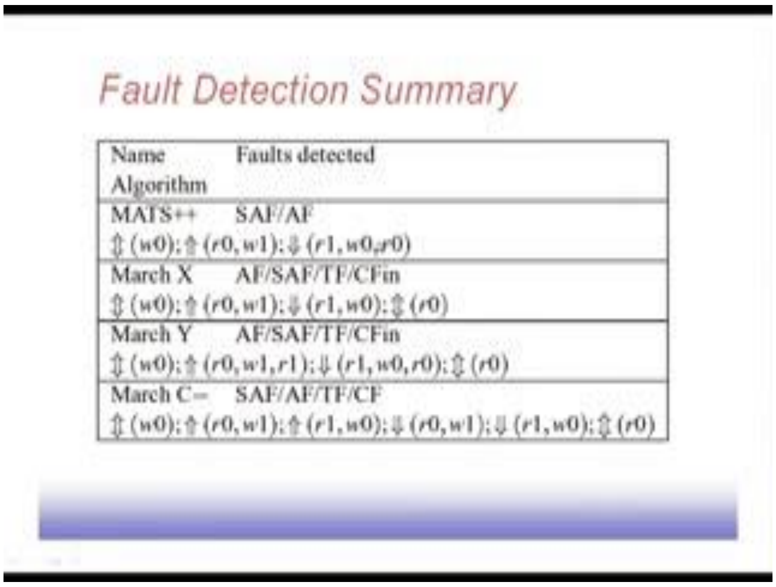
Digital VLSI Testing

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

Lecture – 59
Memory Testing (Contd.)

(Refer Slide Time: 00:21)

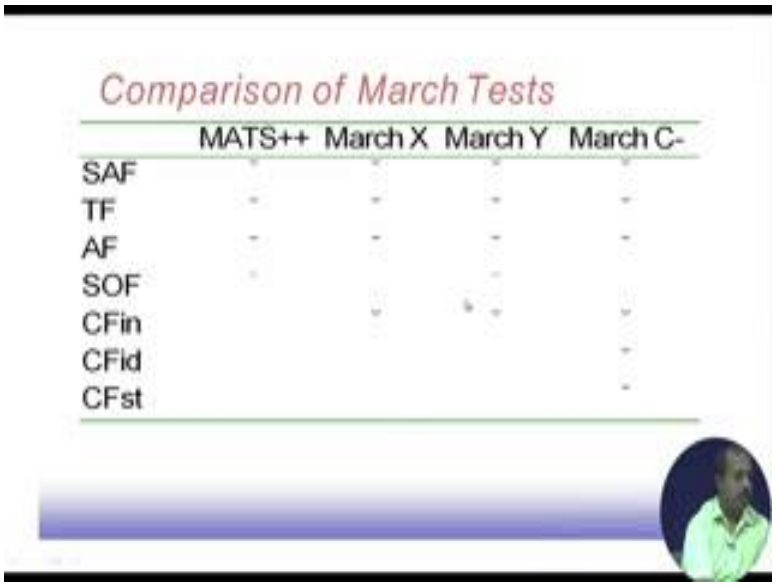


Fault Detection Summary

Name	Faults detected
Algorithm	
MATS++	SAF/AF
$\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0,r0)$	
March X	AF/SAF/TF/CFin
$\Updownarrow(w0);\Uparrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$	
March Y	AF/SAF/TF/CFin
$\Updownarrow(w0);\Uparrow(r0,w1,r1);\Downarrow(r1,w0,r0);\Updownarrow(r0)$	
March C−	SAF/AF/TF/CF
$\Updownarrow(w0);\Uparrow(r0,w1);\Uparrow(r1,w0);\Downarrow(r0,w1);\Downarrow(r1,w0);\Updownarrow(r0)$	

To summarize this fault detection, so we can this table actually summarizes the different faults that are detected like MATS plus plus, detects stuck at fault and address decoder fault, and this is the corresponding march sequence. March x determine this faults, March y also these are some of the well known march sequences that have been reported in the literature. And apart from these also there are many other march sequence that we can find in the literature and accordingly different authors they have claimed a different types of faults getting detected by them.

(Refer Slide Time: 00:54)

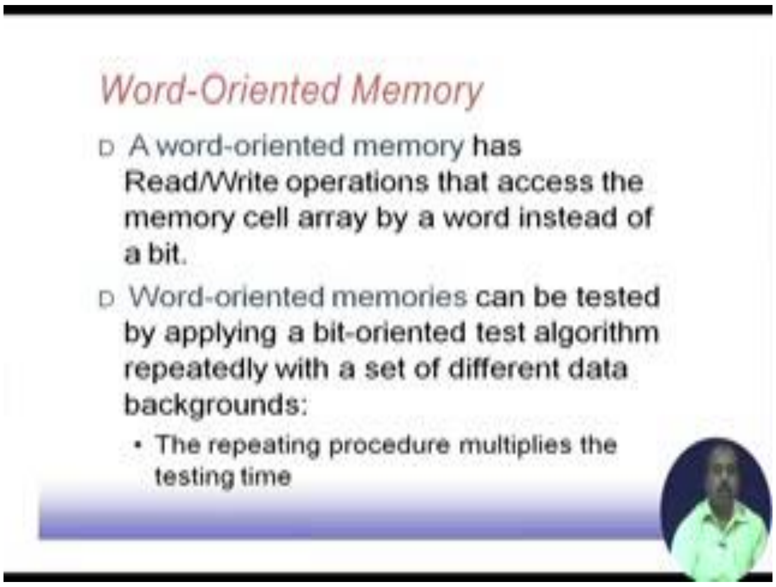


Comparison of March Tests

	MATS++	March X	March Y	March C-
SAF	✓	✓	✓	✓
TF	✓	✓	✓	✓
AF	✓	✓	✓	✓
SOF	✓		✓	
CFin		✓	✓	✓
CFid				✓
CFst				✓

So, this is another comparison. So, you see that this march C minus, so this covers most of the faults expecting this stuck-open fault, whereas so within march Y march X, so these are the different faults they define algorithm they could cover some of these algorithms some of this faults.

