Design Verification and Test of Digital VlSI Designs

Prof. Dr. Santosh Biswas
Prof. Dr. Jatindra Kumar Deka
Indian institute of Technology, Guwahati

Module - 8
Lecture - 1, 02 and 03
Fault Simulation

Welcome to the next module on testing, so this is module number 8. In the last module what we have seen? Is that we have understood what is the basic of the VLSI testing? Why do you require testing? So the main point of requirement of testing, we saw was that means as the yield of the VLSI circuits lower, because we are continuously moving into higher and higher sub micro technology or advanced technology. So our I mean accuracy or design I means, correctness not yet matured and then we move to another higher high end technology. So, we always this is a chances of failures in the chips or defects in the chips, and our main goal is to eliminate out the faulty chips and then sell out or give to the customer normal once.

So and now the yield is lower, because of the continuously upliftment in technology, so what the main aim is that, we have to do good testing or somewhat VLSI testing kind of stuff is required; so that we can eliminate out the good chips and and discard the faulty chips and for that testing is a important paradigm in case of circuits.

Next we have seen that, a testing can be functional and structural; so in functional testing if there are n input, we give 2 to the power n; all combinations of input and checking as a golden response. So here or mean emphasize is to find that, the chip performance fine or the chip performs ok; so all the combination of inputs, but as you know that, if the number of inputs of a circuit is our 100 or 200; then 2 to the power n is a infeasible number and testing will take years and it will run into means a such huge time that is not feasible.

So we have gone for what do you call a structural testing? So in structural testing, what we do? We are not much concerned about the functionality of the circuit; whether we break the circuit into whole circuit we have seen, we break them up into some small, small modules and each module we takes them functionally and and internal width you can say that, we test structurally.

So in that case we have seen that, if your circuit modules or which we are breaking up into some modules, which we are testing functionally; then if your modules are quite large then what happens? Is the number of test input combination are also higher because the 2 to the power n is also not a very small number; if your some modules are higher, but the advantage that you gain is that the number of intermediate lines between the some modules are lower, and so testing or controlling an observing in this intermediate lines is easier.

On the other hand, if you make small modules of your whole circuit, that is if you have a larger circuit and you have small small modules; than a number of input patterns per module is very less that is an advantage, but then the number of intermediary lines in between the modules is very high and controlling them and observing them remains a challenge; for which you read either pinouts or we require multiplex arrangements or you require a ship register so fore.

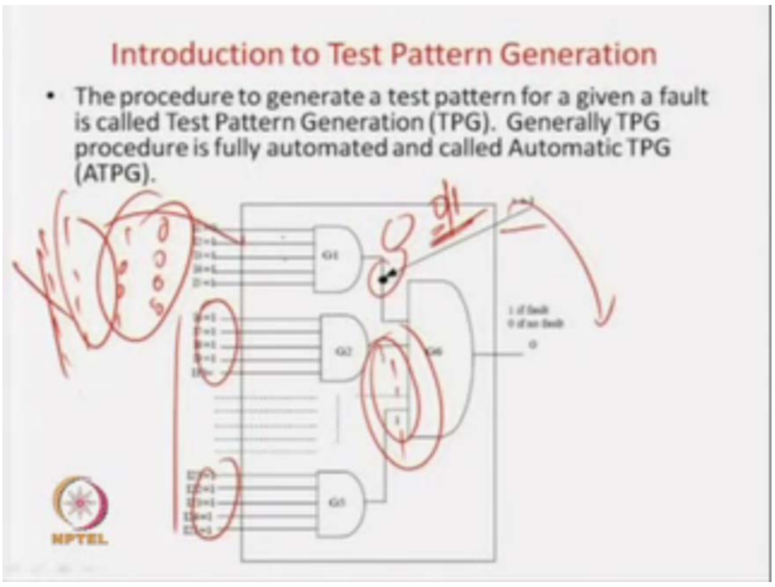
And then we have seen to eliminate out all the problems we have seen; go on for a structural testy using a fault model. What is the fault model? In the structural testing with fault model, we do not concern any we are not concern at all about any kind of functionality of the circuit. Whether we want to find out or whether we want to verify that given a circuit; there is no faults form the fault place. That means you have seen that, stuck at fault model is one of the well accepted fault model; in case which we have seen that, we have given a circuit with AND gets and OR gets; then we have to apply patterns. So that you are able to verify that there is no stuck at faults at any of the lines.

Then we have seen the advantages of stuck at fault models is that, we ignore require intermediate pinouts;3:38 we did not require intermediary multiplexers or ship register to control observe the lines, further the number of one test pattern can be, one pattern can detect more than one faults; so the advantages that, if there are 2 n faults in a circuit where n is the number of lines; still 2 n number of test patterns are not require, we require much less number of patterns because one test patterns can detect a large number of faults.

So there are lot of advantages, when you are going for a structural test with the fault model; as stuck at fault model you will accepted one and also we have seen that structural testing with stuck at fault model is wily accepted because if you can verify that this circuit does not have any stuck at faults, than you can be 99.9 percent plus sure; that the circuit is functionally all you can say that, it can be you can be confident 99.9 percent accuracy that the circuit is free of defects.

So all those things make actually structural testing with stuck at fault model a widely accepted practice or widely accepted model. So in this module what we are going to see that, how can you generate the test patterns? That is given a circuit with n number of faults, then after fault collapsing you can reducing the number of fault; then for the remaining fault what is ever the case, like for a given circuit and there are different faults which are, which are the final list; which has, which has been derived after for collapsing by dominance or equivalence; then how can you generate test patterns? Which can detective that is called test patterns generation. So in this module or in this, in the module number 8; a main emphasize will be how do get patterns which can determine, which can or in the other words, how many type of pattern which can detect this stuck at faults, which are in your circuit.

(Refer Slide Time: 05:23)



Introduction to Test Pattern Generation

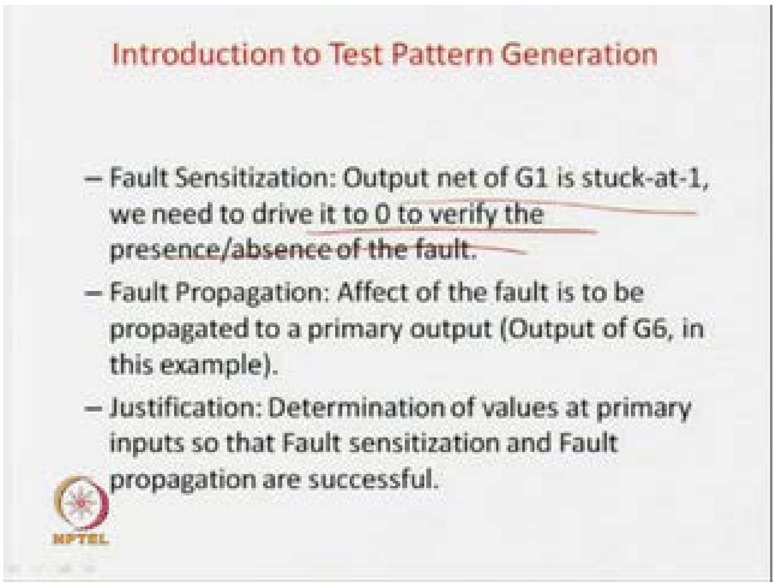
- The procedure to generate a test pattern for a given a fault is called Test Pattern Generation (TPG). Generally TPG procedure is fully automated and called Automatic TPG (ATPG).

So in the first lecture from this is on fault stimulation and it will this a quite long lecture; so you can see will be spending 3 days or you can called 3 modules and 3 sub modules, 1 2 and 3 will be your fault stimulation. So let us explored, so this is our so what is test pattern generation? Just I told you given a stuck at fault, if you get this is a fault; say they automatically you have to find out, which pattern can test this fault. So this is called actually test pattern generation; this called test pattern generation and now will this circuit is quite large, than you need to obviously automotive and therefore it is called automatic test pattern generation; that is called ATPG.

So in this module we are going to look into details of this one; say for example, this is your circuit, which you have also seen in our last lecture. Now say this is a stuck-at-1 fault; now see I want to find out the pattern which can detect this one so, that is my question that is the main problem of ATPG. So now you also try for fault at this point, and fault this point and so for obviously, the number fault will be limited by the fault collapsing algorithm.

Now actually this is a 3 process stage so, you can also find it out as the what do you called adopt; In a adopt also you can find out for example, you can say this stuck-at-1; so obviously, we have to apply a 0 and then this value should be propagated this affect, fault are which propagated to the AND gate. So all these should be once and then actually this the fault affect is you have apply a 0 normal case and the stuck-at-1 to faulty case it will be 1 and this fault affect has to be propagated the output; so all the other inputs of the AND gates are to be 1. Then if all the AND gates output are to be 1; then all the inputs have to be 1 for all the gats and for this case you have to apply, you have to get a 0 over here. So either the inputs can be all 0s or it can be 1 0 0 1; accepting 1 1 1 1 all ones, any other pattern can be applies so, that will be a test pattern generation.

(Refer Slide Time: 07:22)



Introduction to Test Pattern Generation

- Fault Sensitization: Output net of G1 is stuck-at-1, we need to drive it to 0 to verify the presence/absence of the fault.
- Fault Propagation: Affect of the fault is to be propagated to a primary output (Output of G6, in this example).
- Justification: Determination of values at primary inputs so that Fault sensitization and Fault propagation are successful.

That is the test pattern for this fault, that is any other combination other than 1 1 1 1 and all this have to be 1 once; accepting all once at these gate, any other pattern like 0 1 1 1 1 for any other pattern accepting all once will takes this fault, but there is a proper algorithm we required do this, this is actually called fault sensitization propagation and justification.

Let us see what it does, so first is called the again just like a heuristic we have discuss same thing so the first is called fault sensitization. So what do you mean by fault sensitization? We say that the output net of G 1 is stuck-at-1; we need to drive it to 0 to verify the presence or absence of the fault. So just you can take the analogy of a say electric bulb which is fuse; with the so now, if you want to test it you have to make the switch on. So if you switch the bulb on than if it is fuse, than the bulb is not good; that means what, in that case you are sensitizing the fuse fault; so bulb is fuse, so it the fault was sensitization.

So in this case, they are saying that you are sensitizing it; so sensitizing means if it is a stuck-at-1 so, we have to apply the reveres value; so I have to apply a 1 over here, so that is actually called the sensitization. Now next step is actually called the fault propagation; now as already discuss in the last chapter or I mean last lecture, that for structural testing with fault models; you did not need to bring out any extra pinch, neither you require any kind of a multiplex in the neither you require what do you called a shift register kind of a thing.

