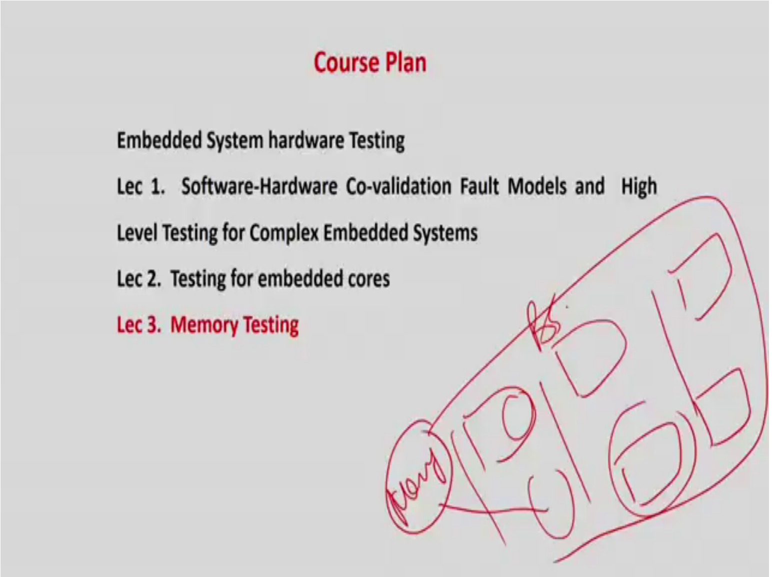
Embeded Systems - Design Verification and Test
Dr. Santosh Biswas
Prof. Jatindra Kumar Deka
Dr. Arnab Sarkar
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati

Part-3: Embedded System Testing
Lecture – 31
Bus and Memory Testing

Hello students and welcome to the 3rd part of the course on Testing of the Embedded Systems.

(Refer Slide Time: 00:36)



Course Plan

Embedded System hardware Testing

Lec 1. Software-Hardware Co-validation Fault Models and High Level Testing for Complex Embedded Systems

Lec 2. Testing for embedded cores

Lec 3. Memory Testing

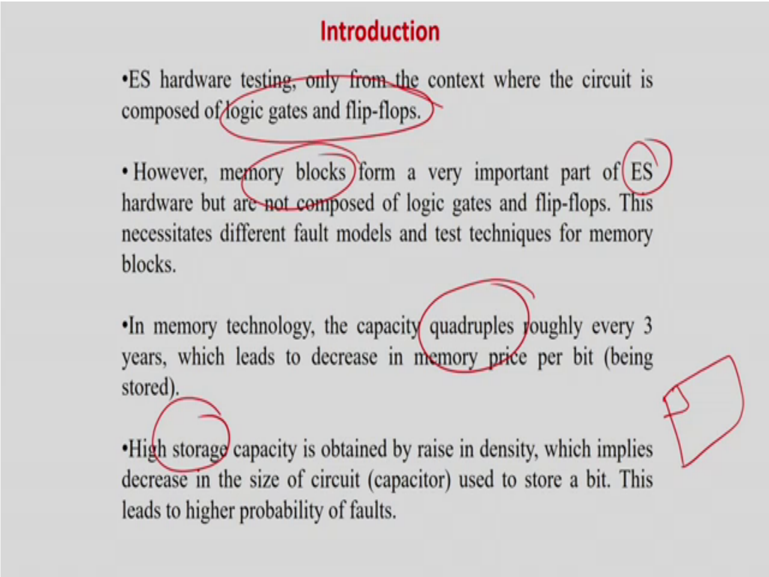
So, as we all know that basically we are now looking at the testing paradigm for embedded systems. And in this module, we have already seen the lectures on software based fault models, and then we have seen how embedded course basically are tested using 1149 dot or P1500 standard.

Basically what we are emphasized in this part till now is that basically these systems are more complex compared to the gate level design in case of embedded solutions; so how can we have more abstract fault models. And once we have this abstract fault model for the cores, and if there are multiple cores in a system, then how we can use boundary scan based technologies to test them. And in the third lecture on this part is based on (Refer Time: 01:13) to memory testing now what.

So, we have already seen in the last two lectures of this part that basically we have some cores right; we have some basically some IP cores within the big embedded hardware. And we have some kind of a boundary scan architecture to access this. But, among these cores, some of the cores will be basically memory, all other cores are basically designed for basically like our normal sequential circuits having flip-flops, registers, gates, etcetera, but few of the cores will be simple memory cores ok.

And even some of the basic inside the basic cores also, we can have some memory module on that part. So, basically we will now see in this lecture that why memory testing is quite different from basically your normal circuit testing, and what extra is involved in testing such embedded hardware, which comprise the memory modules. So, this part is basically dedicated to memory testing.

(Refer Slide Time: 02:06)



Introduction

- ES hardware testing, only from the context where the circuit is composed of logic gates and flip-flops.
- However, memory blocks form a very important part of ES hardware but are not composed of logic gates and flip-flops. This necessitates different fault models and test techniques for memory blocks.
- In memory technology, the capacity quadruples roughly every 3 years, which leads to decrease in memory price per bit (being stored).
- High storage capacity is obtained by raise in density, which implies decrease in the size of circuit (capacitor) used to store a bit. This leads to higher probability of faults.

So, basically as I was saying embedded hardware testing, only from the perspective of logic gates and flip-flop that is what we have seen till now. There are all the modules we have seen basically have (Refer Time: 02:14) gates and flops sequential or combinational circuits. But, memory blocks also form a very important part of any hardware like embedded systems, because cache plays a very important role.

And you will cannot think of any design right now without having a memory chip, because you any anything you have to store, I mean I think I need not repeat this line at any more that no application is can it be even thinkable, if you do not have a inbuilt memory in the course. Like, we always talk about the RAM size of this mobile phone is so much it has so much storage, so you can download and keep so much photographs etcetera, etcetera, etcetera. So, memory plays a very very important role in the hardware paradigm of embedded systems.

And basically memory technology, the capacity actually becomes very very high every 3 years. So, in that what people try to do is that try to make the transistors or basically I should not call it transistors, you basically have to I request that you please look at the memory architecture that basically how a memory cell is designed. You can look at any standard VLSI design or textbook or any NPTEL lectures on VLSI design, then you will find out that how memory cells are designed.

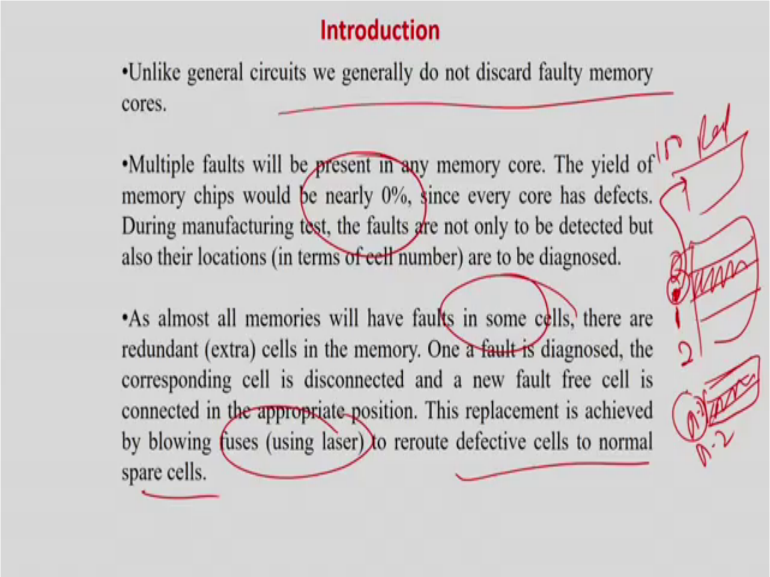
So, memory cell basically is nothing but it has to store a 0 or a 1 as simple as that, then I can put a flip-flop to do it. But, in fact it (Refer Time: 03:25) increase the area of the memory very much, it will almost become like a normal sequential circuit. To avoid this, we have very specialized memory cell design in which case is the capacitor or maybe some specialized circuit to store the charge corresponding to 1; and if get discharge, it corresponds to 0. The idea is that the function (Refer Time: 03:43) very similar to a flip-flop that is store 0 or 1, but it has to be done in a much much lower area compared to a flip-flop design.

So, basically we have to have high storage capacity and we have to have very low small size devices to store them 0 and 1. So, basically what we do, we try to have a thin same die area for memory, and I try we try to increase the capacity. So, how we can do that we have to shrink the design of each of the memory cell device.

Like, if it is a capacitor, you will try to make it as small as possible. If it is a combination of transistors to implement a memory cell, you have to try to shrink the memory the size of the transistor as much as possible, so that in a given die area, you can push more and more memory cells, so that your memory capacity becomes higher given the same die size that is what is the need of the (Refer Time: 04:30) and every year or every means every size cycle of memory (Refer Time: 04:33) we are trying to achieve that goal.

Within a single chip, we want to bring more and more gigabytes into the same memory core. So, if we do that, of course discussed so many time in the introductory lecture these needs to higher probability of false. Because, we are trying to scale down the devices as much as possible, and these leads to increasing the probability of false, just as in the case of normal circuits.

(Refer Slide Time: 04:56)



Introduction

- •Unlike general circuits we generally do not discard faulty memory cores.
- •Multiple faults will be present in any memory core. The yield of memory chips would be nearly 0%, since every core has defects. During manufacturing test, the faults are not only to be detected but also their locations (in terms of cell number) are to be diagnosed.
- •As almost all memories will have faults in some cells, there are redundant (extra) cells in the memory. One a fault is diagnosed, the corresponding cell is disconnected and a new fault free cell is connected in the appropriate position. This replacement is achieved by blowing fuses (using laser) to reroute defective cells to normal spare cells.

But, the probability of occurring a false in memory is much much higher than that of a normal sequential or combinational circuits. So, unlike general circuits we do not discard faulty memory; this is very interesting, you should note this. That in normal circuit what we have told, you take a device, manufacture it, test it by (Refer Time: 05:12) any default, you find you throw that chip and be in the normal ones to the customer.

But, if I want to do it same thing for the memory, the chip yield will be nearly 0 percent, because every memory will have some fault or the other. So, what people do? How they do (Refer Time: 05:24) the solution is basically all memories have first of all built in self-test. So, whenever you switch on your demo computer or in your mobile phone, basically all memory is actually checked before the system starts its operation.

Then if I found something a problem or some cells are having a fault, what I do basically, or even of the manufacturing, if you find out that some of the memory cells have faults. And as again repeating, this is more likely to happen in most of the memory that is fabricated. Unlike the yield is 70 to 80 percentage in case of normal sequential and combinational circuits, but due to these so much shrinking down of the memory cells. This phenomenon is very very high in memory compared to normal circuits. So, almost all chips memory cores will have faults. So, now what it does.

So, almost all memories will have in fault faults in some cells, so what you do. So, whenever I say one (Refer Time: 06:11) of memory, basically I always keep some redundant cells in the memory itself. And I build (Refer Time: 06:16) a building shelters of the memory chip, and whichever cells you find out that is the problem and all, you will any benefit from the memory logically and it will put it in the other structure.

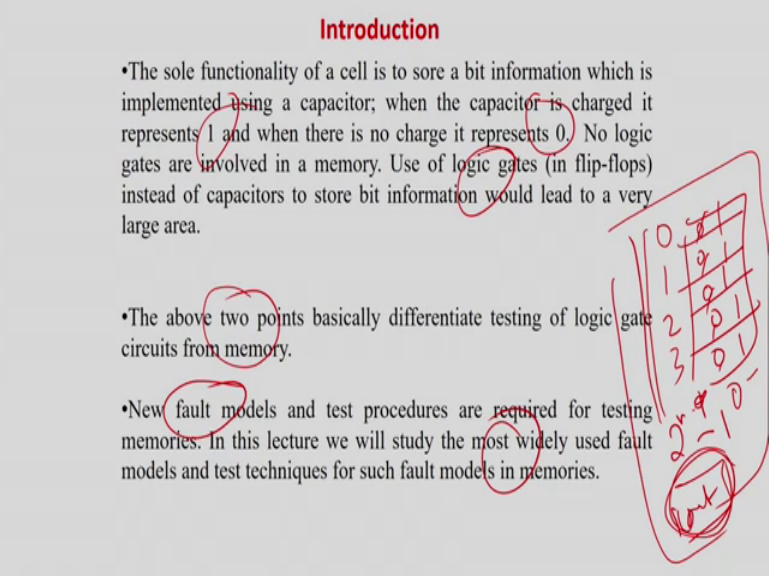
Like for example, I am not going to go into the details, this is a simple hardware reorientation. So, maybe this cell having a has a problem, and maybe you have a logic, so these are some of the logically extra cells. So, this one will be logic 1 address, logic 0 address, this one will be skip, this one will be logic 1, logic 2 and so forth. May be this cell is damaged, so say this maybe n minus 2, so this will be skipped. And the next cell will be made in minus 2 address. So, there is a logical way of handling that you skip those cells logically, but physical you cannot skip it, because anyway the cell will be there in the memory. So, logically you skip these cells by changing the address and basically you reorient so that that cell is no longer in picture logically in the memory.

So, this replacement is achieved by blowing fuses to read out defective cells to normal spare cells that means, somehow you logically map the address of those not redundant cells in case of the damaged cells. So, for example, cell number 1 is damaged, but basically you will maybe this is this is maybe this is some redundant cell is there. So, in this case, if I try to address memory location 1, in fact it will be redirected to with (Refer Time: 07:30) memory location 100.

So, slightly reorientation will be done by blowing up some fuses, so that basically it happens run time. Sorry, after manufacturing, if you find out something like that, you basically try to reorient this by blowing up fuses, so that your chip typically remains of 1 gigabyte, but some of the memory cells which are already found to be faulty or replaced logically by blowing of these fuses.

(Refer Time: 07:52) not very much relevant to this lecture as of today, because we will be not looking at such kind of technology. Because, they are more related to physics and not relate to basically or VLSI design, but not that much needed to embedded systems, but I am just trying to give you the information that unlike normal circuit [vocalized noise] failure probability is much much higher in case of memory. Almost every memory core will have a fault, or cells will have a fault. So, what you do in the solution, you have lot of redundant cells, reorient or reorganize the memory address logically so that the whenever you try to access a logically faulty cell, you access the counterpart redundant counterpart, which is normal corresponding to that defect itself ok.

(Refer Slide Time: 08:31)



Introduction

- The sole functionality of a cell is to sore a bit information which is implemented using a capacitor; when the capacitor is charged it represents 1 and when there is no charge it represents 0. No logic gates are involved in a memory. Use of logic gates (in flip-flops) instead of capacitors to store bit information would lead to a very large area.

- The above two points basically differentiate testing of logic gate circuits from memory.

- New fault models and test procedures are required for testing memories. In this lecture we will study the most widely used fault models and test techniques for such fault models in memories.

So, now as I told you that the sole functionality of a memory is to store 0 and 1. So, I can use a flip-flop, (Refer Time: 08:37) but it would loop to very large area. Therefore, we have smaller devices, like as I told you a capacitor or you can have some orientation of some circuits of transistors, which can do that. You have to basically read out some text with basic materials on memory reside in VLSI textbooks ok.

Now, so this points basically that the job of a memory cell is just to store 0 and a 1 and basically you just store a 0 and retrieve a 0, you do not have to do anything else, just store retrieve, store retrieve, and that also you have to do for 0 and 1. So, this actually necessitates that there should be new fault models and new test algorithms compared to a normal circuit.

So, normal circuit basically our ordinary circuit like sequential and combinational circuits, all the modules will compute some inputs will be there, it will compute the output values. But, here in memory is slightly different, you just store a 0, retrieve of 0; store our, retrieve our. So, this actually is fundamental difference between a memory circuit and a normal circuit, so that is why test patterns (Refer Time: 09:35) will be different and basically test algorithms or ATPG or applying of the test patterns will be entirely different in case of memory compared to normal circuits.

Of course, I think you are getting a very good idea that if some device is there, where you just store a 0, retrieve a 0, store a 1 retrieve a 1, ATPG is nothing. ATPG is just it will say that you store a 0, retrieve a 0 and that is it. So, ATPG is just redundant in case of memory, in fact it is true. So, in case of memory, the ATPG is just saving a 0, retrieving a 0, and saving a 1, retrieving a 1. If that is possible, the memory cell is working fine, only we have to verify that.

So, in fact ATPG is not required in case of memory, only thing is that how do I apply the patterns to a memory is a challenge. Because, the whole memory is quite large and basically we have to apply some kind of test routines or that is in each cell you have to write a 0 and get back a 0 and 1 and write a 1 and read a 1 from each of the cells that you retain do it for the entire memory. So, it may take some more time.

So, interestingly basically as the test pattern is very simple writing all 0s, getting all 0s writing all 1s and getting all 1s, you may not use a (Refer Time: 10:38) to do that. Basically there will be some on chip hardware to do these solutions, which will covering up in the next module, which is called built in self-test. So, in built in self-test what we do, we put a hardware pattern generator inside the chip and analyse that inside the chip.

So, this is used for some different purpose, which we will see in the next series of lectures. But, for the time being, if I say that a memory has 1 GB size, so you have to apply all 0s to all cells read back and for one will repeat it, it will take long time in the (Refer Time: 11:03). So, greater than that what I can do, I can put a simple counter as a hardware in the circuit, and all (Refer Time: 11:09) will write all 0s, then it will read all 0s, and it will be done by on chip hardware, because it is very simple.

The simple counter will count from 0 to the memory size, and in the BIST B I S T is called build in self-test lecture, which will be done in the next week basically, I mean in the forth coming lectures. We will find out that to generate all patterns from 0 to 2 to the power n minus 1 can be done using very small size circuits called linear feedback based (Refer Time: 11:32) register. And some different technologies are there in which case you did not put an explicit counter to generate all these values from 0 to 2 to the power n minus 1.

What, why I am saying 0 to 2 to the power n minus 1, basically there will be a memory. And for all these cells like 0, 1, 2, 3 all the address to the power n minus 1, I am assuming that there are basically minus 1, I am assuming that the address bus size is n. So, you will write all 0s over here, writing 0s, and reading back the 0, then we write all 1s, and reading back the 1s (Refer Time: 12:02) basically do that.

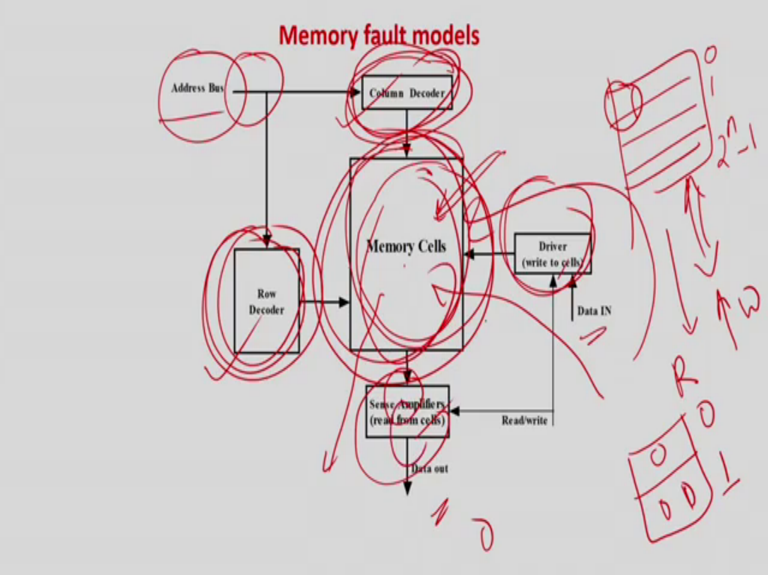
So, in this case, a counter will be there, which will count from 0, 1, 2, 3, 4, 2 to the power n minus 1, and again it will may go back reverse. So, there is a counter up and a down counter, which will address the memory and each cell you have to write 0, get back 0, write a 1, read back 1. So, ATPG is very simple, reading and writing and comparing also very very simple. So, the on chip hardware to do this is a very very simple circuit, only the area over it will be because of the counter, if the circuit memory size is large.

