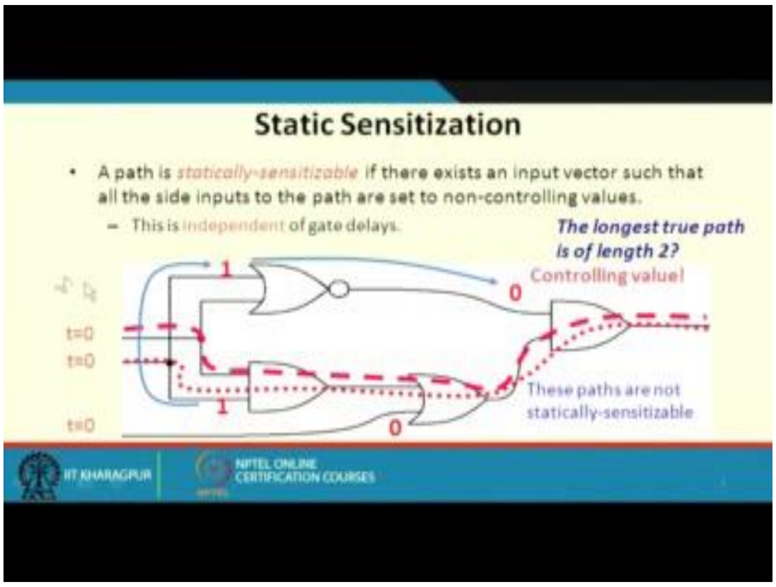
VLSI Physical Design

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

Lecture - 36
Time Closure (Part 5)

So, we continue with our discussion on the concept of false paths. And how we can identify and characterize false paths in a circuit netlist. Just to recall false paths are important to eliminate certain paths from timing analysis to identify the criticality of paths and so on and so forth which can help in avoiding unnecessary calculations and unnecessary processing when you are trying to optimize a circuit with respect to the timing characteristics and parameters.

(Refer Slide Time: 01:01)



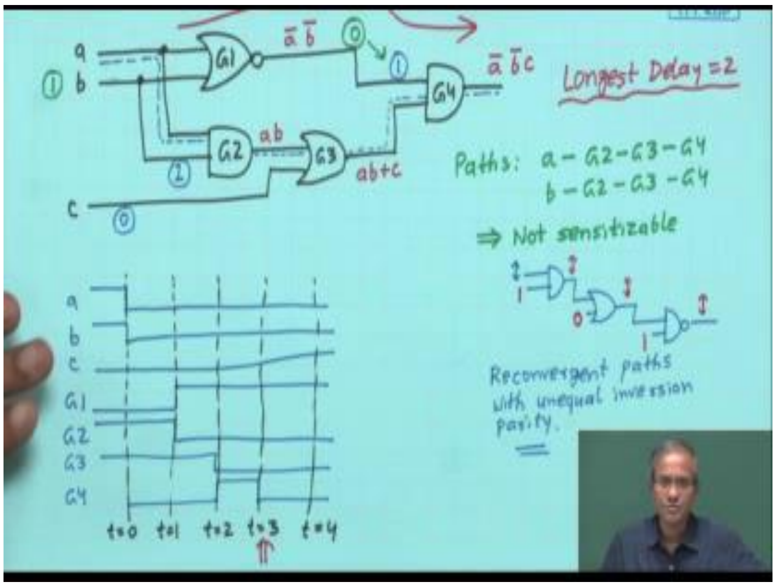
Static Sensitization

- A path is *statically-sensitizable* if there exists an input vector such that all the side inputs to the path are set to non-controlling values.
  - This is independent of gate delays.

The longest true path is of length 2?
Controlling value!
1
0
t=0
t=0
t=0
1
0
These paths are not statically-sensitizable
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, here we talk about something called static sensitization. So I shall be explaining this concept with the help of an example. So what I say is that given a circuit some path is said to be statically sensitizable, if there is an input vector such that, so from the input to the output any changes can be propagated means in all the other inputs to the gates that lie along the way are set to non-controlling values. And this static sensitization will be independent of gate delays. Let me illustrate this with the help of an example let me work this out.

(Refer Slide Time: 01:52)



So, this example whatever is given is a circuit like this. So let us give some name to the gates. G2, this is the circuit which is given this combination circuit. Just if you make a quick computation what is the logic function that is realized by this circuit. A b this is a NOR gate so the output will be a bar b bar. There is an NAND gate this output will be ab OR gate, this will be ab OR c. And this is NAND gate this and this so a bar b bar and ab will cancel out a bar b bar c. This is the final output right. Now what we are trying to say here is that let us consider these paths. Now we consider 2 paths first path is starting from input a to gate G2 to gate G3 to gate G4. This is the final output. And also similarly from the input b, so we are saying that these paths are not sensitizable.

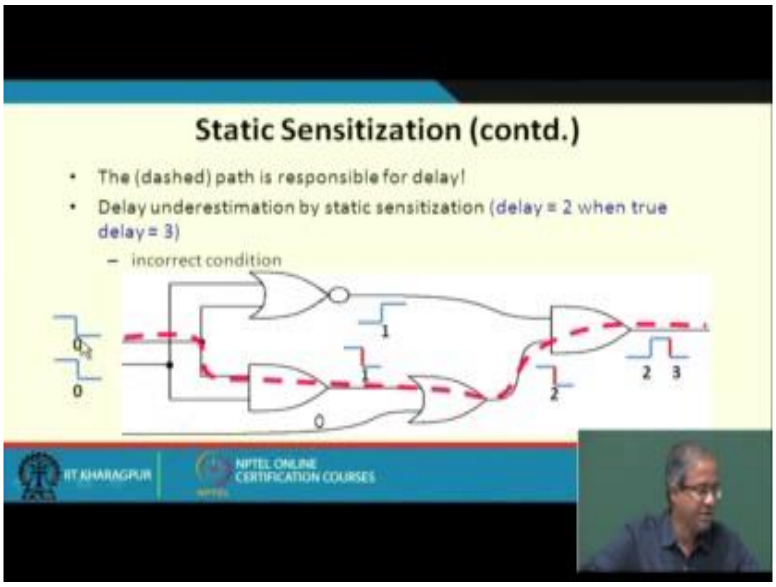
So, what do we mean by sensitizable? We recall, we mention that if I have a sequence of gates along a path let say I have an NAND gate, then I have an OR gate, let say I have another NAND gate along a path. The other inputs are, there so if I want to propagate a signal from one line let say from here, so along this path so I have to apply non-controlling value to the other inputs along the gates, like this transition we will come to the output of this AND gate if I apply a 1 at this input for an or gate I have to apply a 0 for a NAND gate again I have to apply one this will ensure that this transitions will propagate. And depending on the gate the other inputs or the side inputs must be applied non-controlling values so that the change can propagate, right.

