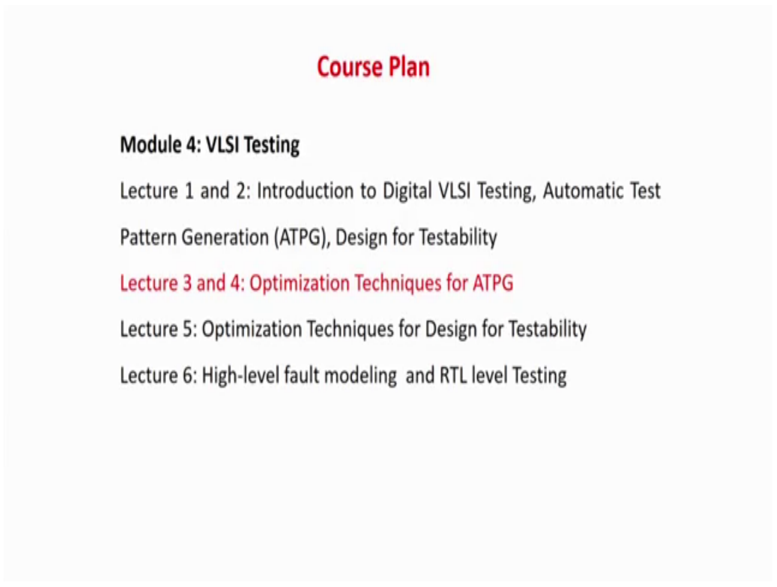
Optimization Techniques for Digital VLSI Design

Dr. Chandan Karfa
Dr. Santosh Biswas
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Lecture – 16
Optimization Techniques for ATPG [Part II]

Hello everyone at present basically we are in the module number 4 on VLSI testing. So, in the last lecture basically we were talking about optimisation of the test patterns that is ATPG algorithms, how we can develop or generate good quality test patterns; which are optimised with certain parameters. Like some for some case how ATPGs or test patterns can be generated for very large circuits that is one way of optimisation other ways of optimisation.

(Refer Slide Time: 00:52)



Course Plan

Module 4: VLSI Testing

Lecture 1 and 2: Introduction to Digital VLSI Testing, Automatic Test Pattern Generation (ATPG), Design for Testability

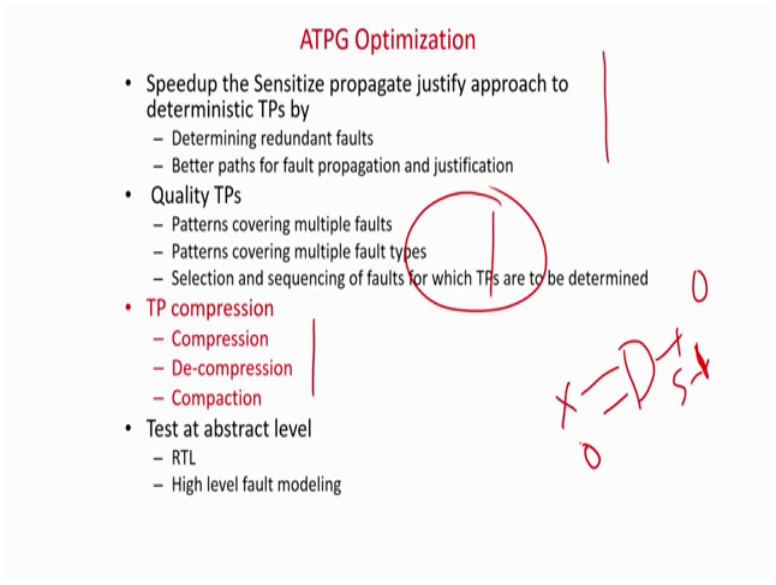
Lecture 3 and 4: Optimization Techniques for ATPG

Lecture 5: Optimization Techniques for Design for Testability

Lecture 6: High-level fault modeling and RTL level Testing

As we were looking in last lecture that how we can develop good quality test pattern such as one test pattern can test stuck at falls as well as more advanced falls like delay faults, bridging faults, detect faults etcetera. So, basically we will now today also we will continue in this direction where we will be finding out how we can develop more optimised test patterns by compressing them.

(Refer Slide Time: 01:10)



ATPG Optimization

- Speedup the Sensitize propagate justify approach to deterministic TPs by
  - Determining redundant faults
  - Better paths for fault propagation and justification
- Quality TPs
  - Patterns covering multiple faults
  - Patterns covering multiple fault types
  - Selection and sequencing of faults for which TPs are to be determined
- TP compression
  - Compression
  - De-compression
  - Compaction
- Test at abstract level
  - RTL
  - High level fault modeling

I will as we will go we will detail out more on that. So, basically we can see that how test patterns can be optimised. One way is that you can have a very good algorithm to find out the test patterns that is; how can we can handle large scale or complex circuits. Secondly, is how we can generate good quality test pattern; what do you mean by good quality test patterns? Test patterns which can detect multiple faults, which can take faults of advanced types apart from stuck at faults like bridging, delay in detect etcetera.

So, basically last lecture was mainly focused on that where we have seen how to generate test patterns which can detect stuck at as well as bridging as well as delay and there can be certain other important formulas which can be detected, but the main goal is that how we can develop test patterns so, that it can handle multiple faults.

The first part is the first topic is basically how we can go for large how we can I mean is a generate good I mean advanced or make the ATPG algorithms intelligent so, that we can detect faults in a much faster manner. So, that by that way you can find out whether path when we can sensitised the fall better path by which we can propagate the falls so, that whatever values you require can be justified. So, there are lot of heuristic based on that we have not emphasized much on that, because that is actually mean the applied when the circuit size are small otherwise I mean this is a part of traditional VLSI design; so, VLSI testing courses.

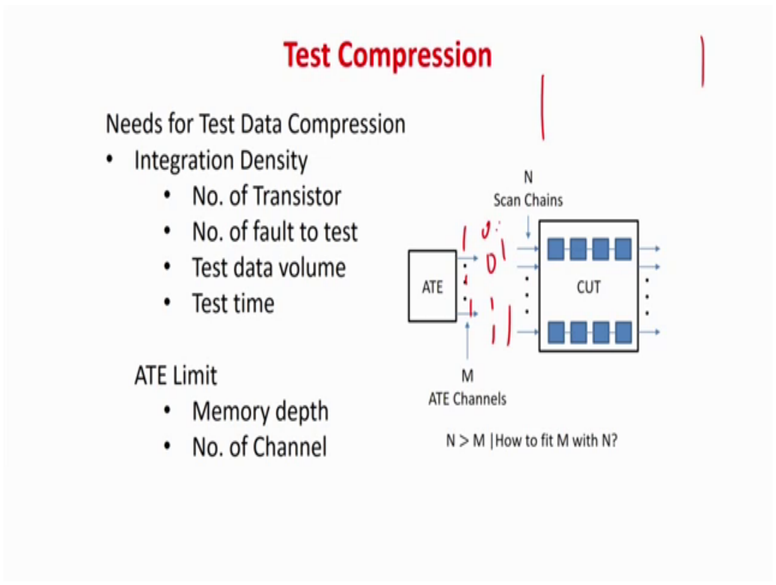
So, you can take of any textbook or any material on care we will find out lot of heuristic which tell you how to find out booth path for fall propagations, synthesization and justification. So, that you get good quality for in the in the way that they generate the test patterns faster. And next important part, basically which you are going to see is something called how can I compact the test patterns? Because you may have lot of test patterns, but one important thing you have to observe that 90 percent of the values of Boolean variables which are which are obtained after an ATPG algorithm are x; I will give you one simple example and a you want to detect the stuck at 0 stuck at 1 fault say.

So, stuck at 1 fault; that means, you have to have a 0, 1 here how we can get 0 you can have 0, 0, 0, 0 or 1, 0 these are the three patters which I can find out to detect this fault, but I can simply write it in this fashion that I can have to detect a stuck at 1 fault I can have x 0, that is this value can be anything I just need to set one value as 0; and I can detect the fault.

So, if you take a large or general circuit practical example such Xs in the test pattern is which can be either 0 or one occupy more than 90 percent of the variable that is very interesting observation to be have found out. Now using those Xs can we compact the test pattern. So, if the input bit width is a 100 and we are there are lot of do not cares variables, how can we compact the test pattern? So, in that case what is going to happen? It will be optimised in the sense that you have to have a you can apply a less bit width patterns by a tester so, that you can save on the tester part

We will detail out by examples; so, that is the ideas of test compaction or ATPG optimisation that instead of having very wide test patterns can be make it small in width. So, smaller in width means less number of channels from the ATE etcetera. Another level is which I have told you that can we do automatic test pattern at a higher level of abstraction which we will see further down there since in the in the in this module, but today we will be concentrating on how we can optimise test pattern by compressing them.

(Refer Slide Time: 04:48)



Test Compression

Needs for Test Data Compression

- Integration Density
  - No. of Transistor
  - No. of fault to test
  - Test data volume
  - Test time

ATE Limit

- Memory depth
- No. of Channel

N
Scan Chains
ATE
CUT
M
ATE Channels
N > M |How to fit M with N?

We should take an example. So, for example, let this be a circuit, there are some input pins. So, there are say N scan chains or I can tell you that N input pins, because scan chains has to be loaded. So, if there N scan chains or N inputs so, N patterns parallel you can load and an ATE is called an acute at automatic test equipment you just say M chains. Generally it happens, that N is always smaller than M, use the circuit may 102824 pins or more number of inputs higher number of inputs like 100 inputs 200 inputs may be there to test the circuit.

Generally, why we call N is less than M, N basically N is the number of parallel channels from which you can apply a pattern store circuit and M, basically and an M is ATE channels; ATE can have lot of channels, but you have to pay. So, for every channel I use and as a speed you use you have to pay, because ATE services are given by vendors. So, ATE may have some 200, 300 channels, but more number of channels you use more amount of money you have to pay to the vendor.

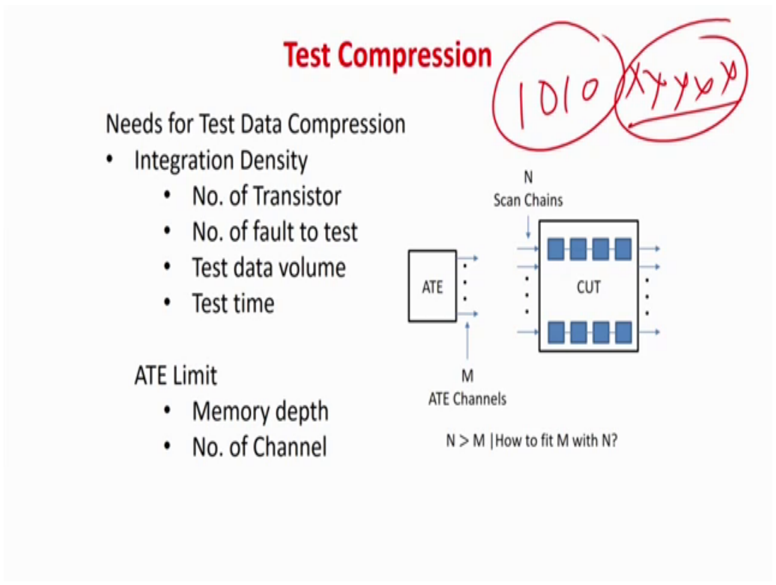
So, what you always try to do is that; we have to optimise in such a fact that out test width is less may be by 10 patterns, 20 patterns, suitable to do, because more test patterns you want to apply in the sense is may be 1, 1, 1, 1, then next pattern is 1011 so, three patterns are there. So, more number of patterns means, more time will be using the ATE. So, more amount of money you have to pay, and also if you use more number of channels of course, their price will raise.

So, what do you always try to do compare the number of test patterns as well as compact from this side, this one if you reduce if you reduce if you reduce number of test patterns? Generally you may lose on coverage, but generally we have already discussed in the previous lectures; that there are lot of faults which are equivalent and basically stuck at one of the beauty of stuck at faults is that due to faults dominance and equivalents, they if their number of stuck at faults or number of patterns are not too large even for a very large circuit; that means, basically the number of patterns to be applied like, 110101 some patterns are there this is generally not too large or in other words I should not say that is not too large.

In other words if you try to optimise you reduce the number of test patterns in that fashion basically you are going to lose on coverage, because ATPG algorithms will always give you try to give you the minimum amount of test patterns required to cover all the faults.

See for example, I have 100 faults ATPG algorithm will give you some 20 patterns or 10 patterns which will be which will be able to cover all the faults and there are algorithms which will give you the minimal amount of factors to do that; if you further reduce it you may lose on coverage. So, generally compaction on this manner or reducing the number of test patterns is are different problem, then test compression which are trying to do to they. So, reducing the number of test pattern means basically I have to find out the minimal set we can cover all the faults modelled. So, it actually comes under automatic test pattern generation and is the well studied area, but now today you are seeing out compaction. For example, if you have 1010 and say all other are XXXX do not required to be anything.

