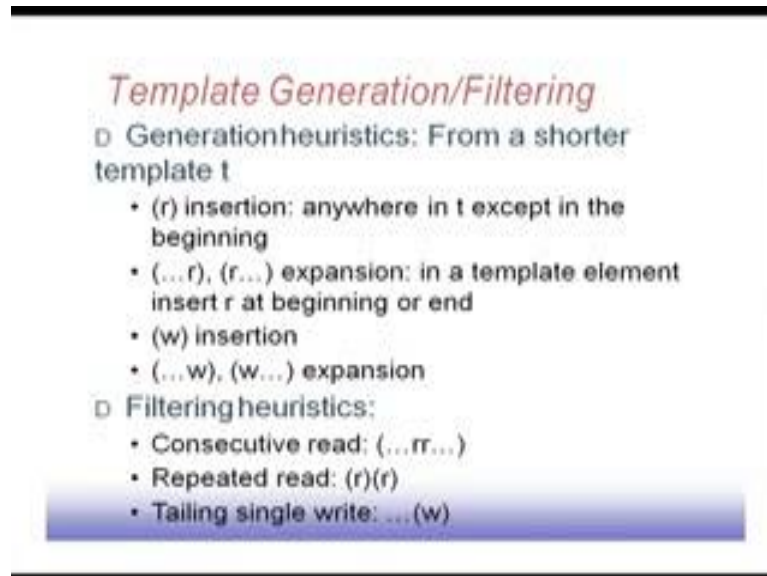


**Digital VLSI Testing**  
**Prof. Santanu Chattopadhyay**  
**Department of Electronics and EC Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 60**  
**Memory Testing (Contd.)**

(Refer Slide Time: 00:22)

---



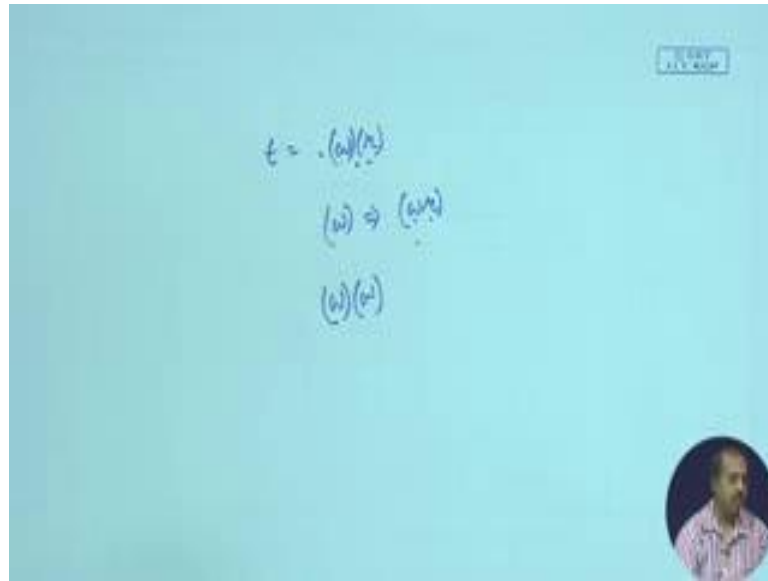
*Template Generation/Filtering*

- Generation heuristics: From a shorter template  $t$ 
  - $(r)$  insertion: anywhere in  $t$  except in the beginning
  - $(\dots r), (r \dots)$  expansion: in a template element insert  $r$  at beginning or end
  - $(w)$  insertion
  - $(\dots w), (w \dots)$  expansion
- Filtering heuristics:
  - Consecutive read:  $(\dots rr \dots)$
  - Repeated read:  $(r)(r)$
  - Tailing single write:  $\dots (w)$

---

So, in the test pattern generation process, the tags algorithm so as we have seen like it will start with some basic pattern, and then it will go on augmenting that pattern set by doing newer and newer memory operations. So, this generation process of this templates, so we can divide it in a two part - one is called template generation, other is called template filtration. So, in the generation part, so if you do not put any restriction then naturally there are so many options that this generation process becomes too costly. So, we follow some heuristics or some procedure by which we can augment this templates. So, what are the augmentations like given a templates, so you can do one read insertion this  $r$  insertion that that is a standalone read that we are inserting. So, this  $r$  can be inserted anywhere in the template  $t$  expecting at the beginning.

