

Design Verification and Test of Digital VLSI Designs
Prof. Dr. Santosh Biswas
Prof. Dr. Jatindra Kumar Deka
Indian Institute of Technology, Guwahati

Module - 9
Combinational Circuit Test Pattern Generation
Lecture - 1

Introduction to Automatic Test Pattern Generation (ATPG) and ATPG Algebras

Welcome to the new module; that is module number 9; that is an introduction to 'Automatic Test Pattern Generation and ATPG Algebras'. So, in the last module that is module number 8, we have discussed two basic things; that is when you have to generate or you have to take a stuck-at fault. So, what we do basically? We first have to find out the pattern, which can detect this stuck-at fault; that is you have to find out a pattern which can differentiate that this circuit is either having a stuck-at fault or the circuit is normal. Like for an AND gate if you want to test a stuck-at 0 fault at the output, you have to apply 1 1 at the input. So, if the circuit is normal you will get one at the output and otherwise if there is the fault you will get 0 at the output.

So, we have seen basically two procedures; one is that is the random test pattern generation, in which we apply one test pattern; and then find out how many faults can be rejected by that. And then we have to find out that there can be another procedure, which is called the sensitize propagate and justify approach. In the sensitize propagate and justify approach what we do? We take a fault then drive it to a reverse value, then we try to drive it to the output at some primary output drive the effect to this primary output, and then we go to justify that for doing this sensitization and propagation. Whatever is required at the primary inputs as those values we try to apply and verify that we can do that.

Now, we have found out that since that what do you call the random test pattern generation it ATPG is algorithmically simple or computationally simple then the sensitize propagation justify approach, because in the random test pattern generation based approach you apply one pattern and then find out more than one number of faults, that can be detected in a general case. But in case of the sensitize propagation justify approach, you take one fault and find out the pattern; take another fault and find out the pattern. So, it may happen that 2 faults which can be detected by the same pattern, you have to lead to 2 differentiations of the algorithm. But, all faults cannot be detected by all test patterns

cannot be generated for all faults, in assert give because some faults are easy to test and some are difficult to test. For easy to test fault we can go for the random test pattern (()) approach, but for the difficult once we can find, we will find out that the number of random test patterns applied and the trials are much more in number; that is, you apply a random pattern do not detect the fault; then you apply a another random pattern, you keep on doing it and this (()) has to go for large number of times the difficult to test fault.

So for that, whenever you find that the random test pattern generation performance is degrading; that is, for I give a new a random pattern not much new faults are being detected by that; so what we do that we go for the sensitize propagation justify approach. Also we have seen one algorithm which is call this swap, that was in the last lecture, that how can you differentiate or how can you demark it in a heuristical manner, that is may be an approximate way with the easy fault and which are the difficult to test fault. For the easy fault, we directly go for the random pattern generation algorithm and for the difficult once we go for a sensitize propagate and justify approach.

(Refer Slide Time: 03:42)

Introduction

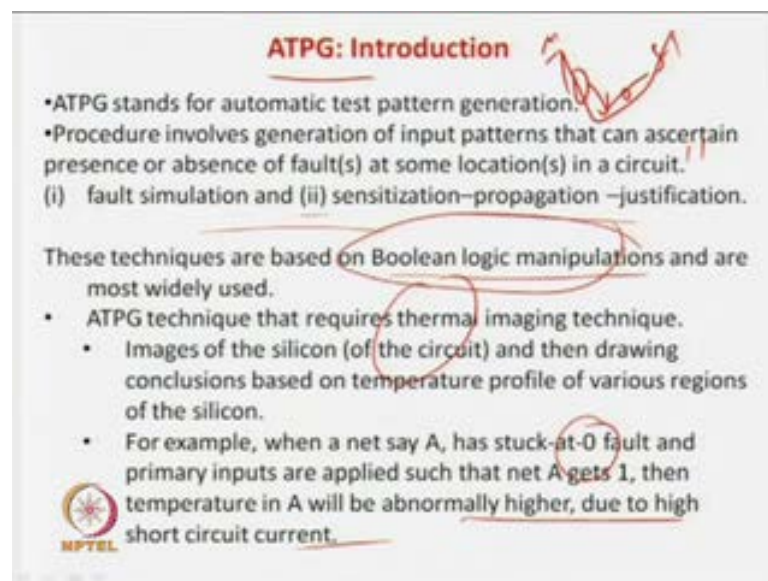
- In a circuit some faults are "easy to test" and some others are "difficult to test".
- Algorithm to estimate the "easy to test faults" and the ones that are "difficult to test". *SG 18*
- Fault simulation algorithms are used to determine test patters for easy to test faults and for the others, sensitization-propagation - justification approach is used.
- Automatic test pattern generation (ATPG) using sensitization-propagation -justification approach.
 - Basics of ATPG and ATPG algebras.
 - D algorithm—the basic and first ATPG algorithm developed.

NPTEL

So, in this module that is module number 9, so in module number 9, what we are going to do? We are going to see automatic will deal automatic test pattern generation algorithm; that is, will deal in details or in a formal way what how do you go for sensitize propagation justify approach for test pattern generation. Before that in this

lecture, we are we want to see that we want to see the basic introduction and also we have to see for some ATPG algebra; that is, if we are using, what do you call this, sensitize propagate and justify approach. Then this Boolean algebra which is 0 and 1 may not suffice; that is, we require a higher degree algebra this higher means will see that, what I mean that, so higher algebra in may be require in which case will have 0 1 and some other symbols which can help in sensitize propagation justify approach.

(Refer Slide Time: 04:25)



ATPG: Introduction

- ATPG stands for automatic test pattern generation.
- Procedure involves generation of input patterns that can ascertain presence or absence of fault(s) at some location(s) in a circuit.
(i) fault simulation and (ii) sensitization-propagation-justification.

These techniques are based on Boolean logic manipulations and are most widely used.

- ATPG technique that requires thermal imaging technique.
 - Images of the silicon (of the circuit) and then drawing conclusions based on temperature profile of various regions of the silicon.
 - For example, when a net say A, has stuck-at-0 fault and primary inputs are applied such that net A gets 1, then temperature in A will be abnormally higher, due to high short circuit current.

NPTEL

The slide contains several handwritten red annotations: a scribble in the top right corner, a circle around the phrase "Boolean logic manipulations", a circle around "thermal imaging technique", and a circle around "net A gets 1".

So, already told you that some faults are easy to test and some are difficult to test so algorithm likes swap. Actually generates this bring for you which easy to fault and difficult to test. So for, sensitize propagate and justify approach; for easy easy faults we go for random patterns and for the difficult we go for sensitize propagate and justify approach. So, in this lecture will see the basics of ATPG and ATPG algebras and in the next lecture will see the DE algorithm which is the basic algorithm for sensitize propagate and justify this approach for pattern generation.

So now will go for ATPG and ATPG algebras will see why it is required. So, basically random test pattern generation is also can be called as automatic test pattern generation that is ATPG. But generally, if you look at the random test pattern generation is algorithm, then we do not generally generate the test pattern. What we do? May we generate test pattern but I mean very formally speaking we do not do that. So what we do we apply a pattern and find out how may faults are detected by this. So, this is called

random test pattern generation, random test pattern generation, I mean random test generation based testing kind of a thing. But, if we truly speaking, what do you mean by a test pattern generation? Test pattern generation or automatic test pattern generation should be given a fault; then find out any pattern that can be detected or all patterns or more one pattern can be that can be detected.

Actually in sensitize propagate and justify approach we do exactly this way, we have a fault that then find out the pattern for doing in. So, we can say code encode that this is exactly automatic test pattern generation. In some places in the literature we may not call I mean, random test pattern generation as automatic test pattern generation because they call it that we apply a pattern and then find out which faults are detected. But, if you want to want to speak broadly, then both of them that is sensitize propagate and justify test approach and random test pattern generation approach, both of them are automatic test pattern generation fault the, for detecting stuck-at fault rigging fault any fault model you can take.