But, as well see in the BIST lecture that you can have very small implementation of circuits are possible, which are called linear using linear feedback shift registers in which has I can generate such patterns or count patterns with (Refer Time: 12:39) very very minimal hardware compared to a normal up and down counter. So that building save the circuits are generally used to do the testing, so that is why people will not put a memory chip to a (Refer Time: 12:50) and do the test. Because, going through this and again alternating all address and put reading (Refer Time: 12:55) reading 0, writing 1 etcetera will take lot of time, which is not required for an (Refer Time: 12:59). It is a much more powerful device, it will give logical patterns, it will analyse it, which we are which are done for normal sequential and combinational circuits.

Memory testing is very simple in one case that the ATPG is very simple, only problem is that entire memory you have to go through all the cells back and forth and keep on writing 0, reading 0 and so forth. So, we do not give an (Refer Time: 13:19) as intelligent machine like (Refer Time: 13:20) do not we do not employ, an intelligent machine like (Refer Time: 13:23) to do this dumb job. We basically built a in on chip counter, in fact I should not call it on counter basically it is a LFSR kind of a register, whose area is very very small compared to a normal up and down counter. So, because they are on chip hardware, because ATM is it is a separate device, you connect it to the chip and tested.

But, in case of on chip hardware means along with the memory, you should have this counter on chip fabricated. So, I cannot have a very high area overhead for that, so but it is a very as we will see in the next few lectures that how we can design such a small area counter, which will generate all the addresses. And then just reading 0, writing 0, writing 0, reading 0, and same for the 1 is a very simple methodology to do that, so that way we test our memory, when you are using a on chip hardware to do it.

(Refer Slide Time: 14:07)



Memory fault models

Address Bus

Column Decoder

Row Decoder

Memory Cells

Driver (write to cells)

Data IN

Sense Amplifiers (read from cells)

Read/write

Data out

Therefore, whenever you boot up your system, you will always see that there is something called memory testing. (Refer Time: 14:10) if you are seeing an old machine or basically if you just when you boot up your computer, you can see that memory testing is going on 0, 1, 2, 3, 4 up to some k memory tested. Basically the memory cells are read and written, when you switch on your machine. And if some problem happens in some of the memory shift, there is a scope of logically reorienting these cells, so that the redundant cells are used.

So, now what will focus in this lecture. In this lecture, we will focus two things; first what are the different fault models for the memory, and how basically test patterns are applied. Because, test pattern generation is very simple, it is simply applying a 0, reading a 0, and writing a 1, and reading a 1 that is the ATPG for memory cells. But, important is how do you generate it, and what are the different fault models, because stuck-at fault model does not at all suffice for women.

Why? Because as I told you stuck-at fault model is suffices for the transistor based AND gate, OR gate, NAND gate. But, memory cells are more and more restricted in design or more constrained in size compared to a normal gate level design, so that is why fault probability is higher in case of memory cells, and also different advanced type of fault models are required or different type of all models are required in addition to stuck-at faults.

Stuck-at faults are very very important, they are fundamental to any design hardware. But, as the memory is more sophisticated devices has more tiny compared to a normal circuit, so therefore you have to put more different (Refer Time: 15:28) fault models apart from stuck-at fault model. So, we will see these two important things in today's lecture; different type of fault models liquid, and how test patterns are applied.

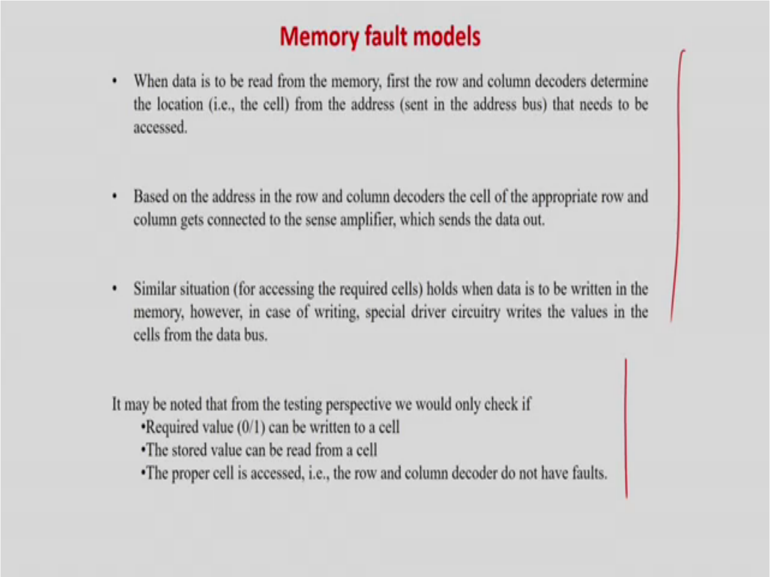
This is your memory, so this is your memory cells. What happens, we know there is a address bus, it is a column decoder and the row decoder, so whatever address you give over here. So, basically the data will be coming out from the memory cells to something called source amplifiers, source amplifiers in the analog circuits, they will take the memory cell values and put the counter version in the data out.

Similarly, when you want to write it, you have to write there are some driver cells, whatever value you give that will be actually given to a driver cells, which will they are all analog circuits, which will correspondingly convert into relevant voltage levels, which will be written into the memory cells. So, details about how about the column decoder, what is a row decoder, how they are accessed, you can read any standard architecture book, you able to find out.

But, for the time being we just need to know that memory is nothing, but there is a lot of cells, it has an address from 0, 1 to 2 to the power n minus 1. If there are n addresses and there is a data bus. So, either you can read it from the bus to the memory cell or you can read it or you can read from the memory cell to the throughout this through this direction that is read from the memory and you can also write from the outer word to the memory block that is called the write mode.

And basically we have some analog circuits to do the voltage translations and interfacing. So, they are sense amplifiers, which will read the values from the memory source to the external (Refer Time: 16:49). And if you want to write somewhere to the memory cell, you have to use a driver circuit. Very fundamental digital computer architecture material, you can find it in your any textbook, I just given this one for the sake of reputation.

(Refer Slide Time: 17:01)



Memory fault models

- When data is to be read from the memory, first the row and column decoders determine the location (i.e., the cell) from the address (sent in the address bus) that needs to be accessed.
- Based on the address in the row and column decoders the cell of the appropriate row and column gets connected to the sense amplifier, which sends the data out.
- Similar situation (for accessing the required cells) holds when data is to be written in the memory, however, in case of writing, special driver circuitry writes the values in the cells from the data bus.

It may be noted that from the testing perspective we would only check if

- Required value (0/1) can be written to a cell
- The stored value can be read from a cell
- The proper cell is accessed, i.e., the row and column decoder do not have faults.

So, anyway whatever I have told you is basically written in this slide, when data is to be read from the memory, first the row and column decoders determine the location that is the cell from the address that needs to be accessed. Based on the address, I mean the values will be written into the corresponding memory cells. So, similar situation holds, when data is to be written, so this holds for (Refer Time: 17:19) to be read and written.

And basically we know that now if this part is important, this is a similar thing as I have told you that the address given, it find out which cells to be read and written, but what I want to test over here. (Refer Time: 17:32) complicated, it seems to be very very complicated that I mean there is a row decoder, there is a column decoder, they are nothing but digital decoder circuit. So, they are nothing but simple combinational circuits simple decoders.

Drivers are analog circuits. And memory cell as I told you is very very important, they are made, this is because these are basically simple pure combinational circuit. I do not, I cannot make any tricks over here to reduce the size, only think is this is the core of the memory, where 1 GB, 2 GB, 3 GB whenever you are saying that many amount of memory cells are available over here. So, the idea is that I want to make it as constrained in size as possible, so that I can make my memory size increased.

So, therefore always people try to attack or falls more (Refer Time: 18:09) or in this part of the circuit that is in the cells, because it is much more cramped, and it always starts becoming cramped to put more data in a single dice (Refer Time: 18:17) similar die size. I want to put more memory, so I have to make this as small in size as possible. So, therefore, we always want to concentrate on testing of this cells.

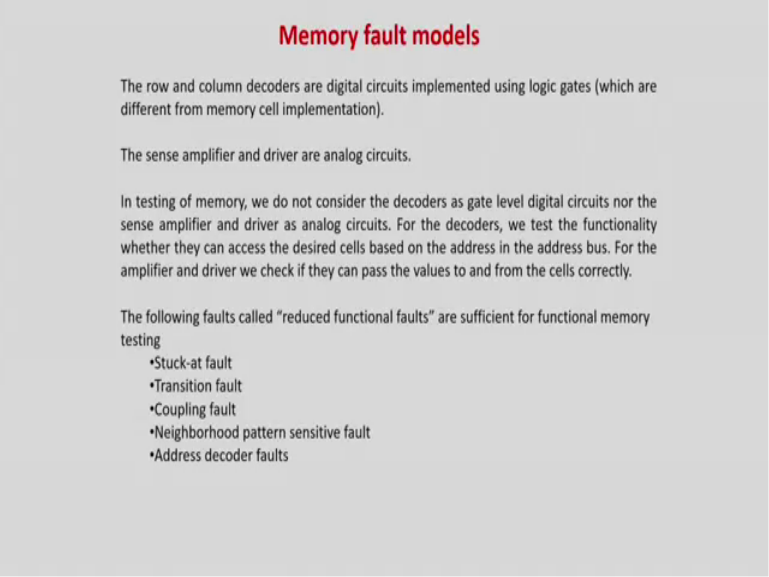
But, then you can ask me that how I will test the decoder, how will test the decoder, how I will test the column and all. People make a very simple rule, I will basically acts I will give some address, I will write to a memory cell 0, I will read from the (Refer Time: 18:38) cell 0. If I am able to do it, you know that address bus is operating address mechanism is happening properly, the cells amplifier is happening properly, driver is operating properly. How? Basically let us assume (Refer Time: 18:49) this memory has only two cells; 0 and 1. First, I will put the address 0, I will write the value of 0, then I will read the value of 0, fine. Then I will go to address number 1, write the value of 0, read the value of 0, and then I will repeat it for 1 (Refer Time: 19:03).

So, accessing is memory cell properly writing a value, reading a value, if everything matches that means, every circuit is operating fine. Because, row decoder is fine, because your (Refer Time: 19:13) giving a proper address, then only you are going to address the proper memory cell. The value you are writing and you are reading it back is proper that means, both the sense amplifiers and drivers are operating fine. So, therefore, I do not want this just like a normal way of fault abstraction.

Like, as I told you in fault model, (Refer Time: 19:27) the fault model, you can say that AND gate is very very complicated, you can test the AND gate, you can test the transistor level, you can do it at the layout level testing, so it becomes a very very complicated system to think in this manner. So, rather what I have tried to do here is basically instead of making the thing very complex, I just to do very simple philosophy. Address a memory cell, read it, write it, read it with 0, repeat it for 1, done with all circuits are operating fine, so that is how basically our memory (Refer Time: 19:52) is going to happen.

We are not explicitly trying to test the memory code itself, then we are not going to again repeat it for the mean that means, we are not separating separately testing this, this, this, this and this, this you are not going to read separately. We are just using this technology, I have told you that access a cell, write a 0, read a 0, repeat it for 1, done. It ensures that the memory access mechanism is fine; so this and this are fine. Value is saved and (Refer Time: 20:19) properly means this is fine. Safe properly means basically driver is fine (Refer Time: 20:23) properly means sense amplifier is fine done, so that is how we will basically do.

(Refer Slide Time: 20:28)



Memory fault models

The row and column decoders are digital circuits implemented using logic gates (which are different from memory cell implementation).

The sense amplifier and driver are analog circuits.

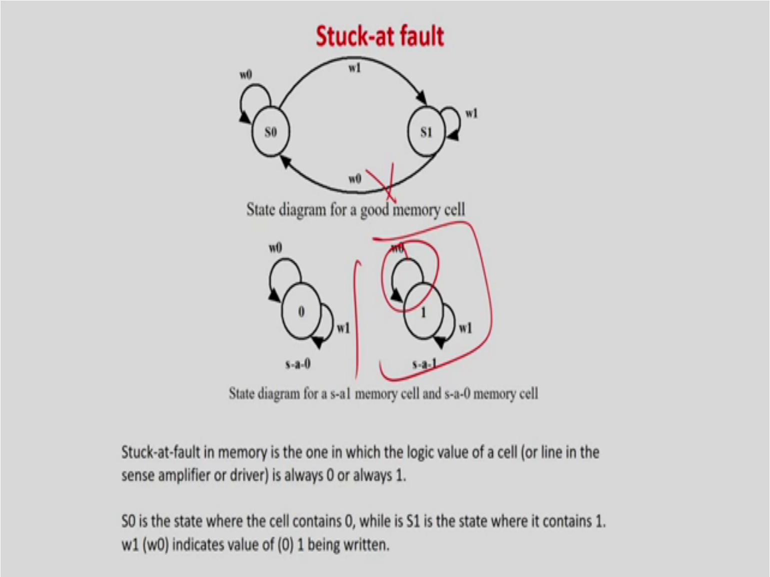
In testing of memory, we do not consider the decoders as gate level digital circuits nor the sense amplifier and driver as analog circuits. For the decoders, we test the functionality whether they can access the desired cells based on the address in the address bus. For the amplifier and driver we check if they can pass the values to and from the cells correctly.

The following faults called "reduced functional faults" are sufficient for functional memory testing

- Stuck-at fault
- Transition fault
- Coupling fault
- Neighborhood pattern sensitive fault
- Address decoder faults

But, what more is required is the fault models. Now, simple fault model like stuck-at fault will not do the lot of other fault model has to be taken into picture, because the devices are much more cramped up here. So, what are the stuff fault models you are going to look at here, stuck-at fault model is very important, then there is something called transition fault model, coupling fault model, neighborhood pattern sensitive fault model, and address decoder faults. We will look at each of the different fault models; how they have to be tested, we will now look at it. Now, why more fault models, the idea is because more are device becomes sophisticated in fabrication, more different type of failures can occur, so stuck-at fault does not along (Refer Time: 21:03) to do that, you have to put more different fault models to do it ok.

(Refer Slide Time: 21:08)



Stuck-at fault

State diagram for a good memory cell

State diagram for a s-a1 memory cell and s-a-0 memory cell

Stuck-at-fault in memory is the one in which the logic value of a cell (or line in the sense amplifier or driver) is always 0 or always 1.

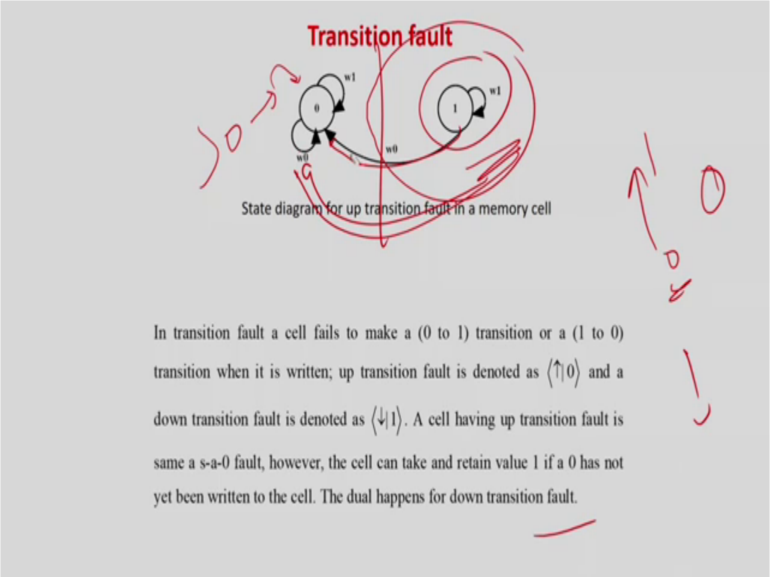
S0 is the state where the cell contains 0, while is S1 is the state where it contains 1. w1 (w0) indicates value of (0) 1 being written.

So, this is first we will start with our single very well known stuck-at fault model. So, what is the stuck-at fault model basically. So, this is a normal state based machine model of the memory. Cell 0, if you write a 0, w0 means write a 0, so 0, 1 0 will be written of it, I am going to be the same. I write a 1, so you go to the next state. S 1 means the memory cell has a value 1 and this means the memory cell has a value 0.

So, from if the memory (Refer Time: 21:34) a value 0, if I say w1, it will go to this state of course. And in the memory cell already a 1 has been written. So, if I write a 1, all being the same state. Memory cell has a value of 0, sorry you have 1, now I want to write a 0 (Refer Time: 21:46) I go back to the same state states this states you very easily understood. Stuck-at fault means what? Permanently add 0. So, state 0 means the memory cell is (Refer Time: 21:55) having a value 0. I write a 0, I live in the same state. Even if I want to start writing a 1, I cannot go, I will not be able to go to the next stage, I will be basically looping in the 0 state itself (Refer Time: 22:05) stuck-at 0.

Stuck-at 1 means permanently the memory cell is in 1. So, if I want to write a 1, no problem in the same state. But, if I want to write a 0, I will not be able to go here, I will not be able to go over here not possible, so I will be looping it. So, these are stuck-at fault model. This is a representation of fault models in memory. So, just like (Refer Time: 22:23) stuck-at 1, stuck-at 0 in text, (Refer Time: 22:25) we are doing in d and d prime in case of normal circuit, we represent in a normal final state machine.

(Refer Slide Time: 23:31)



Transition fault

State diagram for up transition fault in a memory cell

In transition fault a cell fails to make a (0 to 1) transition or a (1 to 0) transition when it is written; up transition fault is denoted as $\langle \uparrow | 0 \rangle$ and a down transition fault is denoted as $\langle \downarrow | 1 \rangle$. A cell having up transition fault is same a s-a-0 fault, however, the cell can take and retain value 1 if a 0 has not yet been written to the cell. The dual happens for down transition fault.

There is something called a transition fault, now (Refer Time: 22:33) fault model. So, what is a transition fault model. In transition fault model, this cell fails to make a 0 to 1, or a 1 to 0 transition that is it says that it is represented something like this, it is a transition for that means, what is this, I want to make a 1. From 0 to 1, I want to go, but it remains as a 0. This is called a up transition error, this is a down transition error that I want to make a transition from 0 to 1, sorry from 1 to 0, but still it remains at 0.

Now, you can ask me basically then it is very similar like this one, you want to go from 0 to 1, it remains at 0, so stuck-at fault right. So, this is what has been done. So, basically send 0, if I want to write a 1, you are going to be at 1. So, this mode is stuck-at fault, but it is slightly different from the stuck-at fault as (Refer Time: 23:18). So, if you are just thinking that your memory cell at the start up or at the point, when the failure has occurred is in the state 0.

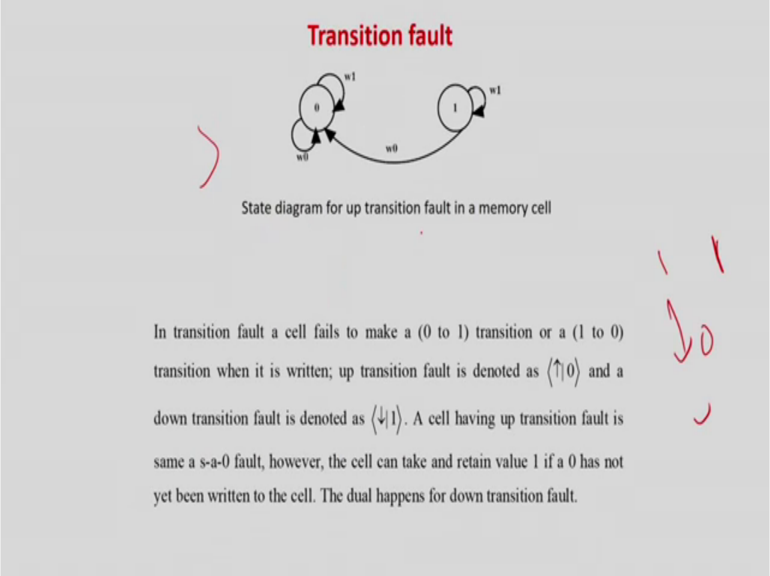
And then basically if I want to make it to 1, basically then this will not happen, it will be remaining in this, it is very similar to stuck-at 0 fault (Refer Time: 23:31). But, say for example, the memory cell is in state 1, when the fault has occurred. In this case, basically what happened. When I want to write a 0, you will be coming over here, and there only the phenomena will be effective.

