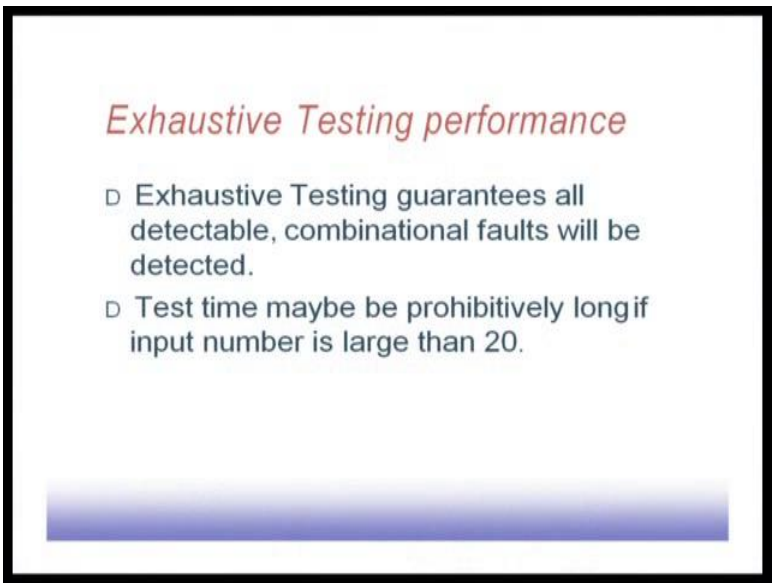
Digital VLSI Testing
Prof. Santanu Chattopadhyay
Department of Electronics and EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 26
Logic BIST (Contd.)

(Refer Slide Time: 00:25)



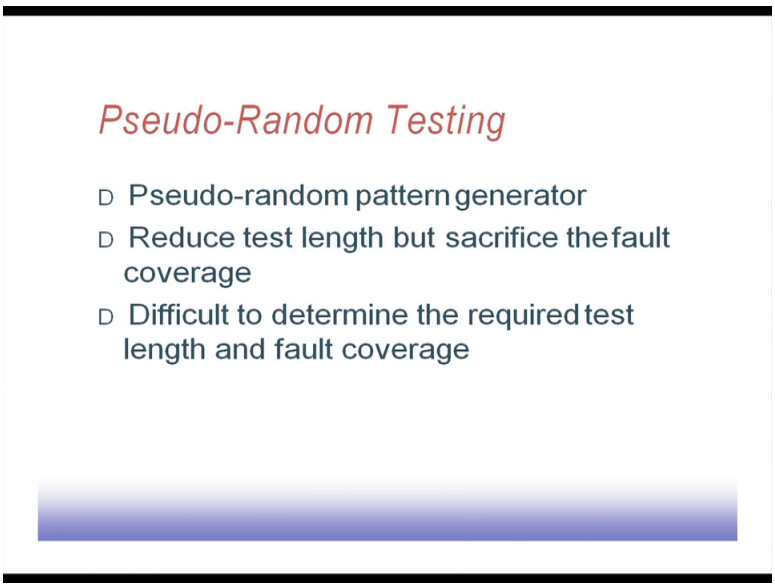
Now, this LFSR or the CLFSR and this counters they can be used for generating exhaustive test patterns because all the patterns are generated by some by these structures. Now the problem is that this exhaustive testing is good because it guarantees all detectable combinational faults that are to be detected, but at the same time test time becomes prohibitively large. So, if number of inputs is more than 20 then I require a large amount of time to generate those test patterns and apply to the circuits. So, each of these patterns are to be generated by the pattern generator, applied to the circuit and the response has to be summed up then the problem is that you cannot stop in between like you cannot say that I will apply only say first 100 patterns.

Because there is no guarantee that the first 100 patterns are very good they can detect large number of faults unlike the determininistic testing where we know that the first patterns that first few patterns that are there in the test pattern set, they are very good. So, they detect large number of faults and all that, but in case of this BIST policy, since we

do not have any control over the patterns that are getting generated. So, we cannot assume that the first 100 patterns would be sufficient for testing the circuit and all that.

So, we need to apply all the patterns. So, that way this test time becomes prohibitively large when it is number of inputs is more than 20. So, what is the way out? Way out is to have some pseudo random pattern generator. So, it is a pseudo random testing has to be used. So, here we do not apply all the patterns. So, we apply only the subset of patterns knowing fully well that I will not get the same level of confidence as the exhaustive testing, but when as we have told in some class earlier that in even if you apply some random patterns. So, it is normally seen that about 80 percent of the faults are may get detected by the random pattern itself random pattern themselves.

(Refer Slide Time: 02:29)

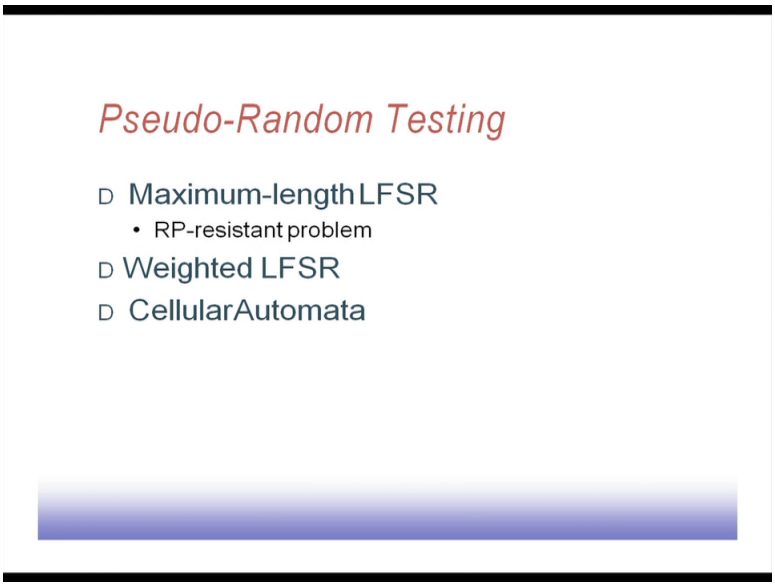


So, that is why, it is possible that we apply some pseudo random pseudo random testing. So, we have got a pseudo random pattern generator. So, it reduces test length, but sacrifice the fault coverage and it is difficult to determine the required test length and fault coverage.