But, if you go by the literature that, we call ATPG for many sensitize propagate, sensitize propagate and justify approach. So, this is what will be terminology will be using. So, till now I check it before we come to details of ATPG, ATPG algebras and ATPG by sensitize propagate and justify. So, let me just give me a quick view that, the random test pattern generation and what do you call the sensitize propagation justify are Boolean logic based test pattern generation; that is, we do mathematically we have a circuit in fact if initially when we are going for automatic test pattern generation or random test pattern generation would not have this circuit.

So, what we do is a we have this soft copy of the circuit or we have a program for this circuit or we have a even given model the circuit. And then we try to apply algorithms to find out which test patterns can detect which fault. So, this is actually call Boolean logic manipulation based test pattern generation but there can some other physical or layout based or some other technique base stuff which is also detect fault.

1 very important part is actually called the thermal imaging. So, within not a pure Boolean manipulation base stuff and in this case what we do? We require the circuit. So in this case what happens, we take a circuit and then when the circuit has been fabricated then we take a thermal image. Actually we open up the fabrication I mean, mostly the

chips we have seen the black color or the with black color cover and all those things which are play out in your computer, mother board or which is in your mobile mother board or some places, were actually package chip. So, if you do not have the package chip, then we have the row fabricated IC, then we put it in the power apply patterns and then we take a camera and take, we take the thermal image. That is based on the flow of currents and voltages in the circuit depending on the flow of current and the application of the voltage are the different parts of the circuit, we get different temperatures. So, this temperature profile is recorded by a thermal imaging camera and then we analyze.

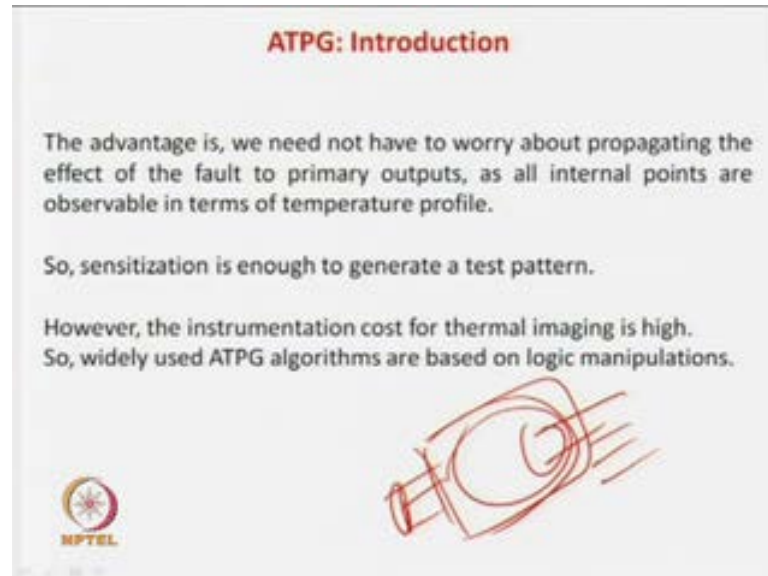
Say for example, where we know that, so this is the line that is going and say this line it is VDD some where it is connected and then this line is stuck-at 0. Then what happens basically, this line is connected to ground. So there is a VDD, VDD that is the 5 volts or the 3.3 volts or whatever is your one logic, one voltage and it is stuck-at 0 means it connected to ground. So, there is a short from voltage VDD source to ground source. So, lot of current, this current flow will be very, very high and there will be a huge temperature in along this region, so this thing when your thermal imaging will show a red spots in the some places of these faults.

So, then you can easily point out that the regions wherever the high amount of voltages where there is low, low I mean, low low temperature where it is high temperature. Based on different temperature profiling, we can find out which areas then may be faults or which areas there may not be any fault. Like for example, as I told you if there is a short from VDD to ground, then there will be a lot of current flow and then there will be it will be a very hot, red hot spots will be there and your thermal imaging camera can detect it is a very hot spots and there may be some defect in this area. So, you can pinpoint those faults.

So, I mean there is actually seen abnormally is higher when that is stuck-at 0 fault and there is again fault which you have seen, because I mean fault areas generally do have, what do you called, may be are these is a stuck-at 1 fault over at this point and for some reason you apply a 0 volt at this line; so then it will also have a source from stuck-at 1 to ground. So, you are applying a 0 volt means connected to ground kind of stuff and the line is net is stuck-at 1; means it is connected to VDD, so lot of current flow will be there and there will be a hot spot. So, all the regions for the faults will generally have some hot

spot kind of stuff. So, if you have too many hot spots then you can find out that those regions may not be normal. So, this way also you can detect the faults.

(Refer Slide Time: 09:11)



So, you can say that these are this is one totally different arena of fault testing I mean, or test pattern generation which we have not here discuss till now and in fact will not be also dealing with this in detail, because this requires lot of information about fabrication physical phenomena and all this it is a out of this cope of this, I want this cope of the course we are dealing with so. For that it is basically covered in physical device fabrication and all those things but still to give you the idea that always test pattern generation may not be based on logic manipulation, we are just coated the example of thermal imaging.

So, what is the advantage of thermal imaging? Thermal imaging is that you can directly have a dye then you have a camera then you can directly image out the thermal profile. So, you did not have to think about how currents are propagated to the primary outputs, how you can sensitize. Because, in case of dye chip, you can on chip based on package chip test, so you can only check the primary inputs and the outputs; control the primary inputs and check the primary outputs. But, you do not have any control over the intermediately lines, because neither you can observe nor neither you can control. If you have to do that you have to put the additional circuit.

But in case of thermal imaging, so there is some input patterns you give and whatever temperature profile you can directly observe with the camera. So, you did not think all you about or think about, how I can generate the test pattern, how I can propagate it to the output those problems are not there. But what is the disadvantage? Again the disadvantage is the time require or or actually the expensive the equipment for them are very expensive. Thermal imaging when you have take the image, go for the image processing and find out which is the hot spot that actually takes more time for testing and if fact the equipments are very expansive.

(Refer Slide Time: 11:24)

ATPG algebra

- Boolean algebra comprising of 0 and 1, is used for studying behavior of digital circuits.
- Circuits with fault: Boolean algebra will not suffice to represent signal values
- $1=1, 12=1$ detects s-a-0 fault at the output, we need to mark O1 as 1/0--indicating that under normal condition O1 is 1 and under fault it is 0.
- mark O1 as 0/1--indicating that under normal condition O1 is 0 and under fault it is 1.
- a single bit (0 or 1) is not enough for representing and studying circuits with fault.

(a) 1/0 is required to represent the detection of fault

(b) 0/1 is required to represent the detection of fault

So, again there is a trade off whether you have to go for logical testing or thermal imaging base testing. So, there is a trade off then again some parts sometimes you go for some imaging and sometimes you go for logic manipulation but generally speaking for the first initial few rounds of silicon fabrication, that many are fabricating this chip for the first few rounds, then there are may be lot of defect based on manufacturing based or design problems.

So, what we can do is that, we can go for thermal imaging or exhausted type of test but once you are in the flow then more or less the chip in this coming to find, there is no more design errors which is manipulating to the hardware errors. And also once thinks a more or less stabilize then you can go for basically in the (()) you are going for mass scale production. We generally go for logic logic manipulation base automatic test

pattern generation. And also for this course this thermal imaging or this physical type of test is out of scope of the course so we will again go back to our logical manipulation base testing.

So, let us see why we required a higher order what you call, order which is other than Boolean algebra why do we require some other algebra for automatic test pattern generation. So we know that Boolean algebra comprises 0s and 1; that is are very well known. So, what we see say? Say for example, if you are a AND gate, so input, input 1, I 1 is 1 and I 2 is 1; so to detect the stuck-at 1 fault at the output. So it is very simple we write 1 1 and in the output we say that in normal case it is 1, fault case it is 0. But, the 0 1 slash what we are writing over here, is not a very good way or formal way of writing. We require another symbol for this, because we have symbol for 0 we have symbol for 1. 0 means it is a ground and 1 means it is logic, a logic higher 0 means logic low. 1 0 what is that it mean that in normal case circuit is 1 and the fault case circuit is 0.