But, stuck-at 0 means you cannot at all go to a state like this, so this like difference in the default model. Means as well see in the, I mean next few lectures that defined them of fault models have different impacts basically, so stuck-at fault model means at no point of time, the memory could have gone from 0 to with the stuck-at 0 means at no point of time, it can go to 1.

But, in case of a transition fault means transition fault in the sense in like this one from 0 to 1 transition. Basically if you are going from 0 to 1, it is not possible, so it will remain (Refer Time: 24:13) to stuck stuck-at 0 fault. But, (Refer Time: 24:16) if it was at 1, so it is possible that it can be at 1 at starter, but stuck-at 0 means it can never be at 1. But, in case of a transition fault, it can be at 1. But, once it goes from here to here, again transition from 0 to 1 is not possible, and the dual will happen for a down transition fault.

So, same transition fault is very similar to stuck-at fault, but only thing is that stuck-at fault means permanently it will be in either this state or this state. But, in case of a transition fault, initially it can be (Refer Time: 24:44). But, when it comes to this state, it will be locked in state 0 that means, this is the only thing difference this is actually a sorry this is actually a stuck-at 0 condition. But, this is the extra part in case of transition fault. So, we once it comes here, it will be remaining as a stuck-at fault so, these (Refer Time: 25:02) four models are quite similar with a slight difference as I have shown over here.

(Refer Slide Time: 25:13)



Transition fault

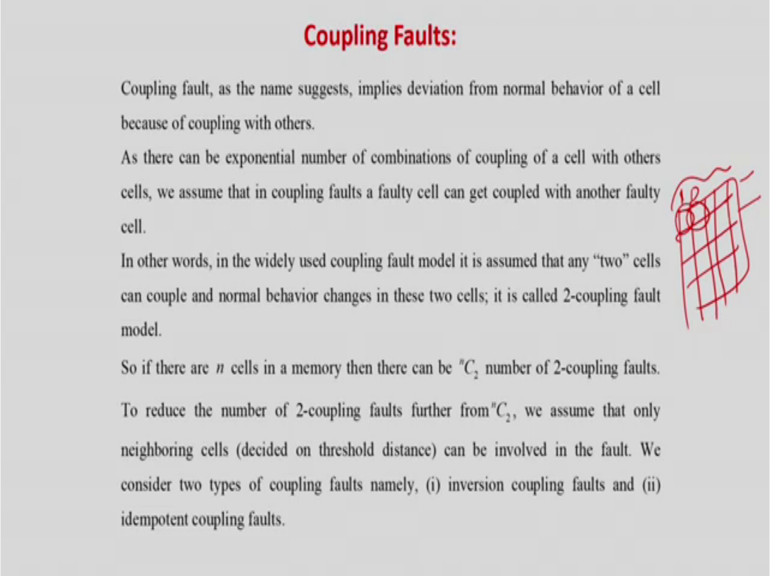
State diagram for up transition fault in a memory cell

In transition fault a cell fails to make a (0 to 1) transition or a (1 to 0) transition when it is written; up transition fault is denoted as $\langle\uparrow|0\rangle$ and a down transition fault is denoted as $\langle\downarrow|1\rangle$. A cell having up transition fault is same a s-a-0 fault, however, the cell can take and retain value 1 if a 0 has not yet been written to the cell. The dual happens for down transition fault.

Similarly, if you can like it will be a dual means, (Refer Time: 25:08) very easily yourself as a homework try for what will be this type of a fault that initially it will be 1.

If it goes to 0, it will be never be there, it will be stuck-at 1. So, if the stuck-at 1 (Refer Time: 25:18) at this case, but not permanently that means, in fact that is like stuck-at fault means basically again repeating, it will be it will be permanently in this (Refer Time: 25:26). But, transition means what? It can be in this state, which is not actually stuck-at 0 now. But, once it comes in this state, it will become a stuck-at 0, and the phenomena will be very similar from them. But, I think you have easily you can easily understand the slight difference between stuck-at and transition fault model.

(Refer Slide Time: 25:43)



Coupling Faults:

Coupling fault, as the name suggests, implies deviation from normal behavior of a cell because of coupling with others.

As there can be exponential number of combinations of coupling of a cell with others cells, we assume that in coupling faults a faulty cell can get coupled with another faulty cell.

In other words, in the widely used coupling fault model it is assumed that any "two" cells can couple and normal behavior changes in these two cells; it is called 2-coupling fault model.

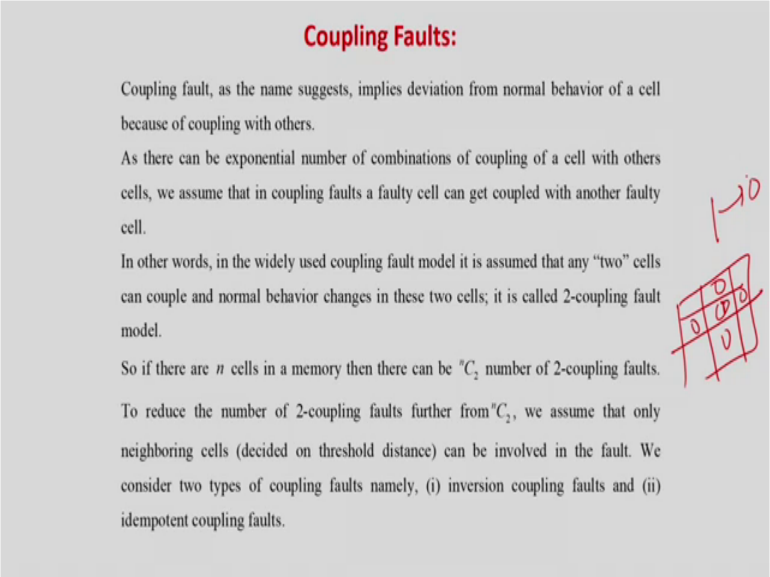
So if there are n cells in a memory then there can be ${}^nC_2$ number of 2-coupling faults.

To reduce the number of 2-coupling faults further from ${}^nC_2$, we assume that only neighboring cells (decided on threshold distance) can be involved in the fault. We consider two types of coupling faults namely, (i) inversion coupling faults and (ii) idempotent coupling faults.

Now, very very important are actually coupling faults. So, this stuck-at 0 and transition faults are from some kind of a older fault models in case of memory, more important fault models in case of memory are coupling faults. So, what is the coupling fault, coupling fault may basically because of the cells are wide near to each other. So, the memory if you look at, the cells are like this, the cells are like this row 1, row 2 and these are the memory cells we call it (Refer Time: 26:06) length.

So, these are very very close, these cells are very very close to each other. So, therefore, one cell having a value of 1 will have an impact on the other side having a value 0 just like maybe this as a value of 1, this as a 0, so I let me just take it a different example. So, coupling means one neighbour as all the neighbours are very very near to each other, they have a they have a habit of influencing the near neighbour cells.

(Refer Slide Time: 26:28)



Coupling Faults:

Coupling fault, as the name suggests, implies deviation from normal behavior of a cell because of coupling with others.

As there can be exponential number of combinations of coupling of a cell with others cells, we assume that in coupling faults a faulty cell can get coupled with another faulty cell.

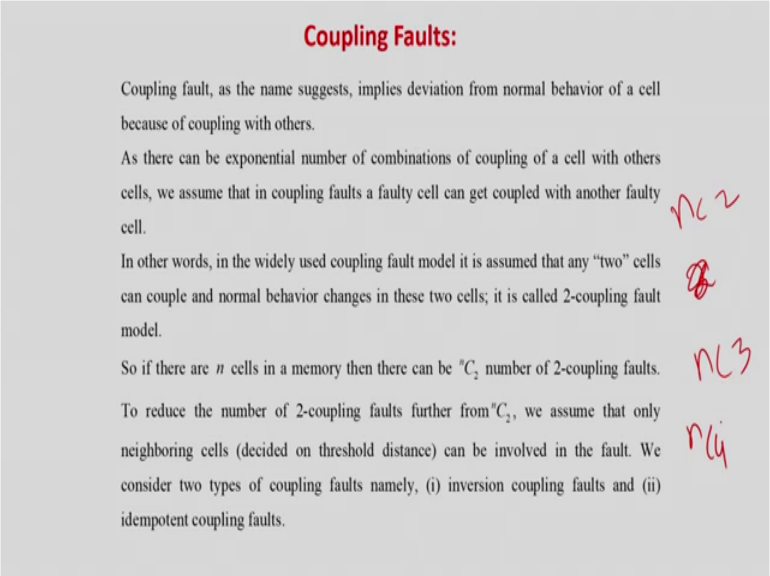
In other words, in the widely used coupling fault model it is assumed that any "two" cells can couple and normal behavior changes in these two cells; it is called 2-coupling fault model.

So if there are n cells in a memory then there can be ${}^nC_2$ number of 2-coupling faults.

To reduce the number of 2-coupling faults further from ${}^nC_2$, we assume that only neighboring cells (decided on threshold distance) can be involved in the fault. We consider two types of coupling faults namely, (i) inversion coupling faults and (ii) idempotent coupling faults.

Like, for example, if I have a memory cell like this, maybe these all are having 0's all are having 0, and this poor guy is having a 1. So, what it can happen, all the other neighbours can pull it to 0 that is from 1, it can pull it to 0, there is a coupling effect. So, all the neighbourhood actually try to have an impact on this, so that is actually called a coupling effect. So, exact examples will take, so but that is the gist that basically in the memory everything is very much nearby all the cells are close, so different nearby cells try to have a coupling or have a means a collusion effect in this case. So, now if you can say what is the number of such cases huge.

(Refer Slide Time: 27:03)



Coupling Faults:

Coupling fault, as the name suggests, implies deviation from normal behavior of a cell because of coupling with others.

As there can be exponential number of combinations of coupling of a cell with others cells, we assume that in coupling faults a faulty cell can get coupled with another faulty cell.

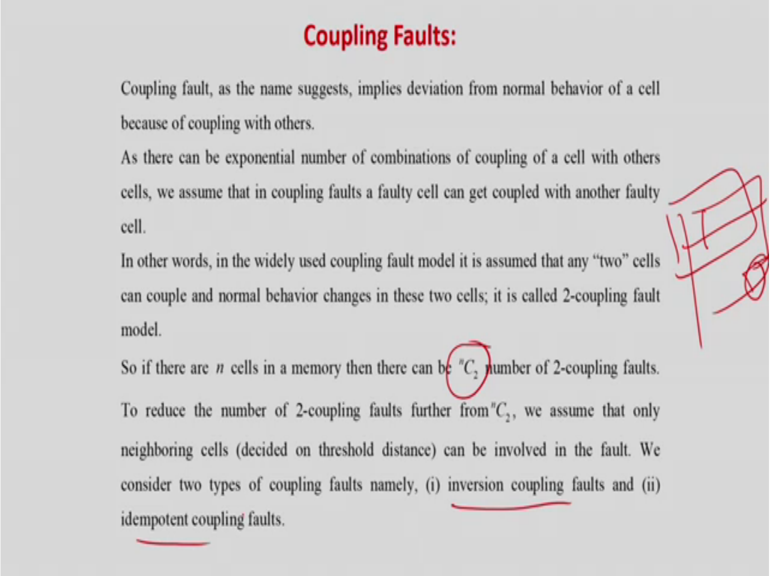
In other words, in the widely used coupling fault model it is assumed that any "two" cells can couple and normal behavior changes in these two cells; it is called 2-coupling fault model.

So if there are n cells in a memory then there can be ${}^nC_2$ number of 2-coupling faults.

To reduce the number of 2-coupling faults further from ${}^nC_2$, we assume that only neighboring cells (decided on threshold distance) can be involved in the fault. We consider two types of coupling faults namely, (i) inversion coupling faults and (ii) idempotent coupling faults.

So, if we assume that two cells can (Refer Time: 27:03) so it will be n C 2. We can say three cells may (Refer Time: 27:07) n C 3; four cells can (Refer Time: 27:09) n C 4. So, it is the (Refer Time: 27:11) number of fault can be possible. So, therefore, handling that is impossible that mean number of faults again.

(Refer Slide Time: 27:24)



Coupling Faults:

Coupling fault, as the name suggests, implies deviation from normal behavior of a cell because of coupling with others.

As there can be exponential number of combinations of coupling of a cell with others cells, we assume that in coupling faults a faulty cell can get coupled with another faulty cell.

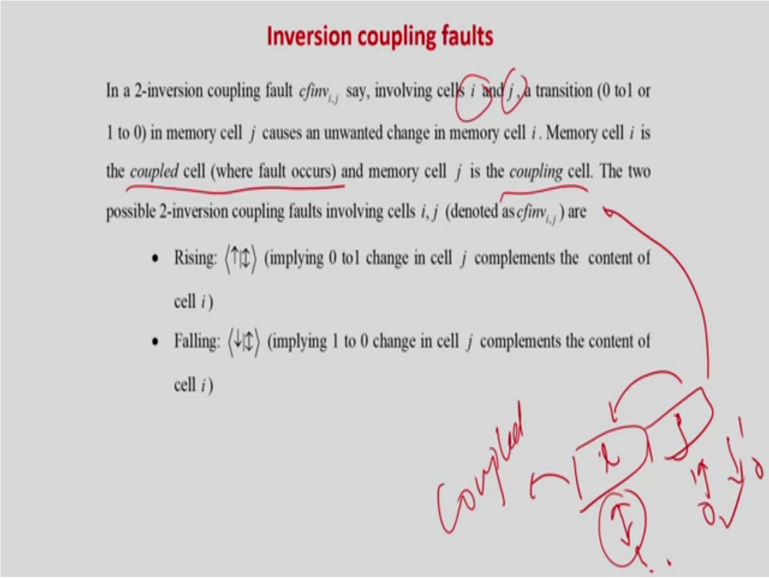
In other words, in the widely used coupling fault model it is assumed that any "two" cells can couple and normal behavior changes in these two cells; it is called 2-coupling fault model.

So if there are n cells in a memory then there can be $^nC_2$ number of 2-coupling faults.

To reduce the number of 2-coupling faults further from $^nC_2$, we assume that only neighboring cells (decided on threshold distance) can be involved in the fault. We consider two types of coupling faults namely, (i) inversion coupling faults and (ii) idempotent coupling faults.

We always sometimes go for a compromise at a little cost of test penalty, so we assume that the coupling can be 2, and that also be nearby cells. So, basically what that any two cells can have a coupling (Refer Time: 27:31) but of course a memory cell here and a memory cell here cannot have a coupling. So, you define a neighbourhood and in that neighbourhood any two neighbouring cells can have a company, so that is what is the idea. So, there are two type of coupling faults; basically inversion coupling fault and idempotent coupling faults as we will see.

(Refer Slide Time: 27:49)



Inversion coupling faults

In a 2-inversion coupling fault $cfinv_{i,j}$ say, involving cells i and j, a transition (0 to1 or 1 to 0) in memory cell j causes an unwanted change in memory cell i. Memory cell i is the *coupled* cell (where fault occurs) and memory cell j is the *coupling* cell. The two possible 2-inversion coupling faults involving cells i, j (denoted as $cfinv_{i,j}$) are

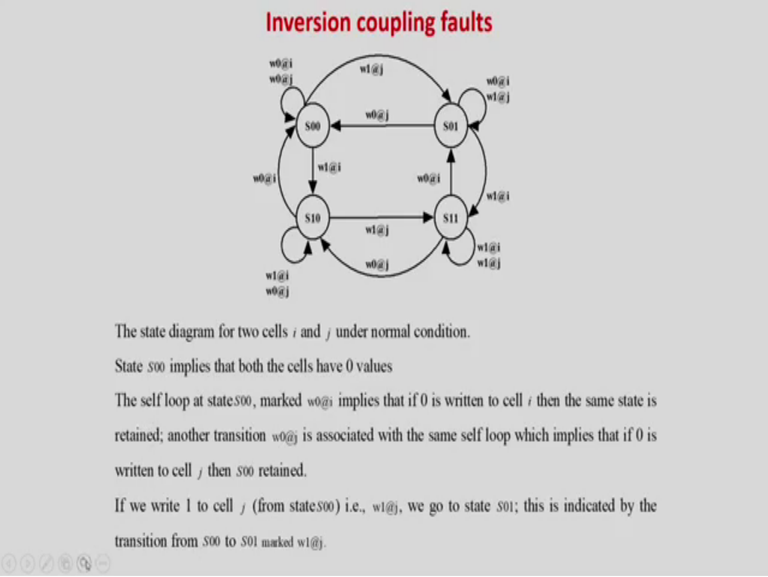
- Rising: $\langle\uparrow|\updownarrow\rangle$ (implying 0 to1 change in cell j complements the content of cell i)
- Falling: $\langle\downarrow|\updownarrow\rangle$ (implying 1 to 0 change in cell j complements the content of cell i)

So, inversion coupling faults. So, there are two memory cells; so one memory cell let me call it i and other neighboring memory cell I am calling it j. So, in a two and will take only two at a time. So, what it is saying in an inversion coupling fault two cells say i and j (Refer Time: 28:04) a transition from 0 to 1 or 1 to 0 that means, in the memory cell j either a transition happens like this or a transition happens like this, causes a unwanted transition here. So, here you are making either 0 to 1 or 1 to 0 will make a undesired transition here, these are all undesired. So, this is actually having an impact on this.

In memory cell j, you are making a up transition or down transition, but unnecessarily transition actually happening over here. This is making a unnecessary transition over here. So, cell i is unnecessarily having a problem, because there is a transition at j. So, therefore, the terminology is called as i is called the coupled cell i is called the coupled cell, where the fault occurs (Refer Time: 28:49) getting is the victim and j is the coupling cell. This is actually the coupling cell. So, in the coupling cell, some kind of transition has happened, and actually it leads to a unnecessary transition, this is a desired transition. So, unnecessarily some transitions are happening in the coupled cell. So, coupled cell is the victim cell.

So, there two types of such kind of faults; so one case is that. So, basically our 0 to 1 transition in the cells here that is the coupling cell will lead to a change in the cell i. And one is the falling coupling cell in which case the cell j, there is a 0 to 1 cell if j and actually it affected, very simple. Two cells are there; one is that you make a rising transition, this cell will be disturbed; and (Refer Time: 29:29) make a down transition, this cell will be disturbed. So, these are the two models.

(Refer Slide Time: 29:34)



Inversion coupling faults

The state diagram for two cells i and j under normal condition.

State $S00$ implies that both the cells have 0 values

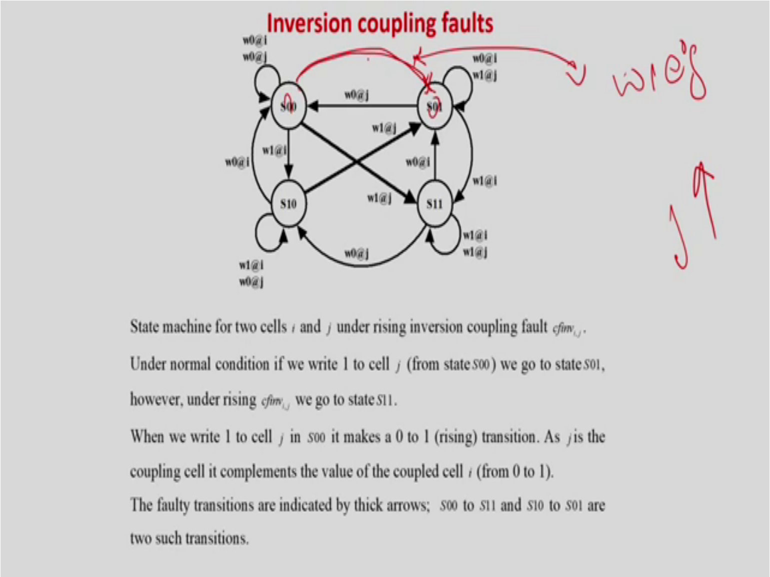
The self loop at state$S00$, marked w0@i implies that if 0 is written to cell i then the same state is retained; another transition w0@j is associated with the same self loop which implies that if 0 is written to cell j then $S00$ retained.