That is if you have a circuit and then you have a lot of gates inside and you are going for structural and these are the primary outputs and these are the primary input, then we do not will not require any kind of extra pinouts or extra what you called this ship registered to control this, like these are not require. So in structural testing with structural fault model, structural testing with fault model; so what is our idea is that, whatever you have to do; we have to do it from the primary inputs and the primary outputs. That is way what we have to do? These faults affect so, what we have seeing? This fault affect the fault affects is what in the normal case? Sorry this is a stuck-at-1; so you have to apply a 0 I am sorry, so the affect this is one; this is actually 0 in case of fault and sorry normal fault 0 and fault is 1. This affect has to be propagated to this output.

So because whatever you do you already do at this output, we cannot have this pinouts; this p knots is not allowed, no p knot is infect extra p knot is allowed. So we have to bring this out, now in the output also affect will be this one so, already it is 1 each fault time 0 is no fault; now this is actually called the fault propagation, now the fault propagation is propagated. Now we have to justify that, what you mean by justification? Just mean determination of values at primary inputs, so that fault sensitization and fault propagation are successful. Let us see what does it mean? So again I have told you, we do not have any multiplexer arrangement here; if not have any multiplexer arrangement here, if not have any multiplex arrangement here. So indirectly what is happening? You cannot control this, likes this ok.

So I mean to have this fault value propagated from this point, this point to this point; you cannot go for any additional control here. So whatever control you have to do at these lines and these lines, you have to do only from the primary input. So structural testing is fault model is nothing but controlling through the primary inputs and observing through the primary outputs. So now you have to sensitization propagated justify, so you have to justify that this affect or this input and this input is by controlling the primary inputs; we have to justify that, these things are successful.

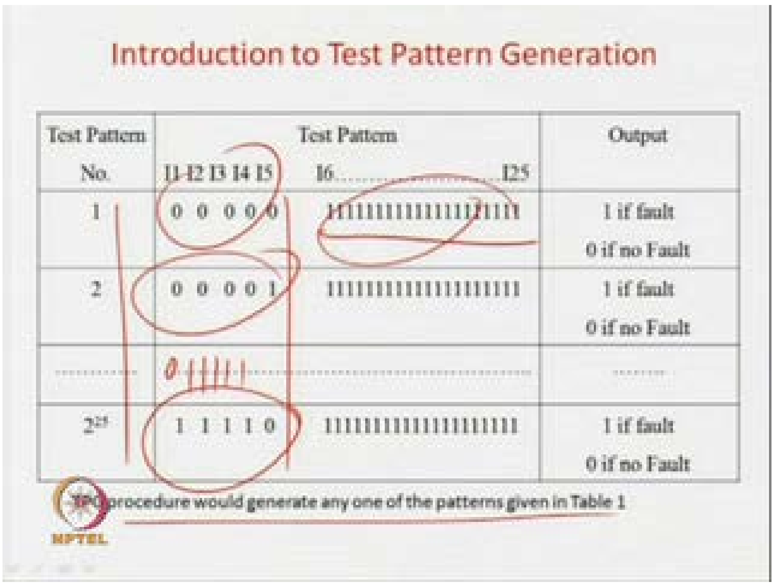
So this is an AND gate, just go from one level; so next level is that, this affect has be propagated to these one. So if this affect has be propagated, all the other inputs of this AND gate has to be 1. So this is the level 1, level 1 justification; this is the level 1 justification right. Now again, now what happens? So again the level 2 justification is saying that, output of this gate is 1; output of this gate is 1 and now output of this gate is 1. So how can you get this justification of the second level to be once? All inputs of AND gate has to be 1 similarly, for all these has to be 1.

Now remains this one, so in this case you require a value of 0. So this has to be justified at the level 2; so again justification AND gate output to be 0, is all pattern or any pattern accepting all once. So your ATPG algorithm will generate 0 1 1 1; it can also the generate 0 0 0 0 0 0 any one of that, that is arbitrary and that depends on the some level of few restricts or so accuracy that will see in the, when will be giving in details or it will see it will written by, whether you will generate 0 1 1 1 or all 0s or 0 0 0 0 1; it depends on near that for kindly the ATPG generation algorithm is justify this 0 by any pattern, which is 0 1 1 1 0 0 anything. So in this case it is assume that, this is the pattern that is 0

1 1 1 has been generated and for justification of 1 1 in this case we get 1 1 1 1 1 1 1 1. So the test pattern generated is 0 1 1 1 and all other inputs are 1 1 1; so it test are stuck-at-0 fault sorry stuck-at-1 fault here.

So this is the basic step, steps of fault sensitization propagation and justification; so this so for all the faults if you do this, you are going to get what you called a test pattern generated for all the faults. Now when all the test patterns are generated, they are stored in the memory; so when this IC is fabricated and it comes from what you called fabrication unit so, each IC you have to apply those inputs and you have to verify, whether none of the faults stuck-at-faults should be presence. So if you apply all the test patterns so, automatically we will find out that, if none of the test patterns is giving a otherwise result; that it is if all the test patterns are verifying there is no fault. So you can say that, your circuit is free of stuck-at-faults and it can be shift to the market.

(Refer Slide Time: 12:21)



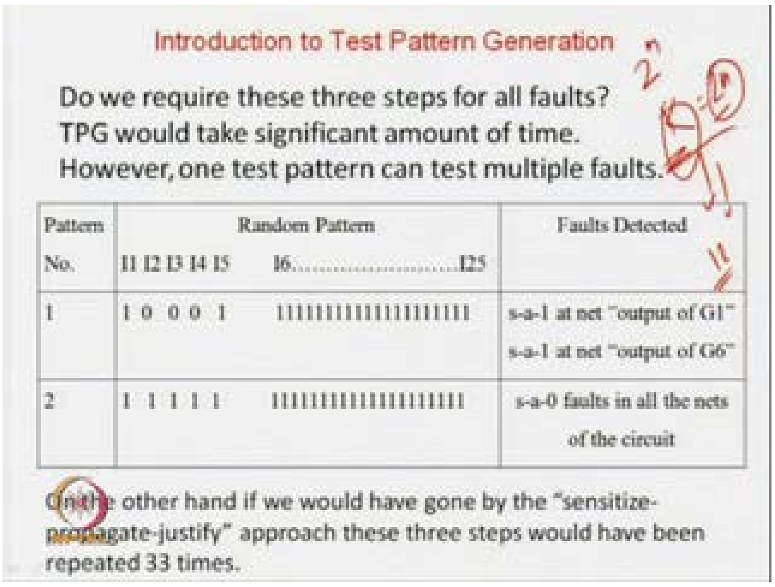
Introduction to Test Pattern Generation

Test Pattern No.	Test Pattern I1 I2 I3 I4 I5	I6.........I25	Output
1	0 0 0 0 0	11111111111111111111	1 if fault 0 if no Fault
2	0 0 0 0 1	11111111111111111111	1 if fault 0 if no Fault
....	0 1 1 1 1 1	....	....
2^{25}	1 1 1 1 0	11111111111111111111	1 if fault 0 if no Fault

TPG procedure would generate any one of the patterns given in Table 1

So as already told you so, the test pattern generation procedure would generate any pattern in the table 1; it can generate all 0s with all 1 so, this can be 1 1 1; it can be 1 1 0. So 2 to the power 25, if minus 1 test patterns can be possible for this stuck-at-fault and it depends on the ATPG algorithm, whether will generate 0 0 0 0 for the first AND gate or 0 0 0 1 for this one or 0 1 1 1 1 1 for the first gate so it depends; for anything would do your job because we require 0 at the output of the first get and all 1s and once at the output of all the other AND gates right.

(Refer Slide Time: 12:55)



Introduction to Test Pattern Generation

Do we require these three steps for all faults?
TPG would take significant amount of time.
However, one test pattern can test multiple faults.

Pattern No.	Random Pattern I1 I2 I3 I4 I5	I6.........................I25	Faults Detected
1	1 0 0 0 1	11111111111111111111	s-a-1 at net "output of G1" s-a-1 at net "output of G6"
2	1 1 1 1 1	111111111111111111111	s-a-0 faults in all the nets of the circuit

On the other hand if we would have gone by the "sensitize-propagate-justify" approach these three steps would have been repeated 33 times.

Now the now that question arises, that do your required to do this individually for all faults. So now whatever our goal in the last module we have seen that, our goal is reduce the test generation type or the test application type; so you have to go down in time. So from 2 to the power n input patterns we came to K, where K is the number of stuck-at-faults, that is 2 n; so if there n lines so there 2 n number of test pattern is stuck-at-0 faults can be possible and if there are 2 K number of sorry, if they are n number of lines and 2 n number of stuck-at-faults can be possible.

So K patterns that mean for each stuck-at-faults if you assume in the most cases one pattern required, so k pattern that is 2 n pattern is required in the maximum case, but again you can see that by structural fault collapsing by dominates and equivalence you find that much much less than even K; so it is K by affect, K is also much much reduce than K. So even if there are n lines in the circuit, number of test pattern I mean stuck-at-faults are much much less than 2 n; because the collapsing also test pattern requires even much less than that.

So now our now again we are going another level down, so that even if we have some say P number of faults, after start after all the fault collapsing so, we require at list P number of test pattern should detected because 2 n is a number of faults or for the 2 n is the number if n is the number of nets in the circuit; so at list we have 2 n stuck-at-faults, stuck-at-0 and stuck-at-one for each line. Like collapsing we have reduce the 2 P; now can we go even less than P, can you have any number of can you have can you declare say that, the number of test pattern requires are even less than P; the answer is yes because sometimes will find out that one test pattern can detect more than one number of faults, that also can be possible.

So that is actually called mean what you called serial pattern can detect multiple number of fault. So therefore if you are repeating, if you are taking the approach of sensitize propagated and justify; so what you are doing? You first take one fault then we do a sensitive propagated and justify and generate one pattern; say P 1 you are generating for this one. Now we take another pattern, do sensitization propagation and justification then you get pattern P 2; similarly, you keep on doing this keep on doing this and after that you say gate, say gate P l is the number of fault. The l-th number of fault you do and you generate say P K, P l sorry P l is the pattern; sorry what do you called i have say that, you have fault P 1; then you have fault P 2; then you have fault P l.

So for that you are generating say P P 1, this pattern 1; than for P fault number your generating pattern number 2 and here you generating pattern number K fault fault P l K, but now you find out that, P l 1 that is sorry that is fault P 1; if the pattern you are saying P P l 1 is equal to P P l K, P P l K or P l, that is for the l-th number of fault and the first number of fault the same pattern is generated by this a automatic test pattern generation, that is sensitize propagate and justify. So you are at a loss because the same pattern that is at P P a 1 or that the pattern for the first fault you are generated; that is what is happing in that, this same pattern is detecting more number of fault; that is the first pattern is detecting fault number 1 as a fault number l. So unnecessarily we are wasting time in automatic test pattern generation using sensitize propagate and justify, where the same pattern can detect multiple number of faults.