So but this is actually, not a very formal way of representing a Boolean algebra, because we require some operation like say for example, will say 1 and 0 is equal to 0. 1 or 1 is equal to 1. So, we actually represent 1 symbol and there is one symbol; where 0s and 1s are symbol. Generally, in algebra we do not use something like 1 slash 0 or 1 slash 0. Then in the 1 slash 0 1 slash 0 basically is equal to 1 slash 0 only, because 1 1 and 0 0 if you are at it is 1 another 0. But generally, this is not a very formal way of representing.


So we require some specialize symbols to represent this fact; that is 0 1. Similarly, if it is a OR gate, says another AND gate we apply 0 0. Then if we can easily detect a stuck-at 1 fault over here while because in the normal case the answer is 0, fault is the answer is 1. But, they represent these two stuffs, we require some special symbols. So, that is the motivation a requirement of some care of extra symbols or higher order algebra in for ATPG generation.

(Refer Slide Time: 13:10)

Roth's five value algebra

Symbol	Implication	Normal Circuit	Faulty Circuit
0	(0/0)	0 ✓	0 ✓
1	(1/1)	1 ✓	1 ✓
X	(X/X)	X	X
D	(1/0)	1	0
\bar{D}	(0/1)	0	1

Handwritten notes: d/0, 0 → D, 0/1, 1, 0/0, 1, 0/1, X, 1



So, what is this, so let us just have a see, so we call them the Roth's [fiveva/five value] five value algebra is generally used for ATPG that you will introduce some more symbols. And the motivation we have seen in the last slide that 1 0 0 1, so these means 0 1 means, circuit is normal voltage is 0 output is 0 and 1 that means circuit is at fault, some kind of fault then answer we are getting is 1. And 1 0 means for the normal case that net or that output is 1 and 0 means because of the fault that net is the have into be 0 but we require some special symbols so denote this, because if you do not have special symbol then Boolean operations is not very formal.

So, let us see what Roth's five value algebra or this Roth's has propose some algebra having more than 2 symbols, so what are they let us see. Symbol 0 and symbol 1; so 0 means basically 0 slash 0. The normal circuit it is 0 faulty circuit it is also 0. Like this is an AND gate say and this is net is stuck-at 0 say. So, if we apply a 0 over here, what does it mean? In a normal case with there is no concept of any fault, then we say that this is 0. But, in case of a circuit having fault this 0 implies that normal case it is 0 faulty case also it is 0.

Let us apply a 0 over here, for a stuck-at sorry stuck-at 1 fault say, stuck-at 1 fault that we have to say 0. So, what does it mean? It mean that normal case the circuit is 0, the faulty case circuit is 0. That is a 0 over here in ATPG implies. Now, 0 0 means the answer is 0. What is the stuck-at 1 fault, the answer will be 1. So, we will put this is a

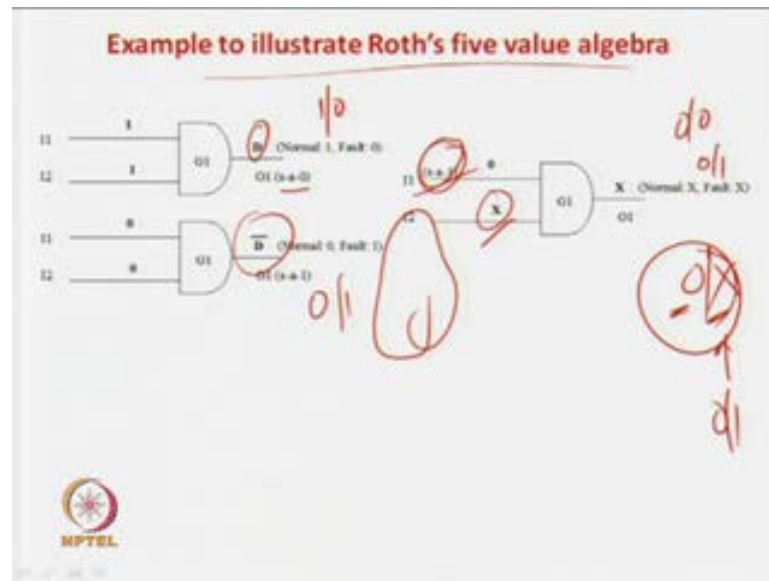
different symbol as will see. So, 0 0 means 0 normal case 0 fault 1 means 1 1 1 means normal case also 1 fault case also 1.

Sometimes we will use the term X. So X, X means we do not know, what is the value? Correct say for example, I will I mean generally in a hardware in a hard circuit, so if there is a AND gate kind of a thing say, and I power it on. I power the AND gate or I do not put anything at the input or I put some inputs which I do not know something like this. Say we will slowly see, when will go to sequential circuits or flip flops, then will see really understand the meaning of X. X actually means that, I do not know what is the value at this input or what is the value of the output. What do we have to mention here very categorically that in case of circuit there is nothing call X, either will be 0 or it will be 1.

But, we do not know because the circuit is not initialize properly that will be more clear when we go for sequential circuit that is, circuit is not initialize properly so we do not know whether the value of the circuit is at present 1 or 0. We represent it X. There is normal case also unknown; fault case it is also unknown. But, we do not know, that is but the circuit is not having any value X in a when the circuit is fabricated is either having 0 or 1. But, as the circuit is not initialized properly we do not know whether it is 0 or a 1 at the time of initializing. So, we say that it is X but we where it is not in fact X it is 0 or 1 which we do not know. So, that will be more clear when we will deal the sequential circuit.

Now, we come to the important paradigm called D. so for example, we have seen the and gate so that is the 1 1 and the answer should be 1; but if it is stuck-at 0, the answer is 0. So, we say that we represent this symbol by D. Then we say that 1 0; normal case 1 fault case 0. You take a AND gate with 0 0, the answer is 0, but stuck-at 1 fault then the answer is 1. So, this 0 and 1 that is normal case 0 fault case 1 we represent it by D bar. So basically, we have introduce three new symbols X, so whose who will be more clear when we go for sequential circuit; but for the combination circuits D and D bar which are two important thing which have introduced. D means normal case circuit is 1 fault case circuit is 0; and D bar normal case circuit is 0 fault case it is 1. So now we have two symbols, now we can do all kind of Boolean operations and whatever you can say using this symbols D and D prime.

(Refer Slide Time: 16:35)



So, that we will make our things more formal so like, if you see that 1 and 1 say for example, to illustrate doubts algebra. So if you say that 1 and 1, so the answer is D; because if for a stuck-at 0 fault, because normal case 1, fault case 0. 0 0 if the answer is 0 what is the stuck-at 1 for the answer is one we represent it D bar. Like in the stuck-at 1 fault here say and we have a X. So what is the X here means? X here means, when the circuit is initialized or it is coming from some combination cloud, you can say vise a vise a sub sequential elements and all, if flip flops are not properly initialized, so we have a value which is X.

It is either 0 and 1 but we do not sure what is that. So, and this line is stack at 1. So, stuck-at 1 means, so have that the circuit be normal, so we would have got the answer kind of a 0, because this circuit is 0 and X whatever will be the answer is 0. But, for the fault case we know that (()). So basically we do not have any clear idea what is happening, so one be it is X. So, do not know with the circuit is normal because if the circuit would have be normal. We would have got the value 0 but if the circuit is having a stuck-at 1 fault over here. We do not know whether it is 0 or 1; we do not know. Because, this X is 0 then the answer will 0 or 0 but if the X is 1 here, so the answer will be 0 and 1 than the fault can be detected.