Because how long will you done. So, basically what we can do is we can do a simulation like this is my random pattern generator. So, I simulate this pattern generator for 100 cycles that gives 100 patterns and for these 100 patterns, we do a fault simulation to see, what are the faults that are detected by this? So, that maybe one possibility, but of course, there is another issue like this response analyzer is also not going to check these

individual outputs separately, but it is only going to give us a signature at the end of the section. So, at the end of the section what are the; for each fault we can see whether these gives a different signature or not for all those words where we get a different signature from the for free circuit. So, those faults are detected others or not. So, that way fault coverage becomes an issue. So, it is very difficult to determine the test length and the fault coverage.

(Refer Slide Time: 03:40)

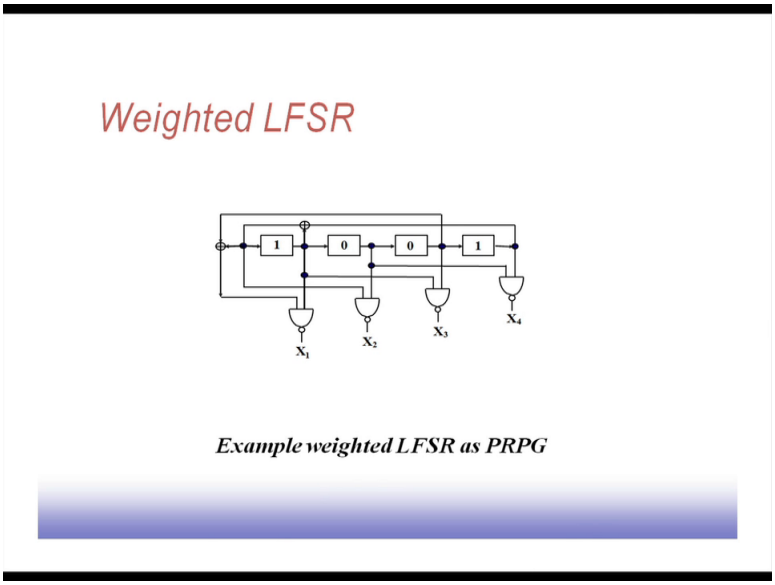


So, this maximum length LFSR; it can be used as a pseudo random pattern generator, but the problem is that there are certain fault in the circuit which are random pattern resistant. So, random pattern resistant faults are those faults where possibility of heating the pattern that can detect the fault is very low.

So, this happens particularly where there is there are only a few specific test patterns that can detect a fault and then and since we are since we are trying a random generation. So, probability of getting that particular pattern generator is a small. So, we can modify it we can modify this maximum length LFSR properties to have some weighted LFSR. So, in the weighted LFSR we change we change the weight factor and say that some portions they will have some of the cells they will generate more ones than others. So, as a result there is a weight factor and the pattern generated is skewed and this skewed pattern that are generated may be helpful in targeting some random pattern resistance faults.

And a cellular automaton is another structure. So, that is also used in many cases the idea is that if your individual cells they depend on its contents depend on its neighbors. So, unlike LFSR, where only the first cell is getting the summed up, values from different cells and in case of cellular automata, every cell consults its neighbor and accordingly based on some logic functions. So, it computes the next value of the cell, but anyway there are they are also the same issues will remain in the sense that this randomness cellular automata. So, that will also be another issue.

(Refer Slide Time: 05:35)

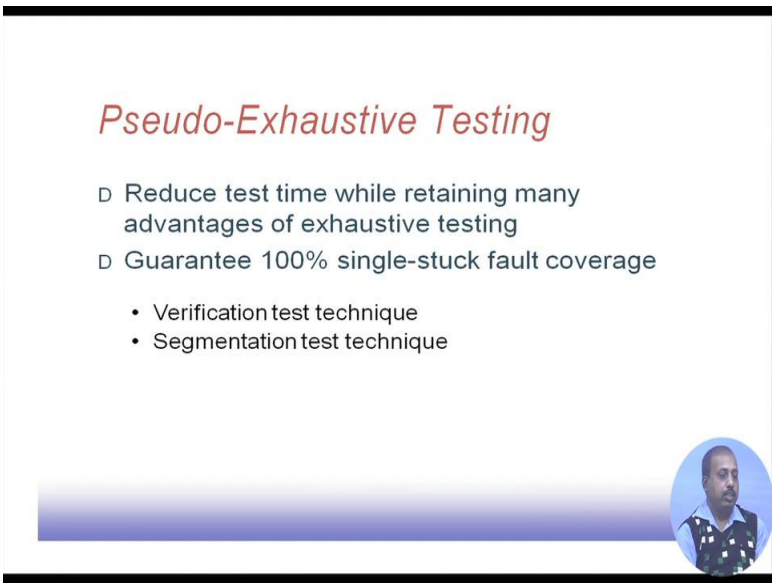


So, what is an weighted LFSR? So, weighted LFSR is like this. This is a normal LFSR where you have got this cell 1, 0, 0, 1 and you see that these LFSR, this is generating, if you if you if you just run it continually, it will generate all the patterns. It is a maximum length one. So, it generates it. Now you see that we put some extra logic. So, these x 4 will be; x 4 output will be 0 only when both this cell and this cell, they are when both are 0, this cell when both of them are 1 then only this output will be 0, otherwise this in and output will be 1. So, assuming that these 0 and 1 are equi-probable, these 2 cells are equi-probable in the random sequence. So, you see that the probability of getting a one at this point is much less compared to getting a 0.

