

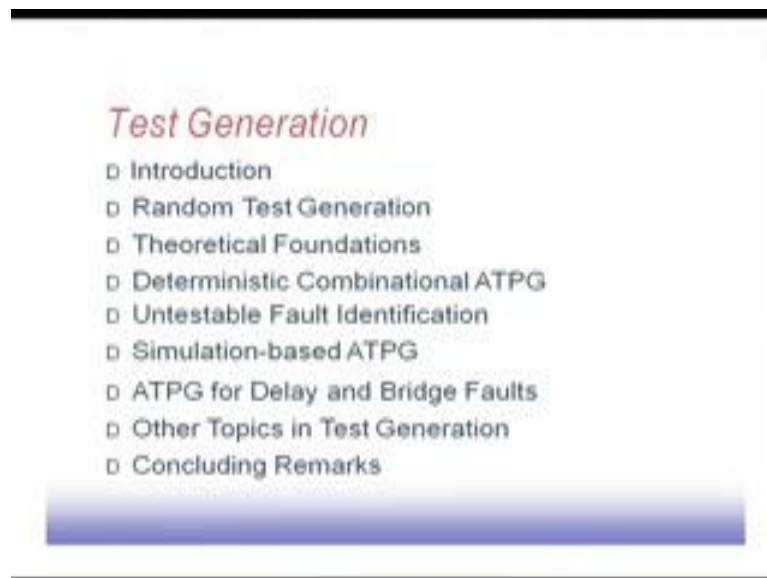
Digital VLSI Testing
Prof. Santanu Chattopadhyay
Department of Electronics and EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 17
Test Generation

This generation is one of the very important steps in the testing process because this is where the test patterns are generated for given a circuit description and a list of faults. So, this test pattern generator, they are supposed to generate the test pattern that should be applied to the circuit to check whether the circuit is or not. So, corresponding to test patterns, there will be a good response that is coming from a good circuit and in the presence of some faults that response will be different. So, based on that we can find out we can detect those faults that may be present in the circuit.

This test generation process, we will go like this. So, we will first look into the introduction general introduction to the problem then we will discuss about random test generation.

(Refer Slide Time: 01:01)

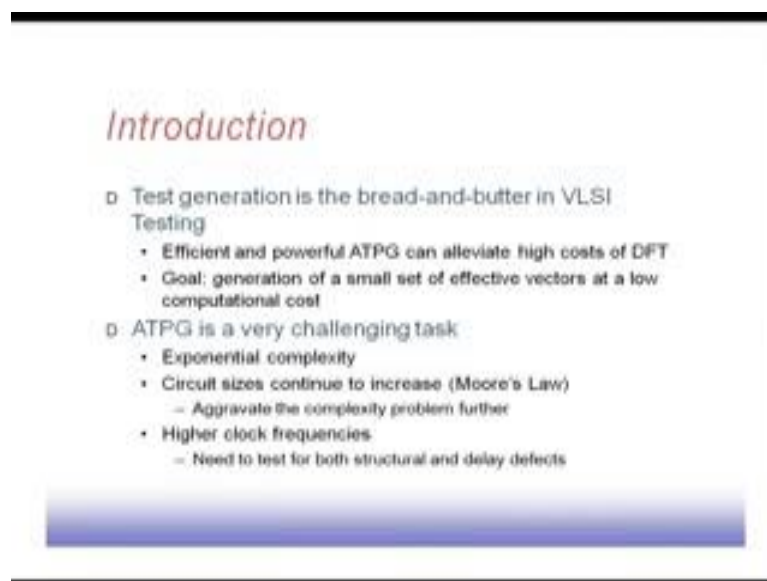


Random test generation means we randomly apply some test patterns and see whether some faults are getting covered or not and it is interesting to note that a good number of faults are detected by some random patterns and it is many times it is up to 70 to 75 percent faults of a circuit that are detected by random patterns. So, the remaining faults,

they there we need to be causes and actually this test generation algorithm. So, this needs to be applied for the remaining 25 percent, about 25 percent of faults.

Then we will go into the theoretical foundations then the deterministic combinatorial ATPGs then how to detect some untestable fault that is some fault that are untestable. So, if we can remove those faults from our consideration then naturally this test generation process will be faster because it will not target on those untestable faults then certain ATPGs are based on simulation based on simulation. So, basically the algorithms that uses genetic algorithm or such some such evolutionary techniques then there are some delay and bridge bridging faults. So, they need some special type of ATPGs. So, that they will be covered at that time some other topics and then we will go into the concluding remarks.