But as you do not know the value of X, you cannot have any idea of what is happening in the circuit. So fault cannot be detected so basically better write it as X; that is how we

do. But, if you could have written like a stuck-at 0 fault over here then we could have say the answer is 0 over here, because 0 slash 0. Because, if the circuit is having a sorry, we cannot directly go for the example right now, like for example I was saying that, if there is a stuck-at 0 fault over here. Then we can very easily say that the answer is 0 if there is the fault even in the presence of an X.

But, here they have to apply a 1 to the stuck-at fault; 0 fault. So we could have say this circuit is, circuit is having a fault we could have the circuit the value of 0 but if the answer is 1 normal than you can get a value of X because 1 and X is X. You just did not know the value. Now if X is 0 this is 0 fault cannot be detected but if except been 1 then is 1 0 the fault could have been detected. So, there is no idea; that if 1 net of a AND gate is X. We do not have any idea what is actually happening at the output of the circuit and faults cannot be detected, so we have to better write it as less than x. So, the value of X we did not worry much about right now, because will be more clear where you going to deal with sequential circuits.


(Refer Slide Time: 18:59)

Example illustrate operations in Roth's five value algebra

- $1 \text{ AND } D = D$; $1 \text{ AND } 1/0$ involves two computations—(i) $1 \text{ AND } 1=1$, (ii) $1 \text{ AND } 0=0$.
- $0 \text{ OR } D = D$; $0 \text{ OR } 0/1$ involves two computations—(i) $0 \text{ OR } 0=0$, (ii) $0 \text{ OR } 1=1$.
- $\text{NOT}(\overline{D})=D$; $\text{NOT}(0/1)$ involves two computations—(i) $\text{NOT}(0)=1$, (ii) $\text{NOT}(1)=0$.

When one input is X in logical operation then output is also X, if others inputs are non controlling; else output is determined by the controlling input. Some examples are given below

- $X \text{ AND } 1 = X$; 1 is non controlling in AND logic
- $X \text{ AND } 0 = 0$; 0 is controlling in AND logic



$$D = 1/0$$

$$1/0 \wedge 1/1 = 1/0$$

Now, as I told you what is the motivation or having two or three more symbols like D, D prime and X; so because we can go for very formal Boolean manipulation. Like for example, 1 and a D; what is that? So, what is D? D is basically equal to normal circuit 1 fault circuit 0, right. So now, this 1 we end with 1; so what is that? This 1 end with 1

means, $1 \text{ slash } 1$; because normal $1 \text{ fault } 1$. So the answer will be $1 \text{ and } 1 \text{ and } 0$. So that is equal to D again.

So, this is how you get this value, so $1 \text{ AND } D$ equal to D . $1 \text{ AND } D$ basically $1, \text{ AND } 1$ 0 , because this is equal to D involves two computations, $1 \text{ AND } 1$ that is equal to 1 and $1 \text{ AND } 1$ that is equal to 0 . So, actually the answer is $1 \text{ slash } 0$ which is nothing but D . So basically, you could have also done the same thing using the algebra like $1 \text{ slash } 0$ and 1 , and you could have got the answer $1 \text{ slash } 0$ but that is not very formal. Because, in such type of slash oblige those things are not define in any kind of a algebra. So, better you have use some symbols like, we are again we are using say $D \text{ prime}$. So $D \text{ prime}$ is normal case circuit is 0 fault case circuit is 1 and we are or in with 0 .

0 means $0 \text{ slash } 0$ we are or in. So, the answer is $0 \text{ slash } 0$ is or and 1 . So, these nothing but again $D \text{ prime}$. So it involves two computations; $0 \text{ slash } 0$ or 0 and $1, 0$ or 1 , because $1 \text{ slash } 0$ here means, $0 \text{ slash } 0$ and $D \text{ prime}$ means $0 \text{ slash } 1$. So, 1 computation comprises these two and other comprises these two. So ok it is what is being written and you get the answer. Similarly, you can have get another direct answer for not of $D \text{ prime}$ is, $D \text{ prime}$ is $0 \text{ slash } 1$ if, you inverting you will get $1 \text{ slash } 0$ that is nothing but D so what is that they are say.

In again it was two computations not of this and not of this. So, you get the answer. So, what basically essentially Roth has done, so here actually introduce two symbols. We could have also done without those symbols but as I told you then it does not make your algebra very formal. So, that is why the importance of this symbols D and $D \text{ prime}$ are being used. So I mean as I have already told you that X is in logical operation then what do you mean, so $X \text{ slash } 1$ is X . Basically, what do you mean by X , X means $X \text{ slash } X \text{ AND } 1$ means $1 \text{ slash } 1$ that is nothing but $X \text{ slash } X$ that is equal to X . So these how it is there but sometimes we could have also says that $X \text{ slash } X$ and 0 . So that is equivalent to $X \text{ slash } X$ and $0 \text{ slash } 0$. It is nothing but 0 . So, actually the answer is 0 .

(Refer Slide Time: 21:48)

Types of ATPG algorithms

- **Exhaustive**
Testing by "exhaustive" technique implies applying all input combinations and checking for "functional" correctness. Exhaustive testing techniques are not used in ATPG because of complexity.
- **Random**
Random ATPG basically involves three steps:
 - Generate a random pattern
 - Determine how many faults are detected by the random pattern (fault simulation)
 - Continue the above two steps till no (or few) new faults are detected by the random pattern.
- **Deterministic ATPG techniques.**
 - Symbolic difference
 - Path sensitization.

NPTEL

So anyway, we bother more about this, what do you call X business in sequential circuit but anyway the idea by want to emphasize over here is that. If you have these five operators in pictures sorry, five symbols in picture then what is the advantage we are getting? The advantage we are getting is that we can go for a very formal way of having the Boolean function manipulations.

So now, we will go for the type of ATPG algorithm, just have a look at it. So there are three types; 1 is call the exhaustive automatic test pattern generation that is for a given fault, you have to find out all patterns which can detected. So how you can do that? There is no other way but we have to apply what do you call called functional correctness that we are going to have discuss in very initial starting in initial faces for testing lecture. When we have said that to test a circuit we apply if the rain inputs we apply 2 to the power n-th patterns and see whether the output comes for this specification.

So, then it is very obvious that if there is a circuit with n inputs and we apply 2 to the power n inputs to test, whether everything is proper or not. And obviously for a given fault you will be able to find out, which faults detect those I mean, which at the which which patterns it is which faults and a circuit is detected and if you are applying all 2 to the power n-th patterns and everything is exhaustive and everything is very clear to us. Then what we can find out that if, are given fault which are the pattern it is detected or

for a given fault which are all the setup patterns which can be detected and also the other way round.

So that is actually called exhaustive test pattern generation kind of a thing. That is for a fault you can find out, which are all the patterns which are detecting. But, in fact for testing if you remember or earlier discussion on testing, that you do not request all possible patterns or a fault. For idea is for our idea is that for structural testing is a stuck-at fault model for a given stuck-at fault, we have to apply one pattern, we have to find out that the sub fault is not there. Because, our number of samples to be tested is very high and our test time in the ATE or (()) test equipment is very, very less

So we have to do as soon as possible. So for a given fault we apply one pattern and we verify that the fault is not there and then quickly go for the other fault. So we have to do it very quickly. So, exhaustive testing generation is not a very good idea, because there is difficult to go for that and also we require only one test pattern to test the point. If then you argue also in that test in that idea, that if you require one pattern to do that and pattern to do testing then if you have the exhaustive test patterns for all the faults. And you can take the best one out of it. Like, you can find out one pattern which is detecting the maximum number of faults you keep that then you take the next pattern which is detecting the next maximum number of faults and keep that and so on.

So this can be done only if you exhaustive test pattern generation algorithm but again that algorithm is exponential in time and you will land up into years if you have a circuit with 100s of inputs. So, that is why exhaustive test pattern generation can be done by functional correctness based testing, and generally not used because of higher complexity. So, we go for basically single pattern that is for a given fault, we just try to find out a single pattern which can detect it and there can be two ways of doing it one is called the random test pattern generation. Basically, that is which we that is sequential concurrent deductive serial which we are already seen in the last module.