Because you will get a 0 whenever either of either of them is you will get a 0 when both of them are 1. So, the probability of getting a 0 is much less compared to getting a 1 because you will get a 1 whenever any of these cell is; cell content is 0. So, you will get

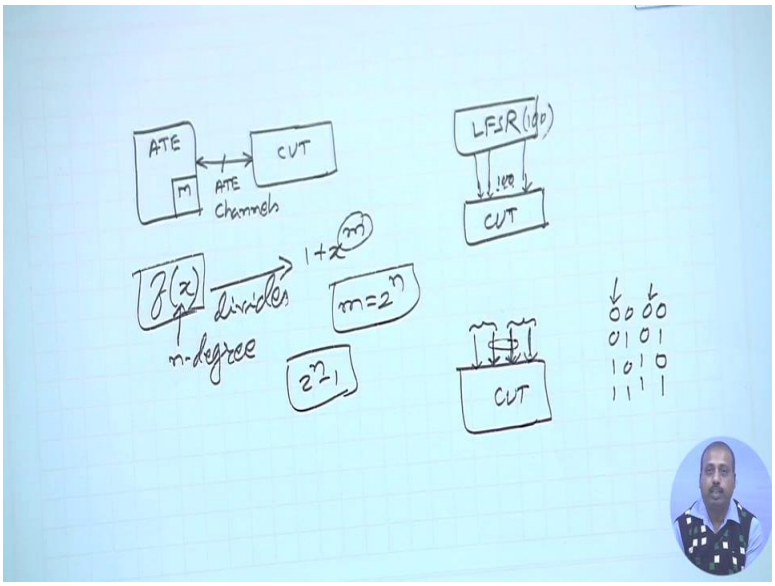
a 1 at this point, similarly if you consider say this x 1, x 1 will be; x 1 output will be 1 when either of these 2 inputs are going to be 0. So, that the probability of getting a 1 and this point is higher compare to getting a 0. So, in this way we can we can use some weight factor and accordingly we can choose input from these different cells and put them into this gates to get some weighted pseudo random pattern generator.

(Refer Slide Time: 07:26)



Another way of handling this testing of this exhaustive another modification to the exhaustive testing is the pseudo exhaustive testing. So, pseudo exhaustive testing; it tries to reduce test time while retaining many advantages of exhaustive testing. So, it guarantees 100 percent stuck at fault coverage. So, there are. So, basically what we are trying to do is if this is a circuit.

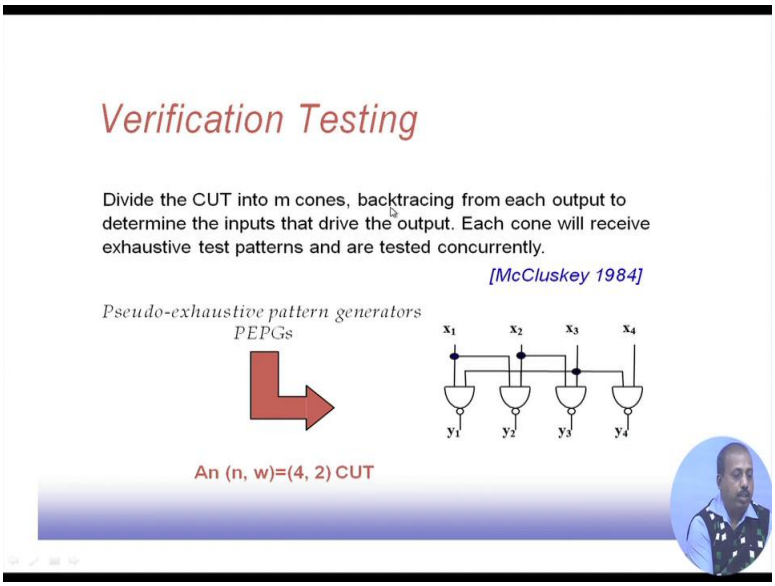
(Refer Slide Time: 07:54)



If this is a circuit that we are trying to test, in a an exhaustive testing, if it has got these inputs then it will be trying to generate exhaustive patterns for all the 4 inputs, but it may be; it may be sufficient that I generate exhaustive pattern for these 2 and I generate exhaustive pattern for these 2 for the remaining one I do not. So, there is the patterns that I apply they are not exhaustive when you consider these 2 inputs.

So, when we consider these, here the patterns are not exhaustive. So, if you are looking to the patterns so, here I am applying 0, 0, 0, 1 and 1, 1 show all the pattern. So, similarly here also I am applying 0, 0, 0, 1, 1, 0 and 1, 1, but you see that if you are considering say this Input and this and input. So, it is not it is not exhaustive. So, it is generating only 2 patterns, 0 0 and 1 1. So, this is another; this is this sort of test; they are called pseudo exhaustive testing. Now for a circuit, it may be the case that we do not want all this exhaustive patterns to be applied only at certain input cones of the circuit. So, we want this exhaustiveness. So, that way we can have this pseudo exhaustive testing.

(Refer Slide Time: 09:18)

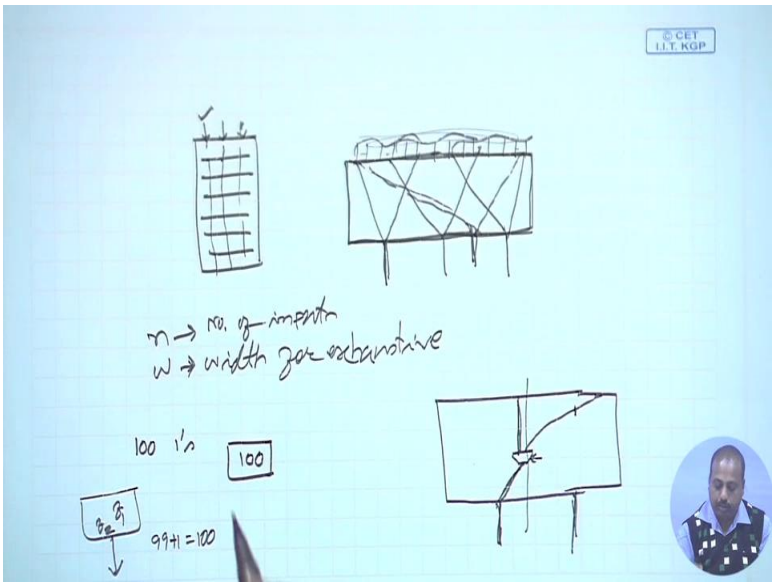


So, first one is known as the verification testing. So, what we do? We divide the circuit under test in to M cones and backtrack from each output determine the input that drive the output and each cone will receive exhaustive test patterns that are tested concurrently. So, what is happening is that suppose in this particular case. So, we have got the 4 to cut where n is the number of inputs and w is the maximum dependency. So, if you look into these circuits. So, if you take this say y 1 y 1 output you see that it is getting input from x 1 and it is getting input from x 3. So, for testing these gates exhaustively I need exhaustive patterns only at x 1 and x 3.