(Refer Slide Time: 01:26)



So, your template  $t$  may be we have got a single write. So, in this we can introduce a single this standalone read operation, so like this, so I cannot do it at the beginning, because there is no point doing a read operation before doing any write operation, so this read operation maybe inserted after the write operation, so that is one extension of the template  $t$ . Other expansion can be that we can finish after one template is over, so we can put another template and insert  $r$  at the beginning or at the end. So, in a template element, we insert  $r$  at the beginning or at the end or at the end.

So, if I have got a template like some say  $w$  as we have taken the example, so this can be extended as  $wr$ , so that is this reading, the difference between this one and this one is that. So, here this  $w$  is applied over all the cells first, first all the cells are written and then all the cells will be read. But in this case, the cell that is retain is read immediately and then only  $w$  can go to the writing of the next cell. So, this is the template element insertion  $r$  at the beginning or at the end. So, in within the template, so we can introduce  $r$  at the beginning or at the end.

Similarly, we can introduce a standalone  $w$  operations, and standalone write operation. So, we have got as we are talking, so we have got this template, so it can be extended by another write operation, so that way it can extend the write operation by putting another it can extend the template by putting another standalone write operation. Or we can have expansion, so this template may get expanded by putting a write at the end or putting a

write at the beginning. So, at any point of time, so if we have got a set of templates then those templates can be augmented, so you can expand individual templates or you can introduce new standalone operations of read and write within the template. Now, so that is the generation part.


So, after generation it may, so happen that you can it generates some of the templates which are meaningless like this consecutive read. So, this consecutive read we have got two successive reads, so these are not very much useful excepting when you are trying to test say read disturb faults, so this is not useful. Similarly, this r followed by r, so that is standalone r followed by r, so that is standalone r followed by standalone r, so that is also meaningless, because of the same reason. And the tailing single write, so if you template set stay if your template ends with a write operation at the end, so that has got no meaning, because writing to the memory we are not checking anything. But so for checking we need to have a read operation, so my template must end with a read operation, so that is the other one.

(Refer Slide Time: 04:31)

**TAGS Example (1/2)**

□ Target fault models (SAF, TF, AF, SOF, Cfin, Cfid, CFst), time constraints  $\infty$ :

$T(N)$	Name	March algorithm
1N	$At_1^1$	$\{w0\}$
2N	$At_2^1$	$\{w0\} \uparrow \{r0\}$
3N	$At_3^1$	$\{w0\} \uparrow \{w1\} \uparrow \{r1\}$
3N	$At_4^1$	$\{w0\} \uparrow \{r0, w1\}$
3N	$At_5^1$	$\{w0\} \downarrow \{w1\} \uparrow \{r1\}$
3N	$At_6^1$	$\{w0\} \downarrow \{r0, w1\}$
4N	$At_7^1$	$\{w0\} \downarrow \{r0, w1\} \uparrow \{r1\}$
4N	$At_8^1$	$\{w0\} \downarrow \{r0, w1, r1\}$
5N	$At_9^1$	$\{w0\} \uparrow \{w1\} \uparrow \{r1, w0\} \uparrow \{r0\}$
5N	$At_{10}^1$	$\{w0\} \downarrow \{r0, w1\} \uparrow \{r1, w0\}$
5N	$At_{11}^1$	$\{w0\} \uparrow \{w1\} \uparrow \{r1, w0, r0\}$
6N	$At_{12}^1$	$\{w0\} \uparrow \{w1\} \uparrow \{r1, w0\} \downarrow \{r0, w1\}$
6N	$At_{13}^1$	$\{w0\} \downarrow \{r0, w1\} \uparrow \{r1, w0\} \uparrow \{r0\}$
6N	$At_{14}^1$	$\{w0\} \uparrow \{r0, w1\} \uparrow \{r1, w0\} \uparrow \{r0\}$
6N	$At_{15}^1$	$\{w0\} \uparrow \{r0, w1\} \uparrow \{r1, w0, r0\}$
7N	$At_{16}^1$	$\{w0\} \downarrow \{r0, w1\} \uparrow \{r1, w0\} \downarrow \{r0, w1\}$



