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

Module - 11
Built In Self Test (BIST)
Lecture - 4
Lecture Name: Memory Testing-2

So, welcome to the last lecture of the course, that is our memory testing module number 11 lecture 4. So, in the last lecture if you remember we have discussed that memory blocks comprise a very important part of circuits of digital circuits. The another important point we said that two basic physical difference, which makes VLSI testing for memory quite different from that of synchronous, or asynchronous synchronous sequential or combinational circuits.

So, what you have seen that in case of memory, almost all the memory chips will have one defect or the other. So, if you keep on finding out the defects in these of the chips you will land into the situation where you will be near about 0. So, these unlike other circuits, where the yield can be as high as 70 to 80 percentage and we shift only the normal parts. So, that we do not have any false parts, but in case of memory chips, the case is other way around.

So, what we have to do is that we have to find out redundant parts of the circuit. Not only we have to detect, but also, to diagnose the faults and then read out the chips by blowing out some chips or blowing out some of the internal connections by lasers or some other techniques, we have discussed in last class. So, that the faulty rows of the memory chips are I mean are virtually taken out, and the normal redundant parts which are already there are rerouted.

So, that I mean the faulty parts of the memory chips of all the rows of the memory chips are bypassed and in place of that the normal rows are brought in. And there existed actually done that by virtual blowing out some fuses some of the some of the connection in the chip by using fuses. Using some lizard technique or some other techniques, which we have already discussed.

Secondly, the thing was that in case of circuits is the very seven symmetric structure and all the cells of the memory chips are packed very close to each other. So, unlike

sequential circuits or combinational circuits the fault here are very regular, structure is very regular in nature. So, faults are generally stuck at 0 stuck at 1 they are already there, but along with that there are several other staffs. Like 1 cell can be having interface can having interference problem with some other cells. So, the other neighboring cells may dominate on that 1, 2 cells may be having dominating effect on other and so forth.

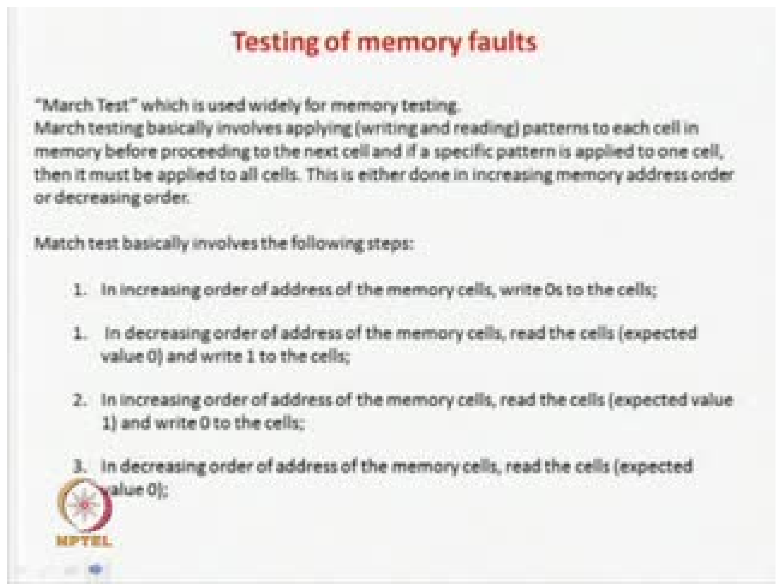
These are all the problems of memory testing, but what is the ease of memory testing? The ease of memory testing compared to sequential combinational circuits is that, in case of sequential and combinational circuits, we have to sensitize the fault propagate through output so forth. So, they may lack in to inconsistency and all those things.

But in case of memory testing, we do not have to do anything. We have to just write something in the cell and read back something in the cell. So, the question of propagating, sensitizing, justifying all those complaints does not come into picture at all. So, you just sensitize the fault that means, you just write 0 and 1 in the respective cells you required, read them back and your job is done. So, although the fault modules are bit complex over here, but the way to generate the test pattern applying them is much easier than compared to other combinational and sequential circuits. So, that we have already seen.

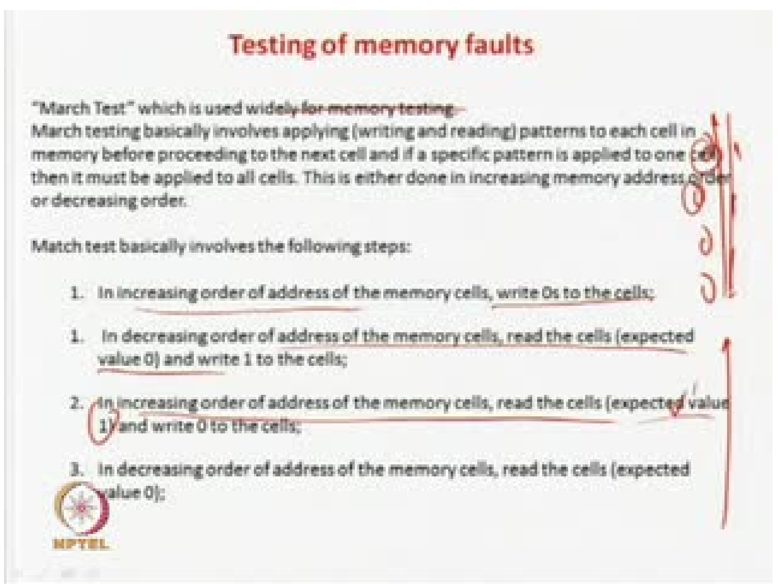
So, now in today's class that is on lecture 4, what will see that how can you test the memory blocks or memory chips by using the philosophy that the cells have to be written into some values and you have to read them back, and that is it and that will be done in a single sequential manner or minor regency and that should be done in a proper way.

So, if that can be done then you can say that the chip is normal or fault , as a whole the chip may not be having may have some rows, which are having defective rows or cells which are having defects. So, after wise there are some redundancies if you have a 1GB memory kind of a stuff. So, you will have more than that and the faulty rows or the faulty cells are bypassed and the normal or redundant normal rows, or normal cells from redundant parts are bought in place all those faulty parts by some switching mechanism and the switches. Switches are blown by some lizard technologies or some other technologies. That is the idea.

(Refer Slide Time: 03:06)



(Refer Slide Time: 03:52)



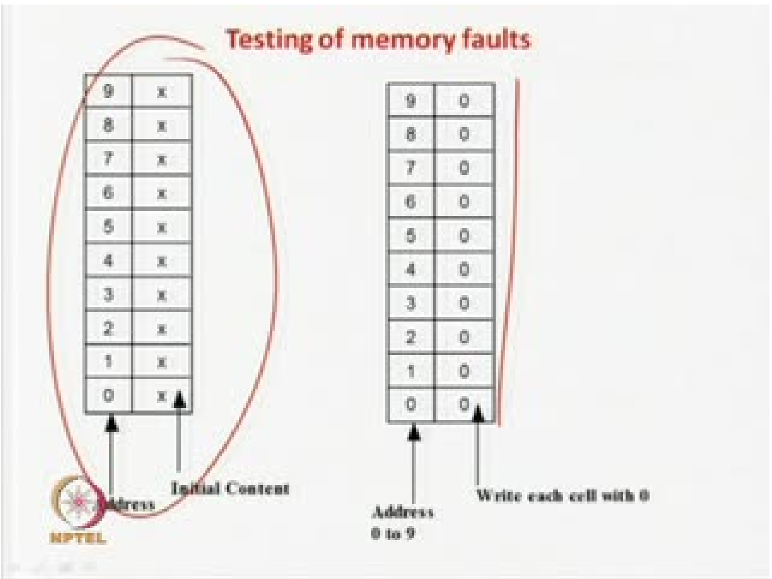
So, today we will see what is the most important thing, which is called the March Test which is I mean the term 1 of the March, which is applied widely used for memory testing and several others, but we are actually looking for the March test. So, March test basically involves applying writing and reading patterns to each cell in the memory block before proceeding to the next cell, and if a specific pattern is applied that is the idea. So, that is basic idea of March testing is that, you read and write patterns in each cell before proceeding to the next cell in a predefined manner that is what is the idea.

So, either you go in increasing order of the memory or go in decreasing order. Either you go from 0 to 2 to the power of n memory location or you come down from 2 to the power of n to 0 and you apply some specific pattern if you require in a particular order. You may be writing 0, 0, 0, 0, 0, 0, 0 then you may be writing 1,1,1, 1, 1, 1, 1 and so, forth or you may doing 0, 1, 0, 1, 0, 1, 0, 1 something like that. So, that is actually a specific pattern is required to applied in 1 cell, then it may be applied to other cells and so forth.

So, the basic idea is that you go this way or you come this way and you either apply all 0's and all 1's or alternatively way with some specific patterns also. So, that is the basic idea of March testing, that you go from 1 direction and then you come back in another direction. Keep on applying some specific patterns as whatever required to the cell and read them back.

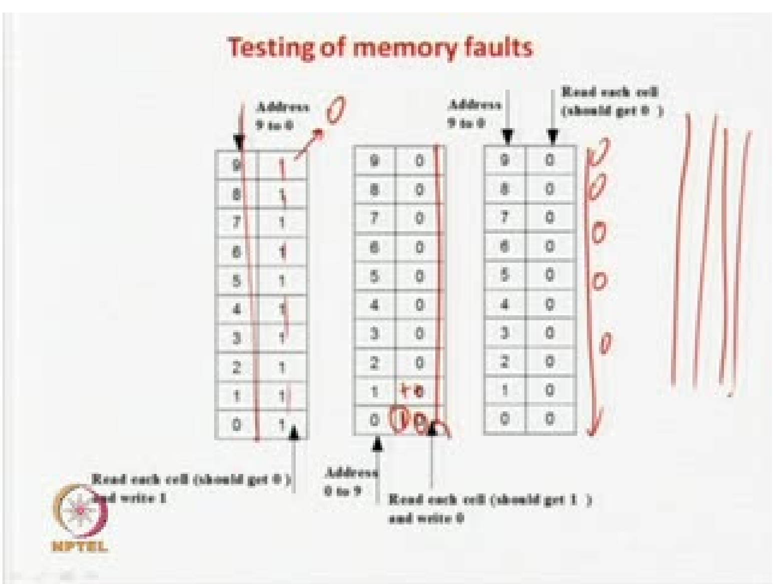
So, March test basically involves the following, in increasing order so, take the very standard definition you go up and come down. So, in increasing order of memory address write 0's in the cell that is the first job. This is the very preliminary ideas of March test there can be some different complex versions. So, what you do? You keep on writing 0's from top to bottom and then decreasing order of the cells, read the cells and write 1 to the cells. Now what you do? From top you come down and you read each of the cells and if their expected value is to be 0 because, you already have written 0. So, you should read as 0 and you write 1 and you read a 0 and when successful you write a 1.

(Refer Slide Time: 05:46)



Now what you do, so this way you have done, now again you go up, now increasing order of cells, read the cell values, expected value is 1 because you have written 1 write 0's to the cells and then again repeat it. So, let us see this in a pictorial manner. So, initially this is your memory contents. So, this will be always be content in this way nothing but arbitrary values are there. So, you assume that our memory cell have 0 to 9 I mean cells .So, that in the order of 2, but for the sake of simplicity of putting from 0 to 9 So, what you do? You just keep on writing a 0's in all the cells. So, you can write a 0.

(Refer Slide Time: 06:07)



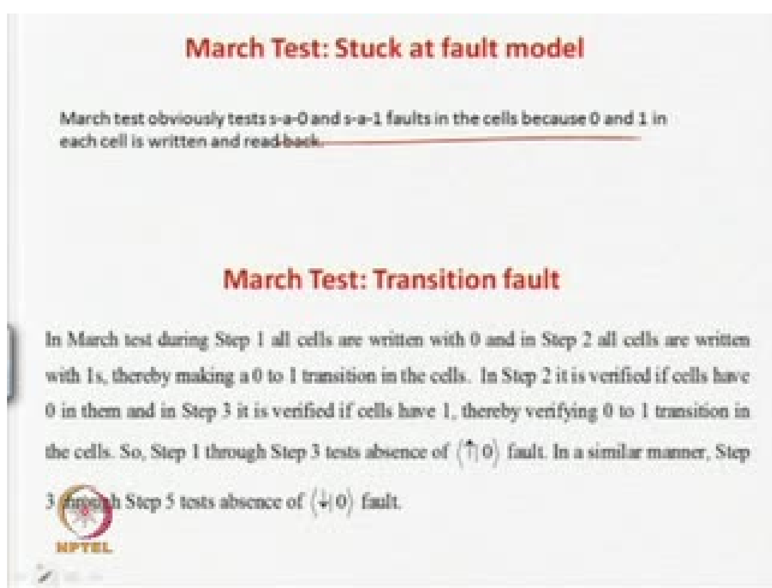
Now, what you do? So now you come down address from 9 to 0 and you read a 0 because already 0 was been. So, you read the values, so, it should be a 0. Either 0 that means your right 0 was successful and read 0 is also, successful. Now, keep on writing a 1. Now what you do? Now again you come down, again you read the cell. So initially the cell was 1 because it was 1 in previous cell. You read 1 you get a 1, then you are very happy that you have written a 1 and you could have read 1 also. Now again write a 0 and keep on doing it and read 1 erase it and write it 0 and so, forth. Again you come down and you should expect to get a 0. At this stage you do not write anything, means you just keep on getting a 0.

So, this is what is the basically March test for memory. You go up write all 0's, come down and read 0's successfully write a 1, go up read 0, 1 successfully write 0s and you

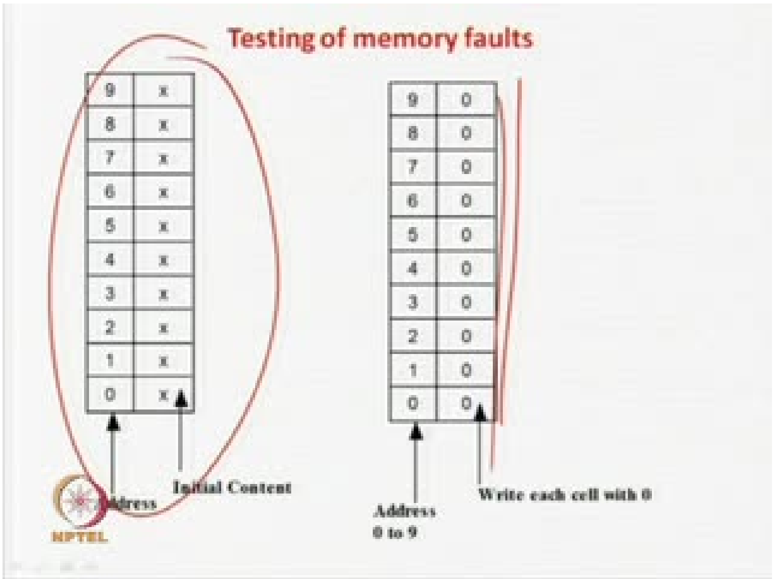
come down and read all 0s successfully. So, that is the very basic idea of memory test, March test based memory for memory.

Now what will we see next? How these tests? Which are the fault false models? Which you have already seen are covered by this and which are the fault models which are not covered by the March test. So that we have to twist our idea a bit. So, that is what we are going to see. This is very basic idea of March test.

(Refer Slide Time: 07:23).

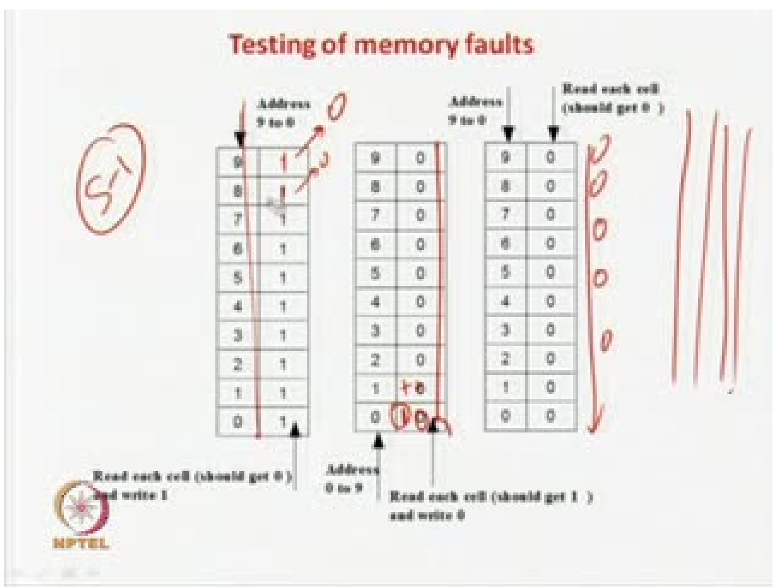


(Refer Slide Time: 07:30)



Now we will see which false fault models are covered by this and also see some of the models are not covered by this. So, we will see I mean and then some changes we have to do. So what is the idea? For the stuck at 0 and stack at 1 fault module. So, you see very easily we can see that stuck at 0 and stuck at 1 fault are covered by this test. So, you write a 0's in all the cells done.

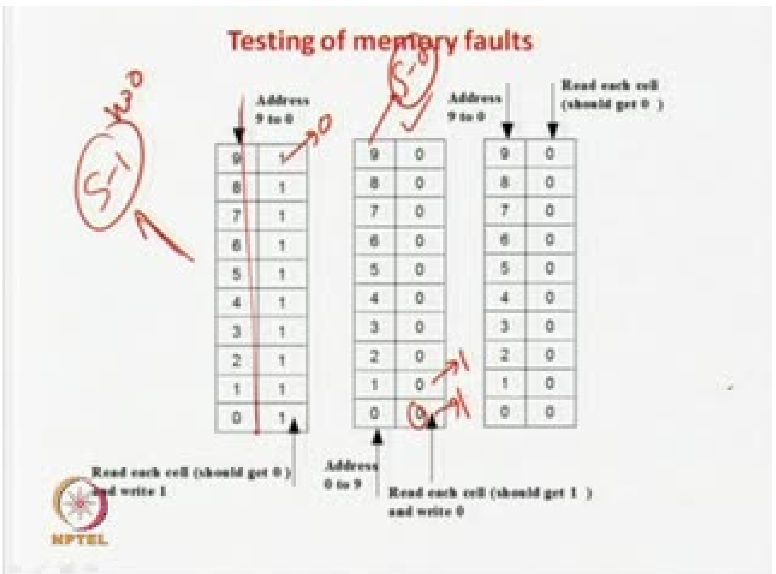
(Refer Slide Time: 07:50).



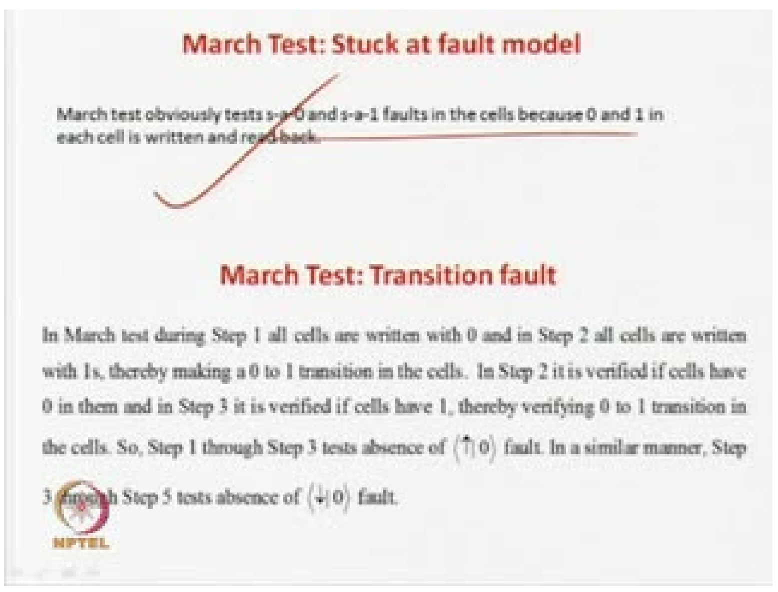
Now you keep on reading all the cells from the back ok? So you are reading it back that this is 0 fine so you are writing a 1, you read 0 fine write a 1 and so forth. So, that means what? There is no stuck at 1 fault in the memory that is not possible that is the 1. Now what you do? In second step what you have to say you have written all 1s. So, ok you know that because you are successfully reading as 0 and writing a 1.So, you can be very, very sure that there is a no stuck at one fault in the memory or no stuck at fault in the memory.

Now the next thing what you have done? You have now once in this step what you do? This assures that there is no stuck at fault no stuck at fault, because you could read 0 from all the cells. Now what you have done here? Now when we are going here, you are reading a 1 and writing a 0, reading a 1 and writing a 0. That means what? There cannot be any stuck at 0 faults because, you haven't write 1 here and reading. Obviously, March test actually does successfully all these stuck at 0 and stuck at 1 faults.