So let us think the try to think it in some other way; can i do something? So that we apply one pattern and find out how many faults are detected and then remove those faults and then try to apply another pattern and see how many faults are detected and so forth. So in this case, what will save? Will save the unnecessary load of the case, where one fault can detect, one pattern can detect multiple number of faults.

Like in this case, pattern number one is detecting 1 as well as pattern what do you call the fault number l; so there is a redundancy, but if you have say that if I apply say pattern

P P 1 and then you find out which faults are detected, then automatically you would be able to tell that detects fault P 1 and also detects fault P 2; sorry P l. So in this case you will be saved from executing in the same algorithm twice for P a fault number 1, fault number l. So will see that, so that is actually called fault detection by random, what do you called random pattern generation; so in this case sensitize propagation and justify by this approach, when you are going for test pattern generation it is called a deterministic approach. In this case you are sensitizing the fault, propagating the affect and you are justifying this and when you are doing randomly you get apply a pattern and then see how many faults are detected and again repeat this, then we call this a random pattern ship registration.

So will see that, so both of them we have them own advantage so you will see; as of now it appears that which is more advantages? The more advantages is the random pattern generation; you apply a pattern and see how many faults are detected. Let us conceder the same circuit, if you consider that will take the same circuit here; the same circuit we are going to consider. This is the same circuit we are going to conceder and now say what can you do? Say all some faults are there, some stuck-at-faults; stuck-at-0 faults are over there. So now what you do? So you say randomly you apply this pattern. Just all of a sometime you dream that, this is very good pattern you apply; than you can find out that, what are the faults are detected? So you can find out that, more than one fault will be detected by this pattern. So 0 0 0 1 1 1 1 1; so this pattern let us see what happens?

So in this case you are applying all 1s and here you have applying say 1 0 0 0 1 some pattern you are apply. So which faults are detected? Obviously, this is 1 1, so this is 0over here; this 1 1 and this is something. So obviously a stuck-at-1 fault is detected over here; so obviously, a stuck-at-one fault will be detected by this because all lines are 1 and you apply the pattern 0 0 0 0 1, this is 0; so obviously, stuck-at-1 fault is there. Similarly, the same thing you can also save that, it is also detecting stuck-at what do you can say is also detecting is a 0 over here; so obviously, it is also detecting a stuck-at-1 faults over here because in this case the answer is 0, but you can stuck-at-1 fault; so the answer will be this one. So similarly you can see that, one random pattern is detecting these two faults.

So we can also have more examples of similar nature; so in this case, now if you see that I apply this pattern. So in this case it is so it is so getting two faults, stuck-at-1 at the net of this one and stuck-at-1 at the net of these one; two faults is taken. Now so the next random pattern is this one, all once; so all once which one is going to happen, so if you are applying all once over here, all once; than obviously stuck-at-0 fault at any point of the circuit is detected. That is stuck-at-0 over here will be detected; a stuck-at-0 will be here detected; a stuck-at-0 here will be detected and so forth.

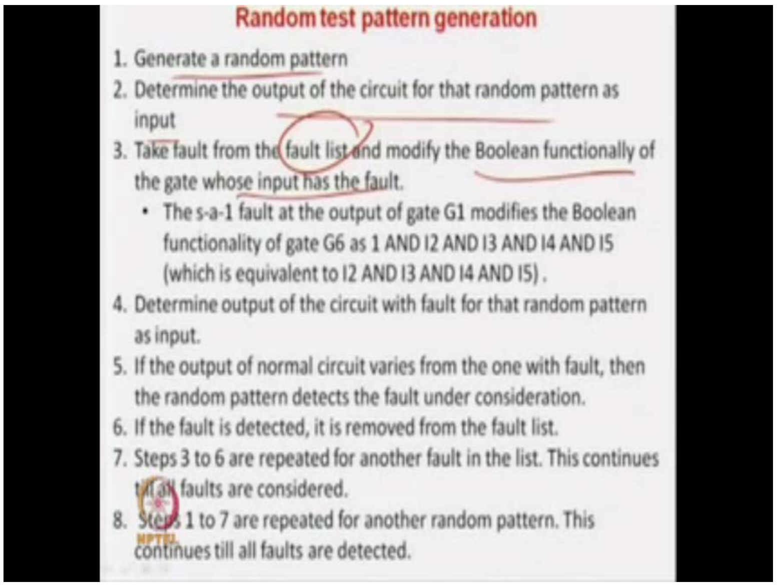
But, if you are going for a if you take the example, let means just elaborate on this part. So by fault collapsing and all those things so, we have seen that generally where are going to have stuck-at-0 at 1 input of the AND gate and all stuck-at-1s will be there; all stuck-at-1s will be there right. So in this case also one stuck-at-0 will be there for one input and stuck-at-1 at all the input lines will be there; all AND getting will having as only one will have a stuck-at-0 by equivalent.

Now the random pattern generate is all once; so you can easily find out that, if we are applying all once, then what can happen is that the same pattern detect a stuck-at-0 fault here; it can also detect stuck-at-0 fault here and also it can detect a stuck-at-fault, stuck-at-0 fault here. So one random pattern can detect stuck-at-0 faults at all lines in the circuit and if you consider fault collapse, we will not have stuck-at-0 faults at all the lines of the circuit, but will have one stuck-at-faults at each input, each AND gate input; so if there are 5AND gates so, you will be able to detect 5 stuck-at-0 faults by the same pattern.

So random, but if you are using the sensitize propagated and justified approach, then for the 5 stuck-at-faults at this points say, at this points say and this point say; so what you have to do? You have to first apply a 1 over here. Like for example, if you take this approach; so it is not simple one, so if you have the stuck-at-0 fault over here; so you have to apply a 1 over here; then you have to propagate this value to be here; so you want to propagate the value here so, all other will have to be 1, than sorry you have to propagate the value here, so it will be in case of normal one faulting case 0 and then again the affect has to be propagated from here because you are testing the stuck-at-0 fault here. So one you have to apply this is the normal one fault 0, normal one fault 0.

Now you have to this is the propagation i have to justify so these lines has to be one right and then if this lines has to be all others has to be one, all other has to be one and here also everything has to be one if you want to apply. So again you have repeat this same thing for a stuck-at-0 fault here and stuck-at-0 fault. So this is a long procedure so, it is a long procedure that is the problem, but now by random pattern generation it is very simple between apply this pattern and you can find out that, we can detect a much more larger number of faults; in this case you can see 33 number of patterns has been, faults has been detected by this approach.

(Refer Slide Time: 21:48)



Random test pattern generation

1. Generate a random pattern
2. Determine the output of the circuit for that random pattern as input
3. Take fault from the fault list and modify the Boolean functionally of the gate whose input has the fault.
   - The s-a-1 fault at the output of gate G1 modifies the Boolean functionality of gate G6 as 1 AND I2 AND I3 AND I4 AND I5 (which is equivalent to I2 AND I3 AND I4 AND I5) .
4. Determine output of the circuit with fault for that random pattern as input.
5. If the output of normal circuit varies from the one with fault, then the random pattern detects the fault under consideration.
6. If the fault is detected, it is removed from the fault list.
7. Steps 3 to 6 are repeated for another fault in the list. This continues till all faults are considered.
8. Steps 1 to 7 are repeated for another random pattern. This continues till all faults are detected.

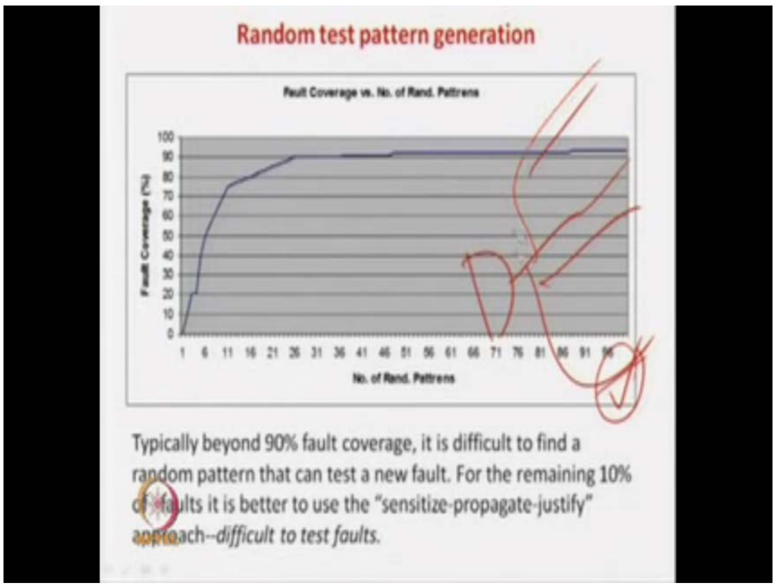
So as of now it seems that, this is a very good idea that you go for a random pattern generation base testy; rather than go for sensitize propagate and justify. So what is the idea of say look at it so, what is the basic algorithm? You check a random pattern, any random pattern; then you determine the output of the circuit in the random pattern; in the normal circuit do not conceder any fault, but determine the output of the circuit for that random pattern.

Now take one fault from the fault list and modify the circuit; so for example, if the stuck-at-1 fault at the output of G1. If you take say for example, the same circuit. So in this case if you are taking as tuck-at-1 fault here so, the output of this AND gate functionality will be output of G2 and output of G3 dot dot dot output of G5, because this is stuck-at-1; so this gate input has no affect. So the Boolean function will be modified as, O of G2 and O of G3 and O of G4 and G5 because this is modify; because this already stuck-at-1 so, this has no control I will affect on the AND gate.

So that is modified; so it is, it is 1 that is already there AND of I2 I3 I4 and I5 which is equivalent to this one, 1 is remove; so Boolean function is modified. Now you determine the output of the circuit, there is a chain circuit; the circuit with the fault for the random pattern for the same random pattern as input. If the output of the normal circuit varies from the once with fault, then the random pattern detects the fault under consideration. So what you have done? We have taken the circuit, we are applied a random pattern and then we have recorded the output.

Now what we do now? Now we change the circuit with the fault; now apply the same same random pattern and if you found find that the output varies, then that pattern random pattern detected that fault. Now, we next what you do? We again we delete the fault and say that it is detected, than we take another fault and find out, find out what is the case with the same random pattern, it takes the another fault or not. So step these are repeated for other fault in the list, this consider these all the faults are listed and this how it is it the random pattern goes.

