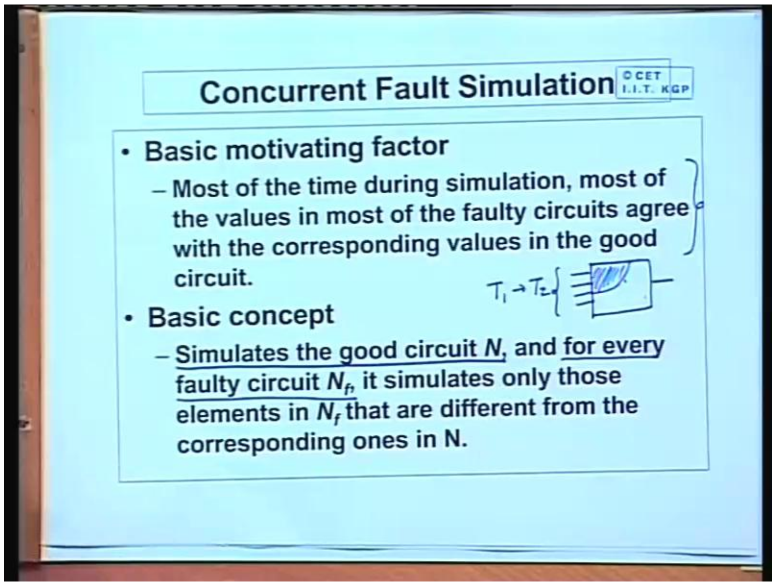
Electronic Design Automation

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

Lecture No #32
Testing - Part III

You know last class we were talking about faults simulation. And if we recall, we had talked about two algorithms for faults simulation namely the parallel faults simulation and the detective faults simulation. So today we will be talking about another faults simulation algorithm namely the concurrent faults simulation which is somewhat simulate to the detective faults simulation.

(Refer Slide Time: 01:34)



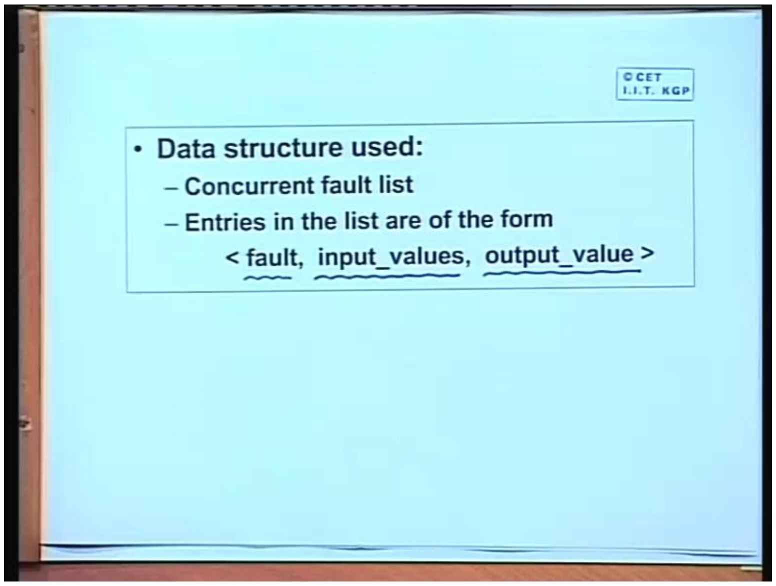
Concurrent Fault Simulation

- Basic motivating factor
  - Most of the time during simulation, most of the values in most of the faulty circuits agree with the corresponding values in the good circuit.
- Basic concept
  - Simulates the good circuit *N*, and for every faulty circuit N_f, it simulates only those elements in N_f that are different from the corresponding ones in N.

©CET I.I.T. KGP
$T_1 \rightarrow T_2$

In the sense that here also we are maintaining a fault list. And we have some mechanism of propagating the fault list. But they are some differences as compare to detective faults simulation. Well in the method of concurrent fault simulation the basic concept is that we try to take advantage of the fact that most of the time during simulation say given a circuit with certain number of inputs we apply a test vector. Possibly we can inject a fault in the circuit and similar to respect to that. Then from one test vector T_1 we go to another test vector T_2. But actually what happens in practice is that if you look at the circuit only a small portion of the circuit will have some changes with respect to the logic values of the lines. The remaining portion of the circuit will not be affected in general okay. So the idea we are trying to take advantage of is that most of the time during simulation most of the values in most of the faulty circuits agree with the corresponding values in the good circuit. So with respect to true faults maybe the values are differ in those local portions of the circuit for the faults of the injected the remaining portion the logic values maybe all same okay. So we try to take advantage of this fact and what we do we simulate the good circuit N. Simulate means you do a true value simulation we find out the logic values. And for every faulty circuit we call it N_f if F is a fault we call the corresponding faulty circuit N_f. Now for the faulty circuit N_f we do not simulate the entire circuit rather we only simulate those gates or elements in that circuits for which some input output values are different as compare to the fault free once okay. This is the basic concept. Now in the method of detective false simulation what we did? We actually compose some fault list which consists of the faults which are actually getting detected. And there was a mechanism of propagating this fault list from the gate outputs to the next gate input from that to the gate output again using some simple rules based on set theory. But in contract in concurrent faults simulation we are actually simulating the gate. We are evaluating the gate and the fault less will not only consists of the fault but also the input output values. So what we actually mean is that the basic data structure we are trying to use here is called a concurrent fault list.

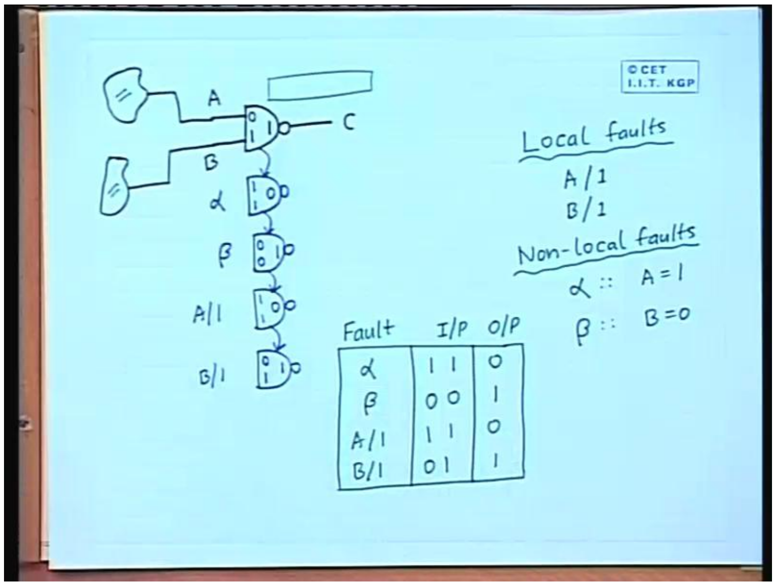
(Refer Slide Time: 04:39)



- **Data structure used:**
  - Concurrent fault list
  - Entries in the list are of the form
    < fault, input_values, output_value >

Now in the concurrent fault list the entries are typically of the form. Of course the fault the input values of the corresponding gate where the fault presides and also the corresponding output value. So in addition to the fault where also storing information about the input and output values, now let us take an example to illustrate this concept.

