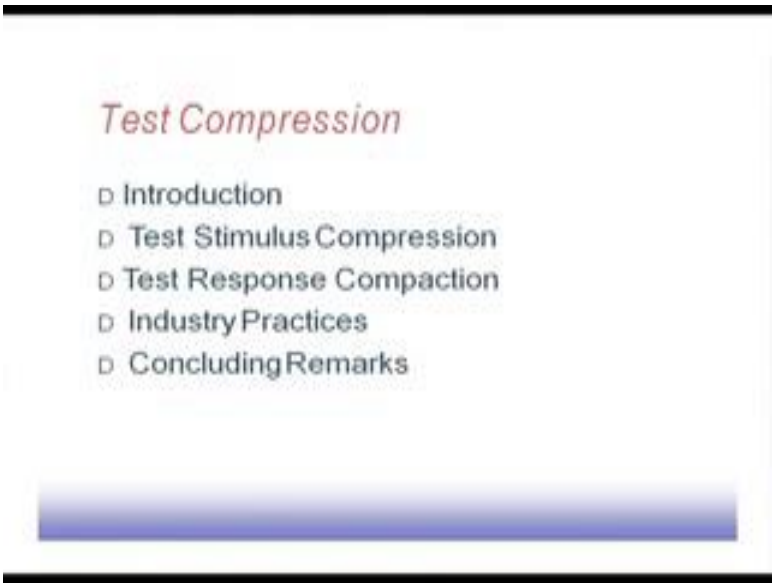
Digital VLSI Testing
Prof. Santanu Chattopadhyay
Department of Electronics and EC Engineering
Indian Institute of Technology, Kharagpur

Lecture - 28
Text Compression

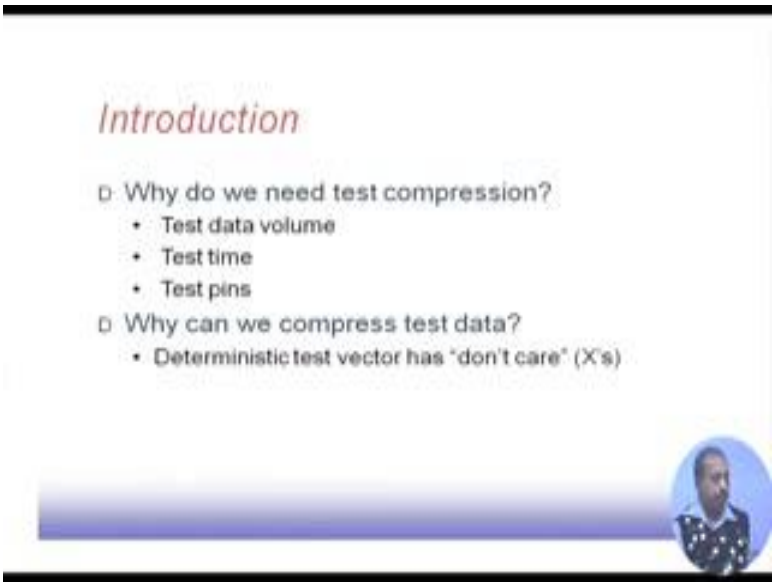
Next we will look in to another very important issue in testing which is known as Test Compression. So, basically when we are storing the patterns in the automatic test equipment we need to have number of patterns small or less and if it is, but that is all that is not always possible because it is not that if you use a small number of patterns will get good coverage that is not going to happen. So, we need to have large number of patterns, but storing them becomes an issue. So, we want to store them in a compressed fashion. Similarly when we are talking about the response part, response part also we may not be willing to store all the responses. So, we want to we may like to store some compressed version of those responses these are the 2 things that will look into in this chapter.