So, basically what is the idea that we generate a random pattern and then find out how many faults can be detected by the random pattern; and then we repeat this one till all the faults are detected or actually we are keep on or we have completed detecting easy to test faults. Then we have to go for something which is called sensitize propagate and justify approach. So, there is another name for that which is actually called deterministic

automatic sorry ATPG, so deterministic automatic test pattern generation algorithms. Why do you call this deterministic? Because we take a fault then we sensitize propagate and justify and we generate the fault. So, that is why sensitize propagate and justify approach is called deterministic test pattern generation. So, and what is random in random we just apply a test pattern and find in. So deterministic test pattern generation can be two there can be many so you are just going to look into two.

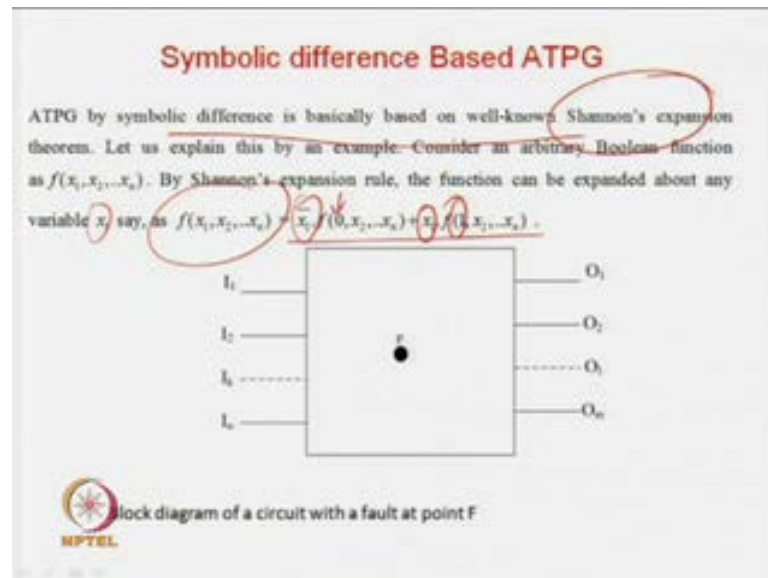
So, one is sensitize propagate and justify approach; which is the most widely applied approach. And we always the most of the algorithms which are using test pattern ATPG to sensitize propagate and justify approach is also called path sensitize approach and there several others. But there is another one is which is called is symbolic deference. So, quickly we have a review of and see, what introduce that what is symbolic deference a based or deterministic test pattern generation. And then will just have a look for it is not use, what is a difficulty in practical difficulty in doing a but when you are learning about deferent type of atomic test pattern generation algorithms, you just have a verity you just know verity of automatic test pattern generation available in the deterministic class.

Because, for the random class we have seen four like serial, concurrent, detective and sequential sorry, parallel, concurrent, detective and random kind of a thing. So four test pattern algorithms, four random test pattern generation algorithms you have seen. So for the deterministic one the most widely accepted is the path sensitization which is sensitize propagate and justify approach but still you have to just look the verity of this our other options available so among that symbolic deference is one which we are going to see.

What is unlike I mean, just like we have seen for you are what do you call this random pattern generation, the most widely accepted are concurrent and deductive once. Because they can mean generate very quickly or random they quickly they can analyze your circuit in one go or some days, if you have a high parallel architecture then we called go for parallel in that manner serial fault generation are not generally used.

Similarly, in this case path sensitization is most widely used and symbolic deference symbolic deference by the ATPG that widely accepted or widely used. We will see why? I mean, just given idea but know this pattern of test may algorithm say just having a look at this. So basically, if you look by symbolic deference based test is comes well know Shannon's expansion.

(Refer Slide Time: 27:02)



So what is Shannon's expansion if you have a function like this f of X_1, X_2, X_3 wherever all these X_1, X_2, X_3 are Boolean variables, it is a Boolean function, binary variables you can say. Then we can write it as X_1 prime there is a X_1 complement dot f of these value you have to make a 0, plus this is X_1 when you have to make this as 1 and we can go for this expansion. When it is can be done in a recursive panel. We all know that this is nothing but Shannon's expansion.

So, Boolean symbolic difference based determines the ATPG what is on Shannon's expansion. So, let us take a block diagram you know assert it has some m inputs and it has some m output. So, say there is a stuck-at f fault that the output, so how to do that. So, if you go for this sensitize propagate and justify approach that is the other. So what will say that, if is just a if it is stuck-at 0 fault. Then we have to apply a 1 then you have to propagate this effect so 0 and a 1 so it is actually D prime. Normal case 0, fault case 1, if it is a stuck-at sorry, you have apply a 1 sorry, stuck-at 0; so normal case 1 fault case is 0 also the it will be a D and this effect of D has the propagate at some output. So, once it is the affect the D propagates at this output some output then we try to justify this one. That is actually sensitize propagate and justify that is of your path sensitization is ATPG.

(Refer Slide Time: 28:27)

Symbolic difference Based ATPG

Let the following Boolean functions hold


- $g = f_f(I_1, I_2, \dots, I_n)$ be the function for the Boolean value at net F (in terms of all the inputs)
- $o_i = f_i(I_1, I_2, \dots, I_n)$, $1 \leq i \leq m$ be the functions for output lines o_1 to o_m , under failure at point F. It may be noted that without fault all the output lines are functions of the inputs. With the failure, the output lines are functions of the inputs and the Boolean value at point F.

Now, Boolean difference, or Boolean partial derivative, of the circuit block output f , $1 \leq i \leq m$ with respect to g is:

$$\frac{\partial f_i}{\partial g} = f_i(0, I_1, I_2, \dots, I_n) \oplus f_i(1, I_1, I_2, \dots, I_n)$$

For detecting s-a-0 fault at location F, we need

1. $g = f_f(I_1, I_2, \dots, I_n) = 1$
2. $(\exists i, 1 \leq i \leq m) \wedge \left(\frac{\partial f_i}{\partial g} = f_i(0, I_1, I_2, \dots, I_n) \oplus f_i(1, I_1, I_2, \dots, I_n) = 1 \right)$



Now will see what is the mean other with symbolic deference that is based on Boolean, I mean, that is based on Shannon's expansion; so that we will see. So same diagram we can explain both of them. So what is that, so let us, so first what you do? You first find out in case of symbolic deference this you first find out the value of this net. Because, this is nothing but a net, for the fault is occur, if I note a value of this net, in terms of this one; so we called it g. This is a function g, these are because say for a example there is a lot of a combinational in clouds, the re inputs and this net is having a stuck-at 0 fault which we are calling a f or something else.

And then the define the function of these one without the fault, that is g of I 1 to I 1. Obviously, because say this net is dependent on all this input you can say and in some inputs are not dependent on this in the value of this one will be X. Then you can be anything but we can easily define the values of this net were the fault is there in terms of these inputs.

So what is that they are saying, say for example, f of f in this be the function of the Boolean value of net f this one in terms of all the inputs. Now fault is not there. Now, if you want then we know that this is actually D sorry. Now, we know that this is the value of g of all the primary inputs, will give you the value of this special specialize net g of all the primary input, so will gave the value of f.

Now, if you say that, we can say that the value of some of these f_i , what is f_i ? f_i actually corresponds to the i -th output. Then we can say right that, f_i is equal to g plus I_1 plus I_2 the dot in with the function of the output line these were X under failure at point f . So now, without the failure, so what is the function? The function is f of i , I_1 I_2 dot i -th with the function of the output line O_i . So it will in 1 to n as there, so we have n m outputs are there, so we have how many functions? f_1 f_2 f_3 dot dot dot f of m and what are the n the functions depends on variable I want to I_a without the fault.

