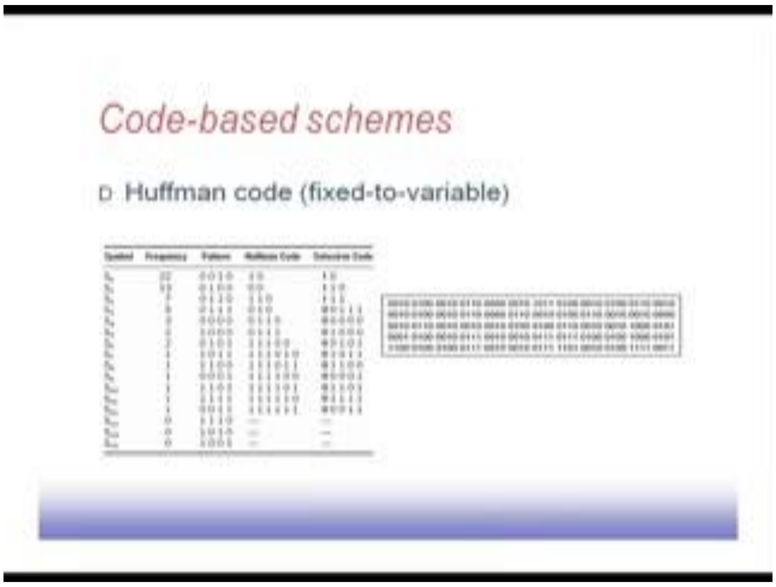
Digital VLSI Testing

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

Lecture - 29
Text Compression (Contd.)

Now, Huffman code to the problem that we had, is we were storing all the patterns in the in the dictionary.

(Refer Slide Time: 00:26)



Code-based schemes

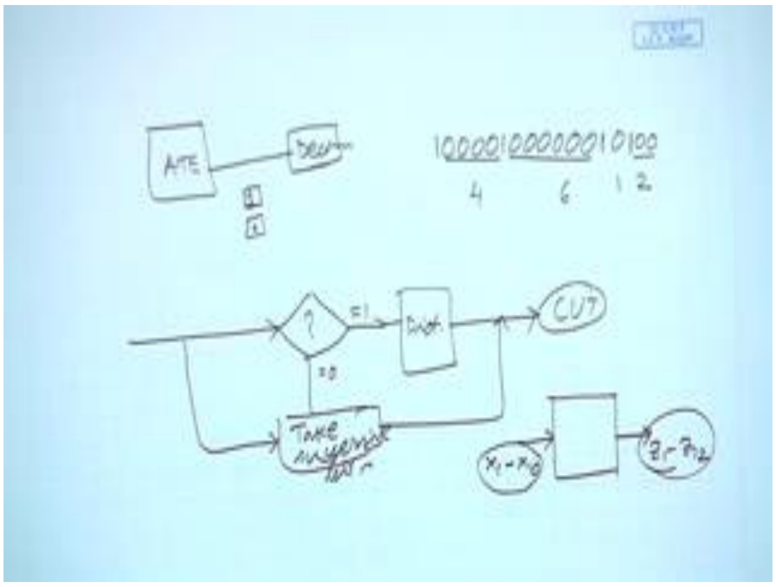
- Huffman code (fixed-to-variable)

Symbol	Frequency	Pattern	Huffman Code	Selective Code
S_0	22	0010	10	10
S_1	13	0100	00	110
S_2	7	0110	110	111
S_3	5	0111	010	00111
S_4	3	0000	0110	00000
S_5	2	1000	0111	01000
S_6	2	0101	11100	00101
S_7	1	1011	111010	01011
S_8	1	1100	111011	01100
S_9	1	0001	111100	00001
S_{10}	1	1101	111101	01101
S_{11}	1	1111	111110	01111
S_{12}	1	0011	111111	00011
S_{13}	0	1110	—	—
S_{14}	0	1010	—	—
S_{15}	0	1001	—	—

[illegible]

And as a result we have to assign ports for even for very less frequently occurring pattern like you see this pattern S 12. So, S 12 is occurring only ones and when it is occurring we are sending a code which is 6 bit vice whereas the original pattern is only 4 bit vice. So, we are losing a lot. So, selecting Huffman coding what it does is that it does not put all the patterns on to dictionary; it only put the patterns which are occurring very frequently on to the dictionary for the remaining patterns. So, it will not put in to the dictionary.