(Refer Slide Time: 01:23)



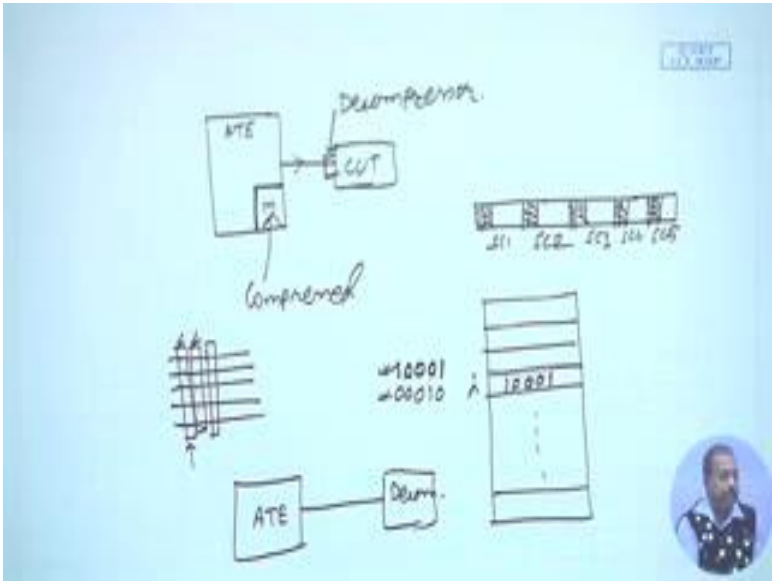
So, compression will be, first will look into the introduction then test stimulus compression how can I compress the test patterns, then test response compaction how can I compact the responses and then some industry practices and then the conclusion.

(Refer Slide Time: 01:31)



Why do we need test compression? So, it is one thing is the test data volume test time and test things. So, you see that if we have got a typical test pattern application environment for external testing we have got an automated test equipment and there is a memory and this is the circuit for which we want to test.

(Refer Slide Time: 01:43)



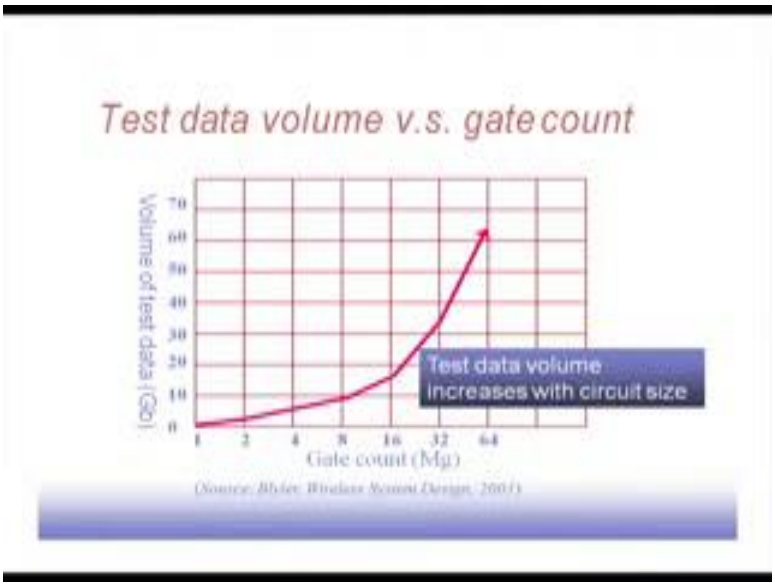
Now, this test patterns are to be transported from this memory to the circle and then they are to be applied to the circuit and then the response has to be collected. So, one thing is that if this memory requirement is large then that will increase the cost of the ATE.

Similarly if I want to transfer a large number of pattern over this line then serial communication is not a very effective one. So, we have to have parallel communication. Now how many such parallel lines can be drawn, how many such parallel channels can be there or things can be there so that is another issue. So, this cost of the ATE is dependent on the number of these things that we have that is ATE channel width and the memory that we require that is to store the patterns. And taste time is also increasing because of this large volume of data that we need to transfer. So, test time will also increase.

So, it is better that we store these patterns in a compressed form in the ATE and we transfer the compressed sequences to the circuit and at the circuit before applying the patterns. So, we have got some sort of, we have got some sort of decompressing mechanism by which this scatter we get back the original patterns and then those that are supplied to the circuit. So, this is the thing. So, what we essentially mean is that hear it will be the patterns will be stored in a compressed form and then before it is applied to the circuit we have got another module which we call decompressor. This decompressor module, this will be this will be decompress the patterns that we have got from the ATE and it will get back the it will get by get back the original patterns and the patterns to be loaded into the circuit. So, why can we compress test data? Because the deterministic test vectors they have do not care.