(Refer Slide Time: 01:12)

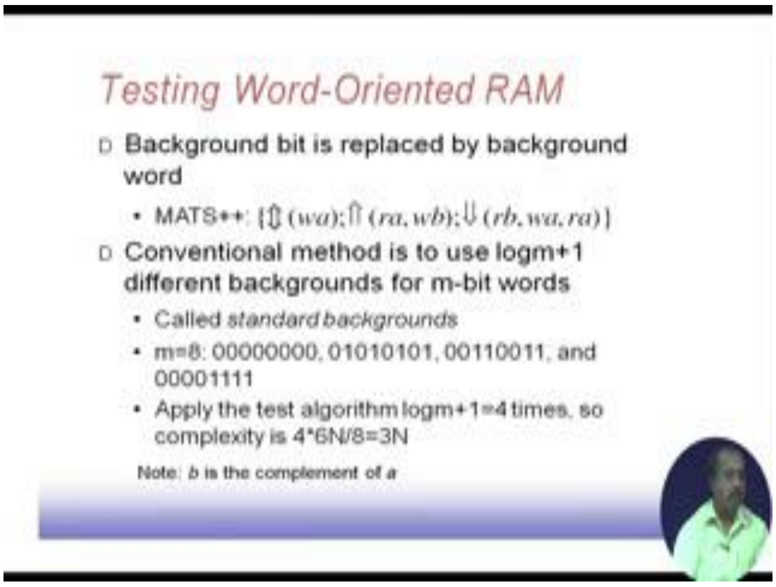


Word-Oriented Memory

- A word-oriented memory has Read/Write operations that access the memory cell array by a word instead of a bit.
- Word-oriented memories can be tested by applying a bit-oriented test algorithm repeatedly with a set of different data backgrounds:
  - The repeating procedure multiplies the testing time

Another way of looking into memory testing is the word-oriented memory. So, so far whatever we were looking into, so we are assuming that we can access individual cells of the memory as separately, and then we can modify, we can do a read-write operation on the individual cells. Some of the memories are word-oriented where the access is for the entire word. So, if they are actually we can also use this march based testing march testing mechanisms for doing the test memory testing. So, word-oriented memory, it has read write operations that access the memory cell array by a word instead of a bit. So, they can be tested by applying a bit-oriented test algorithm repeatedly with a set of different data backgrounds. So, this way most of we can say we can just do it for every bit of the word. So, we can repeat the process, so that is one possibility, so naturally since we are repeating for every bit, so the testing time will be more.

(Refer Slide Time: 02:17)



Testing Word-Oriented RAM

- Background bit is replaced by background word
  - MATS++: $\{\Updownarrow(wa);\Uparrow(ra,wb);\Downarrow(rb,wa,ra)\}$
- Conventional method is to use logm+1 different backgrounds for m-bit words
  - Called *standard backgrounds*
  - m=8: 00000000, 01010101, 00110011, and 00001111
  - Apply the test algorithm logm+1=4 times, so complexity is 4*6N/8=3N

Note: *b* is the complement of *a*

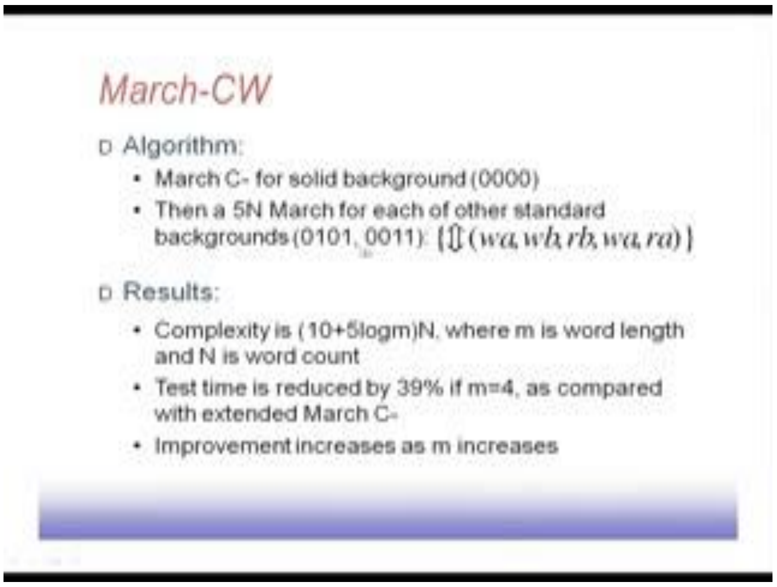
So, another way of doing it is we can we can do it in a word-oriented fashion. So, background bit instead of talking about the background bit, we talk about background word. For example, this MATS plus plus algorithm. So, we can talk about write a, where a is a word pattern. So, if you word is 8 bit then a may be all zeros or a may be also alternate 0, 1’s, so it is a 8 bit pattern. Similarly, so read a and write b, so since we are retain the value a pattern a, so we are expecting to read a pattern a, then we are writing another pattern b, then we are trying to expecting to read the particular pattern b here then write a and read a. So, this is the MATS plus plus in a word-oriented memory.

So, if I have got in between words then we can have different backgrounds like log m plus one different backgrounds are used. So, this is the standard so these are called standard background. So, basically these are the backgrounds like if m equal to 8, so it means that there are 4 standard background. So, these are the standard background. So, all zeros, then alternate zero ones, then two-zero, two-one, so runs of zero basically here it is a rounds of eight-zero. So, it is a run of one-zero, there is a run of two-zero is a run of four-zeros, so that way we can have or we can read this one as run of four-zero run of eight-zeros. So, one-zero, two-zero, four-zero, eight-zero, so we can read it like that. So, these are the all standard backgrounds.

So, for each of this background, so we will be applying the test algorithm, so since it is for the complexity will become 4 into 6 N by 8, because there are six elements in this sequence six operations, so this 6 into N, so and that divided by 8 because I am doing it every word at a time. So, N by 8 is the number of words in the memory, memory has got N bits. So, N by 8 is the number of words. So, total a number of operations is 3 N. So, b has to be complement of a; otherwise we will not be able to detect that stuck at zero or stuck at one type of situation, so this is b, it has to be compliment of a.

So, there is a cocktail-march class of algorithm known as cocktail-march algorithms. So, it says that repeat the same algorithm for all log m plus 1 background is redundant, because they are may be intra-word coupling faults when you coupling only intra-word coupling faults, so this is redundant. So, intra-word coupling fault means within the same word two bits are coupled to each other. So, we can have some different algorithms, we can target different faults. So, what it says, it says that use multiple backgrounds in a single algorithm ran. So, in the same run, so we use different backgrounds. Merge and forge different algorithms into backgrounds into a single algorithm. So, this is what is looking for, so for word-oriented memory, it will be good.