But now if there is a fault at the line, net F then you have to also introduce g . That is because now the output is effected not only by this but i is also affected by this one. So now what you right? Now there is a small notation we are using; so notation is the f of I of g . These some notation so it is basically nothing; but this is the function we are having f of i . So, this is the partial derivative of this function with respect to g . That is what we are doing? So, partial derivative what we are doing is, so we know that now the output function, say some output over this one. Some output say O_m ; so it is out this function is f of m into g , because output of a , output O_m now depends on not only the primary input but also on the fault. So, we write g and then only we can right I_1 dot dot dot I_n . So, these how you can define the output m function.

Now if you say that we differentiate this one partially with respect to f of g . That is we are differentiating this output with g . So, that means we are trying to find out whether there is any deference in O_m with the respect to the fault or not; so because we are going for a differentiation. So, we are, what we are going to write? We are going to write this this one a , f of I that is i -th output with respect to g , then we write it as f of I this one we apply a 1 and this one we apply a 0 .

Basically, what we are doing? So, this the partial differentiation we are saying that and we are doing an X or. So, we are saying that with is than any effect of this fault on this like effect. These are going to study; that is what is the partial differentiate meaning, that whether these fault this fault as a any effect on this at or not. So, when we can have an effect if, this is 1 then let say the let this say the output is O_{m1} . And then if the answer is 0 over here, let us say that O_{m0} O_m ; this is value of the output.

Let us assume that way. That if it is one then the answer f is 1 that is O_m , O_{m1} ; and otherwise O_{m0} is 0 . Now, to deduct this fault at this net, if you do a X or between them

then the answer should be 1. That is they should be deference; in other words, if the if the if this net or what do you call, if this fault have an output the this fault is to have an effect on this output or a if for the output any of the output, then if it is 0 whatever value say X and if the value is 1, then the X output value should be X prime. Or it can be other way round also, that is X prime and X.

So, that is if there is a change from 0 to 1 in this net, this output should also reflecting. If it is the case, then only the fault is deducted as the output, otherwise the fault is not detected of the output. So, if the value is 0 or 1 whatever may be the case here, the output remains the same, this output remains the same, then it implies that these net has no effect, these output has no effect on the fault and this thing cannot be used for directing the fault.

So, we right these are the partial derivation that f of I 1, that this 1 at the fault position and all the inputs; X or all this one should be... Basically, this is what is the partial differentiation and if you want to detect the fault at that net then the answer should be 1. So now, let us say this is this is the definition of the partial derivative, that means this is trying to find out whether there is any deference or whether any deference at the output I for this fault at net g.

That is it one to find out if there is any voltage change at net g, whether there is any effect at the output of net; sorry, at the output of net I, output I; that is what is the definition. Now, put detecting a stuck-at 0 fault at f, what do you have to do? Obviously we have to apply; we are going to detect a stuck-at 0 fault here. So, what you have to do? We have to going stuck-at 0 fault here. So, we have to apply a 1 over here so what we have to do? It is very simple, so first thing is there apply a 1.

So, how do you apply a 1? So, g of I 1 dot dot dot I n should be equal to 1, because this inputs only control this; so we write something like this. g of f of I 1 to I n equal to 1; so this the first condition it has to be satisfied. And then what you have the satisfy? Then you have to satisfy that at least one output, at least. At least one output should be there with this, because one we have applied and this fault is 0. So, 1 0 that is equal to D; now we have got a D, so just to the at least one output where this D value is propagated, can be propagated or in other word, they should be at least one output where this effect is propagated.

So, how do you know that? The effect is propagated, we have to find out that any net, any output should be (()) what we called we can be sensitized or it sensitive to this change. And how do you know this change? So, you have to write this one as, f of j then actually they are saying that there exist at least one j between 1 to m; that the at least one j from between 1 to m that at least one output where this, it is sensitizable, sensitized with the output is sensitized. If you put a 1 over there and a stuck-at fault is there. So, if put a 1, that means it is the normal case and this is the output of, the output at O j, if there is the stuck-at 0 fault; and this is the normal case, because output is not dependent on the primary input as well as the value at f. So, value of f is stuck-at 0 means, this is the fault case, the output value is 1 that means it is the normal case.

Now you are to, we are saying that, that is the partial differentiation of f I with default, so it is saying that there exist at least one output, where if the fault at the at least is at least one output for which the fault the effect is sensitizable. That is, f of j that is j the output, if you apply a 1 in the fault location and all the primary inputs, exhaust with if you have 0 at that fault location and you apply the some primary would the answer is 1; that means, the output is output j is difference with fault compared to normal case.


(Refer Slide Time: 36:23)


Symbolic difference Based ATPG

Similarly, for detecting s-a-1 fault at location F, we need

1. $g = f(I_1, I_2, \dots, I_n) = 0$
2. $(\exists j, 1 \leq j \leq m) \wedge \left(\frac{\partial f}{\partial x_j} = f(I_1, I_2, \dots, I_n) \oplus f(I_1, I_2, \dots, I_n) = 1 \right)$

Unfortunately, due to high complexity the Boolean difference is not an efficient way to compute test patterns for large circuits.





So if says that, this is the case then the fault is detected. Now, what is the test pattern? All the variables that is satisfying these equations and this test pattern. So, you are going to get a exhaustive test pattern for this way and in fact that is, what I told you that

exhausted test pattern. This is nothing but the complexity of this algorithm 2^n to the power n and this is something equivalent to all the means, fully functional testing using 2^n to the power n type of inputs and finding out the where has the default. So that is why, because of the high complexity of Boolean is not very efficient for large circuits therefore, we do not go.

Now, let us just see what they are saying for the stuck-at 1 fault, so if there is stuck-at 1 fault you have to dictate at f , then what you have to do, this is stuck-at 1 fault then you have to apply a 0. So f of at this function, this function is actually f of f say that is j , what do you call. So, in this case you have to apply a 0. So, by controlling the primary inputs in such way, so that this g that is the fault location should have the value 0.

Now, again what is the next case, in next case is the there should be at least one output j between I and m such that in a sensitized sensitible to this effect. That is the stuck-at 1 you have apply a 0, normal case 0 fault case 1; stuck-at 1 we applying a 0; is nothing but D prime. So, this D prime effect should be available at at least some output f_j and how do you represent this? So, stuck-at 1 so this represents fault and 1 it is represent normal. And this value should be different at the output; why because this g is 1 here and g is 0 over here and is controls to the primary inputs.

(Refer Slide Time: 38:05)

Path Sensitization Based ATPG

ATPG by path sensitization method is generally applied for "difficult to test faults" and comprises three phases.

Fault sensitization: In this step a stuck-at fault is activated by setting the signal driving the faulty net to an opposite value from the fault value.

Fault propagation: In this step a path is selected from the fault site to some primary output, where the effect of the fault can be observed for its detection.

Line justification: In this step the signals in (internal) nets or some primary inputs, which were assigned for fault sensitization/propagation, are justified by setting (remaining) primary inputs of the circuit.

In the second and third steps, a conflict may occur, where a necessary signal assignment contradicts some previously-made assignment. When conflicts occur we need to take a new alternative path for fault propagation and see if all signals can be justified.

NPTEL

So now, which variables to the satisfy of equation of which combination of variables should satisfy the equation. All variables, all combinations of I_1 to I_n which actually

satisfy this equation and this equation will we are test pattern for that stuck-at 1 fault. Now again, I mean the exhausted number of test patterns can, I mean what you can call satisfy this equation, so this has actually a very high complex algorithm and therefore, we [given to] even if it can do satisfy automatic test pattern generation algorithm by what you call, deterministic approach but still you do not take it, because it is more or less equivalent to functional test thing which is exponential order complexity.

So, now we will go for the approach which is widely used for test pattern generation, which is called path sensitization based ATPG, or which we are continuously calling as path sensitize propagate and justify. So, what you mean by path sensitization? Already I told you, that if there is stuck-at fault we have to drive in to the opposite value. It is just like, there is a electric bulb, then you have put it is fuse, so you have to make it you have to give power on, so that it goes to value 1; that is it tries to glow. Then if you the bulb fuse () if for example say, that the bulb cannot the switched off it always on. Then what you have to do? You have to try to switch off the bulb then the bulb is, if it is continuously glowing, then you can know that basically with the stuck-at 1 fault or the bulb be always on one kind of a thing. So, send for create the fault you have to always sensitized that is you have to give the reverse value of the fault.