(Refer Slide Time: 07:35)



Test Compression

Needs for Test Data Compression

- Integration Density
  - No. of Transistor
  - No. of fault to test
  - Test data volume
  - Test time

ATE Limit

- Memory depth
- No. of Channel

1010
Xyyyxy
N
Scan Chains
ATE
CUT
M
ATE Channels
N > M |How to fit M with N?

So, what I can do is that; I can put some arbitrary circuit inside which will generate these arbitrary values, because they do not require to purchase ATE channels for them only I use this domestic values from the automatic test equipment. So, this a some arbitrary values I can just put 0s and 1s, I do not require to put channels that is the basic idea of and it test pattern based compression. So, the idea is that number of channels and memory depth.

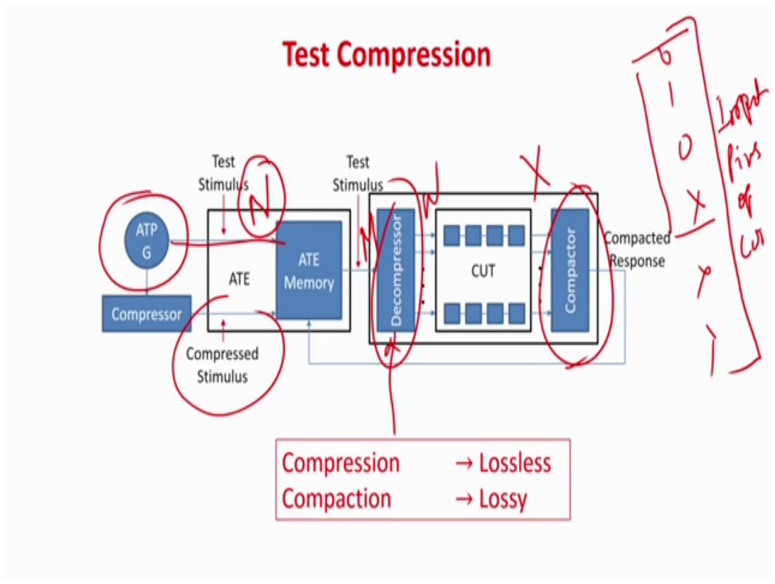
So, what is memory depth? Memory depth means how many test patterns you are applying this directly falls under that. So, more number of test patterns more number of depth and number of channels, as I told you more number of channel means you have more number of pins require more number of parallel loading you can do, but that will actually pay the cost. So, memory means what are the numbers of bits you are using? So if you in the more number of bits more number of patterns and more number of bits per pattern will increase the memory.

So, for memory in the ATE you have to pay; at but memory is not that expensive, because nowadays memories are cheaper more important is how many physical channels you are occupying. So, that if you are using 10, the other 10 can be used by other vendor, the other 20 can be used by some other company. So, you have to pay less.

So, memory is the constraint, but we are more interested in the number of reducing the number of channels. So, basically what we are trying to do; we will be doing basically even if N is there, we will try to limit N as less as possible, but still we want to apply all the test pattern, but what will leverage out? They do not cares bits in the automatic test patterns generated or the patterns which are generated. So, in gist what you are trying to do 100 input lines of the circuit may be ATE lines can be will be do not cares or the 20 lines we have to give justify means exact values for applying the test pattern.

So, can we how can we find out that 20 and what we have to apply in that 20. So, that you use instead of 100 pins from the ATE you use only 20 pins from the ATE. So, that is basically we are trying to do in this, because I told you reducing the number of test patterns falls under ATPG generation and in fact, ATPG algorithms are well optimised; so that they do not generate redundant kind of test vectors, they generate the mostly minimal amount of test patterns require to test all the faults.

(Refer Slide Time: 09:54)



Again describing pictorially; so basically this is your ATPG algorithm which you generate the test stimulus, that is minimum amount of test patterns require to detect the fault, but it will have 010 XXX ATPG is not binding the number of bits; as what are the input number of channels for the ah scan chain or what are the inputs of the chip to be tested that is the bit. So, that is the pin in input pins input pins of cut that is circuit under test. So, that many pins it will that many width will be the of the vector it will generate. So, that is the general ATPG, because already I have discussed. So, that if you are having a stuck at 0 fault. So, stuck at one fault over in the and gate you will generate X 0 or something, but this bit will size will be 2 ATPG is designed to generate this vectors for all the input pins of the circuit under test.

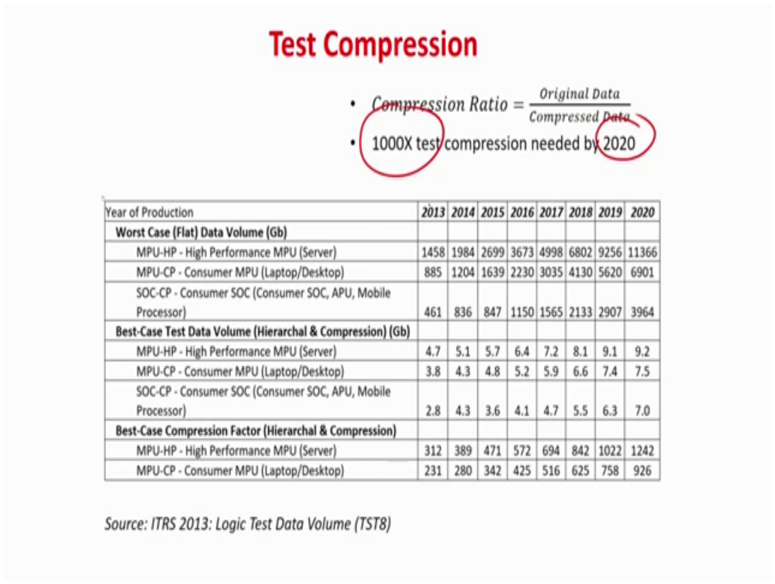
So, now basically let us see what we are trying to do over here. So, this is your ATPG algorithm it is generating some test stimulus. Now this ATE memory. So, what you can do basically some part ATE memory some part of some patterns we can directly put into the memory and some part some patterns, which can be compressed; because I told you if you have a pattern with there is no Xs no do not cares that has to be directly put into the memory, but for most of the (Refer Time: 11:13) 99.9 percent of the vectors will be can be compressed. So, compressed means I will reducing the width and I will put it into the ATE memory.

So, now, you can see basically this width may be equal to N this width may be equal to, because N number of width is there, but then I will it will be compressed; now it will be actually equal to M. So, M is much smaller than N. So, you are using less number of width of the ATE memory, but now this is again, because practically I have to apply, then to this vectors through the cuts circuit under test. So, there is a decompressor circuit in your in your chip which will take this compacted pattern compressed pattern and apply it to the cut.

So, that this is the circuit you have to put in all your Ics, but we have been found out that now design cost is not very high. So, you this also called design for testability. So, you can have this extra compressor and decompressor circuits and still if you reduce the number of pins of the ATE it is a profitable business. So, tried the counter in duty, but it is true basically that even, if you have some extra circuitry, but reduce on the pin outs of the ATE require reduce on the test time of the ATE you were going to save on money and the output is something called a compactor, because of course, the output can also be z X huge number of outputs can also be there, but again we are not going to take all the compactor response or the direct responses from the cut and put in the ATE memory analyse it, will not be a very good idea they we will be going for a compaction.

Basically, slight difference compression is lossless so, when you are taking having a test vector we will compress in such a manner. So, when we decompress it exact value will get we will not have no loss in information, but compaction means is the lossy. So, that the output vectors we will compress we will we will call compaction. So, the output response to be compacted we will give you the examples and then we will feed it to the ATE memory, but in this case it will be a lossy compaction. So, sometimes then be aliasing.

So, aliasing means fault is there, but because of this compaction you may think that it is normal, but there are lot of techniques still it we can reduce the width of the output response and the aliasing are not very high. (Refer Slide Time: 13:19)



Test Compression

- $Compression\ Ratio = \frac{Original\ Data}{Compressed\ Data}$
- 1000X test compression needed by 2020

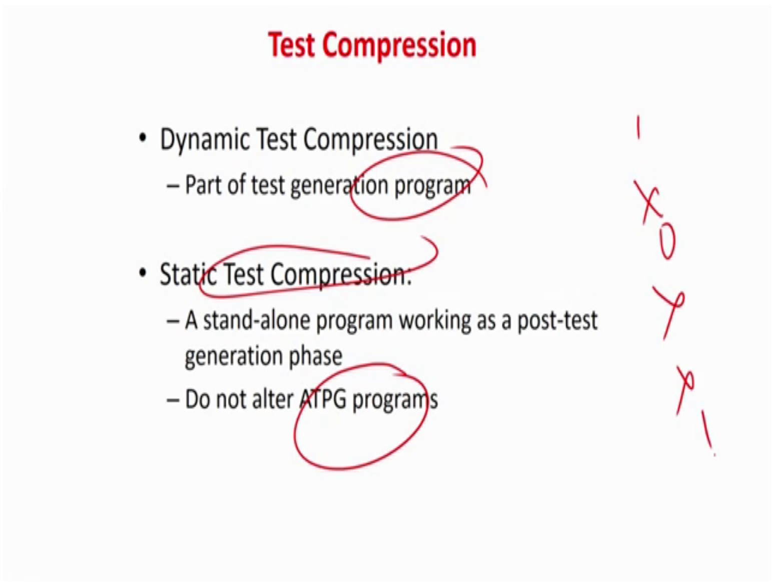
Year of Production	2013	2014	2015	2016	2017	2018	2019	2020
Worst Case (Flat) Data Volume (Gb)								
MPU-HP - High Performance MPU (Server)	1458	1984	2699	3673	4998	6802	9256	11366
MPU-CP - Consumer MPU (Laptop/Desktop)	885	1204	1639	2230	3035	4130	5620	6901
SOC-CP - Consumer SOC (Consumer SOC, APU, Mobile Processor)	461	836	847	1150	1565	2133	2907	3964
Best-Case Test Data Volume (Hierarchal & Compression) (Gb)								
MPU-HP - High Performance MPU (Server)	4.7	5.1	5.7	6.4	7.2	8.1	9.1	9.2
MPU-CP - Consumer MPU (Laptop/Desktop)	3.8	4.3	4.8	5.2	5.9	6.6	7.4	7.5
SOC-CP - Consumer SOC (Consumer SOC, APU, Mobile Processor)	2.8	4.3	3.6	4.1	4.7	5.5	6.3	7.0
Best-Case Compression Factor (Hierarchal & Compression)								
MPU-HP - High Performance MPU (Server)	312	389	471	572	694	842	1022	1242
MPU-CP - Consumer MPU (Laptop/Desktop)	231	280	342	425	516	625	758	926

Source: ITRS 2013: Logic Test Data Volume (TST8)

So, we will now go into depth. So, it has been found out by means ITRS data in 2013 that by 2020, 1000X compression will be required; that means, circuit should be. So, large in nocs and SOCs, but with the increase in size of the circuit the input pins will also raise or the input channels will also raise I should have call it; basically input pins basically in this they are actually scan chains or scan inputs as well as your basically normal combinational inputs, where we can apply the test patterns then will also raise, but with the raise in the circuit size it is very interesting that the number of Xs or do not cares in the test patterns will also raise.

So, there will be 1000X compression will be needed by 2020 or it will be possible, because otherwise ATE cost will become too high and it will not be a profitable business, but again if you are compressing. Then of course, you have a decompressor in a compactor inside a circuit, but as I told you it is very counter iterative, but it is true that even with this extra DFT circuitry, but reducing the ATE time and ATE memory you are making a profitable business that is the module.

(Refer Slide Time: 14:21)


Test Compression

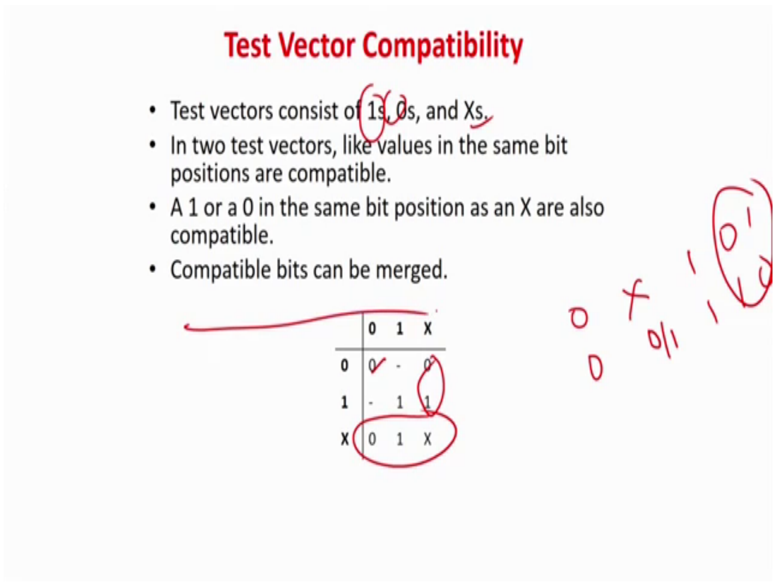
- Dynamic Test Compression
  - Part of test generation program
- Static Test Compression
  - A stand-alone program working as a post-test generation phase
  - Do not alter ATPG programs

So, there are two type of compression one is called dynamic compression one is called static compression. So, dynamic compression means you have to figure the; it is a part of the test generation program embedded in the algorithm. So, in this case you have to motivate or modify ATPG algorithm. So, that you have more Xs more do not cares and also in a very compatible manner as, I will give you the example or in that case you have to a in other words in a layman's language we have to direct your ATPG algorithm. So, generate this vectors which are more compressible. So, this is basically called dynamic test program generation.