(Refer Slide Time: 02:29)



Test generation is the bread and butter in VLSI testing. So, this is quite common statement because this is what this test generation test this test engineer. So, they are doing. So, they are trying to ensure about more and more coverage of the faults in the circuit. So, if we are trying to do that then definitely we should have a very good test pattern set to be applied to the circuit efficient and powerful ATPG can alleviate the high cost of DFT because if the DFT means we need to have some extra controllable and observable points added into the circuit whatever be the technique ad hoc technique or structure technique whatever it is, but if the ATPG itself is very good and if the ATPG

algorithm can itself give us a coverage of high number of faults then definitely we may not be requiring. So, many point, many controllable and observable points in the circuit.

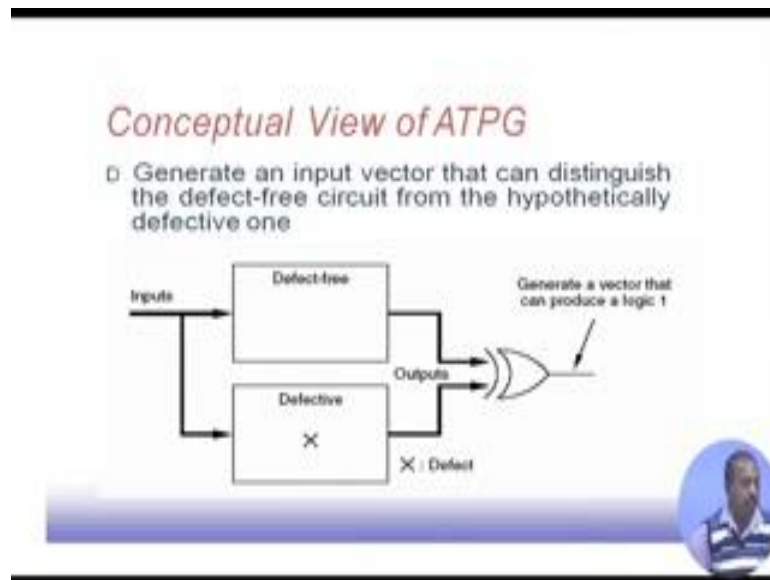
As a result, the cost of DFT can go down. So, the goal is to generate a small set of effective test vector low computational cost. So, this is also another important issue because you see whenever we are generating a set of test patterns or a particular or a single test pattern we need to know what are the faults that this pattern can detect maybe previously we have generated 100 patterns and those 100 patterns are covered some 10000 faults present in the circuit.

Now, the 100 first pattern that we are going to generate, it must be able to uncover some faults which are not yet detected by previously generated 100 patterns otherwise this is a no use. So, we are definitely interested to have a good coverage and we want effective vectors and this effective vector generation process should not be computationally very expensive because then the applicability of that particular algorithm maybe in question.

ATPG is a challenging task because the problem is exponentially in nature. So, if we are trying to generate the exact minimum test set for a particular circuit and for a particular set of faults then this problem is exponential. So, we have to take help of heuristics and the heuristics must be running in reasonable time and giving us a good amount of fault coverage circuit sizes continue to increase by Moore's law. So, it is doubling after a regular interval of time. So, it aggravates the complexity problem further. So, this more number of devices put into the circuit means we have to have more number of faults and the naturally we need more number of patterns to test them.

Clock frequencies are increasing. So, clock frequency, we need to test both structural and delay defects. So, if the clock frequency increasing means a gate delay is should be such that is combinatorial gate delay should be such that the evaluation of the input occurs within the clock duration. So, the clock speed is high then there is a high chance that some of the gates will have some problem in them and they are delays may not be within the clock duration. So, as a result it will have more number of delay defects. So, delay defects handling becomes an important issue.

