling value; non-controlling value is here1 and now if it is 1 in that case here, what I want? Both of the input must be 0, in order to have 1 here, right and because now here you have already assign 1 here, so there is a conflict, you cannot assign 2 values at single input.

So, now here, and that arises because of reconvergent fan out. So, now here, your fault propagation failed. If it is failed, then you have to come back to the previous decision making point and make an alternate decision. So, your previous decision making point is here, right. Now you have to; you already explore propagation through G5 now, you left with only one choice through G6 or try to explore through G6.

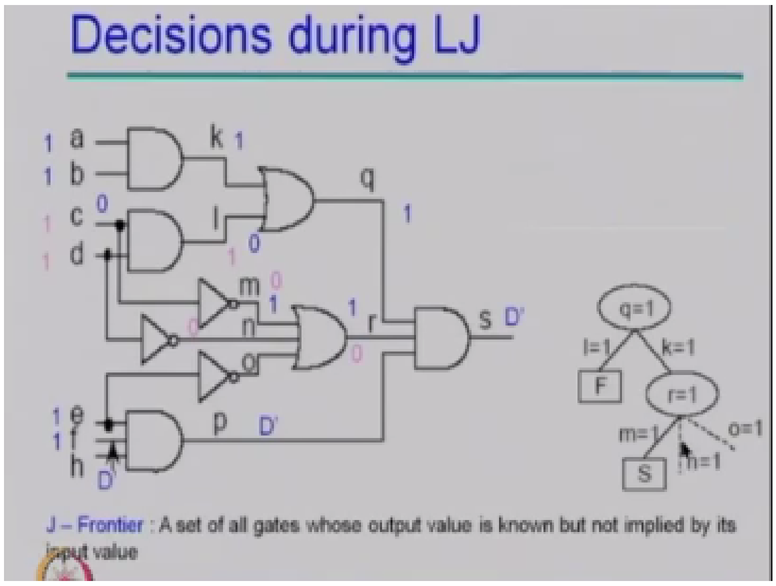
If you explore through G6 in that case here, this D bar will propagate as D bar here, provided that you assign value 1 at the half input of G6. Now, in order to assign 1 here, what you need? You need this value is already 0, so if you assign 0 here, another input of E, in that case here, you will get 1 as output of get G5 and without any conflict, hence you are done. So, now your test vector would be A is 1, B is 1, C is 1, D can be anything x.

Because, it does not depend on that and E must be 0. So, now here 1 1 1 x0 is the test vector, which can detect struck at one fault at the output of gate G1. So, this explain the decision making process, so now here, we had to make a decision between gate G5 and G6. We explored first G5, we failed, we back track and then we explored another possibility, that is through G6 and we succeed and it gave us the test vector.

In order to, so that means here at every point, you have to maintain or stack, where in at least there must be D1 D0 D bar must have accessed in the circuit. So, for that here we define a set that is called as D frontier. D frontier is the set of all the gates whose output value is currently x, so that means output is non-determined but at least one of the input is faulty value that means D or D bar.

So, that means those are the potential gates through which you can transfer or propagate the faulty effect to one of the primary outputs, okay. So, always at every means; any point and time, there must be at least one gate in the D frontier. Otherwise, you block the fault effect propagation and hence your algorithm cannot progress. So, and that tells you that fault is non sensitizable or it is a resistant fault.

(Refer Slide Time: 24:51)


Decisions during LJ

J – Frontier : A set of all gates whose output value is known but not implied by its input value

So, this gives you some understanding about how we can make a decision for the fault propagation? Now another decision you need to make during the line justification, let us see how we make decision during the line justification? So, now here, say in this circuit, you need to have test for struck at one fault at line h. So, in order to have that, I need to assign a symbol D bar. So, in order to excide the one fault, here h must be 0. If h is 0, then in order to; then you have to propagate it.

And in order to propagate, you need to have f as 1 and e as 1, so if you have both these in that case here, this will propagate as D bar. If e is one in that case, you are simply by implication o would be 0, right. Now, this is not the primary output, so you have to propagate to one of the primary output and there is one possibility you have to propagate through this AND gate to output S. In order to propagate that here, what you need?

You need to justify non-controlling value at q and r and non-controlling value for this gate is one so hence you have to q = 1 and r =1. Let us take means now you have to sequential it. So, let us take first, you want to assign q =1, what do you need to do? Because if q is = 1, now here how you; this is a decision making point because this gate can give q =1, if either k is 1 or l is one. Now let us explore, make a decision in favour of l, so assign l =1.

If l =1 in that case here again you reach to the implication that l can be 1, only when c and d both are 1, okay. Now, let us see what happens, if c and d both are one, in that case here this c1 implies m =1 and d implies n =; sorry; c1 implies m=0, d implies d = 1implies n = 0, e =1 implies m = 0, all 3 inputs of these OR gate are 0, hence output would be 0 and then this will block the propagation of fault effect.