But not very popular, because of the idea that if you just use because our ATPG algorithm also should be faster, because you spend too much time on generating the test patterns and then, compressing is not a good idea and the duty. As I told you that if you take a very large circuit also and also take general ATPG algorithm the number of do not cares are very very high. So, in that case you do not itself require to mistreat the ATPG algorithm to generate circuit switch or test patterns with more number of x not required they themselves are basically there so, therefore, dynamic test compression is not very popular basically people go for static compression.

So, static compression means you have normal ATPG algorithm do not alter this; whatever you get you will have anyway you will have get lot of Xs you will have in this vectors you use them to compress it that is called actually post generation phase we generally go by that.

(Refer Slide Time: 15:40)



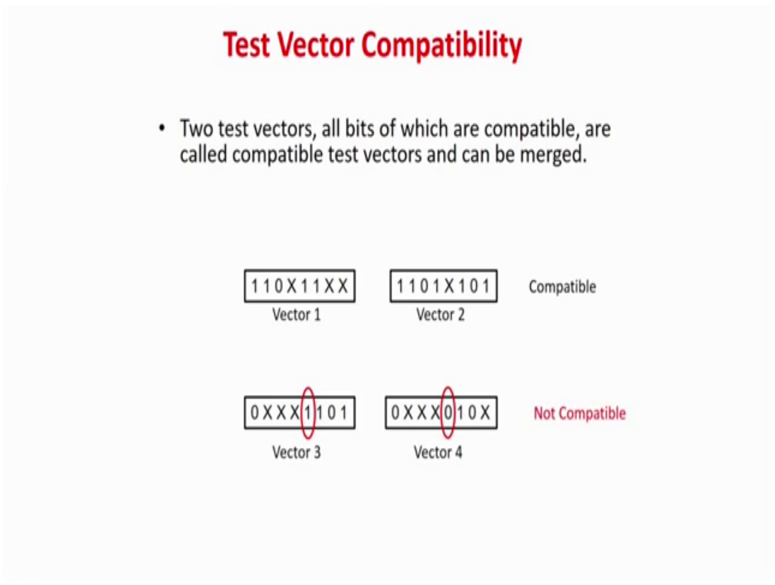
Test Vector Compatibility

- Test vectors consist of 1s, 0s, and Xs.
- In two test vectors, like values in the same bit positions are compatible.
- A 1 or a 0 in the same bit position as an X are also compatible.
- Compatible bits can be merged.

	0	1	X
0	0	-	0
1	-	1	1
X	0	1	X

Now, there is something called vector compatibility that is very very important to understand the compression. So, you have 0s, 1s and something called X, that is do not care it can be anything. So, if you see 2 vectors are there if both of them are 0, 0 it is compatible ok; and if you have X anywhere and it may be 0 or 1 anything it is compatible 1, 1 and compatible 0, 1 and 1, 0 they are actually non compatible very obvious. So, X can be anything. So, you can see that 0, 0 is the comma 0, 0 you can put 0, 0, 1 non compatible hyphen 0 X you can put it is a compatible business 1, 0 non compatible 1, 1 will be 1, 1 X will be one they are all compatible. Similarly, X can be compatible everything so compatible bits can be merged very obvious.

(Refer Slide Time: 16:25)



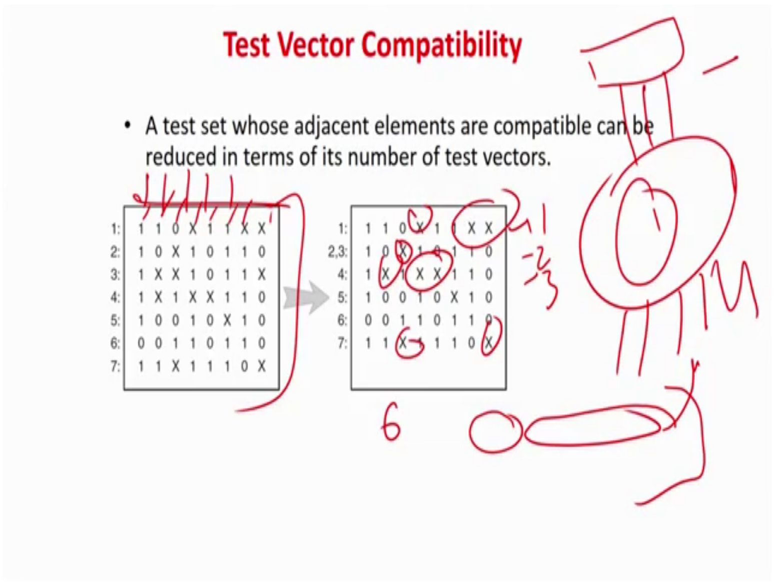
Test Vector Compatibility

- Two test vectors, all bits of which are compatible, are called compatible test vectors and can be merged.

110X11XX (Vector 1)	1101X101 (Vector 2)	Compatible
0XXX1101 (Vector 3)	0XXX010X (Vector 4)	Not Compatible

Example, there are 2 vectors test patterns which are generated. So, you can see that they are compatible, because you can just see that 1101 this is the one over here and it is a X over here. So, it is a compatible set and non compatible is here because in the this position this is a 1 and this as 0. So, these 2 vectors cannot be merged basically not compatible.

(Refer Slide Time: 16:47)



Test Vector Compatibility

- A test set whose adjacent elements are compatible can be reduced in terms of its number of test vectors.

1:	1	1	0	X	1	1	X	X
2:	1	0	X	1	0	1	1	0
3:	1	X	X	1	0	1	1	X
4:	1	X	1	X	X	1	1	0
5:	1	0	0	1	0	X	1	0
6:	0	0	1	1	0	1	1	0
7:	1	1	X	1	1	1	0	X

1:	1	1	0	X	1	1	X	X
2,3:	1	0	X	1	0	1	1	0
4:	1	X	1	X	X	1	1	0
5:	1	0	0	1	0	X	1	0
6:	0	0	1	1	0	1	1	0
7:	1	1	X	1	1	1	0	X

Now, basically there are some ideas I will tell you see for example, these are the test patterns which has been generated let me just learn for you. So, they are some test patterns, which is generated and somebody has sequence there. Now, you can see you can easily see that 2 and 3 are compatible because 10110 X, X, X, 1100110 X. So, basically this is compatible with this. So, you can retain two and merge three that is what they have done. So, it is 2, 3 I have compacted in this manner you can observe the second bit so, it is 0 and it is X.

So, it will be retained as 0. So, you can see that it is retained as 0, now 2, 3 and you can also see that 2, 3 and 5, 2, 3 and 4; there is a compatibility again 110XX11X is a very compatible vector. So, your compaction will be 1, 0 this one will go over here 110110. So, basically we have a compaction like this; similarly you can keep on going about it. Now there are some important things here somebody has already all organised it for me it is array in terms of 1, 2, 3 and 4 are compatible you all see that there are lot of Xs that you are going to generate in ATPG algorithm there is no problem about it, but somebody has to sequence them in proper manner. So, that you can find out the compatible vectors and reduce it.

Now, there are two things over here as I was telling you that basically there may be vectors like 0101, sorry; this test vector one this is test vector two this is test vector 3, sometimes we write in vertical fashion, sometimes we write in horizontal fashion, we write it in vertical fashion, when you were saying that the inputs to the circuit, but because your circuit inputs are like this right.

So, this is a circuit inputs like this; so, in that case we write in a vertical fashion, what; when otherwise also representing we can write that in a horizontal fashion you can observe that there are lot of Xs. So, you can make compatibility, but sometimes you do reorganise the vector, because nobody guarantees in ATPG that first this one will be generated, then this one then this one not like that you can generate in any jumbled fashion, but to find compatibility someone has arranged it in a very nice manner for me. So, that I can get 2, 3 and then I can get four and so, forth.

So, but in general cases there are lot of heuristic, because if you want to try all combinations to find out the minimum number of test that is by merging compatible vectors even in exponential problem, because you have to start study all possible sequencing like 1, 2, 3, 4, 5, 6, 7 all possible combinations of the sequence, which is very very large then only we able to find out the minimum number of ah means vector after combining them, but generally there are lot of heuristics, in which case you do not actually go for all the possible orders basically you find out some of the based orders and to try to find out a minimal set, but the idea here I have shown you that how vectors can be merged.

So, in this case ATPG will give you all then you can apply heuristics find out some good ordering, and then must be compatible one and you can get a minimal set. Now with this minimal set basically you have eliminated some of the vectors, but there is no compromise in the basically in the what do I say there is no compromise in the test coverage, because may be I have reduced vector 2, 3, 4 I have merged may be 2 is deducting stuck at fault at position 1,3, 3 detecting some stuck at fault at position something and 4 is detected at some position something.

Now, if you merge it basically we will find out that this test pattern will detect, this 3 fault together that that can be found out basically that faults can be ah basically detected by this single pattern. So, that is one base some people also put it in built in the ATPG algorithm and some people actually go by this reordering. So, this one way as I tell you can say that this is the very good quality test vector, because it can detect more number of faults.

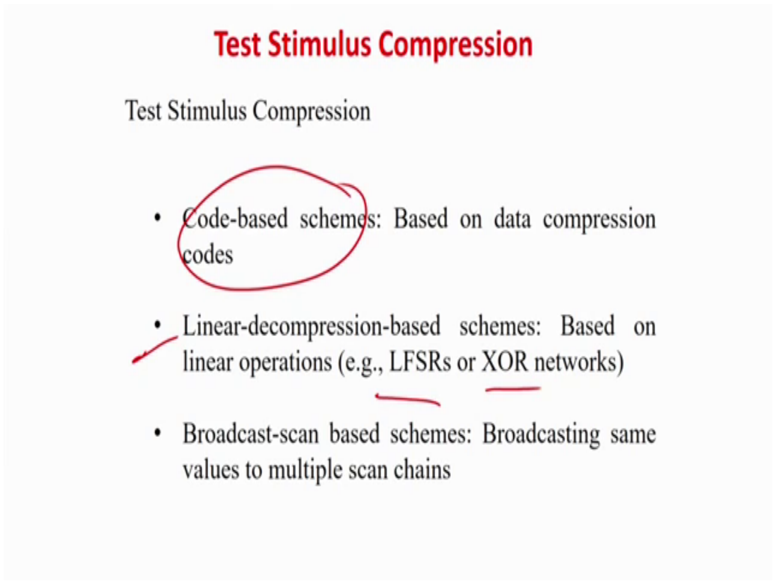
So, detecting one pattern to detect more number of faults can also by can be also obtained by compressing in that manner by vector compatibility, that is one way of doing it, but when I was saying that in the previous diagram when I was saying that I do not want to basically reduce in this, then may be 121101 some vectors are there and do not want to reduce on this, because it will deduction in coverage I was meaning that the vectors are non compatible.

So, if you have some non compatible vectors which are which are to be applied to a circuit to do that; if you cut on that then there will be lot of problems like for example, if I want to merge this 2 vectors or I cannot if I eliminate this they will be coverage loss I was meaning by that, but then if another way of finding good quality test pattern is merging by compatibility we do this, but that is also very simple one another important idea is, that if you see in this case there are 2 Xs, 3 Xs in this case as single X in this case 3 Xs.

So, do I really require 1, 2, 3, 4, 5, 6, 7, 8 do I really require eight channels to do it or can I reduce the number of channels and have some in built circuitry here in this in this in this cut that is; what I was saying in this cut that is I called decompressor can you reduce the width here there actually I have counted. So, we were having some 8 or 9 width. So, really require that can because there lot of Xs like here also there 2 Xs can I do it with 1, 2, 3, 4, 5, 6 can I do with 6 width there is 9 width can I do that, that is very very important or for that basically you will have a decompressor circuit here which will take the 6 bit patterns and generate the 9 bit patterns which is required that is of no difficult problem to solve and it will have a huge impact, because by this you are reducing the number of memory width of the ATE, but you are having the same number of channels.

So, if you are booking same number of channels for a ATE that actually pays a higher cost, because you are taking more amount of hardware, which is dedicated for your circuit.

(Refer Slide Time: 22:38)



Test Stimulus Compression

Test Stimulus Compression

- Code-based schemes: Based on data compression codes
- Linear-decompression-based schemes: Based on linear operations (e.g., LFSRs or XOR networks)
- Broadcast-scan based schemes: Broadcasting same values to multiple scan chains

So, that is why if I reduce this width. So, it will be a better more optimised solution for me in terms of cost effective for the testing a. So, that is a; that is why called test pattern compression by reducing the width there are lot of techniques. So, we will discuss few of them today's lecture few of them will be in next lecture first is the code based schemes. So, code coding theory is entire subjects by itself there are millions of papers textbooks which does coding. So, there will be a input code there will be it will it will modify it make it an output code.

So, input code may be the other way the output code will be the shorter width there can be encryption. So, many things come in the coding theory coding theory means, we will have one short of input they will be encoded to other output other form. So, that, but they have there has to be some means to recover back the input code encoded, that is; the coding and there should be a method to retract back the whole input from which you are generating the output.