Similarly, so, this gate y 2, it gets input from x 1ne and x 2. So, again I need the; I need the exhaustiveness in x 1 and x 2 only. So, in these particular circuits, if I can apply a test pattern set such that it is exhaustive for any for all the 2 inputs like if you pick up any 2 inputs. So, I want to apply a test pattern set such that if you pick up any 2 inputs. So, it will have the exhaustive pattern in that like if it say this test pattern has got say 10 patterns then if you pick up if you look into the lines x 1 and x 3 then out of those 10 patterns, you will find that it has got all those sequences 0, 0, 0, 1, 1, 0 and 1, 1, all those combinations are there similarly if you look into x 1 and x 4 also you will find that somewhere or the other is 0, 0, 0, 1, 1, 0 and 1, 1, those patterns are there.

So, this is this type of pattern generation. So, they are known as this pseudo exhaustive pattern generation. So, this is exhaustive only at only for a subset of numbers and they are tested concurrently. So, what is happening is that I have got this test pattern set.

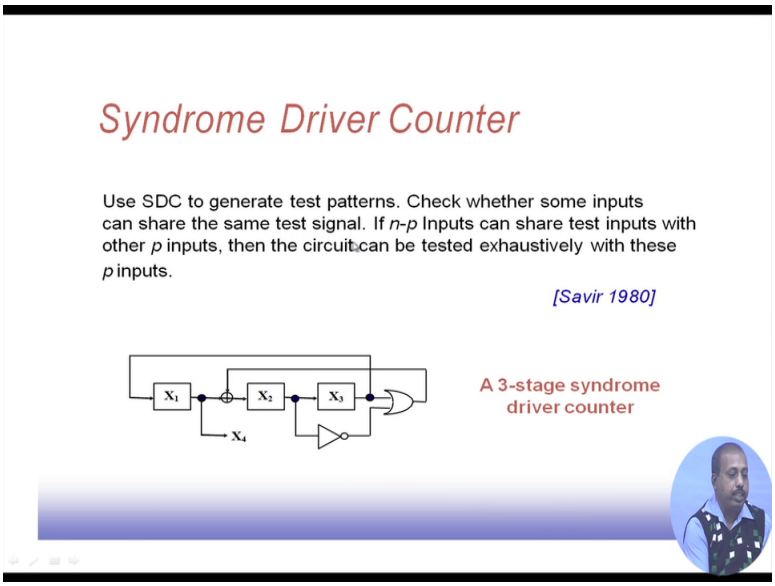
(Refer Slide Time: 11:27)



I have got this test pattern set now in this test pattern set in this particular circuit if you pick up any 2 positions say you pick up say these positions and say this position. So, it is exhaustive there similarly if you pick up these and this position. So, there also it is exhaustive here. So, these patterns you can apply simultaneously to all the circuits all the cones like if this is my circuit and it has got say these 3 are the outputs now if you see the input cones maybe this is one input cone this is another input cone this is another input cone these are the 3 input cones one possibility is that I test this cones separately one at a time. So, that is one possibility other possibility is that I apply the patterns in such a fashion that it is exhaustive for this cone it is exhaustive for this cone it is exhaustive for this cone.

So, for all this subset this for individual cones it is exhaustive, but the testing is done concurrently. So, that is what is said here that the each cone will receive exhaustive test patterns and the naturally they can be tested concurrently. So, this is one type of testing that is there or exhaustive testing.

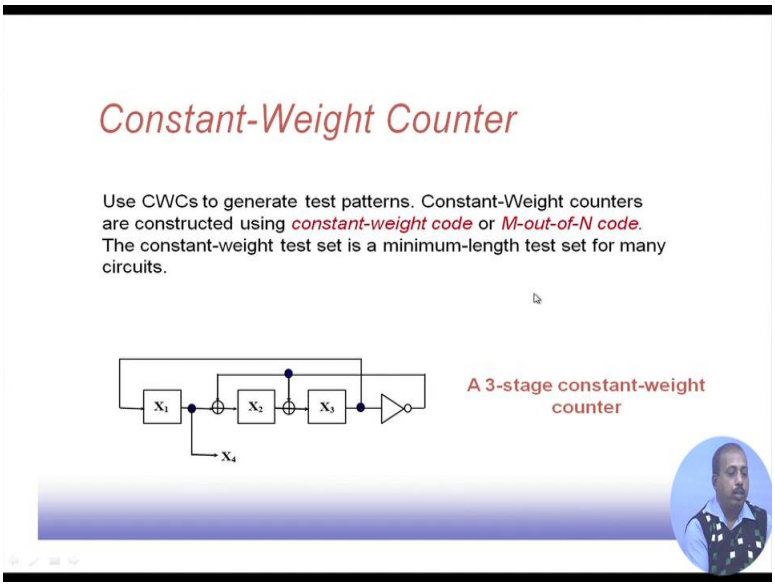
(Refer Slide Time: 12:42)



Another possibility is that we can use some syndrome driven driver counter. So, to generate the test patterns, we check whether some inputs can share the same test signal like if n minus p input can share test inputs with other p inputs then the circuit this can be tested exhaustively with this be inputs only. You see that only in this case if you look into if you look into the circuit, now this x 4 is nothing but x 1 only. So, whatever is x 1 that itself is x 4. Now x 1 and x 4; these 2 whatever pattern will be generated by from this circuit it x 1 and x 4 are going to be same.