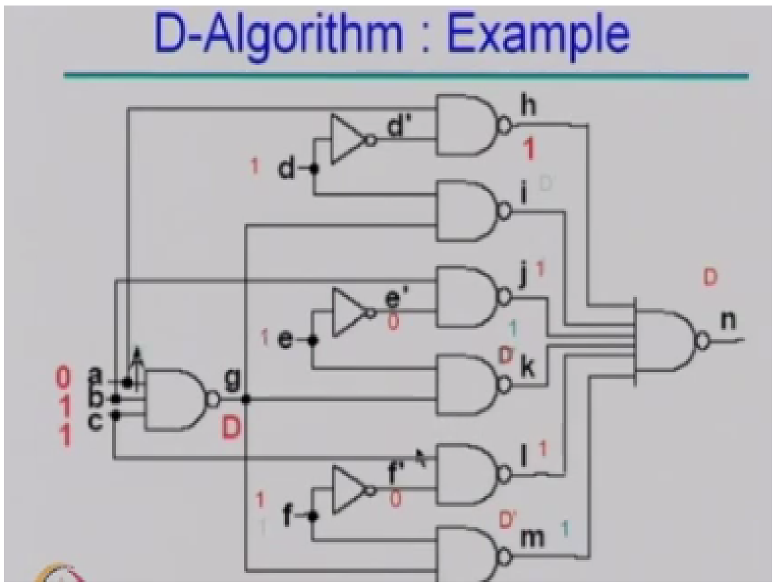
So, now it results into conflict because here you wanted to have one here and then you happen to get 0 and so now you have to go back to the previous decision point. Now, where I made decision? Was there decision here? There was no decision. But, when I was assigning one at the input of this OR gate, I made a decision in favour of l, that was wrong, so I have to make, I have to correct the decision or make an alternative choice and that is k = 1.

If I say k =1, then by implication your, a should be 1 and b should be 1 and then there is no conflict. So, now here k = 1, a = 1, and b =1, give k = 1, that will give you q =1, is still here r = 1 is unjustified value. So, in order to do that here what you want? At least one of the inputs of this gate must be one, right. So, now here say, o is one, if o is 1, in that case here that implies that c must be 0. If c is 0, c = 0 implies that l is 0 that does not block the propagation.

Because output q is still remained as 1 and now this q =1 is justified, r = 1 is justified and now here that justifies the propagation of the fault effect to the primary output, hence now my test vector is a = 1, b = 1, c is 0, d can be any value x, e must be 1, f must be 1 and s must be 0. So, these 1, 1, 0, x, 1, 1, 0 is the test vector. So, now here in order to track the decision when you are going backward here, you need to maintain a set and that set is called as J frontier.

It is set of all the gates, whose output is determined but that is not implied by the input value. So, like here, you want output = say 0 but both of the inputs are x, x of any gate. So, now here through those gates, you can justify these values. So, now here you made a decision that q = 1 from there you made another decision l =1, where in you fails, so you came back to the another previous decision making point k =1.

(Refer Slide Time: 30:19)



When you made k = 1, then here these objectives are justified then you made another decision, then r = 1. For r = 1, you made a decision that m should be equal to one and that succeed and then you are done. So, this way you explore the decision tree. Okay. Let us take small examples that consider both fault propagation and line justification. So, let us say, there is a struck at one fault, at this fault side, so that means just after fan out branch here.

So, if there is a struck at 0 fault here, we need to have D bar sorry; struck at one fault, so we need to have a symbol D bar here. In order to excide that fault here, you need to have 0 values at the primary input A and then now here your fault is activated, you have to propagate that. So, in order to propagate here, you need to have non-controlling value at B and C and since these are primary input you can always assign these values.

So, now here this will propagate this fault effect to the output of this gate and now this was D bar that can be propagated as d at g. Now, here if you look at, this is fan out point and then there are 3 fan out branches. So, if you look at the D frontier; D frontier, consist of those gates whose one of the input is d and output is not determined yet. So, now here one of the input of this gate is d, one of the input of this gate is d, one of the input of this gate is d.

So, now here gate I, k and m are in the D frontier, you have to choose 1 and or propagate your fault effect through that gate. At the same time, if you look at the implication, if a is 0, so that means here, one of the input of this AND gate is 0, that means here, s will always be 1 by implication, okay. Now you select to propagate your fault effect through this NAND gate so if you; then in order to propagate through this what you need?

You need to have non-controlling value at the half input of this gate and that is d; d is the primary input. So, you can always assign 1 and this will propagate d bar here. So, now here if you look at the D frontier, then like this; one of the input of this gate is already d, one of the input of this gate is already d and output is not determined yet and now here one of the input of this gate is also d.

So, now here these 3 gates are in the D frontier; k, m and n and now you have to choose one gate through which you can easily propagate fault effect to the one of the primary outputs and that gate is n, because that is predict lose to the primary output. So, now you choose to propagate through this. This is inverting gate, so now here d bar can be propagated here as d, right. So, now you propagate this as d.

In order to do that, you need to assign all the half inputs to the non-controlling value, so that means j must be 1, k must be 1, l must be 1 and m must be 1, right. If all these things are there, then you can propagate d on the primary output n. Now, let us see how I can justify that. So say, I justify the first value, so now say we want to justify j = 1, in order to justify j = 1, what you need to do?

Because here, one of the input to this gate is already one, so that means the other input must be 0, right. So, that means e bar must be 0, if e bar is 0, in that case here, e must be 1 and if e is 1, in that case here, this d bar; sorry; this d will propagate here at k output as d bar and then there is a conflict because you wanted one and then you happen to get d bar, that is conflict and you have to back track.

So, now here what you do in back tracking? So, you have to come back to your D frontier, now in D frontier, you have the gate k and m, so you have to choose 1. So, in order to choose 1, now you want to propagate the fault effect through gate k and now in order to do that, in order to propagate through this one again you want one here. If e is one, in that case here, this will allow you to propagate fault effect through this one and then this e is equal to one gave s you the e bar as 0 and then here j would be 1.