(Refer Slide Time: 05:08)



Fault	I/P	O/P
α	1 1	0
β	0 0	1
A/1	1 1	0
B/1	0 1	1

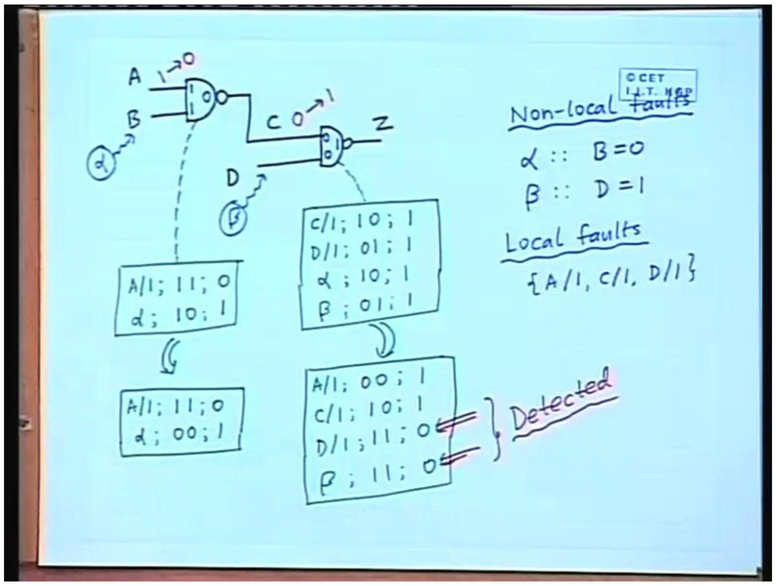
Suppose we have a gate say a NAND gate, two input NAND gate which is being fed by some sub circuit well. So we are not concerned about which sub circuit. So we are concentrating on this particular NAND gate. Now let us see that the two inputs of this NAND gates are A and B and the output is C. Now presently in response to the applied input also suppose that the input values A and B are zero and one and the corresponding output value is one I am showing this like this. And also we consider that there are four faults that we are consider because we are assuming that after fault collapsing some of the faults are getting removed. But just for the sake of in the example we assume that with respect to the gate there are two local faults we are considering and the local faults are A stuck at one and B stuck at one.

And also there are some non-local faults. These non-local faults are say, we just give some names alpha and beta. So actually alpha beta are faults which are taking place somewhere inside this sub circuits which is feeding A and B. The net effect is that is the fault alpha takes place the value of A becomes one if the fault beta takes place the value of B becomes zero. So you remember this for the timing. Now these are the four faults okay. Now with respect to this four faults see we have simulated this circuit with the true value 0 1 1 and the input values. Now if this faults takes place one of the other, let us show it like this. Suppose the fault alpha takes place alpha will make the input value A to one. So the input output value will now become 1, and 0. Similarly if the fault beta takes place B will be 0, so it will be 0, 0 and 1.

And if A stuck at one, fault is there. Then it again be 1, 1 and 0. And if the B stuck at one, fault is there, it will again become 0 1 it will remain 1 and 1. Now this is typically shown in text books like this that this is the original gate and these are the gates in the faulty circuits of relevance that all those faulty circuits for which there is some change. Okay. Or the local faults the local faults of course will have to consider and the other faults for which there some change in either A or B okay. We show it like this. But actually in your simulator when you are actually simulating this you will be storing this in this fault table. The table will look like this the table will contain column fault then the input values then the output value. This will be the four edges of the table. And you will be storing it like this fault alpha input is 1 1 output is 0. Fault beta 0 0 1.

A stuck at 1 1 1 0. B is stuck at 1 0 1 1. So actually although we express is diagrammatically like this. But during simulation with each gate there will be a fault list and the fault list will look like this it will be a table okay. Now let us take another example to show that how this fault list can propagate from one gate to another because its fault list is it is not a static thing. We are creating the fault list for one applied vector. Now if the input vector change is we will again up to re-compute the fault list. And again since this is an activity revenge simulation process. We will not be simulating the whole of the circuit, but rather we would be simulating only those portions where there changes exactly like to the method detective faults simulation. So let see how we do this. Here we can, let us take a very simple example for the sake of illustration.

(Refer Slide Time: 10:06)



Say you have a NAND gates with the inputs A and B. The output of this is feeding another NAND gate whose other in this line lets call it C and this input is D. The output is say Z okay. Now at any given point and time suppose the input applied logic values are 1, 1, so the output will be 0 and D is also 0. So 0, 0 and 1 okay. And also consider that here the faults that we are considering of course will be considering the local faults. Local faults means faults at lines A, B, C and D. But in additional in addition to this will also be considering some non local faults just for the sake of illustration that in the fault list also the non-local faults may appear. Here also we consider that there is fault alpha whose effect is to make the line B equal to 0.

There is another fault beta whose effect will be to make the line D equal to one. So alpha is a fault which is affecting B. And beta is a fault which is affecting D okay fine. Now under this condition, let us see what the fault list would look like. So I am showing the short list in the form of tables. For this gate the fault list will look like this. We just express it like this well. Well again we are we just also consider that after fault collapsing some of the local faults have been removed. So the local faults that we have to look at are A stuck at one, C stuck at one, D stuck at one, these okay. So for this the fault list will consist of A stuck at one, this is one local fault. Why write it like this? This is the fault. This will be the corresponding input because it will remain 1.

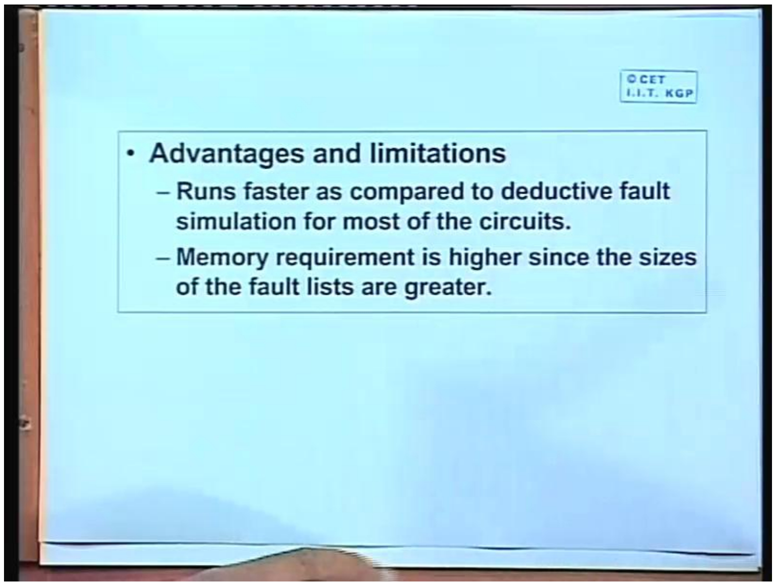
This will be the corresponding the output. Similarly alpha is a fault which can affect this gate which will make D equal to 0. So alpha now the input will be 1 0 the output will be 1. So this is the fault list which is associated with the first gate. Similarly the fault list that will be associated with the second gate it will look like this. The first local fault will be C stuck at one, this will be this will be this will be make the input 1 0 output will be one. Then D stuck at one 0 1 1. Then the fault alpha well here this fault alpha when your calculating you will have two makes use of the fault list of the predecessor gate. Because you know that is the fault alpha takes place in the output is one. So this one will come here it will be 1 0. Similarly the fault beta will make it 0 1 this will also be 1.

So if Z is the primary output you observe one thing that since the faults free output was one and in presence of all these faults the output is also one. So for the presently applied input vector none of these faults are detectable okay fine. Now suppose we change the inputs. Inputs we change in such a way say the value of A which was one it now becomes zero we change it from one to zero. So as you changes from one to zero the output of this gate C will become one. So this value which is 0 the true value. This will now become one okay. So after this change there will be some modification in the fault list. So let us see how this will get modified. This short list will get modified as follows.

The modified fault list will be A stuck at one this will again remain 1 1 because A stuck at 1 0, this will not change. But this alpha will change 1 0, it will now become 0 0. Similarly for this fault list the modified one will be, A stuck at one. Well now A stuck at one will come because for A stuck at one fault the output is supposed to be 0 which is different from the current applied logic value. So this is a logic event okay. The value of the line in presence of this fault is A stuck at one is different from the true logic value which is now equal to one. So that is why A stuck at one fault will also appear in the fault list now. So for this, this will become 0, this will also be 0. Because A stuck at 1 will make this lines 0 okay, so 0 0 1.