(Refer Slide Time: 08:10)

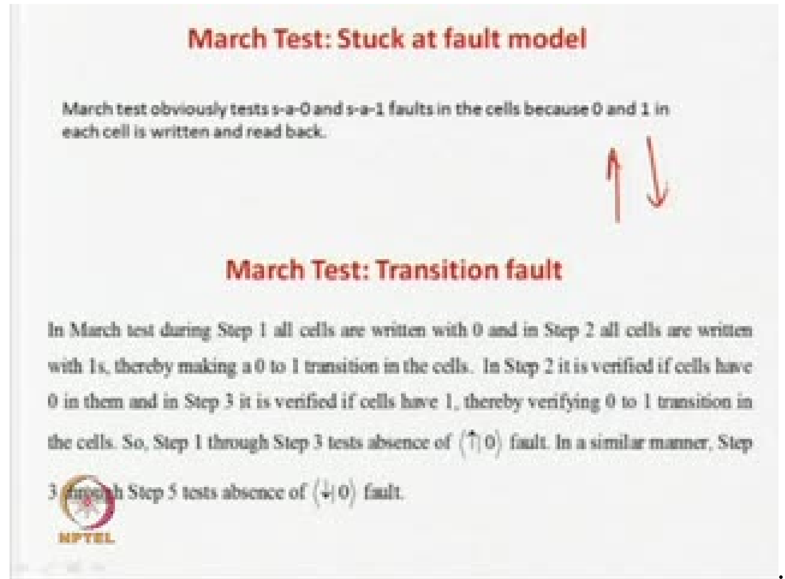


(Refer Slide Time: 08:29).

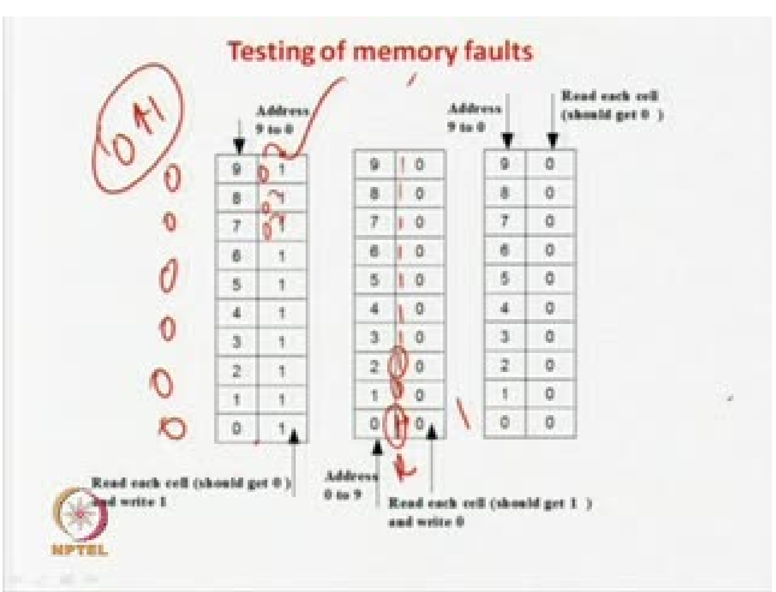


Now is the transition fault. So, what are the transition faults? Transition faults are nothing but. You should able to go from 0 to 1 and you should be able to go from 1 to 0. That is what the idea is. So, in this what is defined here.

(Refer Slide Time: 08:38)

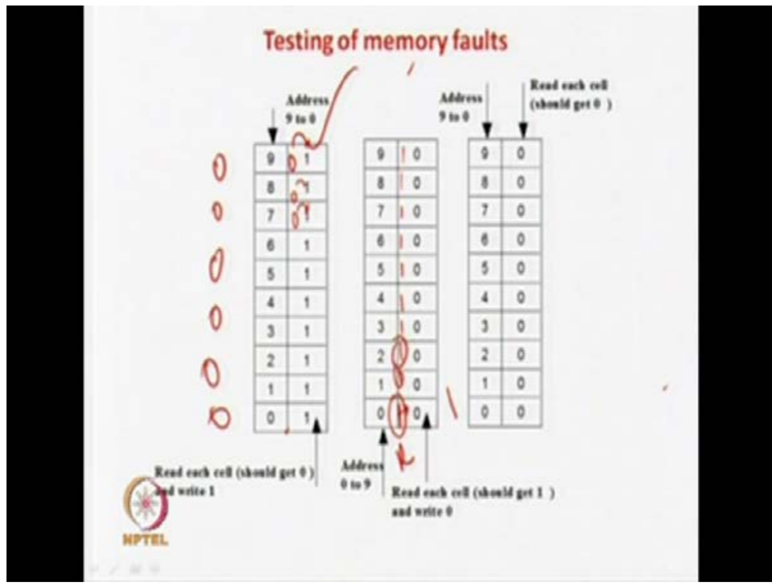


(Refer Slide Time: 08:57)



Now, let us see, how it is successful, so, what we are doing here? So, in the first step we see, we are going to write all the 0s. Next step what we are doing? We are reading a 0 correct and writing a 1 reading a 0 and writing a 1. Reading a 0 and writing a 1. That is what we are doing.

(Refer Slide Time: 09:43)

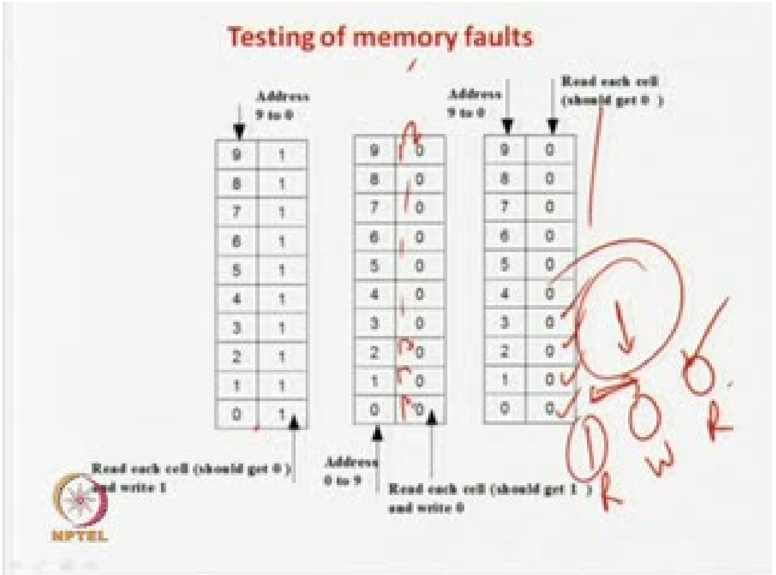


Now here what are we doing? We are reading a 1 and writing a 1. That means what? So, you are reading a 0 here, instead of reading a 1 writing a 1 here. That means, what here are in first step was, all this stuffs were 0.Now you are reading a 0 here. That means, that successfully 0 was written here.

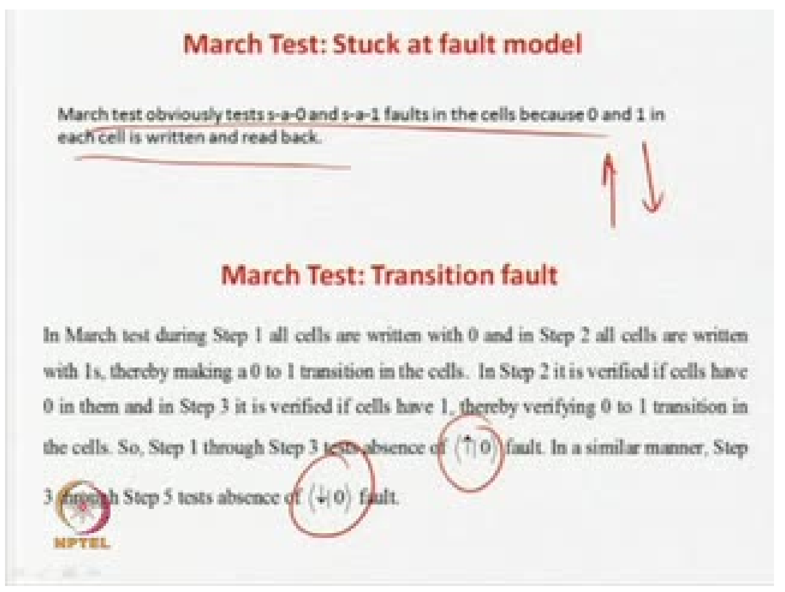
Now you are writing a 1 done? Now in this step you are reading a 1. That means what? The 0 to 1 transition was successful. So, if you read all 1s here and you are writing a 0 of course, but you are reading a 0 and writing a 1 here and reading back the 1. That means what successfully you could go for 0 to 1 in all the way. So, a raising transition, what do you call rising transition fault is not there. So, this is actually raising transition fault that is not present in any cell that you can guarantee. Now let us see the other way, the 1 to 0 the falling transition faults is also not there, that can also, be assured. So, Let us see that how it is possible by March test. So, now in this case you can see, so now.

This was about this 1 here a reading a 1 and writing a 0, reading a 1 and writing a 0, reading a 1 and dot dot dot dot, reading a 1 and writing a 0. Now in this case what you are doing? So, again you are reading a 0, reading a 0, reading a 0, reading a 0 and so forth. So, what you have done essentially? You are reading a 0 writing a 0 and again back reading a 0. That means what? These 2 steps actually are fall transition you have made and here you are very fine. So, reading a 1, writing a 0 and reading back 0 guarantees that there is no what you call fault transition.

(Refer Slide Time: 10:04)

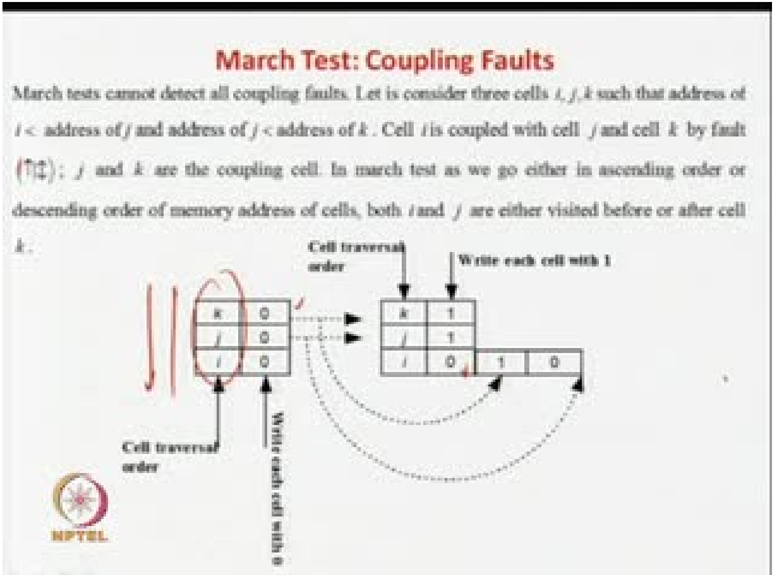


(Refer Slide Time: 10:33).



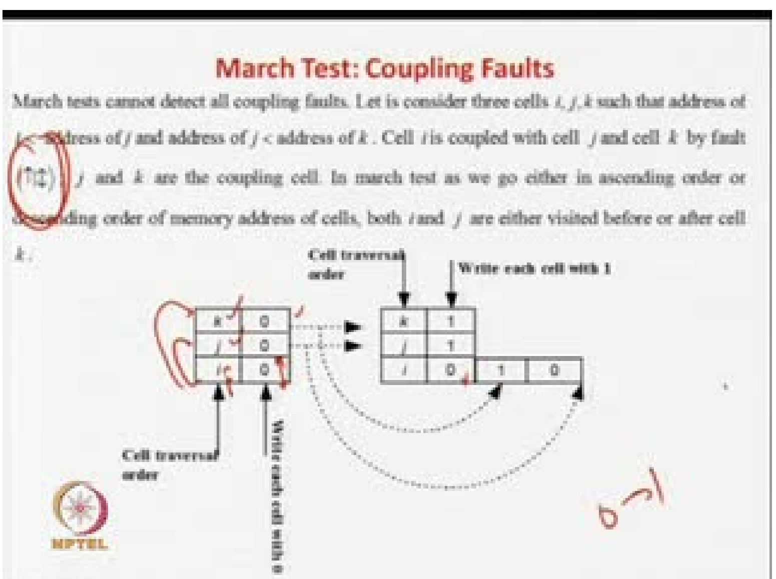
So, raising transition fault and down transition fault are also mean you can test it very well. So, March test actually takes full care of stuck at 1 stuck at 0 faults and transition faults. Now we will see coupling faults. Do March test cover coupling faults? The answer is actually no. So, that you will be very surprised to know that March test can very easily go on for twisting its transition faults and as well as your what you can call this stuck at 0 and stuck at 1 fault. But March test as of now I mean whatever we have discussed we all form does not cover coupling faults. Let us see why?

(Refer Slide Time: 11:05).



March test cannot handle coupling faults. So, how it is done? You just take 3 cells i, j and k. The routine is i is the lowest and j is this. This is the order of memory cell location. So, in March test sequentially we will go this way and we will come down this way. That is what is the idea.

(Refer Slide Time: 11:22).

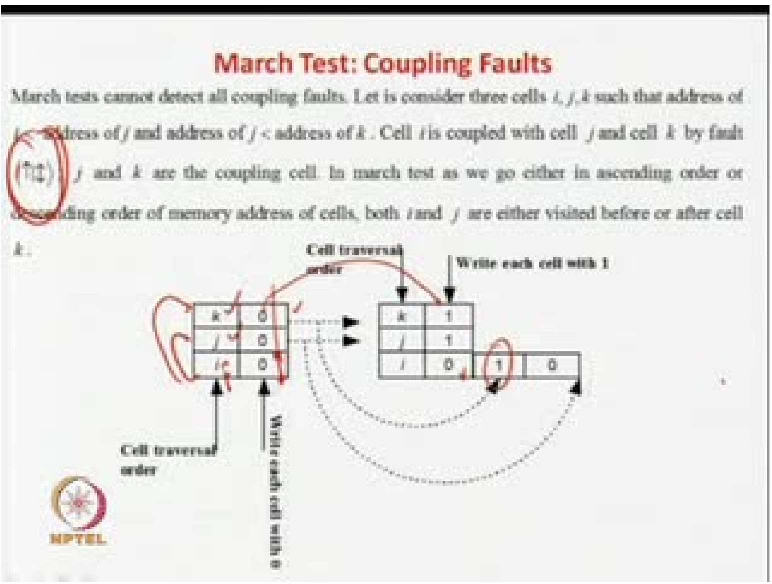


So, now we will see that the coupling faults cannot be detected by March test means what is the idea? So, what is the coupling cell? So, i is coupled with j and i is also, coupled with k, it is coupled with both. So, this is the coupling cell and f is the coupled

cell, so fault will occur here. J and k are dominating cells. That means, that this guy will control f and cause fault over here. So, what is the fault?

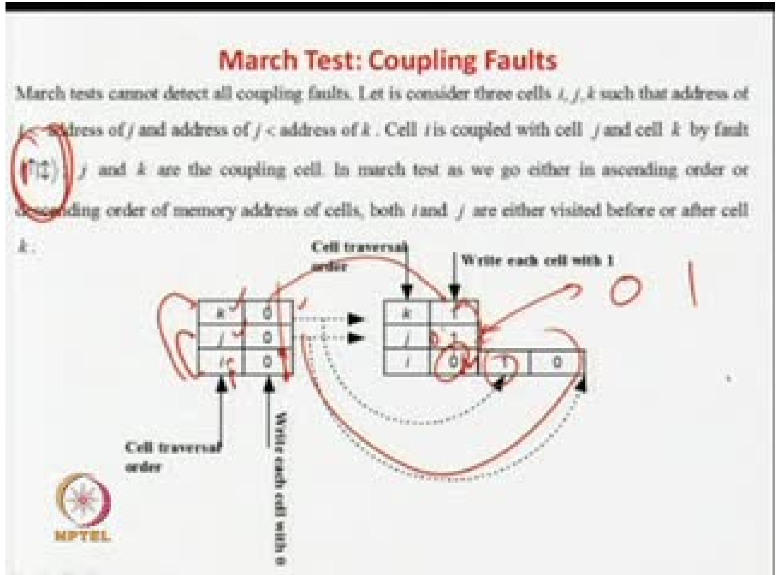
This is the fault. So, whenever there is a raise in cell k or j that is from 0 to 1, then there will be a problem that f and there will be a in swing in i and that is the fault written over here. If there is a change in 0 to 1 in cell j k there will be a change in f from this k also. That is what the phenomena. So, that is what the fault we are taking and we will see that March test cannot detect this fault. Now what is that?

(Refer Slide Time: 12:02).

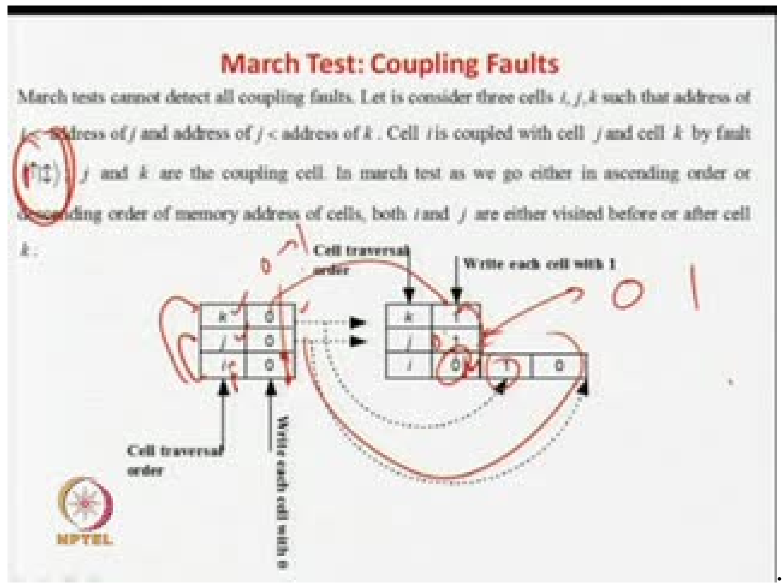


So, as you know in March test we write all cells as 0s done. So, now what you do? First you in up and when you are going down. So, you change k from read 0 successfully and write a 1. So when it is done? So, this as a fault is there as couples are there will be a swing it will be converted into 1. So, this 1 corresponds to swing over here now done. If you have directly read cell i, then you you’ll read that there is a 1 in the cell i which is a wrong and you can detect fault. But in March test we cannot do that because, you have to go in a sequential order. So, now we next come to this cell, read successfully a 0 , write a 1. So, that is what again a rise over here from 0 to 1. Right? So, now again there will be a swing over here.

(Refer Slide Time: 12:28)



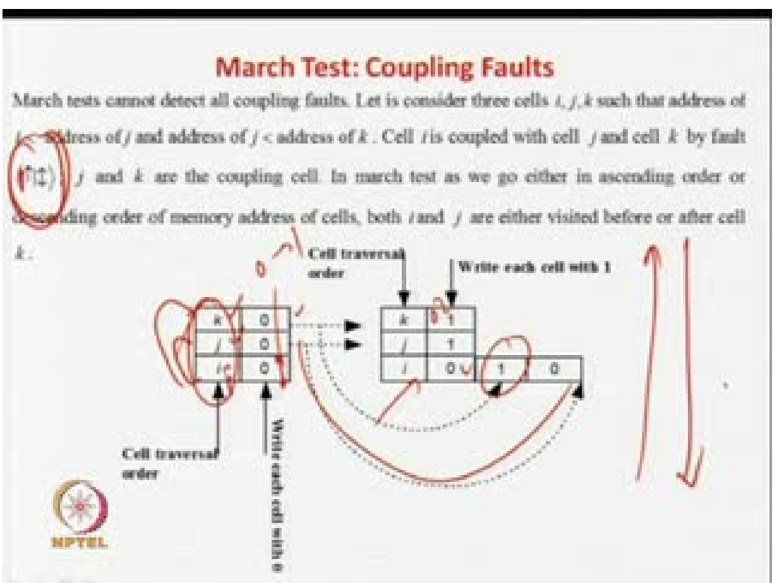
(Refer Slide Time: 012:53)



Now what will be the swing? The swing will be from 1 to 0 which corresponds to this one and this one and whenever reading cell 0, you will find a read 0 successful and you can write a 1 over here. So, even if there was a coupling fault between this you cannot detect it. So, there was a fault masking. So, what are the fault masks? J k and i was coupled change in i, change in k from 0 to 1 that is rising frequently. Now, when there is again a flip in j from 0 to 1 again, i was again flipped and it was made correct. So, fault would not be detected. So, there was a mask.

So, you could see that coupling faults cannot be detected by March test. So, that is the problem because of this interrelation between these three things. But, if you could have first applied the cell here and then you could have checked here, then your job would have been done.

(Refer Slide Time: 13:20)



That is you go for a 0 to 1 kind of read 0 and write 1 then directly check here. So you will get a 1 you can, but as March test go in this sequential and in this sequential order. So, you have failed in doing that. So, that 1 is the one of very big problems in the March test that we could not handle the coupling faults because, we are traversing in this order, but if you could have I mean if you could somehow bypass this what you call sequential traversal manner, then something you can be done. That we will see later.

As of now, the confusion is that March test cannot handle coupling faults. So, that is what whatever I discussed on cell raising, inversion raising inversion coupling fault between i and k mask that is the fault between this i and k , that is the fault between i and k is actually masking the fault between i and j. So, whatever we discussed in other words, we are not success full in doing it.