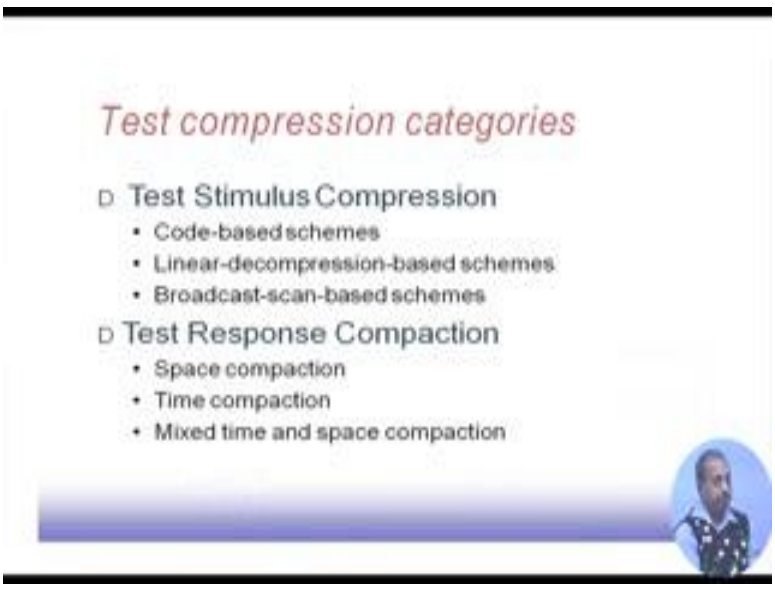
So, that the ATPG algorithm they have generated the test pattern. So, the bits which are not required to be set for particular falls to be tasted, so they are left as do not cares. So, this do not cares can be exploited. So, that we can we can match between the patterns and a number of patterns may lead to a one representative one that can replace all of them that way we can achieve the test compression.

(Refer Slide Time: 04:28)



So, this is the situation that test data volume versus gate count. So, if gate count is increasing in terms of Mega gates, and the volume of test data. So, it is increasing in terms of Giga bits. So, as the circuit size is increasing. So, you see this so many million gates. So, this is this is increasing the volume of test data is increasing at exponential wave. So, this is a requirement that we must have some sort of test compression mechanism incorporated.

(Refer Slide Time: 05:01)

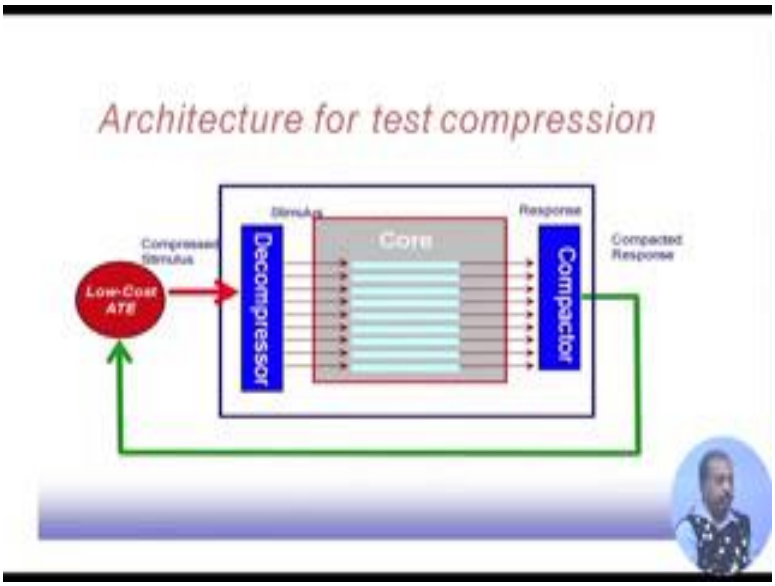


Now, this text compression can broadly be divided into 2 categories - one is the stimulus compression and another is the response compression. So, in this stimulus compression we are talking about the test patterns. So, we compress the test pattern so that we need to store less information on the ATE and the response compaction we will mean that we are storing we will be compacting the response that we get from the circuit may be I have got 10,000 patterns, but I do not store all the 10,000 responses. So, I store only may be a part of the responses maybe in some compress form maybe some modified form and then we compare we will do a comparison with that modified response only. So, we have got this, this response compaction. So, it has got space compaction space compaction it will try reduce the number of outputs, like if a circuit has got hundred outputs maybe we pass it to a space compactor that will reduce the number of output to let us say 10 or 20.