(Refer Slide Time: 05:53)




Conceptually an ATPG can be viewed like this. So, we have got a defect free circuit and a defective circuit the inputs are applied to both the defect free and defective circuits. So, if suppose X is the defects. So, X is the defect it a manifest some fault in the circuit. So, under that fault or defect whatever you call it that defect X the defective circuit output and these defect free circuit output there are they should be different as a result if you XOR them. So, we will get a logic one at the output. So, the challenge is that we are given this defect free circuit and this defective circuit and knowing that we have got an XOR gate at the output of these 2 where the outputs of these 2 circuits have been XORed. So, we have to design an input vectors such that this XOR gate output is a logic 1. So, XOR naturally if these 2 outputs differ then it will be logic 1. So, we have to derive this particular input set.

(Refer Slide Time: 06:56)

Fault Models

- Instead of targeting specific defects, fault models are used to capture the logical effect of the underlying defect
- Fault models considered in this chapter:
 - Stuck-at fault
 - Bridging fault
 - Transition fault
 - Path-delay fault


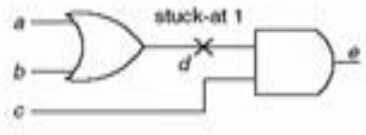


As we have already seen that physical defects are not handled directly rather we have some fault models and with respect to that fault model, we need to generate that test pattern set. So, fault models that will be considering in this part of our lecture as stuck at fault bridging fault default transition fault and path delay fault. So, these are the types of faults that will be covering in this part of the lecture.

(Refer Slide Time: 07:24)

Simple illustration of ATPG

- Consider the fault d/1 in the defective circuit
- Need to distinguish the output of the defective circuit from the defect-free circuit
- Need: set d=0 in the defect-free circuit
- Need: propagate effect of fault to output
- Vector: abc=001 (output = 0/1)

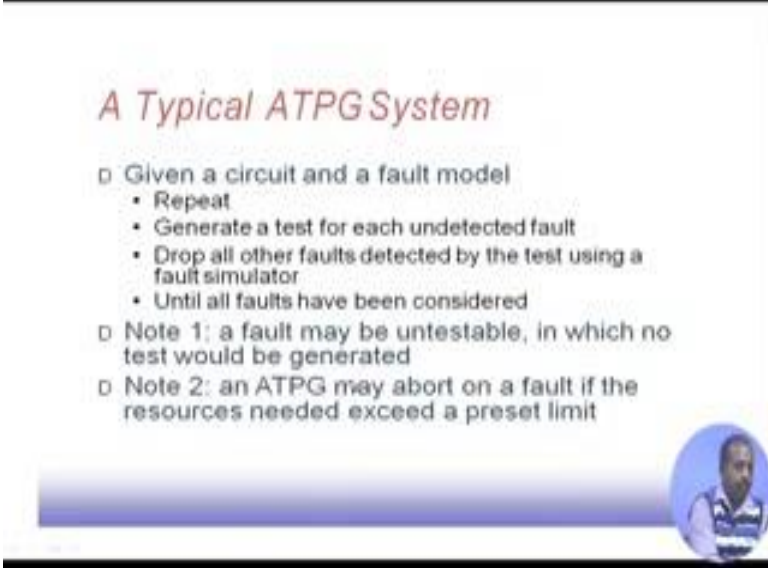


Let us take a start with the simple example. So, consider a fault which is a D stuck at 1, we want to, this is the circuit we having 3 inputs a, b, c and 1 output e and we want to

detect this particular fault D stuck at 1. Now to check to test this fault, 2 things are necessary, since this line is stuck at 1, first of all we need to force this line to 0, any test pattern that will apply here should try to force this line to 0 and how can this be done since this is an OR gate putting a and b both the inputs to 1, both the inputs to 0, I can get a 0 at the point D. So, we need to excite the fault which is in terms of opposite of the fault that you want to detect. So, you want to detect stuck at. So, we have to force a 0 here for that this a and b must be equal to 0.

Now, the line d is not directly observable. So, observable is the line e. So, somehow that effect has to be propagated through this AND gate and we must get the response at e and for that purpose this other input of the AND gate it must hold a non controlling value. So, this c input must be at a 1. So, you see that we have to apply a equal to 0, b equal to 0 and c equal to 1. So, if you do that then in a fault free circuit this line, D is at 0, c is at 1. So, e is at 0 in a faulty circuit this line, D will be at 1 because this is stuck at 1 and c is also 1. So, e will be at 1. So, fault free response is 0 faulty response is 1. So, we say that this particular pattern 0, 0, 1, it can detect this particular fault D stuck at 1.