So, in this circuit also let us try in the same way that can we propagate a signal from via G2 G3 G4, let us try. From a via G2 means I am trying to talk about this path. A via G2, so this is an NAND gate so I have to apply a non-controlling value 1 in the other input, then via G3 is an OR gate I have to apply a 0 in the other input. So again I get into G4 NAND gate I have to apply one here. Then only this will propagate to the output, but the trouble here is that if you just look that, as soon as we had applied a 1 in this other side input this b also has become 1 right. This means that I have to apply a 1 in b.

Now this is a NOR gate, a 1 in any of the inputs will make the output 0, but we require a 1 here. There is a conflict. Same thing you can check for the other parts. Just by applying non contouring values in the side inputs we cannot sensitize these paths. This is actually true just to tell you for the circuits whenever you have a something called RECONVERGENT paths with unequal inversion parity. Like see there are fan outs from a there is a fan out connection one is going via G1 then to G4 other is going via G2 G3 G4.

Now, in one path there is one inversion a NOR, other path there are no inverting gates. So in one path the number of inversions are odd other path it is even there are again merging into a gate. This is called Reconvergent path with unequal inversion parity so whenever such a thing is there is likelihood that you will not be able to synthesize these individual paths. So this is a drawback. Now here, why you were talking about this? You see here your static timing analyzer what it will tell you that, that well, this path can never be sensitized this will be identified as a false path, and it will say that well this is my only path which is of interest. So my longest delay is true, longest delay is true, right.

(Refer Slide Time: 08:26)



Static Sensitization (contd.)

- The (dashed) path is responsible for delay!
- Delay underestimation by static sensitization (delay = 2 when true delay = 3)
  - incorrect condition

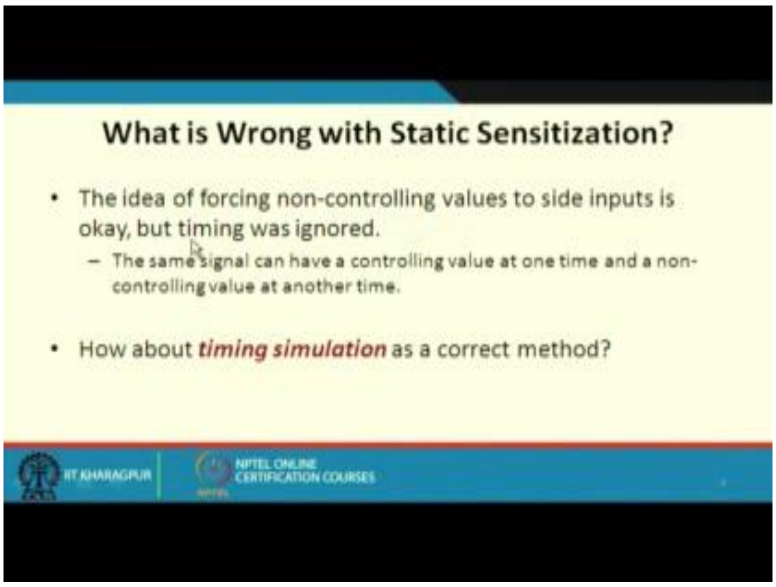
But now, let us suppose what I do is that, the same circuit, I am just applying some inputs and I am trying to do a timing simulation and see what is happening. So both the inputs the third input gets 0 the first 2 inputs are going from 1 to 0 at time t equal to 0. So let me show you the timing diagram so it will be good for you. So I am just marking the time. So this is t equal to 0, t equal to 1, 2, 3, 4 and so on. So now, to show the inputs this a makes a 1 to 0 transition at time t equal to 0. So let say that a input is like this, b is the same, b is also making a 1 to 0 transition at time t equal to 0, c is always 0 let say c is constant 0. Now let us just assume that the gate delays are all 1. So let us say one by one what will be the output of G1. G1 is the NOR of a and b. Initially both are 1, 1. So it will be 0 and after that both are 0, 0, it became one, but delay of 1 so it will become 1 after one-unit delay here and it will remain 1.

Talk about G2, G2 is NAND. Both of them will be 1 and once they 0 they will become 0. So again with a delay of 1 G2 will remain 1. It will become 0 here. G3, G3 is the OR of G2 and c is always 0. So it will be G2 itself, but with a delay of 1. So it means the delay of 1 it will become like this.

And finally, G4 what will happen. G4 is the AND of G1 and G3, G1 and G3, so you see it will be one getting this region, but with a delay of 1. So both are getting 1 here so after delay of one this will come here so it will come here it will remain till here. So you see here. So our false path deduction algorithm tells us that this is a false path therefore, this is a true critical path the longest delay is 2, but for a particular input what we have found is that the output is getting stable only after time t equal to 3, right you see there is a glitch in the output temporary change because the output is supposed to be 0m a bar b bar c I am applying say a0 b0 c0. So the output should be zero, but in between it is going to 1 periodically then it is stabilizing to 0. So the output is getting stabilized only after delay of 3 and not 2 right, this is what this calculation says right.