Now, you will get this h1; I is d bar, j = 1 and k is your d bar, right. Now, then here the output of this gate is not yet determined, output of this gate is not yet determined. Now again here, you have 2 gates in the D frontier, one is m and the other is n, because here there are 2 d bars at the input of this gate. So, now again you need to choose one out of these 2 and then of course you would like to choose gate n because it is closer to the primary output.

So, now here in order to do that, what again you need? You need 1 here and 1 from here, so say you want to justify 1 here. In order to justify 1 here, what you want? You want f = 1 and when f = 1 in that case here, that will give f bar = 0. This also propagate here d bar but now here when you have d bar here 1, d bar here 1 and d bar that also at the same time this will give you d, at output and now here you are able to propagate d at the primary output and you are done.

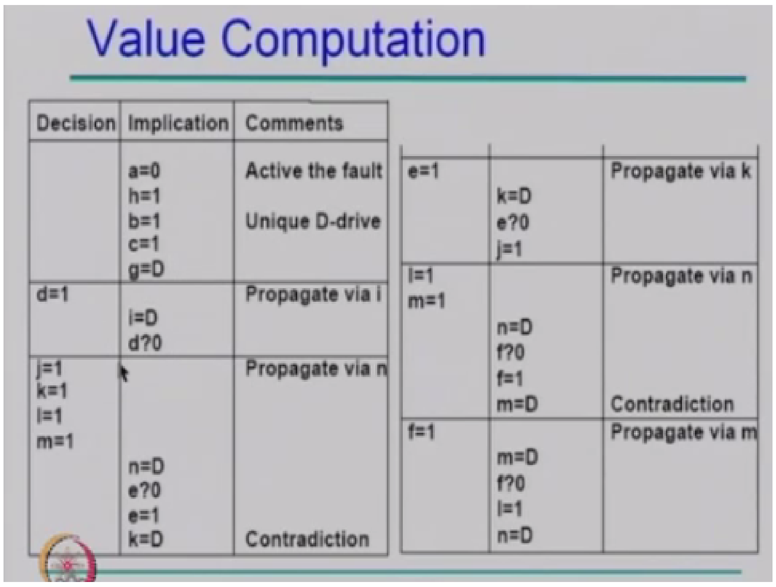
So, now here what test vector can test this particular fault; that test vector is a = 0, b = 1, c =1, d =1, e = 1 and f = 1, so now here 0, 1,1, 1, 1, 1 is the test vectors for this fault. This way we can detect or we can degenerate test vector for a given fault and so as I said that this is very directed approach. In worst possible case, it may need to explore all possible combinations. So, if you look at the complexity, I do not want to go in that detail, but this is NP complete problem.

So in worst case, it may need to explore exponentially or entire search space but observation says that most of the time you; because this is a directed approach, you can quickly generate test and hence this is scalable approach, you can handle large number of gates in a circuit. Then, there means one of the issue here is always you need to justify or assign some non-controlling value at the intermediate points.

And because you cannot assign non-controlling value at the intermediate point, you have to go back all the way to the primary input to assign those values. This problem was further looked at by Goel from IBM and his came up with another algorithm that was known as Podem, in which he says that when we cannot assign any value at the primary intermediate point, why we are looking at intermediate point?

Why not we need to go all the way to one of the primary input, assign some value, forward implicate it and see whether it allows you to propagate the fault effect or not, if it does not allow, if it blocks then you change that value and that algorithm Podem became almost 100 times faster than d algorithm. Then further the problems of Podem algorithms were, or the; in other way, I should say that some of the new concepts were introduced on top of the podem algorithm by Prof. Fujiwara, that is known as Fan out oriented test generation.

(Refer Slide Time: 39:13)



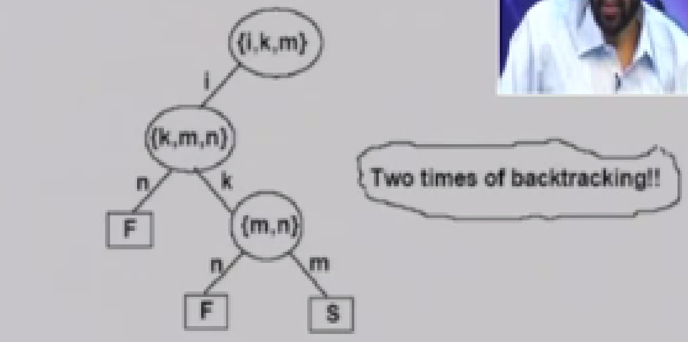
Value Computation

Decision	Implication	Comments
	a=0 h=1 b=1 c=1 g=D	Active the fault Unique D-drive
d=1	i=D d?0	Propagate via i
j=1 k=1 l=1 m=1	n=D e?0 e=1 k=D	Propagate via n Contradiction
e=1	k=D e?0 j=1	Propagate via k
l=1 m=1	n=D f?0 f=1 m=D	Propagate via n Contradiction
f=1	m=D f?0 l=1 n=D	Propagate via m

And that was almost 100 times faster than the podem algorithm. So, these are the various algorithms you can go through the literature and look at the various recent algorithms and as I said that here most of the test generation, engines are based on either podem or fan, mostly fan, okay. This summarise the decision making process you can look at all these steps that we discuss earlier, okay.