(Refer Slide Time: 09:13)



A Typical ATPG System

- Given a circuit and a fault model
 - Repeat
 - Generate a test for each undetected fault
 - Drop all other faults detected by the test using a fault simulator
 - Until all faults have been considered
- Note 1: a fault may be untestable, in which no test would be generated
- Note 2: an ATPG may abort on a fault if the resources needed exceed a preset limit

© 2004 Intel Corporation



A typical ATPG system will work like this given a circuit and a fault model, what it will do? It will repeat the following steps what generated test for each undetected fault. So, initially all the faults in the circuit are undetected. So, it can try; it can try to generate a test pattern for that. So, if it successfully generates one test pattern, certain faults in that

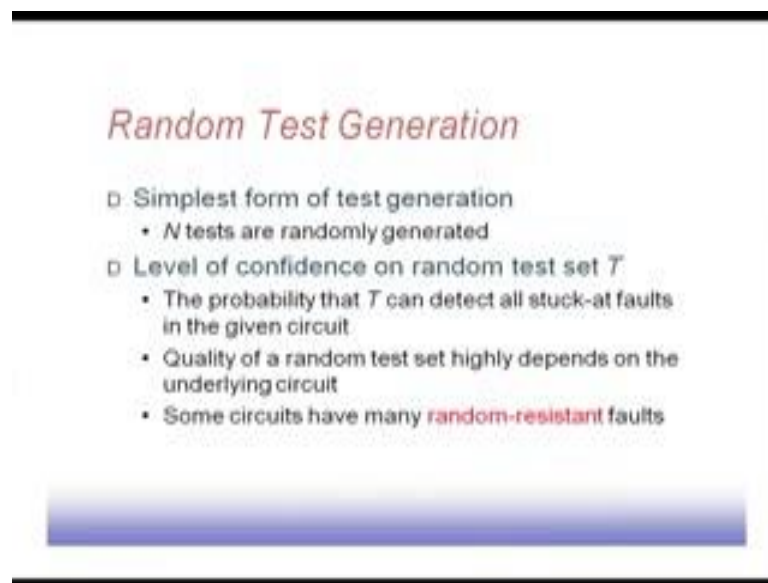
set gets detected. So, those faults we need not consider anymore. So, in our successive attempt we should be trying some other faults which are not yet detected. So, that is what is coded and retained here. So, generate a test for each undetected fault and drop all other faults detected by the test using a fault simulator. So, there are a number of faults. So, we just pick up one of those faults and target it for test generation. So, the corresponding test pattern, it will detect not only this particular fault, but maybe some more faults as well. So, when we do it, when you do that then all those faults can be dropped from the set because those faults are also detected by that pattern.

For example, in this case, you see that if this output *e* changes from 0 to 1, if you are trying to check *e* stuck at 0 then also *i*. So, this pattern 0, 0, 1 will push this pattern 0, 0, 1 will push, it will make an output 0. So, if you are trying for you know this particular pattern is sufficient to detect the fault which is *e* stuck at 1. So, under these 0, 0, 1, this will be this if *e* output is 0 in the fault-free case in the, if we are trying to get a check *e* stuck at 1. So, this pattern is sufficient. So, once you have detected this pattern 0, 0, 1 for *d* stuck at 1 fault will also can drop that fault *e* stuck at *e* stuck at 0 from this set.

Now, a fault maybe untestable in which no test would be generated, if a fault is untestable due to some redundancy in the circuit then that that for that fault will not be detected by any test pattern. So, naturally that type of faults can be detected from that can be [drop/dropped] dropped from the set. So, these untestable faults we if you can separate out from our test generation process then the test generation process may be faster.

Second case, an ATPG may abort on a fault if the resources needed exceeded preset limit. So, at P G they need to run. So, it requires CPU time, it requires memory for its operation. So, if it is the case that for testing a particular fault it requires it is requiring very huge amount of time. So, it is the process is continuing it is not over, but due to the circuit structure or due to the nature of fault it is happening that it is taking a large amount of time. So, after a certain limit maybe we can drop that fault and we can say that this fault will not be detected. So, this fault is not redundant or this fault is not untestable it can be tested, but the current setup of ATPG is not able to do that. So, instead of instead of just looking behind this particular fault the ATPG will look for some other faults because ultimate goal of ATPG is to improve fault coverage. So, if it can detect some other faults. So, fault coverage can be improved.