Similarly C stuck at one will remain 1 0 1. D stuck at 1 will also 1 1, output will be 0 and similarly beta will make it 1 1 and 0. So now you see that the output was still 1. But there are two faults now in this list for which the output is 0. So these are the two faults which we will get detected by this second in test vector. Such this is very small example I will have illustrated. But this procedure can be apply to much bigger circuit as well. And as you can see this method is very similar as compare to detective faults simulation there also we propagate some fault lists. But in detective fault simulation we do not simulate the gates when you injective faults rather we try to compute the fault list using some security proves okay. So for concurrent faults simulation if we want to summarize.

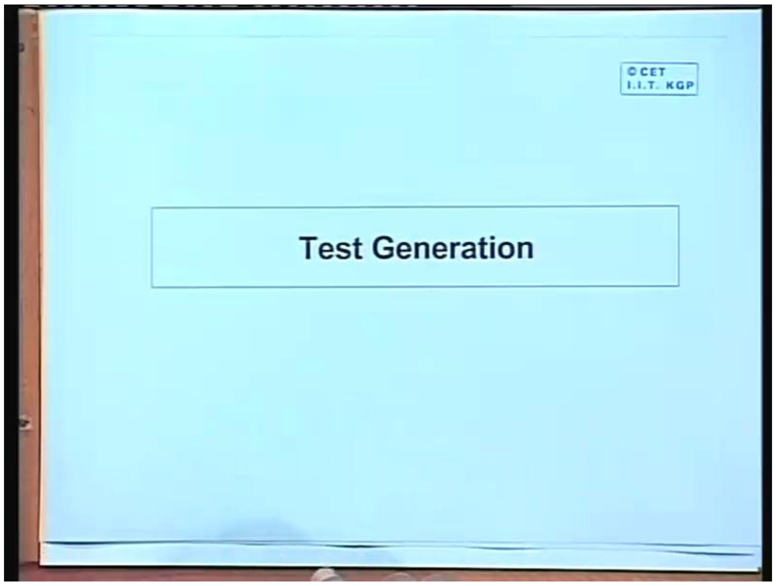
(Refer Slide Time: 17:13)


©CET
I.I.T. KGP

- Advantages and limitations
  - Runs faster as compared to deductive fault simulation for most of the circuits.
  - Memory requirement is higher since the sizes of the fault lists are greater.

Well it runs faster in general as compared to deductive fault simulation. Because in deductive fault simulation you have to carry out some security average operation based on strings which are typically slow. But the problem is that since we have to store bigger fault list here memory requirement is higher. And of course it is also unpredictable just like detective faults simulation okay. So we have talked about some of the algorithms for fault simulation. So just to summarize once more faults simulation is used to assertion or verify the quality of a set of given test vectors. We want to find out what are the faults that I getting detected. What are the faults that are not being detected we have to detect them. And related that you get is called the fault coverage the percentage of faults that I getting detected okay. So now we very briefly look at the problem of test generation.

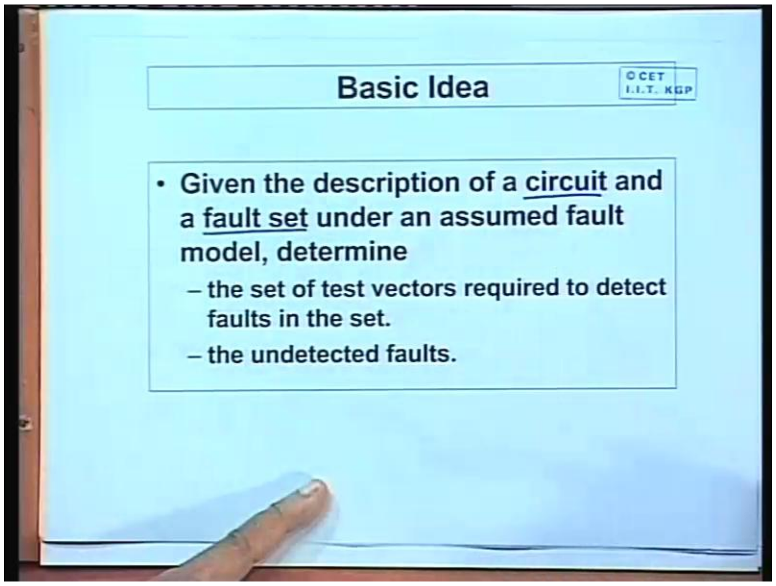
(Refer Slide Time: 18:22)


©CET I.I.T. KGP

Test Generation

Because the test generation algorithms which are used are fairly complicated. And where not going to discuss the algorithms in detail. But rather we try to give you the basic idea okay. The process of test generation is basically this.

(Refer Slide Time: 18:42)

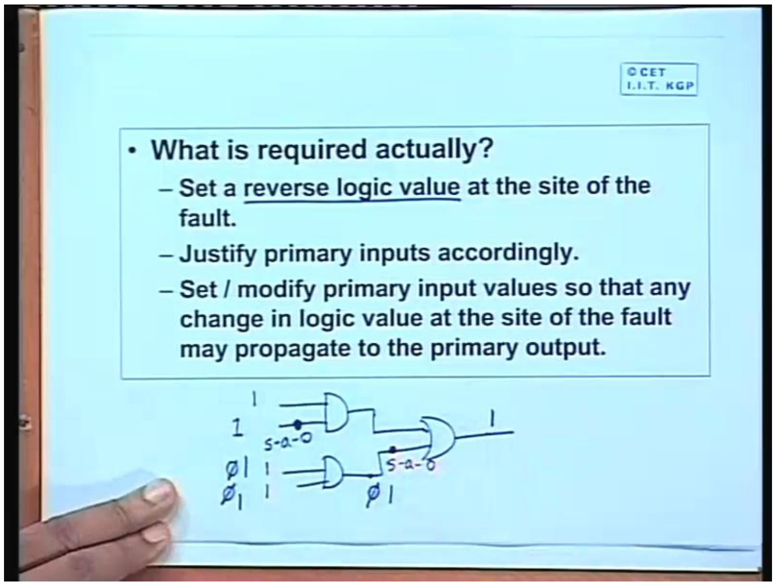

Basic Idea

©CET I.I.T. KGP

- Given the description of a circuit and a fault set under an assumed fault model, determine
  - the set of test vectors required to detect faults in the set.
  - the undetected faults.

Given a circuit a net list and a fault list. So you have a circuit with you have a fault set. This fault set can be collapsed fault set after this fault equivalent and dominants collapsing, you get a reduced fault set. Of course this fault set have been obtained under an assumed fault model typically this stuck at fault model. Now the test generation process could help you to determine the set of test vectors which are required to detect the faults in this given set. Well it can also tell you that for some of the faults it is unable to find a test vector. So it will also give you the list of undetected faults. Now let us try to see for this what are the things that are actually required okay.

(Refer Slide Time: 19:38)


© CET
I.I.T. KGP

- What is required actually?
  - Set a reverse logic value at the site of the fault.
  - Justify primary inputs accordingly.
  - Set / modify primary input values so that any change in logic value at the site of the fault may propagate to the primary output.