(Refer Slide Time: 39:22)

Decision Tree



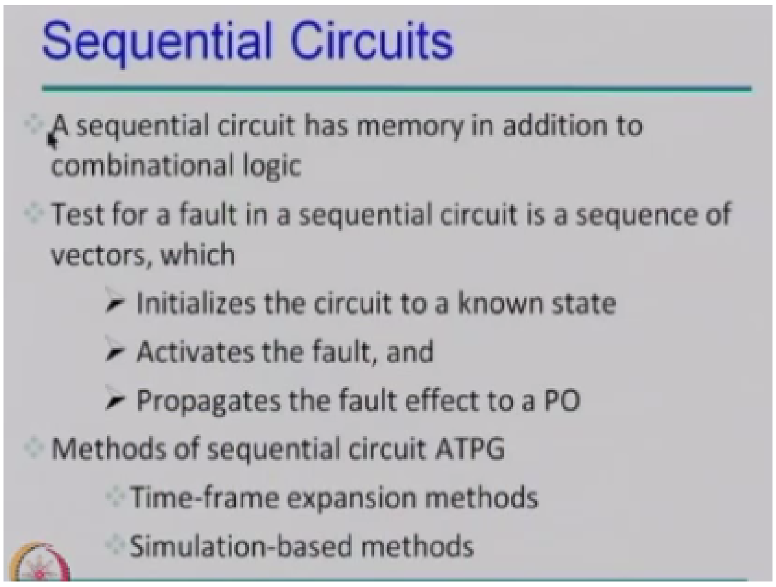
These are the decision tree, which we made here in earlier, your D frontier as get k, l and m. then, we have chosen l and then we introduce another gate n in the D frontier, we had chosen n, we failed then we came back to the precious decision point, where in you had m and n in the D frontier, you have chosen n again, you failed then you come back and which you had chosen m and then you succeed.

So, now here there were 2 back tracks in this. So, now here as I said that critically test generation complexity of a combinational automatic test certain generator is NP complete, but observation says that you can generate test in reasonable time. The big problem is resistance faults. So, if fault test vector access this test, these algorithms can generate a test vector pretty fast, but if it is resistant, it may need to; not necessary all the time, but may need to explore the entire decision tree and that may not be practical.

So, now in general what this practical ATPG tool does? All these practical ATPG tools are based on if they keep a back track limit. If say back track limit is 1000, then if it exceeds 1000, it will above that and idea is; that most of the time, again keep in mind, I said most of the time; most of the time that fault may be resistant fault, but it is not sure and that is the reason we; it is difficult to get very high coverage that means 100% fault coverage.

Because we have to do about that if your decision making points are exceeding or back tracks are exceeding beyond the assigned limit. So, that means if you are about limit is lower, you can generate a test quickly because then if it exceeds beyond that limit, it about that and generate test for the next fault. If you increase the about limit, it will take longer time and then here this time is not purpose, it grows exponentially.

(Refer Slide Time: 42:14)

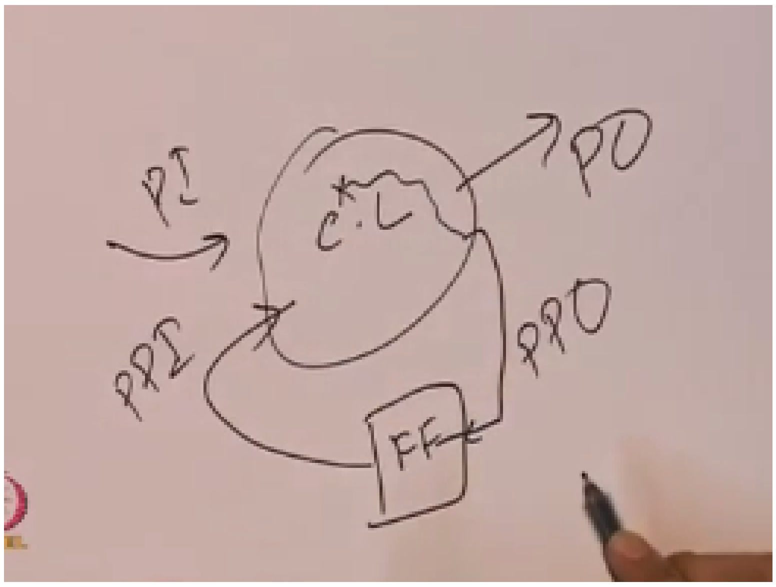


Sequential Circuits

- A sequential circuit has memory in addition to combinational logic
- Test for a fault in a sequential circuit is a sequence of vectors, which
  - Initializes the circuit to a known state
  - Activates the fault, and
  - Propagates the fault effect to a PO
- Methods of sequential circuit ATPG
  - Time-frame expansion methods
  - Simulation-based methods

This was the test generation for combinational logic. Now, in general, we do not have combinational circuit, we have sequential circuit and how sequential circuits are different from combinational circuit? Sequential circuits have additional memory element in that, so now, the bigger problem we may end up to is, your fault effect may propagate to the flip flops or memory elements rather than propagating to one of the primary outputs.