(Refer Slide Time: 12:55)



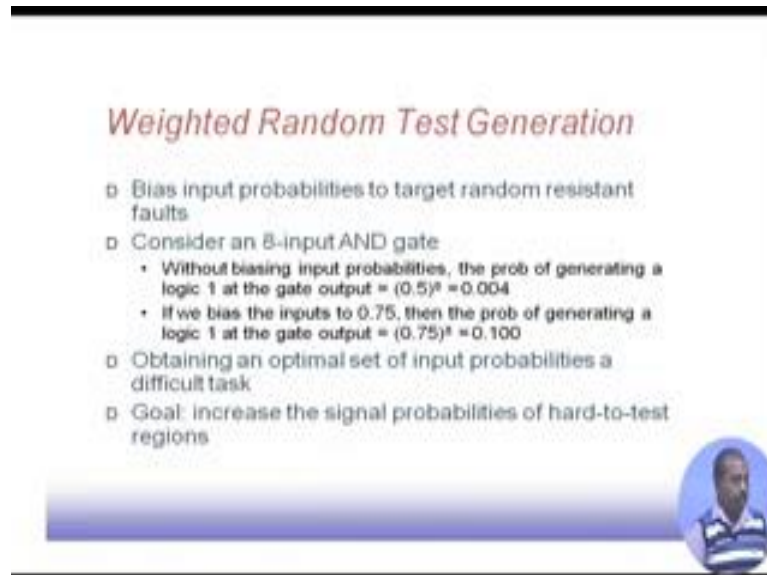
Random test generation, this is the simplest form of test generation. So, we generate N randomly, generate N random test pattern. So, this random test pattern, basically it say if it say 2 valid circuit, 2 valued logic, we apply random 0 1 patterns to the inputs then see if whether what type of output it produces.

We are generating a random pattern and then we are looking into the fault free response then we try out all possible faults in the circuit one at a time. So, if it is stuck at 1 from model that we are stuck at fault model that we are considering. So, it will insert one stuck at fault and see whether that fault is getting detected by this particular pattern. So, it is basically does a fault simulation. So, after fault simulation it will know what are the fault that are detected by this particular random pattern and all those faults are dropped then it will go for the next pattern you generate another random pattern and see whether any of the remaining faults are covered.

This way it does the random test generation. So, if we are trying to see what is the level of confidence on random on random test set T , the probability that T can detect all stuck at faults in the given circuit. So, this is the measure of confidence and quality of a random test set highly depends on the underlying circuit. So, some circuits they have got faults which are mostly covered they are mostly getting covered by random patterns. So, they are easy to detect and some faults are there in the circuit which are random resistant


faults. So, they actually require a specific type of a pattern to be applied to the circuit for detecting that fault.

(Refer Slide Time: 14:49)



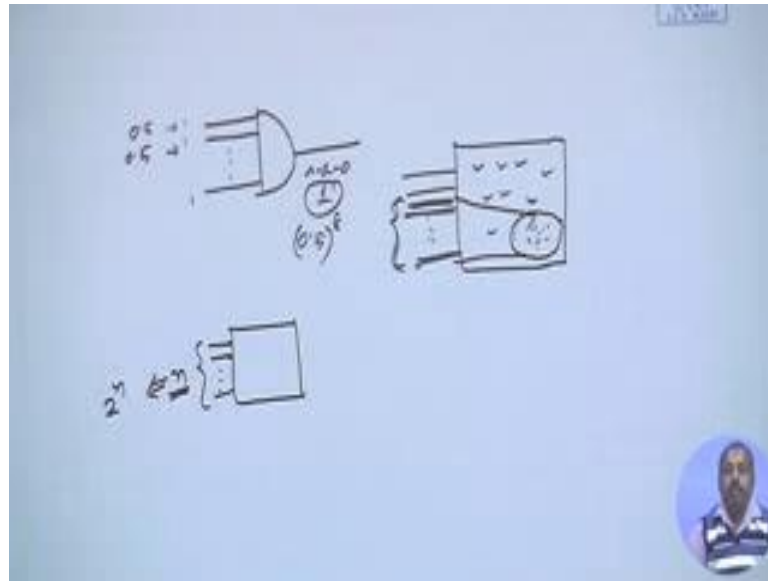
Weighted Random Test Generation

- Bias input probabilities to target random resistant faults
- Consider an 8-input AND gate
 - Without biasing input probabilities, the prob of generating a logic 1 at the gate output = $(0.5)^8 = 0.004$
 - If we bias the inputs to 0.75, then the prob of generating a logic 1 at the gate output = $(0.75)^8 = 0.100$
- Obtaining an optimal set of input probabilities a difficult task
- Goal: increase the signal probabilities of hard-to-test regions