If we write 1 to cell j (from state$S00$) i.e., w1@j, we go to state $S01$; this is indicated by the transition from $S00$ to $S01$ marked w1@j.

Again, I am going to look at the state diagram, because we represent it. So, this is a normal system. So, what is the normal system? We are having two cells, this is for cell this first little bit is for i and this is for j, so S00 means i equal to 0, j equal to 0. So, if I write a equal to i equal to 0, j equal to 0, both I am writing (Refer Time: 29:53).

If I go from here to here, what I write w equal to 1 add j, so j will be becoming 1. So, here from here to here what is that, I am writing 0 at the level of j, so I am going over there. Similarly, you can explain the whole diagram. Like for example, S10 means i equal to 1, j equal to 0, so I write a 1 at i, so I come over here, very easily you can understand the diagram. So, it is a two cell system, and this is the normal basically normal state flow diagram.

(Refer Slide Time: 30:22)



Inversion coupling faults

State machine for two cells i and j under rising inversion coupling fault $cfinv_{i,j}$.

Under normal condition if we write 1 to cell j (from state $S00$) we go to state $S01$, however, under rising $cfinv_{i,j}$ we go to state $S11$.

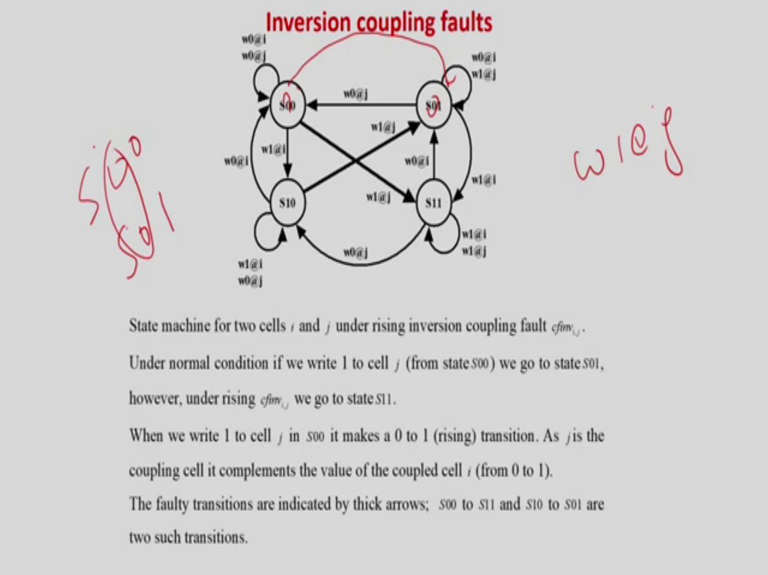
When we write 1 to cell j in $S00$ it makes a 0 to 1 (rising) transition. As j is the coupling cell it complements the value of the coupled cell i (from 0 to 1).

The faulty transitions are indicated by thick arrows; $S00$ to $S11$ and $S10$ to $S01$ are two such transitions.

Now, I will take a fault example, and you can understand. So, this is a fault cell. In this case, basically two state machines i and j as I have explained, and what is the fault over here. So, basically the fault in this case, basically is I will let us try to look at from here, so these are the two thick lines. If you compare this, so if I go over here, if you look at it, so these are the two transitions, which will be missing, which will be changed over here. So, what are the faults over here. So, this is no longer available in this fault model. Let us see what is the changes.

So, in this case you see, I want to write a w 1, in this case if you look at I want to write 1 at j, so I am writing a 1 at j, so it is writing a 1 at j. So, that means what? At j, I am making a up transition, because you are changing this 0 sorry, you are changing this 0 to this one. So, I should have a transition like this sorry. So, I should have a transition like this right, which is actually making w 1 at the rate of j, s, and but I should be ineffective. So, basically S this one should remain 0, this one should remain 0. But, as the fault has occurred, unnecessarily this S is i is also getting a transition. So, this is the unnecessary transition. So, this edge will no longer be there; in fault, this edge will no longer be there, basically you will have a system like this. So, what I am doing you see again, so this is the fault transition.

(Refer Slide Time: 31:52)



Inversion coupling faults

State machine for two cells i and j under rising inversion coupling fault $cfinv_{i,j}$.

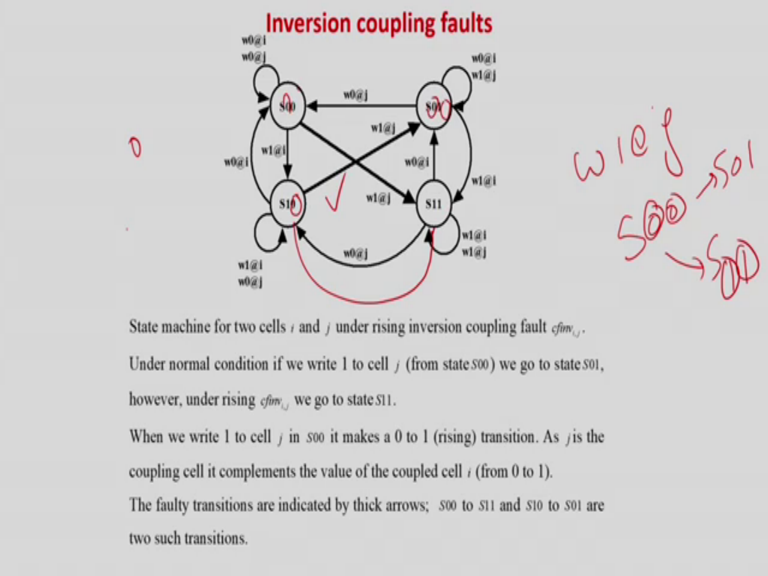
Under normal condition if we write 1 to cell j (from state $S00$) we go to state $S01$, however, under rising $cfinv_{i,j}$ we go to state $S11$.

When we write 1 to cell j in $S00$ it makes a 0 to 1 (rising) transition. As j is the coupling cell it complements the value of the coupled cell i (from 0 to 1).

The faulty transitions are indicated by thick arrows; $S00$ to $S11$ and $S10$ to $S01$ are two such transitions.

So, if you look at what was happening, I was intending a transition from here to here. So, what was I was doing, I was writing w 1 at the rate of j, so that is fine, I should have a transition like this. But, S0 should remain as S0. So, it should S from S00, it should remain go to S01 this one. So, the 0 should not change, but because of the coupling fault, change in j will also result to unnecessarily changing in the x.

(Refer Slide Time: 32:11)



Inversion coupling faults

State machine for two cells i and j under rising inversion coupling fault $cfinv_{i,j}$.

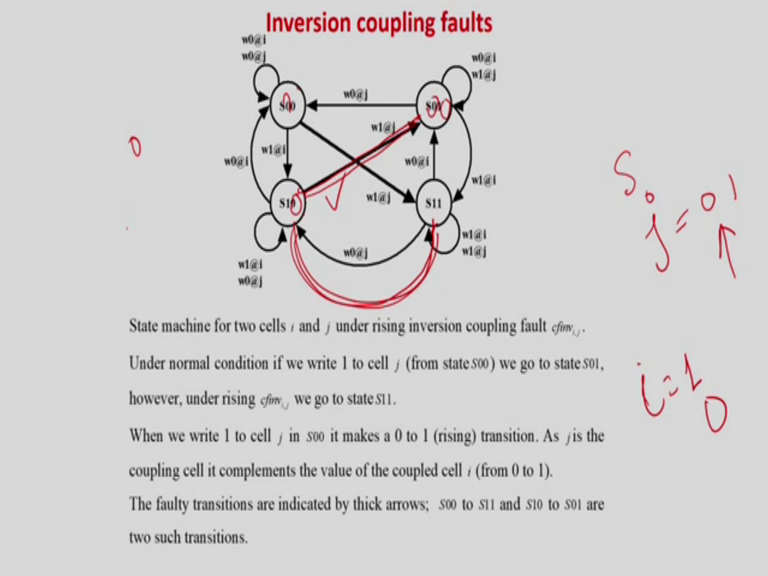
Under normal condition if we write 1 to cell j (from state $S00$) we go to state $S01$, however, under rising $cfinv_{i,j}$ we go to state $S11$.

When we write 1 to cell j in $S00$ it makes a 0 to 1 (rising) transition. As j is the coupling cell it complements the value of the coupled cell i (from 0 to 1).

The faulty transitions are indicated by thick arrows; $S00$ to $S11$ and $S10$ to $S01$ are two such transitions.

So, instead of going from S00 to S01, it is going to S11. So, this is a desired transition change. And I am changing the value of j from 0 to 1, but this is the undesired coupling undesired coupling, it is actually changing the value f x unnecessarily. So, therefore, this thick line actually this thick line is showing this case. Similarly, if you just look at the other case also, I am changing the value of y, the z is j from 0 to 1 fine, this is what I am going to do. So, I am going to change w 1 and this one I should have come over here; that is I am changing the value of j to 0 to 1 and S1 should remain as S1, because i I am not going to change. I this cell i is having the value of 1, I am not going to change it. But again, you do a fault what happening basically, I am writing something like this.

(Refer Slide Time: 33:01)



Inversion coupling faults

State machine for two cells i and j under rising inversion coupling fault $\mathit{cfinv}_{i,j}$.

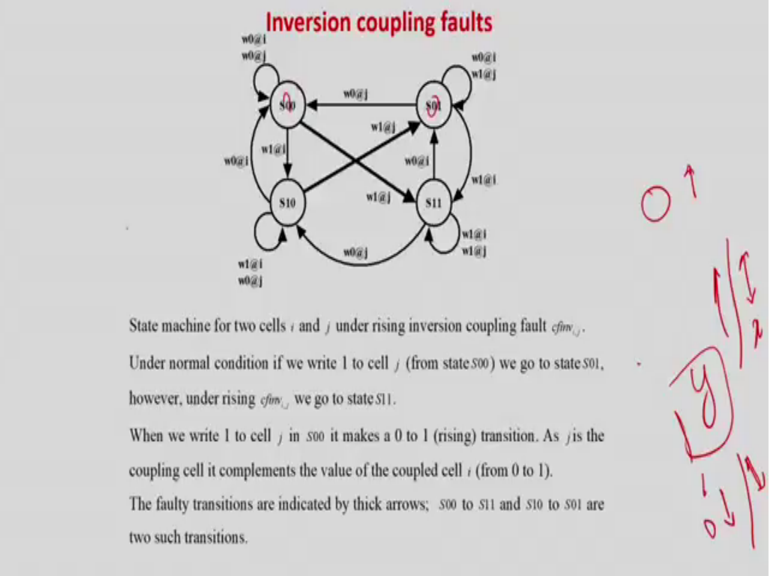
Under normal condition if we write 1 to cell j (from state $s00$) we go to state $s01$, however, under rising $\mathit{cfinv}_{i,j}$ we go to state $s11$.

When we write 1 to cell j in $s00$ it makes a 0 to 1 (rising) transition. As j is the coupling cell it complements the value of the coupled cell i (from 0 to 1).

The faulty transitions are indicated by thick arrows; $s00$ to $s11$ and $s10$ to $s01$ are two such transitions.

So, what I am trying to do here is, I am trying to make cell j, cell j I am trying to make it from 0 to 1, then I am going to give a rise. Now, what happens basically. So, I am trying to go by this. And S cell i is having the value of 1 for the time being that is what is this showing in state 1, 0. Now, because of the flip, i will become a 0, so basically here it is going this way S01. So, instead of this one going from going to this state is basically going to this state. So, again basically i is the so, i is going to get a flip.

(Refer Slide Time: 33:33)



Inversion coupling faults

State machine for two cells i and j under rising inversion coupling fault $cfinv_{i,j}$.

Under normal condition if we write 1 to cell j (from state $S00$) we go to state $S01$, however, under rising $cfinv_{i,j}$ we go to state $S11$.

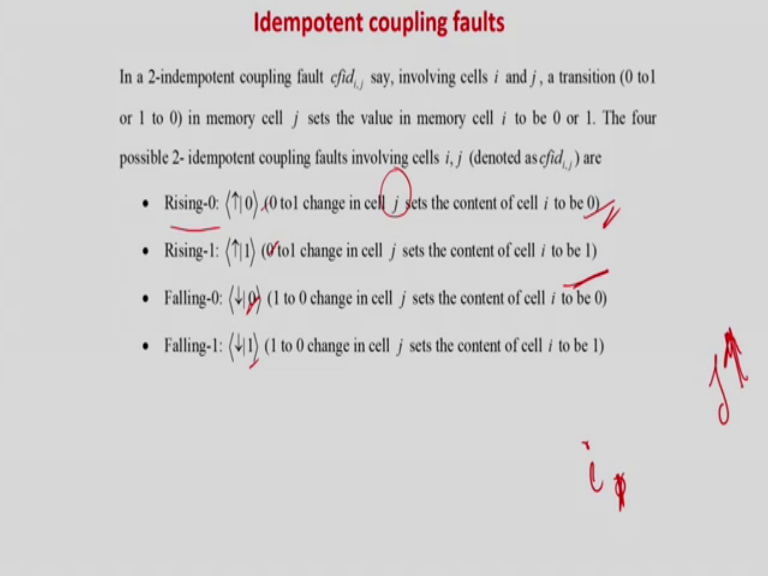
When we write 1 to cell j in $S00$ it makes a 0 to 1 (rising) transition. As j is the coupling cell it complements the value of the coupled cell i (from 0 to 1).

The faulty transitions are indicated by thick arrows; $S00$ to $S11$ and $S10$ to $S01$ are two such transitions.

So, this fault model tells that in the coupling cell that is j if I am making our transition from 0 to 1 that is this is the coupling cell I am making this, the value of x is getting changed. So, this is the cell y, basically this is what is being written over here. I am making a rising in j, so I am making a rising in cell j, and x is getting a decoupling. So, very similarly I can again draw the diagram for the dual part of it in which case if I may make y as down from 0 to 1 to 0, so they will be again a coupling change in the x l, so that diagram you can take as a example.

So, what is fault model say? This fault model says that in the coupled cell is a fault. What is the fault; it gets a jitter, If it changes (Refer Time: 34:14) when it should not. Because, I am just changing the value of cell j that is the coupling cell from 0 to 1 or 1 to 0 that is my desired change, but it has a impact on the coupled cell i, it changes it value unnecessarily. So, this is the normal model, this is the normal model. And in the fault model as you can see, these are the two transitions which will be gone, and there will be (Refer Time: 34:33) transition. So, this is actually inverse coupling fault model. So, again as I told you, so in case of coupling there are two type of faults; one is the inverse coupling fault, and other is the idempotent fault very similar. So, in that case what happens?

(Refer Slide Time: 34:47)



Idempotent coupling faults

In a 2-indempotent coupling fault $cfid_{i,j}$ say, involving cells i and j, a transition (0 to1 or 1 to 0) in memory cell j sets the value in memory cell i to be 0 or 1. The four possible 2- idempotent coupling faults involving cells i, j (denoted as $cfid_{i,j}$) are

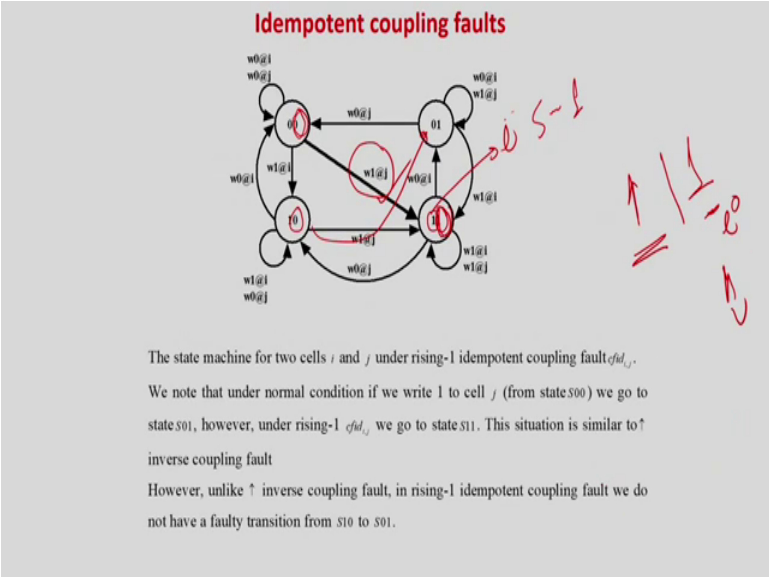
- Rising-0: $\langle\uparrow|0\rangle$ (0 to1 change in cell j sets the content of cell i to be 0)
- Rising-1: $\langle\uparrow|1\rangle$ (0 to1 change in cell j sets the content of cell i to be 1)
- Falling-0: $\langle\downarrow|0\rangle$ (1 to 0 change in cell j sets the content of cell i to be 0)
- Falling-1: $\langle\downarrow|1\rangle$ (1 to 0 change in cell j sets the content of cell i to be 1)

So, in case of this inversion coupling fault, what you have seen. So, if there is a rising or a falling, basically the others couple cell jitters. But, in this case, it is a stuck. So, in this case what is seeing, rising 0 that means, if the cell j changes from 0 to 1, the content of cell will be stuck-at 0 right. Similarly, rising-1 means, if I make a change from 0 to 1, but then the contents of the cell will be set at 1 stuck-at 1. Similarly, I can make for falling also. So, if the cell j is falling from 0 to 1, it is stuck 0; falling 1, it is stuck-at 1.

So, similar inversion and idempotent coupling faults are very simple similar. Inversion means whenever there is a change in the values in the coupling cell, the other cell gets a jitter, it changes that is the inversion. And in case of idempotent, if there is a change in the coupling cell, the other part gets stuck-at 0 or stuck-at 1.

So, there are four different combinations; rising and falling. So, rising stuck-at 0, rising stuck-at 1, fault falling stuck-at 0, fall stuck-at 1. So, in fact, that is what, cell j is there and cell i is there. So, if I making a rising at cell j, this one will become stuck-at 0, so this corresponds to this. And if I make a rising at the cell 0, if it becomes stuck-at 1, it will be stuck-at 1 rising stuck-at rising-1. Similarly, I can also repeat the same thing for basically falling 1, so that is it (Refer Time: 36:11) I know fault model.

(Refer Slide Time: 36:14)



Idempotent coupling faults

The state machine for two cells i and j under rising-1 idempotent coupling fault $cfid_{i,j}$.

We note that under normal condition if we write 1 to cell j (from state $s00$) we go to state $s01$, however, under rising-1 $cfid_{i,j}$ we go to state $s11$. This situation is similar to↑ inverse coupling fault

However, unlike ↑ inverse coupling fault, in rising-1 idempotent coupling fault we do not have a faulty transition from $s10$ to $s01$.

So, again if you look at it, so that is actually one type of this again explains basically another type of idempotent fault model in which case if you look at this bold arrow, it represents a failure. So, in this case, what happens? I am trying to basically write this one from 0 to 1 that the last bit if you look like that is j cell, I am trying to make it from 0 to 1, but what happens basically, the cell x becomes stuck-at 1. So, in fact, this is basically rising-1, this is the idempotent rising-1 fault.

So, in this case, I am changing this one from 0 to 1, so this cell i becomes stuck-at 1 right. But, in this case if you see, basically another cell if I am trying to if you look at this one, so in this case what I am trying to do, I am again taking a rising value from 0 to 1. But, in this case unlike the inversion fault, we do not have a fault transition like this. In this in this case it still remains the same, because it is a stuck-at 1, so it remains stuck-at 1. This is what I am trying to say is that.