(Refer Slide Time: 42:36)

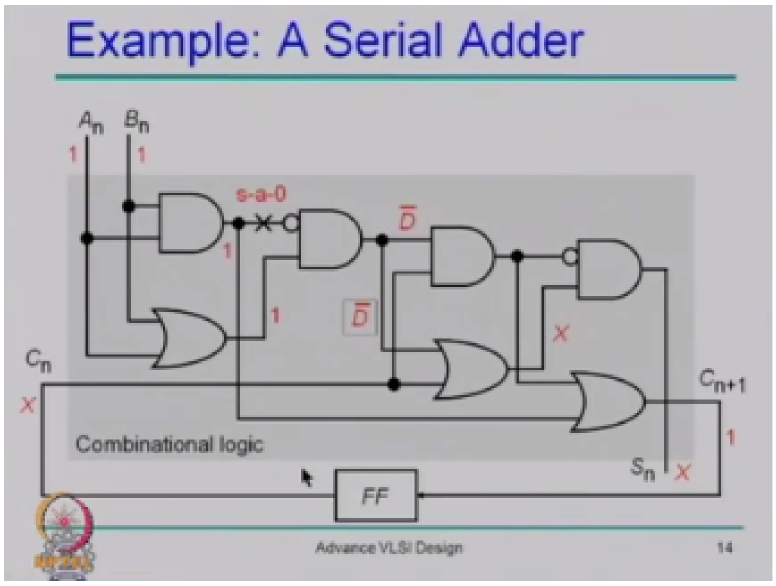


Look at the structure, this is your combinational logic and this is your flip flop, you are getting input from this one. This are your primary input, primary output, these inputs from the flip flop are known as pseudo primary input and the output from the circuit, which are going to the flip flop are known as pseudo primary output. So, your fault effect may fault may be here and then this may propagate to the one of the pseudo primary output and I cannot observe that.

So, now we are; what you need to do? You have to run your circuit again, so the until the time, when you; it propagates to the primary output, where you can observe. Another issue is that here, when you are generating test for this, you may demand some value from this pseudo primary input or from the flip flop and those flip flop may be in any state, when you power up your circuit, right.

So, that means here you how to initialise your circuit? To some fix some value in this flip flop, so it demands, so test for a fault in a sequential circuit is a sequence of vectors rather than a single vector and that sequence comes from; first thing is you need to initialise circuit to a non-state, then you have to activate the fault and if that fault effect is not propagating to the primary output and propagates to the one of the pseudo primary output in that case you have to run your circuit in several cycle to propagate the fault effect to the primary output.

(Refer Slide Time: 44:33)

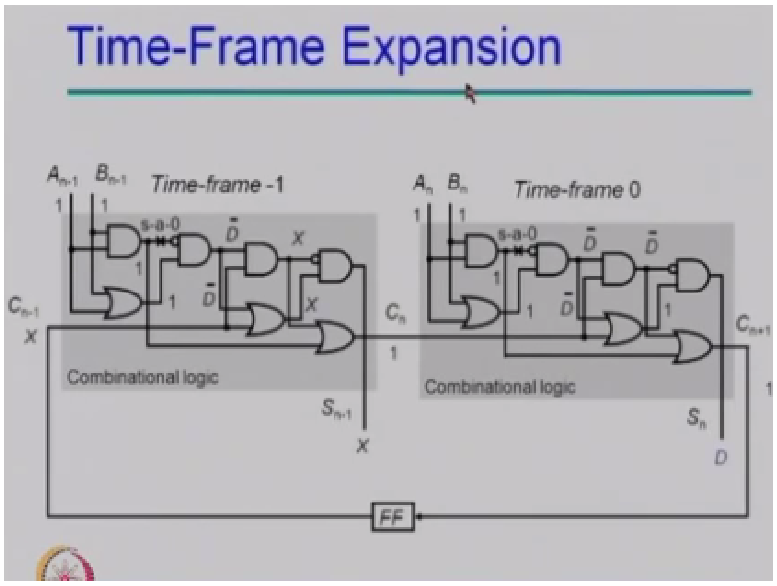

Example: A Serial Adder

So, that means here you need a sequence of vectors rather than a single vector. In order to handle this problem, there are 2 methods; one is the time frame expansion mention method and other is stimulation based method. Time frame expansion method, I will explain you briefly. So, let us look at this example, this is brief example. Say, I guess all of you are familiar with the serial adder, this adder has say 2 inputs A and B and then 2 outputs; one is the output and the other is the carry.

So, now here carry would be stored in the flip flop and then every cycle you are getting new inputs and giving the output of that cycle. So, now here say there is a struck at 0 fault here, for struck at 0 fault here what you demand? You demand one from here then by implication you say that An and Bn must be 1. In order to propagate this fault effect here, there is a D and then here propagate here that as D bar here what you demand, you demand one from here that full fill by implication An = 1 and Bn =1 will gave you this as 1.

Now your fault effect is here, right. So, now here there are 2 parts you have to propagate that fault effect through this gate or through this gate, right. In order to propagate, you need to have non-controlling value here, so here there must be one and here there must be 0. But, because your circuit can power up in any state, in that case you have x and then once you have x here as one of the inputs to theses gates, output would be a x.

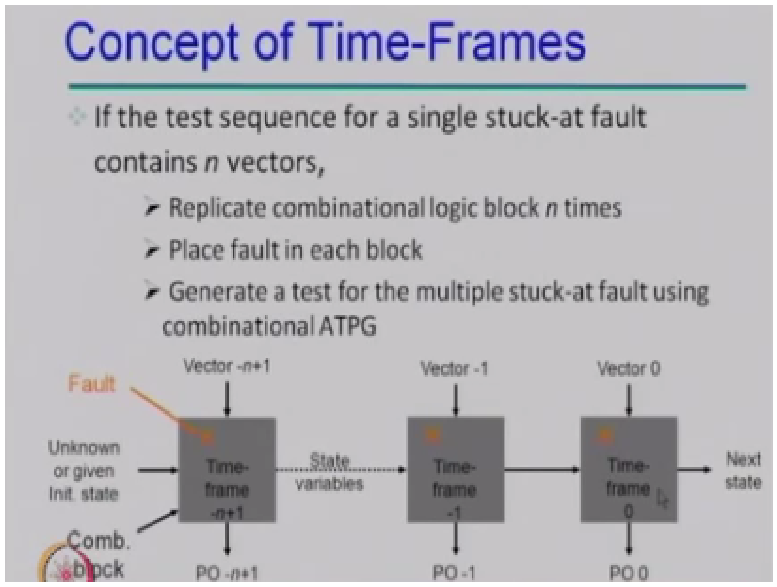
(Refer Slide Time: 46:20)