That way the possibility of detecting that fault in a random fashion is difficult is difficult. So, naturally they are called random resistant faults. So, to solve these random resistance problems, what is done is sometimes we bias the inputs of a random resistant fault. So, how do it make it, this weighted random pattern like let us take a take an 8 input AND gate, now if you do not have any biasing then all the 8 inputs of this AND gate has got the probability of 0.5 at the input. So, if we are trying to check whether, what is the probability of getting a logic one at the gate output? So, that is 0.52 power 8, it is 0.004. So, if we are trying to check a fault which is a AND gate output stuck at 0.

(Refer Slide Time: 15:42)



For checking that 8 input AND gate, output stuck at 0 stuck at 0. So, this lines stuck at 0. So, I need to push a 1 at this point, now to push a 1 at this point, I must be able to apply all ones at the input.

Now, since they are all these inputs are generated in a through a random process in my random test pattern generation. So, probability of getting 1 here is 0.5, here it is 0.5. So, the probability of able being able to produce a one at the output is 0.5 to the power 8, 0.5 to the power 8. So, that makes it cumbersome. So, the probability becomes very low it is 0.004.

Now, if you can bias the inputs, that the probability of generating logic one probability of getting one at an input is 0.75 instead of 0.5. So, this random number generator is modified in such a fashion that in 75 percent cases it generates 1 and 25 percent cases it generates 0. So, in that case the possibility of getting logic 1 at the input of in each of the input is 0.75. So, the overall probability of getting logic 1 at the output of the AND gate is 0.75 to the power 8. So it is 0.1.

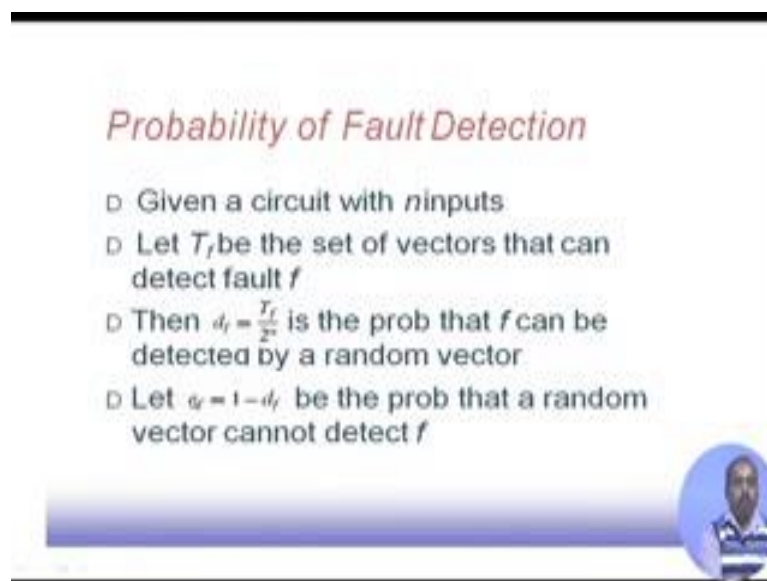
You see from the probability 0.004, it has increased to the probability has increased to 0.100. So, by changing this input probability, we can force certain gates to go to a particular value logic value, the chance of go going to a particular logic value will increase.

However this obtaining an optimal, the set of input probability is a difficult task. So, that is the first thing. So, because that is the analysis that we have shown is for a very simple AND gate, but if it is a complex circuit and we want to do this for all the points in the circuit. So, that way getting of optimal input probabilities will be difficult.

What to do? We want to increase the signal probabilities of hard to hard test region. So, what is the hard to test region. So, if this is a full, this is the full circuit maybe we generate first we apply some random pattern. So, these are the inputs to the circuit. So, we apply some random patterns suppose the faults that are present in these regions of the circuit they are detected, but we see that a particular region of the circuit. So, it is not responding to these random patterns. So, in that case, we know that this region has got faults which are random resistant faults.