So, this is an example, so how this say we have got we started with a single write operation write 0. So, this is r this is a march test with whose length is 1N. So, you have number of memory operation, so if there are if there is a n bit memory then we are doing n memory operation, so this is this has got one template of write operation. Now, when we are extending this w 0 maybe extended this w 0 followed by r 0, so that is a read

extension or we can have one say this one. So, we can extend this by another write operation and then another read operation, so this way we can have this we can generate this march test from starting with the basic template to the successive ones.

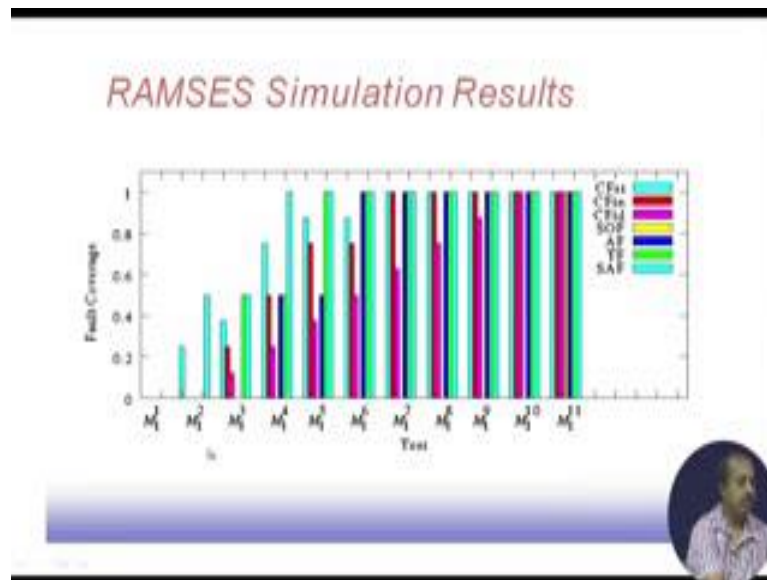
(Refer Slide Time: 05:24)

*TAGS Example (2/2)*

7N	$M_1^7$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \} \{ (r0, w1) \}$
7N	$M_2^7$	$\{ (w0) \} \{ (w1) \} \{ (r1, w0) \} \{ (r0, w1, r1) \}$
7N	$M_3^7$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0, r0) \} \{ (r0) \}$
7N	$M_4^7$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0, r0) \} \{ (r0) \}$
8N	$M_1^8$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \} \{ (r0, w1) \}$
8N	$M_2^8$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \}$
8N	$M_3^8$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \}$
9N	$M_1^9$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \} \{ (r0, w1) \}$
9N	$M_2^9$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \}$
9N	$M_3^9$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \}$
10N	$M_1^{10}$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \} \{ (r0, w1) \}$
10N	$M_2^{10}$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \} \{ (r0, w1) \}$
10N	$M_3^{10}$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \} \{ (r0, w1) \}$
11N	$M_1^{11}$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \} \{ (r0, w1) \}$
11N	$M_2^{11}$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \} \{ (r0, w1) \}$
11N	$M_3^{11}$	$\{ (w0) \} \{ (r0, w1) \} \{ (r1, w0) \} \{ (r0, w1) \}$

So, this is the complete one, so it has it generates go on generating the templates and after that there are this filtering operation that filters out the unnecessary sequences. And only the usable sequences will survive. So, this way you can generate a number of sequences and then the time constraints being infinity, so it will go on generating the test pattern until and unless all these faults are getting covered. So, it will go on generating the different testing templates, so it goes up to 11 N.