(Refer Slide Time: 01:08)

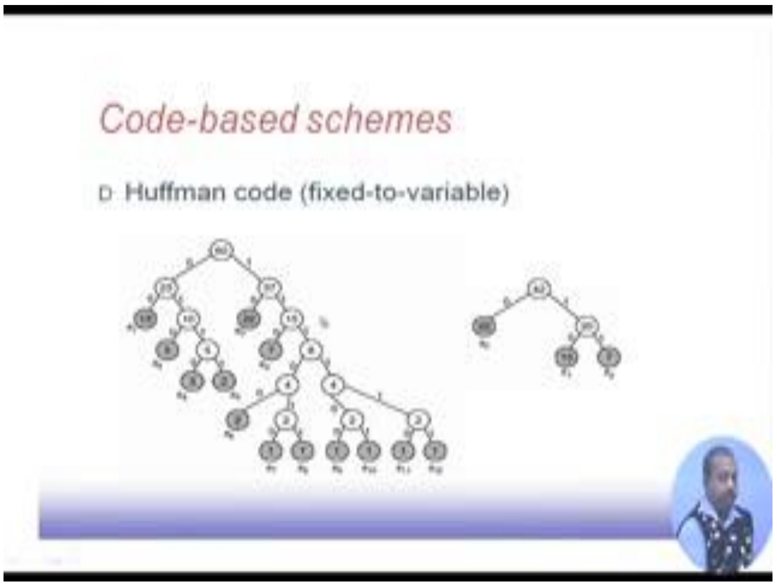


So, what happens is that well while I am doing this transmission from 80 to the decoder; 80 to the de compressor, the when I am doing this transfer the fast bit that is transfers. So, that identifies whether it is a dictionary pattern or a normal pattern. So, if the first bit that is transferred is a 1 that is 1; that means, in the next I am going to transfer some coded pattern and if this bit is 0 in that case it means that next bit I am transferring is not a pattern not a code but it is the actual pattern. So, if we do like this then for the pattern where we see that coding will these not going to be helpful. So, we can transfer them directly to the decoder. So, decoder will simply bypass the dictionary and whatever pattern is coming. So, it will put it on to the output. So, that it can do.

So, decoder the way it works is that its check whether the bit is 1 or not; bit is 1 equal to 1 or not, if it is equal to 1, in that case it will consult the dictionary and from the dictionary, it will get the pattern and that will be applied whereas, if the bit is 0 then will not consult the dictionary, it will take successive bits and it will just pass those successive bits to the output, to be applied to the circuit. So, here I have got to the circuit under test.

So this will be the decoder will be working like this. So, that way, we can save a lot like in this case you see that is S 0, S 1 and S 2, maybe we have decided that will store only 3 patterns in the dictionary; S 0, S 1 and S 2 . So, for them, the code that is assigned is 1 0, 1 1 0 and 1 1 1.

(Refer Slide Time: 03:19)