For this, then we find out the cloning of influence for this particular region maybe this is the cone of influence for this particular region. So, what we do? We do, we change the input probabilities for these inputs. So, that we can get more more biased random pattern generation that will that can target faults in the hard to test region. So, this way this random pattern generators, they can be modified and accordingly we can get some better test pattern set which will be detecting more number of faults.

(Refer Slide Time: 19:03)



Probability of Fault Detection

- Given a circuit with n inputs
- Let T_f be the set of vectors that can detect fault f
- Then $d_f = \frac{|T_f|}{2^n}$ is the prob that f can be detected by a random vector
- Let $a_f = 1 - d_f$ be the prob that a random vector cannot detect f

Probability of fault detection, given a circuit suppose which have a n inputs, if T_f be the set of vectors that can detect fault f then d_f is $|T_f|$ of 2^n it is the probability that f

can be detected by a random vector. So, if this f is detectable then out of this 2^{2^n} vectors some at least one vector will be able to detect it and if this T_f can detect fault f then this d_f is the detect detection probability d_f is defined as T_f by 2^{2^n} and naturally e_f is one minus d_f is the probability that a random vector cannot detect f . So, this is important. So, if e_f is high for some fault we know that the fault is going to be random pattern resistant fault.


(Refer Slide Time: 19:58)

Prob of Fault Detection (Cont.)

□ Then, $e_f^N = (1 - d_f)^N$ is the prob that N random vectors do not detect f

□ Thus, the prob that at least one out of N random vectors can detect f is

$$1 - (1 - d_f)^N$$



So, e_f power n is $1 - d_f$ power n is the probability that n random vectors do not detect f . So, if I apply instead of single random vectors. So, since random vectors they are all independent of each other. So, if you repeat this experiment n times then it is e_f power n is $1 - d_f$ power n , thus the probability that at least one out of n random vectors can detect f is $1 - 1 - d_f$ power n . So, this is the probability that at least one of the n random vectors will be able to detect f .

(Refer Slide Time: 20:34)

Minimum Detection Probability

- ❑ The min detection prob of any detectable fault actually does **not** depend on n , the num of PIs
- ❑ Instead, it depends on the largest primary-output cone that it is in
- ❑ This is because any detectable fault must be excited and sensitized to a primary output

Inputs outside of PO cone are not needed for detection of fault f

What is the minimum detection probability? The minimum detection probability of any detectable fault actually does not depend on n , the number of primary inputs rather it depends on the largest primary output cone that it is in. So, what it says is that say this is the fault that we are going to detect, now if you see that this is the input cone that we have. So, it does not depend on n , n is the number of inputs; number of primary inputs of the circuit rather it depends on this value because this, suppose this output goes to it, it goes to several this point signal line point f it goes to several output. So, this one, this one and this one out of that we for each of these outputs we have got an input cone of influence. So, input cone of influence of a primary output is the set of primary inputs on which that that particular output depends. So, by doing an analysis traversal from the primary output was the primary input we can identify this type of cones for individual outputs.

Now, whichever cone is the largest that has got the largest number of primary output. So, that way primary input, that way, it will depend on that because in the worst case this fault will get detected and it will propagate to this primary output and in that case, it needs to be controlled by all these primary inputs. So, this is the, this path excited this fault excitation and fault propagation. So, this whole job can be done by setting this the inputs in this cone only. So, that for the inputs which are outside this cone does not have any effect on the excitation of this fault or propagation of this fault to the primary output.

(Refer Slide Time: 22:49)