So, here what we are saying is that, the static sensitization method that you are following this is doing some delay underestimation. So true delay is 3, but it will give us a delay of 2 which is an incorrect condition, right.

(Refer Slide Time: 12:16)



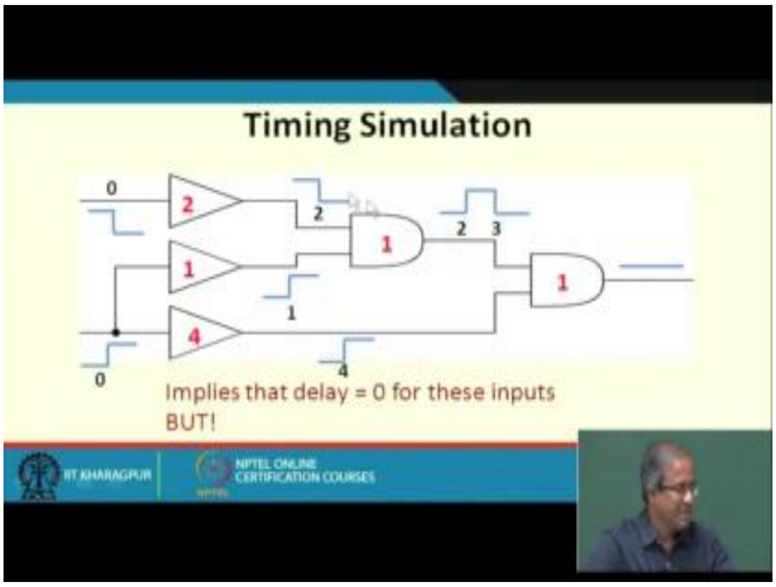
What is Wrong with Static Sensitization?

- The idea of forcing non-controlling values to side inputs is okay, but timing was ignored.
  - The same signal can have a controlling value at one time and a non-controlling value at another time.
- How about ***timing simulation*** as a correct method?

So, what is wrong? So the wrong I said one thing I had said that there is a Reconvergent fanout with unequal inversion parity that condition is holding, that is one problem, but with respect to the controlling and non-controlling values, the issues that we are ignoring the timing when the values are becoming controlling and non-controlling because you see the values of the side inputs are not constants they are also changing in time so sometimes they are controlling sometimes they are non-controlling. So it is not that statistically in a constant way you are applying a fixed value. So temporarily they might the status might change and that creates the problem, right.

So, this is what is mentioned here, the same signal can have a controlling value at one time and non-controlling value at another time. So let us try out something called timing simulation.

(Refer Slide Time: 13:22)



Timing Simulation

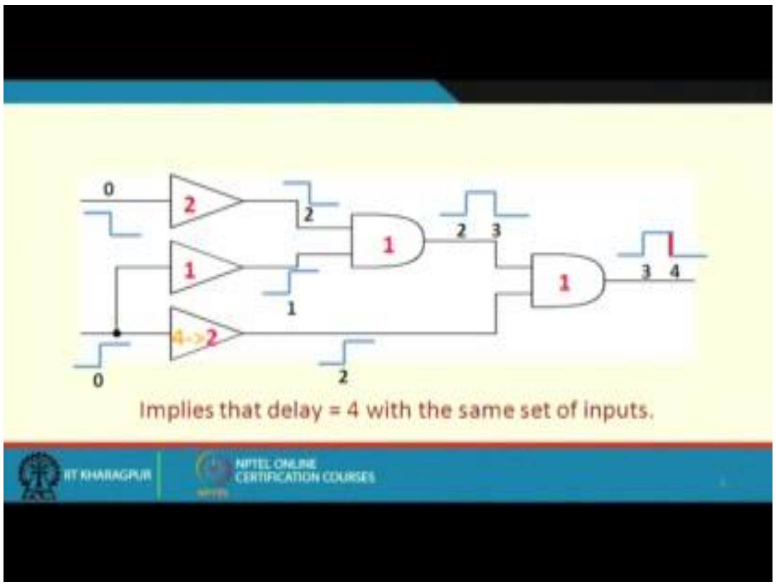
Implies that delay = 0 for these inputs
BUT!

Let us see whether this gives us the correct result. Timing simulation means we simply calculate the times and see when signal transitions are taking place. So we again take an example like this. Let us assume that these are some gates some buffers NAND gate NAND gate some hypothetical delays, let say 2 1 4 1 1. So let us assume that there are 2 inputs the transitions are taking place at time equal to t 0. Because the delays of this gates this transition will go the output with a delay of 2, so this transition will be taking place at 2 delay of 1 1 delay of 4 at 4.

Now, this gate is providing a delay of 1, but here you see at time 1 it is going high at time 2 it is going low. So between 1 and 2 both are both are high therefore, there will be delay of 1 again so between 2 and 3 there will be a period where both are high. Look at the output of 1 this signal is going 1 at 4, but by 4 it is already at 0. So the output is steady 0. So timing simulation will show that if you apply this; that means, one input is 0 other input is also one so this is basically an NAND function. So the output is supposed to be 0 and it is already 0, so it should be good; that means delay is 0.

But you see for this case the delay is showing to be 0, but suppose I reduce the delay of one of the gates let say this is 4.