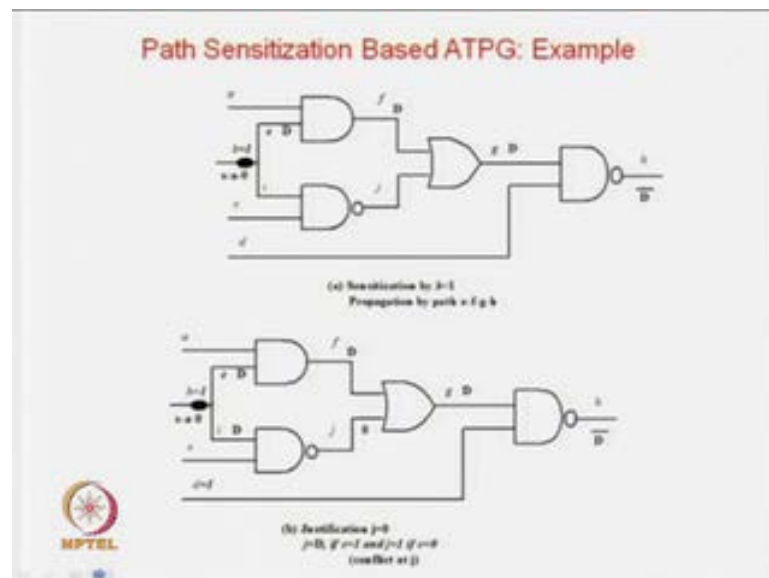
Now, then is I call the fault propagation. That is this fault effect has to be send your primary output, because in case of testing we know that we can only control the primary inputs and observe primary output; intermediate lines we cannot control. So the effect has to be, effect has to be gone to some primary output when it can be observed, right. And this line justification means, because all these steps we are doing manually I mean, all the steps sorry, all the steps we have done required some kind of, I mean all the steps in fault sensitization and fault propagation, all these things has to be controlled by the primary inputs. So, now you have to justify that this sensitizational propagation can be done by appropriately setting the primary input lines, because we cannot control the internal nets.

In this say, signals are the internal nets or some primary inputs which are mean which assigned for fault propagation justify and justify by the setting the primary inputs of the line that is what is the say. So, what we say? We say that, in this step the signals are the internals nets or primary outputs which were assigned in these 2 steps, like for fault sensitization not for the assignment the assignment was if the stuck-at 0 for the apply a 1

and it is the stuck-at 1 for the apply a 0. So that internal nets set we some voltage like for propagation; that means, we say some primary inputs sorry, some internet internal lines; so that the output effect of the fault effect propagate to the output.

So, we said some signals values in this circuit in some way and fault line justification will say that yes that can be done by appropriately setting the primary inputs. So we will see with an example. So, that it is possible, so what you mean by that. So, that is actually call line justification. So once it can be justify, then you are very happy, because sensitize propagate and justify has been done, your fault value has been [approach] the output, a prime fault effect has been propagate the output and you can observe it. But, we are always saying that this is straight forwards faults as say propagate justify for sensitize propagate and justify. But, life is not very simple, because always or many times you can have conflict; that is a conflict. That you has gone for fault sensitization and you have gone for fault propagation but that is do no primary input combination which can satisfy fault sensitization propagation. Then you have to go for alternate path also that will see by an example.

(Refer Slide Time: 41:11)



So that is why it is the very difficult thing. So like that what we are emphasizes that. In case of right now we are just going to, going back (()) it is or a previous lecture, what we say that? In the random test pattern generation our answers are very simple, we propagate the circuit in we are which travels the circuit in one direction and we get the

result, in case of detective or concurrent. Then one full scan of the circuit we can find out these patterns detects there how many faults.

Similarly, if you say that propagate sensitize and justify can find out your faults, that is once you sensitize propagate and justify then there is also one scan of the circuit. Then why you are going for random test pattern generation. So, one advantage of random test pattern is that, we have seen that just say that, the one pattern you can detect multiple faults that can be detected by the pattern. But, at the same time it is also said that this random test pattern, because once scan of this circuit. Similarly, we can also question that sensitize propagate and justify also does once scan of the circuit. You sensitize, you propagate and you justify. But, life is not very simple; many times you have to go for iteration; that is you are sensitize the fault propagate it value of the output, then we find out that these cannot be justify. That there are no primary input combination which can justify sensitize a propagate step.

So, again you have to find out other path or alternative mechanisms to do that. But, this problem of back track is not any random test pattern generation. So we now see a example then things will be more clear, that what you mean by back tracking. So have had back tracking not been there, then always we could we could have totally thrown have a, this random test pattern generation and we could have gone for the path sensitization or sensitize propagate and justify the approach at least some the complexity point of view.

So, let us see, so let us consider the circuit as an example. So, let us take that these are stuck-at 0 faults at this one. So if these are stuck-at 0 faults, so you have to apply a 1; so we say that we apply a 1 over here. So, apply a 1 over here. So normal case 1 fault case 0; that is actually nothing but D. So, now we are going by the Roth's algebra, so we have to write, we do not write 1 slash 0, we write basically D. Now the value, now this is, now this is sensitized; 1 and fault 0 I have put 1.

Now, we have find out the path through which is value can be propagate the output. So, you can see this is one path and this is also another path through this one. So, you can take any one path or sometimes also we can try both the path together. So, let us for the time being first try with this path. So, what is the path? Path is e f g and h; this is the path. So now let us try to see the path.

That is now sensitization I have to propagate it. So, the value of D we are going to get here NAND gate so it is not a non inverting gate, this only value will be there. Again here propagating this is path OR gate AND gate and non inverting gate; so D will propagated. Now, you can see that this is a inverting gate AND gate, so the value will be D prime over here. So anyway both D and D prime detect the fault, because D means normal case 1, fault case 0; D prime is normal case 0 fault case 1. So, both are there and detect the faults. So in (()) the D prime, so we are happy with it, so this call the fault propagation.

Now, sensitization is done; fault propagation is done. Now, we have to justify that this setting of the lines that is this propagation and the sensitization is possible. So in this case, the sensitization is very easy it is possible just by setting b equals to 1, this is the primary input fault so you can easily do that. That means was sensitization is done. Now, we have to see that whether propagation is possible through the primary inputs. So, this is D prime so you have to obviously put a 1 over here, because if we put a 0 over here then in the NAND gate output is 0 then all the problems, all the whatever effort you have doing is loss; so you put a 0 over here.

Now this is 0 is a OR gate so obviously you have to put a 0, in case of a NAND gate. Because if, the OR gate input is 1 then the output will be 1 and our effort is lost, right. So now, you put a 0 over here; now if you want to put a 0 at the output of the AND gate, then what we have to do? So you have to, if you want to get the output of the NAND gate as 0, so this is actually 1. So we want a get the output of the NAND gate 0, then we should have 1 input as 1 and this input as 1, correct? And what do you call and this gate is D, this is D so a equal to 1. But, now you see we have got a big conflict over here. You, it will get the answer a 0 over here, we need a 1 over here as well as 1 over here. But, this net is basically D because of the fault, this net is D. So, we require a 1 but actually D is being applied. So, this actually leads to a conflict and we have to try for alternative path.

So, that is why I told you that although we have, we have telling from the first lecture on was the sensitize propagate and justify, sensitize propagate and justify is the very good algorithm, because for difficult to test fault you can find out the pattern. But, life is not very simple, your sensitized propagate justify there is something called failure. Sensitized propagate justify failure iterate; sensitized propagate failure iterate. You have

to keep on doing till we get a answer or sometimes we may land up and saying that the fault is non testable.