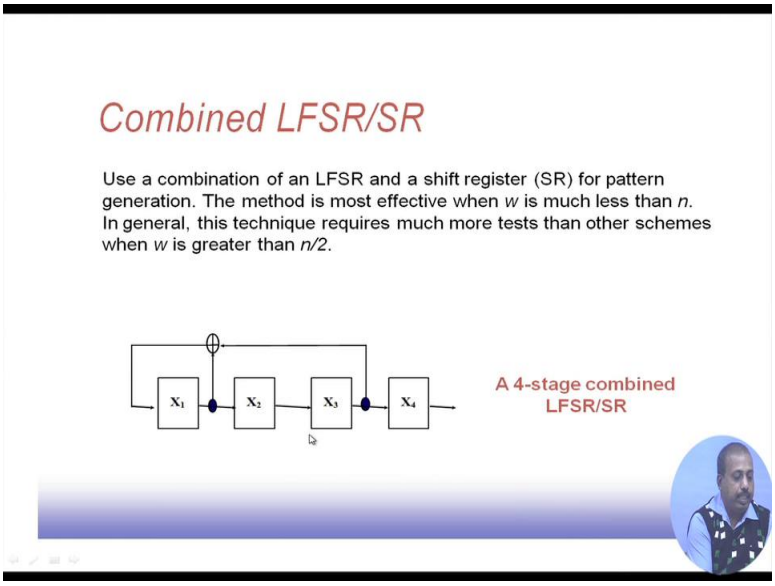
Now, if x 1 and x 4 are going to be same then if it is possible that for the circuit that we are going to check, this I have got a line one which can be shared. So, the line the x 1 the line to which x 1 fits and line to which x 4 fits, they can be shared. So, if they can be shared then 3 stages syndrome counter is sufficient. So, it can generate the patterns and this x 1 and x 4 they are duplicates, but that can feed the circuit.

(Refer Slide Time: 14:00)



Another possibility if there some constant weight counters. So, this is the constant weight counter. So, this uses some M out of N code. So, whatever codes are whatever code patterns will be generated. So, this follows this possibility this one that that out of n positions at N positions. So, it will be it will generate the code. So, this is coming from the coding theory we will not go into the details of that, but this can be generated by LFSR. So, out of m positions as I am you can pick up any M position. So, it will generate a pseudo exhaustive pattern range an exhaustive pattern for the N cells. So, this can also be used for this test pattern generation.

(Refer Slide Time: 14:47)

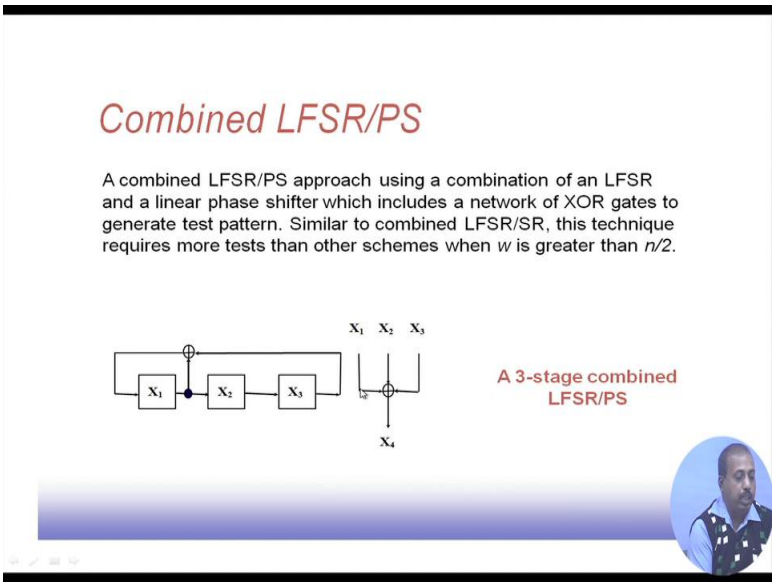


Combined LFSR SR; so here what is done is that first we have got an LFSR and then after that we have got some shift stages. So, this is the; this is a 3 stage LFSR; x 1, x 2, x 3, after that as a 4th stage, we have added another flip flop. So, this is at x 4, you will get a shifted version of at x 3. So, that may be sufficient in many cases. So, this uses a combination of an LFSR and shift register for pattern generation. So, how many such stages will be added? So, that is that is a question and, if w is much less than n. So, w, you remember that n and w, n was the total number of cells total number of inputs and w was the width for which exhaustiveness is needed; width for exhaustiveness.

Now, if you have got this w much less than n, but there are more number of cones in that case what happens is that this x 1, x 2, x 3 so that may be feeding 3 different inputs of the cell or of the circuit. So, that there it is fine, but they are they are generating independent pattern so that that can be fit, but for the remaining input we may be able to feed them through the shifter stages x 4, x 5, x 6. So, we can add a number of shifter stages and for from that shifter stage so we can feed the circuit. So, the technique requires much more tests than other skin naturally and when w is greater than n by 2.

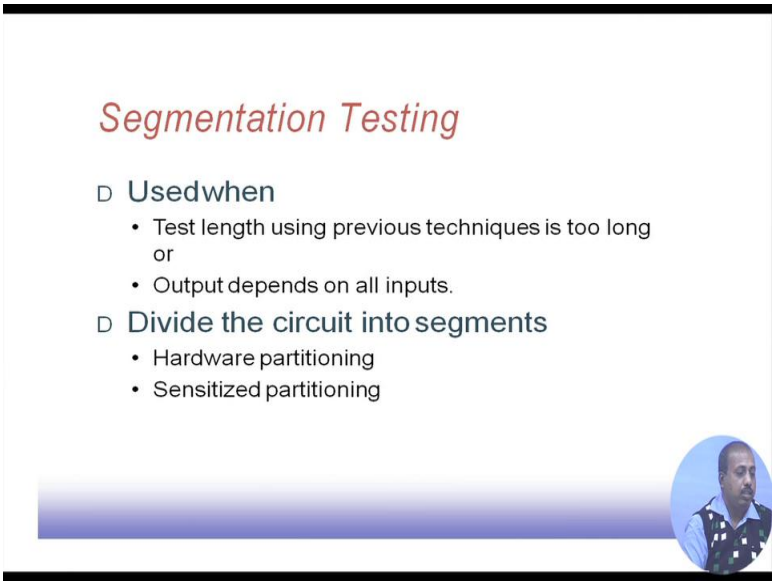
So, if it is more than n by 2 then of course, the there is more number of shift shifting stages has to be added and it will require more number of stage, but if the number of stages number of value of w is less in that case in most of the case it is by adding one or 2 extra shifts stages. So, we can generate the test sequence we can we can generate the exhaustive patterns for the sets. So, we can do some analysis about their dependency and we can find that they are going to be independent. So, accordingly we can feed them with the patterns

(Refer Slide Time: 17:13)



Sometimes we also use some phase shifter. So, this is previously after this LFSR stage. So, we added a shift register. So, now, we added phase shifter. So, what is done? This x 1, x 2, x 3, so they are summed up and this x code input is generated, so this is a combined LFSR phase shifter approach combination of LFSR and the linear phase shifters which includes a network of XOR gates to generate test patterns. So, it is similar to LFSR SR and with more test than when w is more than n by 2. So, the performance of LFSR SR and LFSR PS; they are going to be similar, only thing is that we are instead of using the extra shift stages. So, we are using phase shifter by these XOR gates.