(Refer Slide Time: 10:54)

March Test: Coupling Faults

As Step-1 of March test all the cells i, j, k are written with 0. Following that in Step 2, all the cells (in order of) k, j, i are written with 1 (after successful reading of 0 from the cells). It may be noted that first cell k is written with 1; as cell i is coupled with cell k having fault $\langle\uparrow\updownarrow\rangle$, the 0 to 1 transition in cell k inverts the content of cell i. Following that, cell j is written with 1; as cell i is also coupled with cell j having fault $\langle\uparrow\updownarrow\rangle$, the 0 to 1 transition in cell j inverts the content of cell i again. Now when cell i is read, the value determined is 0 which means absence of two coupling faults (i) rising $cfinv_{i,j}$ and (ii) rising $cfinv_{i,k}$. In other words, "rising $cfinv_{i,k}$" masks "rising $cfinv_{i,j}$".

(Refer Slide Time: 14:15)

March Test: Coupling Faults

Inverting rising coupling fault $\langle\uparrow\updownarrow\rangle$ **between cell** i ***(coupled cell)*** **and** j ***(coupling cell)*:** (i) Cell j is to be written with a 0 and read back, (ii) value at cell i is to be read and remembered, (iii) cell j is to be written with a 1 and read back, and (iv) value at cell i is to be read and checked that it is same as the one remembered (i.e., no inversion has happened).

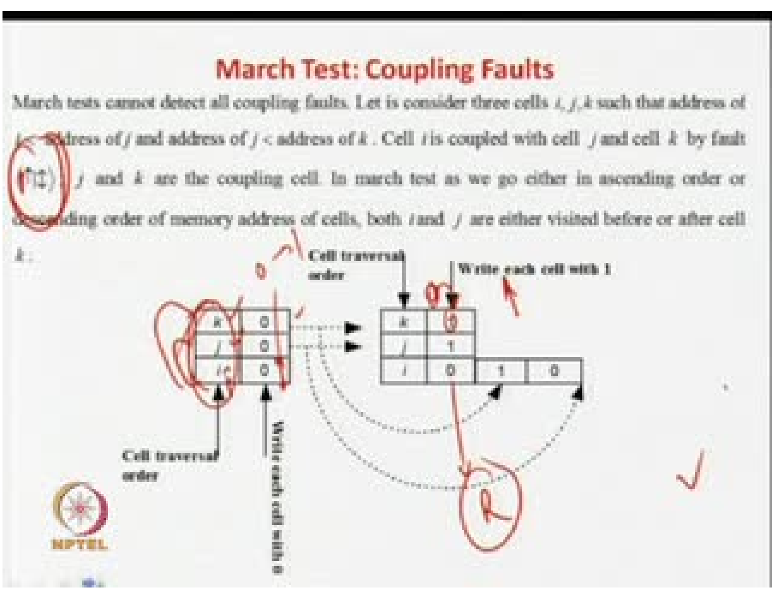
Inverting falling coupling fault $\langle\downarrow\updownarrow\rangle$ **between cell** i **and** j**:** (i) Cell j is to be written with a 1 and read back, (ii) value at cell i is to be read and remembered, (iii) cell j is to be written with a 0 and read back, and (iv) value at cell i is to be read and checked that it is same as the one remembered (i.e., no inversion has happened).

So, now we will see I mean how we can help actually. So, that is what I mean we found that that was actually a big problem because of the sequential traversal we could not do that. So, now we have to see that how problem can be solved. So, let us see that inverse raising coupling fault in this case one, this is a cell from 0 to 1, 0 to 1 transverse if it is there it is the cell k say in the cell i there is actually a swing over here. So, this is the cell dominating cell, the coupling cell and this is the coupled cell where fault is there.

So, now what you can do? Cell j is to be written. So, what have to do the test? So, you read a write a 0 over here say 0 is to read and cell i is to be read and remember, whatever is the value is, write a o in k and remember whatever is the value So, you know that you write in k and remember the value that is x is 0 or 1 it is possible and read back. Then you read them now cell j is to be written with a 1 and it should be read back successfully you have done.

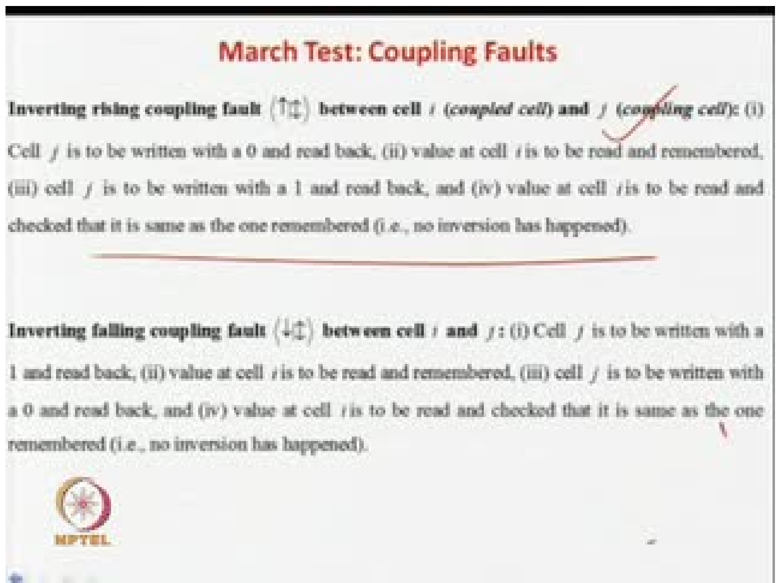
And then again you read back this value of cell i and the value should be same it should not be x inversion. So, this should be remembered and no inversion should be done. If there is a case, then you know that there is inversing raising coupling fault between cells i and j is longer that you can assure. So, basically what we have done? So, we have done a very simple logic.

(Refer Slide Time: 15:37)



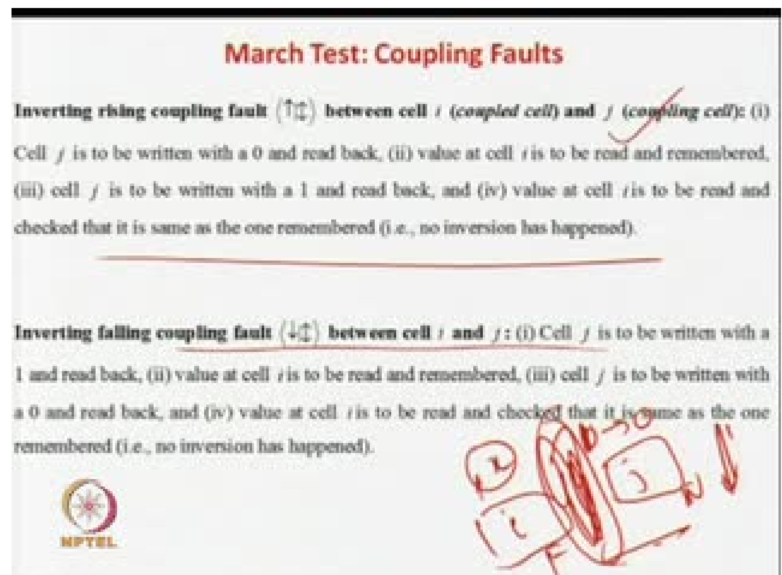
So, we have seen what we have done, basically as I was discussing as that is same thing we have done. So, we have written a 0 verified it, we have written a 1 over here sorry we do not have 0 over her. Read the value of i over here 0 or whatever may be the case, then you write a 1 over here and read back 1. So, that is you have successfully gone for a raising point here, then you are reading back the value of cell i. So, it should remain 0 it should not become 1. So in this case if you have done.

(Refer Slide Time: 16:02).



All this solve this as essential of this have done only March testing and not done anything else, but if the modified March test in which case we have not gone in a sequential way. We have traversed, if the cell i and j are coupled and coupling of it so, we go for i and j in 1 shot 1 after another will be more sensing between i and j.

(Refer Slide Time: 16:18).

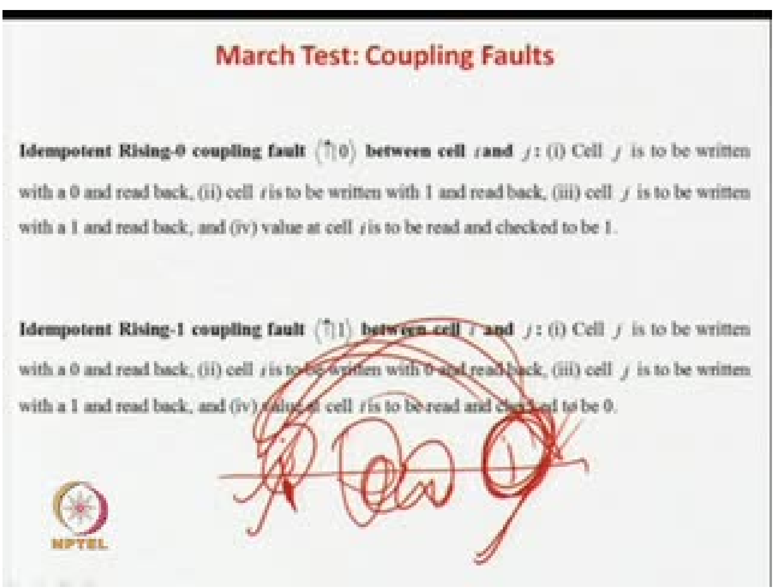


Similarly, for the inverse coupling fault you can do the same thing. For the cell i say this is cell j say this is the normal cell this is the coupled cell fault causing so, here you write actually a value of a 1 and 1 here read back then you read the value of i that is x tell c is

to be 0 write a 0. So, there is a fall transition you have done. Again read back 0 and same thing you assure that i, j is to be 0 write a 0 and j verify that 0 is there, so you verify that down transition here, again read back i and there should not be any x inversion.

So, then you can verify that inversion coupling fault is not there. So, actually again what we have done, we have basically we have done traversed and j. Even if there is lot of other cells in between we have done the testing 1 of the consequents I mean we have done excess i and j consequently even if there are many other cells in that. But, in this simple idea we can use March test for coupling faults.

(Refer Slide Time: 17:25)

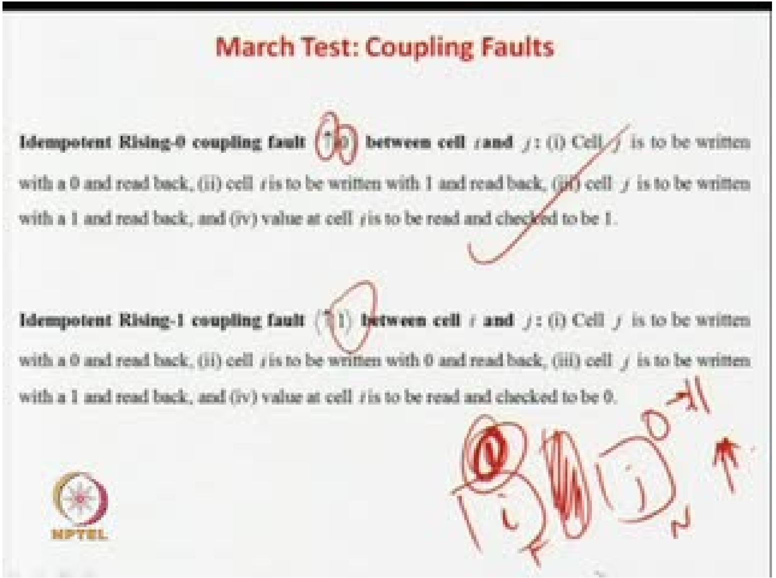


So, similarly, I mean that basically coupling happen faults cannot be handled by the raw March test now why is that because, in March test you go this way and you go this way say if there some fault i and fault j, fault l and fault f something like this. If these 2 are getting coupling effect, these 2 are getting a coupling effect kind of a thing sorry 1 example is fine is always a fine. So, this 2 are coupling effect there will be a lot of cells between it.

So, if you are doing something with this cell and you apply something with this cell then all what happened it may get some making effect from the example. And if you apply something and read something again you apply something and read something again, and you done jump between these 2. Then there is no effect actually or intermediate cells

affect may not come into picture, and we can detect if there is any kind of a fault over here.

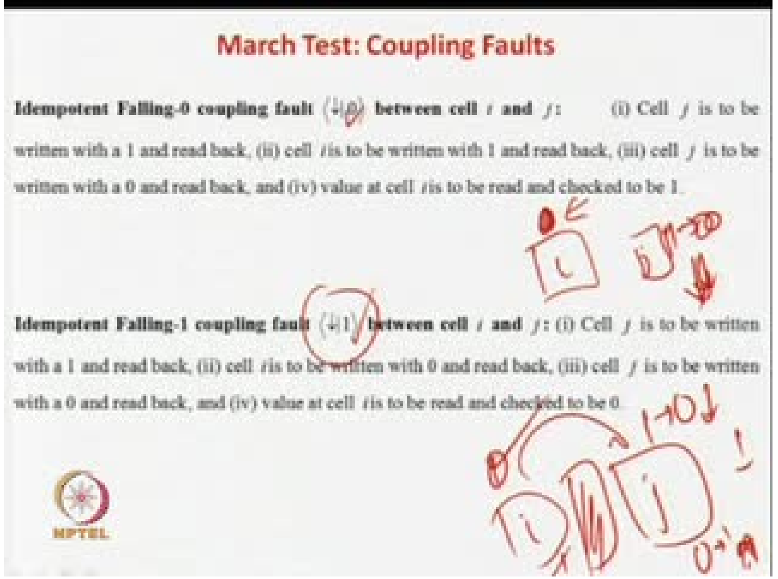
(Refer Slide Time: 18:11)



That can be very easily done. So the idea here is that somehow you have to bypass this cells which lie between i and j so, that your job is actually done, that is the idea. So, now simply you can see that raising idempotent raising coupling faults rising from this is a cell i say j if raising over here, you will get a 0 and the coupled cell that should not be in this case. So, it has to be verified with the 0 and read back and you write a 1 and read back right read back right and cell j is to be written with 0 and read back and then you write a 1 over here and read back and cell j is to be written 1 and read back that is the rising transition you have done and then you read again you read a cell i, so obviously it is 1 and it is done.

So, similarity if you want to go for this impudent raising 1 fall, then what you do? You write a 0 over here write a 0, i read it, ok. Then cell j is to be written with a 1. So, that you get a raising transition here and read back the value of cell. So, that we know that this fault is not there.

(Refer Slide Time: 19:30)



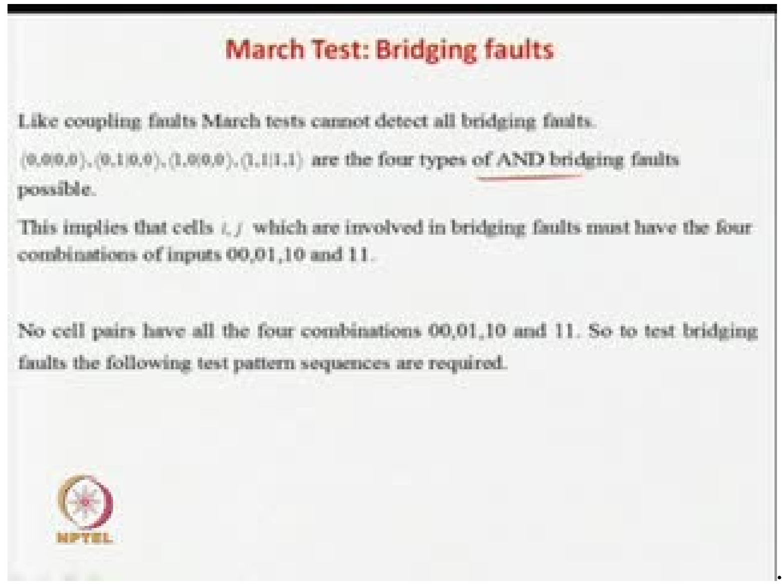
It is very simple the idea is that instead of enabling i and j are not consecutive, may not be consecutive and lot of memory cells are there traversed form there to here. So, very simple logic we can go for this test. Similarly, for idempotent falling coupling fault and idempotent falling 1 coupling fault. So, again this is very simple this is i and this is j th, these are falling. So, you have to write a 1 over here and you have to write 0 over here and read back. You have to write a 0 over here and read back. So, fall is done again go back check that it should be 1. So, write a 1 and read back 1 sorry here is a 1 read back 1. So, it should be 1. So, this is done.

Similarly if you want to go for this 1 this thing. So, you write a 1 over here write a 0 over here and then write a 0 over here read back in this then fall is assured then go back and check 0 is there 0 is there. If 0 is there, then this fault is not there. Because, this fault actually says that whenever a fault is there in this case you will not get 0, i will get a 1.

So, very simple idea by this simple technique I mean first writing j and writing i value of 1. So, if you are looking for this type of fault you write a 1 over here, if you want a checking raise fall 0 and whatever. If you want to test for 1 fault here so, you write a 0, 0 faults you write a 1 then again if it is a 0 you write a 0. You are checking for this 1, you write a 1 if you are checking for this 1 read back this if you want to check for this, read back this, if you want to check for this also. Very simple and a very, very simple idea.

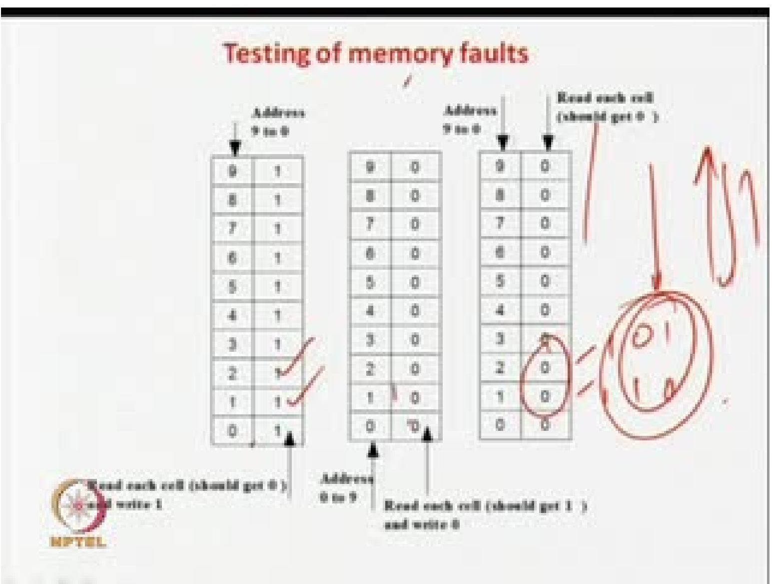
Only you have to note that, there is a lot of cells between them doing about this. Again found the bridging faults.

(Refer Slide Time: 12:02)



Now, let us come from the bridging faults. March test does exactly this again we saw that we are taking AND bridging faults gates. We can see that 0,0, 0, 1, 1, 0 and 1 1.

(Refer Slide Time: 21:25)

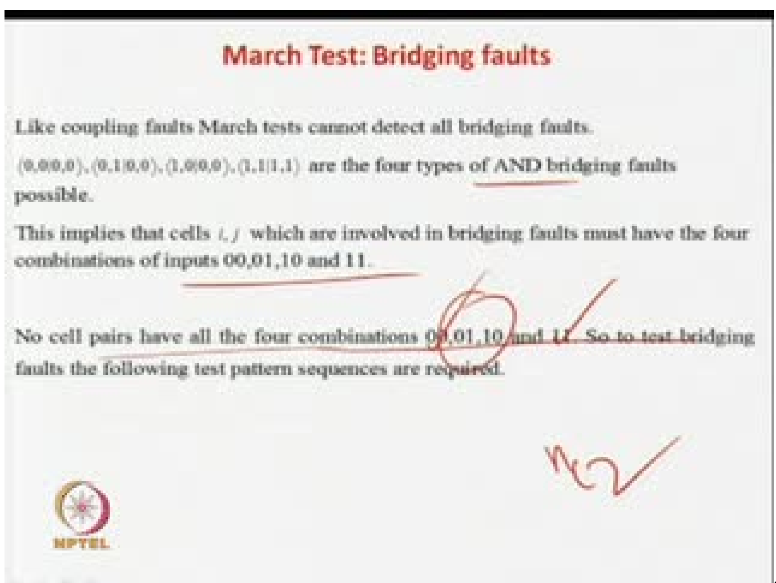


So, all the patterns has to be applied in between all this cells. That is the idea. So, any 2 cells can have a coupling effect. So, what is the idea? You should have? For using fault

test for bridging fault test you have to assume that we are actually saying that we are cross considering these 2 cells fault. So, you should have 0, 0, that is fine. Then you should have 1, 1 that case is also fine. But, at that time you should have 0, 1 and 1, 0 test should be there and we verify that this 2 coupling combination may not be possible 0, 1 and 1, 0.