(Refer Slide Time: 23:49)



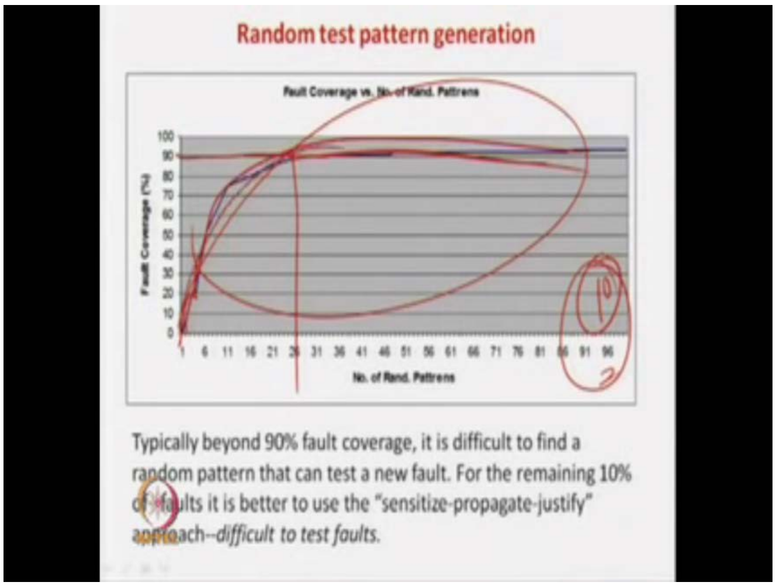
Random test pattern generation

Typically beyond 90% fault coverage, it is difficult to find a random pattern that can test a new fault. For the remaining 10% of faults it is better to use the "sensitize-propagate-justify" approach--*difficult to test faults*.

And you can easily see that, random pattern approach is very easy; just you have to apply the pattern and just you have to find out the output of the circuit. It does not require any kind of like sensitization and one thing I should find out which will be discussing in details in the future lectures, that always sensitize propagate and justify approach may not be successful.

That is you may be able to sensitize a fault propagate the value of the output, but you may not be able to justifying; then again you have to recreate and find out and the way of propagating the fault output, may be there multiple fault to propagate the fault to the output; see for example, if there is a AND gate over here and the fault can be propagated through here; the fault can be propagated through here; the fault can be propagated through here. So you may try out with this one; there what may happen is that you may find out that, you may not be able to justify it or you may not be able to propagate it through this; then you have to try to this part and you have to try to this part.

(Refer Slide Time: 25:00)



Random test pattern generation

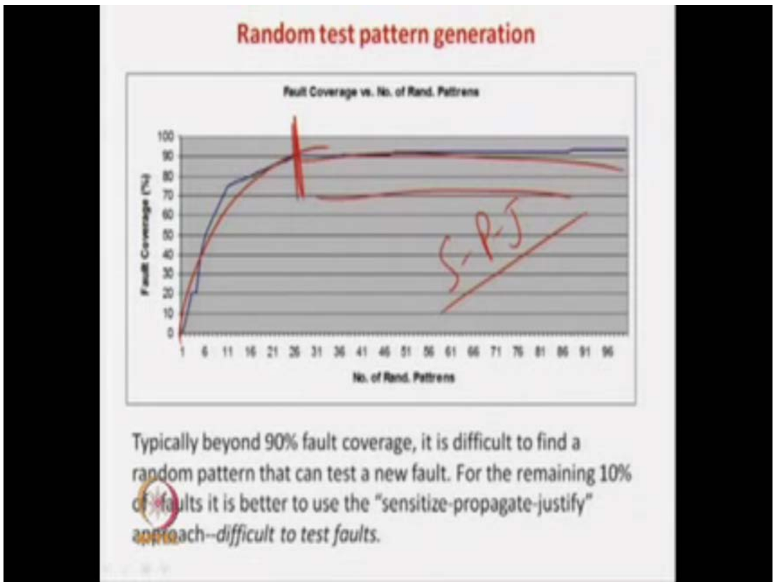
Typically beyond 90% fault coverage, it is difficult to find a random pattern that can test a new fault. For the remaining 10% of faults it is better to use the "sensitize-propagate-justify" approach--*difficult to test faults.*

In other words there can be requirement of lot of back tucks, if you are going very sensitize propagate and justify approach. So in that way the random sensitization is very much efficient algorithm, in which what, what case in what we do is that, we apply the test pattern out, find out the output of the circuit and then we modify the circuit with the fault and see if there is any different; that is a very simple idea. Now the question is then do you require the sensitize propagate and justify approach, the answer is yes.

So statistically this graph has been found; so what is they are saying that, if there are if the circuit has say 100 faults; than what you do? You apply the first random pattern, than if the detects 20 number of faults. Next random pattern you do is say that, generate another 20 number of fault; this keep on goes and then it saturated, that means what? Up to around 90 percent of the faults, you can get a very good fault coverage per random test pattern that means, what you apply one pattern, it detects 10 faults; you apply another random pattern, you get another 10 faults; then you have apply another random pattern, you get 8; if you keep on doing it after that this some faults which are call difficult to test pattern.

By applying random patterns, you may not be able to test this fault or in other words, you apply 10 random patterns; you detect 90 faults. For the rest 10, 10 faults which you are calling as difficult to test pattern, difficult to test for. You have to applying same more than 30 numbers of random patterns; that is first random pattern you have to apply, after 90 no fault is detected. Here the another random pattern, none of the 10 faults are detected; you keep on doing it, say up to 31 random pattern say the fault one of the fault get detected and you keep on doing it so, you have to apply a great number of random patterns to detect those last 10 number of faults. So whenever you find that the numbers of random patterns or are getting saturation or you are saying that it gets saturated; that is the efficiency of random test pattern generation get satisfied, I mean saturated.

(Refer Slide Time: 26:12)

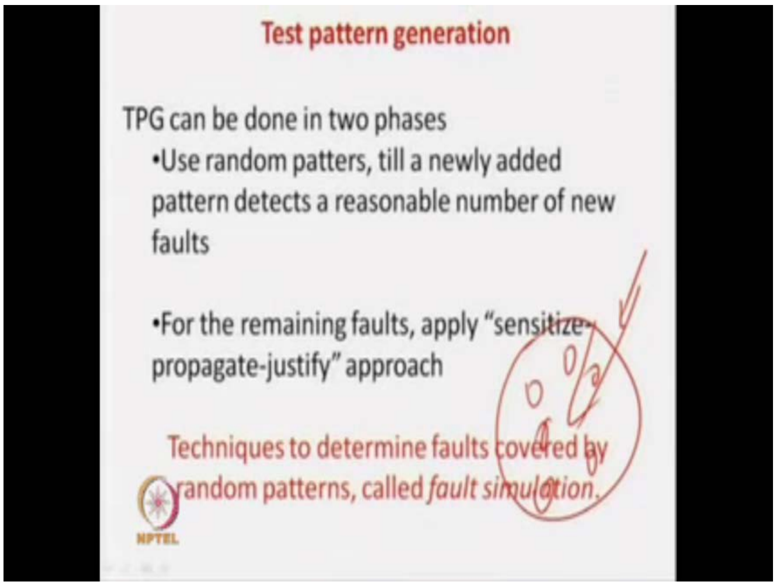


Random test pattern generation

Typically beyond 90% fault coverage, it is difficult to find a random pattern that can test a new fault. For the remaining 10% of faults it is better to use the "sensitize-propagate-justify" approach–*difficult to test faults*.

So have say 90 percent of the faults for the raise 10 percent of the faults; you find out that, after applying 10 random patterns one fault is getting the detecting. So after that, it is actually become more difficult than the sensitize propagate and justify approach; then what we do? After that you stop random pattern generation approach, that is when we find that 3 consecutive random pattern or four consecutive random patterns are not able to detect any leave fault; than what we do? We stop at that point and from this point, we term that term they may have difficult to test faults and for them what we do? We apply sensitize propagate and justify approach of fault.

(Refer Slide Time: 26:50)



Test pattern generation

TPG can be done in two phases

- Use random patters, till a newly added pattern detects a reasonable number of new faults
- For the remaining faults, apply "sensitize-propagate-justify" approach

Techniques to determine faults covered by random patterns, called *fault simulation*.

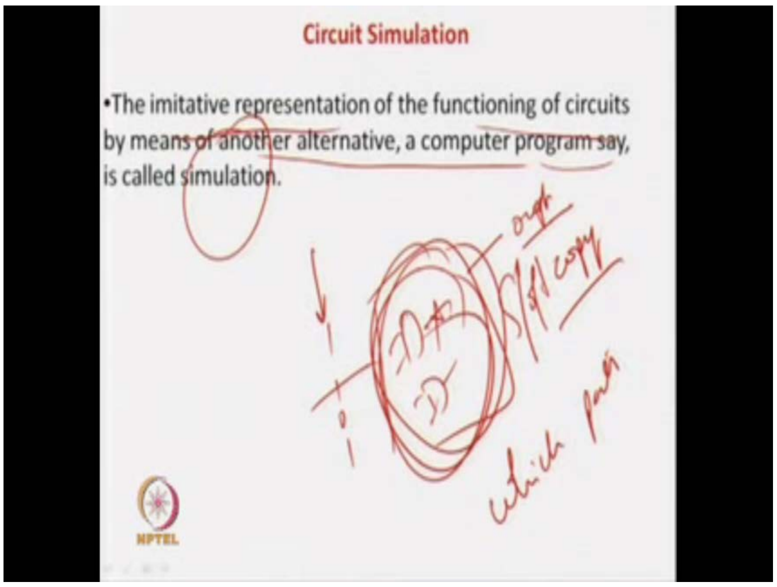
So you cannot say that, sensitize propagate and justify approach is a very, I mean way difficult approach. So we should through away with it and we should always go for I mean, what do you call random pattern generation is not the case; for the first few faults for the 90 percent of the faults random pattern is very good because you can detect large number of faults by the I mean, just applying a random pattern and verifying which faults are getting detected, but once you get a saturation after that the random pattern generation becomes un efficient; because for one pattern you may not be able to detecting anything any of keep on doing it, to do get a fault. So for those faults is greater to go for sensitize propagate and justify approach.

So we can give a very simple analogy so, I think all of you might we gone to pears or meals; where you have balloons. So where this, lot there is a bold where they lot of balloons over there and you have to shoot with a gun; so what you can do it with a lot of balloons in the board? That you can randomly shoot and some of the balloons can blasted that is true. Even if I do not have a very good aim so, what I can do? Either I can randomly fire the boletus and most of the balloons get blasted or at list few of the balloons get blasted.