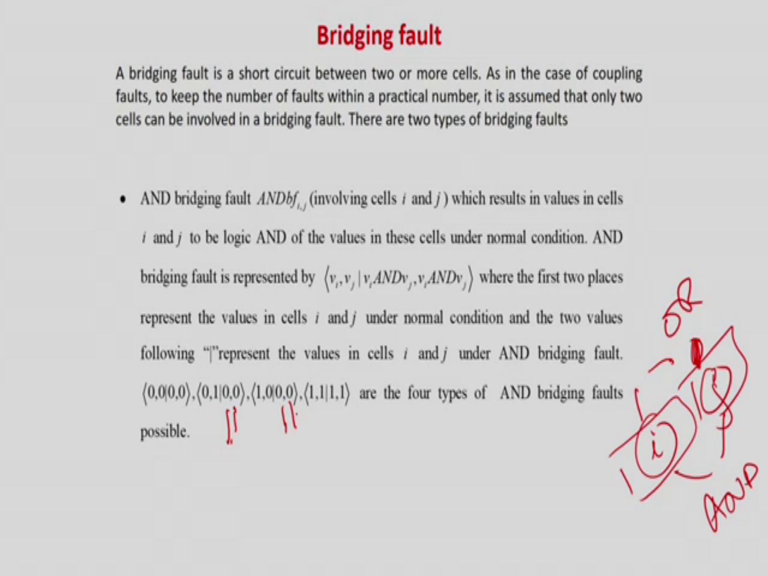
If you look at this inversion coupling faults, this fault is reflected by this transition as well as (Refer Time: 37:14) transition. But, in case of an idempotent fault, only one such fault effect is there, because what is the difference. Because, in case of idempotent fault idempotent rising fault, the state the cell i is having a stuck-at fault, so therefore that is reflected by this. But, in case of a inverse coupling fault, there will be a jitter.

So, jitter means, if it is a 0 to become 1 and vice versa. So, in if it is the basically if, in this diagram, basically if it becomes a inversion coupling fault, so this row will actually go over here as reflected in this slide. But, in this case that is the idempotent fault, not the basically your inversion fault, so there is only that means, there is only one transition, which will capture the fault. I just very simple just look at these slides, it will actually clear out all the fact.

So, basically what I am trying to show here is that there are different type of fault models which has been proposed to look at memory faults. Like as I told you stuck-at is the very basic one, it is not going to suffice all the faults, even in case of a normal combinational circuit, if the designs are more sophisticated, will even in the next module will have dedicated lectures on (Refer Time: 38:18).

In case of memory, basically the memory cells are so congested and coupled with near to each other, so the problem is more difficult. So, apart from stuck-at fault, different different types of fault models are coming into picture. Like, as I told you stuck-at faults, idempotent faults, coupling faults, and some for few more we are going to look at. The fault model (Refer Time: 38:38) very correlated to each other, but more number of fault model means more number of different test patterns will be there, and the circuits will be on the memory will be tested more rigorously.

(Refer Slide Time: 38:54)



Bridging fault

A bridging fault is a short circuit between two or more cells. As in the case of coupling faults, to keep the number of faults within a practical number, it is assumed that only two cells can be involved in a bridging fault. There are two types of bridging faults

- AND bridging fault $ANDbf_{i,j}$ (involving cells i and j) which results in values in cells i and j to be logic AND of the values in these cells under normal condition. AND bridging fault is represented by $\langle v_i, v_j \mid v_i AND v_j, v_i AND v_j \rangle$ where the first two places represent the values in cells i and j under normal condition and the two values following "|"represent the values in cells i and j under AND bridging fault. $\langle 0,0|0,0 \rangle, \langle 0,1|0,0 \rangle, \langle 1,0|0,0 \rangle, \langle 1,1|1,1 \rangle$ are the four types of AND bridging faults possible.

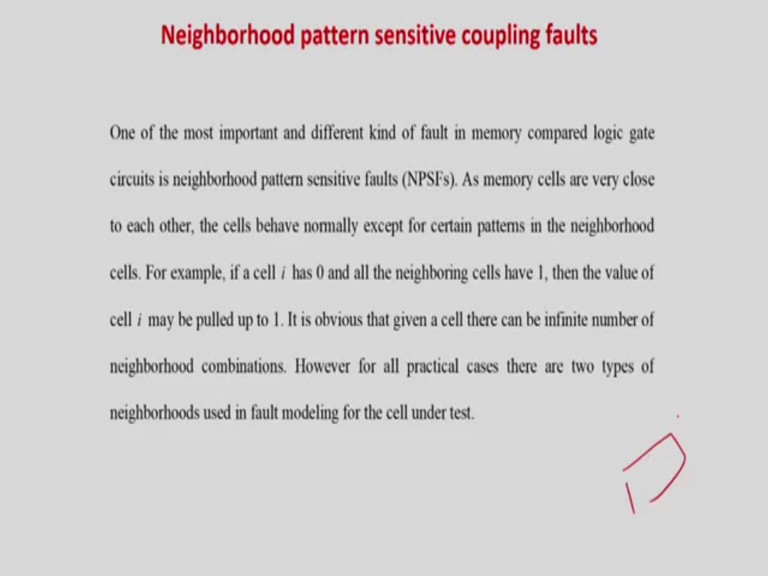
And then important one is the bridging fault. Again, very similar one thing I should tell you is that as the memory faults are very much coupled with each other, so two cells like i and j. We are always concerned about neighbouring cells. neighbouring testing is more and more related to neighbouring cells, two cells near to each other having a coupling effect. So, we have to see what are the different type of coupling effect for possible, and most fault models are based on this theory. So, one we have seen is, this one we have seen an idempotent and inversion. So, here it is logical bridging fault.

Bridging fault means basically these two cells will have a AND effect or they can have OR effect. Like, for example, if you are writing a 0 over here and a 1 over here, and if the fault is the OR fault, so this one will actually make this one as a 1. Similarly, if this a if I write a 1 and a 0 over here and if it is a AND bridging fault, so this one will become a 0, so that means, two cells i and j having a bridging fault means, they will be AND logic or OR logic will come into picture. AND logic means if both of them are 0 means if it is a AND logic means, basically both of the same. In bridging faults, basically I should tell you that both of these cells we are going to have the same value. Now, what same value, depending on the basically the logical AND or OR fault, which we were considering.

Like, in bridging fault, if both of them are 0, 0 or 1, 1, there is no effect. But, AND fault means basically like 1, 1, there is no change, 0, 0 also no change. But, if it is a 0 and it is a 1, and say AND fault means this will also become a 0, similarly vice versa. Like, but it is a OR gate or the coupling fault is there or bridging or bridging fault is there means, if it is 0, 1 means, this guy will make it 1. Similarly, basically if it is some something like that is 1 and a 0 or OR coupling faults means, this one also make as a 1 by virtue of this.

Again repeating bridging fault in case of memory means both these cells will have a same fault. What will be the value of the value will be depending on this one and this one, and the logical operator (Refer Time: 40:40) so that is basically very simple. So, these are the four cases possible written in this manner 0,0 0, 0; 1, 1 1, 1, no problem; only problem is with this 0, 1, 0, this is a AND bridging, 1, 0 0, 0 these are actually AND bridging fault example. In case of OR bridging fault, it will be very simple like 1, 1 and this one will be again 1, 1 that is a AND bridging fault, so both of them will have a same value. So, this is what is the OR bridging fault case, as I was discussing. So, these are the four cases of a OR bridging fault, the very simple one.

(Refer Slide Time: 41:10)

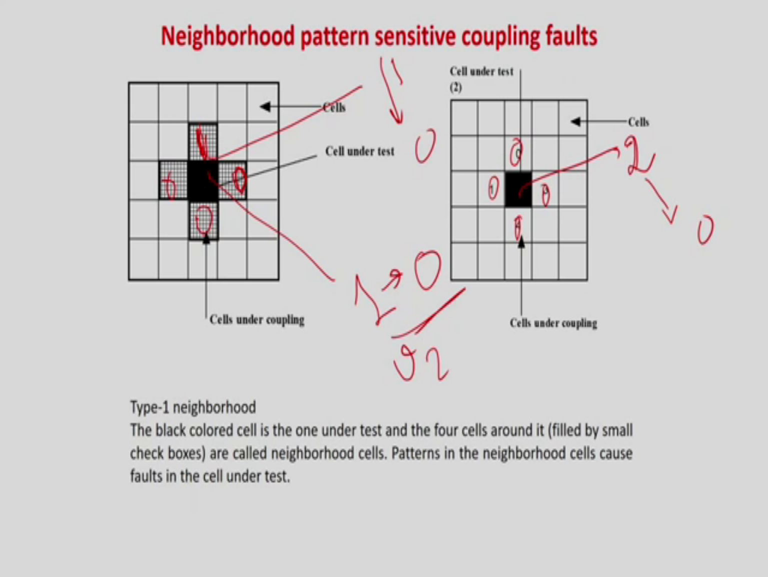


Neighborhood pattern sensitive coupling faults

One of the most important and different kind of fault in memory compared logic gate circuits is neighborhood pattern sensitive faults (NPSFs). As memory cells are very close to each other, the cells behave normally except for certain patterns in the neighborhood cells. For example, if a cell *i* has 0 and all the neighboring cells have 1, then the value of cell *i* may be pulled up to 1. It is obvious that given a cell there can be infinite number of neighborhood combinations. However for all practical cases there are two types of neighborhoods used in fault modeling for the cell under test.

Now, very important is very very important, now actually we are discussing the fault models in a very pedagogical manner that what happened after another. Then we will try to find out that not only two neighbourring cells, which i and j, so only two.

(Refer Slide Time: 41:26)

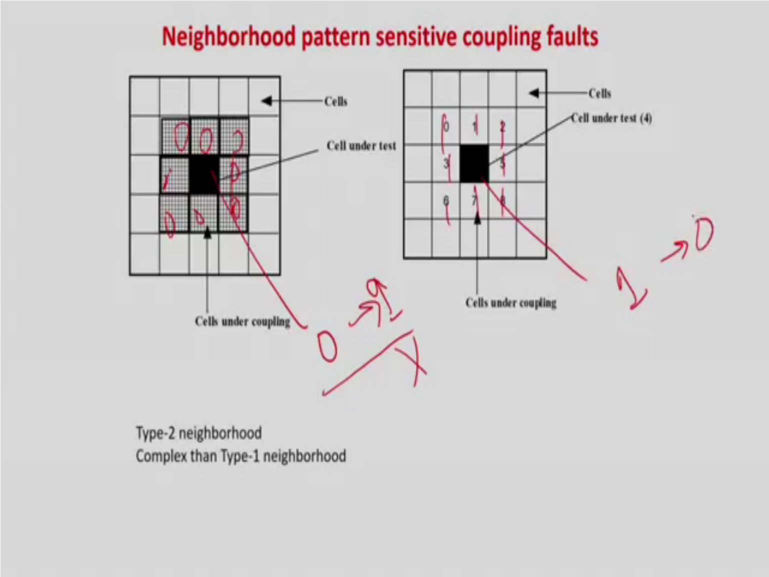


Neighborhood pattern sensitive coupling faults

Type-1 neighborhood
The black colored cell is the one under test and the four cells around it (filled by small check boxes) are called neighborhood cells. Patterns in the neighborhood cells cause faults in the cell under test.

But, basically if you look at like this is what is a memory, I will take a figure that is better, this is what is a memory cell look like. So, i, j model means you are going to take this i this j means pair wise right, you are taking pair wise like this is i, j. Similarly, again these two will be a pair, these two will be a pair, so all pair wise will do you are considering. Then people (Refer Time: 41:41) found out that this is actually becoming more normally realastic, because neighboring means something like this.

(Refer Slide Time: 41:50)



Neighborhood pattern sensitive coupling faults

Type-2 neighborhood
Complex than Type-1 neighborhood

This is a cell, these are this neighbor, or more sophisticatedly, this is a cell, and these are basically the neighbors. So, we in case of this paring sense what we do is that we take two pair, because three pair, four pair, five pairs are very very complicated in nature very difficult to do. But, still people have found out that if you just take the n C 2 combinations and moving window of (Refer Time: 42:08) the faults models may not be very accurate that means, (Refer Time: 42:11) there can be lot of defects, which mean a we may go undetected.

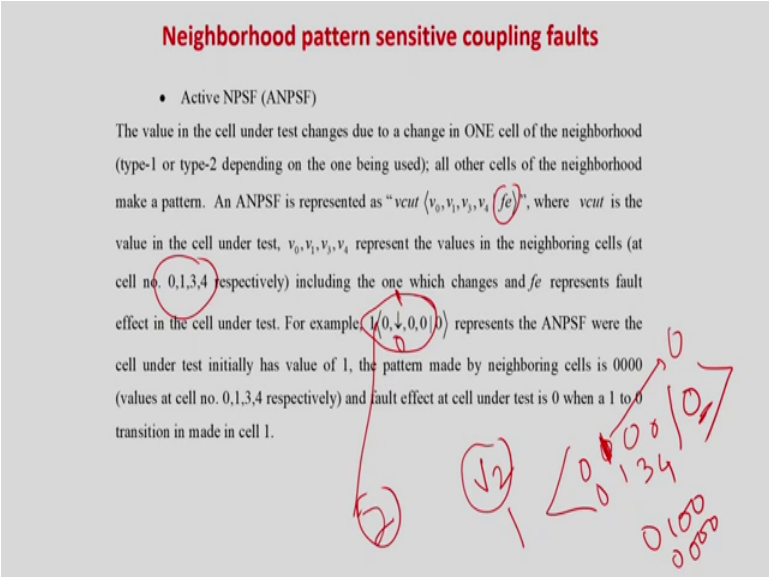
So, people thought of proposing something like a real neighbourhood. So, this is one type of a neighbourhood that is this is the cell number 2, which is under test, 1, 2, 3, 4 and 0; 0, 1, 2, 3, 4. So, basically this 2 is under test. So, this is one short of neighbourhood, this is called type 1 neighbourhood. Type 2 neighbourhood slightly more complicated like 0, 1, 2 then 3, then (Refer Time: 42:37) sorry 0, 1, 2, 3, 4, 5, 6, 7, 8, it will be numbering. Basically these are the neighbouring cells, and this is basically the cell under test.

So, this as you can you can easily understand that coupling faults at this model will be more more realistic, because all these neighbouring cells is going to have a cell on this, (Refer Time: 42:54) and this will be a moving window. Means will first have some something like this, then will be having something like this will be a moving window kind of a picture. So, it is slightly more complicated in nature compared to n C 2, but is also not completely like n C 2, n C 3, n C 4 and so forth, not as complicated at that, but more complicated than n C 2 business. But, it actually gives a more realistic fault capturing or defect capturing behaviour. So, now let us look at it.

So, they are called to neighbourhood sensitive coupling faults. So, basically in what case what it tells basically, so we are going to consider such neighbourhoods. So, it says one of the most important different kind of fault in memory compared to logic gate is pattern sensitive faults. That means what? Basically maybe this one is having the value of 0, so all the others are having the value of 0, then there is no problem. But, if it happens that all 0s I have, and this poor guy is a 1, so all together will pull it down from 1 to 0. So, such type off basically pattern sensitive faults happen in case of memory, so that is what it is saying. So, they are called neighbourhood pattern sensitive faults.

As memory cells are very close to each another, cells behave normally except for certain patterns in the neighbourhood. Like for example, if a cell i has 0 and all neighbour cells has 1 or vice versa, then they may be pulled up. So, for all practical cases, two type of neighbourhood fault models are considered. So, in that case, as I was saying that basically this if all 0's the center one is a 1 and vice versa, so this is a pulling effect. So, people have thought of two different type of neighbourhoods; one is type 1, and basically one is type 2. And among them, you try to find out different type of sensitive pattern based faults.

(Refer Slide Time: 44:28)



Neighborhood pattern sensitive coupling faults

- Active NPSF (ANPSF)

The value in the cell under test changes due to a change in ONE cell of the neighborhood (type-1 or type-2 depending on the one being used); all other cells of the neighborhood make a pattern. An ANPSF is represented as "$vcut \langle v_0, v_1, v_3, v_4 \mid fe \rangle$", where $vcut$ is the value in the cell under test, v_0, v_1, v_3, v_4 represent the values in the neighboring cells (at cell no. 0,1,3,4 respectively) including the one which changes and fe represents fault effect in the cell under test. For example, $1\langle 0, \downarrow, 0, 0 \mid 0 \rangle$ represents the ANPSF were the cell under test initially has value of 1, the pattern made by neighboring cells is 0000 (values at cell no. 0,1,3,4 respectively) and fault effect at cell under test is 0 when a 1 to 0 transition in made in cell 1.

So, there are two type of sensitive neighborhood faults. Like one is called activate networks neighborhood pattern sensitive faults, and that is called the passive. So, what is the formal way of writing it as he as you have look at this is his neighborhood type-1. So, neighborhood type one the cell number two is basically under test, this is the cell number 2. And 0, 1, 3 and 4 are basically the neighboring cells, which actually try to damage the cell number 2.

So, how do you write it, we write it v 0, v 1, v 3, v 4. They are basically the cells, we are trying to damage the culprit the compromised cell, so the fe represents the compromised cell. So, this is actually the value of 1, 2, 3 and 4, and this value is really is the compromised cell. So, here what it is saying that for example, it says that say for example, all the neighboring cells have the value of 0 like something like 0 following means from 1 to 0. This is a fault and 0, 0.

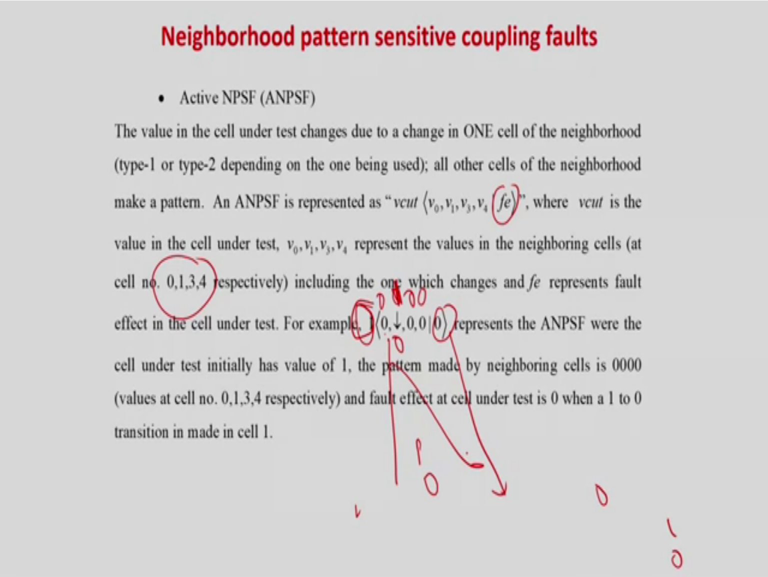
And this one is actually the initial value of cell number 2. So, there are what is written, it is written like 0 0 0 0 cell right, and maybe this is what is the class. So, this is the safe cell number 0 sorry the cell number 0, cell number 1, cell number 3, and cell number 4, all I having a value of 0. And the initial value of cell number v 2 is equal to 1. Now, what happens basically there is a fault. So, sorry this was having the value of 1 sorry. So, this was cell number 1 was having the value of 1. Now, there is a fault. So, the fault says that it will be actually becoming 0 that means, from the cells from 0 1 0 0, it was becoming 0

0 0 0. And then basically what happens with v 2 was 1, but it is coming to 0. So, it is actually that is bringing down the value of culprits victimized cell from 1 to 0.

Let us see in the figure that is better. So, what is it saying say that this poor guy has a value of 1, and maybe this all these all these were having the value of 0. This was having the value of 1, this was having the value of 0, this was also having the value of 0. But, this culprit or this victim cell v 2 was still sustaining the value of 1, because one neighborhood was at least having the value of 1.

Now, these are transition, this one and the said at one of the neighboring cells. So, this one will be coming from 1 to 0. Now, what happens as all the neighboring would serve cells have become 0, this victim cell that is v 2 actually goes to 0 that is not desired, so that is what is actually called a kind of a neighborhood active a pattern and even in a network, neighborhood pattern sensitive fault that is has been represented in this manner.