So, you know the N number of literatures books subjects are available in coding theory, because the application is huge cryptography fault tolerance transmission. So, everywhere there codes redundant codes, then your parity codes you have, then basically we have also c r c codes, they are all used in the communication to avoid errors in transmission or recover errors in transmission.

Cryptography as we know for security. So, coding code is an it is a very very huge area and even they or most of the techniques can be applied for circuits, but as it is a we are not a code means, if I want to elaborate everything on this there should be a separate codes basically on coding theory and then we can only see how it can be applied for circuits. So, whether what I will try to do I will take the very most popular coding techniques and see how they can be applied.

So, that will give you an idea basically how code based scheme can be used for data compression, then you can take any coding technique and you will be able to apply next one is slightly more interesting which is called linear decompression based scheme. So, basically we will be doing decompression based on some linear operations like linear feedback shift registers XOR networks some small circuits we will have which will basically do the compression and decompression one very important idea as you have to understand that; I cannot make this very large, if this because I told you as I have told you will have the decompressor and the compactor inside cyclic forgettable option resultant taking more ATE channels and ATE time, but of course, it cannot be as big as the cut itself.

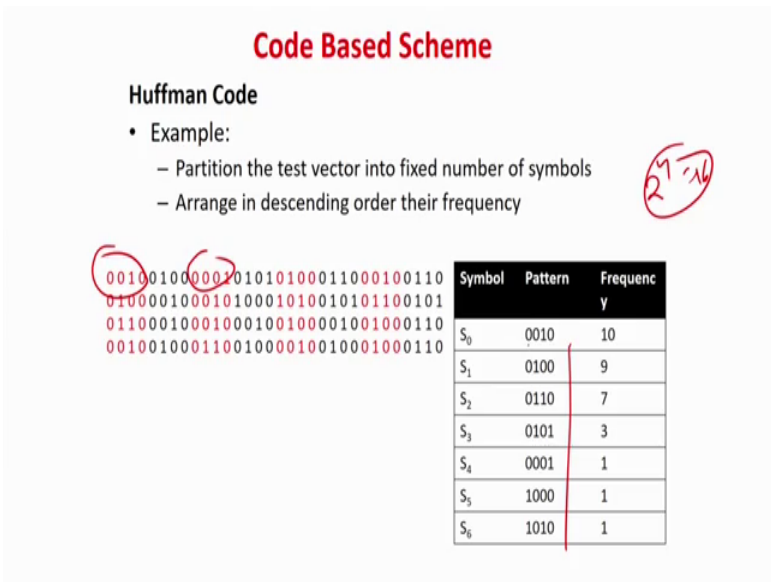
So, you have to make this as small as possible the decompressor and compactor. So, that also has to be kept in mind. So, interesting point is that this linear decompression based schemes use LFSR or XOR networks. So, I mean if you do not have the background on this just study about linear feedback shift registers or linear combinational circuits in any textbook on digital texting you will find out or any textbook on digital design. We will find out that they are very interesting properties like LFSR primary polynomials there are certain polynomials which when loaded into the LFSR as (Refer Time: 25:27) they will be generating all possible patterns or exhausted pattern generators without using a counter.

For example, if you want to design a circuit which will generate values from 0 to say f, f, f, f. So, you will be require a 4, 16 bit counter, but for example, if you do want really want to have the sequence like 0, 1, 2, 3 to f, f, f, f, x I do not require, but all the patterns I will require in any jumbled fashion that is very very important in testing, if you look bits just go and revise bits you will be finding that it is very interesting; that I require all the test patterns from 0000 to f, f, f, f, but I do not require a pattern sequence, because in test patterns generally you can first detect stuck at one then you can detect another stuck at 1 and then the another place stuck at 0; you will not really bother about the ordering, but generating a still bit counter depending on the width is a more costly affair in terms of hardware, but LFSRs are very interesting that if you load with the proper polynomial then you will be finding out that they are very simple circuit with flip flops and simple 1 or 2 XOR gates nothing more than that much much lower circuitry size in accounted, but still will generate all the patterns from 0; it does not generate 0 basically, but from 1s to f, f, f, f, or if the exhaustive statements they will generate, but the circuit size will be very very small. So, that is; what is the idea?.

So, in a you just you can go and revise, but for the time being this understand that LFSR or XOR networks or linear networks a search circuits, which can generate large set of patterns and the circuit size is much much smaller compared to their counter simple digital combinational implementation like a counter or basically a simple decoder the circuit will be very very large, but in that case if small circuit you can generate that huge set of test pattern, but the sequencing may not be obtained.

So, basically that is why this linear decompression base schemes will do the compression decompression for you, but in fact, the circuits are will be very small there is something called also called broadcast scan based scheme. So, broadcasting same values to multiple scan chain. So, can we reorient this scan values or something. So, those same values can be put in multiple scan chains. So, that can be done in parallel. So, if two scan chains of the same value to be pumped in so, there what can I do I can use the same channel of the ATE to do that, but again this I will not discuss in today's lecture that will be postponed to next lecture. When we will be talking about optimisation in terms of design for testability scan is one of the most important part of design for test abilities. So, we will postpone it to the next class, but today we will be mainly looking at this.

(Refer Slide Time: 27:51)



Code Based Scheme

Huffman Code

- Example:
  - Partition the test vector into fixed number of symbols
  - Arrange in descending order their frequency

$2^4 = 16$

```
00100100000101010100011000100110
01000010001010001010010101100101
01100010001000100100001001000110
00100100011001000010010001000110
```

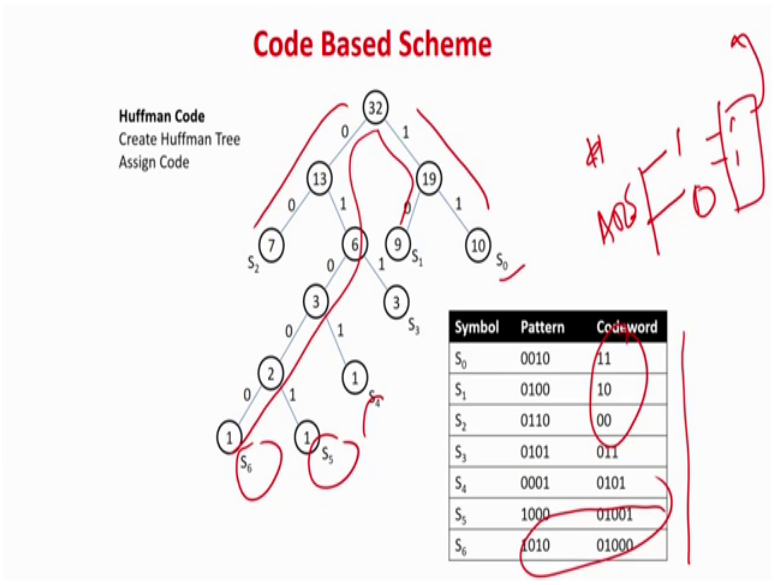
Symbol	Pattern	Frequency
S_0	0010	10
S_1	0100	9
S_2	0110	7
S_3	0101	3
S_4	0001	1
S_5	1000	1
S_6	1010	1

First, I will tell you about a code based scale Huffman code. We always know here the width of the code depends on the frequency is some patterns of higher frequency, then basically your code it will be less is some patterns of rare more rare then your encoding will be larger the idea is that, because some patterns say S is highly frequent. So, you will be sending that multiple number of times. So, you make the bit with less.

So, that you will require less channel width in memory or even in communication, whatever you want to send many times you will require less band width or less number of pins to be set, but what about patterns are occurring more and more rarely. Basically, when if your larger bit size bit width; that is, not a problem, because you will be sending only few amount of time. So, what I do. So, this an example of an ATPG sequence. So, I partition it into fix numbers.

So, I have organised into fixed 4 values and then, these are the patterns I have found out that some symbol lengths I have given S 0 to S 6 we will find out that there 4 bits taken together. So, there can 2 to the power 4, 16 patterns can be there, but in this test pattern basically on the 7 are there. So, this is the test patterns and then we count the frequency like S naught, which is 0 equal to 0010 occurs 10 number of times. So, this is the test patterns I have make groups of 4 and then I am writing the symbol S 0, S 1 to S 6, 7 patterns I have found like I have not found patterns like 1111 does not matter it depends on the test patterns generated and then I count the frequency like S 0 is 0010 the frequency is 10 S 1 is 000100 the frequency 9 is so, forth.

(Refer Slide Time: 29:20).



Code Based Scheme

Huffman Code
Create Huffman Tree
Assign Code

Symbol	Pattern	Codeword
S_0	0010	11
S_1	0100	10
S_2	0110	00
S_3	0101	011
S_4	0001	0101
S_5	1000	01001
S_6	1010	01000

So, generated base Now, I will be encoding is a Huffman fashion. So, you know that in this case we generate an Huffman tree standard discrete discrete algorithms or data structure algorithms course. So, like S 0 the number of we arranged it in terms of frequency the algorithm is already well known same Huffman tree algorithm. So, you can find out that; whatever is the higher frequency like S naught, S 1 they will be towards the root node and whatever is having lower frequency like S 6, S 5 like this one of the lower frequencies they will come to the lower levels of the tree.

The encoding of this one is 11 the encoding of S 1 is 10, S 2 is 00. So, lower frequency will be down the line. So, basically your bit width of the encoding will be larger symbols having higher frequencies will be towards the root. So, the length of this will be left. So, now, you can see if I want to again zoom it. So, that way I will have this pattern. So, for symbol pattern 0010 highly frequent is 11 next frequency is this was 10. So, you can see this S 0, S 1 and S 2 are highly frequent patterns.

So, the code word size is less so; that means, in the ATPG this will be pumped more number of times. So, in this case you will be only required two channels, but these are very rare occurring phenomena. So, in this case you will be requiring 1, 2, 3, 4 and 5 pack 5 channels. So, in channels are basically also multiplexed you can also multiply free channels so, that you can have more efficiency of the ATE equipment. So, depending on requirement patterns on the sent to your chip requirement the same the same ah lines will sent to the other chip for the other vendor, but they will be multiplexed. So, in this you can see that we have more number of patterns means if you are using all this. So, everywhere you will be multiplying by 4, but here actually this are occurring more number of times. So, only 2 pins of them are required. So, that way it basically optimised but again of course, when I it is very important that when I get this as 11; the decompress of circuit should generate inside this. So, may be 11 will go from the ATE. So, I should write this an ATE.

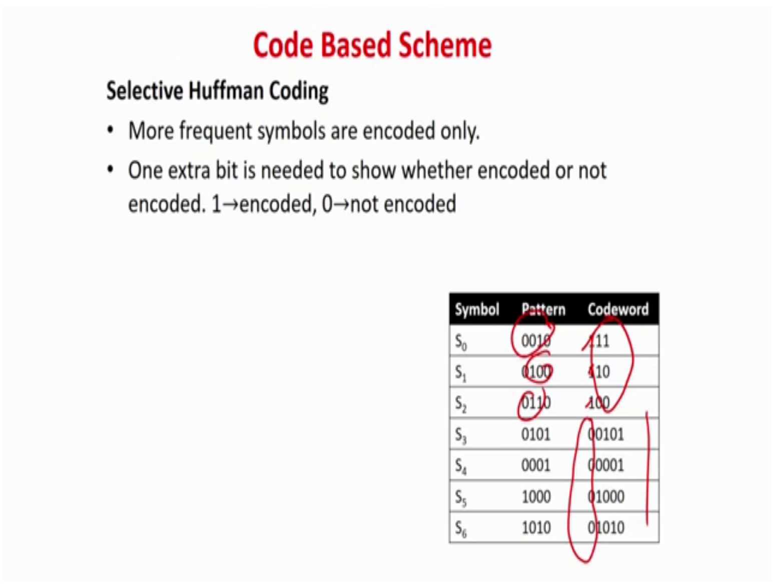
So, ATE will generate 112 pins will be there, I am using for the time being and it will go to use cut. So, cut will be getting 11 inside there should be a circuit of decompressing it will be generating it as 110010. So, there is a decoding circuitry should be there for the Huffman code which is a very standard circuit you can study from any text book.

So, in this case also 1010 will be there will be able to generate this code, but output from this decompression will be exactly this pattern. Now, this is the pattern what has to be applied to the cut under consideration. So, as I have told that there is a way of generating the inputs from the out code words, those you have to generate a circuit for that and you are be able to do it so, idea is that so, now, one people have found out that if you try to compress everything and send then your decompression circuit can be very large.

So, their thought of the lot of actually I have told you that this is the based scheme. Now lot of off shoots have been found out like, there can be another coding scheme called dictionary there is some other coding scheme called brand length coding. So, there are different coding schemes 100 of coding schemes are available people take them apply the base and they do lot of modifications on them I am going to give you an example on what modification people have thought people might think that there are. So, many patterns which are low frequency very very less they occur, but still if you are using this for encoding then your decompress circuit may be large.

So, one option is there I will take only the most frequent code words and I will encrypt them there or I mean I will code them using (Refer Time: 32:41) whatever like in discuss this Huffman code and others I will directly pass. So, that my decompressor circuit is not that large in size does not take more area, because only for the real occurring patterns we require more width basically for others you do not require that much width of the ATE. So, let once in a blue moon one pattern is applying for that I will take all the channels of the ATE rather than making the decompressor circuit too large.