Now why it happens? Because, there is more number of solutions or more number of faults and randomly firing gets the, gets our answer. The same thing happens here, more number of faults are there; so you can just apply random patterns, you can detected the faults, but when the number of balloons remains very less, say most of the balloons you have blasted and a few say 5 or 6 balloons are remaining. So at the time what is what happens? At the time it is very difficult than you have to aim and then find out how far is that, how how how you are holding your gun and all those precautions you have to take care and then only you can blast the balloon.

So is same analogy here; so were lot of faults are there you can randomly do all the thing but whenever the number of faults are coming to be less, than what you have to do? Then you have to go for a very aimed approach, which is called the sensitize propagate and justify; so same analogy here. So in the first we will go so actually test pattern generation done in two phases; first we go for random pattern with a, whenever a random new pattern detects are reasonable number of new faults, when you find that not being done; then we go for a sensitize propagate and justify approach or detecting the faults. So this is actually the second phase so which will be dealing latter in the part of the course, but now will see how we can efficiently do this?

(Refer Slide Time: 28:56)



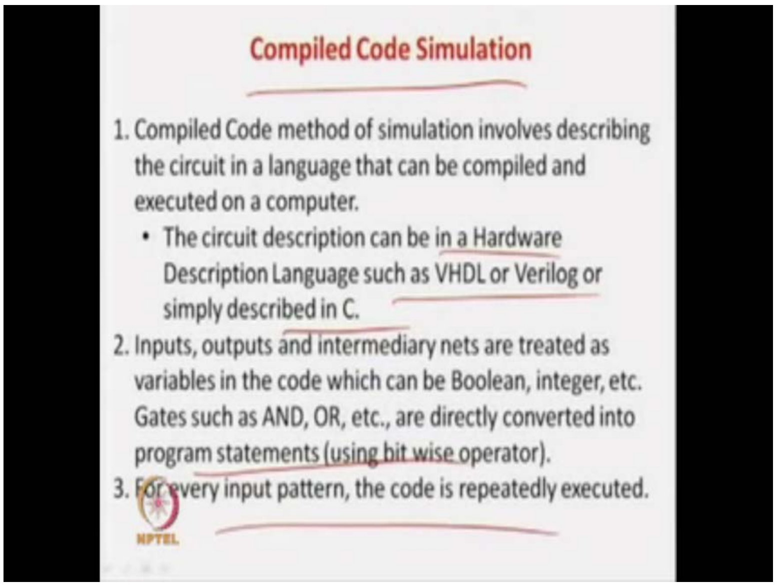
Circuit Simulation

- The imitative representation of the functioning of circuits by means of another alternative, a computer program say, is called simulation.

So for what do you called for we have seen that, for circuit test pattern generation may random pattern so, what we have to do? We have to take the circuit and gives some input and get the output. So that means what? There is the circuit is there because but you have to know the circuit is not yet fabricated. So we are finding out which patterns have to be applied in the circuit is fabricated.

So all this test generation activity is, is being done when the circuit is being design; fabrication has not been done. So we do not have the circuit in the hard form, that is note have the fabricated version of the circuit. So we have a say the circuit is soft copy, so in a program or some other manner this soft copy; so only soft copy is available and then you have to find out which patterns. So by random pattern you are applying some random pattern 1s and 0; so you have to find out the output. So that means what? You have to stimulate your circuit; so what do you mean by simulation of the circuit? The imitation the imitative representation of the functioning of the circuit, by means of another alternative say computer program is called simulation. That is you do not have the hard copy of your circuit or you do not have the hard version of your circuit or the fabricated circuit you do not have; we applied some patterns and this circuit is represent in a computer program and you can generate the output.

(Refer Slide Time: 30:34)



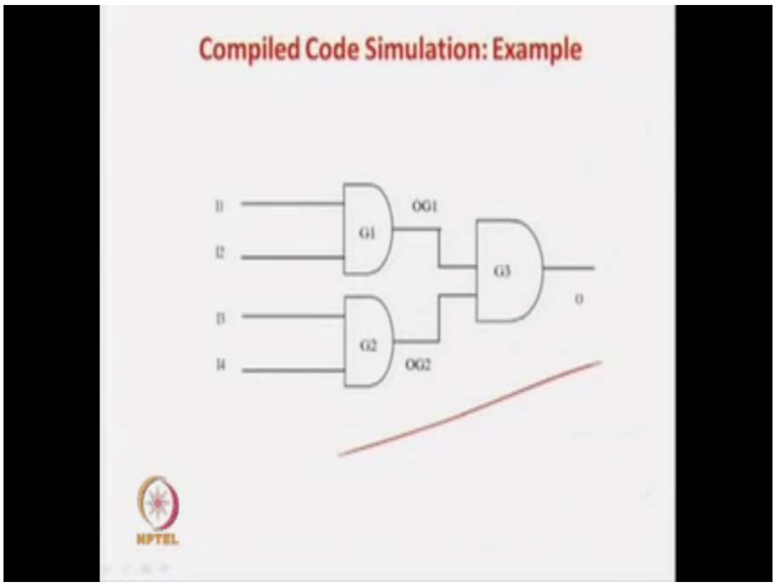
Compiled Code Simulation

1. Compiled Code method of simulation involves describing the circuit in a language that can be compiled and executed on a computer.
   - The circuit description can be in a Hardware Description Language such as VHDL or Verilog or simply described in C.
2. Inputs, outputs and intermediary nets are treated as variables in the code which can be Boolean, integer, etc. Gates such as AND, OR, etc., are directly converted into program statements (using bit wise operator).
3. For every input pattern, the code is repeatedly executed.

After that you can apply the fault and repeat the same thing; so in other words, you do not have the hard copy, but you represent the circuit in a computer program. That is very much required for fault test pattern generation by sensitize propagate and justify approach. So will study in detail circuit simulation an also about fault simulation; so what do you mean by fault simulation? In fault simulation the circuit is there is the program and then a fault is inserted in the circuit, which is represent the program and you find out what is the output computing your random improve.

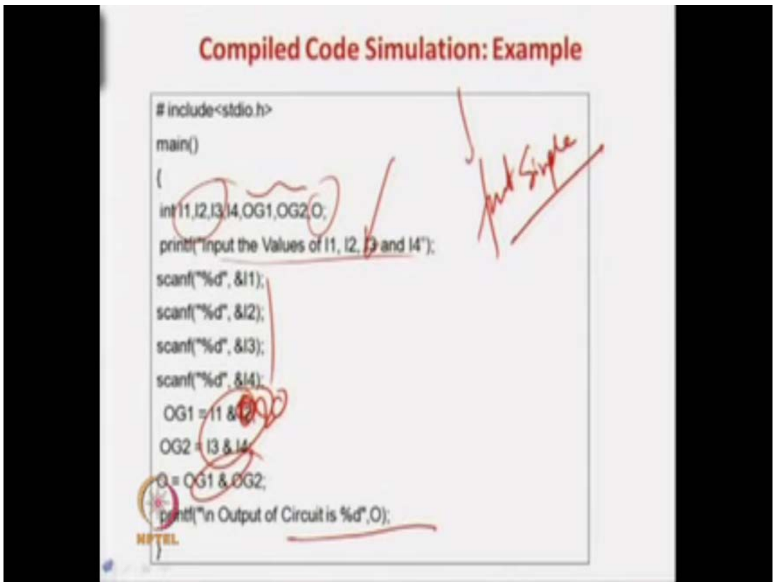
So first we will see, something called a compiled code simulation. So what is a compiled code simulation? The whole circuit is represented as C program or any other program and you generate the output or input this on the C program; that is like compiled code simulation involves describe the circuit in some language which can be compiled in a computer like it can be hardware language were log VHDL or it can be simple C and then you have to give some input and you get the output as you can repeat as long as you one. Say for example, the very simple circuit 2 in 4 inputs and 1 output.

(Refer Slide Time: 30:58)


Compiled Code Simulation: Example

Now how do you do this? So very you can represent in C; so you have lines like 1 2 3 4, these are the input lines and this is the output lines, that temporary variables. Now you printers input the values of these one, then you scanner; then you do this; do ending and then final ending and the output of the circuit is O. So different inputs you can give and defined outputs you can get form the circuits; so this is a very simple approach.

(Refer Slide Time: 31:02)



Compiled Code Simulation: Example

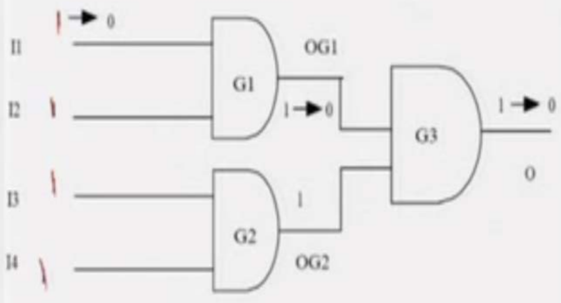
```
# include<stdio.h>
main()
{
 int I1,I2,I3,I4,OG1,OG2,O;
 printf("Input the Values of I1, I2, I3 and I4");
scanf("%d", &I1);
scanf("%d", &I2);
scanf("%d", &I3);
scanf("%d", &I4);
  OG1 = I1 & I2;
 OG2 = I3 & I4;
O = OG1 & OG2;
printf("\n Output of Circuit is %d",O);
}
```

Fault Simple

Now if you want to find out some fault maybe a some stuck-at-1 fault is there or some stuck-at-0 fault is there, you can some is the some stuck-at-fault is therefore this can be ended with 0 and so forth; just you can modify this circuit for the different kind of a fault and then you get the compiled code fault simulation, this is very simple. Just we have to apply I mean what do you called fault for doing it.

(Refer Slide Time: 31:44)

Compiled Code Simulation: Signal Changes



Changes in signal values in a circuit for change in input is low

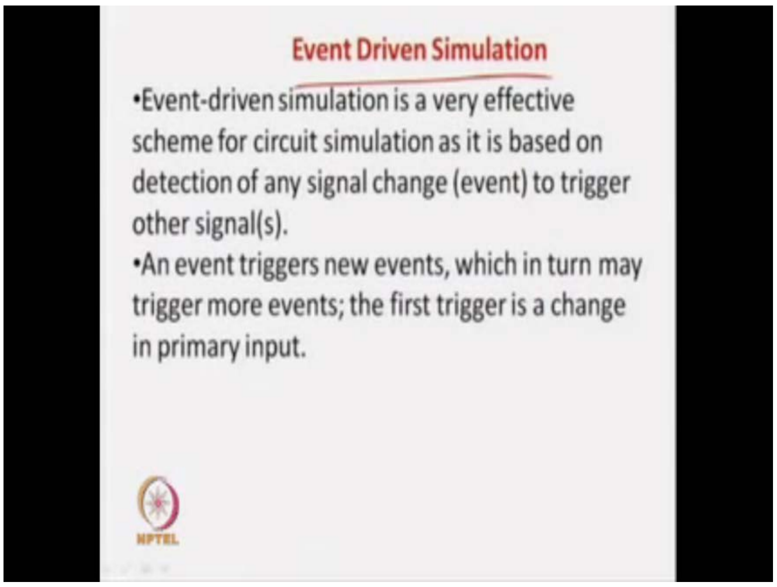
But, now you see what happens; say for example, you first you give input as 1 1 1 1 and then you generate the output. So what will happen? So if you give 1 1 1 1; so first what will happen? First this say set sentence is executed so, it is 1 and 1; so G1 is equal to 1. Now the this things are also 1 1, then O G2 is also compute to be 1 and then this is done; then O is generated to be 1. So is a three take computation so, fine this is done. Next what you do is that say, next we apply 0 1 1 1 that is a next random pattern. So in C language what is going to happen? So again this same code will run and it will take three steps.