Code-based schemes

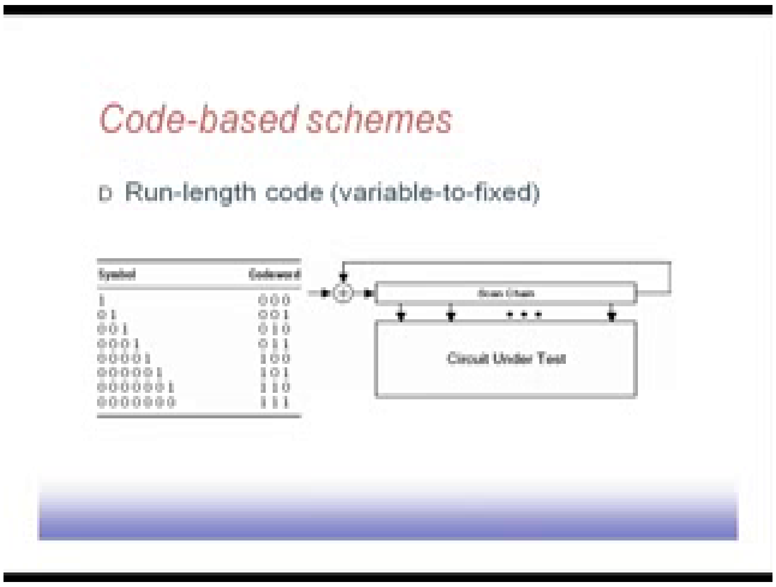
- Huffman code (fixed-to-variable)

Whereas, for the remaining pattern, we decide that will not keep them in the dictionary, for them the most significant bit is 0 and the remaining bits as the same as the pattern that we have it is 0 1 1 1, here also it is 0 1 1 1. So, it goes like this. So, the advantage that we get is that these 3 that we have at the decoder, it is much simpler. So, now, you see we have we have decided to keep only S 0, S 1 and S 2 in the dictionary. So, while constructing the Huffman tree at the first level we combine this S 1 and S 2 and this is occurring 13 times, this is occurring 7 times. So, only combined them, we get a node that has got 20 occurrences and with that this S 0 will be combined. So, we will get 42 occurrence nodes at the top. Now, the code of S 0 is given by 0 and only 1 bit coding is sufficient for S 0, similarly so previously S 0 was requiring 2 bit coding, you see 1 and 0, now it is required only 1 bit coding of course, 1 extra bit is needed which is identifying it, that it is a code not the pattern that is there.

Similarly, for this 13 for S 1 and S 2, the number of bits needed is equal to 2. Of course, again in same thing that 1 extra bit is needed to identify that it is from the; it is a directly pattern; it is a code to be consulting consulted with the dictionary and not a pattern directly. So, this is the advantage of this selective Huffman code. So, selective Huffman code, it can give rise to farther compression compared to the normal Huffman code and the second advantage is that the dictionary size can be drastically reduced.

So, the designer can do it trade off like if you if you think that I do not want to much of effort on the much of area for this holding the dictionary; in the dictionary size can be reduced and by and the dictionary you will be able to accommodate only first few most frequently occurring patterns, but the completion will suffer, but at the same time that may not be that high because the most frequently occurring patterns are definitely covered by the coding.

(Refer Slide Time: 05:23)



Code-based schemes

- Run-length code (variable-to-fixed)

Symbol	Codeword
1	000
01	001
001	010
0001	011
00001	100
000001	101
0000001	110
0000000	111

Scan Chain
Circuit Under Test

Next, we will look into another strategy which is known as run length coding. So, run length coding is like this. So, it first of all it sees what is the size of the runs of 0s like if you see a test pattern. So, if it is it is pattern is say 10001000010 like this, now you see that at this part I am translating 5 0s, at this part I am translating 4 0s. So, instead of doing this, I can say that initially I have got say there are 2 0 0. So, initially I have got a run of 2 0s then I have got a run of 1 0, then I have got a run of 6 0s and then I have got a run of 4 0s. So, this whole thing can be represented like this.

So, if you just tell for how many what is the run of 0s that is that we are seeing while doing this transfer then we can say that we will be we can transfer those many those many bits only the how many what is the run length of that of that thing. So, this is like this. So, this is; so, this first of all we identify the symbols that can be. So, this single 1 that is no 0 is there that that is one case then there is a single 0 followed by 1, 2 0 followed by 1, 3 0 followed by 1, ultimate we have got 6 0s that is the maximum say.