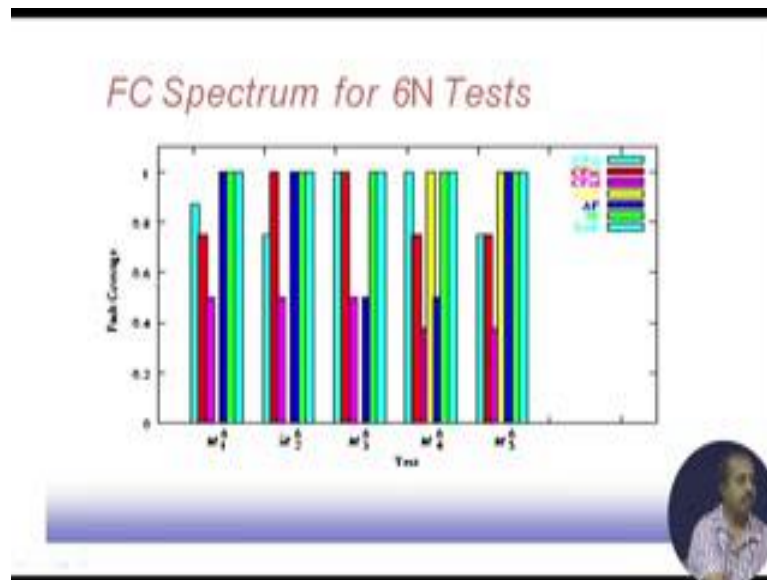
(Refer Slide Time: 05:58)



So, this is actually the coverage information. Like if we have got this M 1 1 that is a standalone write operation, so it is not doing any test, so naturally it cannot have any fault coverage, so fault coverage is 0 for this. Then this M 1 2, so this has got a fault coverage of this one this stuck-at-fault, stuck-at-fault coverage is there, so it is detecting the stuck-at-fault; similarly this is detecting this M 1 3, so it is detecting this much of stuck-at-fault and other faults are also there.

So, different color bars are identify, so interestingly as you are increasing the length of the test, so number of operation read-write operation that we do, so more and more faults will get covered. So, when we come to this M 1 11, so you see that that is the number of read-write operation one, so at that point all the faults are getting covered, so all the faults are getting covered, so you have reach the 100 percent coverage, so we can stop at this point. Our test generation process can stop at this point provided these are the fault models that we have to cover. So, if we are looking for only say stuck-at-fault then we can stop at this point itself, so M 4, we can stop, so only four length test sequence will be sufficient.

(Refer Slide Time: 07:15)



So, this is the another fault coverage, if you take only a single test length, if your test length is restricted to say 6 N, so then what happens is that then also with 6 N restrictions also, there are several test sequences that can be generated by that tags procedure. And the list that we have seen previously, so in that this M 1, M 2, M 3, M 4, so these are various six length sequence. And you see that all of them do not have the same level of coverage for all the faults. So, naturally we will like to have the those tests, which will be covering most of the faults, so accordingly you can select any of this patterns M 1, M 2, so M 1 is really poor, because some of the faults are not covered and all that, so it is not that good, so whereas, these are better. So, this way you can get some selection from the tests.

(Refer Slide Time: 08:13)



*What Can BIST do?*

- What are the functional faults to be covered?
  - Static and dynamic
  - Operation modes
- What are the defects to be covered?
  - Opens, shorts, timing parameters, voltages, currents, etc.
- Can it support fault location and redundancy repair?
- Can it support BI?
- Can it support on-chip redundancy analysis and repair?
- Does it allow characterization test as well as mass production test?
- Can it really replace ATE (and laser repair machine)?
  - Programmability, speed, timing accuracy, threshold range, parallelism, etc.

Next, we look into memory BIST. So, basically what we have said is that, so there is a march test sequence that we need to apply to the memory for testing it, but for how do you apply those test pattern, are we applying from outside. So, if we are applying from outside then we have to have all those controls, so we have to access individual cells and then we have to same individual patterns and all that. So, BIST is a better option for memory testing, because this is BIST can be integrated with the memory itself and once it is signaled to start testing, so it can go on doing that it can generate the pattern as per the template the read-write requests. And it can check whether it is ok or not, so ultimately it can result in a go, no go solution telling whether the memory chip is ok or not.