(Refer Slide Time: 33:06)



Code Based Scheme

Selective Huffman Coding

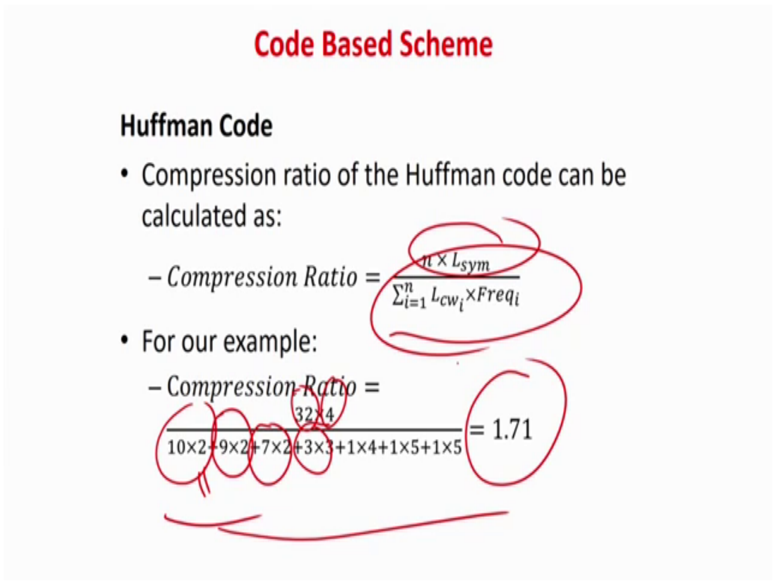
- More frequent symbols are encoded only.
- One extra bit is needed to show whether encoded or not encoded. 1→encoded, 0→not encoded

Symbol	Pattern	Codeword
S_0	0010	111
S_1	0100	110
S_2	0110	100
S_3	0101	00101
S_4	0001	00001
S_5	1000	01000
S_6	1010	01010

So, they call it basically selective Huffman based coding. So, in this case there is something interesting. So, you to have 1 l s b, which one will be telling whether the code is whether the; it is encoded or not like as I told you they are highly frequent the first 3 bits are highly frequent. So, I am sorry they are highly frequent. So, you can see that the m s bs are 1 so; that means, these are the 3 stuff which you have to decode or decompress, but in the last one you can see all this l s m s bs are basically 0; that means, they are non compressed codes and they are directly passed.

So, if this one is basically one so, you do not have to pass it through the decompressor directly you can apply in the circuit on that test, but for all this you require a you have to use this Huffman decoding to find out the exact values like this and this. So, what is the game? So, the game is that your decompressor circuit inside the cut will be of smaller area, but what is the cost you are paying or certain patterns which are rare in that manner will take a higher width channel width and, but from the ATE. So, cost will rise, but no to worry that much because that they are very rarely occurring patterns.

(Refer Slide Time: 34:10)



Code Based Scheme

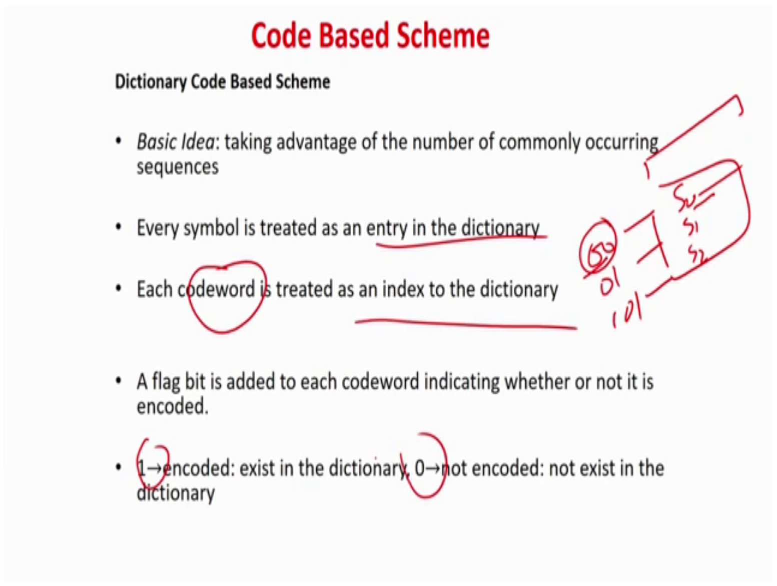
Huffman Code

- Compression ratio of the Huffman code can be calculated as:
  - $\text{Compression Ratio} = \dfrac{n \times L_{sym}}{\sum_{i=1}^{n} L_{cw_i} \times Freq_i}$
- For our example:
  - $\text{Compression Ratio} = \dfrac{32 \times 4}{10\times2+9\times2+7\times2+3\times3+1\times4+1\times5+1\times5} = 1.71$

This is the expression given for this compression ratio. So, you can see that if you are using flat. So, it will be N into the number of symbols that whatever 4 bits are there. So, 4 bits and 32 patters where there is flat and in this case, if you using an Huffman coding. So, 10 into 2; that means, 10 patterns or there are bit means still patt[ern]- frequency is 10. So, those patterns basically only require 2 bits.

So, 10 into 2; there are 9 patterns which will also nine frequency, which is also be encoded into 2. So, 7 these are where 22 bit they were some 3 patterns which require 3. So, if I do all this simple calculation will I will find that the efficiency is 1.71; that means, that much less amount of channel width is required simple calculation that I I apply that decompression is there flat or if I just count the bit into frequency and I divide it I get the compression ratio. So, in this case the compression ratio will be slightly less, but again the hardware circuit use for decompression will be lower.

(Refer Slide Time: 35:11)



Code Based Scheme

Dictionary Code Based Scheme

- *Basic Idea*: taking advantage of the number of commonly occurring sequences
- Every symbol is treated as an entry in the dictionary
- Each codeword is treated as an index to the dictionary
- A flag bit is added to each codeword indicating whether or not it is encoded.
- 1→encoded: exist in the dictionary, 0→not encoded: not exist in the dictionary

Then I am going to take another because there are so, many examples I can go on, but I will just take one more, before I switch to the linear circuits based scheme. So, another is called the dictionary based scheme. So, in dictionary based scheme what is the basic idea; take the advantage in the number of again commonly occurring sequence for mostly the idea is similar that basically if some of the bits which are having more frequency you try to use them for lower number of bits encoding and for which we are having higher number of more number, sorry; which are more frequent you embed using lower number of bits and for higher ones less frequence operating less frequency of occurrence, there you can use slightly more number of bits to encode.

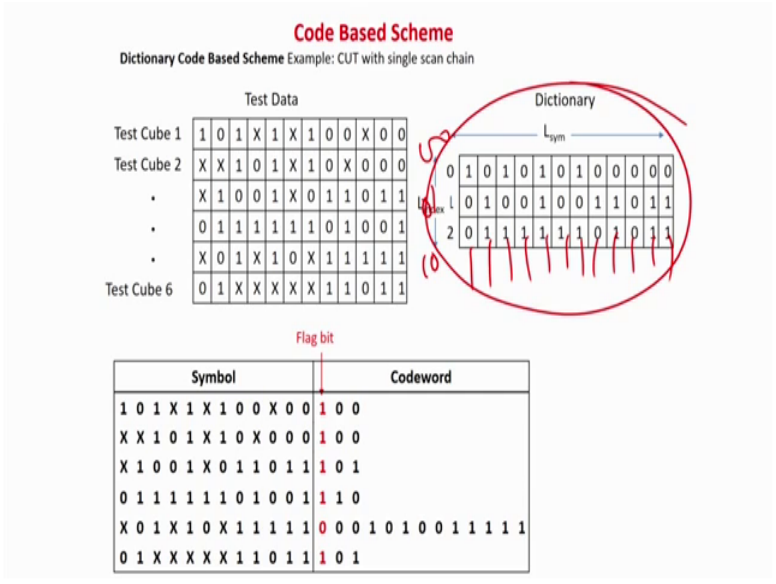
Basically, this is what is the coding idea or another way in the worst case can be higher big width compressing equivalent for all that I will try to see in the next scheme, but in most of the code coding based scheme that is the thing we try to follow that based on frequence we will try to shot it, but again for most there are some other coding scheme also they will take it in a different manner, but mostly I am telling that this is, what is the idea. We take based on frequency; we do it that is what frequency based coding, which is the most popular in case of this test compression technique.

In case of circuits, but as I told you there can be other round other many there can be other different ways of handling in and basically in this lecture we are not going to elaborate in all the such schemes, because this is huge in number we are just going to give you the gist of idea of how compression is done using coding scheme and you can after doing this lecture you can read any scheme and you as you have the motivation that, why it is required you can easily understand them.

Next, we are going to take a dictionary based the idea is taking advantage of commonly occurring sequence same like this, if symbol is treated as an entry in the dictionary, because like a symbol means here we are in the previous example we are made chunks of 444 and we encode may made a symbol it will be an entry in the dictionary each code word, then is symbol we are coding in each word is each word is treated as a index to the dictionary; that means, they will be a dictionary where there will be all the symbols like S 0, S 1, S 2 symbol means basically your test patterns and this is the codeword each one of them will be encoded some value I will encode them.

Now, basically this encoding is actually the index to the dictionary. So, if I want to apply S 0 basically you will be applying this index. So, this dictionary will be will be the on chip circuitry which will take this index and it will generate the corresponding test pattern very simple that this one will be of larger bit of course, and indexing will be a smaller as we all know in English dictionary and we search by the index. So, one we give the input the output will be there. Again here also we have to see how we can actually make this dictionary size small in hardware. Again as I told you just like the modification from (Refer Time: 37:55) Huffman coding here also people use it one means it is encoded available in the dictionary 2 means 0 means it is not encoded not available in dictionary you have to directly apply it all this things are done to make the basically your dictionary size compatible or sorry dictionary size reasonable.

(Refer Slide Time: 38:08)



Code Based Scheme

Dictionary Code Based Scheme Example: CUT with single scan chain

Test Data

Test Cube 1	1	0	1	X	1	X	1	0	0	X	0	0
Test Cube 2	X	X	1	0	1	X	1	0	X	0	0	0
.	X	1	0	0	1	X	0	1	1	0	1	1
.	0	1	1	1	1	1	1	0	1	0	0	1
.	X	0	1	X	1	0	X	1	1	1	1	1
Test Cube 6	0	1	X	X	X	X	X	1	1	0	1	1

Dictionary

L_{sym}

L_{index}

0	1	0	1	0	1	0	1	0	0	0	0	0
1	0	1	0	0	1	0	0	1	1	0	1	1
2	0	1	1	1	1	1	1	0	1	0	1	1

Flag bit

Symbol	Codeword
1 0 1 X 1 X 1 0 0 X 0 0	1 0 0
X X 1 0 1 X 1 0 X 0 0 0	1 0 0
X 1 0 0 1 X 0 1 1 0 1 1	1 0 1
0 1 1 1 1 1 1 0 1 0 0 1	1 1 0
X 0 1 X 1 0 X 1 1 1 1 1	0 0 0 1 0 1 0 0 1 1 1 1 1
0 1 X X X X X 1 1 0 1 1	1 0 1

So, there are some kind of test patterns as you can see let me zoom it for you. So, this is your test patterns and, but here also here we are actually we take care of compatibility unlike the previous method. In case of Huffman we generally do not think about the compatibility in that manner you can also apply there can be modified Huffman plus compatibility approaches, because you can make merger an all, but generally two preliminary approach we have discussed for Huffman that is not thing of compatibility as such. So, what they do basically; you can see I in this case in dictionary method. So, what we can do is you can reduce compatibility just look at this set. So, you can find out that there is vector some Xs will be there.

So, there can be lot of compatibilities like for example, let us take the first 2. So, you can see one X1X11X0111 this one. So, very easily gone a observe the test cube 1 and test cube 2. So, this two test patterns are compatible. So, I can write this one you can see these are the two compatible vectors this are the two compatible I made it compatible and then I give the code word as 0000 and this index to be dictionary. So, and the flag bit is 1; that means, basically they are compressed. So, you have to sorry they are available in the dictionary.

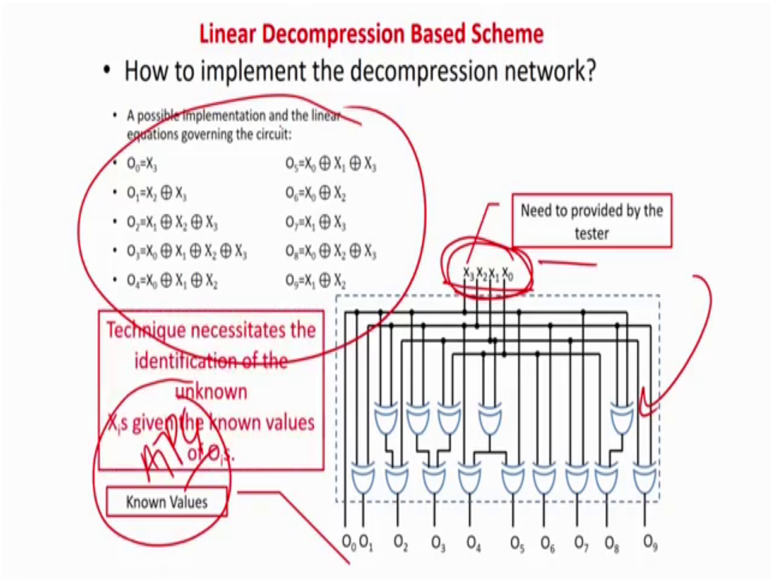
So, if I give the input 00 you have to generate the compatible test vector to this. So, compatible here will be 10101X can be anything 1100, this 0 will come over X00 and 0. So, basically if you look at it this is the first test vector which is actually available the compatible version. Now this will go so, you can find out that this one actually is compatible. The third one will be compatible with this 6, if you can find out you can just study this one will be compatible with this. So, again so, this will be going to the next index 0101; this is the next index and this is the vector which is not compatible.