(Refer Slide Time: 46:59)



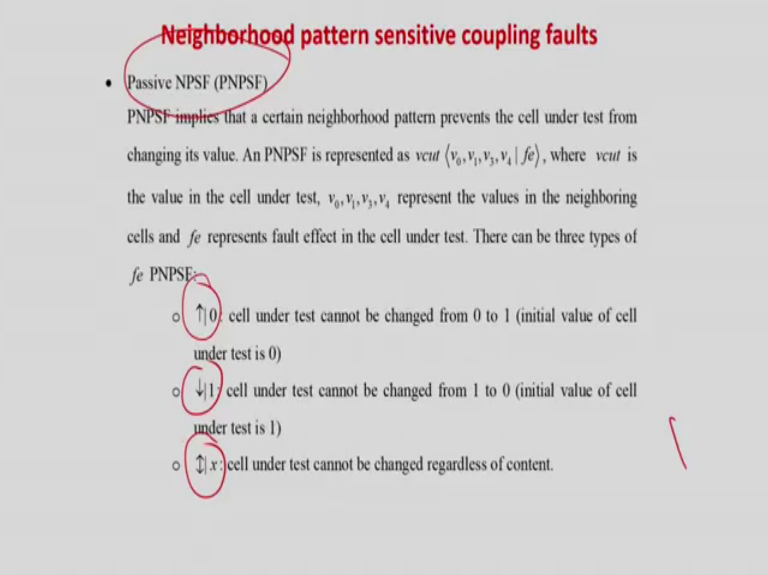
Neighborhood pattern sensitive coupling faults

- Active NPSF (ANPSF)

The value in the cell under test changes due to a change in ONE cell of the neighborhood (type-1 or type-2 depending on the one being used); all other cells of the neighborhood make a pattern. An ANPSF is represented as "$vcut \langle v_0, v_1, v_3, v_4 | fe \rangle$", where $vcut$ is the value in the cell under test, v_0, v_1, v_3, v_4 represent the values in the neighboring cells (at cell no. 0,1,3,4 respectively) including the one which changes and fe represents fault effect in the cell under test. For example, $1 \langle 0, \downarrow, 0, 0 | 0 \rangle$ represents the ANPSF were the cell under test initially has value of 1, the pattern made by neighboring cells is 0000 (values at cell no. 0,1,3,4 respectively) and fault effect at cell under test is 0 when a 1 to 0 transition in made in cell 1.

That is what it says that basically this is what is the initial value of your cell number 2. And the initial value of cell all the other cells were 0 0 sorry it was 1 0 0. Then what happens basically, it gets change it is remains same it gets changed from 1 to 0, so all the neighboring cells has become 0. So, finally v 2 has changed from 1 to 0. So, therefore basically it leads to a defect or it leads to a fault representation in the crump culprit or the victim cell that is actually cell number 2.

(Refer Slide Time: 47:29)



Neighborhood pattern sensitive coupling faults

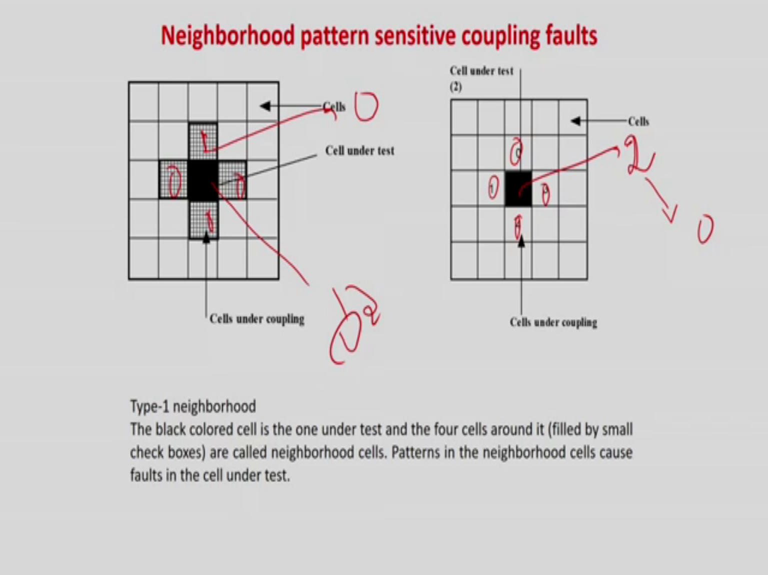
- Passive NPSF (PNPSF)

PNPSF implies that a certain neighborhood pattern prevents the cell under test from changing its value. An PNPSF is represented as $vcut\langle v_0,v_1,v_3,v_4 \mid fe\rangle$, where $vcut$ is the value in the cell under test, v_0,v_1,v_3,v_4 represent the values in the neighboring cells and fe represents fault effect in the cell under test. There can be three types of fe PNPSF:

- ↑|0: cell under test cannot be changed from 0 to 1 (initial value of cell under test is 0)
- ↓|1: cell under test cannot be changed from 1 to 0 (initial value of cell under test is 1)
- ⇕|x: cell under test cannot be changed regardless of content.

So, that is one example of a active NPSF, it will be more clear, when I tell you about the passive.

(Refer Slide Time: 47:45)



Neighborhood pattern sensitive coupling faults

Type-1 neighborhood
The black colored cell is the one under test and the four cells around it (filled by small check boxes) are called neighborhood cells. Patterns in the neighborhood cells cause faults in the cell under test.

So, active means what? Active basically means something like some value it was having some well as I have told you, it has it has some value of it was having a some value of 1, and all other values were 0, and it was having a value of 1. Now, this guy is changing the value from 1 to 0 says all neighbors are becoming 0, so by error this also becoming 0.

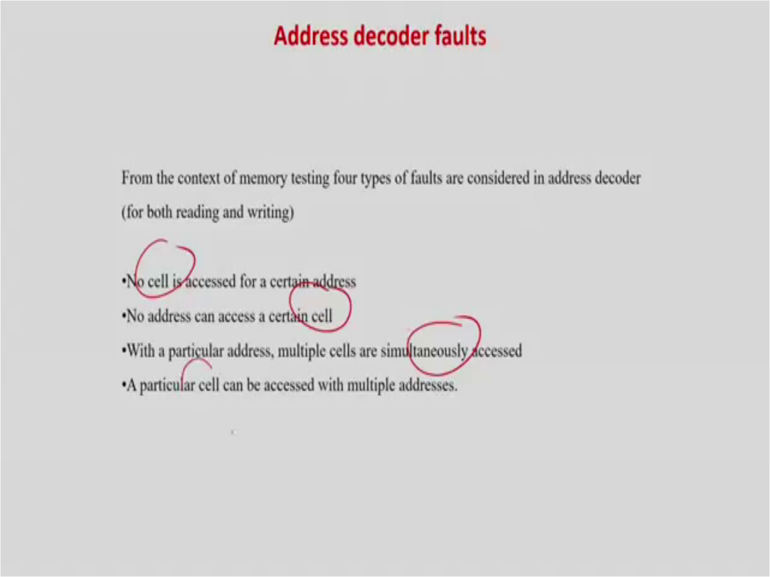
So, there is some dynamic change in the neighborhood. So, the victim is getting penalized, so that actually says something like active PSF.

But, there is also another fault model, which is called passive NPSF in which case what it says, right there are different types like this one. The cell under test cannot be changed from 0 to 1 right. What did he say? That is due to some effect; due to some effect may be due to some of the neighborhood values, maybe all these values are all the neighborhood values are having sorry I can take this example also.

So, let us I mean assume that all the cells are having the values of 0's. So, in this case this was a 0, I want to make it as a 1. So, this is not possible, so this actually called static that means, there is no change in the neighborhood. Only thing is that because all the neighborhood cells are having a value 0, you cannot make it from 0 to 1, so that is what is called the static. So, you cannot change the value of the victim cell from rising, you cannot make it rising its stuck-at 0.

Similarly, it cannot be changed from 1 to 0, so this is also possible. If it is something like all the neighborhood's are having the value of 1 1 1 1 1 1, you cannot make it fall from 1 to 0 that is also may not be possible. (Refer Time: 48:57) risk static effect is possible. Also sometimes it can happen that because of some condition, you cannot change the value of some set that is also possible. So, this is actually called the passing NSF fault model.

(Refer Slide Time: 49:13)



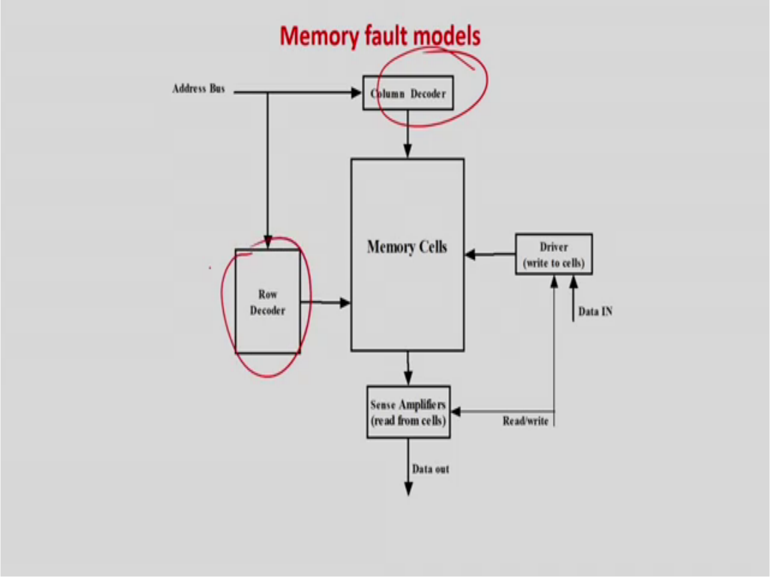
Address decoder faults

From the context of memory testing four types of faults are considered in address decoder (for both reading and writing)

- No cell is accessed for a certain address
- No address can access a certain cell
- With a particular address, multiple cells are simultaneously accessed
- A particular cell can be accessed with multiple addresses.

So, basically as we and this is one of the (Refer Time: 49:10) latest fault model, which people try to adopt right. So, these are different types of also consist of address decoder faults like means the problems in the addressing.

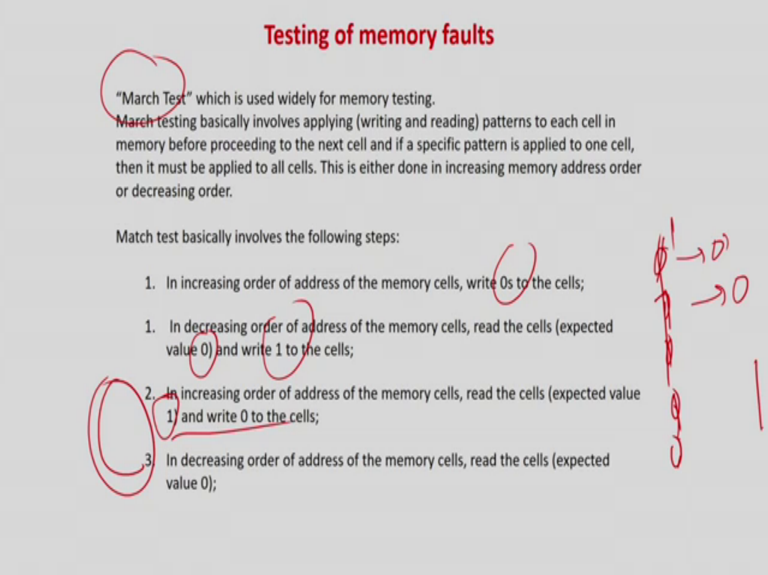
(Refer Slide Time: 49:26)



That is as I have shown in the basic structure if you look at it, so in the basic structure if you see, we have something called a column decoder and row decoder. So, these are some of the fault. This can also these can have some kind of faults. So, what type of faults are possible, it is very simple no cell is accessed for a certain address. No address can access a certain cell that is some cells have become non-accessible. With a particular address, multiple cells are addressed.

With a particular cell can access multiple address that means, these are some of the fault models, which is problem with the address decoding that means, you try to access one cell, other cell is accessed. Some cell you can never access that means, some of these address can never be generated by the address decoder that is the problem of the decoders. So, we have so there are different type of this type of fault models in the decoder.

(Refer Slide Time: 50:04)



Testing of memory faults

"March Test" which is used widely for memory testing.
March testing basically involves applying (writing and reading) patterns to each cell in memory before proceeding to the next cell and if a specific pattern is applied to one cell, then it must be applied to all cells. This is either done in increasing memory address order or decreasing order.

Match test basically involves the following steps:

1. In increasing order of address of the memory cells, write 0s to the cells;

1. In decreasing order of address of the memory cells, read the cells (expected value 0) and write 1 to the cells;

2. In increasing order of address of the memory cells, read the cells (expected value 1) and write 0 to the cells;

3. In decreasing order of address of the memory cells, read the cells (expected value 0);

So, now what we have till now what we have seen. So, what we have seen is till now, different type of fault models in the memory stuck-at, transition, coupling, neighbor base coupling and so many fault model. Also in the decoder I very quickly tell you about told you about the different type of fault models that is because, address decoders are more or less like combinational circuit, so I am not emphasized more.

But, what the fault models there, then you are having problem in addressing. You need to address cell 0, you are addressing cell 1. You can never address cell 2 address, cell 2 can never be addressed, because your decoder is not generating address 2 and so forth. So, what I built? So, can I place all the fault models like which I have done in circuit fault, the answer is no. If you want to do this, the n number of different fault models are actually possible for a memory.

So, like fault, stuck, transition and different type of memory consideration never consideration etcetera. So, should I target test patterns, generate test patterns for everything and taste it the idea is no. If you want to do this, the takes time will take huge. So, what people do is something very interesting called a march test that is very logical as I was telling in the beginning, that I will write 0 0 0 0 0 0 in all these cells, I will again read all these 0 0 0 0 in all these cells, again I will write 1 1 1 1 all these uneventually.

Now, what I do? Now, I will try to see, how many fault models have covered. So, again repeating below this is the mode most important flux of memory test, you can tell me there is a very simple procedure or I would write all 0, read all 0, write almost 0 almost, then why are you telling me about so many fault models which are unnecessary, in fact no. I am not going to make this pattern generation for each of the fault models, unlike I do in a combination of sequences. It rather what I will do rather what people have done rather that is they are going for something called a march test that is what I have told you.

Increasing order of address rise 0 to the cells, decreasing order you read the cell value 0, it should be 0. And then instead of just that is you write 0 0 0 0 read all 0s, while you are reading all the 0s, (Refer Time: 51:59) start writing 1s there. And then again basically increasing order of these cells, read the cell values which would be 1, write 0 and basically again read the 0, and your job is done that is what is March test. That is write all 0s, read all 0s, while you are reading on the write all 1s, next you read all the 1s, write all 0s and do it.

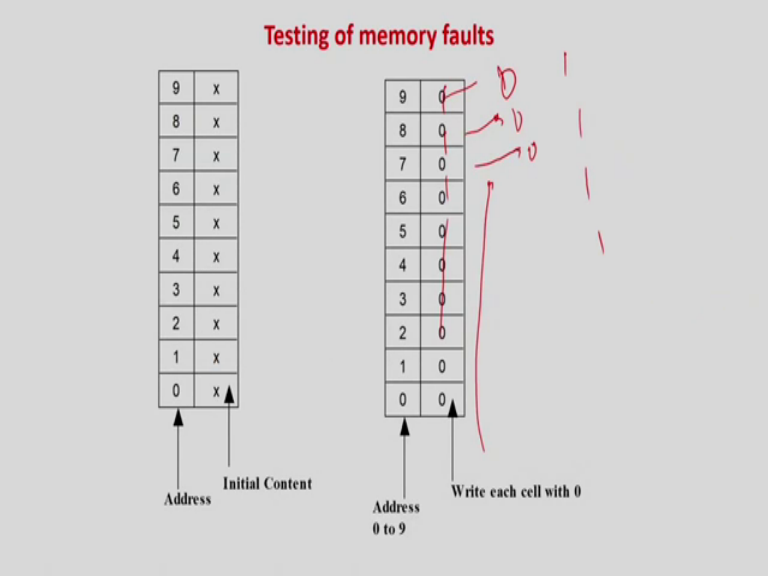
So, basically what we are going to achieve out of this that means, if you look at it I could have stopped here, so what would I stopped over here that is actually I am writing 0 0 0 0 0, reading all 0s. And basically after reading the 0, I make it 1, again reading this 0, I am making into the 1 1 1 1. And then after that I could have read all 1s and my job would have been done.

But, again I am not doing it, I am making another more step basically again when I am reading the 1, again I am repeating again, I am writing 0, and doing it, why? Basically, these two steps will also help me to study for the transition as we will see. So, the idea is that March test is very simple, these four steps. Write all 0s, read all 0s, at a time write 1, again read all 1s, write all 0s, read that all these 0s, this March test.

Now, people have shown that this test basically this March test covers most of the important part that is what is the idea. So, fault models may be found out by a different set of engineers, March test was found by different set of engineers. People have found out that March test covered most of this important formulas that is why both of these things are very important, and we should study both of them together.

Because, it is not like in a circuit what we would have done, we would have found out all the fault models, then we have done the ATPG for all the different fault model, I have studied the coverage. In case of memory, we are doing the other way round. We have found out the March test is very very logical, because memory means you want to read 0, write 0, read 0 and do it for the one same we are doing it. But, then we are trying to analyse what we fault models we have covered, and we have found out that March test does pretty well.

(Refer Slide Time: 53:44)



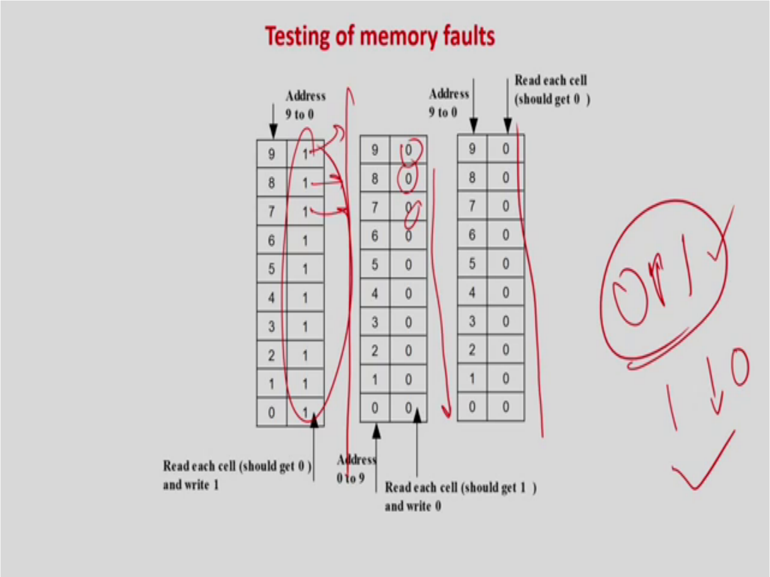
Testing of memory faults

9	x
8	x
7	x
6	x
5	x
4	x
3	x
2	x
1	x
0	x

9	0
8	0
7	0
6	0
5	0
4	0
3	0
2	0
1	0
0	0

Now, we are going to see how March test does pretty well for all the fault models, and very blocks. So, this is the first slide on the March test, so 10 memory locations all are having initially 0. So, initially I will go in this direction increasing order of address, and I will write all 0s over here.