So, what are the issues in BIST design. First of all, what are the functional faults to be covered, so which faults you are going to cover, so that is an important issue, so there may be static or dynamic faults, can be operation mode, so what are the different operating mode that the RAM, the memory has. What are the defects like opens, shorts, timing parameter, voltage, currents what are the defects we are going to cover. Can it support fault location and redundancy repair? So, if does it have the capability to locate the faulty location, actually what happens is that memory as per the technologies concerned, so this is an array of same type of cells. So, if one cell is faulty we can replace that cell by another one, so my address decoder can be modified, so that it refers to a different location when that particular address is generated, so that way if I can do

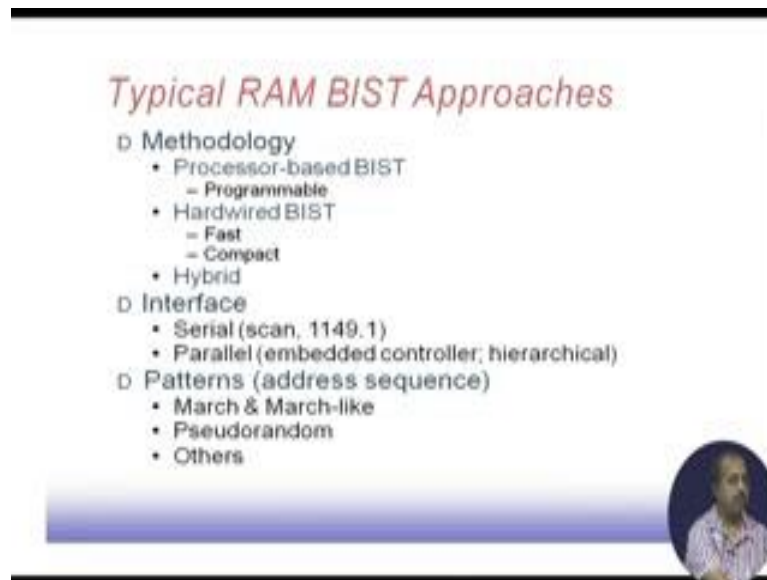
this. So, if I can support this correction this repair part, and also this redundancy because some of the array locations may be redundant, because of this duplication.

Does it support burn-in? So, actually this burn-in is that when this design first comes into a existence, so it is kept on for quite some time and then if there is a problem then many of the wrongly manufactured module, so they will die down, so that way this our yield can be improved by removing this burn-in pattern burn in chips. So, this can it support this burn in detection. Can it support on-chip redundancy analysis and repair, so that is also important. So, does it allow characterization test as well as mass production test? So, characterization test is that can check the memory probably or mass production test is those test that does not do thorough check, but it is much faster than mass production though all faults may not be covered, so does it fault allow both of them.

Can it really replace ATE? So, ATE and the laser repair machine, so this is actually the thing like if we detect that there is a fault, so then many a times what is done is by laser technology, we can rectify some of those faults. So, that way there may be some laser repair machine, so that can take care of this faulty chips, and they can introduce some connections, so two lines are short, so it can put a laser beam there and remove the connection between them. So, like that it can do many operations, so this can it, so this can it ATE plus laser machine, so can it may be avoided if I use a BIST. So, the necessity is the programmability, speed, timing accuracy, threshold voltage and the parallelism all this things. So, by when I am talking about a BIST, so these are the questions to be answered like which type of features will the BIST provide.



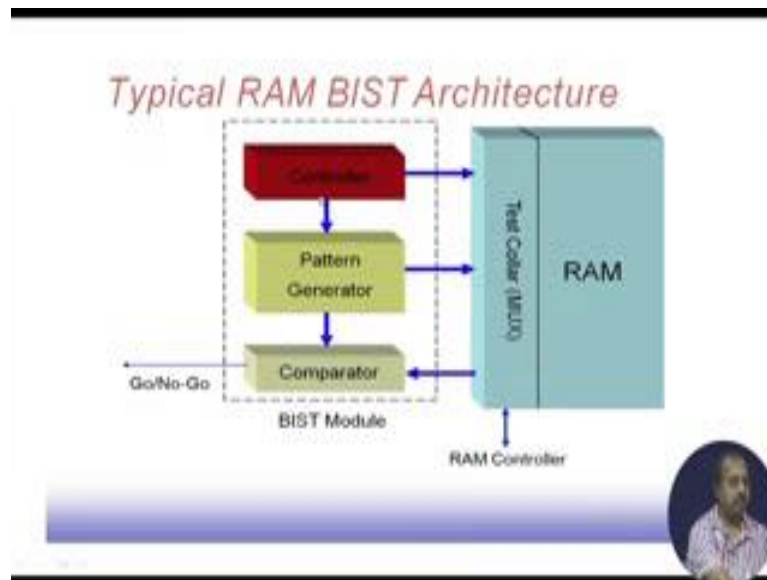
(Refer Slide Time: 12:03)