So, I am putting with another index called 10, but all the l s bs are the flag bits are 1, which means that they are find can be found in the dictionary. So, this 00 is the first compatible vector 0110. So, by this codeword these are the vectors. So, if I put 00. So, this pattern will be generated, if I give the input as 10 this N 01 is this 10 is this 1. So, of course, basically now your channel width for your ATE is. So, only 2 bits you will give and it will at the dictionary will be finding the index and this 1 will be generated. So, basically it is nothing, but it some kind of a combinational circuit you can design with input 001011.

Now, I will come to the interesting story and maybe I find out that this test vector is very very rarely occurring. So, I will put 0. So, if such test vector will be directly given as input. So, if so, in most of the cases you are to be purchasing two ATE channels, but for only in a rare case you will be purchasing a longer set of ATE. So, your job will be done very simple idea, now question is that how can I design this circuit. So, your input is 000110 and outputs are this many.

So, very simple karnaugh map you can optimise two variable karnaugh map how many outputs will be there 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12. So, 12 karnaugh map I have to solve and I can get a digital circuit and I will get this very simple problem it is going to be a very large implementation. So, if you go by a flat structure as I have told in this case you are going to land into problems, because you are encoder decoder whatever circuit you put in your chip in terms DFT in the name of DFT will be very large. So, in the next chain circuit they use linear decompression based.

(Refer Slide Time: 41:45)



Linear Decompression Based Scheme

- How to implement the decompression network?
  - A possible implementation and the linear equations governing the circuit:
  - $O_0 = X_3$
  - $O_1 = X_2 \oplus X_3$
  - $O_2 = X_1 \oplus X_2 \oplus X_3$
  - $O_3 = X_0 \oplus X_1 \oplus X_2 \oplus X_3$
  - $O_4 = X_0 \oplus X_1 \oplus X_2$

$O_5 = X_0 \oplus X_1 \oplus X_3$

$O_6 = X_0 \oplus X_2$

$O_7 = X_1 \oplus X_3$

$O_8 = X_0 \oplus X_2 \oplus X_3$

$O_9 = X_1 \oplus X_2$

Technique necessitates the identification of the unknown X_is given the known values of O_is.

Known Values

Need to provided by the tester

Scheme like LFSR or XOR base network interesting idea of such circuits is that you can do such operation not always in I am not going to do the theory of LFSR or linear circuit as a extremely complex coding theory entire literature is available, if required you can go through there I will just give you the idea that there is some as I call you linear circuits like linear feedback shift registers generate their flip flops there are feedbacks and we are XOR gates in between and this is a combinational circuit type is simpler with simpler version you can see the inputs output will be only connected through a chain of serial XOR gates. They are actually linear operations in when will be written in active terms, we will be doing in terms of and I will tell you this standard theory.

So, basically this is nothing, but some circuit how we are getting this circuit; I will tell you, but there is a lot of mathematics behind it that how can we design such a circuit the only thing you have to note that there only 3 gates and simple equation form is X 0 equal to X 3. So, if you can see X 3 and the output is X O 0. So, very simple O 0 is equal to X 2, if you study O 1 it is output 1 if X 2 or X 3. So, output one if you see it will be the this gate and is nothing, but XOR and X 1 simply if you just study the circuit it is a very simple you can find out that I can write this equation.

But, how the circuit is gone obtained; we will tell later, but entire theory is very very complicated we will not go into that say some circuit is there and somebody has told you the property that whatever we are discussing here like some input is there you can generate all the like colour like some circuit, we can generate by normal karnaugh and optimisation that some combinational circuit is there input is there equal output will be generated, but that will be large because it will have ands or not gates.

But basically this such kind of linear structure is also have the same properties, but it will it cannot be always guaranteed, because in this case whatever input patterns and whatever output patterns you require you can get a little circuit for that, but in such cases the input output is restricted like for all possible combinations you may not be able to obtain such a circuit, but there exists lots of input output combinations, because there are lot of Xs I will tell you what they are then will be more clear.

So, these are some bits less number of 4 bits which the tester will give and these are the outputs, which are the input to the cut this we know this is your ATPG this is your ATPG pattern and, but we all using a less as already we are trying to do here your number is also some 11 or 12 when you are using 2 bits to do it here I use the having 4 bits as input and output will be some 10 bits. So, they are known values means they are basically your ATPGs that has to be applied to the circuit under test, but somehow we have to compress it using 4 bits or 5 bits or whatever.

So, generally there is a lot of mathematical theory in coding in coding theory of linear networks, which tells you that if this values are known and we know the number of force number of bits required to do it; in this case I am resting it to 4 or 5 whatever this number is also fixed and this also all this variables are known; that means, and the idea is that as I told there are lot of Xs do not clears.

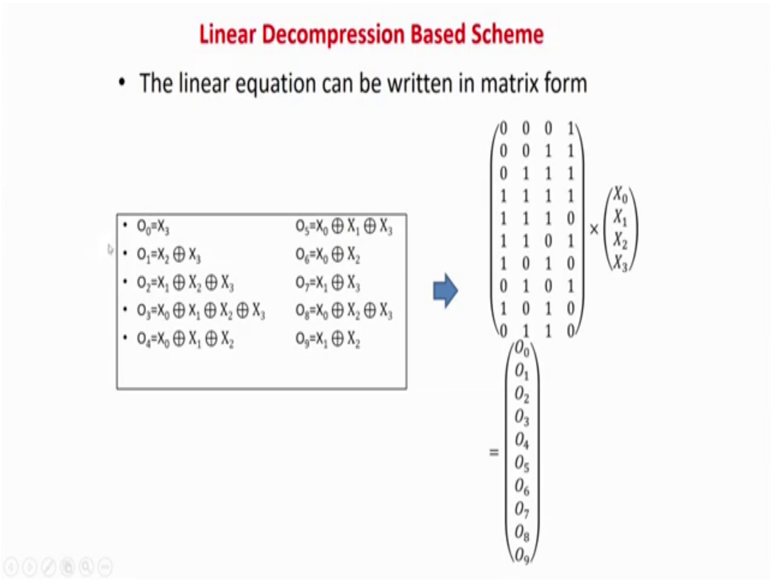
So, then it can tell you that whether such a thing can be implemented or not I will better to take it an when I will take an example it will be clear, but the question here is that some values I know then be lot of do not cares that is; what is the only Boolean I am having then can I design a very simple linear circuit, but only XOR gates such that I can have a smaller width values.

So, that if I keep on applying all the known values can be generated and this should not be a as large as combinational that is the question. So, careless circuit be generated like this the idea is yes for many of the cases it can be generated, because there are lot of do not cares here, but is not a general case generally you may not be able to do it generate make from it may be possible by 4 it has to be increased to 5 to 6 that is and of course, if you have you have 10 lines here if you have 10 lines here of course, is a direct mapping solution can I do it by 9 possible 8.

So, more you reduce and of course, you have to be very very careful that basically the circuit is a linear circuit you cannot put lot of complications. So, far here so, from that is 9, 8, 7, 6 type 4 you can keep on going it, but at what position; because more less numbers here is more important, but it is not always possible to go beyond the certain limit to generate it.

So, those is not possible to generate means you will not have such a simple circuit for that, but have theories extremely complex I am not going into it, but just take the idea right. Now, that for most of the cases it is possible that somehow I can get a sufficiently good number which is much much less than the number of output bits and still I can solve it.

(Refer Slide Time: 46:22)



Linear Decompression Based Scheme

- The linear equation can be written in matrix form

- $O_0 = X_3$
- $O_1 = X_2 \oplus X_3$
- $O_2 = X_1 \oplus X_2 \oplus X_3$
- $O_3 = X_0 \oplus X_1 \oplus X_2 \oplus X_3$
- $O_4 = X_0 \oplus X_1 \oplus X_2$

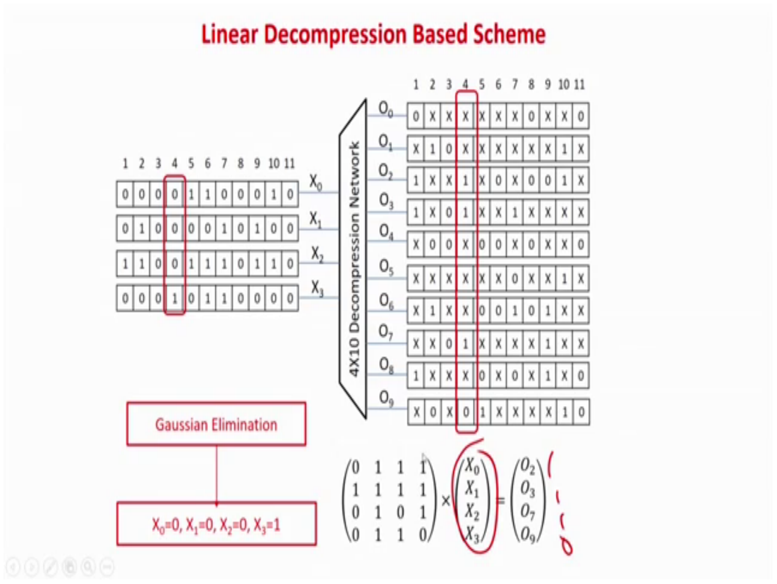
$O_5 = X_0 \oplus X_1 \oplus X_3$
$O_6 = X_0 \oplus X_2$
$O_7 = X_1 \oplus X_3$
$O_8 = X_0 \oplus X_2 \oplus X_3$
$O_9 = X_1 \oplus X_2$

$$\begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} O_0 \\ O_1 \\ O_2 \\ O_3 \\ O_4 \\ O_5 \\ O_6 \\ O_7 \\ O_8 \\ O_9 \end{pmatrix}$$

So, if you have this equations just you can map it I can simply represent it is in a matrix form. So, the matrix form is a something like I will zoom it for you it is very straight forward operation like in this case this whole matrix I am representing. So, 9 will be the out[put]- 10 will be the output. So, here we will have 10 I think 1, 2, 3, 4, 5, 6, 7, 8, 9, 10; 10 is the number of columns number of rows is 4 4 inputs.

So, 1, 2, 3, 4; so X 0, X 1, X 2, X 3 the inputs and 10 are the outputs. So, row by column matrix we have if you see 0001. So, this X 0 is 0; X 1 equal to 0; X 2 is also equal to 0; X 3 equal to 1. So, X 3 is equal to O O 0. So, that is; what is the first equation? Similarly you can do it. So, whatever is XOR over here is and in the map sheet.

(Refer Slide Time: 47:06)


Linear Decompression Based Scheme
4X10 Decompression Network
Gaussian Elimination
$X_0=0, X_1=0, X_2=0, X_3=1$

	1	2	3	4	5	6	7	8	9	10	11
X_0	0	0	0	0	1	1	0	0	0	1	0
X_1	0	1	0	0	0	0	1	0	1	0	0
X_2	1	1	0	0	1	1	1	0	1	1	0
X_3	0	0	0	1	0	1	1	0	0	0	0

	1	2	3	4	5	6	7	8	9	10	11
O_0	0	X	X	X	X	X	X	0	X	X	0
O_1	X	1	0	X	X	X	X	X	X	1	X
O_2	1	X	X	1	X	0	X	0	0	1	X
O_3	1	X	0	1	X	X	1	X	X	X	X
O_4	X	0	0	X	0	0	X	0	X	X	0
O_5	X	X	X	X	X	X	0	X	X	1	X
O_6	X	1	X	X	0	0	1	0	1	X	X
O_7	X	X	0	1	X	X	X	X	1	X	X
O_8	1	X	X	X	0	X	0	X	1	X	0
O_9	X	0	X	0	1	X	X	X	X	1	0

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \times \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} O_2 \\ O_3 \\ O_7 \\ O_9 \end{pmatrix}$$

So, the simple matrix representation; now I am directly coming to an example how I have got this patterns we will we will not think about it say this is the pattern set I am getting again I am saying that is the representate in a vertical manner. So, first pattern is 0 X something is there the second pattern is 2 X 1 some values are there observing that I have 4 deterministic values only.

So, in this case 1, 2, 3 and 4 here also 1, 2, 3, 4 so, in most of the all all that channels if you see all the test patterns you see, then only 4 deterministic values all are do not cares. So, that actually motivated me that can I have 1, 2, 3, 4 bit inputs which I split and there will be a decompression network of course, this decompression network is very simple if I make it extremely complex then you can have any mapping you take this as input and you want to generate this as output you take this second as input you generate this as output.