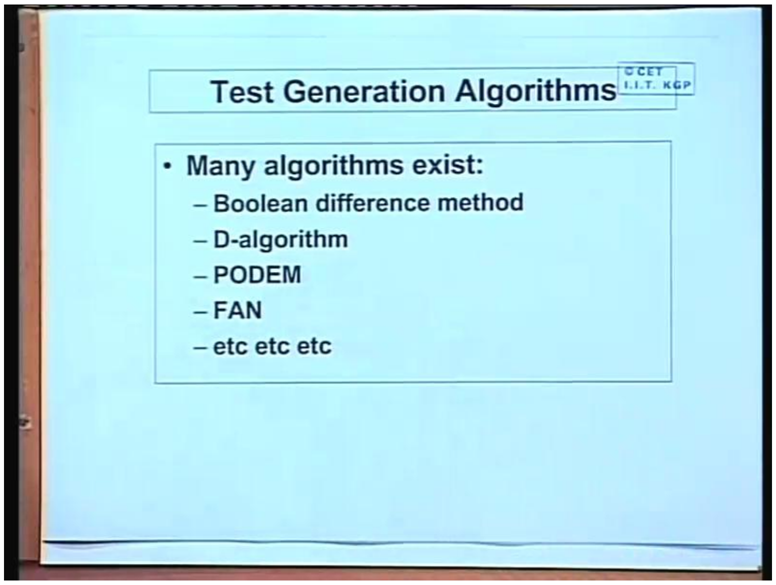
1
1
s-a-0
1
ø1 1
ø1 1
s-a-0
ø1

The first thing is to set a reverse logic value at the site of the fault like what I mean to say. Let us take a simple example. Instead of simple circuit like this, suppose we want to detect or you want to find a test vector to detect the stuck at zero on this particular line, now the first requirement for detecting this faults is that you will have to excite this particular line with reverse logic value. Now since this is the primary input you can straight away apply logic one to this input. And justify the primary inputs is required if this is an internal line for example if we are considering a fault say here stuck at zero fault here.

Now in order to apply the reverse logic value one here we will have to trace back and find out that well we have to apply 1 1 to this AND gate this is called justification of the primary inputs okay. So in order to apply the reverse logic value we have to trace back if necessary and we have to apply the corresponding input values. Not only that, we will also have to set some values for the primary input so that any change on that line we will get propagated to the output. Like for example if you again consider this stuck at zero fault models, well I am applying logic one here fine.

But suppose to the other AND gate I had applied 0 0, then this output gate will be 0, then any change out here I have to apply a one on the other input also. So that any change out here will propagate here and since this input is 0, this change will also go to the output and you can detect. But suppose I have applied 0 1 here, not 0 1 1 1 1 1. So since this an AND gate the output will be one and irrespective of what is happening in this gate the output of this OR gate will be one and this fault cannot be detected. So we will have to apply suitable logic values to the gates. So that the changes are propagated to the output so we will just see with an example how this is done. In fact in practice there are number of test generation algorithms which have been developed. These are some of the classical once.

(Refer Slide Time: 22:24)



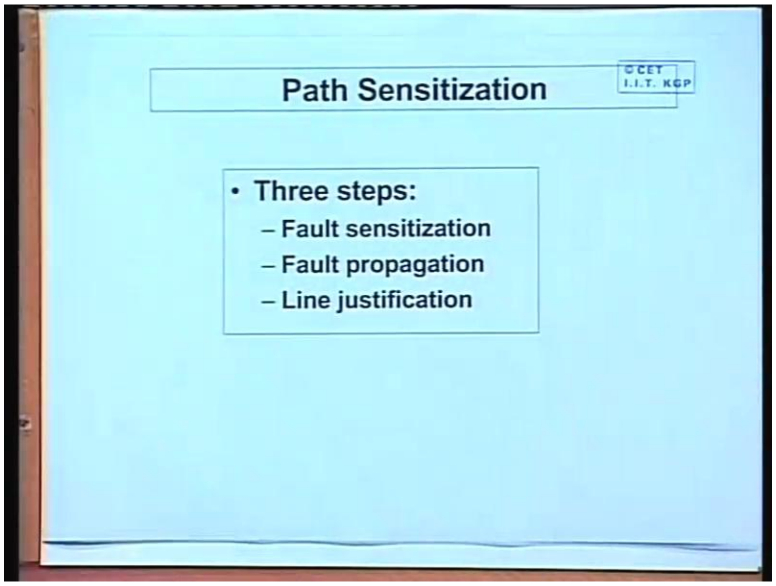
Test Generation Algorithms

© CET I.I.T. KGP

- Many algorithms exist:
  - Boolean difference method
  - D-algorithm
  - PODEM
  - FAN
  - etc etc etc

But after that there are many modifications in the improvement which have taken place. Boolean difference method D algorithm PODEM FAN and so there are number of such algorithms. Now all this algorithms will other than Boolean difference, this is an algebraic method. This all this methods try to do or try to carry out a process which is called path sensitization. So let us try to understand what is mean by path sensitization.

(Refer Slide Time: 22:57)



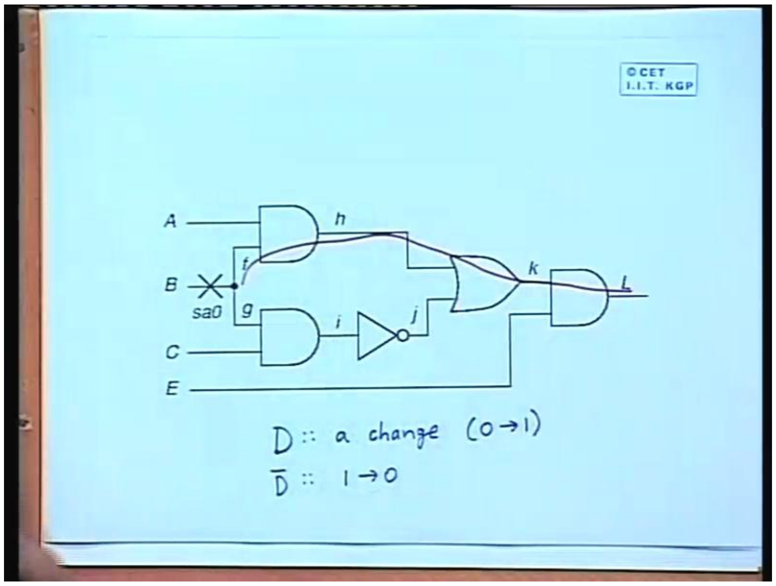
Path Sensitization

© CET I.I.T. KGP

- Three steps:
  - Fault sensitization
  - Fault propagation
  - Line justification

So as we had illustrated the small example that we need to do or carry out three steps. Fault sensitization means applying the reverse logic value at the site of the fault. Fault propagation means we have to find out a path through which the fault will be propagate to the primary output. And line justification means in order to do that we will have to apply some suitable logic values to the other lines of the circuit. So we will have to do that systematically okay. So let us take an example to illustrate this how we do this okay. Let us take this example.

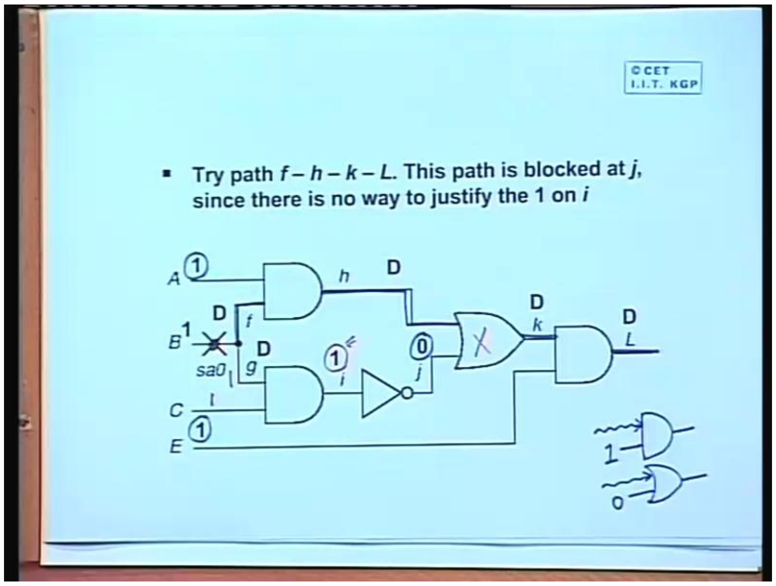
(Refer Slide Time: 23:41)



This is a small yet an example with some complexity because there is a fan out. Just you observe one thing. This circuit has a structure which in terms of testing is consider to be a complex structure. It has a fan out and also it has a reconvert that means a fan out branches are finally margin into a gate down the line in this OR gate. So this is called ready conversion fan out okay. So in this example these are the primary inputs A, B, C, D, F, G at the fan out branches h, I, j, k, l, these are the lines. And suppose we are trying to detect a fault stuck at zero on the line B. Now it be using some notation well what we want is that, that any change on the line B should be observable as a corresponding change on the primary output. This is stuck at zero if apply a one on B the output value which was there for the good circuit could change. So just by observing the output I could say that the fault B has occur or not okay.