And then your primary output would be x and then you will assign value 1 in the flip flops. So, that means here, you are not able to propagate fault effect to the primary output. If you start from value x, now here what is the solution? And solution is that here you have to initialise this flip flop into a non-state and that non-state is one that you are demanding, so that means here you have to control your circuit in multiple time frames.

Unroll means here, you have to have 2 copies of this circuit; 2 copies of this circuit does not mean that you implement or fabricate 2 copies, you are fabricating only one copy of the circuit, but now here for the test generation purpose, you replicate that copy. So, now here in this; in both of the copies because the same fault x is here as well as here. So now, in order to excide that here, you need to have 1, 1 as input from here.

(Refer Slide Time: 47:38)



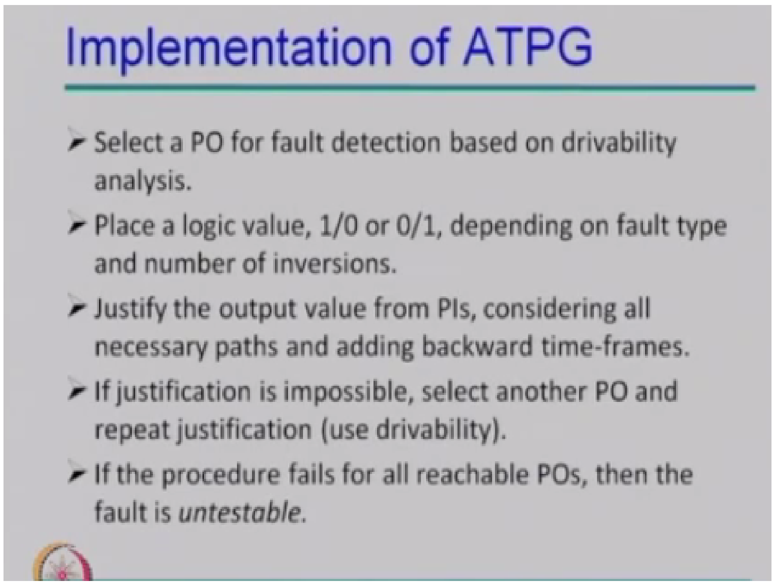
Concept of Time-Frames

- If the test sequence for a single stuck-at fault contains *n* vectors,
  - Replicate combinational logic block *n* times
  - Place fault in each block
  - Generate a test for the multiple stuck-at fault using combinational ATPG

And then you are demanding one from this one. In order to have one here, you need to; you demand one from here and in order to demand one here, you need to have 1, 1 here. So, that means here, you need to have a sequence of 2 inputs; 1, 1 followed by 1, 1 that contains this fault. So, that means here this gives; tells you that the how complex test generation for sequential circuit is, okay, so here what we need to do?

We need n vector, so we need to replicate our circuit n times and now here the same fault may accessed in n copies, so that means in total combinational circuit that I have, I have n number of faults. So, now here this is the instances of multiple faults rather than single struck at fault. Now, the time when I excide the fault that we say vector 0 and then here time frame that we use to initialise our circuit, we call at -1, -2, -3, -n , right.

(Refer Slide Time: 48:50)



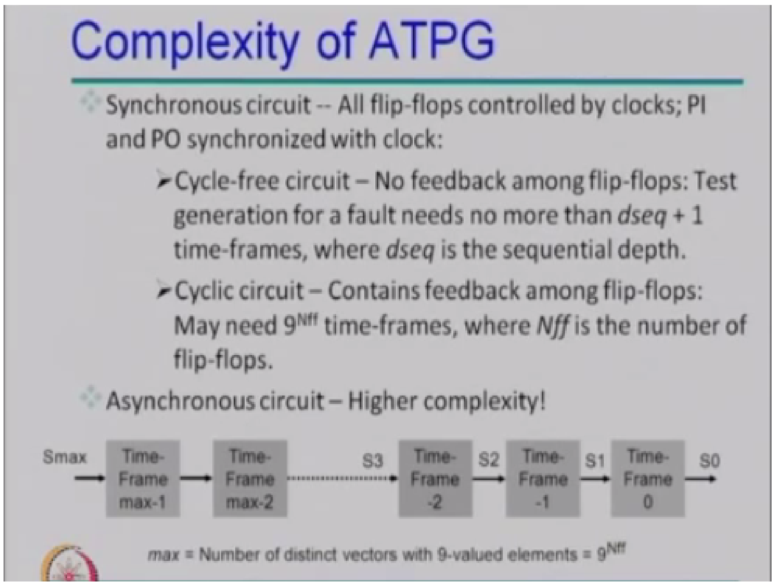
Implementation of ATPG

- Select a PO for fault detection based on drivability analysis.
- Place a logic value, 1/0 or 0/1, depending on fault type and number of inversions.
- Justify the output value from PIs, considering all necessary paths and adding backward time-frames.
- If justification is impossible, select another PO and repeat justification (use drivability).
- If the procedure fails for all reachable POs, then the fault is *untestable*.