So, I can silly solve some karnaugh map optimisation and I can have a general digital circuit for that combinational, but that I do not want to do it I only want to do it by linear operations. So, that the this decompression circuit is less. So, only by XOR gate structure. So, is it possible? So, the theory says that basically you are seeing that we have all only 4 ones here. So, let me just try to keep this number as 4 is it possible by 3 inputs may be, but I do not know because the theory exist whether it will old or those theory will prove you that whether it is possible with 3 or not can I do it by 2 more difficult for most of the case it will be no, but can I do it with 8 inputs is this channel with 8 it be slight it will be a more easier problem to solve.

So, that actually lot of theory exist to find out whether it is possible or not so, for the time being I am just giving the theory. So, with that mean we are finding out that for every case the number of one's is these are mistake values are 1. So, I may hope that this will be ok. So, then what then basically, what happen I am trying to do let me just take this 4th vector. So, this 4th vector if you look at this X1 so, O 2 is very important O 3 is very important O 7 is very important and O9 is very important.

So, just if we look at it we are just taking the deterministic values. So, O 2 O 3, O 1 and O 0 these are the most important 1. So, I will have the equation O 3, O 2, O 3, O 7, O 9 and X 0, X 1, X 2, X 3 is fine, but now how I am getting this smaller vector, because this O 3, O 2, 3, 7, 9. So, O 2, 3, 7, 9 are the most important one so, 2 012 so, this 2 then 3 and some 7 and 9.

So, these rows I will select and I will have a tranquited matrix. So, this was the tranquited value very simple. So, now, basically we have to this is given to you. So, this is the reduced vector reduced matrix, which I have got from there these are the inputs and are these output this I know this we have already found out as O O 2 is something like 1110 this 1110. So, this 1110 now we have to find out the value of Xs.

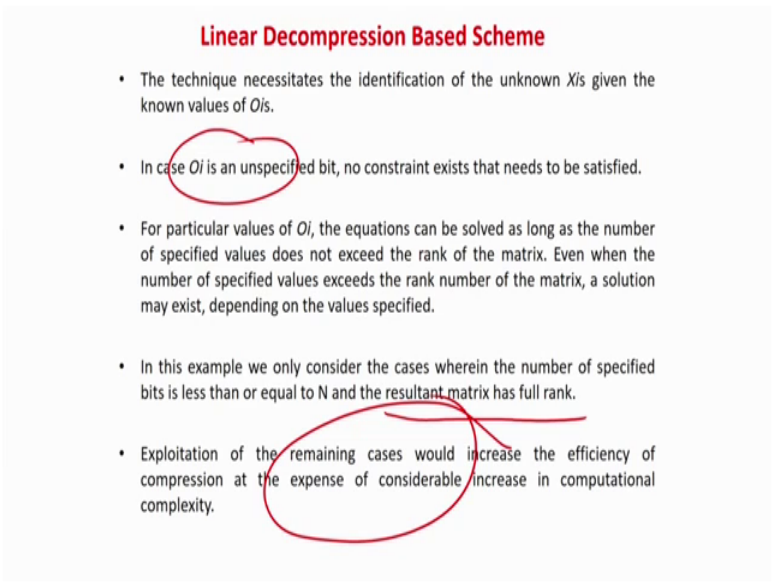
So, we can go for a standard Gaussian elimination based equation solver and that one will tell you that what are the values of this variables? So, it will tell that X 0 is equal to 0001. So, it will put it as 0001 now you have to solve it for all the levels 1, 2, 3, 4, 5, 6 every cell solve all the values. Now, this one is for 1, if you solve it you will get this value and so, forth. Now, it is very interesting that you can using this circuit using this circuit, basically if you apply this patterns you are going to by this four patterns you are apply this you are going to get this value, but the Xs can be 0 and 1 that id do not know, but still all the meaningful values we will get by applying this.

So, this decompressor circuit is now not a full combinational circuit, but it is a rather a simple linear circuit and you are staying to state the yet the effect. So, that is the idea of linear decompression based scheme or by using an LFSR also a sequential circuit to do the same thing the idea is that, we will be using less number of width and we will be going to generate a higher bit size, but this circuit I has been not using as a full combinational blow up I will be using linear operations and. In fact, it is possible to do, but this is theory exist that; what is the input size? What is the number of output size? What is the number of Xs required and where that do not case request to do it for possible?

For all combinations it may not be possible. So, of course, as I told you it is a it is a not very simple problem, because it may be possible if I have some may be 1 X here and 11 here can be ATPG algorithm a t p test pattern will be changed like that can I test some faults by keeping a X over here and a 1 over here. So, then again it is a feedback to the ATPG algorithm. So, it is a it happens is a queued fashion.

So, it tries to find out I will only have 3 channels and this be pattern require I found out that they are does not exist in linear circuit to be there. So, either I can use it 2, 4 or I can somehow modify the test patterns. So, that it becomes compatible. So, there are lot of in utility based heuristic algorithms to solve the problem. Now, slight more theory I will tell you that when it is possible.

(Refer Slide Time: 52:16)



Linear Decompression Based Scheme

- The technique necessitates the identification of the unknown *Xi*s given the known values of *Oi*s.
- In case *Oi* is an unspecified bit, no constraint exists that needs to be satisfied.
- For particular values of *Oi*, the equations can be solved as long as the number of specified values does not exceed the rank of the matrix. Even when the number of specified values exceeds the rank number of the matrix, a solution may exist, depending on the values specified.
- In this example we only consider the cases wherein the number of specified bits is less than or equal to N and the resultant matrix has full rank.
- Exploitation of the remaining cases would increase the efficiency of compression at the expense of considerable increase in computational complexity.

So, the technique necessitates that the unknown x I is to find out the X is given the O is; that means, you know these values you know these values and you have to find out the value of Xi, but O number of O is and number of C is are fixed. So, if Oi is a unspecified bit you did not solve anything; that is what I have seen that is always 2, 7 X 7 for whatever you are considering you have to find Gaussian elimination for whatever is specified for a particular values for particular values of Oi the equation can be solved as long as the number of specified values does not exceed the rank of the matrix. So, what is a rank of the matrix this either told is the more theoretical this one?.

So, basically broad sense rank of a matrix means how many independent columns relationship can form. So, in this in this basically example this is a full rank matrix; that means, none of the columns basically in this matrix none of the columns are dependent of any other. So, if you have a, but how the columns has been generates the column has been generated best on the circuit.

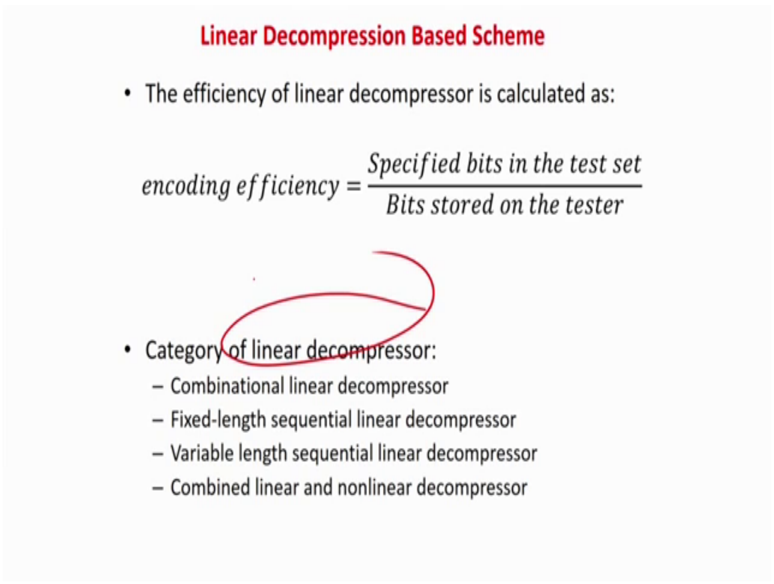
So, there is a plain chloride saying that the for particular values of Oi the equations the equations can be solved as long as the number of specified values, that is specified values means which are non Xs in Oi do not exceed the rank of the matrix even, when the number of specified values exceeds the rank of the matrix a solution may exist depending on the values specified then we do not guarantee.

So, in this case if you see we have taken only 4, 4 as specified bits and also, if you see that the 4 as I told the full rank matrix. So, the rank will be there will be all columns are independent. So, here I safely can put 4 values which are specified in each of the patterns.

So, in this case I am always able to get a solution therefore; that means, what in this is a full rank matrix, then I have only 4 over here. So, basically I will be always able to find out a solution which will generate this, but this are the not very optimised. So, you can even increase the number of specified values, but still whether can I get a solution with four inputs that may not be guaranteed that will depend on the outputs, because in that case the number of specified values will be more than the rank of the matrix in this example we consider only the case where the number of specified bit is less than or equal to N and the result matrix has full rank.

So, basically we have considered specified to be N to be 4, only because only specified bits is 4 as the rank matrix is 4. So, in this case you are satisfied, but exploitation of the remaining cases is are complex problem and we are not dealing with it that is; whether if you have 5 values specified and the matrix is not a full rank matrix then, whether we can find out certain linear circuit or not. So, there lot of theories available for the extreme it is a bit complicated subject, but you can look over it, but what I am trying to show you is that basically in linear compression based scheme; if we have less number of specified bits and we have a reasonably large rank matrix, then you can have a simple translation scheme and this decompression network will be very very small involving only XOR gates or in some of flip flops and XOR gates full not full blown up.

(Refer Slide Time: 55:19)



Linear Decompression Based Scheme

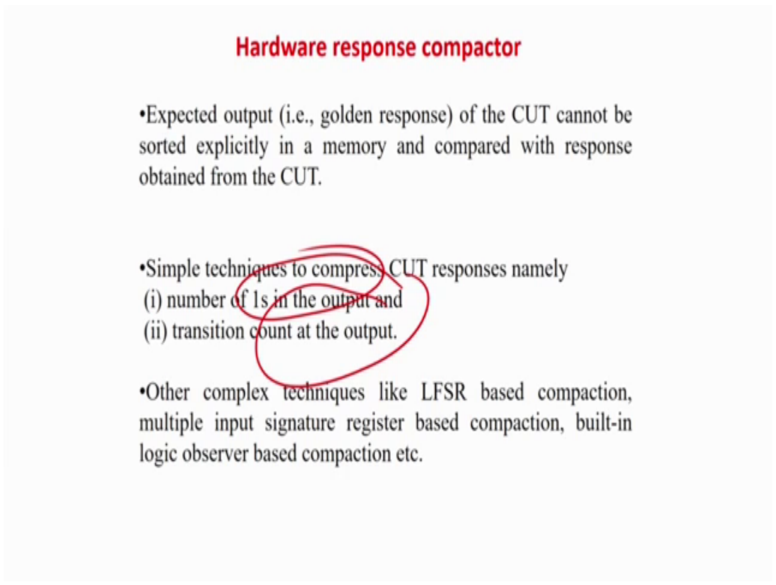
- The efficiency of linear decompressor is calculated as:

$$encoding\ efficiency = \frac{Specified\ bits\ in\ the\ test\ set}{Bits\ stored\ on\ the\ tester}$$

- Category of linear decompressor:
  - Combinational linear decompressor
  - Fixed-length sequential linear decompressor
  - Variable length sequential linear decompressor
  - Combined linear and nonlinear decompressor

So, basically this is the encoding efficiency and lot of linear decompressors can be their fixed length variable length sequential non sequential.

(Refer Slide Time: 55:32)



Hardware response compactor

•Expected output (i.e., golden response) of the CUT cannot be sorted explicitly in a memory and compared with response obtained from the CUT.

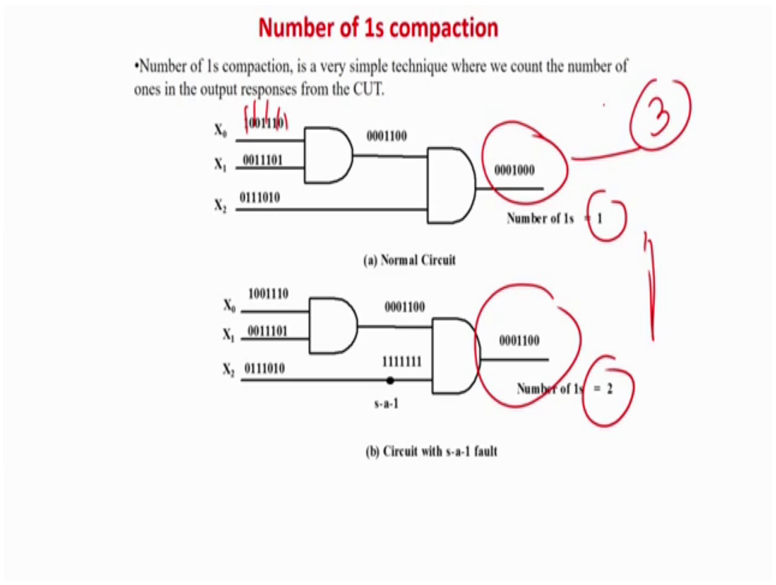
•Simple techniques to compress CUT responses namely
(i) number of 1s in the output and
(ii) transition count at the output.

•Other complex techniques like LFSR based compaction, multiple input signature register based compaction, built-in logic observer based compaction etc.

So, many are there; so basically this are two main techniques which are used for actually encoding or compressing your inputs, third one as I told you will be about your scan chains or broadcast scan which I will cover in the next lecture. Now, before we complete quickly we will look at; how the; what about the output size; because output size also output also you cannot the full one and put it in the memory that is also not possible.