(Refer Slide Time: 05:38)

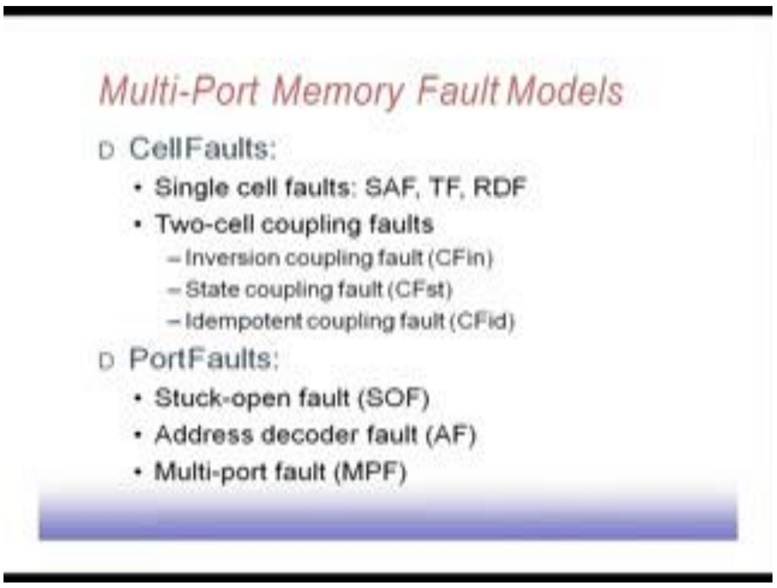


March-CW

- Algorithm:
  - March C- for solid background (0000)
  - Then a 5N March for each of other standard backgrounds (0101, 0011): $\{\Updownarrow(wa, wb, rb, wa, ra)\}$
- Results:
  - Complexity is (10+5logm)N, where m is word length and N is word count
  - Test time is reduced by 39% if m=4, as compared with extended March C-
  - Improvement increases as m increases

So, this is basically the thing that we are talking about. So, march-CW, march minus CW, it says that march c minus using the solid background 0000. Then a 5 N march of each of other standard background, so 0101 and 0011, so this will be done like this. So, write a, write b, read b, write a read a. So, if there are two backgrounds a and b, so there are being used and so that way we can do it. So, the complexity is given by this expression 10 plus 5 log m into N, and the test time is reduced by 39 percent. So, this as a memory size increases as or word size increases, your improvement will also increase - percentage improvement also increase.

(Refer Slide Time: 06:28)



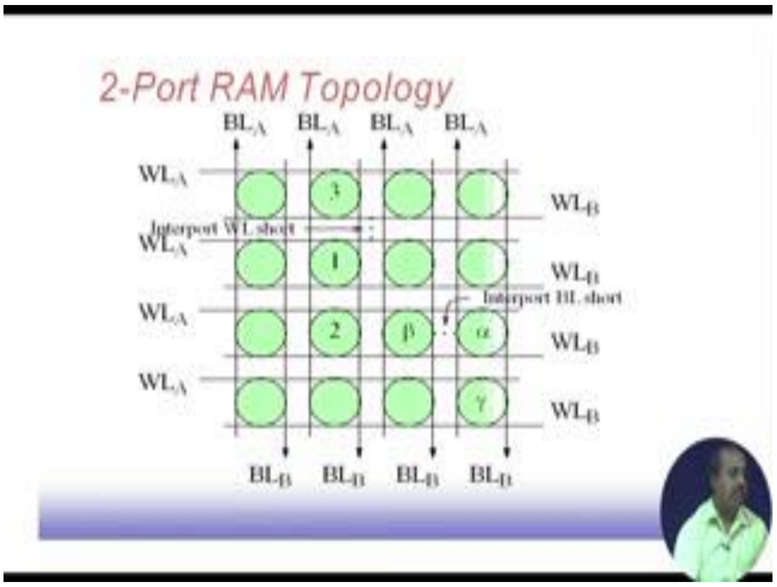
Multi-Port Memory Fault Models

- Cell Faults:
  - Single cell faults: SAF, TF, RDF
  - Two-cell coupling faults
    - Inversion coupling fault (CFin)
    - State coupling fault (CFst)
    - Idempotent coupling fault (CFid)
- Port Faults:
  - Stuck-open fault (SOF)
  - Address decoder fault (AF)
  - Multi-port fault (MPF)

Another type of memory that is common is known as multi-port memory. So, multi-port memory means you can there are more than one read write port available. So, you can do more than one read write operation to the memory simultaneously. So, this is particularly useful if you have got a multi-processer system, so different processors. So, they will ask for content for memory or different memory location content. So, if problem with single port memory is that even if I have got multiple processors, so they will have to wait for memory access which is sequential in nature. So, if I have got multi-port memory then they can be accessed parallelely.

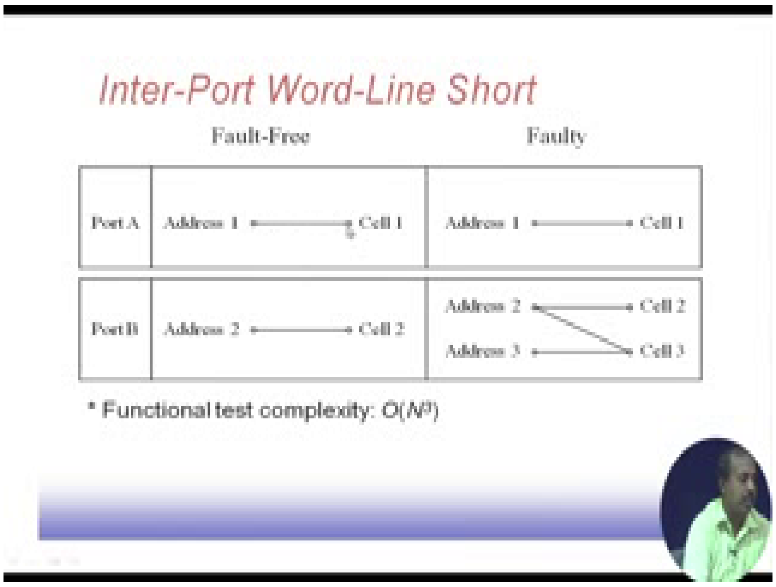
So, in the problems that can occur in multi-port memory, so they apart from this cell faults which is traditional, so we can have port faults. So, port fault is a stuck-open fault, then this address decoder fault and multi-port fault, so these are known as port faults. So, multi-port fault means across ports, so there is a problem. So, apart from that, we have got the standard cell faults like stuck-at-fault, transition fault, read destruction fault so like that. So, we have got those faults, but apart from that we have got port faults.