So notational we denote the symbol D as a change well D it does not matter with its 0 to 1 or 1 to 0. D indicates some change well as a matter of convenient. Let us say any 0 to one transition on a line it can be denoted by D; D is a change. So D bar will denote the corresponding reverse change. So D or D bar means that some changes going on that particular line. So when we talk about a change we are actually talking about change with respect to the good circuit. For the good circuit the line had some value and when the fault occurs the line is having some other value okay. So this we denote by D or D prime fine. Now in this circuit we want to detect this fault. Let us try the first alternative the first alternative is that we try to see that if we can propagate the fault through path F h k l. Let us try this path whether you can propagate this path this fault. So, this path okay. Let us see it.

(Refer Slide Time: 26:31)



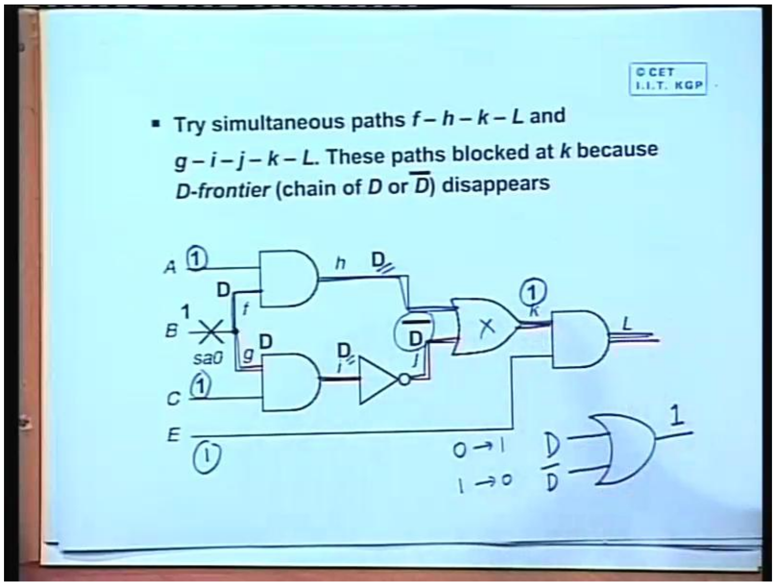
- Try path *f* – *h* – *k* – *L*. This path is blocked at *j*, since there is no way to justify the 1 on *i*

So we are trying to propagate it through this path f, h, k and l okay. In order to propagate this, let us see the condition just you observe if it is an AND gate and if you want to propagate a change on one of the inputs the other input value must be set to one. Because if it is set to zero the output will be permanently zero. Similarly if it is an OR gate, if you want to propagate a change on one input, the other input must be set to 0. Similarly for NAND and NOR okay. So these are the rules which you have to keep in mind. Now we have to propagate this fault through this path. Since this is an AND gate I must applying one on the other input there is an OR gate down the line. So I must have a 0 on the other input.

There is an AND gate to in order propagate this I must have one here. So these are the requirements I must have 1 0 and 1 at the other inputs of the gates which are following in the path. But of these three A and E is not an issue at all. Because this is a primary input I can orders applying 1 1. But let us see if you can have zero and j this a NOT gate if you have want to have a 0 here. If you do a back tress this means that you read a one here. Now this is an AND gate if you need a one here you must have C 1 and also G 1 permanently. But this is the site of the fault if the fault is there then G will be zero and this line will no longer be one this will be 0. So in presents of the fault you cannot propagate the fault because in presence of the fault J will be equal to one and the fault will get blocked here the fault will not get propagated okay.

So this example shows that this line justification in order to propagate the fault may not be that easy a task it is a difficult it may need considerable searching across the different path in the circuit okay. So now let us try another alternative. Well instead of trying the path f, h, k, l, let us see if we can excite two paths together concurrently we can propagate the fault to the true paths. Now here you observe one thing, the change here we were indicating by D. So D through an AND gate there is no inversion 0 to 1 becomes 0 to 1 D, D and D. There are no inversions in between no NOT, NAND and NOR. So D does not become D bar. So any 0 to 1 change here we will reflected as a 0 to 1 change. Here 0 to 1, here and 0 to 1. Here they all remain D.

(Refer Slide Time: 29:51)

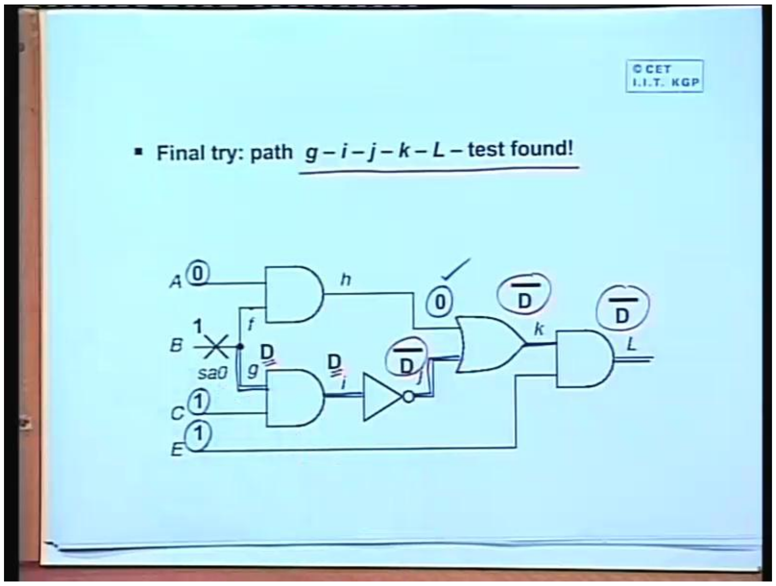

© CET I.I.T. KGP

- Try simultaneous paths *f – h – k – L* and *g – i – j – k – L*. These paths blocked at *k* because *D-frontier* (chain of *D* or $\overline{D}$) disappears

Now the next attempt we try the paths f, h, k, l and j, I, j, k, simultaneously. Which means the first path we try is this and the other path we try is this. So now let us see what is the implication. This is the site of the fault now if you want to excite both this paths okay. So first thing is that I have to apply a one here to this AND gate this AND gate we have to apply a one here again okay. Now this is stuck at 0, I have to apply a one here. So actually this point I am applying a D. So D on f D on g. Since you have applying one here and one here. So D will also come out here and here this, an inverter. So to the output of the inverter we get a D bar. Now you observe one thing you have an OR gate you have an OR gate one of its input is D the other input is D bar.

What does it mean? That this input is going from 0 to 1, the other input is going from 1 to 0, so in the first case the inputs are 0 1, other case it were 1 0. So in both cases the output will be one and output there will be no change output will be permanently at one. So in this case the two paths are mutually masking each other there cancelling of the effects with a neat result is that there fault is not getting propagated beyond this gate okay. There will be no change on k and hence no change on L right. So if you try to propagate the two paths together it is also not possible.