So, the expected output golden response of the cut cannot be stored explicitly. So, basically there are lot of ways to compact this is not a compression compaction. So, there will be the lossy compaction. So, sometimes there will be aliasing and taking two most important 11 is counting the number of ones and one is the transitions, how many changes are there? There are several other techniques like, LFSR signature register etcetera for this compaction, but I in this course I will be taking the most simplest one.

(Refer Slide Time: 56:19)



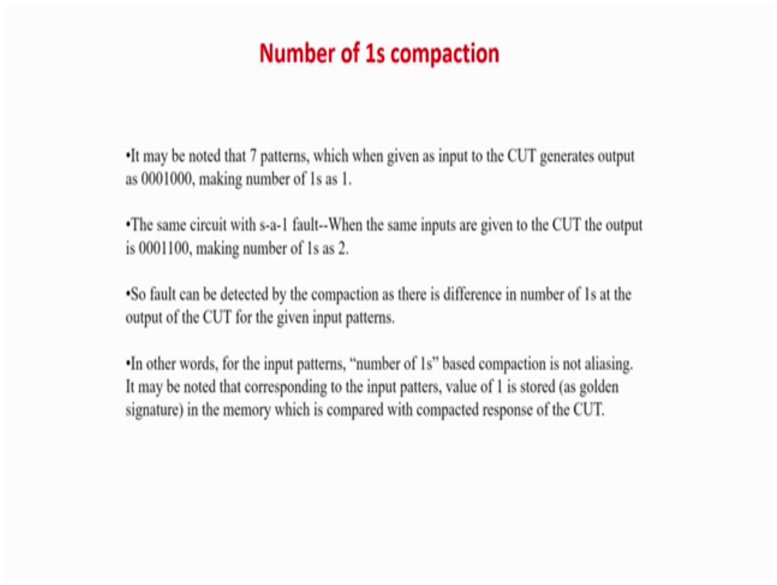
Number of 1s compaction

•Number of 1s compaction, is a very simple technique where we count the number of ones in the output responses from the CUT.

So, is a very simple example this is a circuit may be the patterns I am applying is 110 then I am applying 110 and so, forth. They are 1, 2, 3, 4, 5, 6, 7 test pattern because they are I am applying in a showing in a vertical manner. So, this pattern is there and this is the output the number of 1s is a 1. So, in this case there will be a counter which will count the number of one and to generate the answer as one which will be fed to the ATE. Now say there is a stuck at fall at the output at this line so, the number of 1s. So, if you do an we will find that the output is this so, output is this. So, the number of ones will be 2. So, we will just send the count to the ATE memory you will find out that there is a mismatch in the number of ones. So, this circuit is coming up fault.

So, instead of serving sending 7 bits we can just send 2 bits to encode the number of 1s, because is 7 is the number of test patterns. So, maximum 7; so, 3 bits will be required at the output. So, you can count how many numbers of one in the fault will be detected.

(Refer Slide Time: 57:15)



Number of 1s compaction

•It may be noted that 7 patterns, which when given as input to the CUT generates output as 0001000, making number of 1s as 1.

•The same circuit with s-a-1 fault--When the same inputs are given to the CUT the output is 0001100, making number of 1s as 2.

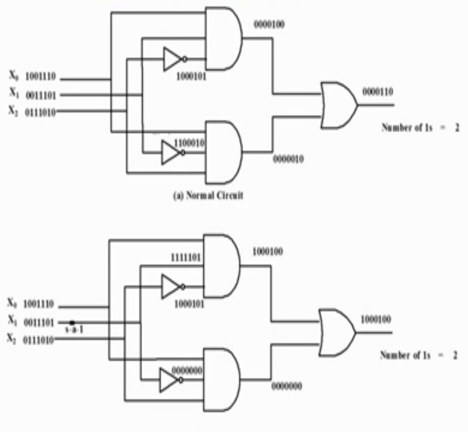
•So fault can be detected by the compaction as there is difference in number of 1s at the output of the CUT for the given input patterns.

•In other words, for the input patterns, "number of 1s" based compaction is not aliasing. It may be noted that corresponding to the input patters, value of 1 is stored (as golden signature) in the memory which is compared with compacted response of the CUT.

But so, in this case I whatever I told you in the slide you can do it, but it is not always the case let us take a slightly modified example same thing how many patterns I am giving 1, 2, 3, 4, 5, 6, 7, 8; 1, 2, 3, 4, 5, 6, 7, again same patterns are there this is circuit if you do it this the normal circuit the output is number of one's is 2.

(Refer Slide Time: 57:22)

Number of 1s compaction



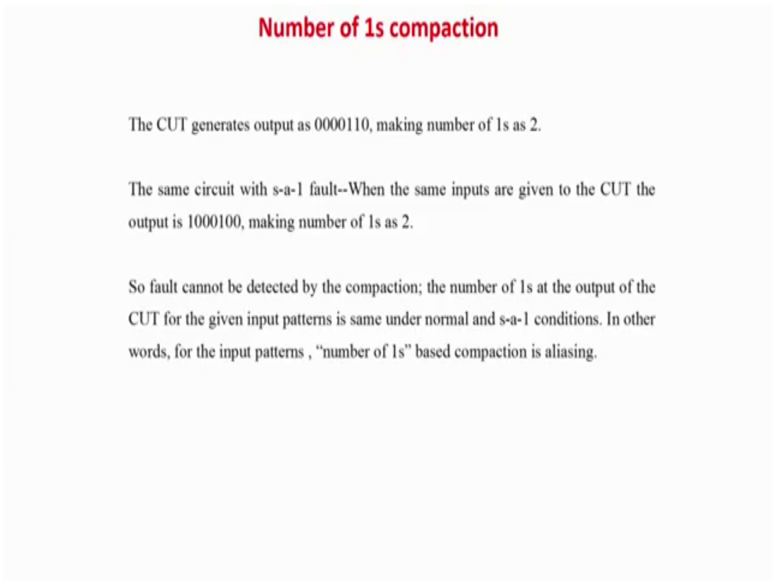
(b) Circuit with s-a-fault

Now I take a stuck at one over here the output pattern is this one this pattern and this pattern are definitely different. So, very easily you can find out the stuck at fault if for feeding the 7 width as the output without compaction, but now I am compacting it by the number of 1s.

So, here also 2, 1s here also two ones the positions are different, but I am not keeping the positions in mind I am just taking the number. So, in this case if fault is just you will be not able to detect which is called a aliasing. So, is a lossy compression. So, you are having a aliasing effect. So, as I told you counting number of one's is a very rudimentary approach there are lot of advances are there, but I am just giving the idea so, that you can refer to any of the techniques; because all things are all the things are based on coding theory.

Coding theory the theory itself is so, well developed that you should have a full course on coding theory to get into ideas an all this things, but I am telling you from this view point of the circuits. So, once we have the more idea on coding theory you can easily map it to the case of VLSI.

(Refer Slide Time: 58:31).



Number of 1s compaction

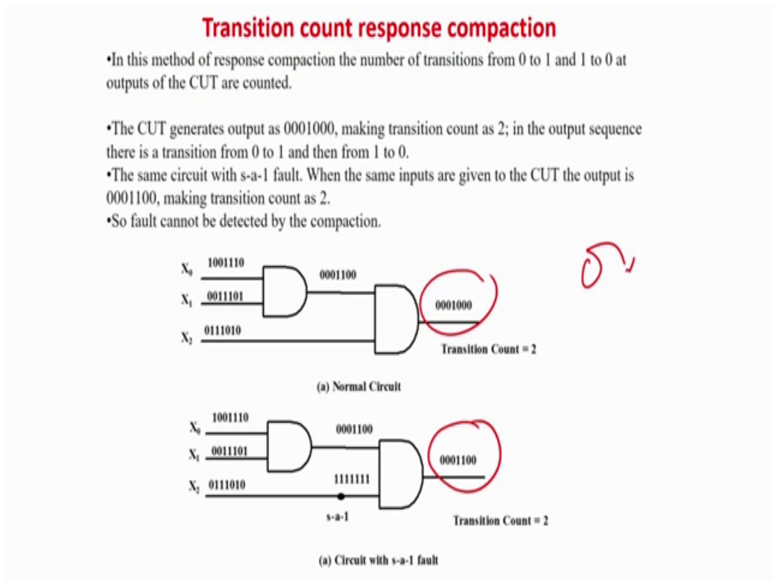
The CUT generates output as 0000110, making number of 1s as 2.

The same circuit with s-a-1 fault--When the same inputs are given to the CUT the output is 1000100, making number of 1s as 2.

So fault cannot be detected by the compaction; the number of 1s at the output of the CUT for the given input patterns is same under normal and s-a-1 conditions. In other words, for the input patterns , "number of 1s" based compaction is aliasing.

So, this is actually called number of one compaction given aliasing. So, in this case the fault is not detected.

(Refer Slide Time: 58:35)


X0
X1
X2
1001110
0011101
0111010
0001100
0001000
Transition Count = 2
(a) Normal Circuit
1111111
s-a-1
Transition Count = 2
(a) Circuit with s-a-1 fault

Transition count response compaction

•In this method of response compaction the number of transitions from 0 to 1 and 1 to 0 at outputs of the CUT are counted.

•The CUT generates output as 0001000, making transition count as 2; in the output sequence there is a transition from 0 to 1 and then from 1 to 0.
•The same circuit with s-a-1 fault. When the same inputs are given to the CUT the output is 0001100, making transition count as 2.
•So fault cannot be detected by the compaction.

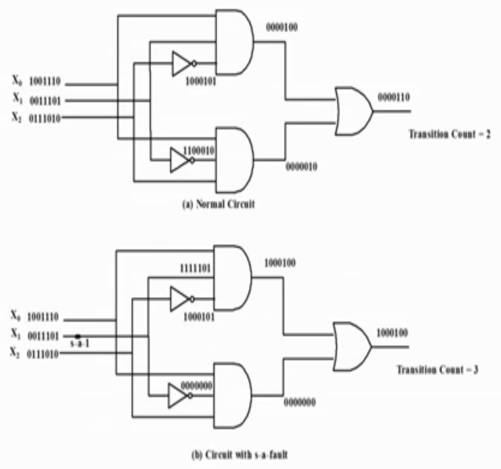
Another technique which is called we apply is called basically your transition normal circuit samne patterns I am applying so, you can see 0001 so, one jump from 0 to 1 and one jump from 1 to 0; so, that tw2o type of transitions. So, the transition count equal to 2. Now I am having again the same stuck at 1 fault. So, here also the transition count is 110. So, 1 transition and 2 transition so, here the transition is 2, but again the fault is not detected. So, another technique I have applied which I am showing is basically transition by compaction we just find out how many times it is changing from 0 to 1 and 1 to 0.

So, there is two jumps over here and there is also two jumps, but you will the patterns are different. So, if you are using the counting one sequence the fault can be detected, but when you are using a transition count this fault is getting alias, but for other cases it will be the vice versa case like; the same circuit I am taking the second example same inputs the pattern is something like this is the output the transition count is 2, I am changing the circuit. So, in this case I am making taking the stuck at one fault.

Now, the pattern is something like these patterns are of course, different very easily you can detect it, but here also the transition count is not changing. So, in this case you see 1 transition, 2 transitions and 3 transitions. So, here the transition count is 3. So, in the second example transition count best compaction can detect your fault, but not the counting number of ones and which was reverse in the other example so. In fact, what I mean to say here.

(Refer Slide Time: 59:59)

Transition count response compaction



Was basically when you get the output we do not compress it rather what we going to for compaction and there are lot of compaction techniques 2; I have very preliminary I have given you an idea, but for many of the cases just like we are very very successful in compression for many of them in most of the cases aliasing do not happen or very few very very rare aliasing happens. So, we slightly modification in technique like if, you are using LFSR based compaction or some other methods of compaction the aliasing effects are very very rare very very less and we can actually have a very small bit width representation of the output to detect a get a very good fault coverage.

But here I have given you two very examples to show the strength of each of the compaction scheme, that it may work for one circuit and it may not work for another, but for a general case for most of the good compaction techniques this fact this phenomena is less. So, with this we come to the end of this lecture in this lecture we have tried to show that given a circuit and given some ATPGs how one will be mainly compact by compacting the width. So, 10 bit is the input pattern to the cut how can we represent by 4 bits or 2 bits so, that we can save on the ATE channels.

And secondly, also the output is also having some width. So, how can we compact the width by compaction so, that we can have more efficient; in terms of complexity. So, that we can have very for very large circuits still we can optimise the test patterns so, that we have a reasonable cost in the ATE I should tell that this is a solution we are getting by paying some cost in terms of coverage and also some extra area over rate in the circuit for compression and decompression, but still we have we have a very very large circuit there is no way you have to go by this otherwise your test cost will be.

So, high that you will not make a profit in the market so, basically even if slight trade of in fault coverage guarantee you can go for this compaction. So, with this we close now and from tomorrows lecture, we will see how DFT circuits can be optimised mainly in terms of scan chain to have better quality or how we can I mean whatever ATPG compaction we have done how it can be again helped by the what do I say is the scan chains, how such optimisation can be done taking in the mind this scan chain structure.

Thank you.