And how many times I need to unroll? I need to unroll until I reached to a point where in I start from all access from the memory elements or flip flops and then here because the fault effect may propagate to the pseudo primary output, in that case, again i need to unroll and I need to unroll until I am able to propagate fault effect to one of the primary outputs. So, I need to unroll it multiple times. So now here, the typical ATPG implementation.

Select the one of the primary output based on the drivability, I skip that for a while, but here say to randomly one of the primary output, it plays as logical value 1 or 0 or 0 or 1 depends, so these are the composite values; D or D bar depending on the fault type, you justify a output from the primary inputs and considering all necessary paths and adding the backward time frames.

(Refer Slide Time: 49:42)



Complexity of ATPG

- Synchronous circuit -- All flip-flops controlled by clocks; PI and PO synchronized with clock:
  - Cycle-free circuit – No feedback among flip-flops: Test generation for a fault needs no more than *dseq* + 1 time-frames, where *dseq* is the sequential depth.
  - Cyclic circuit – Contains feedback among flip-flops: May need 9^{Nff} time-frames, where *Nff* is the number of flip-flops.
- Asynchronous circuit – Higher complexity!

max = Number of distinct vectors with 9-valued elements = 9^{Nff}

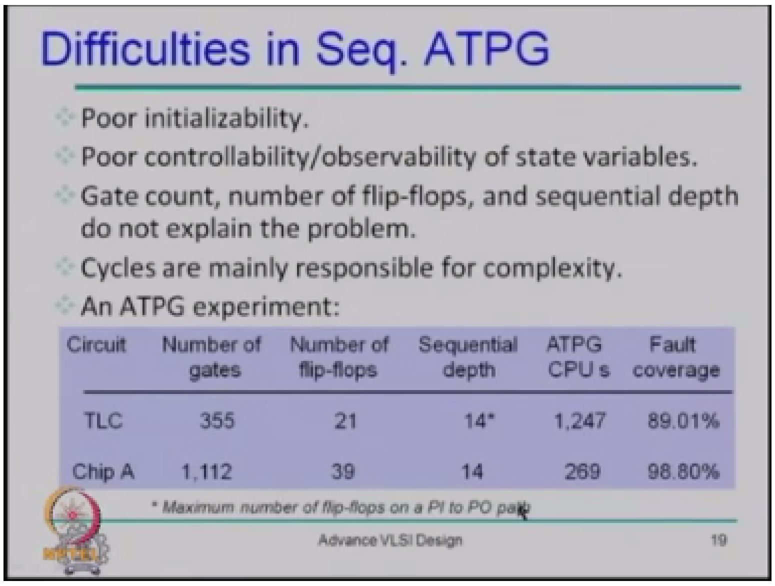
If justification is impossible in that case, you need to select another primary output to propagate and you repeat all these things. If you procedure fails, for all reachable primary output in that case here, you have to say that fault is untestable. Now, here you are unrolling this in multiple time frames and now the question is worst possible case, how many times you need to unroll it?

And this gives you some scary number; your circuit may be cycle free or may have cycle. So, that means here you may have feedback or you may not have feedback. If circuit is cycle free, in that case, you need to unroll your circuit maximum sequential depth of the circuit = 1 and sequential depth of circuit is defined as maximum number of flip flop appears in any pair of input, output.

So, if you say in one of the path, if there are maximum 3 flip flops are appearing in that case here, total 4; you need to unroll your circuit and you can easily understand that because if your circuit has sequential depth 3, that means here in 3 time frames, you may be able to initialise all the flip flop. Because, there is no feedback, there is only feed forward. If circuit is cyclic circuit and there are n number of flip flops, in that case here, you need; you may need to maximum unroll n raise to the; 9 raise to the power number of flip flop times.

That is really scary figure, because assume that you have only 2 flip flop, then you may need to unroll it 81 times, 2 flip flop circuit is pretty small circuit, though this numbers are scary but most of the time you do not need to unroll that many times, you need to unroll pretty small number of times then then these numbers, but in order to give a guarantee, you need to explore this.

(Refer Slide Time: 51:59)



Difficulties in Seq. ATPG

- Poor initializability.
- Poor controllability/observability of state variables.
- Gate count, number of flip-flops, and sequential depth do not explain the problem.
- Cycles are mainly responsible for complexity.
- An ATPG experiment:

Circuit	Number of gates	Number of flip-flops	Sequential depth	ATPG CPU s	Fault coverage
TLC	355	21	14*	1,247	89.01%
Chip A	1,112	39	14	269	98.80%