Similarly, we have got time compaction. So, time compaction. So, this will be reducing the. So, we do not store the response for all the 10,000 patterns. So, we will look into less number of instances and then we have got mixed time and space compaction. So, that is that is a combination of these 2.

On the other hand for the stimulus compression also there are several techniques some of them are code based schemes. So, these were some codes for doing the compression then there are some linear decompression based schemes. So, linear the decompression based scheme. So, they will be trying to use some sort of linear structures like a LFSR to perform this compression and there are a broadcast scan based schemes.

(Refer Slide Time: 06:54)



So, this is the typical situation of test compression. So, we have got this low cost ATE. So, as I was telling that the cost of the ATE is too large and if you are having, if you are using one ATE, low cost ATE then the cost of the ATE is determined by the memory requirement and the memory that it has and the number of channels. So, if you are, if you are designing a new versions of chips or new chips that are having more number of test patterns or more number of chip ins to be for texting then it does not mean that we should go for a new ATE all together because the ATE investment will be a big question. So, somehow we have to use this low cost ATE itself for testing this high end chip. So, how to do this thing? So, for that purpose in the ATE we will store the patterns in a compressed form the compressed stimulus. So, they will come to the circuit and before they are applied to the circuit under test. So, they are good, it is passed through a decompressor state.

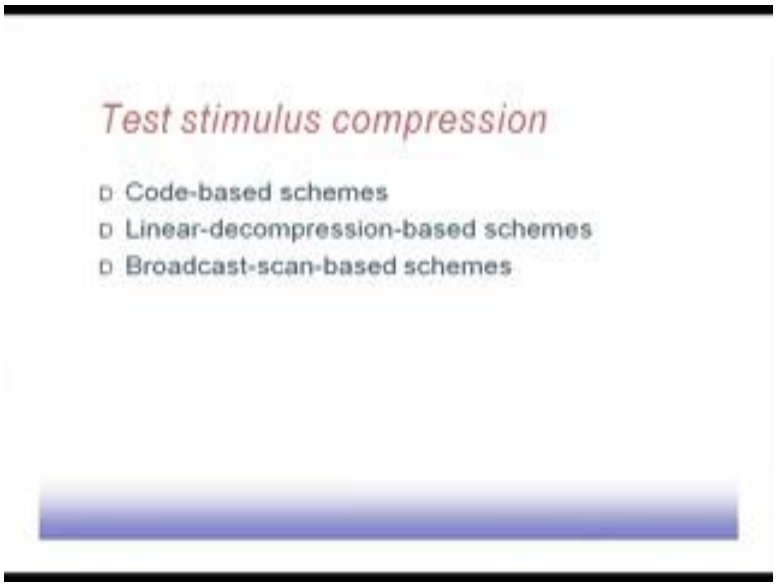
So, decompressor will convert this pattern into this stimulus the compressed stimulus into the actual stimulus and then these, after this circuit has given the response. So, it is there that outputs are collected by a compact that the response compactor and this response compactor the compacted response will be coming back to the ATE. So, comparison will be done in with respect to this compacted respond only. So, ATE will be will be comparing with respect this compacted response only.

So, many a times the circuit under test we call it code. So, this is the actually this test compression problem is more vital when we are having code based design. So, in code base designs what happens is that for entire system we take codes from different vendors. So, codes are may be soft description of the system maybe net list level description. So, for a system that had got CPU memory and say one IO controller. So, we may take the net list for CPU net list for memory and net list for IO controller from different vendors and integrate onto the same silicon floor to get the whole system, so to get the system on chip type of concert.

Now, in this process now how do you test the CPU, like I do not know the net list of the, I do not know the high level structural description of the CPU now I cannot generate the test patterns for that. So, text pattern also I have to rely on the vendor. So, these are called codes. So, code vendors they also providing the test pattern and they are telling me that if you apply these test patterns if you get this type of response then the code has been fabricated properly.