So, typical RAM BIST approaches are like this, so there can be two different methodology or three different methodology. One is processor based BIST, so processor based BIST, so they are programmable, so there will be an underline processor integrated with the memory, and it will be generating the pattern. So, since it is a processor, so you can program it to generate a different types of patterns, so that way it is good because you can cover a number of different fault models.

Then hardware BIST, so hardware BIST will be faster than normal this processor based BIST, because this software part is not there, so it is fast and compact and hybrid is mix of these two, some part is hardware and some part remains in the processor. Then the interface whether it is a serial scan or 1149.1 the boundary scan technology or it is parallel, so they can is then embedded controller and hierarchical structures like that. So, patterns the march march-like patterns or pseudorandom patterns or others, so what are the different types of patterns that the RAM BIST will generate, so these are the different issues while we are discussing about RAM BIST.

(Refer Slide Time: 13:15)



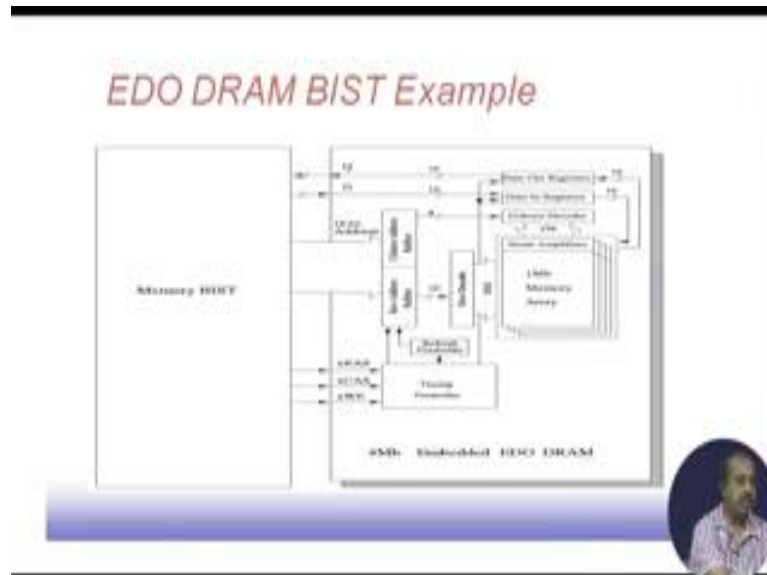
So, this diagram shows a typical architecture of RAM BIST. So, this is the ram, so that has got a test column basically some multiplexers, so that it can get the address and data lines from this RAM, from that BIST module as well as it should get address and data lines from the processor for normal operation of the RAM. So, they are naturally there will be some multiplexers that will be selecting between the two. So, there is a pattern generator, so this pattern generator modules, so this will be generating different march patterns or if it is a march based test or if it is pseudo random test will generate pseudo patterns like that.

And there is a comparator, suppose I am generating a read zero, read zero is the pattern that is given. So, this read zero is coming to the comparator as well and it is going to this cell, so that particular location is read and this value is available where comparator, so comparator will check whether the value coming from the pattern generator is same as the value coming from the memory cell. If they are not same then there is a problem, so this location is faulty. So, faulty is detected, so that is why it generates a go, no go type of signal.

And if it is a write operation of course, comparator does not have to do anything, so the corresponding location will get retained. So, corresponding to the read operations only, this comparator will check the value available from the pattern generator and the value

available from the RAM to see whether they are correct same or not, so these are typical RAM based architecture.