* Maximum number of flip-flops on a PI to PO path

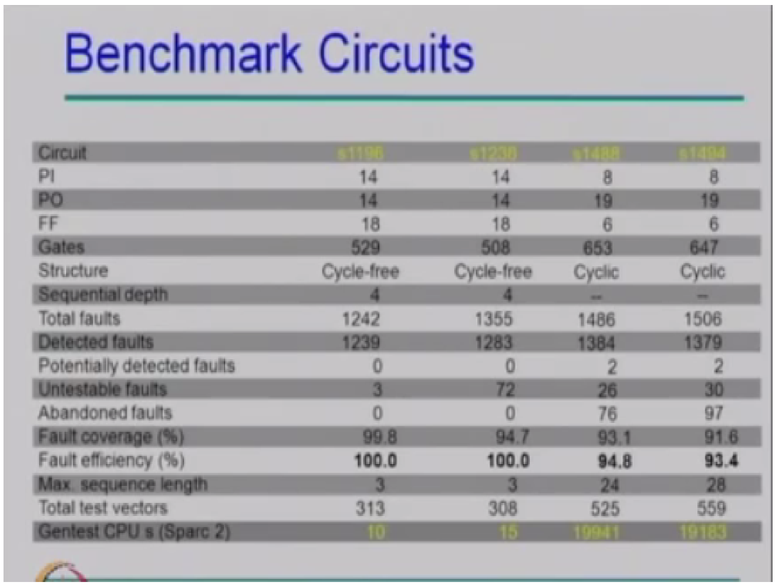
Advance VLSI Design 19

So, now here it is very very, very difficult to generate test for reasonably large sequential circuit if it has cycles involved in this. Now, what is the solution and solution so now; means the difficulties are poor initializability of the sequential circuit, poor controllability or observability of a state variables. Because you cannot directly control or observe those variables, gates count and number of flip flops are increasing.

And then sequential depth is increasing and cycles are main responsible for this complexity. If you look at the test generation time and fault coverage, so if you have a small circuit then traffic light controller that has say 355 gates and 21 flip flops and sequential depth is say 14 in that case here, ATPG time would be something 1247 seconds and fault coverage is still 89%.

Whereas, the another circuit here you have chip A, that has pretty higher and number of gates and number of flip flops and sequential depth of 14, but here is still it needs less number of; so now here the; what does this slide tells you that gate count and number of flip flops are really not the determining factor. It is the sequential or cycles, which are determining the how much time or how complex this process is?

(Refer Slide Time: 53:21)

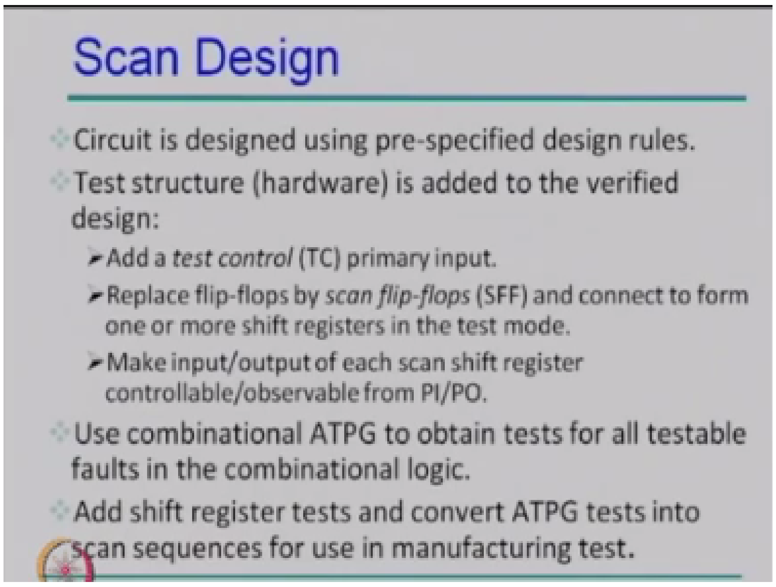


Benchmark Circuits

Circuit	s1196	s1238	s1488	s1494
PI	14	14	8	8
PO	14	14	19	19
FF	18	18	6	6
Gates	529	508	653	647
Structure	Cycle-free	Cycle-free	Cyclic	Cyclic
Sequential depth	4	4	--	--
Total faults	1242	1355	1486	1506
Detected faults	1239	1283	1384	1379
Potentially detected faults	0	0	2	2
Untestable faults	3	72	26	30
Abandoned faults	0	0	76	97
Fault coverage (%)	99.8	94.7	93.1	91.6
Fault efficiency (%)	**100.0**	**100.0**	**94.8**	**93.4**
Max. sequence length	3	3	24	28
Total test vectors	313	308	525	559
Gentest CPU s (Sparc 2)	10	15	19941	19183

If you look at some of the bench mark circuits, some circuits; these are (()) (53:28) 89, bench mark circuit, this as 1196 and s as 1496. So, these 2 first 2 circuits are cycle free circuits and last 2 circuits are cyclic circuits. Though here, all are having the similar kind of flip flop like here a cyclic circuit do have 18 flip flops, now you can see the test generation time, a cyclic circuits will takes 10 to 15 seconds, whereas the cyclic circuit are taking about 20000 seconds.

(Refer Slide Time: 54:17)



Scan Design

- Circuit is designed using pre-specified design rules.
- Test structure (hardware) is added to the verified design:
  - Add a *test control* (TC) primary input.
  - Replace flip-flops by *scan flip-flops* (SFF) and connect to form one or more shift registers in the test mode.
  - Make input/output of each scan shift register controllable/observable from PI/PO.
- Use combinational ATPG to obtain tests for all testable faults in the combinational logic.
- Add shift register tests and convert ATPG tests into scan sequences for use in manufacturing test.

So, huge difference and hence here, it is not practical to generate test for a significantly large sequential circuit. So, you have to adapt some other technique and as we know that this complexity is coming from the cycles in the circuit and poor initializability and controllability of the circuit. So, if we happen to initialise circuit by any random value and we can observe any value from the flip flops.

You can always use the same combinational ATPG to generate the test and then your test generation complexity is same as the combinational circuit test generation complexity. With this, I stop here today; will continue with the test generation for sequential circuit in the next class. Thank you very much. Good day.