(Refer Slide Time: 07:46)



So, this is a 2-port RAM topology. So, if you see that there are as I said that there are two ports. So, one port is port A, another is port B. So, for port A, we have got bit-line A, and bit-line A bar. So, they are bit-line A and bit-line, so these first line, so they are corresponding to bit-line A. And then the second line, so they are corresponding to bit-line B. Similarly, while going like this, so these are the word-lines, so we have got word-lines for port A and word-lines for port B. Now, if there is a inter port word-line short, so if these two lines shorted, so that is WL B is getting shorted it WL A. And similarly, another possibility is inter-port bit-line short. So, this bit-line A is getting shorted with bit-line a of bit-line B is getting shorted with bit-line A, so that way that can be. So, this type of faults they are known as inter multi-port fault or inter-port fault.

(Refer Slide Time: 08:53)



Inter-Port Word-Line Short

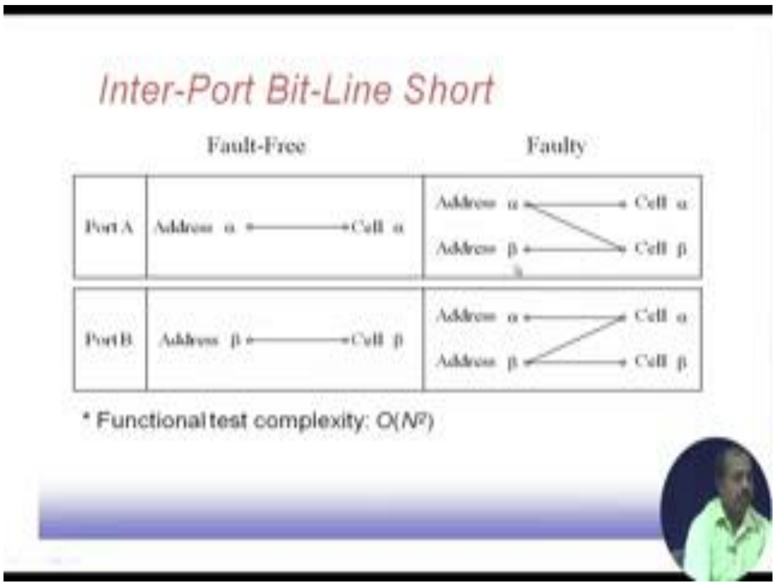
	Fault-Free	Faulty
Port A	Address 1 ⟷ Cell 1	Address 1 ⟷ Cell 1
Port B	Address 2 ⟷ Cell 2	Address 2 ⟷ Cell 2, Cell 3 Address 3 ⟷ Cell 3

* Functional test complexity: $O(N^3)$

Now, what is the situation? So, if there is a word-line short then in a fault free access port A if put address 1, so it will be accessing cell one. Port B you put address 2, so it will be accessing cell 2. In a faulty situation, address one accesses cell one. So, address, so if it is a case of word-line short, so we have talking about, so this particular short. So, word-line A, it accesses a particular cell, so word-line A is so this one. So, it is accessing cell one as you see here. So, word-line A, the address one is accessing cell one.

But word-line B, so this is actually shorted; so word-line B shorted here, as a result it will access the address 2, it will be accessing cell two and cell three; for address three it is accessing cell three, but for ever is putting address 2, so it will be accessing cell 2 and cell 3. So, this word-line B, so this is sorted with word-line 2, so when it is trying to access this cell 2, so it is also accessing cell 3. So, that way complexity will become of course, this big O N cube complexity is high, but this maybe the situation.

(Refer Slide Time: 10:19)

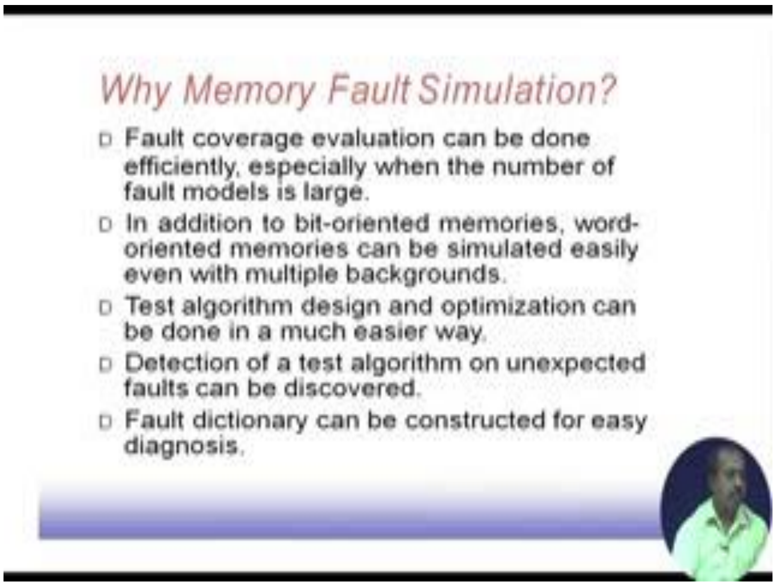


Inter-Port Bit-Line Short

* Functional test complexity: $O(N^2)$

Similarly, this bit-line short, so here also fault free situation this address alpha will access cell alpha, address beta will address cell beta. But in a faulty situation, so address alpha will access from port A cell alpha and cell beta; and this address beta in case of faulty situation, so it will access cell alpha and cell beta on port b, so that way we can have multiple location access. So, this address decoder part, so they become faulty because of this word-line and bit line shorts. So, these are about multi-port.

(Refer Slide Time: 10:53)



Why Memory Fault Simulation?

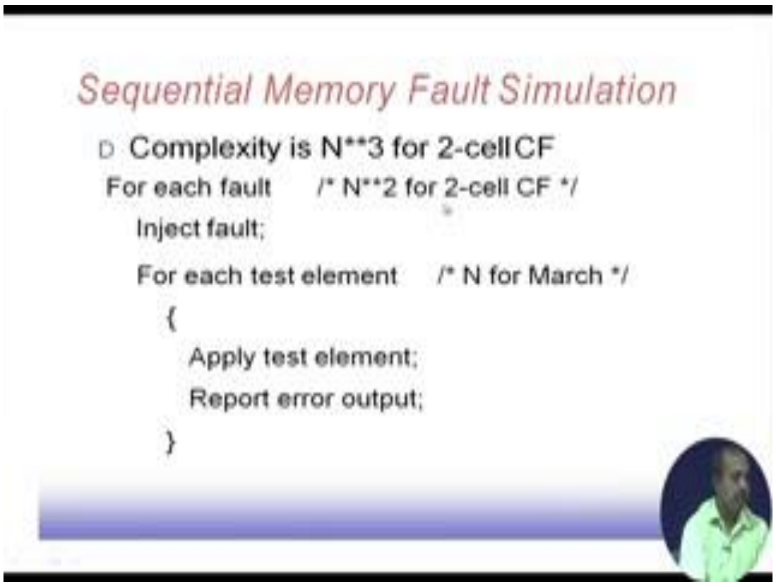
- Fault coverage evaluation can be done efficiently, especially when the number of fault models is large.
- In addition to bit-oriented memories, word-oriented memories can be simulated easily even with multiple backgrounds.
- Test algorithm design and optimization can be done in a much easier way.
- Detection of a test algorithm on unexpected faults can be discovered.
- Fault dictionary can be constructed for easy diagnosis.