(Refer Slide Time: 14:51)



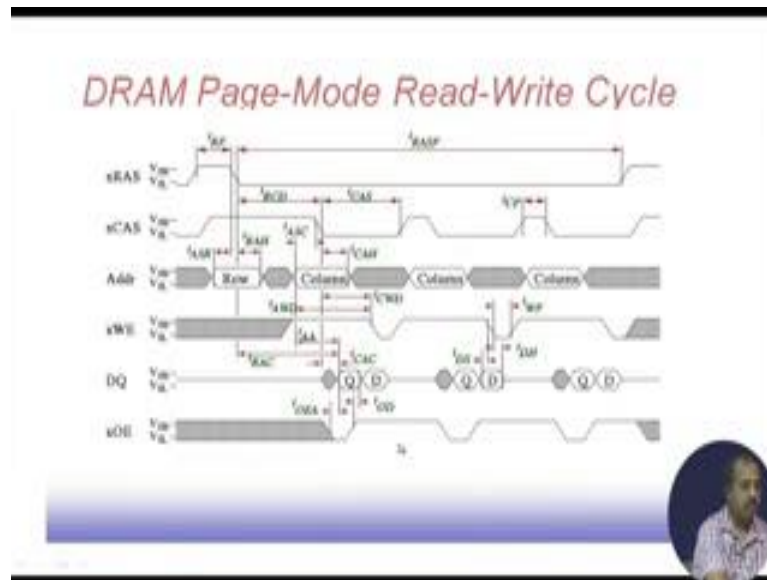
So, these actually shows a DRAM structures. So, this is the memory array, thus got sense amplifiers and all that. And as we know that the memory is arrange in a 2D fashion, there will be a row decoder and a column decoder. So, row decoder these are 1024 rows, each of 256 columns, so they will be generating any address that we have, so this address has got row address part and the column address part. Row address part will comes to the row decoder, column address part comes to the column decoder and they are actually selecting a particular location.

Now, if you want to write something then this D lines, so they actually get the data 16 bit data line, so that is coming to the data in registers. Similarly, if you want to read something from the memory cell then this 16 bit data is available through the Q line. So, normally these lines are coming from the processor, but in a memory BIST environment, so they can also come from the memory BIST. So, the test color is not shown here it is basically that we will have the multiplexers at all this points.

So, this row address selector, column address selector and writable and these are some timing controls that are given. So, in this row address is putting to this line, then this row address selector is turned on. Similarly, this column address selector will turn on the column address. And this timing is also controlled by the refresh controller, so since it is

a DRAM, so there is a refresh controller which also generates the address, so that is there. So, we got several sources of this address part and accordingly it will be generating they need to be multiplexed, so that they can be applied to the RAM.

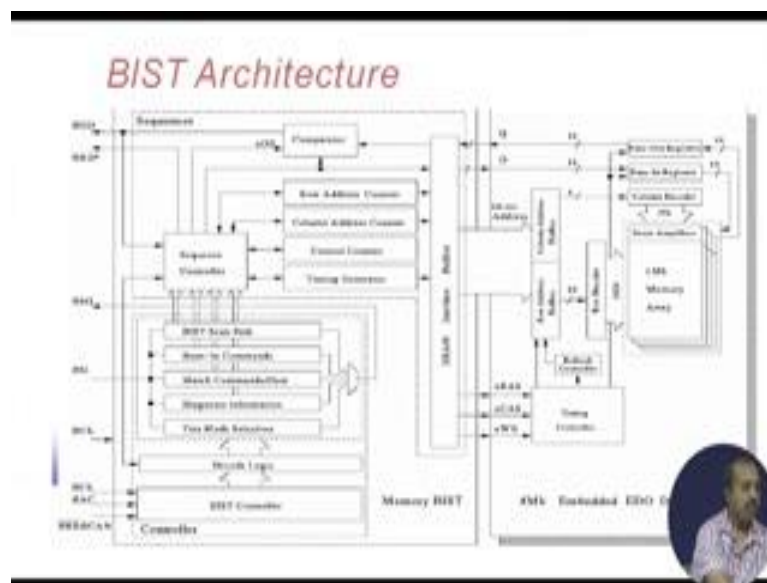
(Refer Slide Time: 16:39)



So, this is the operation. So, initially this row address selection when it is making a transition from high to low, the row address is provided at that time that gets latched. Similarly, when the column address selection line makes a transition from high to low this column get selected. So, whenever this column address selection line is making a transition high to low, this corresponding column address is selected. Then this write enable signal is given, when this write enable signal is low making a transition from high to low then the D value is available, so that will be retain onto the, so RAM cell.