So it will be 0 and 1 that is equal to 0, this is 1 and this is 0 and 1 is equal to 0. So answer is 0, but again takes three steps, but you clearly absolve the circuit, we do not require to do so much activity, but if because you see, this we are changing from 0 to 1 fine, this is no change; this is no change; this is no change. So output here is changing from 1 to 0 and this one is changing to 0. So only 1 bit, 2 bit computation or 2 line computation is fine; that is you need to compute this obviously this you did not compute is already same; obviously, you need to compute this and do it.

So that is what the circuit? There are some changes, but if you are going for a compiled code simulation, then the problem is that you have to have a simulation for the entire part of this circuit. In other words, now you consider is the very big circuit and you say that there is no changes in other parts of this circuit and only a minor change here which is effected only part minor change.

But now if we are going for a compiled code simulation; then what will happen is that, the hole circuit will have to be re-simulated and which is actually going to give you a big problem; that is unnecessarily just for a minor change, your whole circuit you are going to go for a compiling compile being changes; then you are executing everything is revaluated. So even if a very small part of the circuit is being change, but for the compiled code simulation, then you are changing or you are re-computing or you are doing redounding computation for other parts on the circuit.

(Refer Slide Time: 33:46)


Event Driven Simulation

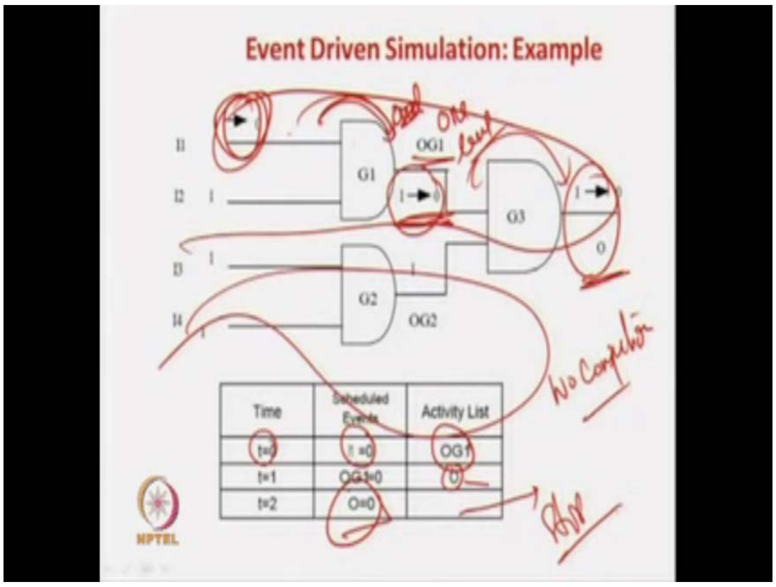
- Event-driven simulation is a very effective scheme for circuit simulation as it is based on detection of any signal change (event) to trigger other signal(s).
- An event triggers new events, which in turn may trigger more events; the first trigger is a change in primary input.

So to avoid this one, we called event driven simulation; so what is the idea of a event driven simulation? So the idea of event driven simulation is something like this say for the same circuit we take; so for the same circuit if you take, in case of a event driven simulation what is happening is say, this is the event and there is no event in this part of the circuit maybe some other big circuit is this, there is also no event.

So this part will not go for a any kind of a computation; so what will do? Only will go for computation of this part of this circuit where there are some event changes so, that is actually call event driven. This is the some of the events happening; so this is actually called event driven simulation. So it will save lot of time in your, in your computer execution of the for test pattern generation random pattern. So event driven simulation is very efficient, because it detects any signal changes and that regards other signals.

So it is un-recursive manner we will see within example, how it is done; so for example, the same circuit we takes so, this is a first change. So time t equal to 0; so one changes this one; I equal to 1 so this change is 0. Now only the next level test patterns or next level signals from this gate because this is revaluated will come in the activity list. So only O G1 will be activity less; now why is that so this will be because of the output of this gate is only O G1 and only one level jump is allowed, one level jump; only one level is allowed. So as this is a change here so, only this can be in the activity list.

(Refer Slide Time: 34:41)



Event Driven Simulation: Example

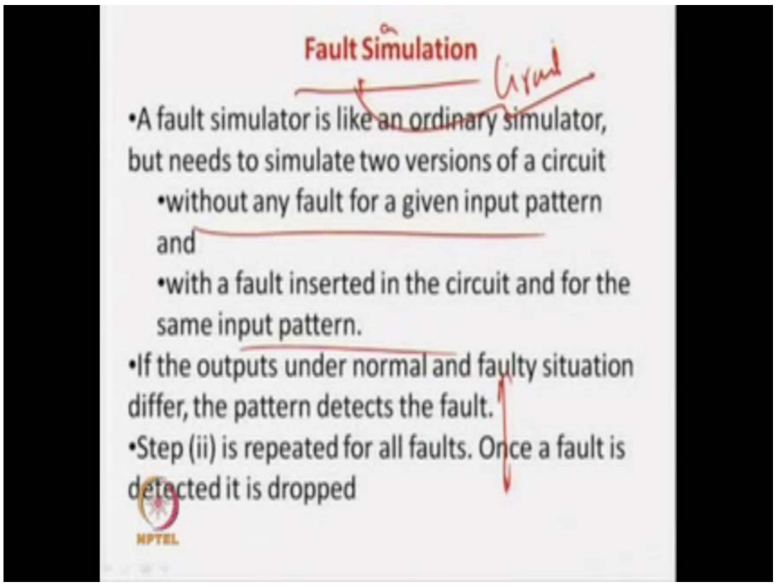
Time	Scheduled Events	Activity List
t=0	I1=0	OG1
t=1	OG1=0	O
t=2	O=0	

Then fine so you revaluated this, this is the case; so O G1 changes from 0 to 1 so, O G1 change from this one. So the only the activity list will be output of this one because only one level is allowed; so this change is there, so this will come in the activity list, this one is the case and then this valuates to be O equal to 0 nothing in that activity list and you stop.

So you just require a very small number of computations are to do it and you did not do any kind of a no computation require; no computation is required for this, all you require this. So what is the basic idea here, you have a circuit and you have some gates; so basic filozofia of this one let me discuss. So what we do is we have something like we have a gate like this; some signal change, some signal change is there; then output of this one will be in the activity list, that is it and again any change here will be again corresponding to this activity change. For example, if you have something like this say, say this had been in the case say for example.

Now in this case what would happen? Say this was our initial pattern was 1 0 1 1. Now you change this from this one; now what will happen? I equal to 1 to 0, activity list is O equal to 1. Now activity 0 1 and a 0 and a 0, the activity list is 0; so initial it was also 0, now it is also 0. So the activity list will be empty in this case because it is 0 to 0 so, no change is there and everything will stop over here and this change is also not there. So in level two, the propagation again stop, but if are taking a compiled code simulation; then again you have to evaluate this, evaluate this, evaluate this and so forth and is a wastage of time in computation. So event driven simulation is helping a lot.

(Refer Slide Time: 37:03)



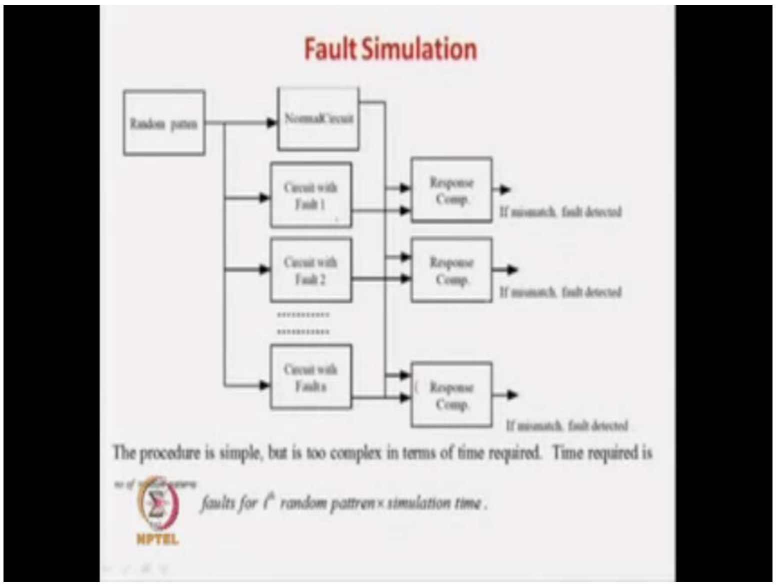
Fault Simulation

- A fault simulator is like an ordinary simulator, but needs to simulate two versions of a circuit
  - without any fault for a given input pattern and
  - with a fault inserted in the circuit and for the same input pattern.
- If the outputs under normal and faulty situation differ, the pattern detects the fault.
- Step (ii) is repeated for all faults. Once a fault is detected it is dropped

Now we have to go for circuit simulation; so what do you mean by a circuit simulation? In case of a fault we have seen our circuit simulation, circuit simulation means what? A circuit is there and then you one applying some inputs and you are finding out the output beet complete code simulation or event driven simulation, but for a fault simulation what do you know that for random test pattern generation, what we have to do? Have to put the fault in the circuit and then you have to go for fault simulation, circuit simulation. So fault simulation is nothing but if we have a circuit with a fault and then you to simulate the circuit for the out some output for given, for some given input is find out the output is called fault simulation. You can also called fault circuit simulation, also you can called you can also call it fault circuit simulation.

So fault simulation is like a ordinary simulation, but it has two versions; one is without fault and one is with fault; that is very simple idea. So the E and if the output of the fault, faulty circuit is not matching with the output or the normal circuit; that is what is relate then you know that the random pattern or the pattern is relating the fault, that is what is the idea of fault simulation.

(Refer Slide Time: 38:05)



Fault Simulation

The procedure is simple, but is too complex in terms of time required. Time required is $\sum$ faults for i^{th} random pattren $\times$ simulation time.