(Refer Slide Time: 18:02)



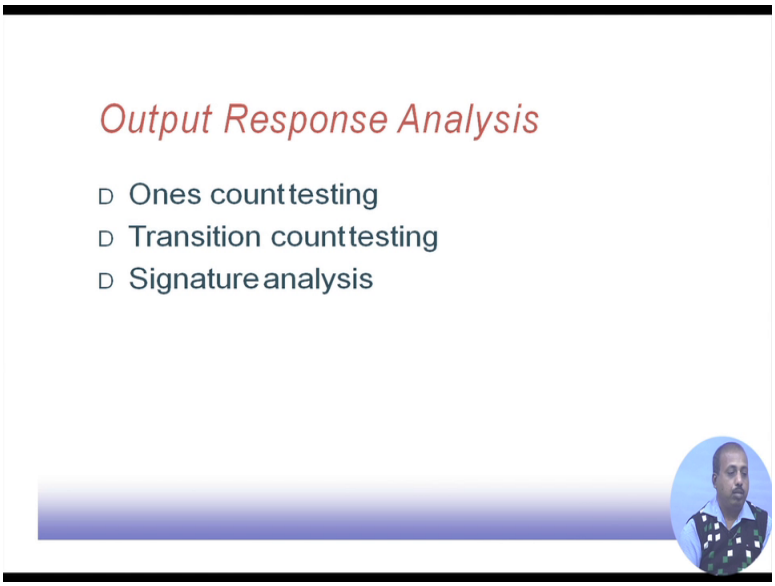
Another one; another strategy that can be followed is the segmentation testing. So, what can happen is that the circuit may be very large and as a result and with the output depends on more number of inputs like here you see that we thought about this policy when this individual output their cone of dependency was much less than the total number of inputs in the circuit, but it if it happens that it is not the case then there is at least one input whose cone of dependency spans over almost all the inputs almost there is one output of dependency stands over all the inputs.

Then for testing this output; we have to apply exhaustive pattern for all this input. So, that makes it combustion. So, all the methods that we have discussed, they will not be able to solve this problems. So, test length will become too high. So, it is in that case what is suggested is that we divide the circuit into segments. So, when the how do we divide like you see that if I in the circuit I may have say 2 outputs this one and this one. So, what we do we put some multiplexer and using this multiplexer.

So, normally maybe the input is coming from a faraway input so that this can include say so what we do we have some other input multiplexer on to it in the test mode in the test more. So, this input is changed to this input. So, as a result, this cone of dependency for this particular input well for this particular output will change. So, the circuit can be. So, in this way, if we can do something so that the circuit gets partitioned, so the circuit is divided. So, one partition corresponds to this output another partition corresponds to this output.

So, we can now partition the circuit and we can do this testing. So, whether it is possible or not that depends on the circuit actually and what we can do we can do some hardware partitioning. So, hardware partitioning it is possible if the circuit is if this hardware can be modified by the designer. So, that this partitioning can be the hardware can be divided into portions or we can do some sensitizing partitioning where physically the partitioning is not there, but by applying some control signals. So, we can sensitize certain parts as a result other input does not come to the circuit at that point. So, that way we can have some sensitized partitioning. So, whatever we do ultimately the idea is that we divide the circuits as if the original circuit is into a number of sub circuits and the sub circuits are being tested separately.

(Refer Slide Time: 21:05)

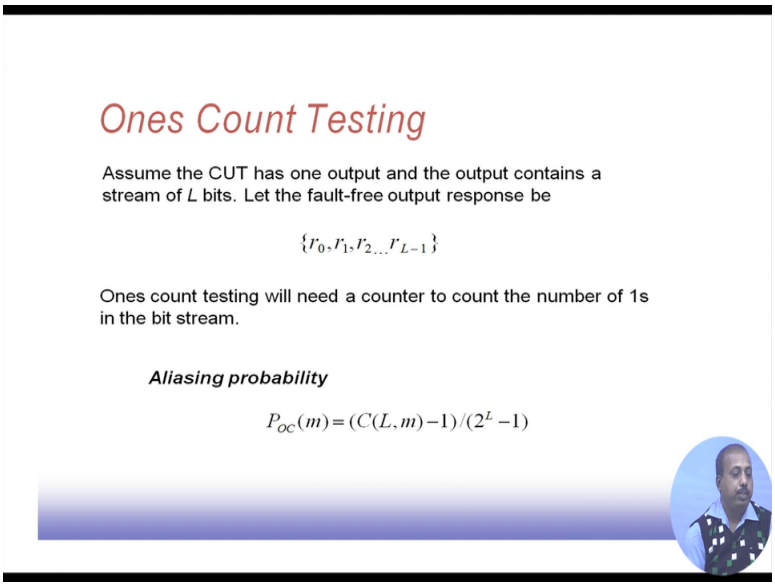


So, that is the segmentation testing next after this pattern generator the next important module that we have in BIST testing is the response analyzer. So, this response analyzer it may do many things like what is happening is that from the circuit at on the application of every pattern generated by the pattern generator some response is generated now we cannot store all those responses in the circuit itself because the pattern generator. So, it was running on in an autonomous mode. So, it was generating the patterns they were getting applied to the circuits, but the responses cannot be stored in the circuit or because that will that will be huge for the sake of comparison. So, what is required is that we need some sort of signature. So, so that after sometime we can we see that.