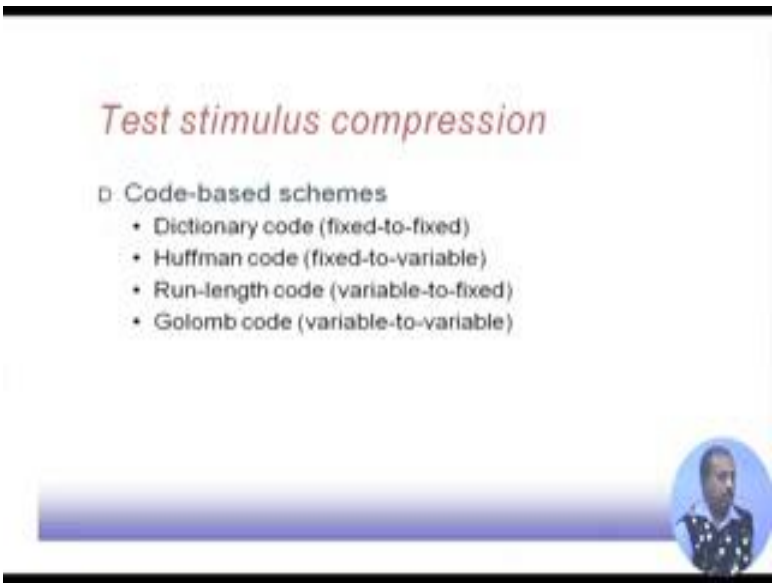
So, that way what was happened if you are integrating large number of course, then the amount of test data that you have to that is also increasing significantly and the code vendors since the code vendor does not know about the situation into which this code will be going to be placed then they cannot optimize on the test pattern, they cannot reduce the number of test pattern. So, they try to cover as many situations as possible. So, that way the test patterns it becomes huge and that test pattern it is given to the system integrator who takes all these course and integrate into a system. So, that way the integrator has got no option, but to apply all those test pattern. So, that is why this exponential increase in the test pattern size is, tested size that we have seen, that is becoming more prominent.

(Refer Slide Time: 10:45)



So, stimulus compression, as we have said that they it can been code based scheme, linear decompression scheme and broadcast scan based schemes.

(Refer Slide Time: 10:54)



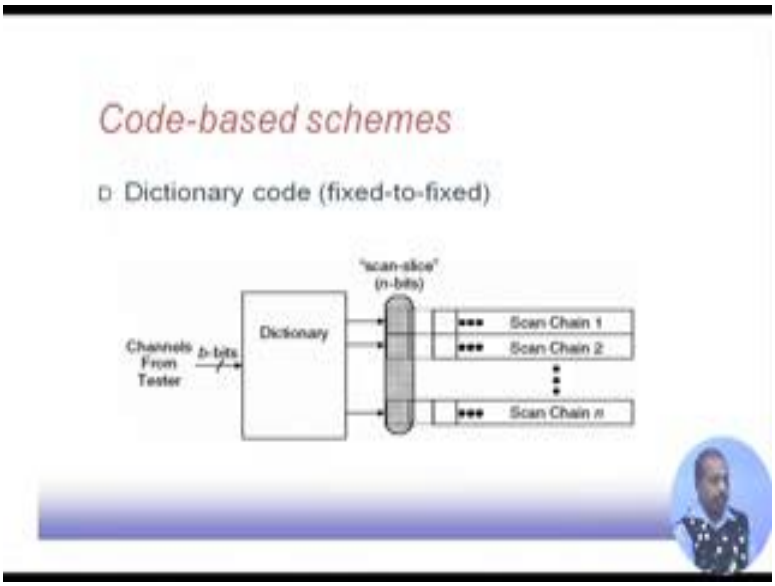
Now, code based schemes there are four different examples that will look into and they represent four different categories - the first one is fixed to fixed so; that means, you are test stimulus portion that you want to take for coding that is also fix and the code that is generated. So, that is also fix. So, input and output both of their width are fixed. Huffman then there is a fixed variable coding where the input stimulus chunk that we take for

coding that is that size is fixed for the length of the code is not fix that is variable. So, then we have got a variable to fixed. So, here we have got this thing the input chunk length is variable, but the code that is generated that is of fixed length and we have got variable to variable where the both input chunk and the code they are variable. So, we will pick up individual categories and try to see how those codings can be done. So, fixed to fixed coding. So, this is typically known as dictionary based coding.