So, there are some test pattern generation techniques for testing this multi-port faults. Now, this is another type of this memory fault test algorithm generation is via simulation. So, what we do is that we generate some test algorithm and do a simulation to see whether it can detects a number of faults or not. So, many time this memory faults simulation is done. So, why is it so important? So, it has to be done efficiently when the number of fault models is large fault coverage evaluation will be done efficiently. So, if the number of fault models are large, then if you want to simulate, if you want to check whether the faults are getting covered or not then for each of the fault models, so we have to simulate the memory once. So, if we have doing like this then it will take huge amount of time. So, an efficient evaluation means I will be simulating the memory for multiple fault model simultaneously. So, as a result the fault coverage evaluation will be faster.

So, again this bit-oriented and word-oriented memories can also be simulated. And test algorithm design and optimization can be done in a much easier way. So, why, because test algorithm design is nothing but such a proposing some new march elements or march sequences. So, since I have got a simulator, so I can immediately check whether which faults are getting covered by the new pattern that we have new test element that we have introduced into the test. So, that way if the fault simulation is strong then we can do it fast.

So, detection of a test algorithm one unexpected faults can be discovered. So, for some unexpected fault means some faults are that we did not model previously also, so that can be that can get detected by a test pattern test algorithm. So, that will also happened due to this fast fault or efficient fault simulation. And faults dictionary can be constructed for diagnosis. So, we can identify which location are failed and accordingly we can identify like which cells are faulty and try to do a replace and all that.

(Refer Slide Time: 13:20)



Sequential Memory Fault Simulation

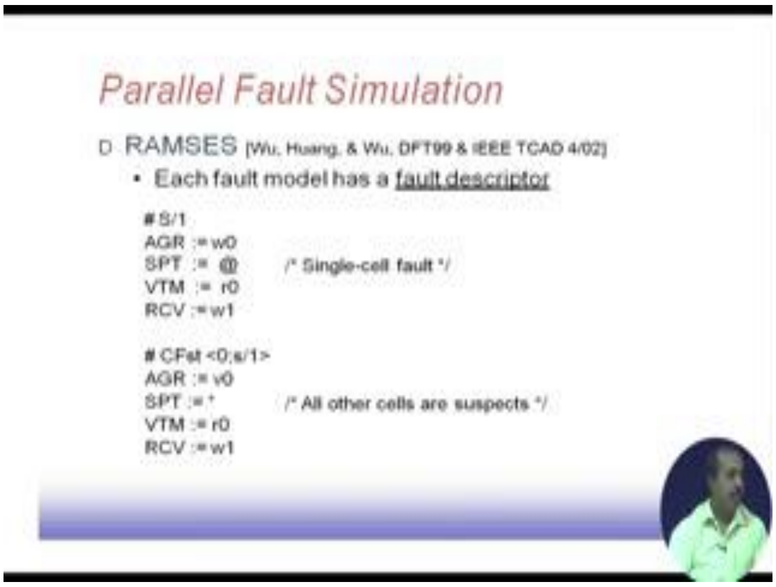
- Complexity is N**3 for 2-cell CF

```
For each fault      /* N**2 for 2-cell CF */
   Inject fault;
   For each test element     /* N for March */
      {
         Apply test element;
         Report error output;
      }
```

So, these a typical example. Say we had we get sequential memory fault simulation. So, complexity is N cube for 2-cell coupling fault. Why, because if we have to check a 2-cell coupling fault then there are N number of cells then there are N square type of coupling. So, you can choose 2-cells N choose 2. So, it is N square number of coupling fault that can occur. Now, for each of this faults, so I need to inject the fault. And for each of the test element, so we have to apply the test element and report the error output. So, this has to be done for march test at least N operation has to be done.

So, we have to it is say at least some constant multiplied by N for march algorithms we have seen it is 4 N, 6 N like that. So, this is again some sort of N. So, this N in to N square, so this is the overall complexity becomes N cube. And N being large this N cube is going to be relatively high. So, if we are doing a sequential memory fault simulation, so we do not get the correct answer because we have to correct the answer in the sense that the with the amount of time that is needed to do the simulation, so that may not be able to afford that time.

(Refer Slide Time: 14:41)



Parallel Fault Simulation

- RAMSES [Wu, Huang, & Wu, DFT99 & IEEE TCAD 4/02]
  - Each fault model has a <u>fault descriptor</u>

```
# S/1
AGR := w0
SPT :=  @        /* Single-cell fault */
VTM  :=  r0
RCV := w1

# CFst <0;s/1>
AGR := v0
SPT := *         /* All other cells are suspects */
VTM := r0
RCV := w1
```

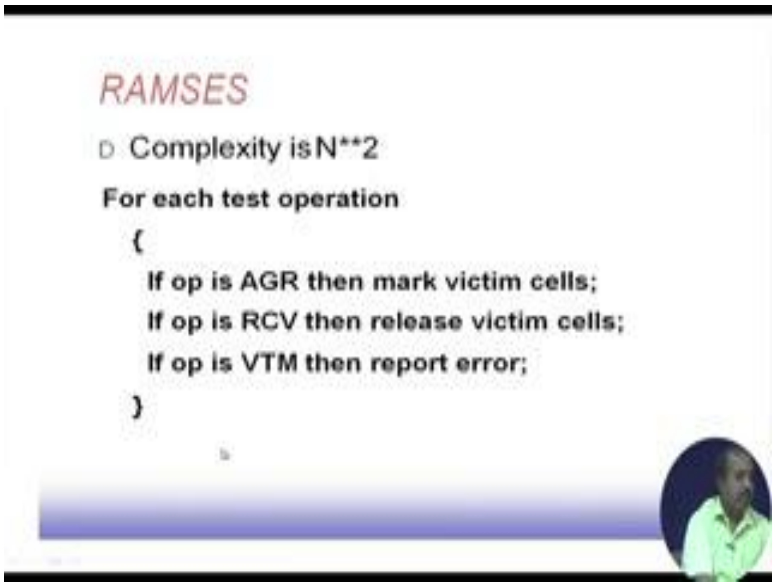
So, what is done is the parallely fault simulation. So, this is one technique RAMSES that has been reported in 2002, where each fault model has a fault descriptor. So, what it is says is that, so it is the first fault that we have considering is stuck-at-1. So, it has got a descriptor, how? So, it has got a aggressor, this AGR, AGR is the aggressor. So, aggressor pattern, so that is which will create that detection of that particular fault. So, this is basically if I do a write operation, so this is the aggressor write 0. So, I want to exide this stuck-at-1 fault, so naturally I have to write a 0.