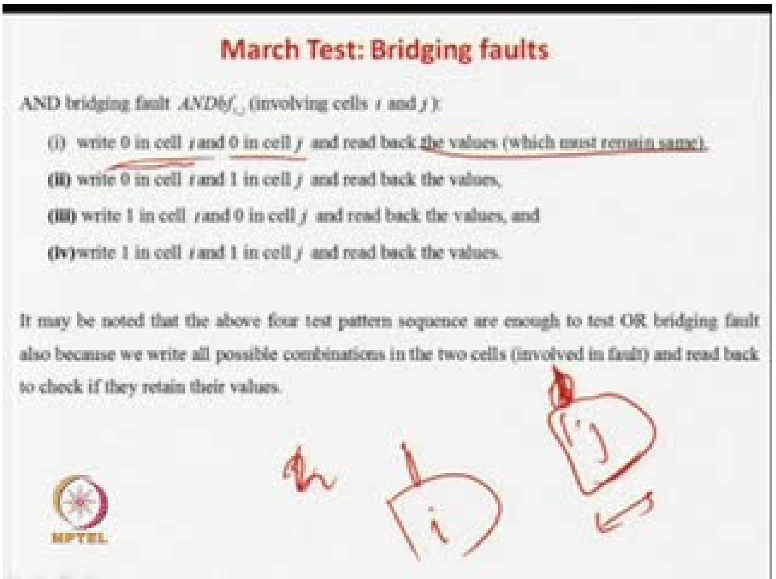
These 2 cases may not be possible or not possible in memory in March test also, if you go strictly in this and this and this that may not be possible. So, just again as you have modified the March test is the best in case of coupling faults, so, you do the same thing for the bridging faults. So, implies that i, n and j must have 4 combinations of this 1.

(Refer Slide Time: 22:02)



So, no cell pairs have all the 4 combinations that is true for each n c 2 is the combination I mean all pair of cells should have combination 0, 0,0 ,1, 1, 0 ,1, 1, 1, 1 is fine as I think it may be there, but 0, 1 and 1, 0 may not be there. That you can verify and subsequent tests are there.

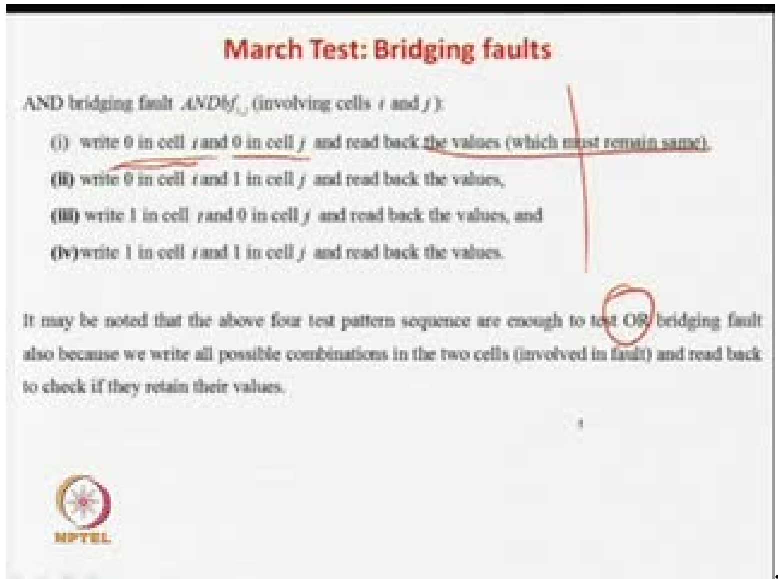
(Refer Slide Time: 22:22).



So, what you have to do is a very simple idea. Again i, this is j say j both bridging faults both of them have an effect. So, you write 0, read 0, write 0, read 0 and again go back. So, write cell i and cell write cell 0 in cells i, 0 in cell j, you write both of this things and read back the values (( )). Similarly, you write a 0 over here, write a 1 over here and read back both and both of them should be 0 and 1.

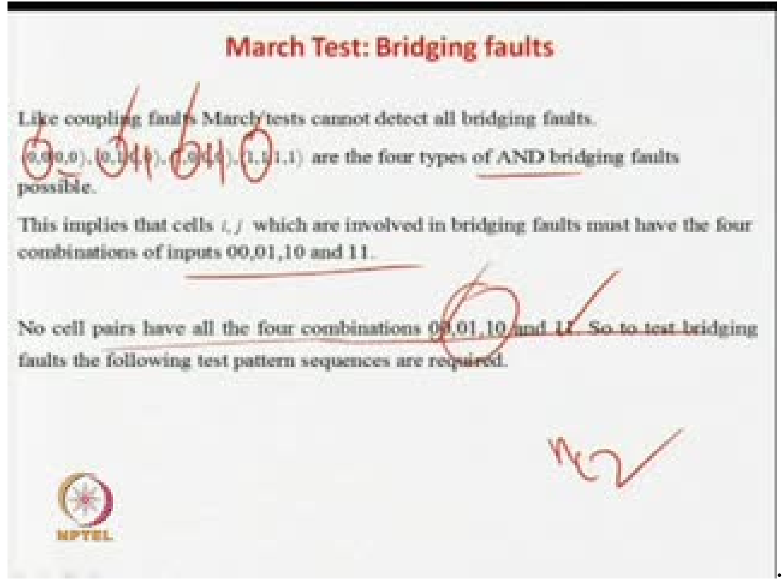
Then you write a very simple idea you write a 1 over here and 0 over here read back both the values, it should be 0 and 1. Similarly, you write a 1 here and write a 0 over here and read back and both should be 1 and 0 and both should be same. There should not be any kind of the change.

(Refer Slide Time: 23:08)



So, now you see that is what we are expecting. So, as you know that these are all the test for AND bridging faults and similarly, these are all OR bridging faults also, because OR bridging faults and AND bridging faults whatever you take.

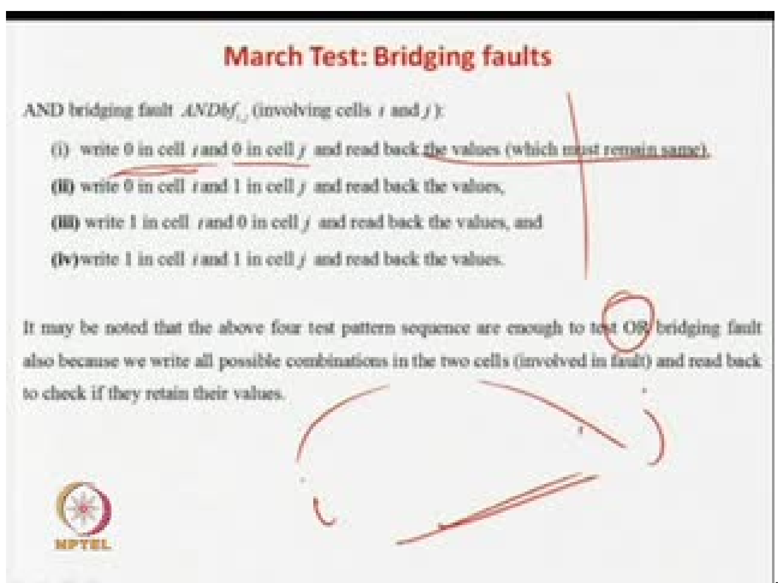
(Refer Slide Time: 23:22)



So, you will find out that 0, 0, 0, 1 these are the only 4 conditions have to be applied either in OR bridging faults or AND bridging faults. What you are verifying is that if you write 0, 0 you should get back 0, 0 and if you are writing 0, 1 you are going to get 0, 1. If you are writing 1, 0 you should get back the value 1, 0 and 1, 1. So, this is what verifying

the memory in case bridging faults and for OR also, you do the same thing only that the fault affects will be different, because, in this case it will be 1, 1 and this will be 1, 1 and so on.

(Refer Slide Time: 24:22)

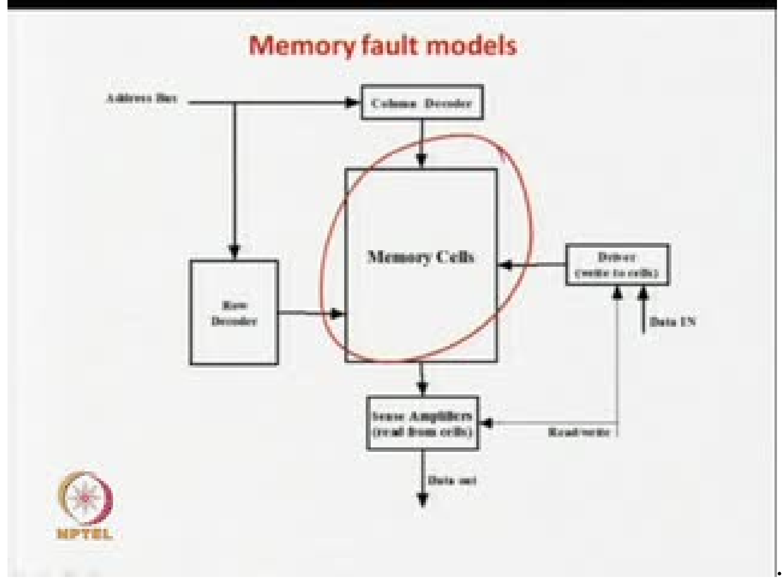


So, we are actually just verifying that in case of bridging fault we are verifying that whatever we are writing 0, 0, 0, 1, 1, 0, 1, 1, they should remain as such. So, we are not saying that what is the output whether it belonging to AND bridging or OR bridging? That is not important. We are verifying that we have written 0, 0, 0, 1, 1, 0, 1, 1 and they are retained. And if they are retained our job is done. So, obviously, it will be AND bridging or OR bridging. So both of them are tested by the same method.

(Refer Slide Time: 24:16)

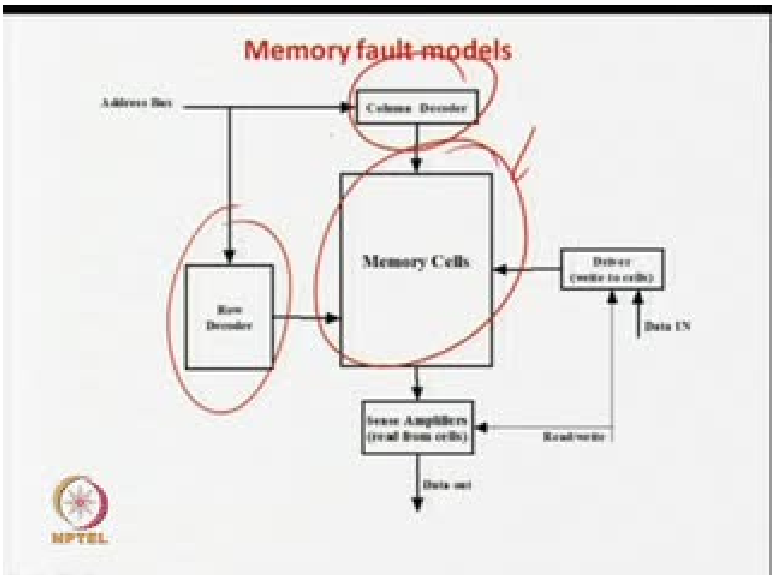
That is what just by this I mean these are all also you can call March test, but only thing is that we are not going in a sequential manner from i to j because i may be here, j may be here. A lot of intermediate cells will be we are jumping between these 2 locations, that is what is the idea. Now you see, now we come back to our what you call these address decoders. So, what you mean by the address decoders faults? So, the address decoders faults means as already we have seen that this architecture was.

(Refer Slide Time: 24:47)


.

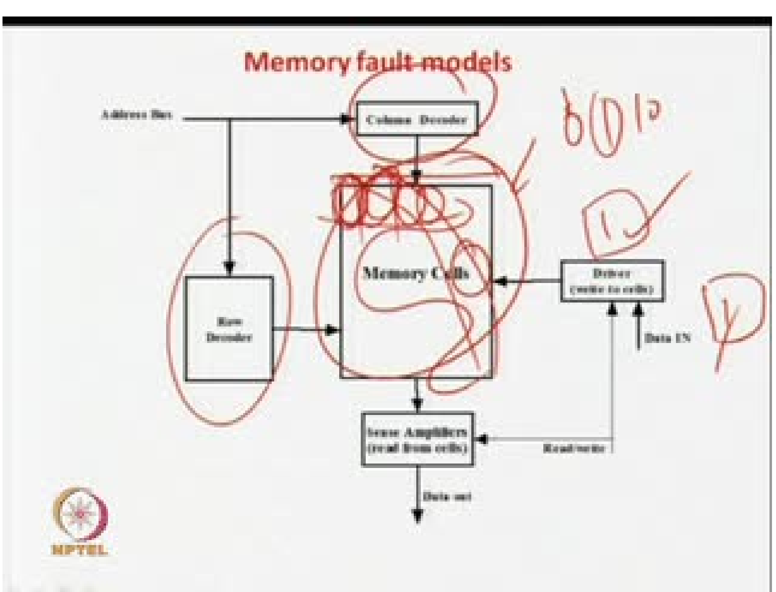
So, this is what our basic architecture. So, what we have seen so, all the March test basic algorithm coupling faults neighborhood pattern faults or the idempotent faults whatever are mainly to get the memory cells. And this is the row decoder and this is the column decoder.

(Refer Slide Time: 24:55)



So, these two we have to test already we have said that.

(Refer Slide Time: 25:20)



These 2 are nothing but some kind of combinational circuits. So, we can go for stuck at 0 and stuck at 1 and d algorithms and all the parts. But, again we have to know that mean

we are not going to do that because these are not frequent. Because, what is the job of row and column decoder are very much fixed, that is whatever value you give, they access the corresponding memory cells.

So, here what is the basic rule we use that say for row and column decoder, you select this and then this, then this and then this and you go in the sequential circuit and such kind of a sequential manner. And you read back 0 and write 1 and you read back 0 write 1 all these things you do. So, I mean if these 2 things are done properly, and at no point in time, you will get any defect or any kind or errorless responses from the memory cells. Something like this 0 write a 0 and 1 write a 0 and a 1.

In such a particular sequence you write, in the particular cell order and again you read back, right? Then if your memory and you know that 0 in 0, 0 in first cell of this 1 and say you write in second cell and so forth. So, you know that where you have written 1, again you write everything now and again you start reading back. So, if this row decoder and column decoder operating fine.

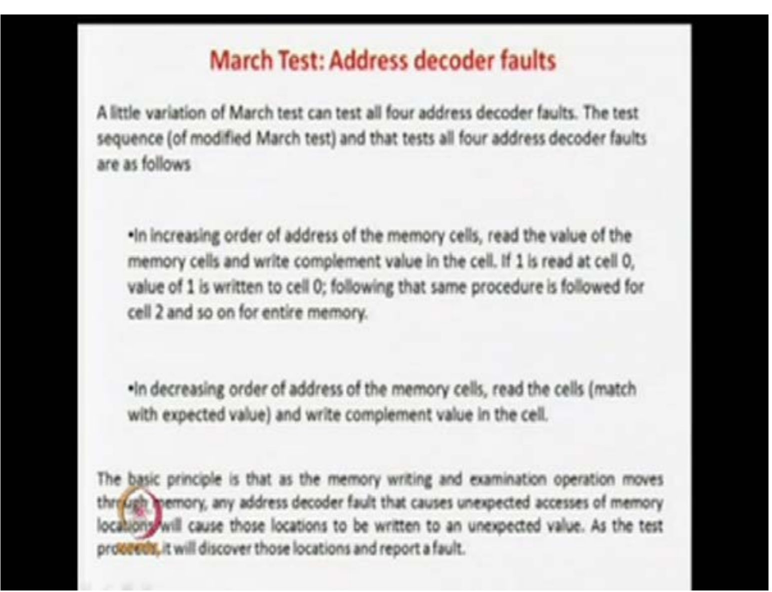
So, whenever you want to access this that file whenever you want to access this it will access this and whenever it will access this never it will happen this if you want to access this and access this. So, there can be some aliasing. Generally it is found out that, if you are traversing in 1 order and then you come back in some other order that is 1. So, you are getting every data correct, then you can be very much assured that there is no what you can call row decoder for this means, this is a very good high chances are there. Because, it is obvious that there can be a March for example, this is the first cell you have written 1 and in this cell also, you have 1, we are accessing. You want to access this instead you are accessing to this 1.

So, there is some probability is there, but you may miss it, but while writing a different type of patterns and reading it back and writing it all the ways. So, the probability can be made very high. So, you are not discussing that how the probability is made high, that you can find out that references given in the course. So, what you can do is that what they have done is that there are different sequence of patterns, and you read back them in a sequential manner read them back.

And then you get all the date correct not only the memory blocks are correct, but also, assume that your row and decoder blocks accessing the corresponding the cells on the

memory you requires are all fine. Similarly, that these drivers and do not write separately idea is that you write read something from that. So, I mean you read every value every BIST correctly. Then the idea is that cells are working properly. So, here you are writing something and reading it back. So, if you are able to write and read back that means what? Your driver cells are working properly. That is the basic philosophy we will use whether we are not going to use some analog techniques to do this job. So, by this philosophy we will see how memory of this March test or different forms of March test can be decode.

(Refer Slide Time: 27:46)

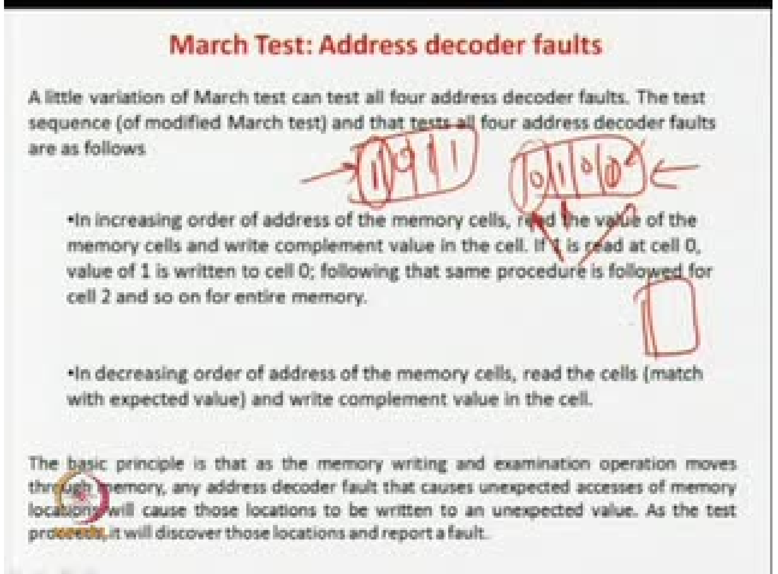


So, you will see, a little variation of March test can test all 4 address decoders forms which we have seen. What were the 4 types like types you should not get access to a wrong memory cell. So, I mean you should not get, it should not happen that 1 memory address should not access 2 memory cells. It should not happen that 1 address is accessing nothing and should also, not happen that you may get other kind of coupling effect, all these 4 models we have seen.

In fact the idea was that whatever value you want to get, that value should come out. It should not happen that you want to access the value of x and you are getting the value of y. Similarly, it should not happen that you are accessing x and you get the value of x and y both. All these things should not happen. So, I mean the idea is that if you can access something properly, read and write back properly, whole memory cell it is very, very

high probability that row and address decoders and read and write drivers cells are working fine. That is the basic idea.

(Refer Slide Time: 28:40)



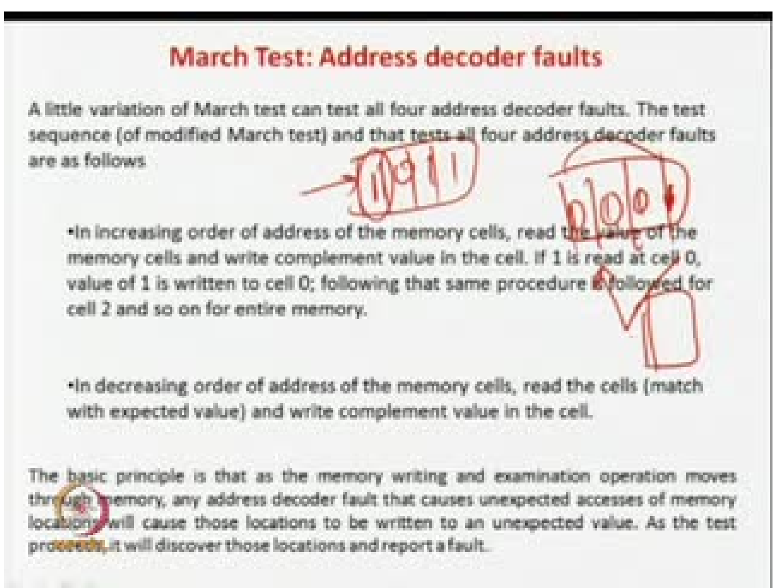
So, what they do? In increasing order of the address of the memory cells, you read the value of the memory cell and write the complement of the cells. If a 1 is read, write as 0, if a 0 following that the same procedure of it. If a value 1 is read at cell 0 and values of 1 is written into cell 0 and just write a reverse of it and it is done.

In decreasing order of the memory cells read the expected value of cells and write the complement value in the cell, right. The example is a very simple idea like for example if you have 0, 0, 0 write, you read 0 write 1. So, let us not take 0, 0, 0. Let us take the memory cell value 1, 0, 1, 1 something like this these are the 4 memory cells you assure. Then what you do? You first read 1, write a 0. Then 0 you write a 1 and 1, 0, 0 again you read back 0, 0, 1, 0.