(Refer Slide Time: 31:56)

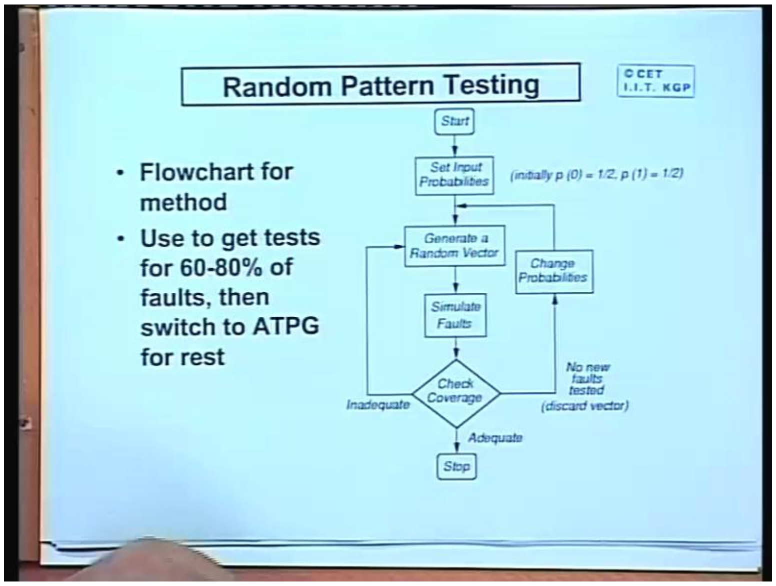


- Final try: path $\underline{g-i-j-k-L-\text{test found!}}$

Now let us look at the last alternative. Now will find that in fact we can find out a solution that we just look at propagating through this path single path the other path. Now you look at the gate switch encounter this, an AND gate. So I have up to apply one here this is an OR gate you have to apply a zero here AND gate you have to apply a one here. Now this an OR gate in order to apply zero here you back track but you cannot apply a zero here because you are affecting a change here. But the other input you can always apply a 0, this, an AND gate if you apply a 0, here this will always be 0. So this is possible. So now with respect to the D propagation here, it is D output of the AND is also D output of the NOT gate is D bar OR gate other input of0. So the output will also be D bar and for input at one is also D bar. So here we have at last found a test
0 1 1 1 is the corresponding test vector for detecting the fault B stuck at 0.

So you see that this is a very small example yet we have to explore. So many paths and possibilities before actually successfully finding a path. So this is the problem we will have to remember that in general when you have a circuit of much higher complexity much bigger circuit much larger number of gates doing this kind of deterministic test generation may sometimes we very expensive in terms of computation time. So people have explored other alternatives as well. Well, I am stating one alternative here but exactly how this is done? This we will be discussing later. This alternative is something called random pattern testing.

(Refer Slide Time: 34:07)

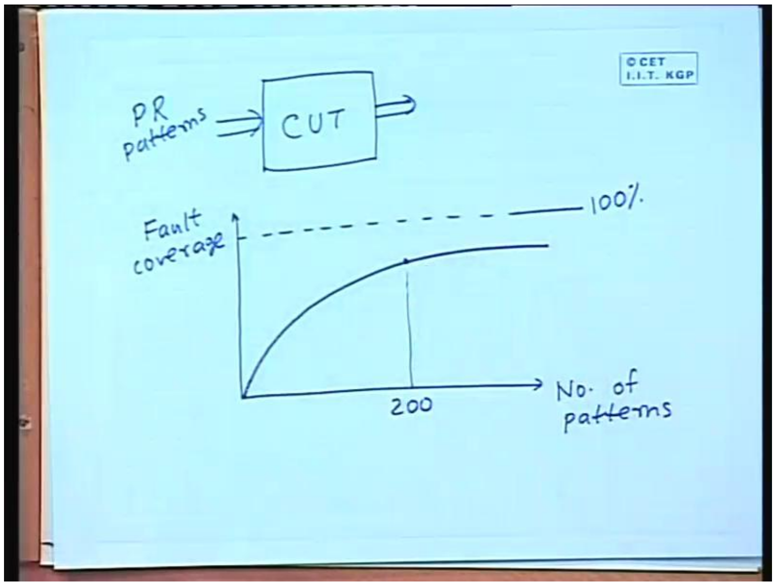


Random Pattern Testing

- Flowchart for method
- Use to get tests for 60-80% of faults, then switch to ATPG for rest

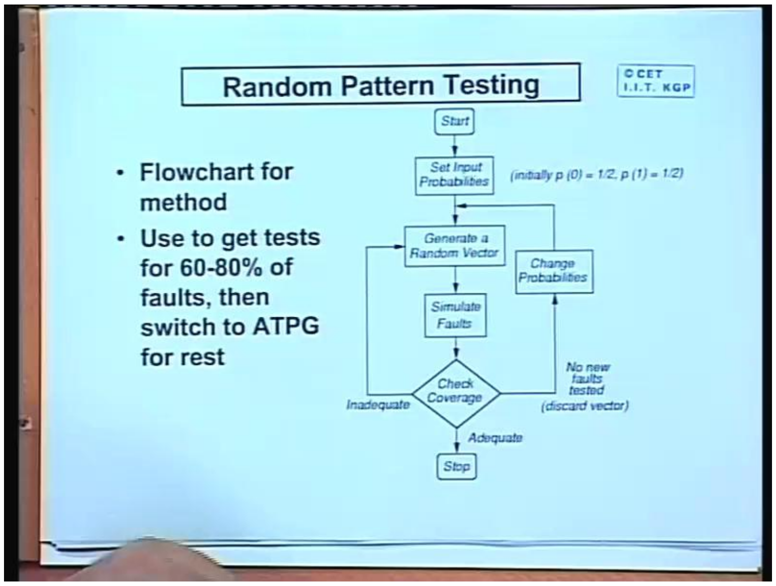
Well, actually it is not purely random you should call it pseudo random. Because its pattern side generated using some algorithm and the same pattern can be repeated. That is why it is not purely random it is pseudo random. And it is a random pattern testing the idea is that.

(Refer Slide Time: 34:29)



Well I am trying to first explain you with the help of our simple example. Suppose you have a circuit under test these are the inputs these are the outputs. Now instead of deterministic will be generating the test vectors to detect the faults using a test generating algorithm. It has been found that if you simply apply pseudo random patterns in the input blindly, apply random patterns. Then for most of the circuit a pattern like these holes, here if you plot number of patterns you apply. And here fault coverage then typically the curve looks like this. So this maybe the point of 100 percent. We will find that say for example you apply 200 patterns you get coverage of 85 percent or so. So what people do is that, they sometimes use pseudo patterns of the first few test vectors they do not use in deterministic algorithm. Just generate them and using faults simulation they find out what are the faults they are getting detected. Now after doing this for certain number of patterns they will find out that what are the faults that remain to be detected and finally they use it test zero to algorithm to detect those remaining faults okay fine.

(Refer Slide Time: 36:11)


Random Pattern Testing

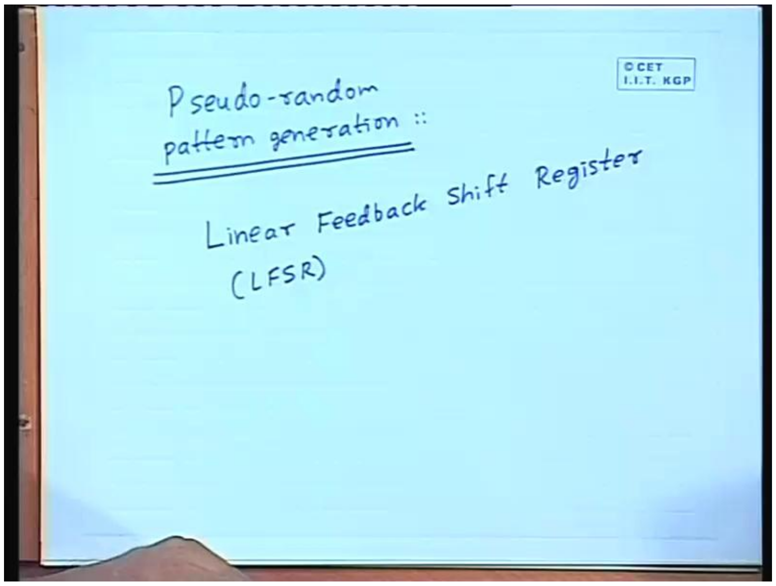
© CET I.I.T. KGP

- Flowchart for method
- Use to get tests for 60-80% of faults, then switch to ATPG for rest

So if we just look at the overall flow of this random pattern testing. Well this is an optional step set input probabilities if it is a pure pseudo random pattern the probability of a 0 or a 1 applying on the inputs or all “0.5”. These are the input probably initially there all half “0.5”. We generate a random vector simulate faults to find out how many faults are getting detected we check the coverage. If it is inadequate you go back. Well you either do this or you do this for a fix number of times okay. This is an optional step as I told you not all people do it. If you find that the fault coverage is not improving beyond a point you analyze the circuit and the faults are yet to be detected.