(Refer Slide Time: 53:59)



Testing of memory faults

Address 9 to 0

Read each cell (should get 0) and write 1

Address 0 to 9

Read each cell (should get 1) and write 0

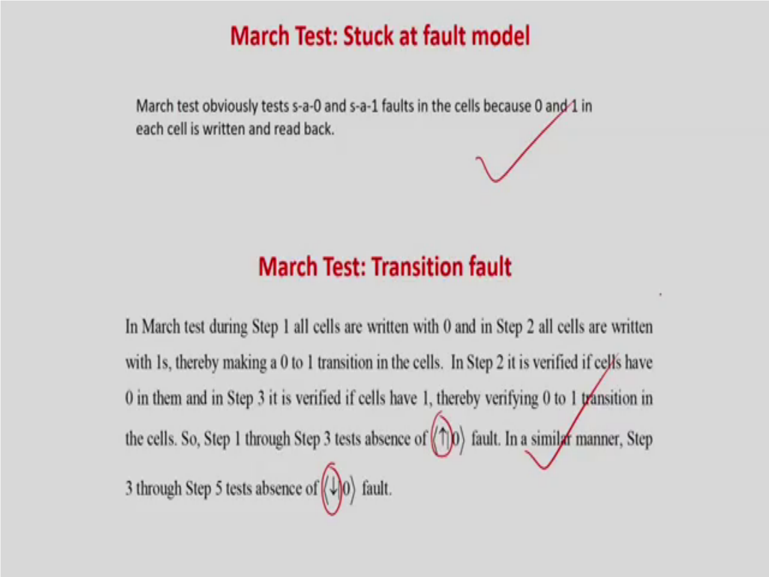
Address 9 to 0

Read each cell (should get 0)

Then basically what I will do? I will read all the 1s, read all the 0 that means, from here I will start reading 0s, 0 is read, 0 is read. When reading the 0, I will also start making all the values just 1s. So, these what has been done is cell is reading a 0, and then I am all writing the value of 1s. Then I am going to basically I have written a 1, written 0, read a 0, written a 1. Now, I have to read all the 1s back. So, what I am doing is that I am reading all the 1s here, and I could have stopped over here.

But, some additionally two steps I am doing for as we will see there will be helpful for transition fault detection. So, what I am doing is that 1 1 1 1 1 1 1 1 already I am reading over here, and just after reading a 1, now I am going to make it a 0 simultaneously. So, basically again repeating so all 0s are read, written with 1s right. Reading all the 1s and writing back all these 0s, and again reading back all the 0s (Refer Time: 54:50). In the 4th step I am not going to write the any values, it is done this my March test is complete.

(Refer Slide Time: 54:56)



March Test: Stuck at fault model

March test obviously tests s-a-0 and s-a-1 faults in the cells because 0 and 1 in each cell is written and read back.

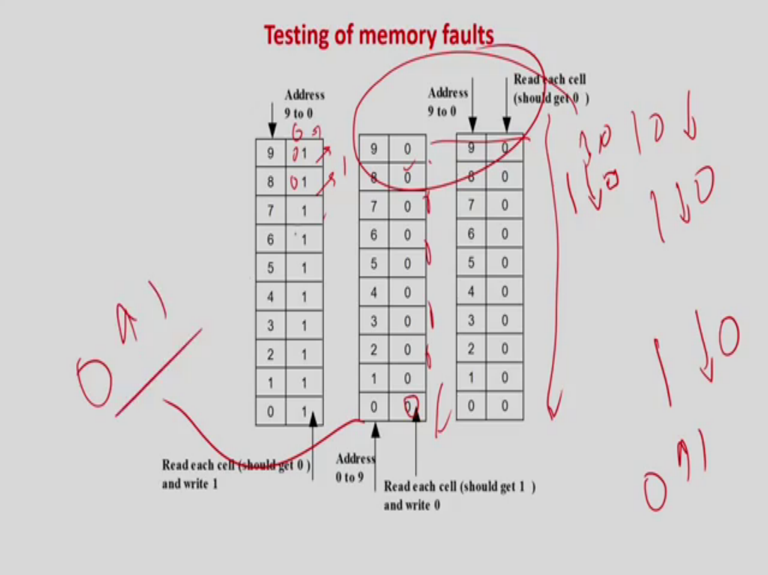
March Test: Transition fault

In March test during Step 1 all cells are written with 0 and in Step 2 all cells are written with 1s, thereby making a 0 to 1 transition in the cells. In Step 2 it is verified if cells have 0 in them and in Step 3 it is verified if cells have 1, thereby verifying 0 to 1 transition in the cells. So, Step 1 through Step 3 tests absence of $\langle \uparrow 0 \rangle$ fault. In a similar manner, Step 3 through Step 5 tests absence of $\langle \downarrow 0 \rangle$ fault.

Now, I have to find out, what is the capability of a March test. First of all very simply all stuck-at faults are covered, because you are this is a very obvious did not say again, because here you are reading sorry here you are reading writing all 0s, reading all 0s, writing all 1s, writing reading on 1s. So, no stuck-at faults, so all faults covered. Now, transition; so, what are the transition one means? Basically it should rise to 0, and it should fault both should be allowed. There is 0 to 1, and 1 to 0 should be allowed.

So, here what so here I am writing all 0s, here I am writing all 1s. So, basically here what I have done, I have written I have read or written all 0s, and then I am read all 0s, and writing all 1s, here I am reading all the 1s. So, 0 to 1, so this fault is not there, transition of 0 to 1 is not there. Now, what I have to do, now I have to study from 1 to 0 this I have to study. So, these are these two steps actually which is going to help me out. So, now I am have written read, I have written all the 1s, which is true. Now, I read a 1, and write a 0, read a 1, write a 0, read a 1, write a 0, I am (Refer Time: 56:03) this one. And then I am going to read all this 0, so this one; that is why, I told you that this step and this step basically are enough to actually deal with all stuck-at faults.

(Refer Slide Time: 56:19)



Testing of memory faults

Address 9 to 0

Address 9 to 0

Read each cell (should get 0)

Read each cell (should get 0) and write 1

Address 0 to 9

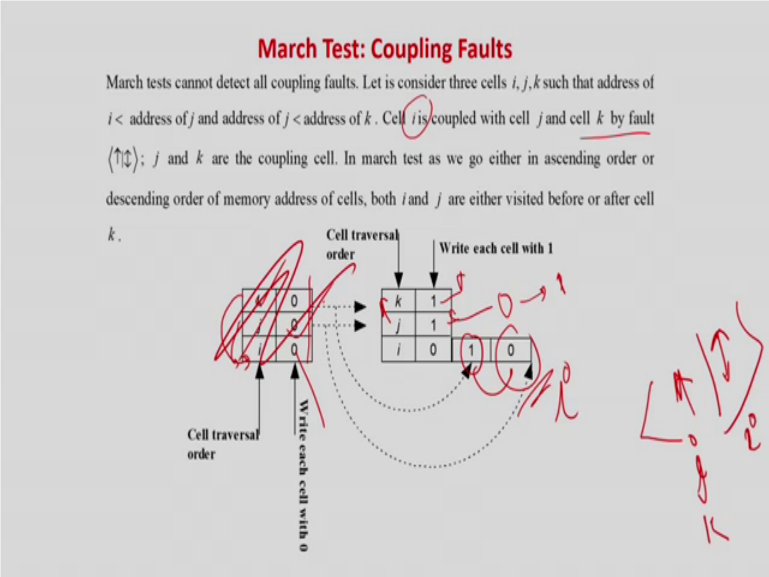
Read each cell (should get 1) and write 0

Two additional steps I am doing over here are to test the fault transition that is 1 to 0 should be possible, again just repeating it once more quick for your help. So, March test means what, I am writing all 0s over here. And in the next stage, I am reading all these 0s and writing a 1s, all 0s I am reading and writing a 1. So, stuck-at one fault is tested, because you could read 0, write 0, read 0.

At the same time, I am also going to write a 1 over here. And next day I am going to read all the 1s, and writing a 0. So, reading all the one successful here means, basically I am able to make a transition from 0 to 1. Similarly, here I am reading all the 1s and writing all these 0s over here, all these 0s are over here, and read 1 is successful. Now, from here I am going to read all the 0 successfully that means, basically from 1 to 0 also is possible. So, these two are testing the 1 to 0 transition, these two are from 1 to 0 transition sorry 1 to 0 transition is all tested by these two fact, and other facts are testing from 0 to 1. So, this is how basically test the transition fault model also.

Now, limitation; so, same way you can study that what is possible for the March test, and what is not possible for the March test. Like some of the fault models I have told you, like transition stuck-at fault done, transition fault I have already shown you. Coupling faults, you can study (Refer Time: 57:31) bridging all these you can study that whether it is possible or not.

(Refer Slide Time: 57:37)



March Test: Coupling Faults

March tests cannot detect all coupling faults. Let is consider three cells i, j, k such that address of $i <$ address of j and address of $j <$ address of k. Cell i is coupled with cell j and cell k by fault $\langle \uparrow | \updownarrow \rangle$; j and k are the coupling cell. In march test as we go either in ascending order or descending order of memory address of cells, both i and j are either visited before or after cell k.

But, I will give you an example of a coupling fault that some of the force of course is not possible in March test, you have to slightly tweaked march test. So, March test was a fundamental test pattern for testing your memory fault model. You know I found out that most of the fault walls are covered, but for some of the fault model slight weakening of the compensate this March test will be required.

I will give you an example, because if I elaborate too much on the March, because March test was the beginning of the ATPG for memory, after that lot of events have happened. So, those things are not possible to cover in a single lecture, rather I will just give you an problem that I have shown to successful cases. These are the two successful cases I have shown you, here I will show you one unsuccessful cases and how people have recovered I will tell you, then you can try to do as a homework by reading and (Refer Time: 58:21) applies to other fault models or not right.

So, let me just take these are simple three cells i, j, and k. And basically I am assuming that basically this was something this are these are this is the increasing order of address. Cell i is coupled with j and k that is this i is having a problem with both these cells that is k and j both are coupled with i that is i is coupled with j and k. Therefore, any problem with j and k will have an effect on i right.

Now, let us assume the fault model is coupled with j, where they fault this one. So, what is the fault? The fault model is rising there will be a transient i that means, if any rise transition happen in cell j or k, there will be a cell in there will be a simple change of values in cell i that is what if the fault model I have assumed. And I will show that this is not possible to be detected. Again repeating the fault model is any rise in the coupling cells like i, j and k will lead to a transition in the coupled cell i that is any rise in k or j 0 to 1 will make a flip of the values in self i.

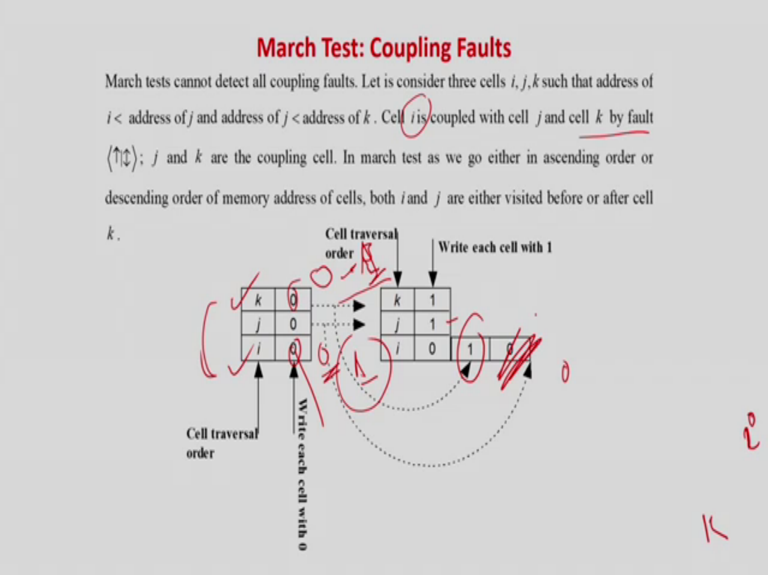
Let us say them, it can be detected or not by the March test and we will see that it is not possible. So, we are going to write the first if you are going to write all 0s, then you are going to read back all 0s. So, there will be no problem in this case, because this coupling fault only happens, when this cell j or k goes from 0 to 1. So, all 0s will read write, and we will read there will be no problem nothing will be taken.

Now, I am going to make a 1 and a 1, you know that this type of March test. We read all 0s, and then we write read all 0s still that there is no (Refer Time: 59:57) we will start writing a 1. So, whenever you start writing a 1 over here like this one, we are going to make a transition from k equal to 0 to 1. So, there will be rise over here. Whenever these arise over here, this one will be a flip. So, each one will from it will change from 0 to 1, I should have predicted fault.

If I would have immediately read i after key, but March test will not go there. March test says goes up, and then comes down. Then again it is going to do repeat it same for the j. So, it is again going to read a 0 fine, it is going to read a 0 very good, and it will make a 1 that is what is the March test. So, again our rise in j cell is happening, if again coupled with i, so again there will be a flip in this cell i, so again there will be a flip from 1 to 0, so again it is going to 0.

So, when you are going to read i, you are going to find out that I is having a 0, so there is no problem. But, in fact the fault was there, but it got masked. Why it got masked, because this cell is coupling with this as well as this cell having a coupling effect with this. So, once this fault made it go from 0 to 1, and this fall actually made it go from again 1 to 0, then the fault is not detected. So, this is one example which shows that March test cannot cover all coupling faults. Then how I can do it? Then basically slight changes in the sequence you have to do that is instead of just going to 0, 1, 2, 3, 4, 5 and again coming down.

(Refer Slide Time: 61:20)

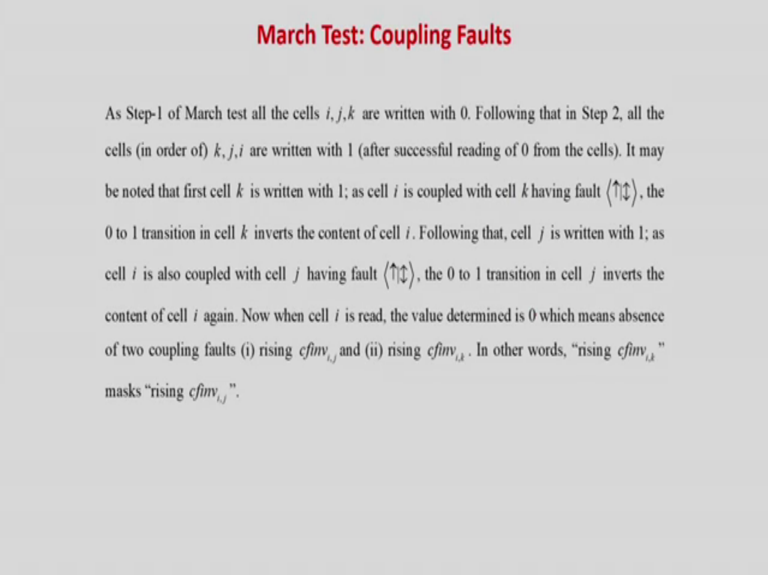


March Test: Coupling Faults

March tests cannot detect all coupling faults. Let is consider three cells i, j, k such that address of $i <$ address of j and address of $j <$ address of k. Cell i is coupled with cell j and cell k by fault $\langle \uparrow | \updownarrow \rangle$; j and k are the coupling cell. In march test as we go either in ascending order or descending order of memory address of cells, both i and j are either visited before or after cell k.

If I am checking for these two coupling, so basically you read this 0, you read write this read 0 fine no problem, then you write a 1, and then immediately you have to read the value of this one (Refer Time: 61:26) i. Basically, because we are assuming these two are coupled, you have to jump j, and you have read the value of i. So, in this case 0 to 1, so this one we flipped from 0 to 1. If you are directly going to read i, you are going to find out that this is an error this situation is happening, because you are actually evaluating cell j before i. So, much slightly changing the order of applying the patterns in March test, you can detect coupling faults.

(Refer Slide Time: 61:51)



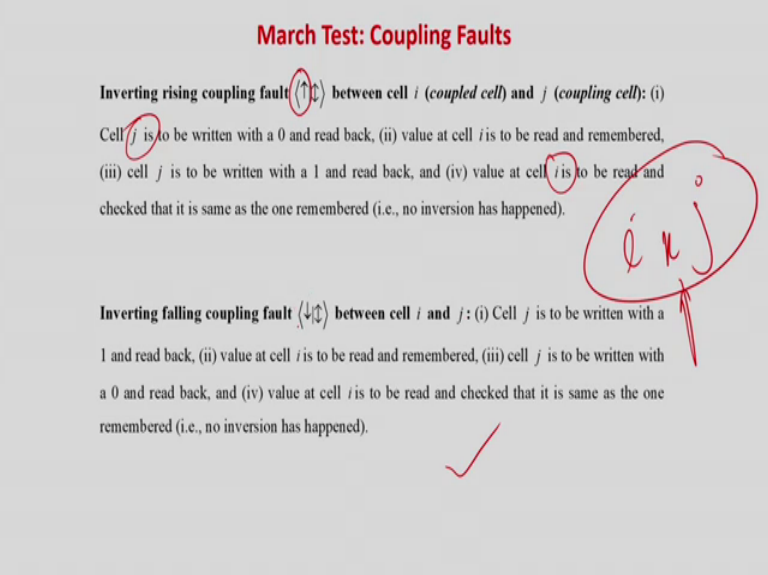
March Test: Coupling Faults

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 $cfinv_{i,j}$".

So, this is what is written in this slide you can read over it. But, what I am did this time going to try to tell is that this kind of March test mostly (Refer Time: 61:58) most of the (Refer Time: 61:58), but few (Refer Time: 61:59) cases of the fault models I have discussed, may not be covered by march test. But, we slightly the changing the pattern or the sequence of these cells to be applied to the pattern, we can test most of the fault models.

So, therefore march test or modified versions of March test are the de facto standard of testing memories, so that part is an assignment for you that you take all other fault models, find out what is possible by normal march test, what is not possible, and you may really look at advance papers are how people have modified March test to cover this model, more other faults model.

(Refer Slide Time: 62:31)



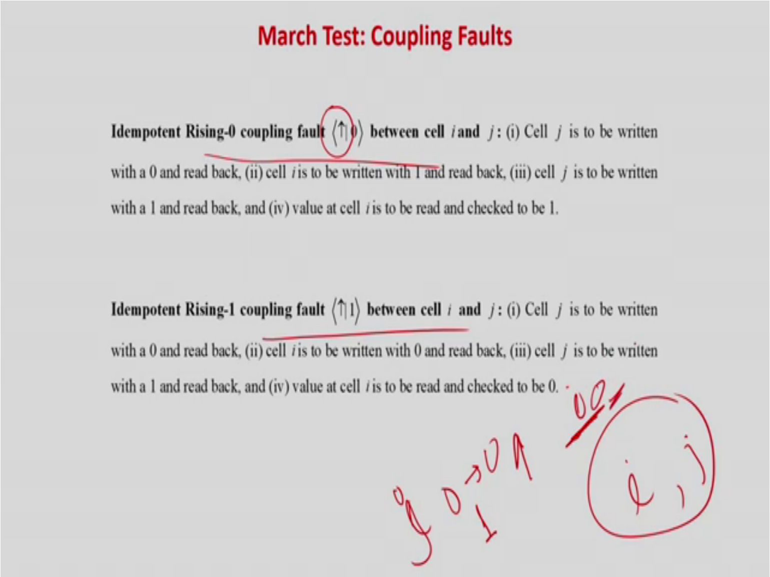
March Test: Coupling Faults

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).

Again some example actually, I have given you for coupling fault. So, there are different type of u you have you can find out that inverting rising coupling faults, inverting fault coupling fault.

(Refer Slide Time: 62:39)



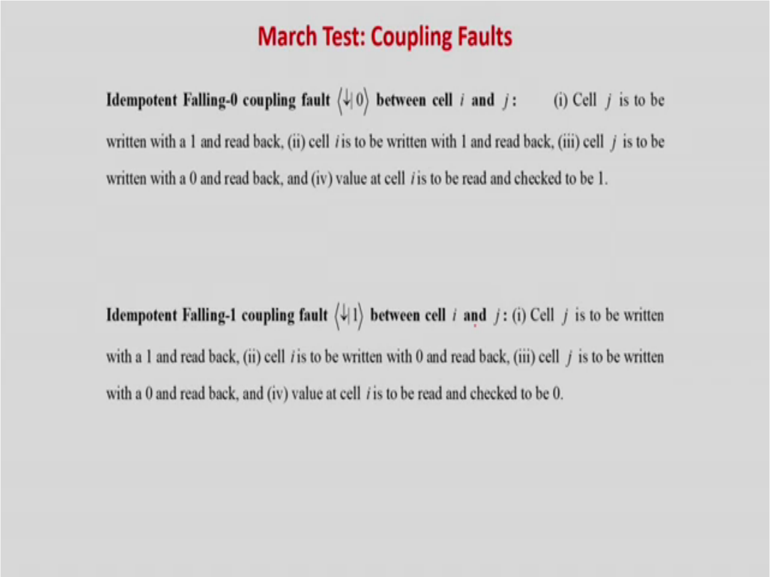
March Test: Coupling Faults

Idempotent Rising-0 coupling fault $\langle \uparrow | 0 \rangle$ **between cell** i **and** j**:** (i) Cell j is to be written with a 0 and read back, (ii) cell i is to be written with 1 and read back, (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 to be 1.