So here the idea is if you can do that successfully, then the idea is that you are accessing the proper cell, right? If 1 was there you inverted it and so that after you access this proper cell you have to get the value of 0 it. So, by chance if there is a problem that I mean if this cell was there and by chance you want to accessing the last cell, but by chance you are accessing this cell. So, you will get the value of 1 and you can find the way of address decoders. But, always you know the effect of aliasing effect also, say for example, if you want to accessing this cell if you come and access this cell then what can

happen is that you can find the 0. So, it may be a proper thing. So, it may find address decoders in that, but in fact instead of accessing this you are accessing this one. But, now again it can be very easily changed, where it is checked why you are I mean so, these are the problems between this cults and the cell.

(Refer Slide Time: 30:01)



So, you can just take the 2 as I told you different values, now in this case 1, 1, 1, 0 something like that. So, different patterns you are checking it will become 0, 0, 0, 1. If you read this 1 and you are getting the answer of this 1, then you can find out the actual decoder fault over here. So, that is what I am saying the idea is that you do this compliment and try 1 or 2 different patterns, so what you are going to find out that by using proper number of patterns in proper sequence you can find out that even the aliasing affect of address decoder fault testing will be minimized.

(Refer Slide Time: 30:40)

March Test: Address decoder faults

A little variation of March test can test all four address decoder faults. The test sequence (of modified March test) and that tests all four address decoder faults are as follows

- In increasing order of address of the memory cells, read the value of the memory cells and write complement value in the cell. If 1 is read at cell 0, value of 1 is written to cell 0; following that same procedure is followed for cell 2 and so on for entire memory.

- In decreasing order of address of the memory cells, read the cells (match with expected value) and write complement value in the cell.

The basic principle is that as the memory writing and examination operation moves through memory, any address decoder fault that causes unexpected accesses of memory locations will cause those locations to be written to an unexpected value. As the test proceeds, it will discover those locations and report a fault.

So, I mean the basic principle is that the memory writing and examination operations moves through the memory. Any address decoders fault that expects unexpected access of memory locations will cause those locations reduces an unexpected value.

As the test processed, it will discover those fault locations and report a fault file report a bulk file. The idea is that you write something read something in different patterns and in a sequential manner. So, that I mean whatever cell you want to read if you are not able to get the value from that cell value or you are getting the value from some other cell it eventually it is wrong. That is the basic idea of address decoder fault. We are not explicitly testing whether, I mean whether exist in decoder fault or whether there is a problem in address decoder fault as a combinational circuit or sequential circuit kind of a thing, so that is the idea.

(Refer Slide Time: 31:20)

Basics of memory BIST

- For March test an address generator (increasing and decreasing order) and a data reader cum writer is required. So BIST for March test will be simply an LFSR and a data reader cum writer. As in the case of logic BIST, the LFSR should have primitive polynomial (so that it generates all numbers from 1 to 2^n), and along with this the LFSR for memory BIST should the following features
  - Be able to generate all the 0 pattern to access the last memory location
  - Be able to generate forward and reverse order patterns i.e., if 1-0-2-3 be the sequence of the LFSR (when initialized with 1) then there should be a mode to generate the sequence 1-3-2-0.

Now as I told you now we come to another paradigm that is called basics of memory BIST that is memory BIST. Now why do you require memory BIST? So, what is the idea of BIST? As we already discussed so the basic idea of BIST was that if there is a chip, which is in the market which was having no faults you tested using an (( )) everything was done properly now we are shifted it to the market. Now, in the market the chips are in heating effect micron effects and lot of heating effects and all the affects are there. So, now the chip may have a problem when the circuit is when it is operating online and the idea is that I mean that is in your system.

Now, what you do with that? So, one thing is that find out if there is the problems again you have to debug it, find out where is the problem and send the chip, I mean send it to the customer or company at your work station wherever the test facility is there. There is a long procedure which is causing a lot of problems in your life.

So, in this case what you have seen that you put a built in self test circuit that will at every time your circuit starts, as it will have hardware pattern generated inside, there is a LSFR which will generate some patterns. It will find out if there is a fault, if there is any signature compressions, it will find out the faults if there is a fault it will find out ok this chip is having problem. So, that report you can easily you can find out a very handled and a very small tester. So, whenever you start up the system like your mobile phone or

your laptop so, it will say that this chip is having a problem that because that chip has reported a BIST error and circuit system stops working.

So, immediately you know that ok this chip is not operating and you will take system to the local vendor and he will replace the chip and your system starts working. That is we have already discussed in elaborately discussion on BIST. So, now as I told you that combinational circuits and sequential circuits are as such that the yield is like ok kind of a thing like 60 to 70 percentages. So, more or less the chips are fine in case of memory, you have found out that every time every chip you sell more or less there can be less problems in the memory cells because, they are very compact and very near to each other lot of faults can be possible.

So, now what you do? That is a big problem. So, now what you have to do? So, what is the basic idea? How can you solve the problem? To solve the problem, what you have to do is that we should have a BIST there because memory is more complex than combinational or sequential circuits. So, idea is that we have sequential circuits are more complex and error, where faults are probable or there can be lots of issues.

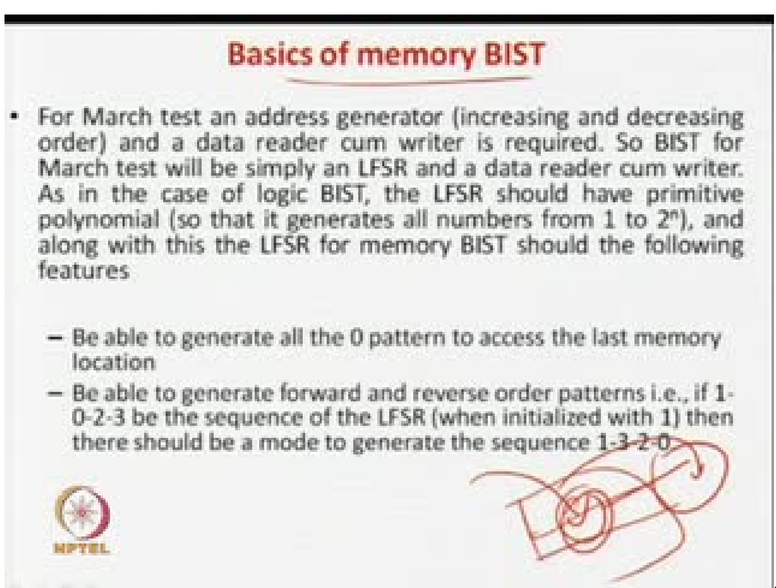
So, BIST is even more mandatory over here or more required to have BIST in memory cells. So, all memory chips have BIST. Now in case of combinational sequential circuit if the BIST circuits reports an error, what we will do? We simply say that there is a problem over here. So, just you flag out an error saying that there is the problem and you have to replace the chip. But in case of memory it is not that. So, what you can do? If you have a very advanced what you can call reallocation, or changing of the faulty chips procedure on circuit, just like I will tell you an example.

So, for example if you have suffocated 1 GB of memory, it has already some redundant cells as I already told you some chips some rows may be faulty so, what you have to do? You have to sometimes bypass these faulty cells and so, that you can get an access I mean faulty cells can be bypassed. So, that instead of fault you can access some redundant or extra normal cells. So, that bypassing arrangement is done by multiplexer by blowing out some circuit switches. But now on chip on the memory of operating system normal cells can go back BIST will capture that.

So, that is what very much mandatory or very much required to have BIST in a memory testing circuit or in a memory chip. Now BIST will says that there is a problem over here

so, you have to do something. Now what you can do? Now you cannot simply throw off the chips if the memory chip is lost so, you throw up this cannot be done because in every cell we got to in every ten weeks, or I mean we should not call weeks for every 6 months or 7 months some frequency will have, some cells of memory chips getting damaged and that may easily happen because of the architecture of the memory. Now the BIST will have to detect as wells as to diagnose that. So, that is not only enough. Say, for example, if you have a memory like this.

(Refer Slide Time: 35:00)



And your this part is gone bad and that you have found out in BIST. So, what you can do? You can actually you cannot you should not be able to access this cells that is the idea. So, now what you have to do? So whenever you have a program is accessing this cell you should actually bypass that to another part of the memory. That is that bypass routine or bypass mechanism should also be present. That can be done on seem to by blowing of some switches you can do bypass, there is also, some mechanism which are possible, we are not discussed because which are advanced memory repair technique and then there even if the BIST detect an error. So, all these cells are not operating fine.

(Refer Slide Time: 35:35).

Basics of memory BIST

- For March test an address generator (increasing and decreasing order) and a data reader cum writer is required. So BIST for March test will be simply an LFSR and a data reader cum writer. As in the case of logic BIST, the LFSR should have primitive polynomial (so that it generates all numbers from 1 to 2^n), and along with this the LFSR for memory BIST should the following features
  - Be able to generate all the 0 pattern to access the last memory location
  - Be able to generate forward and reverse order patterns i.e., if 1-0-2-3 be the sequence of the LFSR (when initialized with 1) then there should be a mode to generate the sequence 1-3-2-0.

So, what you do? You take this one from here and put the radar in the redundant cell and connect it back. So these are just bypass mechanism. So, physically these cells are here, but logically they will be connected to some other parts of the cell and internally there should be a mapping of the memory locations from here to the redundant cells. So that is the very sophisticated idea, but this things has to be done and as say you are using a memory chip for say 1 year or 2 years. So, after sometimes you will find out that the extra cell are exhausted. Now then what will happen? Then you will not get a 1 GB full memory, even if you have purchased the RAM chip of one GB. So, after a long a time of use you may find out effective memory may be some 900 MB or something like that. You will not get a GB of memory.

(Refer Slide Time: 36:18)

Basics of memory BIST

- For March test an address generator (increasing and decreasing order) and a data reader cum writer is required. So BIST for March test will be simply an LFSR and a data reader cum writer. As in the case of logic BIST, the LFSR should have primitive polynomial (so that it generates all numbers from 1 to 2^n), and along with this the LFSR for memory BIST should the following features
  - Be able to generate all the 0 pattern to access the last memory location
  - Be able to generate forward and reverse order patterns i.e., if 1-0-2-3 be the sequence of the LFSR (when initialized with 1) then there should be a mode to generate the sequence 1-3-2-0.

Now why that is done? Because extra cells are over so, after the cells have been exhausted, even if there is a fault over here so, what you can do? Even if there is a something like that even if there is a fault over here. So, you have to just bypass this and you have to do this here. So, there is no guy over here to replace this and here extra cells are also, exhausted. So, only thing is that you have to assure that you cannot access these cells by any means. Because if you read or write something like that we are not able to refract back the data. So, if there is some it has access to that it has to be bypassed again from here to here. But there is no more extra cell.

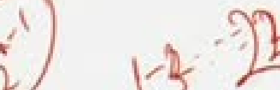
So, I mean no more redundant cells and you cannot do the repairing. So, that is why if you are using a ram cell for a long time in your computer or my computer in CD properties of your RAM main memory you find out that if you purchase a 4 GB memory of RAM 2 GB of memory in your machine, but it is much lesser than this. But still your chip is working.

That means, some part of the memory have gone bad, but internally memory BISTs and repairing architecture has been flexed. So, that your programs are not accessing those problems, not accessing those stuffs. So, that is what has happened. We will not be going into this elaborate discussion on how memory is repaired and all those things. So, rather what we will do, we will see how memory BIST can be done. We will see the BIST architecture for memory.

(Refer Slide Time: 37:26)

Basics of memory BIST

- For March test an address generator (increasing and decreasing order) and a data reader cum writer is required. So BIST for March test will be simply an LFSR and a data reader cum writer. As in the case of logic BIST, the LFSR should have primitive polynomial (so that it generates all numbers from 1 to 2^n), and along with this the LFSR for memory BIST should the following features
  - Be able to generate all the 0 pattern to access the last memory location
  - Be able to generate forward and reverse order patterns i.e., if 1-0-2-3 be the sequence of the LFSR (when initialized with 1) then there should be a mode to generate the sequence 1-3-2-0.



That is how we can detect or diagnose the faults and in which cells there is a problem and we are going to look into. Ok? Further we have how the BISTs will look like. Ok so again like, as you already seen that 2 types of LSFR modular, LSFR standard and LSFR same thing will be also, be used from this one and we have also, seen that memory are primitive polynomial. So, you can generates sequence from 1 to dot dot dot 1, 3 something it can be arbitrary fashion. And finally, you will get the answer 2 to the power of n minus 1. So, you can start from 1, 2.

(Refer Slide Time: 38:16)

Basics of memory BIST

- For March test an address generator (increasing and decreasing order) and a data reader cum writer is required. So BIST for March test will be simply an LFSR and a data reader cum writer. As in the case of logic BIST, the LFSR should have primitive polynomial (so that it generates all numbers from 1 to 2^n), and along with this the LFSR for memory BIST should the following features
  - Be able to generate all the 0 pattern to access the last memory location
  - Be able to generate forward and reverse order patterns i.e., if 1-0-2-3 be the sequence of the LFSR (when initialized with 1) then there should be a mode to generate the sequence 1-3-2-0.

1, 6, 7, 9, dot dot dot 2 to the power of n minus 1. So, that sequence of patterns can be generated, but the sequence may not be in a random fashion depends on the (( )). One important thing to philophisical again to philophical differences of the BIST from the normal and sequential and combinational circuits compared to memory. So, in case of sequential circuits if you remember on combination of circuits, we cannot generate all 0 sequential patterns in a standard manner, that was not required also. If that I mean that very, very less required all 0 patterns are not applied and there will be a very huge lot of cells that will not be applied there are that is not the case in 1, or 2 cells can be lost it is not a very general case.

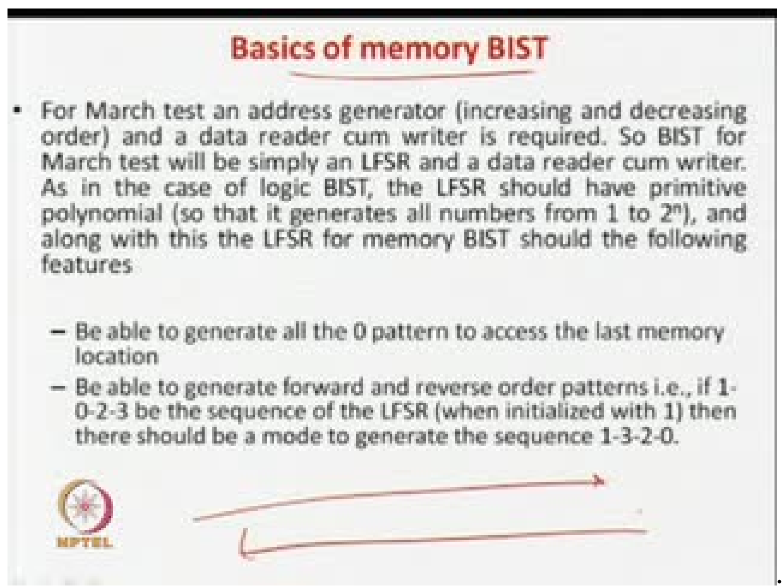
(Refer Slide Time: 38:44)

Basics of memory BIST

- For March test an address generator (increasing and decreasing order) and a data reader cum writer is required. So BIST for March test will be simply an LFSR and a data reader cum writer. As in the case of logic BIST, the LFSR should have primitive polynomial (so that it generates all numbers from 1 to 2^n), and along with this the LFSR for memory BIST should the following features
  - Be able to generate all the 0 pattern to access the last memory location
  - Be able to generate forward and reverse order patterns i.e., if 1-0-2-3 be the sequence of the LFSR (when initialized with 1) then there should be a mode to generate the sequence 1-3-2-0.

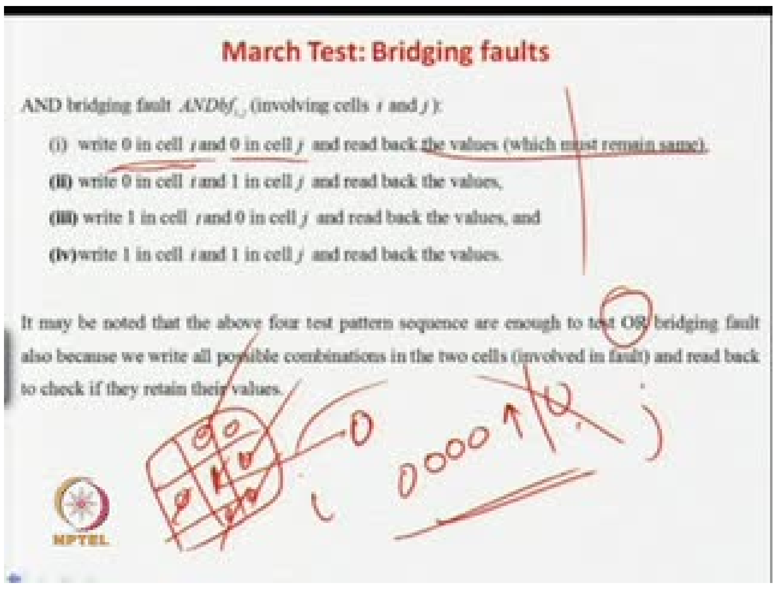
So, in that way that was not causing a very deep problem for us, but in case of memory if you remember that all 0 is a very, very important pattern because the first memory location is 0 and from there you go to 2 to power of n minus 1 and whatever. So, first location is 0 so, all 0 location has to be handled. That is a very, very important pattern so, second this thing is that if you remember that March test or some test.

(Refer Slide Time: 39:00)



So, actually what we do? We go in some order and we come back in same way of order. That is a basic I mean basic fault test module if you remember you have seen is that a stuck at fault transaction fault similarly, coupling faults ok? All the coupling faults are there.

(Refer Slide Time: 39:25)



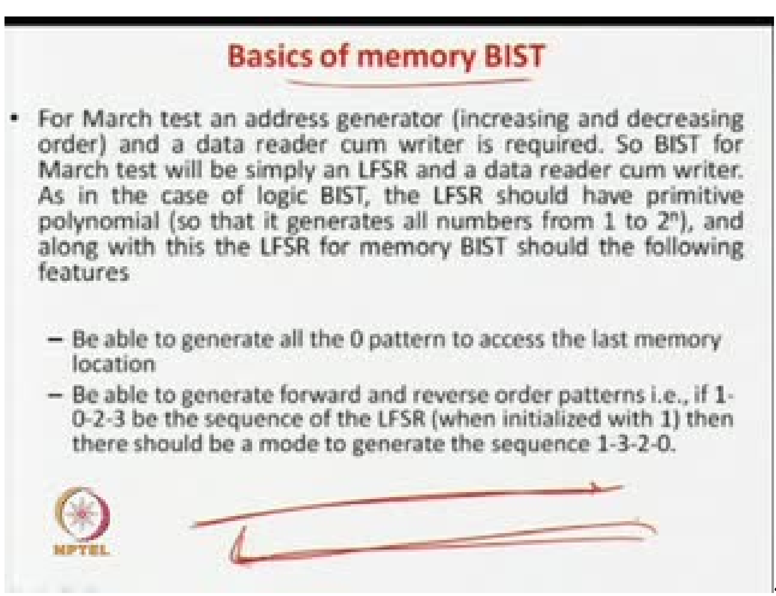
So, in a very similar way we have not discussed here, but you can find out that this neighborhood pattern sensity test faults can also, be easily tested by a March test kind of a thing, so, they are very simple like if you are taking this kind of thing. So, if you have

0, 0, 0, 0, 0, 0, 0. So, if you know that the fault is such like something like this. So, all faults are there 0s are near about that you may not consider this. So, any fault mode whenever any fault model this type of neighborhood if all cells have 0 and then from 1 to 0 you cannot have a kind of a fault like 0, 0, 0, 0 and then this thing you are going to do so always stuck 0 kind of thing here.

Then that you want to do that it is very simple you just put write 0 read 0, write 0 read 0, and write 0 read 0. These are all the nearby cells whichever the cells required to write a 1 over here, make 0 and read back a 0 is possible. Just like all the coupling faults neighboring patterns faults test can also, be tested like in a very simple manner by using March test. Only thing is that again this memory location may not be in sequential manner.

So, you have to access this, access this, access this, access this, access this. So, that the idea is that accepting the stuck at faults and what you call this transition fault for all other coupling faults we enable pattern sensitive faults, bridging faults, address decoder faults, may be all faults you required to have a March test, but the order of access of the memory cells will be not be sequential. So, ok, but for transition faults and stuck at faults what is the idea? You go in this way and you go in this way.

(Refer Slide Time: 40:44)



So, a very, very basic requirement of memory testing is that very basic requirement we say because, we share lot of time constraints and flying constraints. So, if somebody says

that ok I have to do only this very preliminary test that is stuck at fault and transition fault because, they are very simple. You apply this order, you apply this order nothing you have to remember. You have to keep on going from 0, 1, 2, 3, 4, 5, 6, 7, there 7, 6, 5, 4, 3, 2, 1, 1 that is it and you have to write 0 read 1, write 0 read 1 and your job is done.

But if you are going for coupling fault and neighboring pattern sensitive faults, then the order of access of the cells are also very, very important. So, if you remember the order of the access of cell that is some pattern of the cells you have to remember. Already you have seen that remembering pattern is very, very difficult in case of BIST. Because if you have to remember pattern you have to go by the designed in a standard digital design way and there is to be a lot stare in the idempotent machine, so that, the area will be huge.