This is the signature. So, that signature has to be obtained and that signature will be compared between the fault free and the quality cases. So, that are different types of response analysis that can be done once count testing it comes how many once we have seen that maybe one type of signature like if the circuit is good maybe we for set of test pattern we see 100 ones and in the another if the circuit is faulty we get ninety nine ninety five ones. So, when we get 95 ones it does not match with 100 we say that there is some fault in the circuit. So, that is once found testing then there can be transition count testing how many transactions are occurring. So, once we got a 1 and then 0 or 0 followed by 1. So, when these transitions are occurring.

So, what are the transitions? So, that can be counted. So, this transaction count testing can be done or we can have some signature analysis. So, signature analysis is the circuit the whole session will run at the end we will have the pattern that will be available in the response analyzer. So, that is the golden pattern. So, that has to be that is called that will be called signature and we will try to see whether the signature obtained is same as the golden signature or not.

(Refer Slide Time: 23:14)



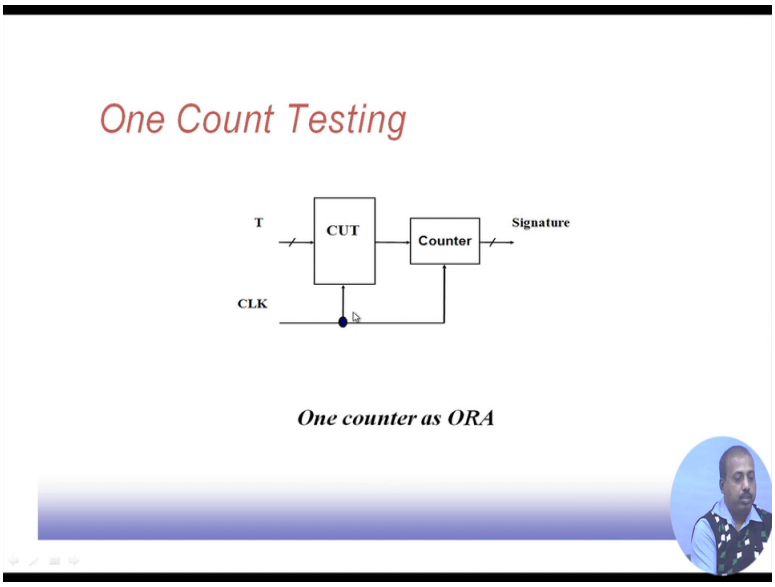
So, to start with say one count testing, so we assume that the circuit has got only one output and the output contains a stream of L bits. So, fault free response; fault free output response is r 0, r 1, r 2 up to r L minus 1 and so we are applying some test patterns say L test patterns and different cases that is only one output. Now this output is giving us in the first cycle it is given as 0 and then it give out r 1, r 2, r 3, up to r L minus 1 then so if this values are fed to counter then the counter will count how many ones were there in the bit stream so that way it is going to be the case. Now you see that if the possibility the aliasing that the situation where see maybe that in my fault free case suppose there are 100 ones. So, my golden value is 100.

Now, it may. So, happen that in the circuit that we are operating. So, in one there are 2 due to some fault f one. So, it reduces this account by one. So, it tries to make it ninety nine, but due to the presence of another fault f two. So, this becomes the transition gets one count increases. So, that makes it 100. So, that way at some point for some input for

the one should for some input where the output should not have been one. So, that has become one as a one account has increased and for some other pattern where the output should have been one, but it has become 0. So, the one count decreases as a result the one count finally, remains at 100.

So, we see that this particular situation is not fault it is a faulty circuit, but the fault is not detected by the system. So, that is called aliasing. So, faults are getting aliased. So, not only aliasing is not only between the fault free and faulty circuit it maybe it is also between the fault. So, it may so happen that a number of faults they generate the same signature the same signature it is a is equal to the golden signature then the problem is more severe because we cannot detect the faulty behavior of the circuit, but even if the even if the signatures are same the signatures are not same as the golden signature, but still the signatures are same. So, that gives rise to this problem of this number of fault getting aliased to the same value and this one so out of this, so aliasing probability is given by this expression. So, the aliasing probability that m will be out of L cases, there will be m ones will be there. So, that will be serial m minus one divided by 2 to the power L minus one this is the total probability of this aliasing.

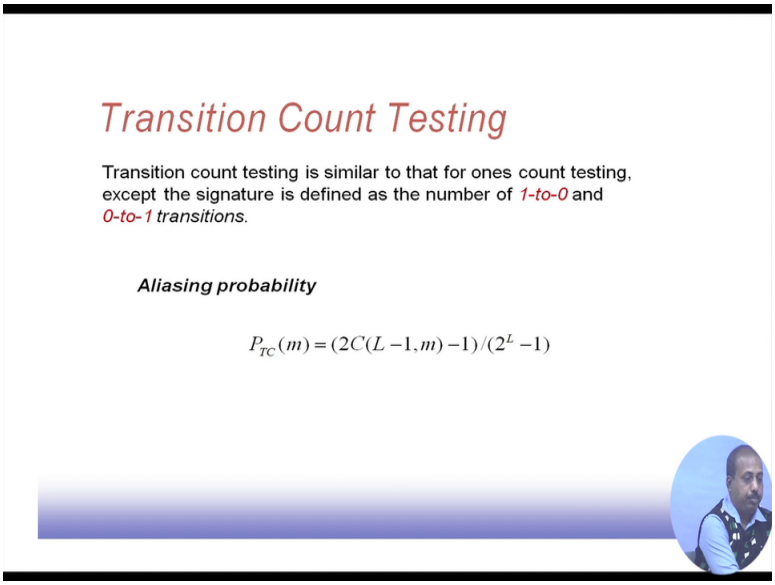
(Refer Slide Time: 26:26)



So, one count testing circuitry is like this. So, we have got this test patterns are applied. So, there is only a single output. So, clock is applied to the circuit as well as the counter.