(Refer Slide Time: 15:04)

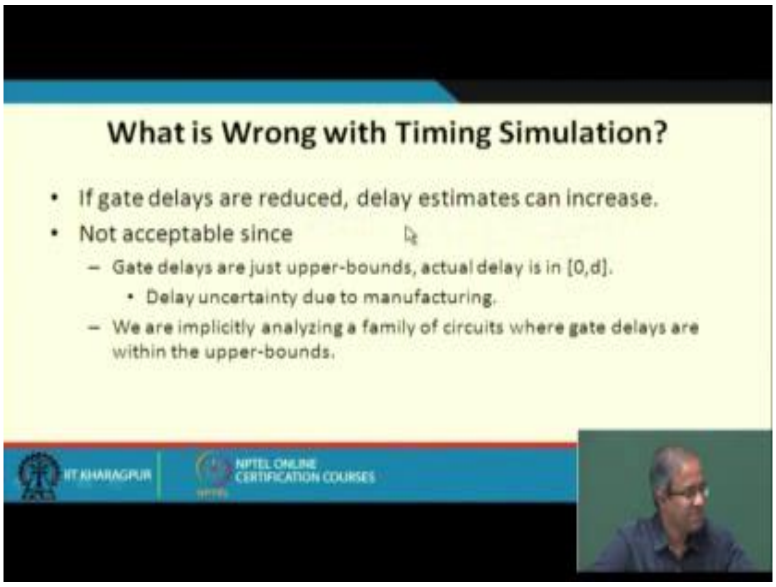


Implies that delay = 4 with the same set of inputs.

Let us make it 2. And a let us do the same calculation again this part is same. So here this transition will be appearing earlier at time 2, so here also it is making a transition so now, if you do an NAND these 1s and 1s will overlap. So there will be a transition after a delay of 1 between 3 and 4.

So, you see only after time 4 the output is getting stabilized. So for this case as soon as we reduce the delay, for the same input values same input transition the delay is be showing at 4. So this also gives you a very interesting thing sometimes a static timing analyzer might show you an increase in delay whenever you are reducing the delay of some components. So you expect things to become faster, but as this example shows that for the same set of input transitions, earlier there was no delay virtually because the output is 0 it is always zero, but now only after time 4 the output is getting stabilized right, this is one issue.

(Refer Slide Time: 16:24)



What is Wrong with Timing Simulation?

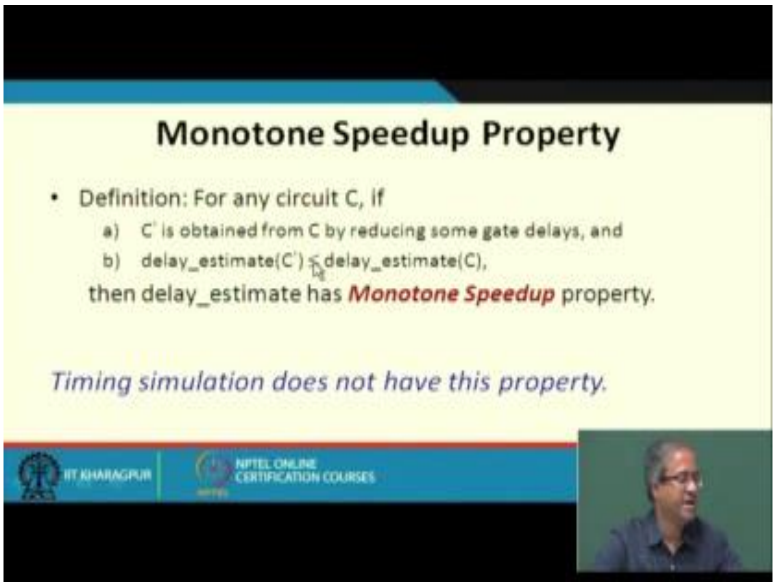
- If gate delays are reduced, delay estimates can increase.
- Not acceptable since
  - Gate delays are just upper-bounds, actual delay is in [0,d].
    - Delay uncertainty due to manufacturing.
  - We are implicitly analyzing a family of circuits where gate delays are within the upper-bounds.

Fine so what is wrong here? This is one thing I just now mentioned. If we reduce some gate delays it is possible for some delay estimate to increase, but this is actually contradictory right. So we would expect that if I reduce a gate delay my path delay or circuit delay should also reduce, but here it is happening the other way round. So my timing simulation is not proper in the sense, but even if I am reducing the delay of a gate my timing simulation is saying that the delay is increasing.

So, this is not acceptable because in practice many gate delays can actually get reduced because the gate delays value that we usually use in simulation they are upper bound. We say a delay is 4 which means this is not actually 4 the maximum delay is 4. Because when you are fabricating gates in a chip there can be lots of variations in fabrication. Some gate maybe 3 some gate may be 2.5 some gate maybe 3.8. 4 is the maximum limit you are saying that is the worst case delay, right.

So, here actually we are implicitly analyzing some circuits where gate delays are within some upper bound. So this creates a problem.

(Refer Slide Time: 17:51)



Monotone Speedup Property

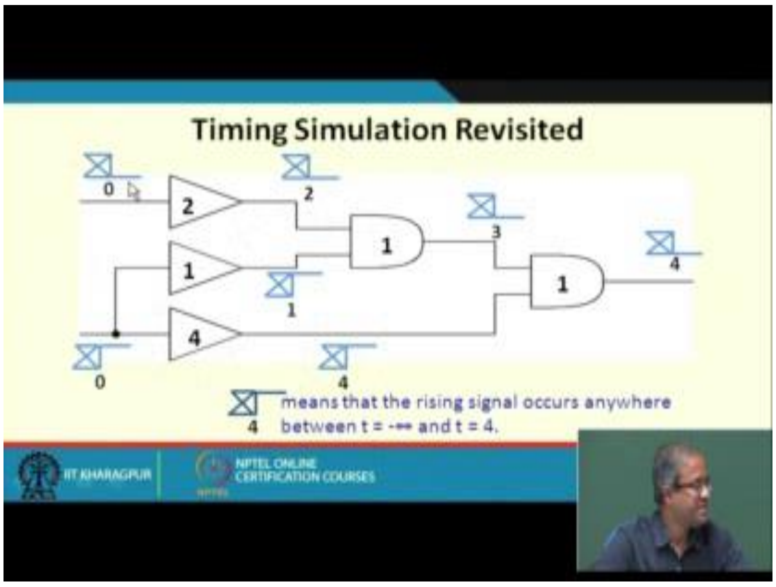
- Definition: For any circuit C, if
  - a) C' is obtained from C by reducing some gate delays, and
  - b) delay_estimate(C') ≤ delay_estimate(C),

  then delay_estimate has ***Monotone Speedup*** property.