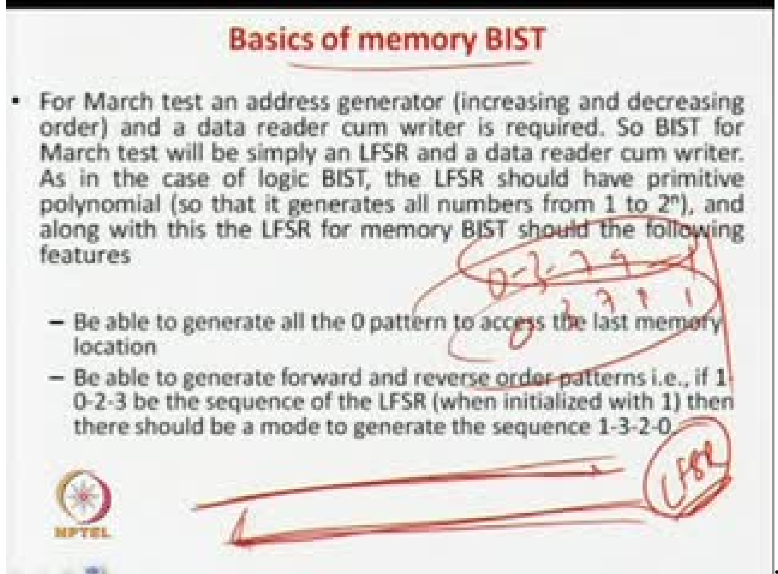
So, if you have to want to go in the arbitrary pattern 1, 2, 3, 4, 5, 6 and so on that is also, very huge amount of data, but you have to go less than in a predefined manner. Predefined manner makes the area to be bit higher, but what is the easiest? So, if you can go 1, 7, 9, 6, 3, 4, 2, 1 you can again come back in a random way. So, if you can traverse the cells in some random nature, then the area over it will have to be very, very less. So, that is the thing you will be using. Like that again let me tell the philosophy again so, if you want to access the memory cells the best idea is to test all the memory. So, you have to go for very sequentially if the 2 cells having a coupling faults, then those two cells should be access one after the another writing 0, reading 1 and so forth.

If you want to patter sensitive faults for neighborhood cells then you have to go for 1, 2, 4, 1, 0, 1, 3, 4 something like that. So, this particular order of cells access redirecting 0 or 1 in all those things that is in other word you have to access the cells in the predefined manner. That is going to be a big problem because, we have already seen that if we want to apply patterns from any standard patterns generator, which will actually in this case generate the address of cells predefined manner is very difficult because the area over all next it is difficult to do all kind of test so, we find BISTs in a memory because of the area over (( )). Now if you want to say next level is what?

Next level is actually stuck at faults and transition faults minimum. So further what you have to go is 0, 1, 2, 3, 4, 5, 6,7,8,9 and say come back. These is also possible and again as you are going from 0, 1, 2, 3, 4, but still there is some sequence because of ascending

order and descending order remember something. In some ordering that will also, kill you the area of the BISTs, that is going to be a very killing factor.

(Refer Slide Time: 43:10)



So, what the idea they are going to do is that we can access the cells in a very random manner like we can go for 0, 3, 7, 9 dot dot dot all patterns will cover one and something like this and again we will follow back 1, 9, 7, 3 and 0 they are very random in nature. So, you can say that what I will do that random user normal LSFR because, LSFR can generate the values of any random pattern depending the value of the nature.

(Refer Slide Time: 43:43)

Basics of memory BIST

- March test can be modified by replacing "sequential read/write" with "arbitrary order read/write, but covering all cells" without loss in test capability.
- We illustrated cell traversal from 0 to 9 and then from 9 to 0.
- However, the test capability will not change if sequence of cell traversal is any other sequence, for example, 1-0-2-5-7-3-4-6-9-8 while moving in ascending order and 1-8-9-6-4-3-7-5-2-0 in reverse order.

So, that is why when you are going for random BISTs so memory BISTs what we are going to use? We are going to use March test and basically test on the called what you call this stuck at fault and transition fault also, by advanced studies advance studies have shown that if you go for transition fault as well as for stuck at fault some of the bridging faults, like I mean you may not have all the sequences 0, 0, 0, 1, 1, 0 and 1, 1 ,0 ,0 ,0, 1 all kind of that patterns are possible.

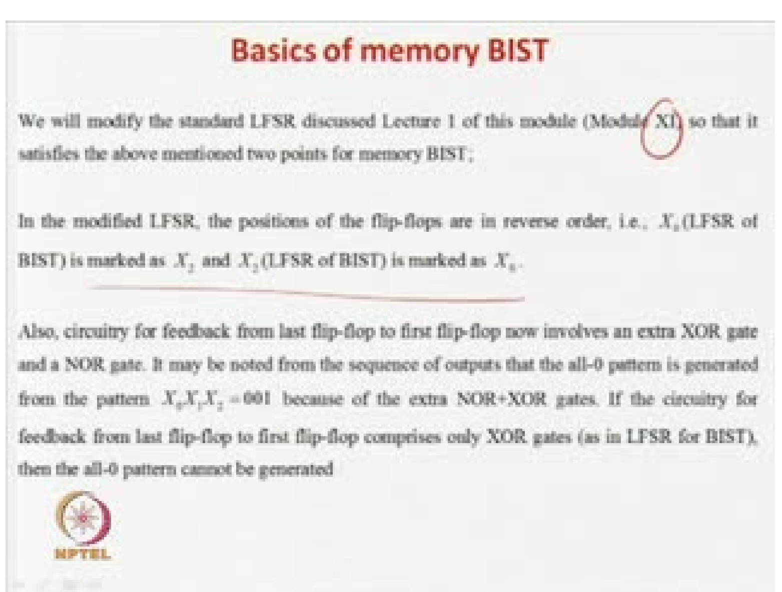
(Refer Slide Time: 44:25)

But at the 0, 0 and 1, 1 patterns will be possible something like this. Some of the patterns may be possible. Similarly, for the coupling faults for the neighborhood pattern sensitive faults not all the patterns are may be sensitized, but few of them will be done. So, you can get a pretty good feeling that whether your memory is operating fine or not. So, what we have done? The basic philosophy is that of we will not go in a sequential manner. We will go in a arbitrary order without loss in test capability for this stuck at fault and transitive faults.

Now you apply 0 first and first location, second location, fifth location and 7 like this. 0, 1, 2, 5, 7 this one and descending order 1, 8, 9 this one. So, we are just doing in the reverse order, these are very random in nature. We will be depending on your LFSR design and your feedback. So, if you are d this already just doing in a reverse order very random in nature do this you already done. So, that is what the philosophy be using for memory BIST.

We will go for simple March test ok, and then we will cover single stuck at fault and transition fault and few fraction of some other faults which we are not discussing. And generally, if you have time you can read through the references this shows that they will also will cover some few of the other faults percentage of them, and we need not go for a very sequential manner.

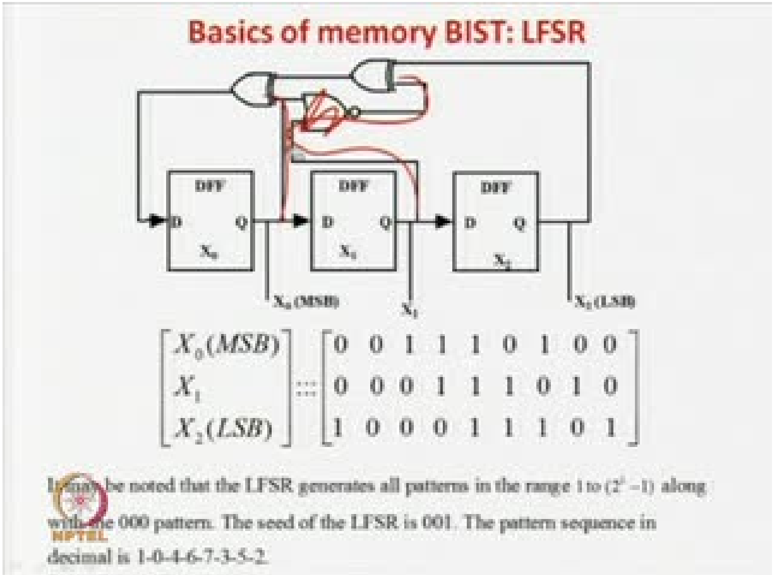
(Refer Slide Time: 45:17)



We will do 1 than 3, 7 and 9 and obviously that are any 2 compromise test capability. Because, in case of March test also, what we do is that March test for stuck at faults and for transition fault there is no coupling effect. So, each individual cell we studied in a different way like concept write 0 read 0, write 1 read 0 and so forth. For each cell we verify independently irrespective of others. So there is no need to bother 0, 1, 2, 3, 4, 5 you can go in a arbitrary order. So, that is what we are going to apply over this and we will see how it is done.

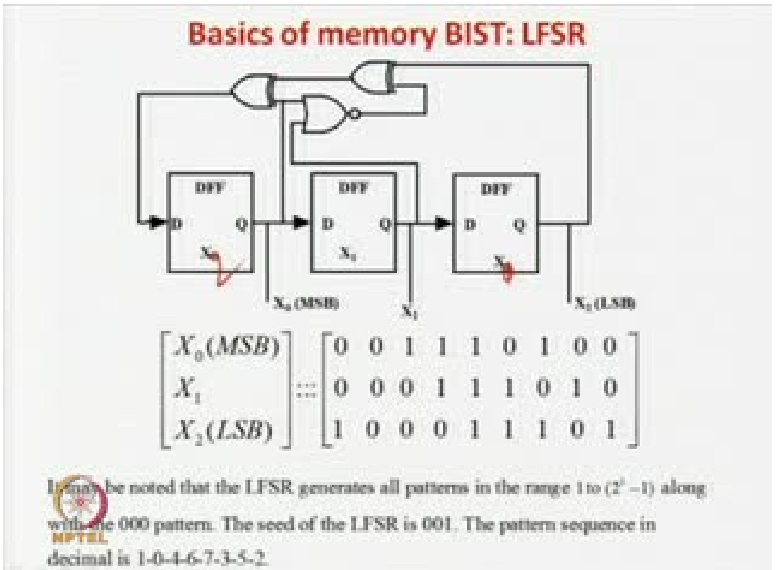
So, what is the basic difference between memory BIST architecture LSFTR and we studied and which we discussed in. So, this is the standard architecture so, which we have module 11 say last module if you remember we have also discussed the standard architecture BIST for standard sequential and combinational circuits and we are also, modifying that one only for memory BIST.

(Refer Slide Time: 46:30).



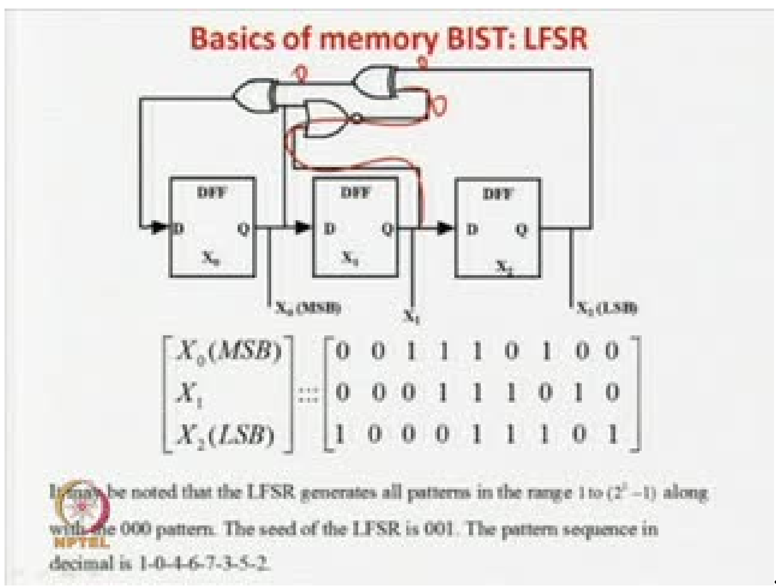
See for what the modification is required for this one. Ok? So, here actually if you look at the architecture so, here actually we have reversed so we are using the standard LSFR modular can also, be designed. But, we are not looking into that , but in this case the first thing is that we reverse this order .So, if you remember that here we used to have x 0 x 1 and x 2 and but now we have reverse the order .So, what you have done ? We have reversed the order this is the first thing and in the feedback, we have if you remember the other case so, we may have a feedback from here then we may have a feedback from here and we may also have a feedback from here also, no nand gate or something like this.

(Refer Slide Time: 46:46)



But here we have introduced 1 nand gate over here also. Ok. So, in this case there are 2 differences 1 thing is this was x 0, x 0, x 0 here x 1 and x 2 over in the standard case of sequential and combinational circuits, there is normal LFSR or normal circuits and for memory this is reversed and we have put a extra nand gate. So, now why you extra nand gate you will find out?

(Refer Slide Time: 47:15)

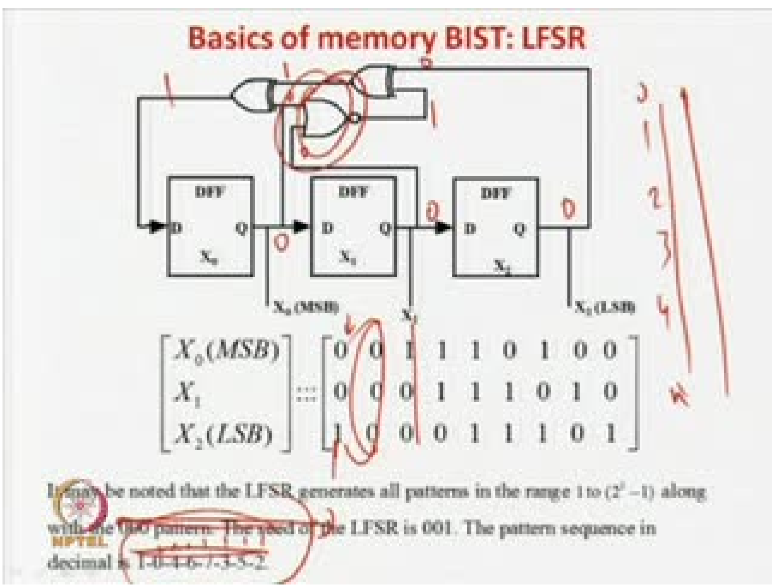


.

Sorry NOR gate this is NOR gate why? Because you have to get the all 0 pattern, that is very, very important. So, there are two things. Because of this extra x NOR gate and

nand gate combinations so, we will get the all 0 pattern which was not possible in case of this standard LSFR or modular LSFR or normal general circuits like if you remember that this is the feedback was like this so, if you have a 0 over here, if you have a 0 over here, if you have a 0 over here, that is the output of all the switches are 0 so x add 0 so it will stuck at fault at all 0s we will have already seen that. To avoid that we have use this one.

(Refer Slide Time: 47:33)



Let us see how this is done. Ok .So, let us have this is like that x 0 is equal to 0, this is equal to 0 and this equal to 1. So, let this be the current seat ok? And in case of a standard normal LFSR for normal kind of a such thing, we remember that we may have to go here and stop because this all are the may be patterns which will enough to do all the testing for you.

So, we actually have to select the seat in such a fashion so that ,that means all the important patterns of testing get apply in the very first few iterations and this we can avoid from here to BIST back. But in case of memory, it is not there. We have to apply all the patterns in the row that is from all the patterns because, all the memory locations have to be accessed. So, all the patterns are equated. So, we start with the 0, 0, 0 we get the standard over here. You start from location 1 and then you will go back to location 0. Because if you use 0 as the feed, that also you can do. But generally idea is that, we first use all the 1, 0, 1 seed, then from there we go to 0s and again all will be progressed in

this manner and all the locations will be covered in this way. So, this is the pretty standard seed case in memory BISTs and in case of a normal circuit's kind of a staff sequential and normal combinational circuits.

They are the type of circuits we use the patters such that the important patterns will be applied first. Now it is done let us see we have applied the pattern 0, 0, 0, 1. So, now the next phase you get a 1 over here right this is a 0 over here 0 over here so nor 0 means it is a 1 over here 1 and 1 will actually make a 0 over here right it will be a 0 over here in a 0 over here will actually convert a 0 over here right? So, now what is going to happen here this 0 will be shifted over here and this 0 will be shifted over here. So, you will get the all 0 values now you see all 0 patterns will get stuck at in case of the LFSR of normal kind of the circuit how it is not done here.

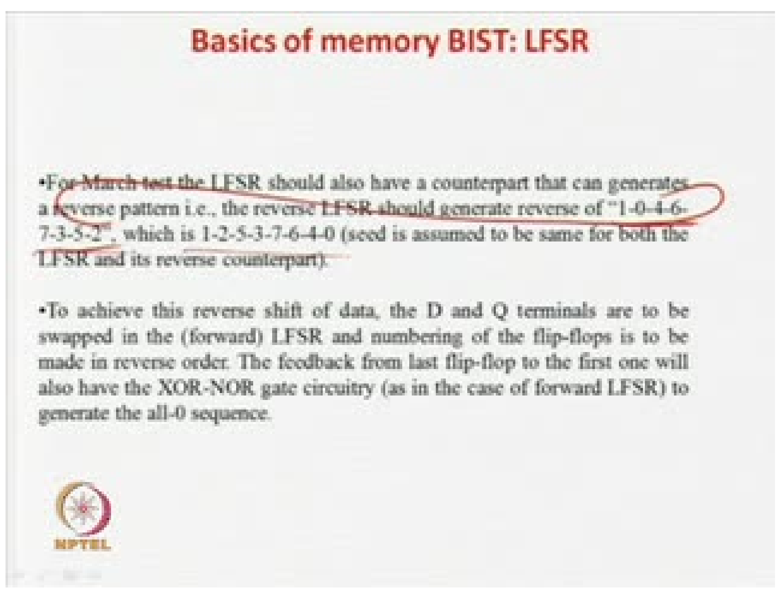
So, if you just see that all 0 patterns over here so, now what is then you get a 0 over here, now this is the 0 over here remember 0 and 0 NOR gate 1, 1, 0 you get the value of 1,1,0 and so again, you get the values. So, now because of this gate arrangement which is not stuck at 1 thought it is not stuck at 0 or o patterns is not there. So you get this pattern 0, 1, 0, 4, 6 so, it covers all the 1, 2, 3, 4, 5, 6, 7, 8 so, 3 bits, all the 8 memory locations can be covered using this.

So, just by using a simple NOR gate, your job is done. And these are very, very standard seat you start from 1 and do this. Now again March test is there what you have seen that we apply some pattern in this order or you write 0 read 1, write 0 read back and so forth. So, you keep on applying this here. So, now what we have to done? You have to again traverse this, this, this, this, this, this and again read back and again write. So, what is very, very important in March test is that you write it in some random order not a problem, but when you traverse back and also again you have to get the same order in a reveres manner. That is very, very important. We write like 0, 1, 2, 3, 4, 5 up to say 10 now we go this way and come back this way and this problem.

Now, because of this sequential problem the area may be higher so, we are allowed to go arbitrary like 0, 4, 5, 2 something like this, but what you cannot forgo is that once you a have traversed this you have to again come back. So, while coming back you cannot traverse in a arbitrary manner like 3, 7, 5, 9 like this because, then you cannot remember anything. We don't want to remember which cell we are accessing. Just want to

remember that first cell, second cell, third cell, fourth cell, fifth cell and again coming back to 4, 3, 2, 1 and 0, but now we are actually avoiding the sequentiality because to reduce of the area overhead of the LSFR.

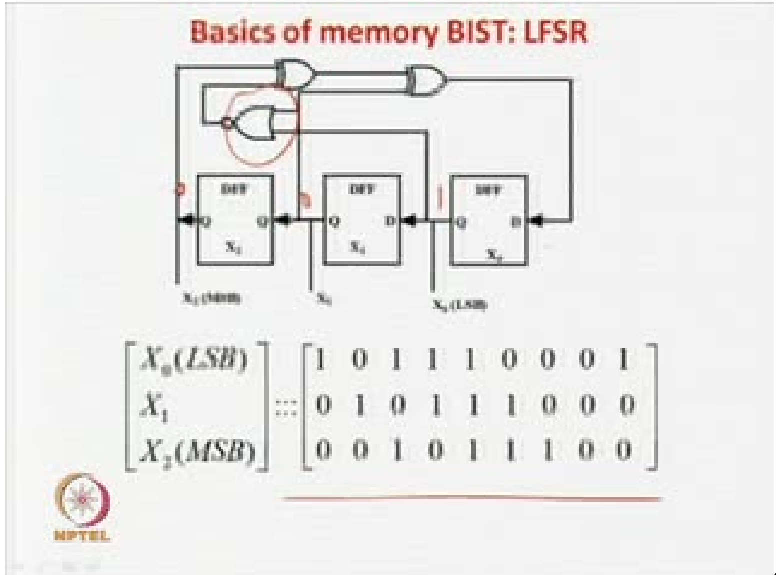
(Refer Slide Time: 51:21)



So, now what we do is that while turning back you should again take 2, 5, 3, 7 like this. It should be in strict reverse order. So that, we can know that this is the straight order and while coming back accessing the cells are in random manner, but the sequence is just reverse. So, that we don't required to exactly remember what is there.