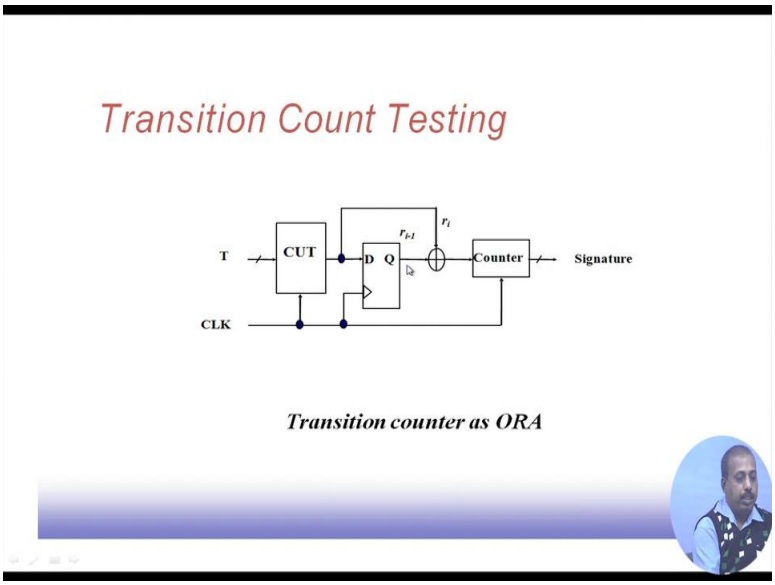
So, after applying all the test patterns, this counter value will have how many ones we have seen at the output and that is the signature.

(Refer Slide Time: 26:47)



Transition count testing; so this is similar to this aliasing count, this similar to this ones count testing except that so we need to count this 1 to 0 and 0 to 1 transitions, so it is not only the values 1 or 1, but it is also the transitions that aliasing probability will be given by this expression.

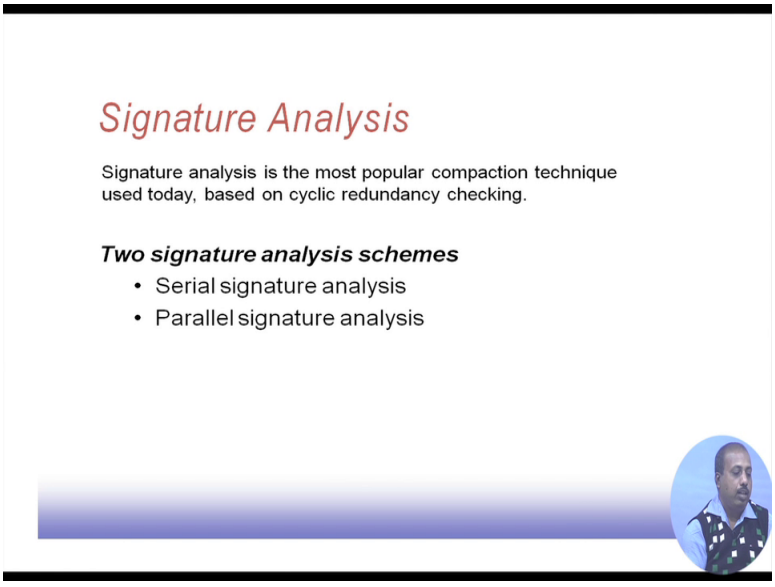
(Refer Slide Time: 27:10)



And the circuit for this will be something like this. So, whenever this circuit output is one this value is latched here and you see that there is a XOR gate.

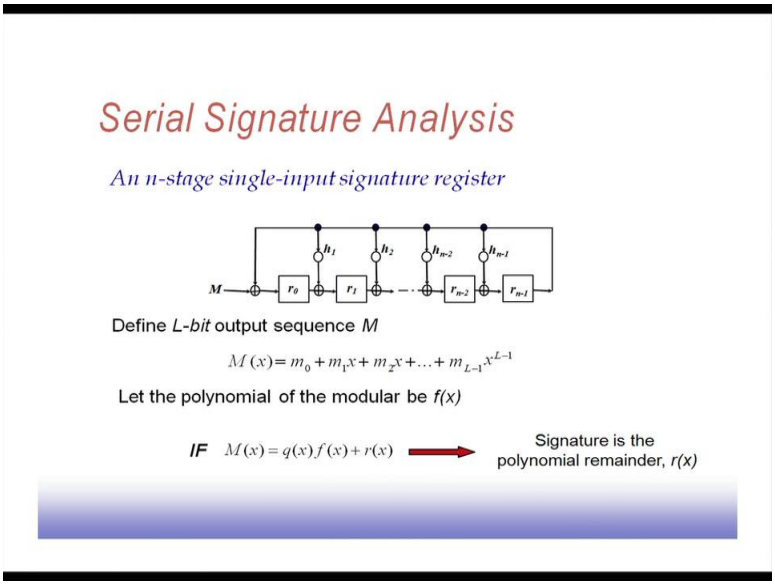
So, in previous value was 1 and this value is also 1 then there is no transition, but if there is a difference between the previously stored values and the current response generated by the circuit. So, this is they are different then the transition will be generated. So, this value will be one as a result counter will count it. So, this count will get calculated and this signature will have the transition count available as the signature.

(Refer Slide Time: 27:53)



Another possibility is it is more elaborate. So, that the signature analysis for this for this type of situation where for the after applying this entire session. So, we have got some sort of circuitry where the content of the circuitry will be collected. So, content of this particular model will get collected and after the after the collection in the end of the session whatever is there in the collection in the collector whatever is the final collected value. So, that is taken as a signature.

(Refer Slide Time: 28:40)



Now, there are 2 signature analysis schemes, one is known as serial signature analysis and other is called parallel signature analysis. So, how they are done in case of serial signature analysis? So, we have got an n stage single input signature register.

So, what is happening is that this L bit output sequence M. So, this is basically the value coming from the circuit. So, circuit is generating some output. So, that is fed serially into this line M. Now this output; this L bit sequence can be defined as m 0 plus m 1 x to x square plus m L minus 1 x to the power L minus 1 and the polynomial of the this is the modular LFSR be f X. Now if M x is given by q x into f x plus r x then the signature of the polynomial is the remainder r x. So, what happens is that when these patterns are actually fed into the structure. So, this polynomial; the f x is going to divide this M x. So, it can be shown that the operation is mimics the behavior of division and whatever remains in this structure r 0 to r n minus one is the reminder and this reminder forms the signature.

So, after all the L bits have been shifted through this registers or through this M, it has come to this LFSR then we can say that whatever be the content of the LFSR so that is actually the remainder and that we use as the signature.

So we continue in the next class.