Timing simulation does not have this property.

Now, we need some simulation, where some property called monotone speedup should be satisfied. So what is monotone speedup property? This says that if you are given a circuit C then we can get an circuit C dash by reducing some gate delays and in the process the delay estimate should also be reduced or at least be equal not go up. There is a monotonicity. So if you reduce some of the gate delays your total delay estimate of the circuit should not increase. This is the condition which is mentioned, there has to be monotonicity in the property. If this property satisfied you can say that your algorithm satisfies the monotonicity property.

(Refer Slide Time: 18:56)



Timing Simulation Revisited

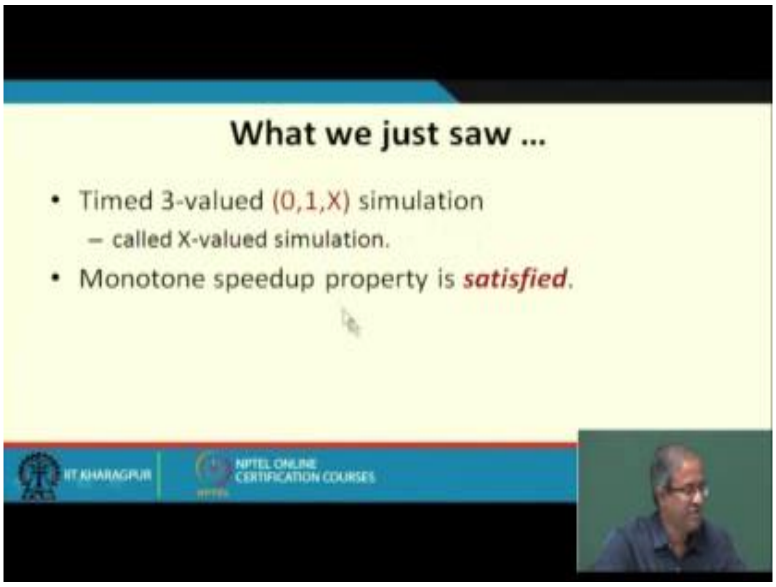
4 means that the rising signal occurs anywhere between t = -∞ and t = 4.

Now, timing simulation as we have seen; obviously, does not satisfy this property, but we can make some modification and see just one simple thing I am explaining with an example, this is something which is called ex value simulation. This is also timing simulation, but we are not showing exact signal transitions from 1 to 0 or 0 to 1. We are using some special logic value called x. This notation where the transition is taking place at time 4, this means that this rising edge can occur anywhere between t equal to minus infinity and 4; that means, this change can occur anywhere in the past history. So if you follow this x value simulation it can be shown that the problem that you are facing that we will not be facing anymore right.

Let say you are applying 0 here you are applying, say one here. And you are saying that previously it was x; that means this condition was true. This transition can happen anywhere in the past not exactly at 0. So after delay 2 this 0 will be shifted to 2, this 0 will be shifted to 1 this 0 will be shifted to 4. This has a delay of 1. So this if we just look at the property of this x, this will shift to 3, because at 3 definitely both of them are at 0. After delay 1 before you do not know both maybe at 1. That is why it is x and finally, here 4 you are going up 3 you are going down so at 4 definitely you are going down.

So, here you can easily see this simulation will tell you that only after 4 it is getting stabilized unlike the earlier case where you are saying that in output you are getting a constant 0. So if you are using x value simulation you can get some accuracy in this calculation right.

(Refer Slide Time: 21:05)



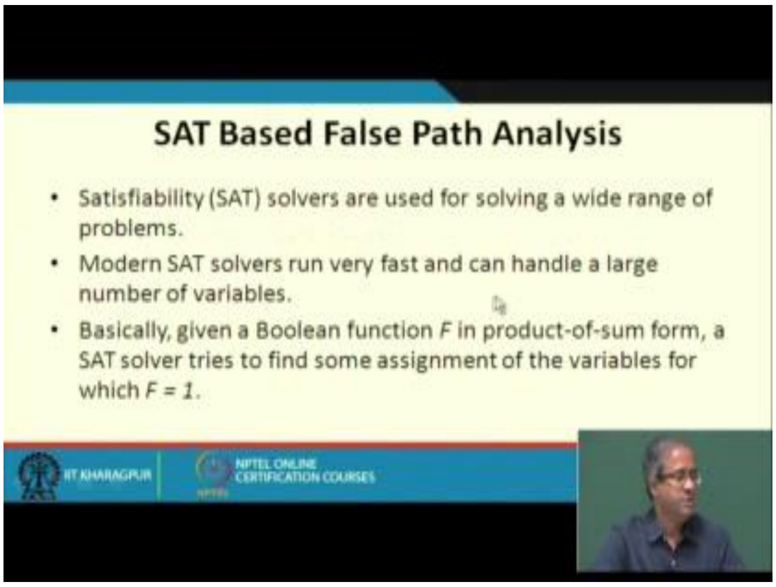
What we just saw ...

- Timed 3-valued (0,1,X) simulation
  - called X-valued simulation.
- Monotone speedup property is ***satisfied***.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, this is referred to as timed 3 valued simulations where instead of 0 and 1 you have an additional state called x, whose interpretation we have already seen here. It means that the signal transmission can take place anywhere before this time, not exactly at this time right. So with this modification the monotone speedup property is satisfied, this is the good thing.