Let us see this is the block diagram which shows this so, there is some random pattern; now what happens? So you are applying what do you called this is a circuit with fault 1, fault 2, fault n; all the faults are there and this is normal circuit output you are comparing this one and then if there is a mismatch this fault is rejected, this fault is rejected and you remove that and you take random pattern number 2; this how it is done.

So we have and if you also how you are improving with you are improving in using what do you call the event driven simulation. So if you are using event driven simulation so, you are simulating circuit one; circuit two; circuit three with faults and at this you are also with fault one, fault two, fault three. So even if are doing it with faults, so there can be little amount of changes in the whole circuit.

So even if you are doing a what do you called event driven simulation; so you can say lot on the computations. So that is one is once once one way you are saving, but only one way another factor you are saving; we are not doing another thing is that in one random pattern we are trying to see, if one fault is rejected. So there also we can paralyze, so two ways we are making inefficient, one is event driven, which is already seen and that is parallel; that is here, what we are doing? We are taking a circuit applying one test pattern, random pattern see the faulty detected; then with fault 2 and fault 3 and so forth.

So we need to also think, if we can find out that given a random pattern; how many faults it can detect in one group? That is one good way of doing it, another is event driven because if you are going for compiled code simulation, then for if in for a small number of fault mean fault change or the input pattern change get go for a full code simulation. So that is one thing we have, these are two factors we have to improve; so one improvement is this event driven and that is parallelisms.

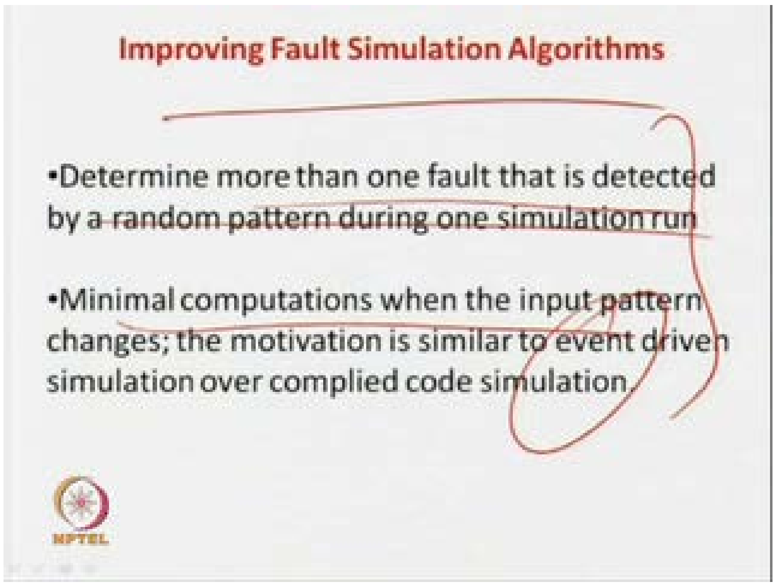
So both of them we are going to see in details. So, but if you are not do anything for this, this for even random pattern generation; random test pattern generation is also not very simple because the time is how much number of faults for the I S random pattern, say for the say the I-th random pattern still say for example, we start with 100 faults. First, first pattern for the first pattern, how many faults we required to check for all 100s; we required to check in pattern number 1 because 100 faults are there you have to check for 100; say for 10 patterns, get 10 faults get detected by the pattern very good.

Now pattern number two comes, then again 90 remaining faults has to be tested for that; say say 20 faults get detected sub by sub detected. Then for pattern number 3 you have to go for 70; so similarly you go as say for the tenth number fault is only 10 patterns remaining and say some only one fault get detected. Then for the eleventh pattern, 9 faults have to be checked; 12-9 fault because this is saturates. See up to 100 patterns, 9 faults has to be tested; generally we stop at this point because things have get saturated, but you see for the first pattern 100 of faults has to be checked for fault, fault pattern to 90 faults has to be checked for and so on.

So I mean, if you are do not have a parallel fault simulator that is I mean if you, you can somehow paralyze the algorithm; that is for a single pattern you can check whether among this 90 faults say 100 faults are there, can you check for parallely; that whether all the faults are detectable by this random pattern would be a very good idea and also and apart from that, also you should able to do maybe you should able to simulate your circuit for minimum requirement, only those things are change; that only you should be taking into picture and not more.

So we say that, the number of that is even driven simulation so, whatever minimal changes is there only that part has to be simulated and other things are to be retained; you should not avoid ram what do you called you should avoid redundant simulation. So we say that the time fault simulation is faults for the i x random test pattern into simulation time; that is for simulation normal values always there and number of random patterns you are going for that; that is not a very small number.

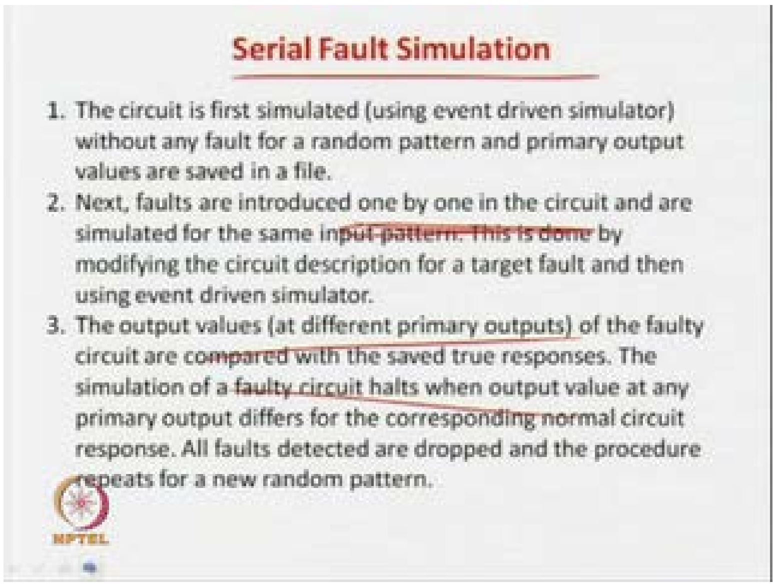
(Refer Slide Time: 41:36)



Improving Fault Simulation Algorithms

- Determine more than one fault that is detected by a random pattern during one simulation run
- Minimal computations when the input pattern changes; the motivation is similar to event driven simulation over complied code simulation

(Refer Slide Time: 42:03)



Serial Fault Simulation

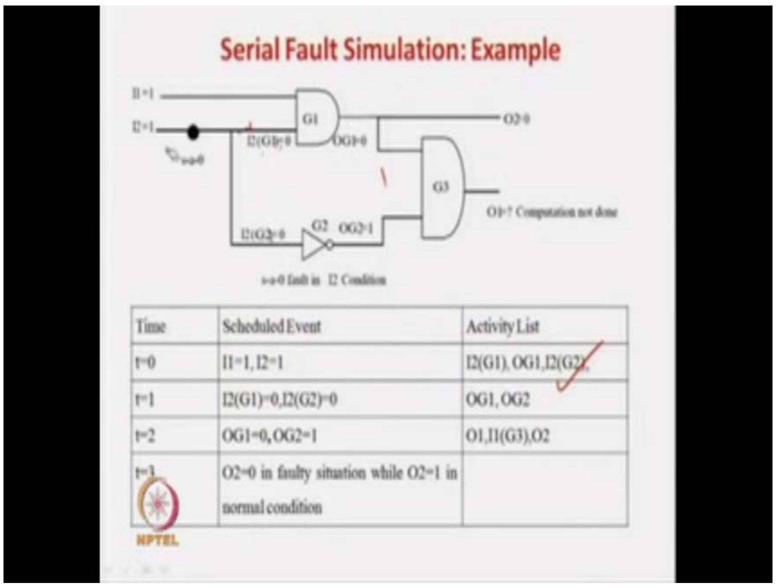
1. The circuit is first simulated (using event driven simulator) without any fault for a random pattern and primary output values are saved in a file.
2. Next, faults are introduced one by one in the circuit and are simulated for the same input pattern. This is done by modifying the circuit description for a target fault and then using event driven simulator.
3. The output values (at different primary outputs) of the faulty circuit are compared with the saved true responses. The simulation of a faulty circuit halts when output value at any primary output differs for the corresponding normal circuit response. All faults detected are dropped and the procedure repeats for a new random pattern.

So we have to see, so that is what is improving fault simulation algorithm, that is minimum computation change that is even driven simulation; so whenever is the pattern change or whenever there is a fault change, minimum changes are there. Then you have to go for only incremental simulation, that is even driven simulation you should not simulate the whole circuit and determine more than one fault can be detected by random pattern; that is check is paralyze. So if these two allegro, this two futures we can build in there our things should be improve, so that is what we are going to do.

So first we are going to see a very simple fault simulation algorithm, which is call the serial fault simulation; that is the very simplest one. So in this case what it is done so, it is nothing but you apply a pattern then you apply one fault the serial because serial you are going to check; then you see whether you apply a pattern, then whether see apply a pattern then take one fault and see if the fault is detected by the pattern, if so the fault is dropped and that is fault is consider and then this ability fault been tested; then you take another fault, see whether it detected by the same random pattern and so on; one by one you test for all the faults which is remaining and then where all the faults have been checked for say you get that, n number of faults has been detected.

So that you remove from the circuit that out of 100, sometimes faults have been detect removed; now for the 90 faults you repeat this same procedure for the next pattern. So that is the next fault are introduced one by one; that is a very important thing. In this one by one you are introducing the faults and being the fault simulation. So if the output is different fault are detected and so forth.

(Refer Slide Time: 43:07)



Serial Fault Simulation: Example

Time	Scheduled Event	Activity List
t=0	I1=1, I2=1	I2(G1), OG1, I2(G2)
t=1	I2(G1)=0, I2(G2)=0	OG1, OG2
t=2	OG1=0, OG2=1	O1, I1(G3), O2
t=3	O2=0 in faulty situation while O2=1 in normal condition	

For the simple serial pattern so, that is the very important; that is the main idea is that one by one you are taking, but again what you have to remember that, always we are going to use even driven simulation and even then we are going to improve on serial, parallel, detective and all those things slowly they will come in to fixture. So let us study serial fault simulation with example; so let this be the circuit. This is circuit you can see so, one thing I should say that, this is input I1G; so this is all fanouts are different if you remember. So this is I2 G1, this is I2 G2, this is input 2 of G2 some names you have given; so this is these output is O2, this is O1. So in testing as we know, all fanouts are different; so I am putting a different name to this one, also this I1 G3 say input 1, say this is the output G2 no problem. So this is the 1, this is I should remember I1 G3 this is sorry I1 G3 this is, this net.