Then this is the suspect. So, whom are is suspecting we are suspecting the same cell. So, it is at the rate means same cell we are suspecting. Then what is the victim marking victim marking I can do while doing r 0. So, if I read now read and expect values 0, so how to recover from this. So, we have modified the same content to 0 by the aggressor. So, you have to write a one to go back to the to recover from the fault because this is the stuck-at-1. So, if it is written one; that means, the fault will go from the cell.

So, this is actually a pattern. So, this tells that if you find that a particular algorithm particular test algorithm, it is fast writing is zero, then it is doing a read zero and there it is doing a write one, in a march element it is doing that that means, that particular march element can detect stuck at fault at that particular cell. Similarly, we can determine this coupling fault this is so it is zero and then go from there it is going to stuck-at-1. So, this way we can we can again identify the aggressor pattern. So, it is rising it is v 0 then this is suspect all other cells then we have reading the content zero as the victim and for recovery we have to write a one here.

So, this way we can; so this actual notation like v and w, so they are documented in this paper. So, I am not going into that, but essentially what it says is that in the template I should have this type of pattern, and from there we can identify that this particular test pattern is going to identify the fault.

(Refer Slide Time: 16:56)



RAMSES

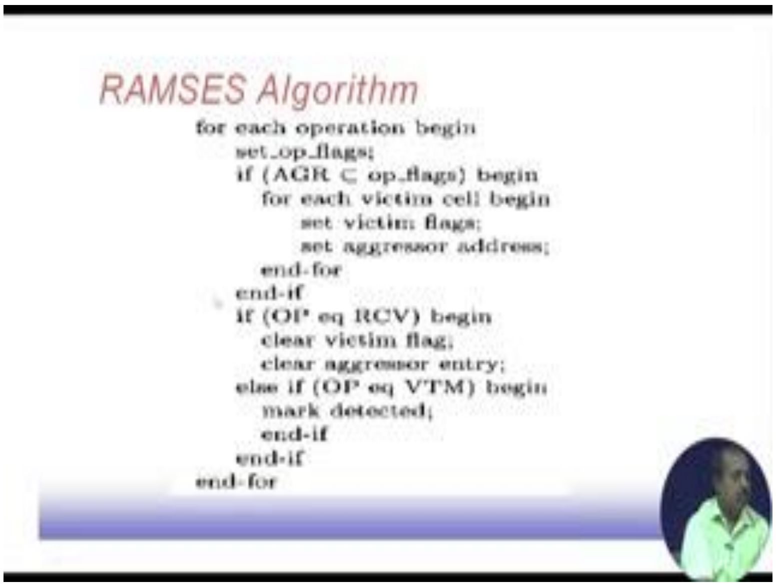
- Complexity is N**2

```
For each test operation
  {
    If op is AGR then mark victim cells;
    If op is RCV then release victim cells;
    If op is VTM then report error;
  }
```

So, what it says is that for each test operation, so these are RAMSES algorithms it says is for each test operation if the operation that we are doing is an aggressor then mark the victim cells. Like if we find that honestly we are doing some aggressor operation writing zero at a cell or writing one at a cell, so what are the victim. So, we can mark the corresponding victims. And then if the operation is recovery then we release the victim cells, so we have to write back the original values that should be there the recovered. And if the operation is victim then we report error.

So, victim you see that in this case victim is r 0. So, if you find something else, so r 1 if you get the value as 1 that means, we can that is a. So, the victim is detected, so we can report an error. So, this way, all the templates all the all the templates that we have in a in a test algorithm, so they can be tested parallely and for all the types of faults that you are going to detect. So, what we have to do is that for all the fault models that you want to consider, so you have to write down the corresponding descriptors like this, and then for the test algorithm that we have specified in terms of march elements. So, you can do this, for each test operation we can figure out that whether this aggressor is part is matching, and if the aggressor part is matching then which one is victim and all that, so that you can do.

(Refer Slide Time: 18:53)



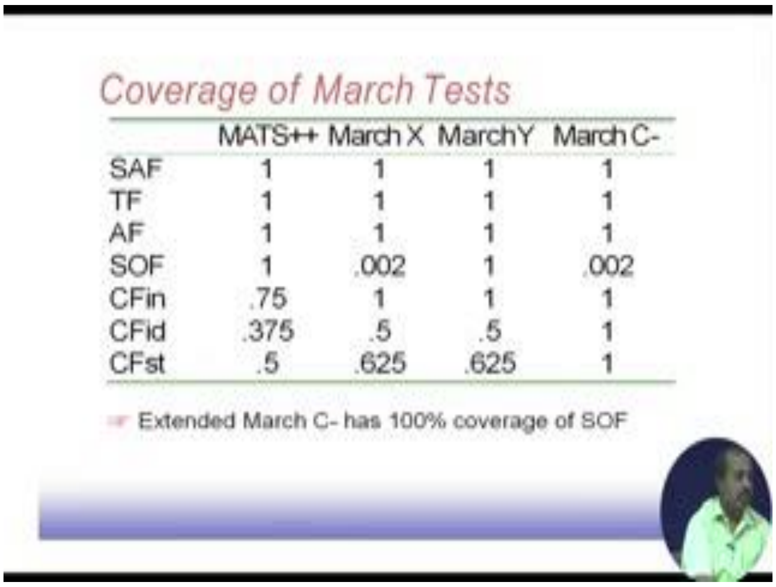
RAMSES Algorithm

```
for each operation begin
    set_op_flags;
    if (AGR ⊂ op_flags) begin
        for each victim cell begin
            set victim flags;
            set aggressor address;
        end-for
    end-if
    if (OP eq RCV) begin
        clear victim flag;
        clear aggressor entry;
    else if (OP eq VTM) begin
        mark detected;
        end-if
    end-if
end-for
```

So, this is the RAMSES algorithm. So, this in detail it is telling, so it say it is telling that for each operation set operation flags. So, if aggressor is a subset of operation flag that is aggressor is there in the operational flag like w 0 because that is an aggressor. So, this aggressor is there in the operation flags. Then for each victim cell we say though, so we set the victim flag and set aggressor address which address has created this aggression. And on the other hand, if operation is recovery then we have to clear the victim flag in clear aggressor entry; otherwise if operation is victim, then we say that ok it is detected. So, that way while doing the simulation, so if we find at this victim got detected because of the difference from the expected value, so this we say that the fault getting detected.