So, basically we maintain a dictionary of codes and depending upon the input sequence. So, it take it take the code from the dictionary and use it for coding. Similarly fixed variable coding we have got Huffman code which is a lossless compression and then this 1 Ram link code is there for variable to fixed and Golumb code for variable to variable. One thing that we should keep in mind that is test stimulus compression it has to be lostless like if you look into this diagram. So, at this point I really want the original test patterns back. So, this compression whatever is done that must be a lossless compression.

On the other hand this side, this response compaction. So, response compaction need not be lossless because if it is lost less than all the information are anyway there. So, we are not going to save on the amount of data that we are going to transfer. So, this in most of the cases this response compaction is a lossy compression where we have got less number of output bits compared to the number of input bits; however, in this case this must be lost case we must be getting back the original patterns. So, the first one is known as dictionary based code.

(Refer Slide Time: 13:20)



So, for understanding this, we must understand something called a scan slice. Suppose we have got a circuit that has got this scan chain, scan chain 1 to scan chain n and from the ATE, this test bit are coming. So, test bits will be feeding this scan chains now in each cycle each clock cycle. So, each of these scan chain will be getting 1 bit of data that will get shifted to the next position. So, if there are n chains then I can, if you consider the same bit position for each of the scan chain so that construct that constitutes one scan slice. So, in the first clock cycle the first slice is loaded all the n chains they get their first bit then in the next clock cycle the next slice will be loaded. So, all the n chains will get the next bit. So, this way this loading will be done.

So, what happens is that from the ATE to the channel I get a b-bit pattern, b-bit width pattern. So, this is used to index this dictionary and from this dictionary the corresponding test pattern is found and that is actually loading the scans slice. So, the idea is that maybe if this is the full test pattern set that has got all this patterns in it, if it is a full test pattern set then we can see that, we find out the slices. So, this is the first slice this is the second slice. So, this way we find out the slices, this is the third slice.

Maybe this bit is, if I take the first slice may be the first slice is 1 0 0 0 1; that means, to the first scan chain I will be 1, seconds scan chain 0 and third scan chain 0, fourth scan chain 0, fifth scan chain 1. If there are five scan chain. Similarly in the next slice. So, this is called first one for the second slice I may have 0 0 0 1 0 that is the next bit that will be

fed into the five scan chain. So, this is the actually slices actually this is a if this is the test pattern this is a full test pattern. So, this full test pattern is going to fed to five different scan chain. So, I can say that it is divided into say five difference portions and this part is going to feed the first scan this is first scan chain 1, this is first scan chain 2, this is first scan chain 3, scan chain 4 and scan chain 5.

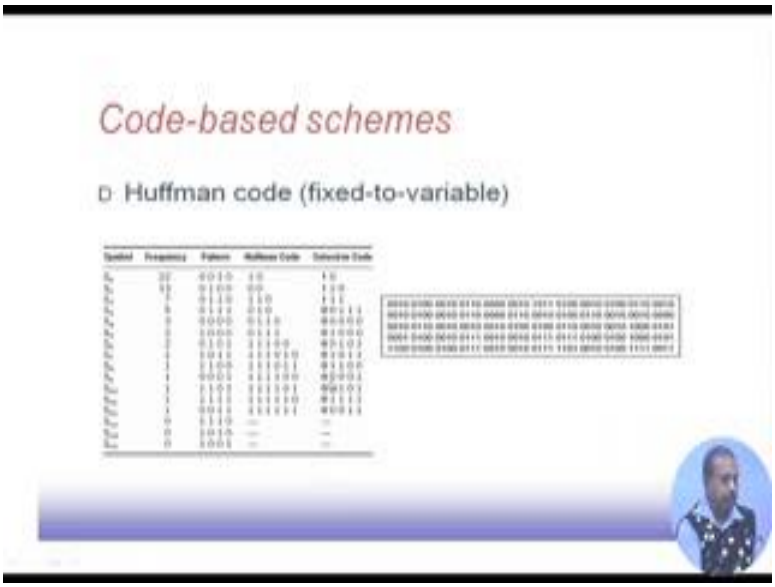
Now, out of that if you consider the first bits of each scan chain, so this one. So, they will constitute 1 slice. So, this is basically 1 sustain. Now if you consider second bit of each scan chain like this, this one. So, that will constitute the second slice. So, this is a second slice, this is actually the formation of slices now they are going to be fed to all the scan chain in this fashion. Now what we do in the dictionary try to see for this one this 1 0 0 0 1, if in the dictionary if I store all these pattern all the scan chain all the slices that are going to occur if I put them on a dictionary. So, this is a dictionary.