(Refer Slide Time: 21:37)



SAT Based False Path Analysis

- Satisfiability (SAT) solvers are used for solving a wide range of problems.
- Modern SAT solvers run very fast and can handle a large number of variables.
- Basically, given a Boolean function *F* in product-of-sum form, a SAT solver tries to find some assignment of the variables for which *F = 1*.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

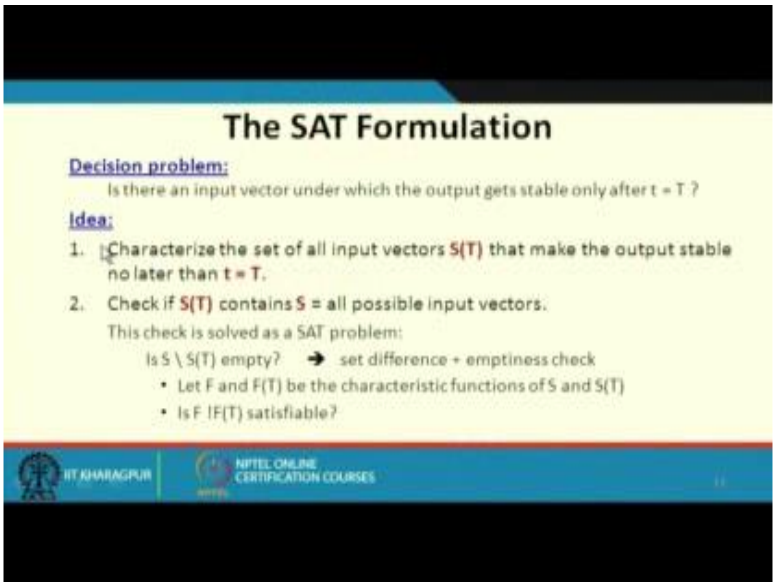
So, now we look at another approach to detecting or identifying false path that is based on using a SAT solver. SAT stands for satisfiability. Satisfiability problem is very wellknown problems in switching circuits, where given a Boolean function in some special form say for example, the product of some forms, with SAT solver will be trying to find out whether there exists some combination of the input variables for which your given function is equal to 1. If so it will also tell you for what combination of the inputs the function is 1. That is the task of the SAT solver.

So, SAT based false path analysis what it tries to do is that, now the thing is that SAT solver is a very general kind of a tool it can be used to solve a wide range of problems. And the SAT solvers that are available today many of them are freely available. They run very fast and then can handle quite large number of variables. Thousands of variables they can handle. Now this is what I mentioned that a Boolean function is given in product of some form. And a SAT solver will try to find out some values of the variables for which f equal to 1.

So, let us see how the false path analysis problem can be mapped into a product of sum expression and you can use a SAT solver to solve it right. So the SAT formulation goes like this. This is the most important step whenever you are using SAT to solve a problem you have to map it properly. The decision problem is here we are trying to ask this question is there any input vector for which the output gets stable only after t equal to capital T. Like here it is something like this suppose we expect that capital T is our time budget with which everything should get stable.

Now, we are asking this SAT solver can there be some input combinations for which the delay can exceed capitality. This is where you are posing the problem, right.

(Refer Slide Time: 24:03)



The SAT Formulation

Decision problem:

Is there an input vector under which the output gets stable only after t = T ?

Idea:

1. Characterize the set of all input vectors **S(T)** that make the output stable no later than **t = T**.
2. Check if **S(T)** contains **S** = all possible input vectors.

   This check is solved as a SAT problem:

   Is S \ S(T) empty? ➔ set difference + emptiness check
   - Let F and F(T) be the characteristic functions of S and S(T)
   - Is F !F(T) satisfiable?

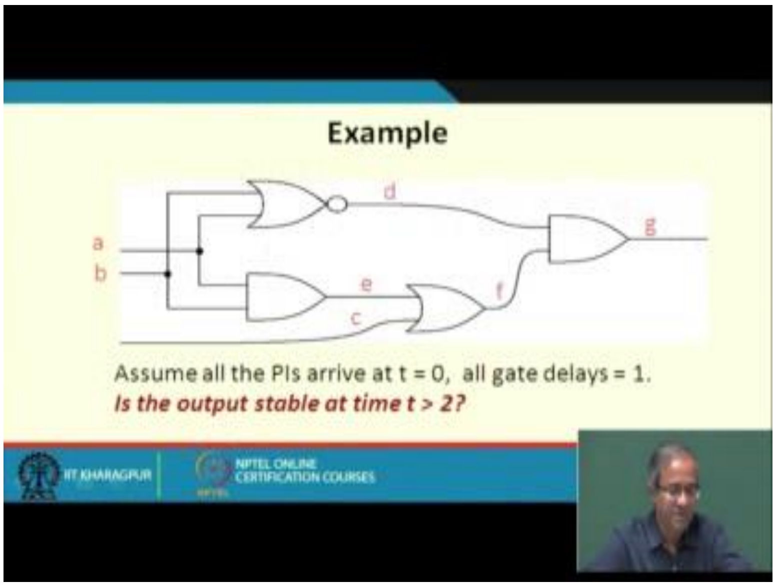
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, this is one possible formulation there can be other formulations also you can try out. So here what we are doing we are trying to characterize the set of all input vectors S capital T that make the outputs stable means earlier or at T equal to T no later than so ST is a notation it means this denotes all vectors for which the signal becomes stable within time T. And capital S denotes all possible input vectors for which the output becomes true or something.

Now, the SAT problem the question you ask is that, whether this set difference S minus S capital T is it empty? See this should be equal. So for all possible test vectors so you should have this property, your output should be stable within ST. So if you do a set difference S minus ST this set should be empty. So you take a set difference and check for emptiness, but if you say this is not empty this means this is not satisfied; that means, there is a problem here. There is a timing violation.