So, it is irrespective of the fault model that we have considering that is the beauty of this algorithm. So, it is not looking separately whether it is detecting a stuck-at-fault or coupling fault or transition fault like that, so if it matches whichever template whichever descriptor it matches, so that it has detected that particular fault, so that way it is irrespective of a fault model. So, all the fault models are being simulated parallely in this mechanism.

(Refer Slide Time: 20:20)



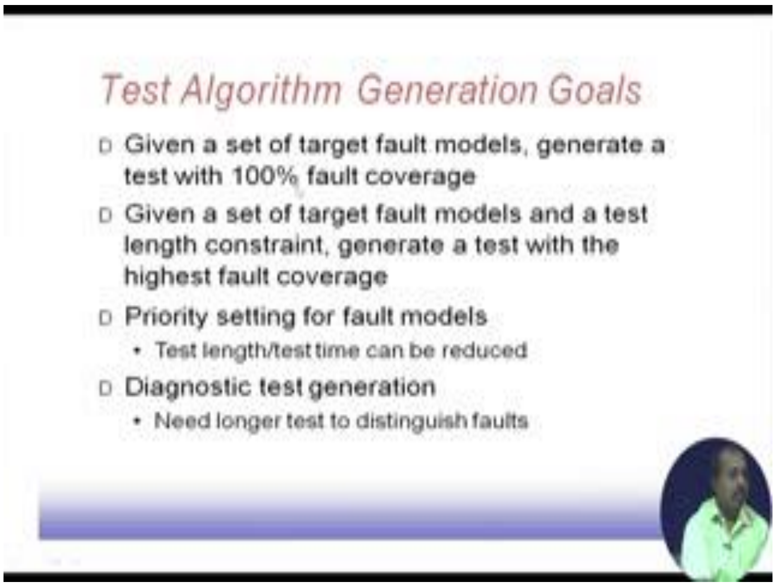
Coverage of March Tests

	MATS++	March X	March Y	March C-
SAF	1	1	1	1
TF	1	1	1	1
AF	1	1	1	1
SOF	1	.002	1	.002
CFin	.75	1	1	1
CFid	.375	.5	.5	1
CFst	.5	.625	.625	1

- Extended March C- has 100% coverage of SOF

So, these are the various sequence of various results like this stuck-at-fault, so this MATS plus plus is 100 percent, so all these faults, so when you pass through this RAMSES algorithm. So, MATS plus plus, so it has it has found that all these faults they 100 percent can be detected; for coupling fault 75 percent can be detected; for coupling fault, (Refer Time: 20:47), so 37.5 percent can be detected, so these are the various measures that we can do. So, if the RAMSES algorithm was not there, then for each of these faults, so I have to do a separate simulation, so that will take lot of time.

(Refer Slide Time: 21:00)



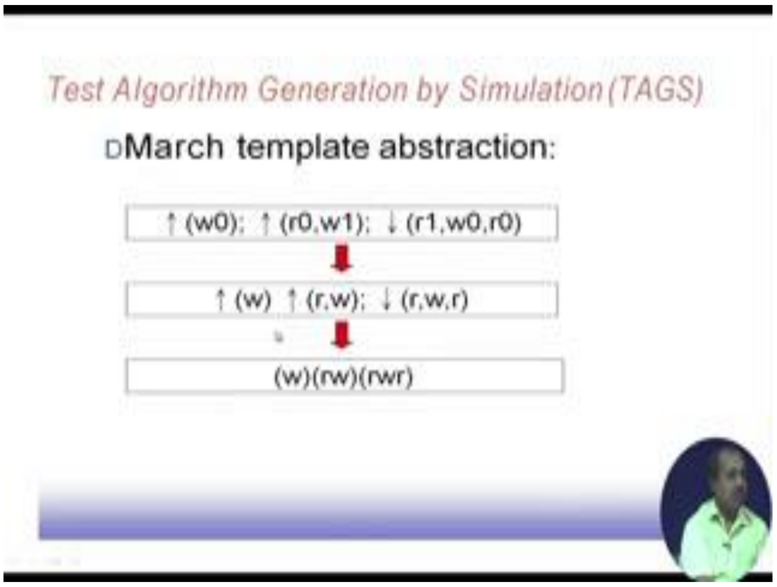
Test Algorithm Generation Goals

- Given a set of target fault models, generate a test with 100% fault coverage
- Given a set of target fault models and a test length constraint, generate a test with the highest fault coverage
- Priority setting for fault models
  - Test length/test time can be reduced
- Diagnostic test generation
  - Need longer test to distinguish faults

So, how can we do a test generation using this type of simulation. So, given a set of target fault models, we want to generate a test say with 100 percent fault coverage, so that is what we are looking for. And other possibility is a given a set of target fault models and a test length constant generate test with the highest fault coverage. So, one thing first point tells that there is no restriction on the number of march element we have, we can have any number of march elements until and unless we have got 100 percent fault coverage for the given fault model. So, we go on adding elements to the march sequence, so that way we can we stop at 100 percent fault coverage, but naturally as we are adding more and more templates into that mode sequence, so the complexity of the test algorithm is going up.

So, if there is a test length constant like say 7 N or 10 N or 15 N like that, so how can we generate of a test set with a highest fault coverage that is the other goal. So, these are the two ways in which we can view the test generation algorithm, so either of these two. So, we can have a priority setting for fault model. So, we can say that ok, we will try to cover this fault model test length and test time can be reduced. So, this is one possibility. And the another possibility is that we can has a diagonal logistic test generations also. So, we want to distinguish between the faults detected, so naturally for diagonal logistic test generation, so the test length will even be further longer.