Idempotent Rising-1 coupling fault $\langle \uparrow | 1 \rangle$ **between cell** i **and** j**:** (i) Cell j is to be written with a 0 and read back, (ii) cell i is to be written with 0 and read back, (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 to be 0.

Idempotent rising, idempotent 0 that (Refer Time: 62:42) can be detected by March test right.

(Refer Slide Time: 62:44)



March Test: Coupling Faults

Idempotent Falling-0 coupling fault $\langle \downarrow | 0 \rangle$ **between cell** i **and** j**:** (i) Cell j is to be written with a 1 and read back, (ii) cell i is to be written with 1 and read back, (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 to be 1.

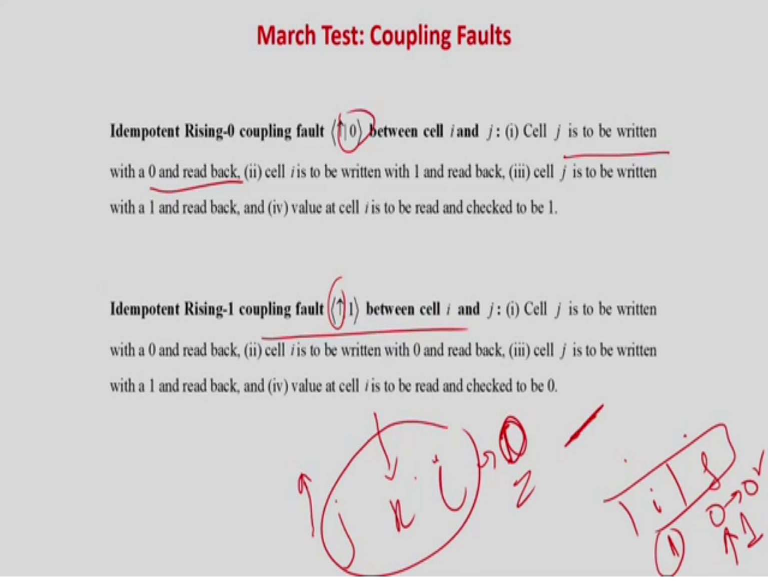
Idempotent Falling-1 coupling fault $\langle \downarrow | 1 \rangle$ **between cell** i **and** j**:** (i) Cell j is to be written with a 1 and read back, (ii) cell i is to be written with 0 and read back, (iii) cell j is to be written with a 0 and read back, and (iv) value at cell i is to be read and checked to be 0.

So, many I will just take one example, and I will show you. So, all type of fault models in these slides are being shown, how they can be deleted by the March test. Only thing is that if cell i and j are under the fault consideration, you have to read and write them in sequence. You cannot bring any other guy in between like j, you cannot bring in between. Like for example, it shows that the rising coupling fault between cells i and j, and there is a flip. So, this is what is what was the saying, there is a rising coupling fault if there is a rise from here to where, the x is going to be coupled.

So, in this case what you can do, you have to first examine cell j, and then basically you have to examine cell i, k should not come in between sorry here we are talking about i it is between i and j have told you. But, basically in between i and j, what is these two stills coupling in between nobody should come in between no cells should be examined in between like this one. Here all the problem was happening, because we were studying between j, k and j we were studying the coupling fault, some kind of a culprit has come in between.

So, basically this shows that basically if the two cells which are coupled are considered into picture, you have to only consider those two cells in sequence. So, this one will tell you how to do it bit in between i and j, if these are following one just add well of that. Similarly, you can also see for the rising and stuck-at 0 is very simple, you take cell you take cell j in cell j if it is a 0, then you read as of 0, and then you write a 1, so in that case you will be i immediately you read cell basically i, which was written with 0, you have to get back the value of 0.

(Refer Slide Time: 64:30)



March Test: Coupling Faults

Idempotent Rising-0 coupling fault $\langle \uparrow | 0 \rangle$ **between cell** i **and** j**:** (i) Cell j is to be written with a 0 and read back, (ii) cell i is to be written with 1 and read back, (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 to be 1.

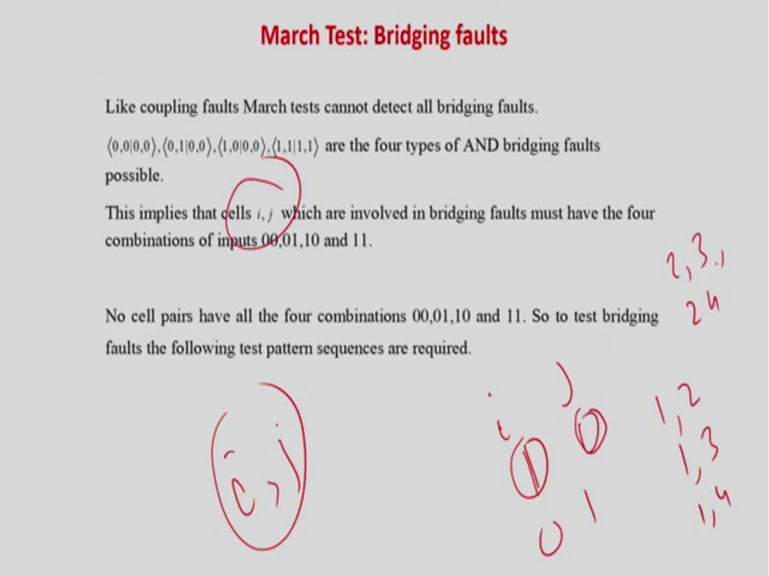
Idempotent Rising-1 coupling fault $\langle \uparrow | 1 \rangle$ **between cell** i **and** j**:** (i) Cell j is to be written with a 0 and read back, (ii) cell i is to be written with 0 and read back, (iii) cell j is to be written with a 1 and read back, and (iv) value at cell i is to be read and checked to be 0.

In this case, basically means again let us do it in steps that is better rather than making it its abstract. So, what is saying? Cell j is to be written with 0 and read back. So, this is cell j, this is say i. So, you are reading a 0, and reading it back no problem cell i is to be written with 1 that I am doing. Then basically cell j is to be written with a 1, you are reading it fine, you are reading it a 1. So, it is a rise, so if the rise and the fault is basically stuck-at 0 for i, again you have to read it back. If you are still remaining as a 1, your fault is detected.

Similarly, all the cases I have shown in these two slides, very simple idea. Only I want to emphasize the point is that say this says that sorry this says that cell j if the rice, the other cell should be stuck-at one. So, it is very simple, you go to cell j making the rice transition, and in the cell i you make a value of 1. So, you make a value of 0 (Refer Time: 65:15) correspond to stuck-at one. Just see that the 0 value remains, it does not go to 1. So, it is very simple.

Transition, you give the effect in the cell j. Victim cell you go and read that whether this stuck-at fault or the whatever fault effect is not present that is the only thing you have to do. But, only thing is that j and i should be done in sequence no other cell can come into between.

(Refer Slide Time: 65:41)



March Test: Bridging faults

Like coupling faults March tests cannot detect all bridging faults.

$\langle 0,0|0,0\rangle, \langle 0,1|0,0\rangle, \langle 1,0|0,0\rangle, \langle 1,1|1,1\rangle$ are the four types of AND bridging faults possible.

This implies that cells i, j which are involved in bridging faults must have the four combinations of inputs 00,01,10 and 11.

No cell pairs have all the four combinations 00,01,10 and 11. So to test bridging faults the following test pattern sequences are required.

(Refer Time: 65:36) other cell comes into between like this one there can be masking effect. But, then you can tell me that, it is very simple. Similarly, you can also check for the bridging faults. So, bridging faults are also very simple. You have to apply 1 0 or 0 1 in the cells and see that the values are there that is you apply in i and j if they are having some bridging faults, so you have to apply a 1, read a 1 write as write a 0, read a 0, then you can find that if it is the (Refer Time: 66:01) fault this one will also become a 0.

So, very simply you can test this fault is very very logical, I mean I should not spend more time on that. But, only thing I am going to re-emphasize is that that i and j should be taken into together, you cannot put anything in between, then there may be a problem. But, then actually the test pattern what I want to emphasize here is that then test pattern application be very complicated, because March test is very simple 0, 1, 2, 3, 4, 5 you come down and go this way.

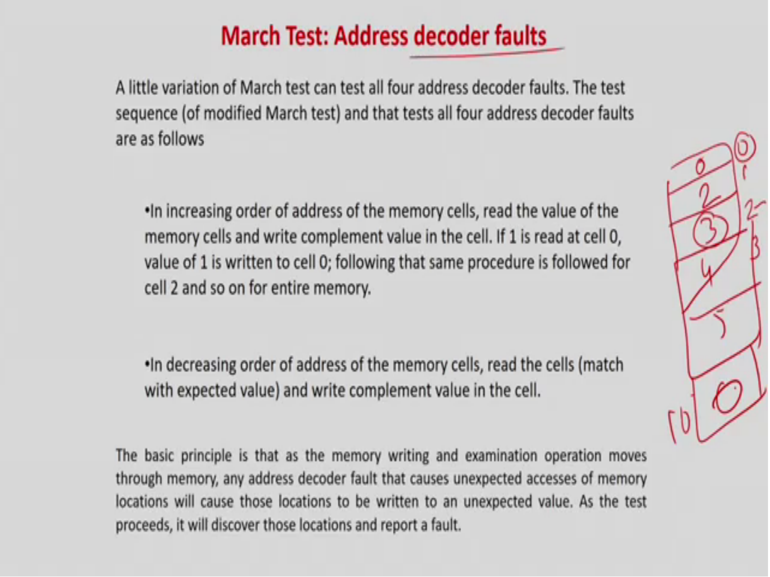
So, this way, so this way test pattern application, the counter designed by LFSR, because very simple. But, if I have to say that i and j, so many combinations of i and j will be there 1, 2; 1, 3; 1, 7; 1,6; 2, 1; 2, 3, so all these combinations sequence has to be generated in patterns. Then actually it will make the test pattern application type more complicated, like 1, 2; 1, 3; 1, 4, so forth. There will be 2, 3 then 2, 4.

So, you can understand that, now I have to apply the value in a predetermined sequence, it cannot be like 0 to n come back and go forth. So, this specific patterns has to be generated by the address by the addressing mechanism for the test. So, I have the access cell 1, then cell 2, then 1, 3, then all the specific patterns has to be generated by the counter with (Refer Time: 67:15) as the memory address, so that will actually make the hardware area over will be higher.

So, they in fact that is the truth, you cannot make a magic and simplify that. So, if you want to just go for transition fault test and stuck-at fault test, then you can go by this mechanism. But, if you want to go for a bridging fault test, coupling fault is different neighborhood test, then specific values of i and j or the specific value of the pair neighbors, so that way yourself has to be accessed. Like if you are taking a neighborhood business like this one, then you have to address all first this one, then this one, then this one, then this one, and this one, you cannot go in sequence like this that is not possible. If you go in sequence, you may lead to masking.

So, in fact for high level of fault models like which you have discussed, you have to pay more time more patterns basically, more time will be required or more hardware area will be required to generate the specific address of the memory cells. And in fact if you want to better quality of test, you have to do that. But, if you want a very simple quality of test to a low area, you just go for a March simple March test, and you will see that I am going to having stuck-at fault and transition fault a certain distance, so that way, so it depends on the trade-off what you want.

(Refer Slide Time: 68:20)



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.

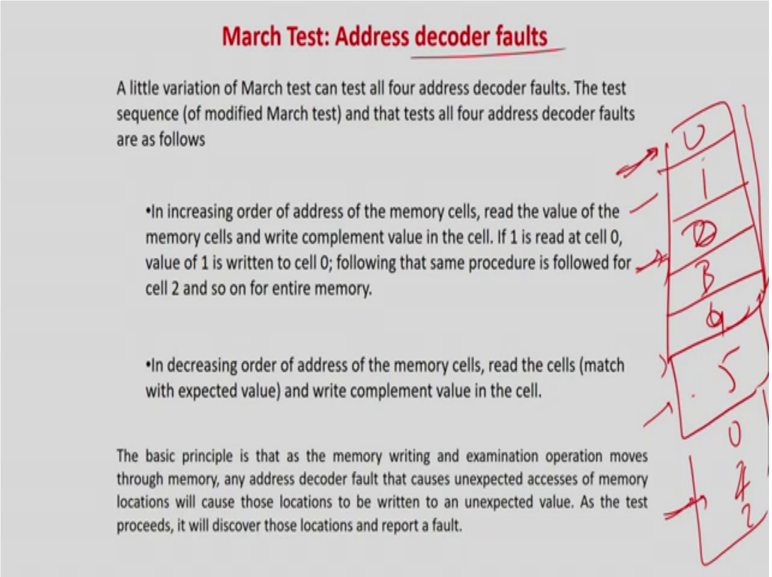
Last, but not the least address decoder fault test. So, address decoder fault test is very simple, just what you have to do is that you have to find out when the addresses are properly generated or not. You can see that if I am using a March test with a slightly changing that method I can find out whether the cells are generated properly or not. Instead of reading this slide, I will just tell you what it is done.

So, increasing order mean that means, you are going to see that these are the memory, maybe you have to see that whether first 0, then 1, then 2, then 3 are generated or not that is there should not be any kind. If I want it to generate the address 0, it is not generating address 3 that is what if I able to find out, then I (Refer Time: 68:54) the address decoder as no fault. But, instead that will actually make it very complicated to test that whether exactly what (Refer Time: 69:01) how can I do that.

Then I have to have some value of 1, 2, 3, 4, 5 these are the contents, which I should be able to put in the memory and read it (Refer Time: 69:08) then only you will be able to know. Even if it is not 0, 1, 2, 3, 4, you should have different specific values in the different memory locations. Then only I will be able to know that if I have generated address 2, the specific value is there or not that means, if for example, if I if the decoder generates, if I want to generate the address 0, and it generates the address say 10. And if this content also is having the content 0, then we will be not able to find out whether there any problem in address decoder business.

So, how can I do it? Simple, put different values in all the memory locations like so from 0 you start with 0, 1, 2, 3 whatever be the memory location that value put as the value in the memory. And start generating the address 0, 1, 2, 3, 4, 5, 6, 7, 8, you read out the value, and the address should also be the content should also be in sequence. Or you can put some different values in all the memory locations, and find out that if the corresponding values are coming out of the data bus, if I am giving the corresponding address, but that will actually make (Refer Time: 70:02) very very difficult and very very cumbersome and area over it will be higher. So, what people do? They do it a very simple way.

(Refer Slide Time: 70:10)



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.

Instead of having n different location contains to verify absolute working of the memory decoder, people go for a very simple logic and try to find out that whether more or less it is working fine. So, whenever I have simplicity in the testing, you should always understand that I will compromise (Refer Time: 70:26) inequality. So, what they do? They write a 0, read a 0, they write a 1, read a 1, they write a write a 0, read a 0, write a 1, (Refer Time: 10:35) so alternative they will do. So, that means what. When I am accessing this cell 0, I am writing a 0, reading a reading writing a 0, reading a 0, then 1 I will go back.

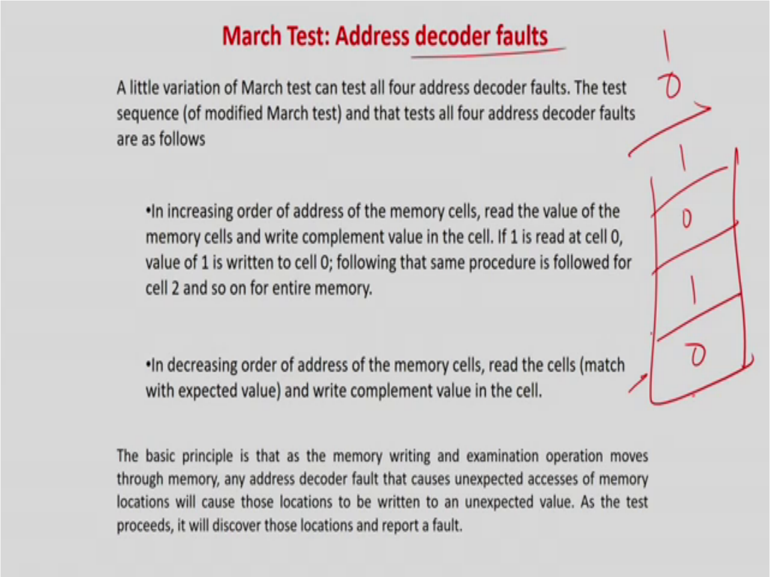
And while coming back, again I should get the alternate variables that means, if there is a slight problem that after accessing this, it by chance I accept access something after the accessing this cell, I am at going to access the second cell, so I should get a 1. But, instead (Refer Time: 70:58) mistake if I access some cell here or some cell here, where the content is 0, you know that the decoder fault.

What is the probability? 50 percent; masking probability is 50 percent. That is what, that is I want to access this cell, if I am accessing this cell is a mask, but the fault is there. But, if I access any of these cells corresponding to 1, I know that there is a addressing decoder fault, assuming that the memory cell is fine. Because, anyway we know that single stuck-at fault stuck-at fault single fault models we take, either the fault is in the memory cell or the fault is in the decoder.

Here we are when you are doing a test for the address decoder, we are assuming that the memory is free of faults. So, now you can tell me of course, 2 means the probability is 50 percent. I can have 0, 1, 2, 3, 4, 5, so the probability will be one-fifth that way. So, means if I am accessing this, I should again I will start from 0 mod 5, 1, 2, 0, 1, 2, 3, 4 like that.

So, if I want to access the cell 0, I should get a value 0, 1, 2. If there is a problem, instead of 2, if I am going to get a 5, I know that is an error, but again I am repeating the values, so 0, 1, 2, 3. Again, if I am getting the value of mapping is 2, but here also the possibility of mod value be 2, it will be error will be March very simple. But, instead of I mean having all these mod and making it complicated, people just have binary values, because March test because we want to map everything to March test.

(Refer Slide Time: 72:14)



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, March test instead of putting all 0s and all 1s and reading in sequence, people do it in the 0, 1, 0, 1, 0, 1, so basically I should get alternate values. So, if I am getting the alternate values and more or less 50 percent assure that there is no problem in the address decoding, but the masking probability is 50 percent. But, anyway as I told you if you want to get a full decoder based test, then all the memory cells should have individual values, reading it should give unique values, again we it will be more area overhead and all these things will come up.

So, this is what is the and again as I told you March tests people have found out to be having good coverage for most of the fault models. So, people apply March test, and then try to see what is possible and what is not possible. Slightly modified March test like this, you can find out 50 percent direction probability will therefore the address decoder faults.

And people have found out that more or less the reliability selling it to the market, because again I told you that testing is not very something mathematical, I want to test as much what is required to maintain a quality and I should make profit in the market. So, people have found by doing this 50 percent accuracy business in case of decoder, I am fine with it. So, therefore, this is the model, which is accept; and this is the algorithm, which is actually slightly modified March tests, (Refer Time: 73:21) do for address decoder testing. And people have found reliability to be fine.

So, when this basically I come to the end of this lecture. So, what we have shown basically, first we have shown different type of fault models, which are applicable to memory, because memory are more complicated in from the testing perspective, because their cells are very near to each other. Now, we have, we are now not going to target each of the fault models and try to find out ATPG for that we are not doing in memory. Rather somebody has given us algorithm called March test, which is very simple, writing 0s reading 0s and so forth. Then try try to find out that which are the fault models covered by March test. And we have seen that we slight modification in March test, many of the important fault models can be covered. So, for memory, March test with slight modification in the defect (Refer Time: 74:01).

So, with this, we can come to the end of this lecture. And in the next module on testing, we will try to see some advances in hardware design for embedded systems, and what are the test techniques required for such cases. Advanced design for embedded hardware, and what are the I means modification required in traditional testing, which we have been looking at till now, how they have to be modified, or what are the advancement required that we will look in those lectures.

Thank you.