So, suppose corresponding to S and ST the corresponding Boolean functions are referred to as is F and FT. So the expression that you are giving to the SAT solver is F and not FT, is this product satisfied because set difference means this is true and this is false. This is true and this is false. Let us illustrate it with an example.

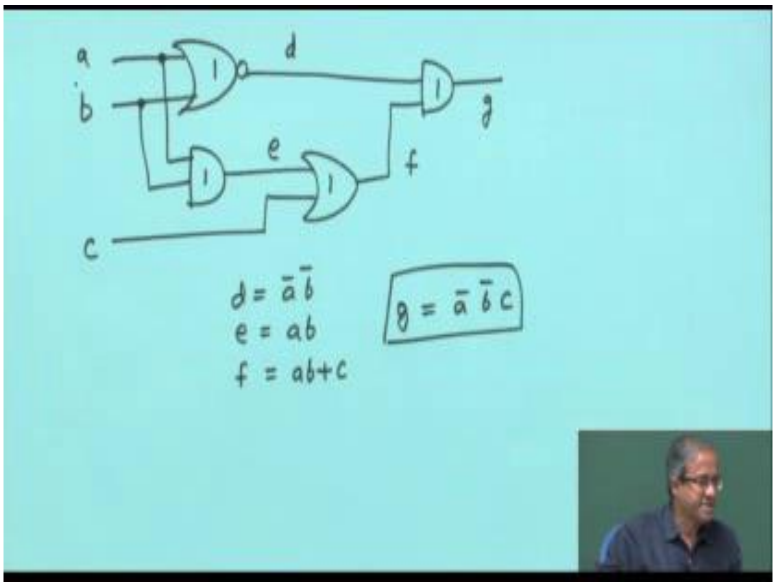
(Refer Slide Time: 25:56)


Example

Assume all the PIs arrive at t = 0, all gate delays = 1.

Is the output stable at time t > 2?

(Refer Slide Time: 26:13)



$d = \bar{a}\bar{b}$

$e = ab$

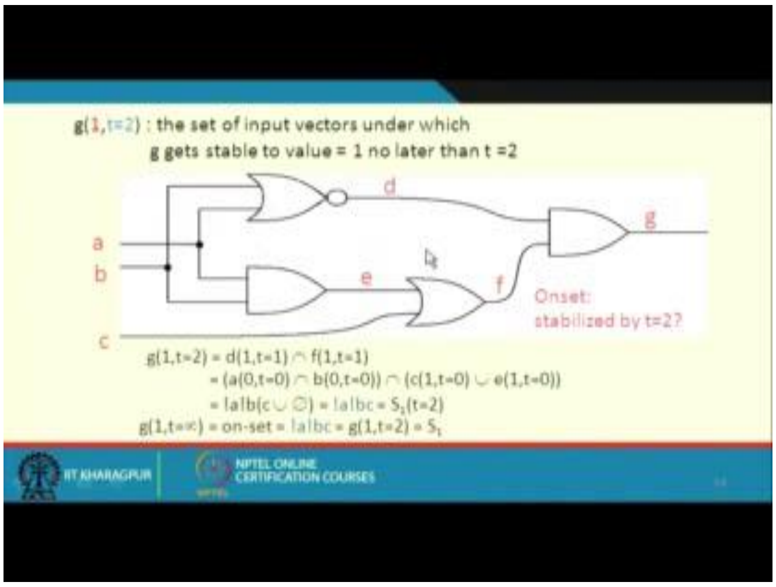
$f = ab + c$

$\boxed{g = \bar{a}\bar{b}c}$

So, a very simple example here, we consider the circuit. So let us just work out he circuit once more. So we have a circuit like this. Essentially we have like this a and b are applied to both these inputs and we have an OR gate, c is applied here then you have a AND gate this is applied here. So this is g these are the delays of the gates so I will assume to be 1. Let us call it d, let us call it e, let us call it f. So if we just calculate the output expression your d is a bar b bar, this already I have just mentioned earlier e is ab, f equal to ab or c, and therefore, g becomes and of d and e, d and f, so it becomes a bar b bar c. This is what g is.

Now, in this problem in this example, here we assume that all the primary inputs are arriving at time t equal to 0. So here we assume that all primary inputs arriving at t equal to 0, all gate delays are 1. So the question we are ask asking is the output stable at time t greater than 2, because 2 is our expected deadline. So here our timing analyzer told us that the delay should be 2, so let us see whether 2 is the correct figure or not.

(Refer Slide Time: 28:00)



g(1,t=2) : the set of input vectors under which g gets stable to value = 1 no later than t =2

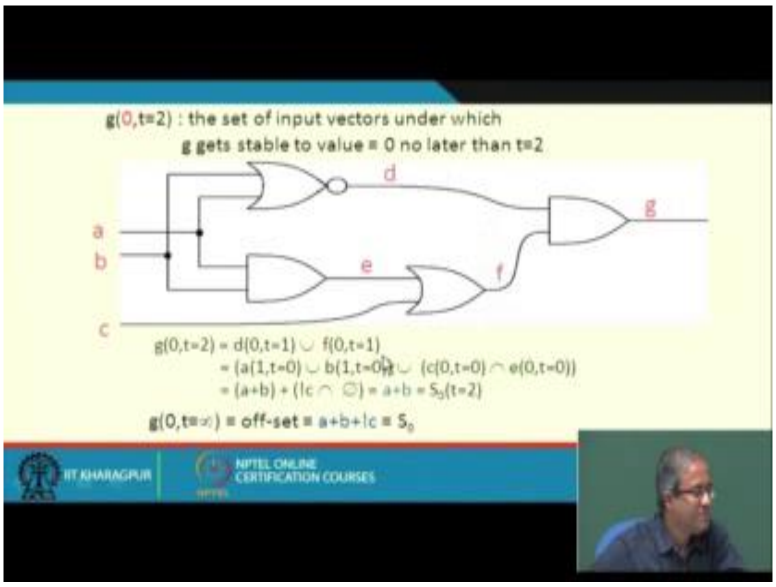
$g(1,t=2) = d(1,t=1) \cap f(1,t=1)$
$= (a(0,t=0) \cap b(0,t=0)) \cap (c(1,t=0) \cup e(1,t=0))$
$= !a!b(c \cup \emptyset) = !a!bc = S_1(t=2)$
$g(1,t=\infty) = \text{on-set} = !a!bc = g(1,t=2) = S_1$

So, we are just showing you with this example that how the SAT formulation is actually done. So how you do it map this into a Boolean expression. So we use a notation like this G1 comma t equal to 2. This denotes the set of input vectors under which the output g gets stable to a value 1 no later than time t equal to 2.