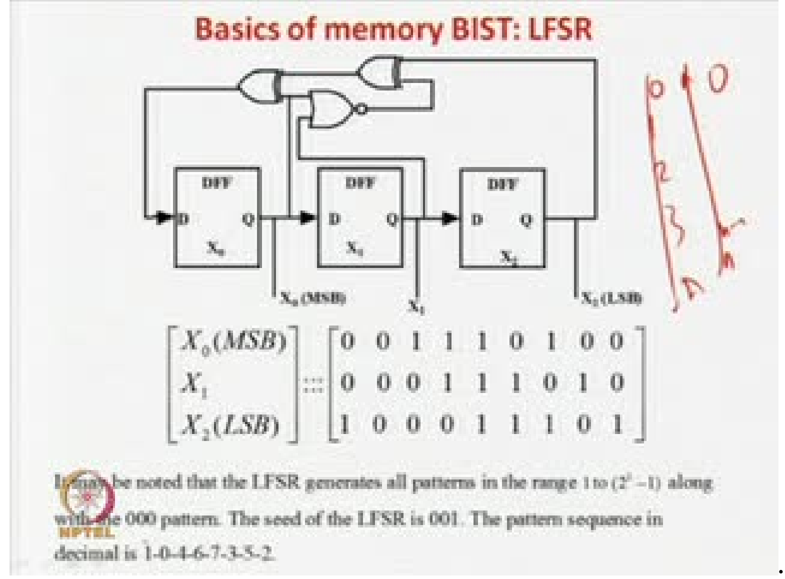
So, this is what the general case so, this is the reverse case. So, that you have to do 1 is the starting point everywhere so, 0, 4, 6, 5, 2 so just you have to reverse it 2, 5, 3, 7, 6, 4 so this is what is the idea. Because of accessing you are not accessing the sequential manner; you are accessing a random manner, but while coming back this strict frequency should be ordered. Ok then we will also, see that this is very simple to that.

(Refer Slide Time: 51:55)



So, because already we have this LSFR now we require another LSFR in combination with just a reverse of this. So, what we have done? So, in this case x 0, x 1, x 2 so, in this case we just have to reverse it mix the x 2, x 1, x 0 and the connections are the same way in this one use the NOR gate x 0 cases and you can find out that here will generate like if you have the seed is like x 0 the seed it will be like 1, 0 and if you have the seed memory piece architecture. So, what we have seen here is that in case of memory BIST, the LSFR size has to be smaller and as in case of the normal general kind of circuit.

(Refer Slide Time: 52:22)



(Refer Slide Time: 52:52)

Basics of memory BIST

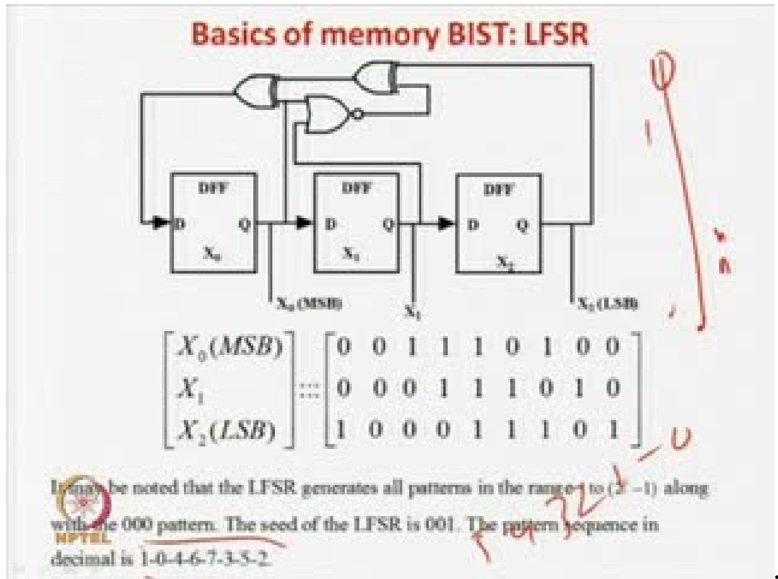
We will modify the standard LFSR discussed Lecture 1 of this module (Module XI) so that it satisfies the above mentioned two points for memory BIST;

In the modified LFSR, the positions of the flip-flops are in reverse order, i.e., X_0 (LFSR of BIST) is marked as X_2 and X_2 (LFSR of BIST) is marked as X_0.

Also, circuitry for feedback from last flip-flop to first flip-flop now involves an extra XOR gate and a NOR gate. It may be noted from the sequence of outputs that the all-0 pattern is generated from the pattern $X_0X_1X_2 = 001$ because of the extra NOR+XOR gates. If the circuitry for feedback from last flip-flop to first flip-flop comprises only XOR gates (as in LFSR for BIST), then the all-0 pattern cannot be generated

So, what we are looking is that, memory BIST you actually see in case of March test we go for 0, 1, 2, 3, 4 up to n and then again you have to come back. Ok. So, this reverse sequentility forward sequential and reverse sequentility has to be there. That is very well understood like n minus 1 dot dot 0, but now again you are going to go in a sequential manner that may be a lot of problem for area overhead for the BIST so, what we can to do is that just we have to which already discussed twist the idea a bit. So, now instead of actually going from 0, 1, 2, 3, 4 do dot. So, what you can do for the random manner you do for 2, 5, 7, 9.

(Refer Slide Time: 53:30)

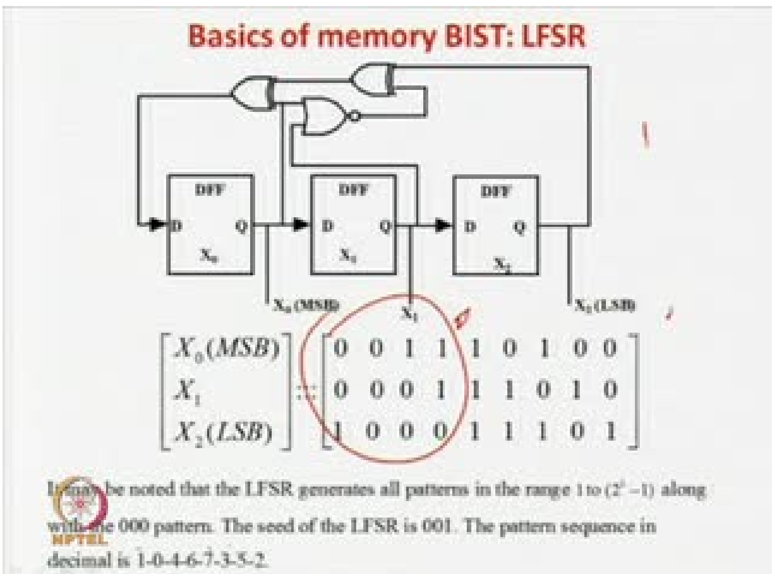


3 like this. So, the idea here is that you generate the sequence all this oblivious sequence form 0 to 2 to the power of n minus 1 has to be generated. But, now here you do it in random manner. So, that can be very easily done if you're using a BIST architecture pattern generator like LFSR which we have already discussed. In the case of general circuit and so, you can generate a pattern in a very random way depending on the seed.

So, now only there are some differences I mean because if you remember in case of this general circuits, we have to generate, we don't required all 0 patterns because very rarely if the case is happens that all 0 patterns are very much mandatory for large number of faults, this is not generally the case. So, but in case of memory BISTs you always have to accept the, 0th memory locations, so 0 pattern is very much required.

So, secondly another philosophy is that so, you are allowed to traverse say for example there are 8 memory locations. So, you are allowed to transverse in case of March testing 1, 0, 4, and 6,7,3,5 that is absolutely true. But, now again if you want to come back so you cannot come back in another arbitrarily fashion like 5, 4, 3, 2, 1, 0 not like that.

(Refer Slide Time: 54:30).



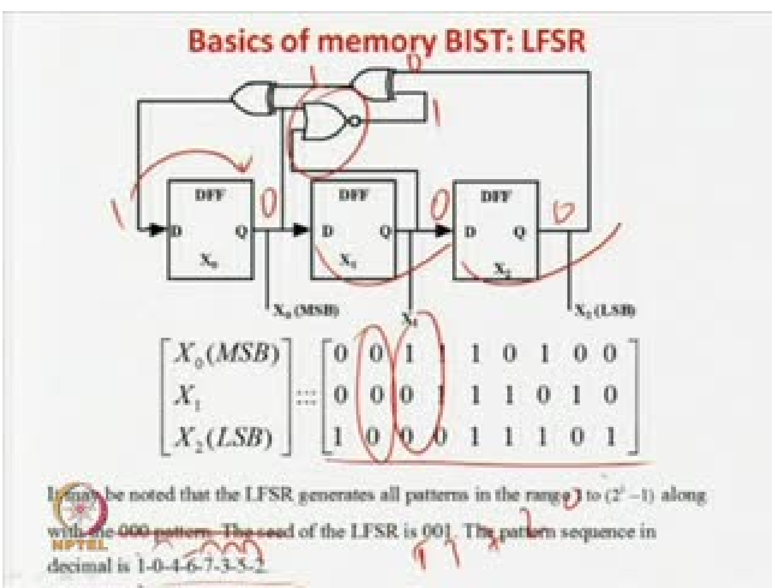
The idea is that you apply some sequence of patterns in some sequential memory cell, you don't remember exactly which memory cells you have written and don't remember exact memory location, what you do is that is the cell you write something read something and keep on doing this and while you come back you just remember that i am just traversing this cells in a reverse way in a exact order. You don't have to remember the cell location which is doing a saving a lot of complications.

So, that is what you are required in designing the memory base. That that you are generating a order in a random fashion, that is fine because, the random order generation is simply in the LSFR, but while travelling back you have to exactly travel 2, 5, 3, 7, 6,4, 0 and 1 something like that . You cannot have any of the arbitrary order of traversal,. Because you have to remember horribly complex for us. Now, we will see how the memory cell can be designed.

So, there is actually bit different somewhat different from the standard LFSR moderate LSFR for normal circuit, so what we do? If you remember in the standard design for general circuit it was x 0, x 1 and x 2. But for memory BIST we have changed the order we have made the x 2 x 1just as the reversal as well as we have put a NOR gate now why we have put a NOR gate? We will see that this is actually help you to generate all 0 pattern.

Now again let us see another important thing you remember in that BIST forward say for the general purpose (( )) sequential combinational circuit. We used to select a circuit in such manner so, that all the important patterns which are required to test a fault are get generated in the first few iterations. So, that from here you can reset your circuit and many time we have also not used the primitive polynomial many time all possible patterns are not required for this one, but in case of memory BISTs all patterns are required. So, we required a preliminary all these primitive polynomial so that all 1 to 2 patterns are required that is one important thing. These all memory location is to be accessed.

(Refer Slide Time: 56:02)



So, in case of the memory BISTs standard what you call cell is a 0, 0, 1 and very standard seat for memory BISTs. You can start with this 1 and then you will find out that you will traverse all the patterns in a random manner. If you will see that you which are the patterns 1, 0, 4, 3, 6, 5, 2, 7 and something like that. Ok so we will see that what the purpose of this NOR gate. So, if you apply 0, 0, 1 so now you see that1 over here. So, in this case 0 over here, 0 over here. So, again on 1 over here and you get the 0 as the output from 0 to 0 and 0 to 0.

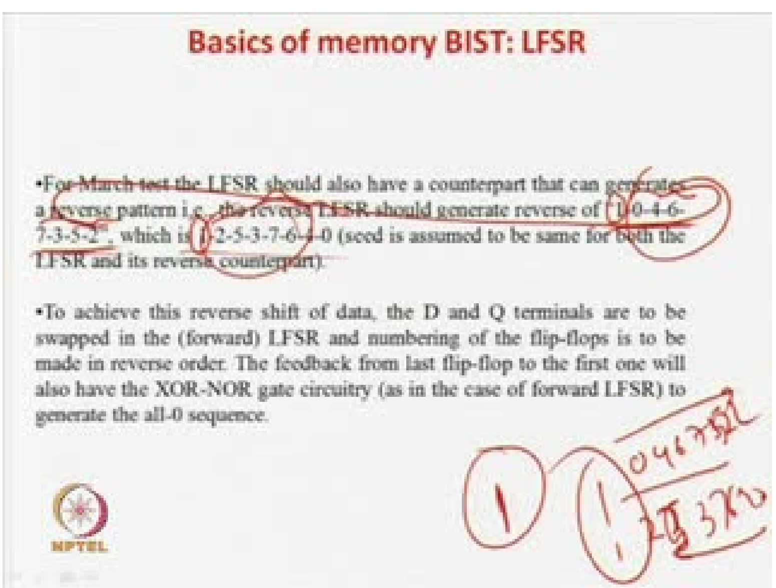
So, next pattern you keep all the 0 it is quite interesting and if you do not have a NOR gate, then it is always start with the all 0 patterns as you have see, now it is all 0s if this pattern assume this is not there, then what is the case? This will be a 0 this will be a 0

and this will be a 0 and everything will be stuck. So, that is the case in memory LSFR when you are going for general sequential circuit, but in case of this memory BISTs all 0s are required to access memory location of the memory and you have to traverse all. So, put an or gate if see that 0 and 0 or on 1 is 1 n or of 0, 0 is 1 0 and x or is a 1 so, now it is a 1 over here so this will be the case.1 and 0, 0 so 1 here 0 here 0 here so, this is the next pattern.

So, if you just design a BIST it is somewhat different form the sequential and combinational circuit so, this is one extra or NOR gate we have used and reverse order so, you can find out that you can generate the pattern in this way. Now, this is not allowed so we have generated a random pattern it will access all your memory cells in this particular order. So, the order may be very arbitrary and but this thing when the job is done.

Now another LSFR we also required because you have to traverse 2, 5,3,7,6 you cannot have another LSFR and another arbitrary seed and traverse in a order like 5, 3, 7, 2, 0, 1. So, because if you do that you have to remember that 1 is accessed 0 is accessed and when you are traversing back, which is the alternative way of access. So, that will make you a life horrible.

(Refer Slide Time: 58:21)



So, idea is that you just access this in the order and you access back in this order. Then you don't require remembering anything. You know that first you have access this and

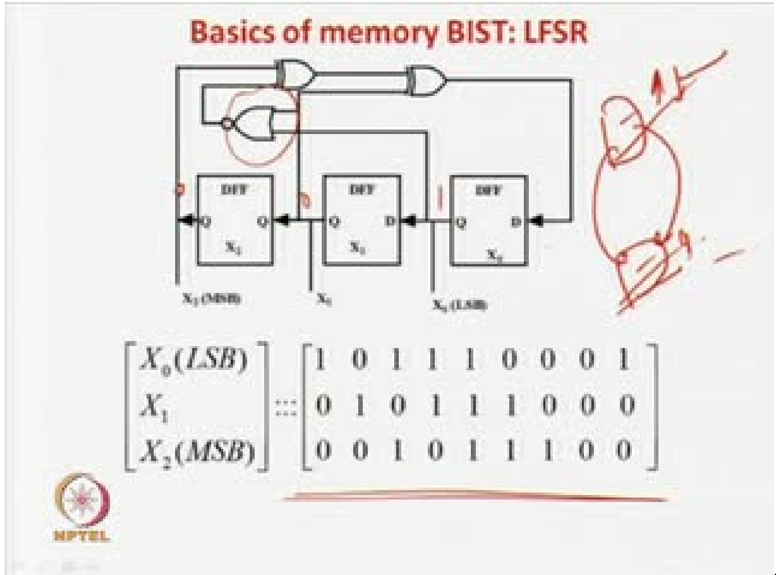
last you have accessed this that is the only thing you have to remember. So, what we do? we actually that is written in a very elaborate way in this slide that if this is the first order, then the reverse order should be exactly this one. So, what we actually do is that 1 is a pretty fine seed.

So, you always start from 1 and end in 1, but for the other parts that is 0, 4, 6, 3, 7, 2 and this part we actually reverse I mean you will get in a reveres fashion forward fashion, and reverse fashion the idea is that you do not the full sequence, I mean that reverse fashion should be actually be 2, 5,3,7,0 and after that 1 should be the case. But making that is very difficult because of the NOR gate arrangement and all if you have a star seed as 1 way you can do something very simply in that you can find out the little work you can find out.

So, basic idea is that in very initial seed is 1, we shall see this 1 and always we start from 1 and we end in 1 and the pattern is 0, 4, 6, 7, 3, 2 this is exactly reveres in the other way. So, you will get 2 ,7, 2, 3, 5 the reverse it will be 2 , 5, 3, 7, 6, 4, 0 that is these two patterns are all reversed and 1 is the initial and this one. So, then you know that I am accessing this way and reverse way I am accessing this way so, don’t have to remember the cell name.

So, we are using a BIST in this way but what we have doing by accessing the cells in a random way what is the advantage? The area of the BISTs LFSR are pattern generator for the LSFR is very simple. It is just equals to some x or gate and NOR gate access for generate patterns. So, we are doing a March test only, but we are not go in a sequential approach. So, going in a sequential approach we will make a very high area operate for the address generator, which is the pattern generator in case of LSFR.

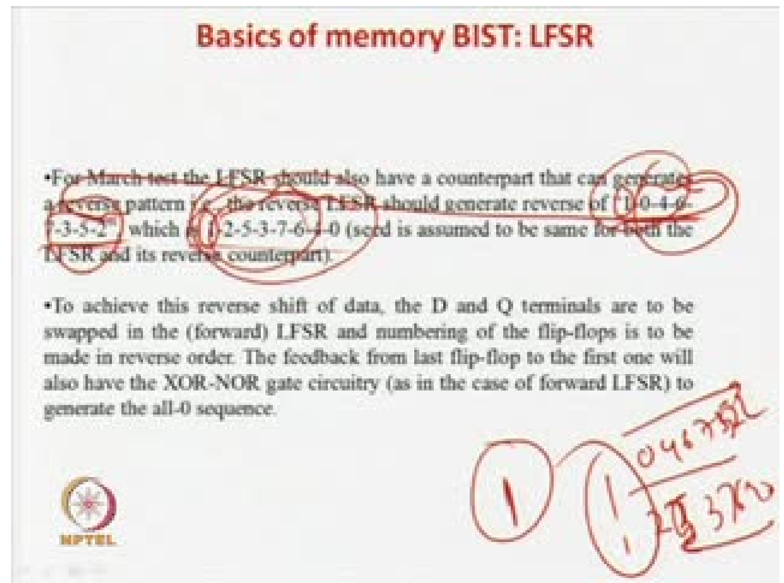
(Refer Slide Time: 60:11)



By this technique we are very easily testing stuck at 0 fault and transition faults. Because the stuck at 0 fault and transition faults, we are taking 1 cell and verifying the other cell. Writing 0, writing 1 and writing 0, writing 1. Similarly, for this cell also. So, we are not taking any kind of coupling in between this 2. We are checking with stuck at 0, stuck at 1 stuck at 0 , stuck at 1, raising, falling and raising falling.

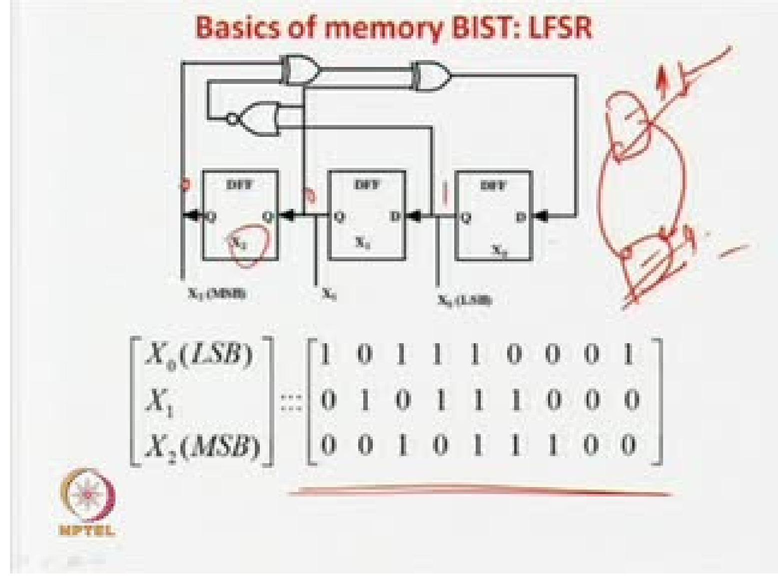
So, we do not check any kind of coupling. So you either access this 1 first or this 1 first, it does not have any matter, there are no requirement for any kind of sequence or pre-defined sequence for accessing. The only thing is that you should remember that I have accessed this one before and this I have access this one after this because, otherwise there will be a problem.

(Refer Slide Time: 60:42)



That is why if you are going in a sequence I mean 1, 2, 3, 4 if you are going in a sequence of this one in the forward iteration and backward or if you are going or a coupling fault testing or neighborhood sensitive testing in this case the architecture will be difficult. You have to remember that cell 0, cell 1, cell 3, cell 4, you have to write in this particular order, what is the value of write, this and again you have to access cell no 2 and defective cells and so forth.