(Refer Slide Time: 22:37)



Test Algorithm Generation by Simulation (TAGS)

- March template abstraction:

$\uparrow(w0);\ \uparrow(r0,w1);\ \downarrow(r1,w0,r0)$

$\Downarrow$

$\uparrow(w)\ \uparrow(r,w);\ \downarrow(r,w,r)$

$\Downarrow$

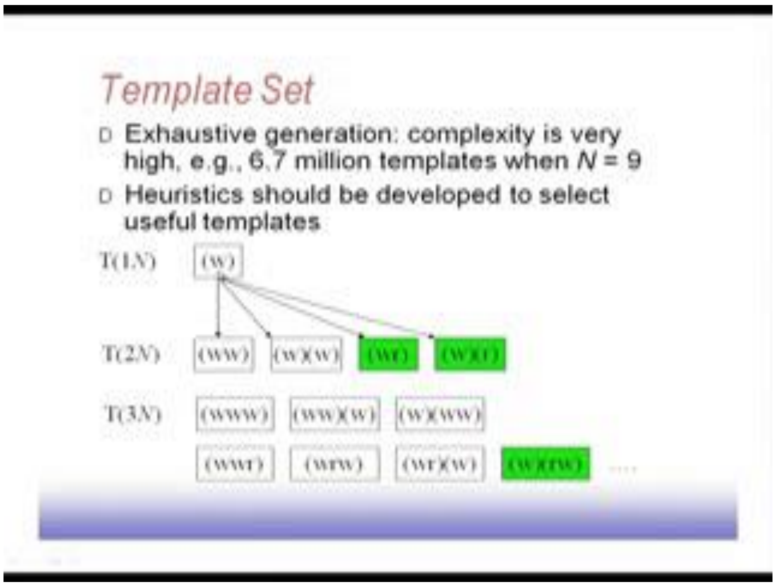
(w)(rw)(rwr)

So, this is the idea. So, there is an approach known as test generation algorithm by simulation or TAGS - T A G S. So, what it does first of all this march templates that we have there abstracted. You see that in this particular sequence, so we can say that whether it is write 0 or read 0, it is not a big matter, but what matter is that it is a write operation and it is a read operation. So, I can abstract it like this as this is write operation, this is a read followed by write, this is read followed by write this is read followed by write followed by read so like that and then you can also drop this arrows - up arrow and down arrows and we can say ok this is the template.

Now, you see that if we can generate this type of templates, so forget about the first two rows that we have here, we just concentrate on the last one. Suppose, we can generate these type of templates, we does not have arrows we does not have this zero-ones. Then I can extend this to something like this I can extend this to something like this I can extend this by is so first it if we going up it may be going down. So, that way I can generate different types of sequences fine. So, this may be up arrow, this may be down arrow, this may be again up arrow or any of this those variants.

Similarly, this read and write operations, so this write may be a write zero or it may be a write one. Similarly, whatever be the write value here, this read value must be that one; and similarly whatever be the read value here, this write value should be compliment of that, so that way I can generate some zero-one combination. So, if I started is so here this particular case it is starting with w 0, I could have started with w 1 also, so that way this particular abstracted template, so it can represent many of the actual march templates. So, what this TAGS will do is that it will generate this type of templates, and abstracted templates and then it will populate those abstracted templates with those zeros-ones, and up arrow, down arrows and accordingly it will generate a sequence of a number of march element or march test sequences. And then I have got that RAMSES algorithm by which you can evaluate the individual march sequences, and we can take the best one from them, so that is what is done.

(Refer Slide Time: 25:08)



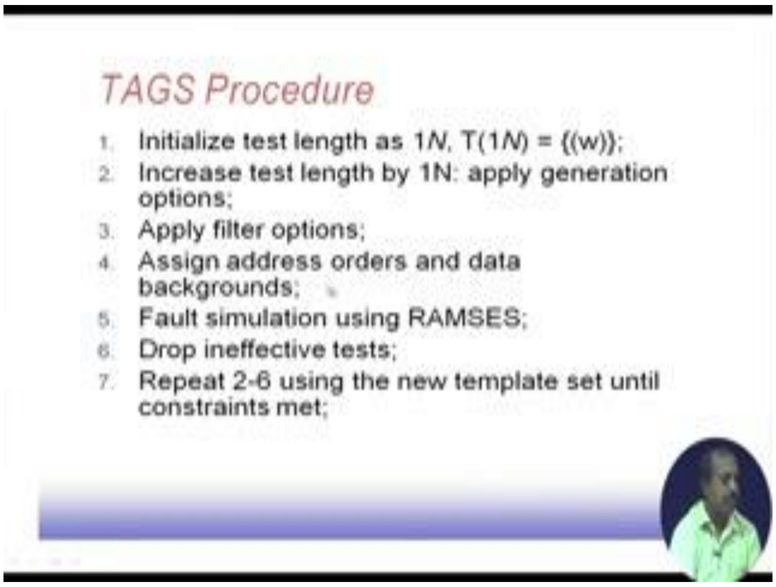
Template Set

- Exhaustive generation: complexity is very high, e.g., 6.7 million templates when $N = 9$
- Heuristics should be developed to select useful templates

So, we start with say it test length of 1 N, so that is 1 N means it has to be write operation because I cannot start with a read operation. So, it has to be a write operation. So, start with a write operation. And then when I am trying to extend the length of this write, so I have to I can do it like this, I can put another write after this. So, I can add another element to this, which is a write or I can in the same element, I can put a read operation or I can put a read as separate a march element. So, these are the different ways by which I can extend this element, this sequence having one in complexity to a sequence having 2N complexity. Then from this again I can do extension. And again I can get a number of search alternatives. Now, out of that is this w w does not have any implication, because to successive writes, so the last write will prevail. So, there. So, nothing to be detected.

Similarly, these w w is this also does not have march elements, because they are all write operation. But this is write followed by read, so this is important because we can write a value and then check the value of value that we have read that we have written. And similarly this w are this is also same thing. So, here actually for every cell, I will do a write followed by read here will do it for the entire memory, I will do the write and then for the entire memory I will do the read operation in a sequence. So, this way we can have different types of templates. So, exhaustive generation complexity is very high 6.7 million templates for N equal to 9. So, we have to use some sort of heuristics for restricting the number of ways in which we can generate the templates.

(Refer Slide Time: 26:59)



TAGS Procedure

1. Initialize test length as 1N, T(1N) = {(w)};
2. Increase test length by 1N: apply generation options;
3. Apply filter options;
4. Assign address orders and data backgrounds;
5. Fault simulation using RAMSES;
6. Drop ineffective tests;
7. Repeat 2-6 using the new template set until constraints met;

So, this is the TAGS procedure. So, initialize with T 1 N equal to single w, and then increase the length by 1 N apply applying the generation options, then we apply the filtering options to filter out the thing, then assign address order and data background. So, basically this address order up arrow, down arrow and data background zero and one. Do a fault simulation using RAMSES. So, ineffective tests will be dropped. And then will some test will survive, so for those test will again try to increase the length by another 1 N operation, another read or write operation, and accordingly again we do the simulation till we are satisfied. Satisfied to the extent that 100 percent fault coverage has been reached or we have come to the situation where the test length was mentioned and we have done up to that much. So, then we whichever the sequence a number of sequences has been generated, so whichever sequences give me the best coverage, so we will accept that one.