And change the input probabilities there are ways of changing probabilities we would talk about this later. And by changing this input probabilities we can actually generates zero random patterns for the probability of a 1 and 0 appearing or something other than “0.5” there are ways of doing that. So Pseudo Random testing is very interesting method because, well the first thing is that I have shown this plot, that you can simply apply zero random patterns and you can estimate the fault coverage.

(Refer Slide Time: 37:43)



Pseudo-random pattern generation ::

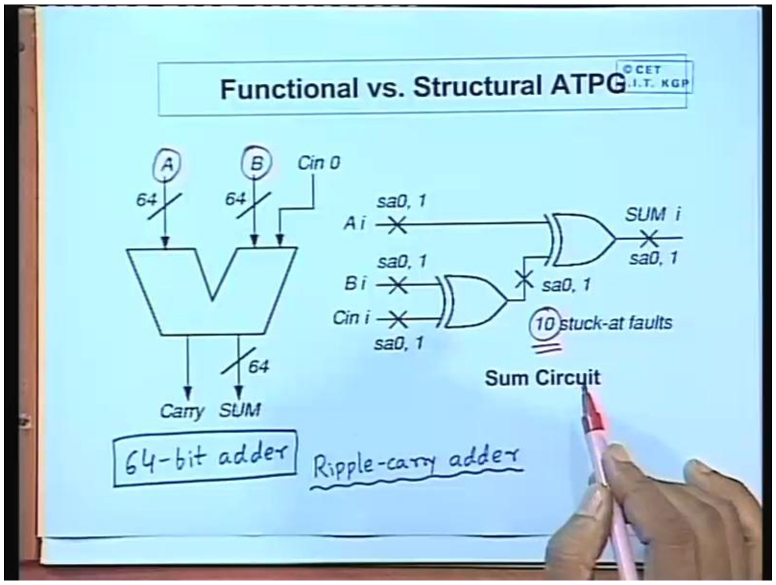
Linear Feedback Shift Register (LFSR)

Not only that there are some other advantages like pseudo random pattern generation. This can be done very efficiently in hardware. This is another big advantage. This is practically done using a simple hardware structure called linear feedback shift register or LFSR in short. We will see later that in how an LFSR are looks like what its properties. But actually thing is that you can you an LFSR simply to generate this random patterns and to detect the first few number of faults.

In fact first many numbers of faults. So the faults are remain to be detected could be detected later by using test generate algorithms okay. Now the example that we are taken earlier for test generation was for a gate level circuit. Now in fact most of the modern day test generators algorithms or tools they work for gate level circuit and some of them also work for mixed level circuits where some of the net list of part of the net list may be in terms of gates.

There may be some other blocks like multiplexer decoders ALU's and so on a mixed of those things. But very recently another school of thought of um emerged many companies or industries. They do not want to spent much time and effort for this test generation at the gate level and fault simulation. Rather they look at the functional blocks at high level and try to generate test for it. For example an adder, for example an ALU, a decoder, but there are some problems involve there I will take care I will take an example illustrate what are the problems. Because, if you generate test set of functional level at a high level you cannot have a good you can say, you cannot have a high confidence regarding fault coverage in terms of the in terms of the faults that is taking place actually inside that functional block. We are only talking about input output behavior. We will take at simple example into illustrate this problem for this is all about.

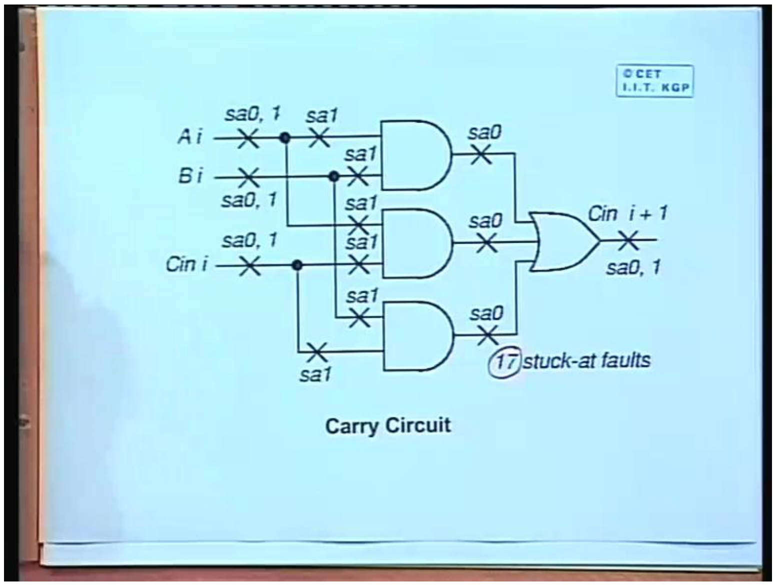
(Refer Slide Time: 40:33)



Actually we are trying to compare functional automatic test pattern generator ATPG is a term which is used automatic test pattern generator versus structural ATPG okay. Now the example that we are taking is a 64 bit adder and just to keep things simple we assume that this is a ripple carry adder. Just for the sake of illustration which means there are 64 full adders which are connected in cascade okay fine. So the 64 bit adder will look like this functionally. This, an adder, this the input A which is 64 bits, input B 64 bits, carry in sum will be 64 bits and a carry out. So if you credit as a black box there are 129 inputs and 65 outputs okay fine. Now we also consider the other alternative that instead of treating it as a block box we considered it as a repeat carry adder where there will be 64 full adders which will be connected in cascade okay.

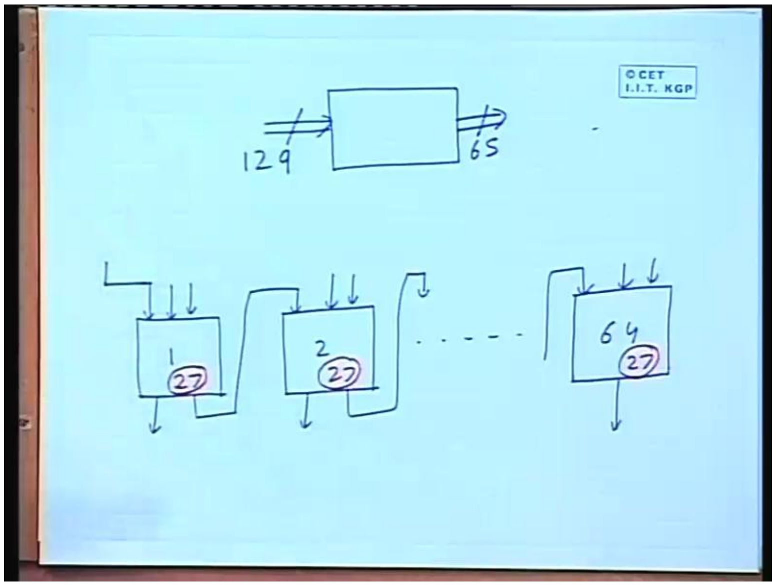
There will be 64 full adder connected in cascade and each full adder will have a sum circuit on a carry circuit. So there will be 64 such blocks and we can have a gate level net list for each of those blocks okay. This is a other alternative we can explore. So in the other alternative you know the sum circuit looks like this. Sum is the exclusive OR of Ai, Bi and say this is the some circuit of a full adder. So if exclusive OR gate is available as a basic gate, then this will be the net list. And after fault you can say for collapsing for XOR gate you cannot do much for collapsing. So there will be 10 stuck at faults possible in each individual sum circuit okay. Similarly, if we look at the carry circuit.