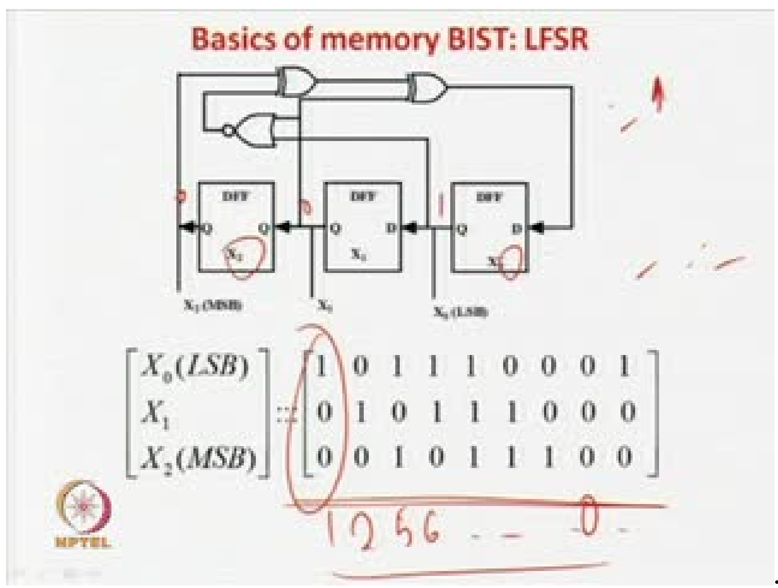
(Refer Slide Time: 61:23)



So, in that case you BIST architecture will be very complex. Many times we not go on for that complexity because your BIST area hardware will be very, very high and we also, discussed that it has been observed that if you go 100 percent testing for stuck at and transit faults memory in BIST. So, other faults will also, get listed that is given the limitation of the area of that.

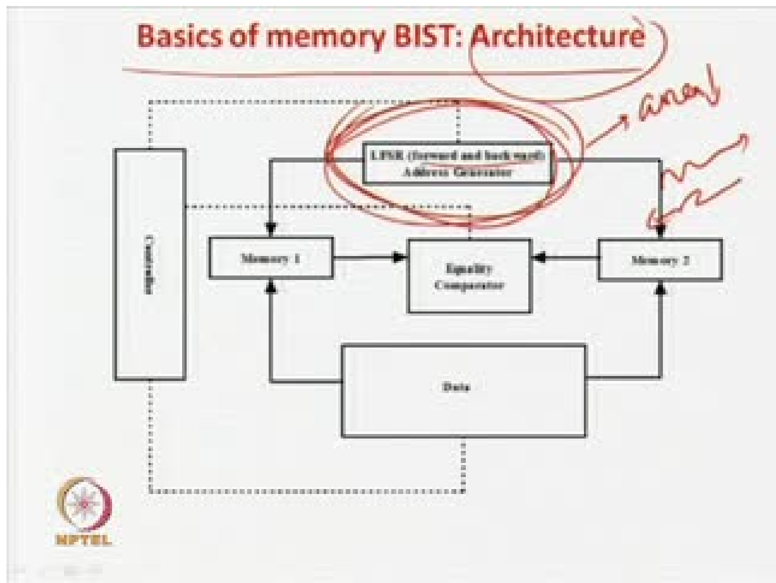
So, again the same LFSR just a bit reverse in this case you make it 2 over here and x 0 over here and it will generate a required pattern like 1, 2, 5, 6 and what you call this pattern which you are generating 0 and this one in a reverse way. So, this LFSR. So taken this same LFSR this is the LSFR which is generating the forward sequence like 0, 1, 4 this thing and now we are doing it in a reverse way just we are reversing this 1 and connected 1 are reversed.

(Refer Slide Time: 61:50)



So, as I told you generally use 0, 0, and 1 at the seed and you can see that you can generate the pattern over here in a reverse way.

(Refer Slide Time: 62:09).

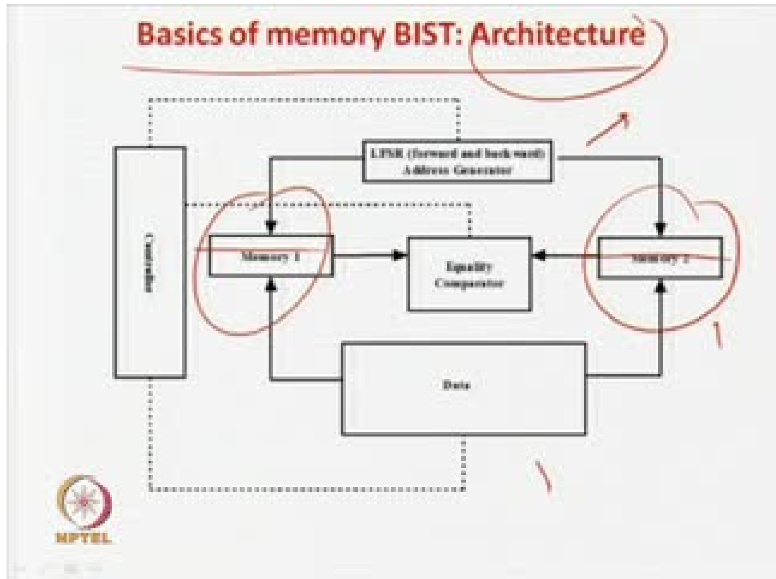


So, what is that very simple LFSR, but we require a pair model to go about the March test. OK. So, this about the pattern generators, so this is the very important part this is this is the basic part of whole BIST architecture for the memory. So, this LFSR we have already discussed. So, it has a forward and backward address generator. So it was actually generate the address in some sequence 1, 0, 3,2,5,7 and 2, 5 something like that.

So, this is the LFSR which will generate at the forward and backward address. So, we are using a random order of addressing and again coming back in the reverse order. So, we have seen that can be very easily done with the LSFR just by adding a few x or gate and the NOR gate that is very simple. So, this area is very, very rare and we are able to go in the some part in random order forward and some random order in the backward that is possible.

Now what you do is that so, if you remember in case of standard BISTs or that is in the BISTs for normal circuit kind of a thing so, what we have done? So, we have taken what you say that you have to remember the golden response stored in a ram and we have done it, but in this case what we do is that and again where do you store the golden response in some kind of a memory.

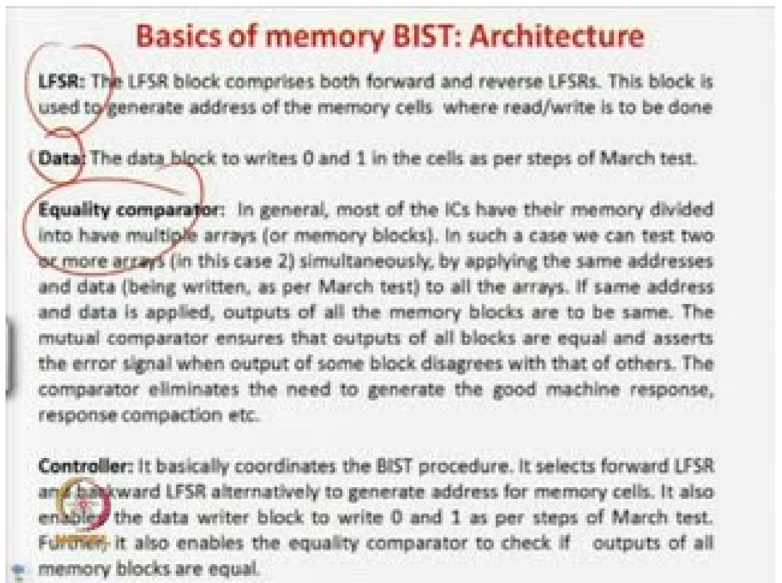
(Refer Slide Time: 63:15)



So, in this case already we have a memory. So, don’t have to put any extra memory stored in the golden response. So, what we do is that we divide the memory in to 2 blocks ok and what you do? You write same access say it may be have memory may be say 0 to 100. So, what we do? We go from 0 to 50 and 51 to 100 over here. Now what you do is that whatever you do with this cell, you do the same thing for this cell. Whatever you do for this cell you do same thing for this cell. Now what you do? You read from this and you read from this and you match cells with their equal amount. So you can think that this is your golden memory, and this is your test memory or the other way you can think that this is the golden memory and this is the test memory.

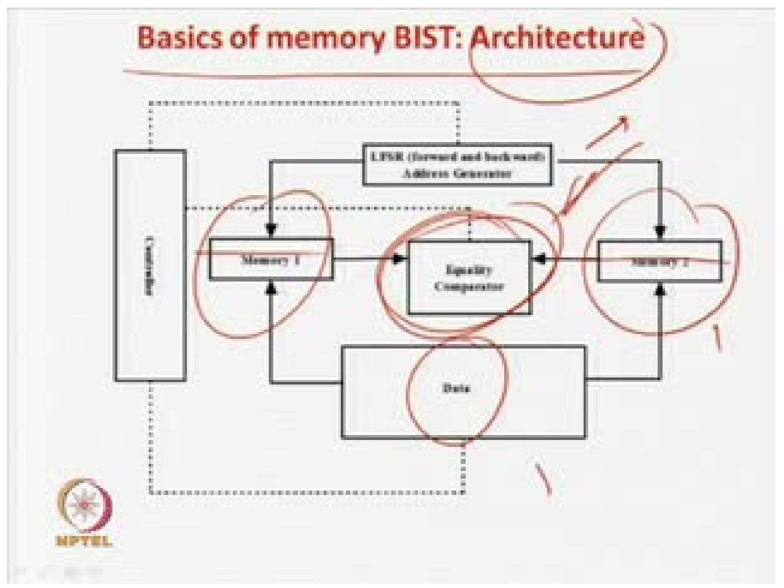
So, whatever you do, you do with 2 memories blocks separately. When you want to read something you verify both of them are equal, if both of them are equal then you are done. So, this is actually a 0 and 1 because you have to write a 0 and a 1. So, whatever the data you want to write that is the 0 in some cell. So, this is your driver already we know, this LSFR address generator are directly connected to you row and column decoder and data is connected to you read that is your read staff that is your cell circuit what you call the driver circuit into the data. The very, very important point is that we do not require a extra memory to store the golden response, because we already have our memory blocks.

(Refer Slide Time: 64:46)



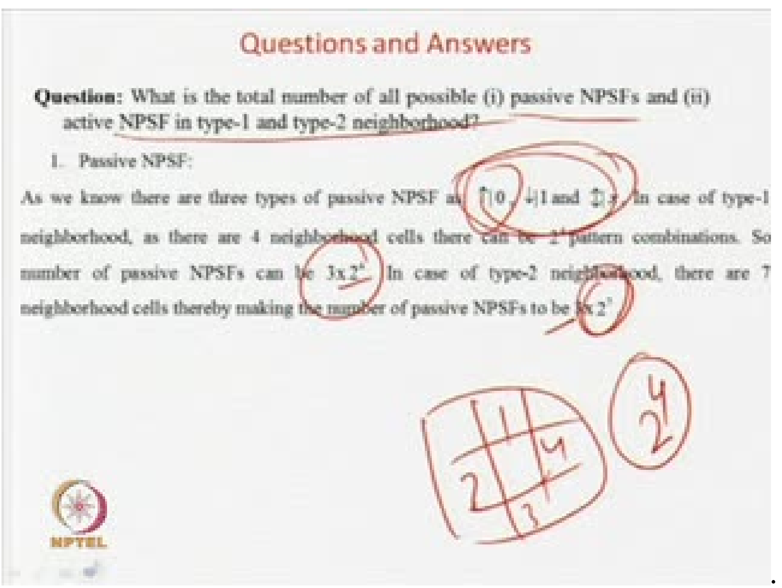
So, what we do is that we divide the whole memory in to 2 halves whatever you do with this memory March test, same thing you do for this 1 while comparing you find out the some difference that is all you can say this is the golden part, this is the test part, of this is the golden part and this is the test part. The very advantage of memory disk block is BIST that we do not have to do any extra wrong to store the golden response ok. So, this all about the definitions block that we have designed LSFR already told that is generates the address in some particular random order. Data which actually writes 0 and 1s in the cells and that is nothing but your what you call cell and driver circuit.

(Refer Slide Time: 65:05)



Equality comparator that is the 1 thing to definitely to have, so just to find out the data equal or not. So, these are very simple combinational circuit and obliviously you require a controller to control the control sequences for all of them. That is very simple and data actually will have what you call driver cells and equally you have compare this equality operators which is simple by calling a series of x OR gates is simple by calling a series of x OR gates. So, this completes actually our design for what architecture for memory BISTs. Now you find out the faults, so you have to go for repairing already told you so, we are not discussing because it is very complex.

(Refer Slide Time: 65:40)



With this we come to the phase question and answer part of it. So, let us see the first question is what is the total number of all possible passive network pattern sensitivity faults and active passive network passive sensitivity faults in type 1 and type 2 neighborhoods. So, we know that what is the passive fault? So, you know that passive faults were that so, I mean that there is actually some you can think about this 1 and this 1. So what are the passive faults?

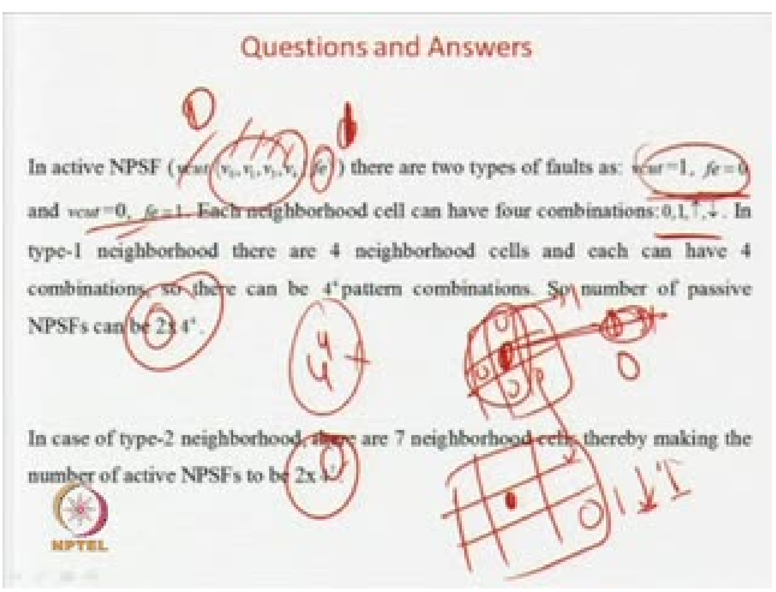
So, there can be some patterns over this area in type if you are considering in type 1 neighborhood. So, we know that this 1, say this 1, this 1, this and this 1 say you put all 0s ok and then you are not able to go for a raise fault. So, this is something kind of a thing. If you want to raise it, still you t to get the value as 0. So, similarity if you have all 1s here and you want to make a 0 over here. You make a fall like here 1 to 0 over here. So,

that you cannot do it. So, all 1s will keep the value as 1. So, this is all about the passive thing, because there is no change in periphery of your fault cell. So, that actually holding the cell value.

So, there are actually 3 types of this thing. So, 1 thing is that periphery is not allowing to go to 1, periphery is not allowing to come down to 0 or periphery is not allowing to change it. So, these are the type 1 neighborhood and this is type of faults in type 1 neighborhood already we know that there are 4 cells 1, 2, 3, 4, ok and 3, 4 . So, what is the all possible patterns over here? It is 2 to the power 4 of ok 0, 0, 0, 1, 0, 0, 0, 1, 0, 1. So, there are 4 cells are there so, you can have two three patterns and what will be the 3 types of faults this are the 3 types of faults. So, 3 into 2 to the power of 4. So, all possible (( )) possible.

Now we know in type 2 neighborhood how many cells we considered? Along this 4 another boundary cells are there. So, there are 7 what you call 7 cells in the periphery of the faulty is type 2 neighborhood. So, all patterns is possible is 2 to the power of 7 and what is the number of faults? It is 3. So 3 into 2 to the power of 7. So, different types NPSF faults can be there.

(Refer Slide Time: 67:40)



Similarly if you go for active staff, active NPSF you know that so, what is the idea? So, the idea is that let us consider the type 1 only. So, there are four neighborhood cells. And there is some activity change over here in some either of the cells like for example, we

have already discussed about this like 0, 0, 0, 0 if you are changing it from 0 to 1. So, it may happen that this guy is also, getting change from 0 to 1, it can be stuck at kind of a thing.

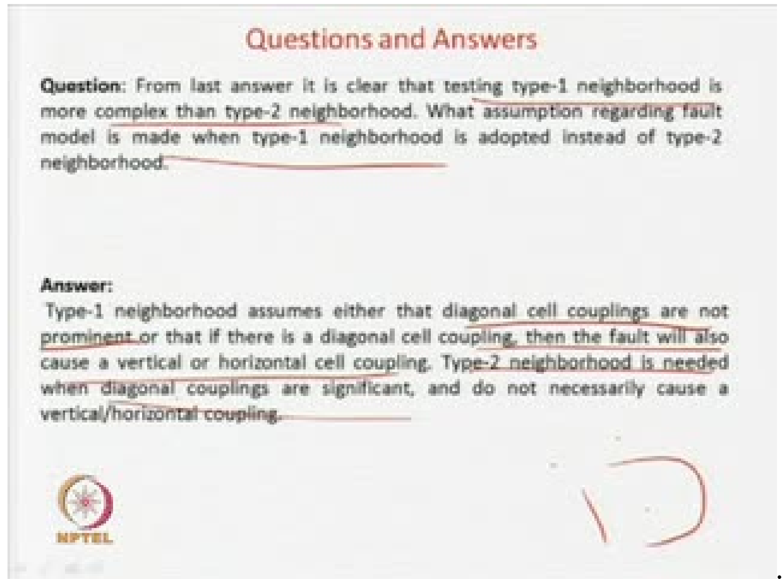
So, there should be some activity in b1, b2, b3 or b4. Then fault affect is there. So, there are two types of faults as we know that if v cut is 1, then you can have that stuck at 0 faults here. If v cut is 0, then we can have stack at 1fault over here. Right if the value is 1 we can make it 0, stuck at 0. If this cell is one, we make it to 0. So, stuck at fault 0 and stuck at 1 fault are possible in this one. But, there should be some activity in peripherals.

So, now each cell can have 3 values , each cells can have 0 (( )) say values raising and falling 0 and 1 are static, but raising and falling one of them must be in 1 of the cells to cause the faults. So, there four possible staff here 0, 1 and raising and falling. So, in this case 4 to the power 4 different kinds of patterns can be possible in the cells like 4 cells are there and 4 different types of patterns are possible.

How many faults are there? There are 2 kinds of faults stuck at 0 and stuck at 1 kind of things. So, all possible patterns are 2 in to 4 to the power of 4. Now if you take a neighborhood patterns type 2 neighborhoods. So, we know that numbers of cells in the neighborhood are how many here? 7. So, 2 into 4 to the power of 7. 2 types of faults are there. If you take this type of neighborhood cells 7 cells will be around, ok? And this is the cell. So the 2 types of faults, like this 1 is 1 fault and this 1 is 1 fault.

So, 4 different types of patterns are possible, in each cell number of cells are 7, the no of faults are 2, no of cells are 1. So, next question is the last answer it is cleared that testing type 1 neighborhood is more complex than type 2 neighborhoods. Because it is 2 to the power of 7 or 4 to the power of 7 if you remember, ok. But still why we go for type 1 neighborhood and what is the basic assumption?

(Refer Slide Time: 70:10)



So, basic assumption is that if you remember in this case, in type 1 neighborhood we assume that these are the 4 cells which can affect you. So, assume that these diagonal cells are not actually affecting the cells because we assume that these are very near to each other and diagonals cells are bit far to each other.

So, this is the very basic assumptions that we are taking that, either the diagonals cells coupling are not prominent or not in minute or diagonals cells fall into. The idea is that that diagonal cells couplings are not prominent because they are more far than other from central cells or cells under faults and even if your diagonal cells coupling are there will cause a vertical or horizontal cell coupling.

So, the idea is very simple in other (( )) words saying that diagonal cells are not affecting I mean that much the central cell only horizontal and vertical cells are causing more effect because of the symmetry and others diagonals. So, type 2 neighborhoods are needed when diagonal coupling are significant. For some kind of case, if you are going for higher coverage or higher conformity to these compliances, so sometimes go for diagonal parts also.

So, but you know that number of faults is 2 to the power of 7 and 4 to the power 7 order. So, you have to pay more time and more cost. So, if you want to go for more accuracy and all those stuff I mean more confidence higher will be the cost. So, depending on your

needs, depending on your test types, so you can go about this. Which neighborhood we can take?

So, with this we come to the end of this lecture and end of the course. Thank you very much for attending the lectures. So, for I mean do we can say, if you want to take more so, you can want more to the handouts, which are available there will be more video lectures, you can download them and lot of references are there. So, wherever you have found out there I mean because it was having 3 courses like design, verification and testing, grasp the one action so, that many time you have to go for references and we have save that both kind of references.

So, interesting readers can go through the references which are available in the handouts and you can read them in details about it. But, being a must course you have trust affects of VLSI design in VLSI verification and VLSI test. So at some point you could not go on it, in depth we have overview of this. For any kind of queries you can use the question-answer I mean blog or question- answer tab of the NPTEL website or you can directly e-mail us to me or professor Dekha in the mails and you will get the responses. Thank you once again for attending the lectures and all the best for your future.

Thank You.