So, this is one option is gone; now, what is the next option? The next option is we have to try this another alternative path that we can try from this one, correct? So let us see the alternative path. Now D is there; this is inverting gate. So, you will get a inverting over here. Is now this is the path these are a propagation path, sensitize already is done. That was successful. So now, again D prime because this is non inverting gate; this was a non inverting gate, correct? So these are non inverting gate so again D prime, now this is a inverting gate so you have get the value of D over here.

Now, let us see how can we justify this. So, in this case you have to propagate the value of this one to this one; so this is D to D prime sorry, D prime to D because a inverting gate. So, you put the value of 1, because this is the controlling value; so non controlling value for NAND gate. Now, you can see this D prime has to propagated over here all gates, so we have to get the value of 0 over here, correct? And this is also D that you have to know, because this is a D, because stuck-at 0 and 1 but to get the output 0 at the and gate you put a equal to 0. So, you are done. So now, this path sensitize propagate justify everything is satisfied. So let us see, what we have the satisfied right now.

So, sensitization stuck-at 0 1, so you apply a 1; so that is fantastic sensitization is successful. Now you have to propagate the value through this output; so you have propagated this one. Now, we get a 0 sorry we need a 1 over here. Now OR gate, so these are the propagated, so you get a 0 over here you have to apply 0 over here, because this non controlling input for OR gate and for a AND gate if you want to take the output as 0 you have to apply 0.


So, the pattern a equal to 0, b equal to 1, c equal to 1, d equal to 1 is the test pattern you detect the fault. So, what is the test pattern a equal to 0, b equal to 1, c equal to 1 and d equal to 1 and output is D that is normal case 1 failure case 0. This is the test pattern we detect a stuck-at 0 fault over here. So, we have found out a test pattern to detect this fault.

(Refer Slide Time: 47:55)

Questions and Answers

Question: Using path sensitization and Routh's algebra generate test pattern for the s-a-0 fault in the circuit given below:

Answer: To sensitize the fault, 1 is to be applied at the output of the AND gate. Also, there is only one path for fault propagation-- fault location to E. Now, to justify we need to apply a 0 at the second input of the OR gate. This leads to a collision. For justification we need a 0 in the second input of the OR gate and to sensitize the fault B is to be 1, which makes the second input of the OR gate a 1. This is illustrated in the figure below in terms of Routh's algebra.



But, we have to remember here that it has come with a cost of iteration. So, sometimes there can multiple iteration that you can fail in both the paths. Then you have to try to propagate the effect to both the paths together. And you may have to try keep on doing it till you are exhausted and we found that there is no more alternative paths so do it; then you have to declare that the fault is a non testable fault. So, you can understand that to detect the state stuck-at 1 fault or stuck-at 0 fault by sensitize propagate and justify approach. So many times you may have to iterate to the algorithm, because you may have sensitize propagate justify failure, sensitize propagate justify failure. And keep on trying alternative path. That is why we always go for sensitize propagate and justify approach only where we fail by random test pattern generation, because of random test pattern generation sometimes you may not detect fault you have to stop I mean, you have to draw you have to retry those faults.

But, this is not direct clear concept of iteration. That is we apply a random pattern some easy to test faults are detected, someone not detected we take them for the next round and keep on doing in. So, in a in a broader sense, in sensitize random test pattern generation, there is no concept of iteration as bold as in bold code encode which is there is sensitize propagate and justify approach. Therefore only for difficult faults we try this approach.

Now, let us go for a question and answer session; so we have a fault over here that is the stuck-at fault over here. Now, this is circuit; now we try to generate a test pattern for that using sensitize propagate and justify approach. So we will stuck-at 0, so you apply a 1; correct? So, correct you apply a 1 so normal case 1, you can get the fault case to be a 0 over here; right? So, in this case normal value is 1 fault value is 0 so actually sorry, the answer is that not a D prime it is D, because you have sensitization you have to do you apply a 1 then 1 and the normal case fault case is 0, so it is a D over here; sorry, (()) it is D. So now, this is non inverting gate so you also get the answer as this one, correct? So you stuck-at 0 you apply a 1, this is the value so this is nothing but D; and then this is non inverting gate so you get the answer as D over here. So, this is the propagation path, because there is only one propagating path in this case, there is no other fan out here. So, this is only one path where you can propagate the value. So, that is what we have done D and D over here. Now let us see what happens.

So, sorry let us again re draw it is 1 D I am sorry, it is D. So, it is an OR gate, we know that we have to get a value of 0; correct? So, this the value of 0 over here, because we have to now you have say proper I mean, a sensitized and stuck-at 0 we apply a 1 this is sensitization; that is D value is propagated to here and justify it whether this path is possible true but you can called a I mean, primary combination of primary input. So it is 0; so it is very correct that you have to apply a 0 over here for OR gate. So, applying getting a 0 over here we implies that b is equal to 0. So, b is equal to 0 implies that this is a 0 over here; so we line into or conflict but to state a stuck-at 0 fault we required a 1 over here but again to propagate it we require a 0 over here and it is again a 0 over here so these are conflict.

So, now you can see that this is a big problem over here, because there is no other alternate path for the circuit. So, this fault is a un testable fault. So, sometimes will be happen that in this case there is only one fan out, so we have in one iteration we stopped and found out the fault cannot be deducted. But, for the general case we may it may happen that there is a large number of alternative path, and for none of them the answer is successful. So, always you have to go for sensitize propagate justify failure. Sensitize propagate justify failure. We keep on doing it for some 100th option possible and then we find out that the fault is non detectable and how we are algorithm cans taken a lot of time and we are in a bad situation.

Now, let us just analyze for very quickly that why this fault is not detectable. So faults are not detectable only if there is some redundancy in this circuit. Like in this case, what is the function? This function is $A \cdot B$, at this point and it is and this one is actually $A \cdot B$, this $A \cdot B$ and this or plus B . So, this is nothing but B into $A + 1$ kind of a thing so this is nothing but B ; correct? $A + 1$ is 0, so this net is nothing but actually A straight line called B . So this is A , this circuit has lot of redundancy. So if you actually optimize this circuit we will find that A is equal to nothing but B so that is nothing but that is your circuit.

So, because of the some, if there is some redundancy in this circuit and fault those, those lines cannot be detected. But, sometimes we do required have to have redundancy in this circuit to math clock q and several are the designed paradise. So, all those redundant paths if there is a fault those faults cannot be detected. But, our ATPG algorithm if sensate propagate basically there the culprits, those what do you call those redundant paths are actually the (()) black nightmares for the testing people or the test engineers. Because, if there is redundancy in the circuit neither random test pattern can detect those fault. So, they will be classified as difficult to test pattern like we have seen in this flop algorithm that because of some redundancy in the circuit.

Then this scope value should was six but actually it should be infinity because that fault cannot be tested so any ways scope will classify them other difficult to test fault. Because, scope will never give the value infinity kind of a thing, because of the heuristic or I mean what do you called because of some redundancy because some some simplicity we have kept in the algorithm to fall make it quicker algorithm. So, that you get the answer of fault. So, scope will give a very high value but you will never say infinity so difficult to test fault and if you keep on trying sensate propagate and justify approach, and every time you will be failing. So, in this case this example circuit is only one fan out path.

So, we have just tried in one go and you have failed and you have got and you did not go for iteration, because there is no other option. But, for a general circuit there can be lot of options, all the options are defiantly bound to fail. Now, why we are bound to fail because they are redundant circuits and those faults cannot be tested. So, you will keep on doing sensitize propagate and justify and keep on failing and in the end, what you are going to do is that? You are going to declare that the fault is not testable by that time,

you have taken lot of your, what you computation time. So that is lot of very good approach of doing in.

So now, so this comes to end of the basics of ATPG and ATPG algebra and how we can generate test patterns using sensitize propagate and justify approach. In the next lecture, we are going to take ATPG that is actually these sensitize propagate and justify approach, we are very well known algorithm which is called the D algorithm. So, we will see formally, how D algorithm is used go for sensitize propagate and justify approach base test pattern generation. So, that will be looking up into the next lecture.

Thank you.