(Refer Slide Time: 43:08)



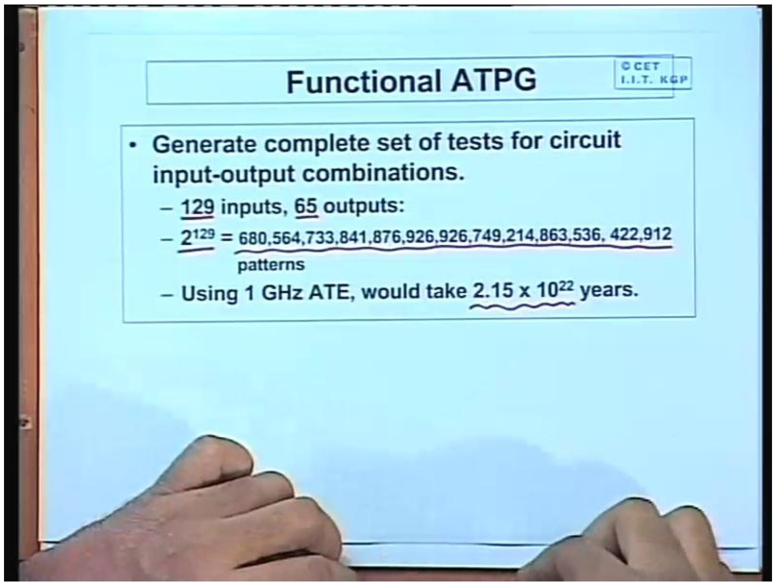
Forthecarrycircuit, carry circuit is AB or BC or CA. So for the carry circuit this will be the realization in terms of AND and OR gates. And using fault collapsing and fault equivalence dominance collapsing you can reduce the number of faults. So it becomes 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17. So for the carry circuit there are 17 stuck at faults. So what does this mean?

(Refer Slide Time: 43:46)



This means that at one extreme we have a realization which is block box realization with 129 inputs and 65 outputs, and in the other extreme we have 64 full adders connected in cascade. These are the full adders 1, 2 and 64. So each of these will be having three inputs and one some and one carry will be going to the other one okay. So it will look like this. So the first one the carry will be coming from outside. This is the carry in okay. Now if you if you can come back and look at the number of faults that we are talking about for this some circuit we are set there are 10 stuck at faults for the carry circuit there are 17 stuck at faults. So there are 27 stuck at faults. So with respect to the gate level net list inside each of these there are 27 possible stuck at faults? Okay. So this is the scenario. So now let us compare these two alternatives what is the complexity of test generation. Say the first alternative the functional test pattern generator. Well if you treat the whole thing as a block box then nothing will give you confidence. Well unless you apply all possible inputs and see the output because you do not know that which sub set of input to apply in order to in testing okay.

(Refer Slide Time: 45:50)

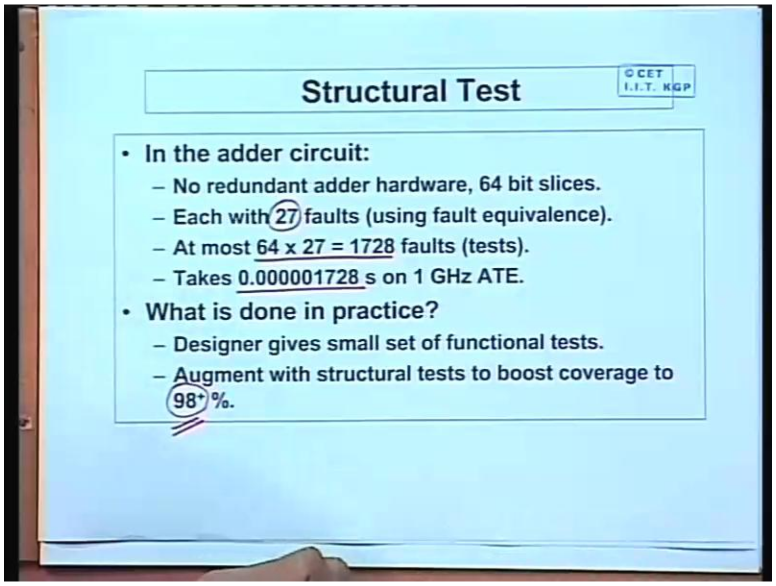

Functional ATPG

© CET I.I.T. KGP

- Generate complete set of tests for circuit input-output combinations.
  - 129 inputs, 65 outputs:
  - 2^{129} = 680,564,733,841,876,926,926,749,214,863,536, 422,912 patterns
  - Using 1 GHz ATE, would take 2.15×10^{22} years.

But in this case applying all possible input is simply out of the question. Why? There are 129 inputs of course 65 outputs. So if you want to apply all possible inputs it is 2 to the power 129, which is a number so large? And even if you assume that you can apply 10 to the power 9 patterns per second at the rate of 1 Giga hertz, still it will take you 2 into 10 to the power 22 years to apply. So many patterns okay. So just using this blind kind of a functional ATPG is simply out of the question. So what people normally do they used this functional ATPG with some kind of an intelligence some kind of an intuitive you can say experience regarding what kind of faults that can occur in the circuit. And what kind of functional behavior you can expect in the presence of faults, so based on design experience they try to design the input test Z okay fine. Now the other alternative. Now if we go for structural test.

(Refer Slide Time: 47:18)



Structural Test

© CET I.I.T. KGP

- In the adder circuit:
  - No redundant adder hardware, 64 bit slices.
  - Each with $\boxed{27}$ faults (using fault equivalence).
  - At most $\underline{64 \times 27 = 1728}$ faults (tests).
  - Takes $\underline{0.000001728}$ s on 1 GHz ATE.
- What is done in practice?
  - Designer gives small set of functional tests.
  - Augment with structural tests to boost coverage to $\boxed{98^{+}}$%.

Now in the adder circuit the triple carry adder we are set that each full adder stage will have 27 faults. So there are total of so many faults. So just assuming that each fault will require one test well in general one test vector can detect more than one fault. So it will be less than this. So even assuming that one test will detect only one fault. So you need 1728 test vectors for this single stuck at fault model. So on a 1 Giga hertz ATE it takes about micro second. So you can see that this is a very practical way of handling the problem. If you treat the circuit as a gate level you need a time for testing gates drastically reduced.

So in practice what people do is that designer initially gives a small set of functional test like for an adder. It tries to find out it has to give the input at one of the input is 0; other is something else what happens. You gives up input so that there is a carry you add the two maximum number all one patterns these are some of the intitutive inputs that the design a gives. So in addition to that you use some structural test and try to boost the fault coverage as high as possible. Typically the industrial people will be happy with a fault coverage in the tune of 98 plus. Now this simple example will give an idea that this functional test all duty is good.

It can sometimes require enormous amount of time and effort in order to generate the test vectors. Okay. The next point that we want to address is that well whatever we have discussed. So for the example illustration we have given. We are taken only for combination circuits. But all practical circuits are sequentially in nature. There will be some storage elements likes flop flops as well. So if we have the storage elements in them they complexity will also go up accordingly. So our next topic of discussion could be how to address the problem of testing sequential circuits. Well they are all ways of generating test vectors for sequential circuits.

But they are much more complex as compare to the combinational test generation problem. So what normally design as do there is something like this. See you have a given circuit see earlier, what happened the designer and the test engineer these were two independent teams. The designer designed a circuit or a chip they gave it the test engineer in the test engineer starts goes to test the chip. Now as the complexity of the chip is increasing the number of components divided by the number of pins that in increasing day by day. So the task of the test engineer is becoming more and more difficult.

So now what happens the two teams very closely collaborate with each other? The test engineers provides some inputs to the chip designers in the sense that they say that well. Here us some of the design rules which do it want to be incorporated in the designs that you come up with. So if you follow those design rules then the final product that you get the chips. They will be such that it will be easier for us to test. So there is a philosophy called design for testability just come up. Now in our next lecture we shall be we shall be trying to discuss more on the design for testability issues in the different techniques and where is people go about doing it in some detail. Thank you.