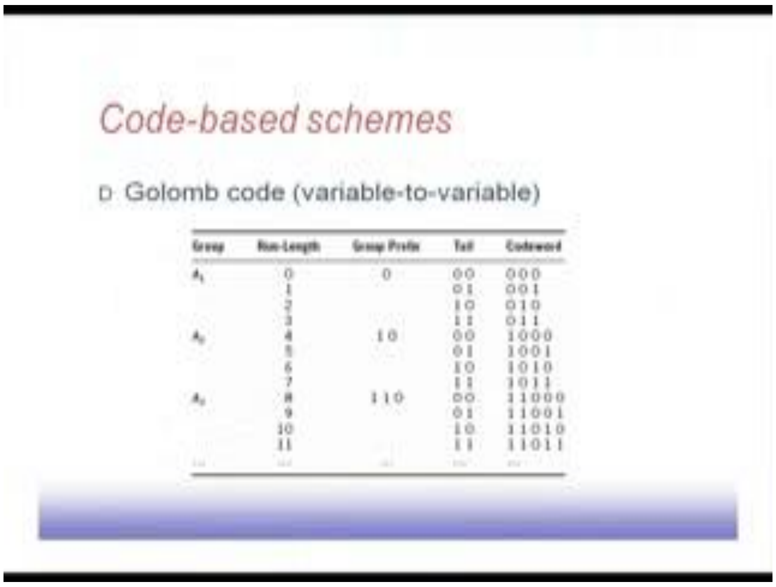
So, this is there so, what is done? This one; the symbol 1 is coded with 3; with the code word is 3 0, the 0s run of 1 is coded as 0 0 1, run of 2 is coded as 0 1 0. So, till this much you see that we are either transmitting more number of bits in the code word or we are transmitting the same number of bits at the code word, but from that point onwards, you see that this is not going to be same. So, this is going to be 0 0 4 bits, instead of sending 4 bits, we are sending 3 bits. So, what the decoder can see at the de compressor can see is that it just checks the line coming from the ATE, so if it finds next 3 bits as 0 0 0, it will know that the actual the symbol is 1 if it finds that the next three bits are 1 0 0, it knows that in the scan chain I have to put in this particular pattern 0 0 0 0 1.

So, that is to be putting. So, at the de compressor in, you can de-compute the test pattern given the code word information. So, if the code word code word that in the dictionary in the ATE, we are storing the code word and instead of storing large runs of 0s. So, we are storing the corresponding codes only that code is getting transferred to the decoder and the decoder is intern computing that the original symbol. Now what happens is that many a times so, wild testing the pattern so, while shifting the pattern so, you see that the pattern is going to be same like say this is the in this case say this is the scan chain. So, in the scan chain, we already have some pattern, now the bit that is that we are going to shift if it is same as the previous pattern that is shifted out then I can say that this is this result is this bit is 0 then what will happen?

The same bit that is coming here will be coming at this point it will be shifted here. So, if this to the next bit to be shifted is same as the last bit here then I can shift in a 0. So, from the ATE actually this input is coming. So, if I have got more number of 0s here then the coding will be more efficient. So, if we have the scan chain structure slightly modified. So, that is the ATE input instead of feeding this scan chain directly if it feeds this XOR gate and this scan chain bits are bit that is scanned out from the scan chain. So, it is XOR with the next input bit then it is very much possible that we have got a large number of 0s generated in the sequence that has to be fed here because the bits maybe same and that happens the due to a correlation between the test patterns and there are do not cares in the test pattern. So, do not cares can be filled up so that we can we can get more similar type of test pattern so that this 0s will occur more frequently so that bits will be same.

As a result, I can feed a 0 here. So, this run length code is a variable to fix length code by modifying the scan chain architecture slightly we can increase the number of 0s that we have to shift into this from that we have to shift from the ATE. So, decoder can the compression mechanism can take help of that information. So, instead of storing the test pattern, we are actually storing the difference and is the difference is having more number of 0s which is very much likely then we can we make a shift in less number of we may shift in we may have to shift in less number of bits because long runs of 0s can be found.