You see G1 t equal to 2, so the output is supposed to be 1. So when it will be 1 when this is a AND gate both d and f should be 1. When just one time you need before the delay of this is 1. So this will be equal to d 1 comma t equal to 1 and that is intersection f 1 t equal to 1 because time is 1 less. So if d equal to 1 means what both the inputs of these not gates should be 0. And 0 so it is a 0 delay is again 1 now time will be 0 and b equal to 0, both together. And f equal to 1 mean there is an OR gate so either e will be 1 or c will be 1. So c will be 1 or e will be 1. So if you just now calculate a 0 t 0 means what; that means, at time t equal to 0 I have all the inputs are coming. You are saying that a is 0, I write it as not a at time equal to 0, b equal to 0 I write it as not b, and here at time t equal to 0 c equal to 1 that is only c. Intersection at time t equal to 0 e equal to 1 you cannot say e at time t equal to 0 it is undefined. So this is empty phi so this is only c a bar b bar c. So let us call it s 1 t equal to 2.

Well here and if I just assume t equal to infinity; that means, the correct logic function this already we have seen that and the function that is realized is a bar b bar c. So this is defined as the onset for t equal to infinity, so delay is not considered. So we see that these 2 are equal, so if you subtract these 2 it becomes null. So for the onset there is no problem.

(Refer Slide Time: 30:25)



g(0,t=2) : the set of input vectors under which g gets stable to value = 0 no later than t=2

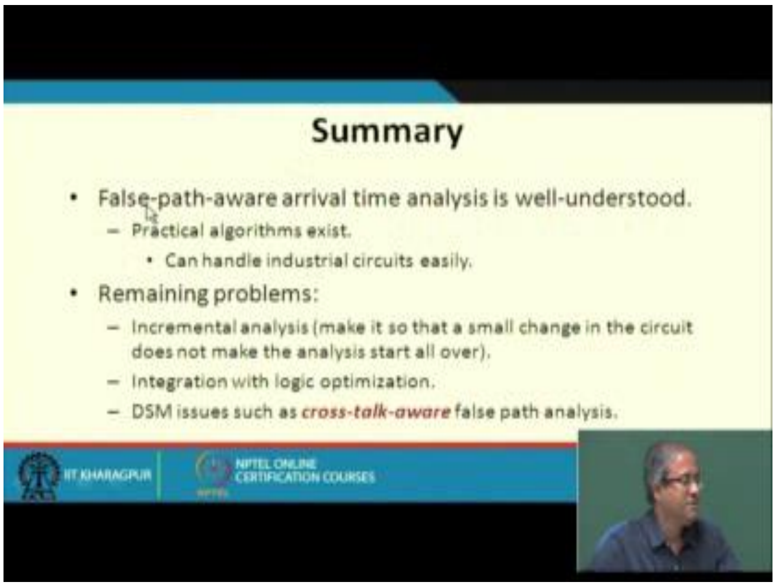
$$g(0,t=2) = d(0,t=1) \cup f(0,t=1)$$
$$= (a(1,t=0) \cup b(1,t=0)) \cup (c(0,t=0) \cap e(0,t=0))$$
$$= (a+b) + (!c \cap \emptyset) = a+b = S_0(t=2)$$

$$g(0,t=\infty) = \text{off-set} = a+b+!c = S_0$$

But let us look at the offset. G 0 t equal to 2, so when g equal to 0 means anyone of d and f can be 0. So d equal to 0 at t equal to 1 or f equal to 0 if f equal to 0, means both e and c are 0 f 0 means both c and e are 0 and d 0 means anyone of them can be 1 a is 1 or b is 1, so a is 1 you now write down this means a time t equal to 0 a equal to and; that means, a or means plus or notation this is b c 0 t 0 is c bar intersection phi this becomes 0 only a plus b, but offset this g 0 t equal to infinity already you have calculated should be a plus b plus not c. So if you take a set subtraction these are not the same.

So, to summarize, you have got this as a plus b you have got this b. If you take a set difference; that means, I said this not this then you get a bar b bar c bar; that means, there is one input combination 0 0 0 for which delay equal to 2 is not satisfiable. The delay will be equal to g. So this is roughly the idea how this method works.

(Refer Slide Time: 31:56)



Summary

- False-path-aware arrival time analysis is well-understood.
  - Practical algorithms exist.
    - Can handle industrial circuits easily.
- Remaining problems:
  - Incremental analysis (make it so that a small change in the circuit does not make the analysis start all over).
  - Integration with logic optimization.
  - DSM issues such as *cross-talk-aware* false path analysis.

To summarize, false path aware analysis is important. And this is quite well under understood many practical algorithms exist. So we just mentioned a couple of them. They can handle very large circuits quite easily because their computation and complexity is not so high. So the problems that need to be looked at is that incremental analysis if we make small changes in the circuit do you have to do the analysis all over or we can do some correct incremental analysis. And this logic optimization also we usually do a lot of optimization, so when you do optimization can you integrate the same process within that. And again for deep sub-micron issues we need to consider the cross talk and noise issues, some of these we shall be discussing later.

So with this we come to the end of this lecture.

Thank you.