And if it is just reading the value then of course, when this write enable is high, once the address is that been put this value is available on the Q line. Many a time what happens is that this D and Q lines though it is shown separately; a here D and Q lines are shown separately, but they may be together, so they are written as a D Q, so this D, Q is written like this. And this OE line, so output enable line, so this will be when this makes a transition from low to high, so at that time this Q value becomes available onto the bus. So, this way this DRAM operation takes place, so it can read an entire page in this fashion by generating different column addresses for all the columns for a particular row it will generate the data.

(Refer Slide Time: 18:05)



Now, for the BIST as for the BIST architecture is concerned, so this part is that EDO RAM, so this part is the BIST part. So, this has got several components in it, the BIST controller, decode logic, test mode selection, so march, so you see some of them like BIST scan paths are there then burn-in commands, march commands diagnosis information, so they are all going to the sequence controller. And the sequence controller accordingly generating this row addresses counter, column address counter, control counter timing generators and all that, so that this addresses will be put onto this interface buffer and they will go to the row address buffer, column address buffer etcetera.


Similarly, when the values are read, so they are coming to the comparator. So, comparator has got entry data available from the sequencer as well as this memory cell, so they will be compared and accordingly it will give a go, no go type of signal. So, there are many more signals, so I am not going to the much detail because that itself is quite complex, but essentially what means is that the controller, so it is design to provide all the read-write instructions, and this addresses to the memory getting its content change for faults and all that.

(Refer Slide Time: 19:26)



*Concluding Remarks*

- ▷ BIST is considered the best solution for testing embedded memories:
  - Low cost
  - Effective and efficient
- ▷ Further improvement can be expected to extend the scope of RAM BIST:
  - Timing/delay faults and disturb faults
  - BISR and BISR
  - CAM BIST and flash BIST
  - BIST/BISR/BISR compiler
  - Wafer-level Bi and test
    - Known good die



So, to conclude, so BIST is the best solution for testing a embedded memories, because the cost is low and it is effective and efficient. So, if you are using ATE, then ATE for testing so many locations, so many memory locations it will take lot of time. Then further improvement can be expected, so we can have some timing or delay faults and disturb faults, so they can be covered. Built-in self design, and built-in self repair, built-in self diagnosis BISR; and built in self repair, so these are the two directions where works are going on, because diagnosis will diagnose the particular cell which is faulty and repair will try to repair it in terms of may be replacement and all that.

Then this content accessible where memories so for then BIST is a challenge because that the way they are accessed is different from this DRAM and for a flash memory also they are difference. So, accordingly we will have to have different type of BIST technology. Then BIST BISR, BISR compilers have to be there, because they in that case they will get automatically introduced into the memory. So, if you say that this memory should be BISTED then the BIST will get introduced in to the memory. Wafer level burn-in and tests, so that also has to be there. So, accordingly we will come to once we know that this dyes are not good, so you can get some information about the known good die, how many dyes are good in your in entire wafer, so that can be collected, so that gives an indication about the quality of production.

So, this memory testing as far as algorithms are concerned, so it is dependent mostly on the march sequence generation, and we have seen that unlike other test pattern generation algorithm say ATPG, so here it is based on simulation. So, you can have your own test generation procedure, so you can think about anyway of generating this read-write commands and then you can take help of this simulator MCs to generate that to check whether the faults are getting covered properly or not. And accordingly you can get a confidence about the quality of the test generation algorithm that we are proposing, so that is on the test generation site.

On the test application site, we have got this BIST, so this BIST can be used for generating this test pattern applying this test patterns to the memory array. So, combining these two, we can get effective solutions for this memory testing.