Now if this pattern 1 0 0 0 1 stored in this position then it is sufficient that from the tester I just tell what is the index i. So, this index i is told so that this decompressor that I said this is decompressor actually having the dictionary. So, from this channel I just get this index i and this index i is used to locate the corresponding pattern this, 1 0 0 0 1 and this 1 0 0 0 1 will be fed into this scan chain size. Like in this example that you are looking into, here this b-bits that are coming from the channel of the tester, so dictionary is a part of the decompression so (Refer Time: 17:52) to that particular slice is located and that slice is outputted onto this onto individual scan chain 1 bit for each scan chain, each scan chain.

Now, naturally, if my test pattern set is such that it generates different type of slices, different type of slices all possible slices are generated then of course, there is no saving. So, we cannot save anything in the dictionary because dictionary become too large, but if it happens that only slice, only some slices are there it is not that all possible slices are appearing as part of this test pattern sequence then of course, if the total possible slices are less then this dictionary can be of finite size and then dictionary whatever slices we have that have, that are stored into the dictionary. So, dictionary size is finite, so we can put it into a as a part of the decompressor. So, this is the dictionary based coding, fixed to fixed number of coding.

So, the saving that we can have is that from the dictionary I am getting b-bits. So, in the in the tester I need to store only the b-bits for individual slices. Now from the dictionary, dictionary size is of course, a factor, but otherwise there can be saving.

(Refer Slide Time: 19:25)



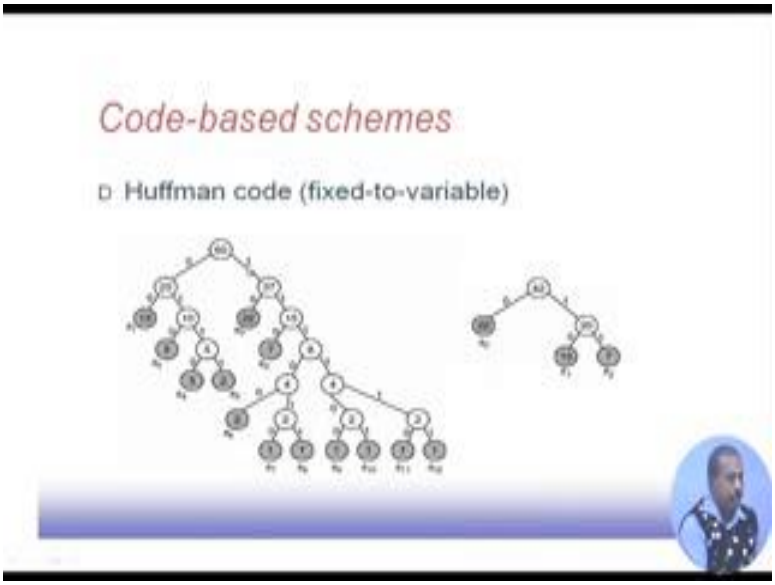
The next one is code based scheme which is known as; it is a fixed variable length coding which is known as Huffman coding. Now you see that suppose this is the test pattern set that we have, now in that we see that how many times this pattern 0 0 1 0 has occurred. So, 0 0 1 0 if you count, this is occurring once here 0 0 1, then this is once, this is once, then this is once. So, in this is way if you count you see that this particular pattern has occurred 22 times.