Lemma 1

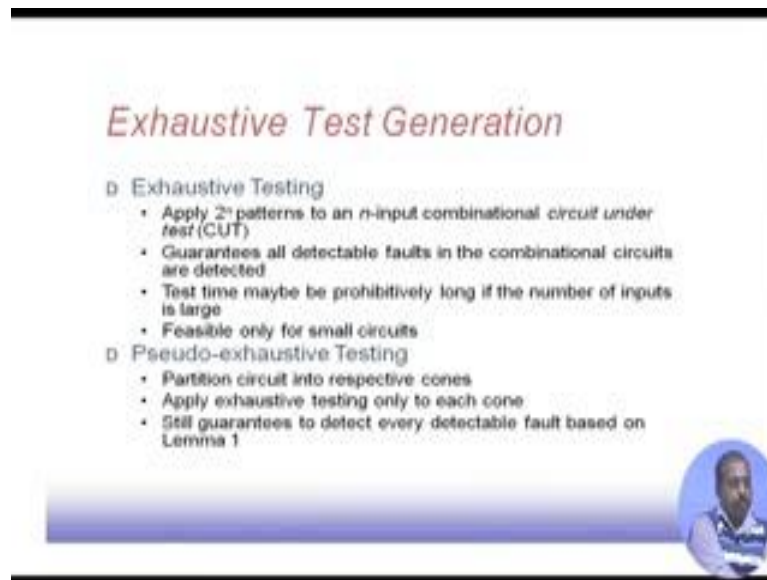
- In a combinational circuit with multiple outputs, let n_{max} be the number of primary inputs that can lead to a primary output. Then, the detection probability for the most difficult detectable fault, d_{min} , is: $d_{min} \geq (0.5)^{n_{max}}$



This is the thing that minimum detection probability. So, it will depend on the largest primary output cone into which this particular faults. So, there is a result that shows details that in a combinational circuit with multiple output, let n_{max} be the number of primary inputs that can lead to a primary output. So, if you consider all primary output cones. So, n_{max} is the maximum number of primary out; input associated with some associated with the primary output. So, there maybe number of primary outputs. So, for each of them we compute the cone and we find that for the largest number of inputs associated with any primary output is n_{max} .

Then the detection probability for the most difficult detectable fault is d_{min} greater or equal 0.5 to the power n_{max} because this will be the each of these inputs may be set to 0 or 1. So, as a result, if the probability is 0.5, 0.5 to see each of these inputs are to be set to some value and that way it is 0.5 to the power n_{max} . So, d_{min} will be greater or equal this quantity. So, that gives a better estimation of the number of of the detection probability of a fault.

(Refer Slide Time: 24:00)



Exhaustive Test Generation

- Exhaustive Testing
 - Apply 2^n patterns to an n -input combinational circuit under test (CUT)
 - Guarantees all detectable faults in the combinational circuits are detected
 - Test time maybe be prohibitively long if the number of inputs is large
 - Feasible only for small circuits
- Pseudo-exhaustive Testing
 - Partition circuit into respective cones
 - Apply exhaustive testing only to each cone
 - Still guarantees to detect every detectable fault based on Lemma 1

Next we will look into different test generation techniques to start to it will look into exhaustive test generation. So, exhaustive testing it says that apply 2^n patterns to an n input combinational circuit under test. So, what we mean is that this is the circuit it has got n number of inputs it has got n number of inputs. So, irrespective of what this, what type of faults that may be present etcetera. So, ultimately we are going to apply one of these patterns like this n inputs can have at most 2^n different combinations the different permutations sorry different combinations that can come here. So, naturally I can, any test pattern is nothing but a combination from these 2^n set.

In the worst case, so we if you apply all to 2^n test patterns, all to 2^n possibilities to the circuit then we should be able to detect all possible detectable faults in the circuit. So, this is the exhaustive testing. So, we apply 2^n patterns to an n input combinational circuit under test I guarantees that all detectable faults in a combinational circuits are detected because no more pattern can apply any test pattern set is a subset of this 2^n .

This time maybe prohibitively long if the number of inputs is large. So, if n is small 10, 15, 20, fine so, you can do this thing, but if n goes to 100s then it is out of questions. So, it required huge amount of time. So, it is feasible only for small circuits. So, this is the exhaustive testing, but the advantage is that we do not need to consider the type of faults that are present. So, for every fault that if that can be detected. So, we have applied a test

pattern to the circuit. So, all those faults will get detected. So, we do not need to consider the fault models.

Another category of this exhaustive testing is a pseudo exhaustive testing. So, in this case the party is partition the circuit into cones. So, one primary output depends on a subset of inputs. So, it is not necessary that for to excite all possible outputs at the all possible output for the primary output line. So, it is not necessary to excite all the input. So, it only necessary to excite the input cone with all possible patterns so that way we if we can partition in 2 cones and we exhaustively test each cone. So, then that way the complexity maybe less, but still guarantees to detect every detectable fault based on the previous result that we have shown that the d_{min} expression.

So, continue in the next class.