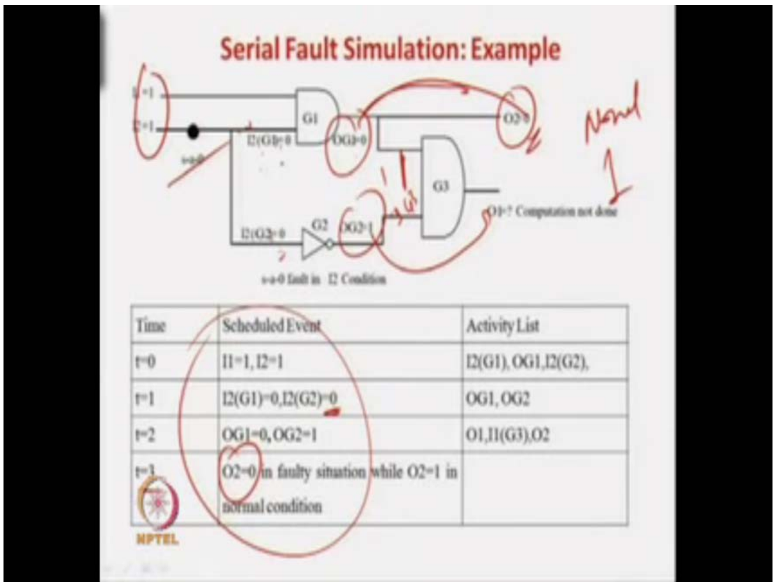
So now so there is a stuck-at-0 fault over here; so what you have to do? Say for example, we do a random pattern is say 1 1; so the what is the output of the circuit normal case O2 equal to 1, O1 equal to 0; that is a very simple because 1 1, so is the answer is 1, so it is 1 and in this case it is1, it is 0, so the answer is 0. So 1 0 is the result for the normal serial and this is the random pattern, right. Now we are going for this one, so the same random pattern when we are going to see whether the stuck-at-fault is deductible or not. So this serial, so will take one fault at a time; so this is 1 and 1 even driven simulation. So as I to already told you that, only one level is allowed so, this is in the activity list and this is in the activity list.

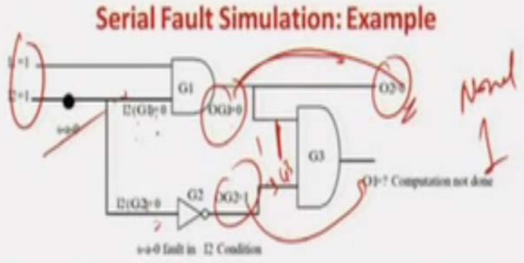
So only two things will be in the activity list so, what do you have here? Is I1 equal to 1 and I2 equal to 1, sorry and this one will also be there because this is also be in the list. So activity list I2 G1, I2 G1is the single level; O G1, O G1 same same level and I2 G2 this I2 G2 because this is the signals we have and this is the one level change from the signal. So one level change from this things; so you are going to have these things as the activity list because you are saying the this signal changes, that is input equal to 1 and input equal to 1, this two equal to 1 1 are going to have direct impact on this three points only; this one this one is direct impact. So these are in the activity list. Now if you have this 1 and 1 over here so, you can compute this to be 0, because of the stack-at-fault this is 0. So I1 G1 is 0 and then this is also I2 G2 is also 0; this you can compute, but this you cannot compute, because in the second round this values is blocked out.

So what is in the new activity list? This one is the in the new activity list because of this and this one remain in the activity list, this is could not you compute it because in the first value; this, this one was not known to us correct. So now in the next activity list O G1, O G2; now what happens? Now what we do is that? So now we have the values over here so, we have O G1 equal to 0; this one we know O G1 equal to 0 and O2 G1 equal to 1; this things we know. Now what is in the activity list? The activity list will be know O1 because of this one; we have O1 because this value is known; so this point is also known which I call I1 G3, this is I1 G3 and O2 obviously, this one will be there because this is there so one value jump will be there; which is one level jump is allowed.

So this is your thing; now you can easily see that, if this one is a 0; so next jump O2 is equal to 0 and if you look at the normal circuit O1 was a 1. So you can stop at this point say that, 1 1 detects this stack-at-0 fault because in the stuck-at-0 fault this is zero and normal case it is a 1; so it detects the fault and the thing is that so, this is by a random pattern simulation. So this we are doing event vise because do only for those parts wherever there is a signal change. Now so one fault has been detected.

(Refer Slide Time: 47:04)


Serial Fault Simulation: Example



Time	Scheduled Event	Activity List
t=0	I1=1, I2=1	I2(G1), OG1,I2(G2),
t=1	I2(G1)=0,I2(G2)=0	OG1, OG2
t=2	OG1=0, OG2=1	O1,I1(G3),O2
t=3	O2=0 in faulty situation while O2=1 in normal condition	

Now let us see what next? Same pattern is there, where verifying serial fault simulation; so we are going for this fault, another fault we have taken. This is stuck-at-1 fault over here; so this one as already taken I1 G3, this is there for you. Now again we are going to look at for all this things; so now what happens? So this 1 1 was say random pattern. So now in this case this 1 1 so, activity list at this 3, this point; this point and this point. This is the value, now we are going to get 1 1 over here because this fault is here not here this time. So we are going to get I1 and this one; so these values are there so, if these values are there; so we are having O G1 as this activity value already it was there, it will become to the new one added is O G2.

Now you see so this is 1 and in normal case the output should be 0, but stack-at-1 O G2 will be 1, it is not 0; because the stack-at-fault and O G3, if you see I1 G3 sorry I1 G3 is one because this value is propagated; so this value is you get and sorry, sorry this is the pattern which was saying so, G1 and O G2 you get so, G1 is 1 and O G2 is also 1 because of the stuck-at-fault over here. The activity list in this case is output one because this value we have and then I1 G3, because of this value; this is also in the activity list and obviously O2 will be in the activity list.

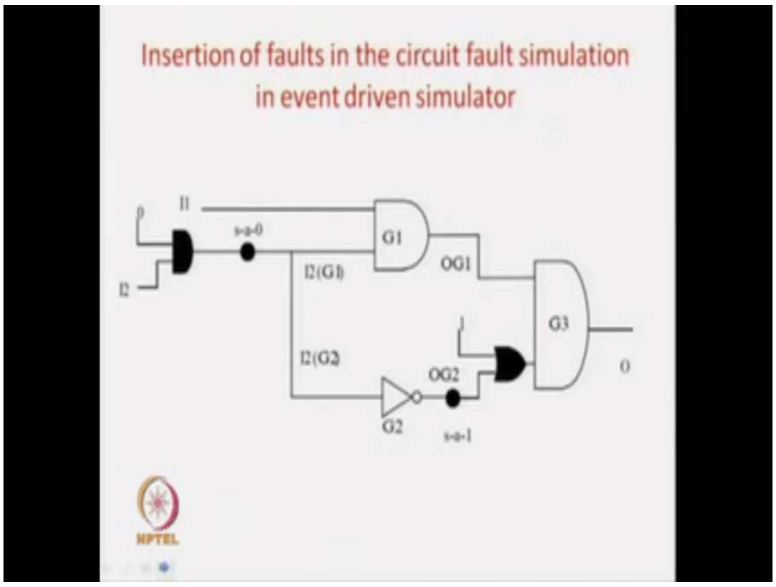
Now what you do? Now new value which is your obtaining out here is O2; so this O2 is equal to 1 because the initially we having; now you are having a value of 1 over here; so O2 is equal to 1 and as well as here also having this value from here to here it is 1; as well as to here to here it is 1; so I once, I1 G3 1 that is this fanout is also having the value of 1 and in the activity list is now 1 because this single is ready. So it was also there, but initially we could not compute the value because this value was not ready, value of this fanout was not ready; so in the what do you can called in the second step, the value was not ready, so you could not compute it, but now in third step this value of 1 is ready.

So in the value of step number three, this value is also they this values so you can compute O1. Now this is in the activity list in the first stage, O1 equal to 1; so this is the fault is captured because in the normal case the answer was 1 and this answer was 0; so fault is detected, but there is a one deference you have to observe very carefully here, that here we require four steps to do and here we require only three step to do; so that is the beauty of compile fault simulation.

So we are going event wise whenever you detected that, there is are deference within the normal circuit immediately stop. So for this fault we could not did not go for computation of this one, which require four steps. So we could have easily stop that the third stage and we find out that the answer is the there is fault simulator circuit fault has be detected and we are done, but for, but for this one, this one we have to go for stage number 4; why we are to go for stage for four because the fault is detected by the this get and not the this output; so we have to go for stage number 4, but had it mean a compile code simulation, then we are not going in a step wise manner.

We are get simulating the circuit totally and then we are comparing this one way this one way. So unnecessarily many times will be going for, here also will going for t; all for all circuits we have to go to the exhaustive level; so here also you would have gone for unnecessarily keep for which is not require. So for big circuit example we can verify that, always you will be at lot of gates.

(Refer Slide Time: 50:21)



Insertion of faults in the circuit fault simulation in event driven simulator

If you are going for a compiled code simulation sorry, you are going for a event driven simulation; so this was one example in case of serial fault simulation. Now one small thing I would like to add before concluding on today’s lecture, it is we are always thing the this level is stuck-at-0, this level is stuck-at1 then but algorithm circuit nature how do insert a fault? This is very simple; so if you want to add a stuck-at-0 fault at this net, you put a AND gate with this is a normal, this input was normally I2, you put a AND gate with one bit 0; so always it will be 0.

So this limits are stuck-at-0 fault; if you want a stuck-at-1 fault over here, then you this is your O G2 and you put a OR gate with one input fixed point; so this will manicure stuck-at-1 fault and this is going to manicure stack-at-0. So this how can manic, this was again simplified, but why do we require this because when you are using the fault simulator, then you cannot say this stuck-at-0, this stuck-at-1; because writing it stuck-at-0 stuck-at-1 we lot the handle by a circuit simulated. So what will to do? You have to represent stuck-at-0 and stuck-at-1 by means or some gates and there will be mimicry is required. And this mimicry will let use fault simulation for this part of the circuit.

So with this which talk today and so what we have this remaining for the next two series of lectures in this fault simulation, so here we are improved on fault simulation by we are not using compile code; so you have with the better technique call even driven simulation. But, still our faults may serial; so we took fault 1, fault 2, fault 3 and so on for a given random pattern; that is not good.

Then tomorrow or in the next lecture what you are going to see is that, how can you apply a pattern P 1 and see in parallel, can you check whether fault 1, fault 2, fault 3 and fault 4 are detectably 1; go by this test pattern. So that we are going to see in the next class, and similar other similar algorithm which can but you can which can enhance the performance of your fault simulation algorithm.

Thank you.