Then we find that the next pattern 0 1 0 0 it has occurred 13 time. So, what we do? We find out this individual patterns and count how many times they have occurred, this particular patterns like 1 1 1 0 1 0 1 0 and 1 0 0 0 they have not occurred at all. So, their frequency is 0, but others have occurred like this. So, in a fixed dictionary structure fixed to fixed length coding what you have done is we will store all these S 0 to S 12 this patterns on to the dictionary and for decoding we will just send the index of the dictionary.

But what happens is that for every since there are 12 entries, so for every slice I need to send a 4 bit address, the b is has to be 4 bit. So, that increases the complexity in the sense that the dictionary size and your this thing the bits that you need to transfer will be more.

So, instead of doing that what we can do? We can code it using some Huffman coding strategy, we can code it is some Huffman coding strategy such that the highest occurring pattern. So, this will be coded with 1 0, then the next pattern that is coded with 0 0 2 bits, the next occurring pattern is coded with 1 1 0, next occurring pattern coded with 0 1 0 like that.

(Refer Slide Time: 21:48)



So, you see that after when it gets this bits, when the decoder will get this particular bit sequence. So, it will work like say it gets a 0. So, once it is gets a 0. So, look into this structure say is the circuit, this is the ATE from where I am getting the bits. So, this is coming to the decompresser or decoder, is coming to the decompressor.

Now if we get a 0 at the line then we know that the next bit, we come to this side and then we again if we get a 0 we know that we have the symbol that we are referring to is S 1. So, this S 1 is 0 0. So, you see that this is the pattern. We have we have detected that that ATE is trying to send the pattern S 1 which corresponds 0 1 0 0 instead of sending this four bits the ATE could send only this 2 bits and that sufficient to identify this particular pattern.

Similarly, if after getting 0 if I get a 1 on this ATE channel; that means, it is, I could not come to any decision I have to go for the next input as well. So, this is 0. So, the next input is also 0 then I know that this is S 3. So, 0 1 0 is the code for S 3. So, 0 1 0 is a code for S 3. So, if you come to this point. So, you know that this going to be; this going

to be the pattern that you are looking for. So, this is the way this Huffman coding will work fixed variable length coding.

Now, another, this way this coding work for all these entries. So, they are to be, we have to have this big sorry; we have to store this big tree into the decoder and the decoder will follow this tree to do the operation to find out the actual code, actual pattern that it needs to construct from the code that is transmitted. So, that is known as that is a Huffman code. Now you see that what has happened is that the patterns whose frequency is more they have been coded with lesser length code. So, this is its frequency is the highest. So, it has been code with 1 0, next one is 13 so that has been coded with 0 0.

So, this way the coding has been done and also we can see that the way the coding is done is that we try to group the patterns. So, this S 1, S 3, S 4, we write them, we just say this S 0, S 1, S 2, S 3. So, we just write them in a sequence and then the lowest occurring pattern, so they are grouped. So, S 11, S 12 since they are occurring the least amount of time. So, they are grouped into 1 load whose occurrence is this, occurrence becomes (Refer Time: 24:59). So, the individual occurrence is one. So, I can say when I combine these 2 into 1 group the occurrence is 2.

Similarly S 9 and S 10, so they are occurring 1, only and they are combined into 2. So, these are also combined into 2. So, after this has been done, we are actually add these level. So, we have got these occurrences of patterns which are 2. Now again we combine this 2 occurrence group, so 2 and 2 that gives us 4 and similarly this gives us 4. So, in this way you see the pattern S 0 which is occurring 22 times. So, this is coming at a much higher level in this tree and in the decoding process. So, this is given a code like 1 0 whereas, for this pattern is S 12. So, it is given a like 1 1 1 1 1 1 so many, so that is it a 6 bit coding.

So, this S 12 has been coded into 6 bit coding whereas, S 0 has been coded with a 2 bit coding. So, since S 0 will be S 0 is occurring more frequently. So, S 0 will be sent more number of time, so it will lead to saving compared to this S 12.

In the normal case, in the fixed to fixed length dictionary case what will happen is that both S 0 and S 12 they will be coded with same number of bits as a result the savings will be less, but in this case since we are doing it on a lesser more frequently occurring number. So, they are given less code less white code. So, there we can have some saving.