(Refer Slide Time: 10:59)



Code-based schemes

- Golomb code (variable-to-variable)

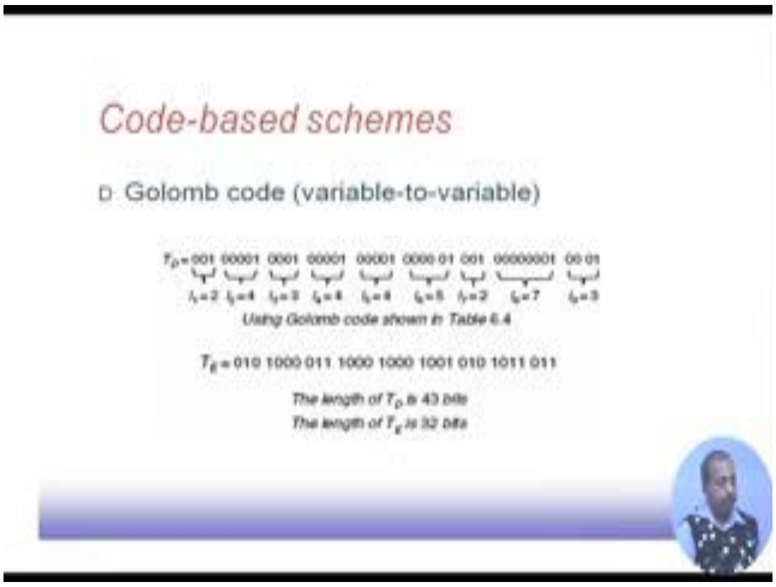
Group	Run-Length	Group Prefix	Tail	Codeword
A_1	0	0	00	000
	1		01	001
	2		10	010
	3		11	011
A_2	4	10	00	1000
	5		01	1001
	6		10	1010
	7		11	1011
A_3	8	110	00	11000
	9		01	11001
	10		10	11010
	11		11	11011
...	...	...	...	...

Another could based scheme is known as Golomb code. So, this A variable to variable length coding so, here we have got these groups like a one group. So, A 1 group so, what is done is the original pattern say original run length that we have. So, it is divided into groups of equal sizes. So, A 1; the group A 1, it has got the run length 0, 1, 2 and 3, group A 2 it has got run lengths 4, 5, 6 and 7, A 3 has got 8, 9, 1,0, 11. So, like that so run lengths are grouped into equal sized partitions you can say then for each group we have a group prefix; for group A 1 the group prefix is 0; for group A 2 the group prefix is 1 0, for A 3 it is 1 1 0. Now, individual run lengths within a group, so they are coded with the individual tail patterns like say there are 4 run lengths in each groups. So, they are to be coded as 0 0 0 1 1 0 and 1 1. So, the run length of 0 is finally, coded as group prefix followed by the tail 0 0 0.

Similarly, run length of 1 will be coded as 0 0 1, run length of 3 is coded as 0 and then tail is 1 1. So, 0 1 1, so when you go for larger run length, you see lots of savings like when I have got a run length of 11 you see the group prefix is 1 1 0 and tail is 1 1. So, this 5 bits are sufficient for sending the things.

So, this way when we are having when the run lengths are small then the group prefix is also small. So, that way it is saving the bits in the group prefix and the tail part is of course, same for all the cases. So, group prefix part, if a pattern is occurring, if you get a longer run length then this lead to saving.

(Refer Slide Time: 12:56)



Code-based schemes

- Golomb code (variable-to-variable)

$$T_D = \underbrace{001}_{l_1=2}\ \underbrace{00001}_{l_2=4}\ \underbrace{0001}_{l_3=3}\ \underbrace{00001}_{l_4=4}\ \underbrace{00001}_{l_5=4}\ \underbrace{000001}_{l_6=5}\ \underbrace{001}_{l_7=2}\ \underbrace{00000001}_{l_8=7}\ \underbrace{0001}_{l_9=3}$$

Using Golomb code shown in Table 6.4

T_E = 010 1000 011 1000 1000 1001 010 1011 011

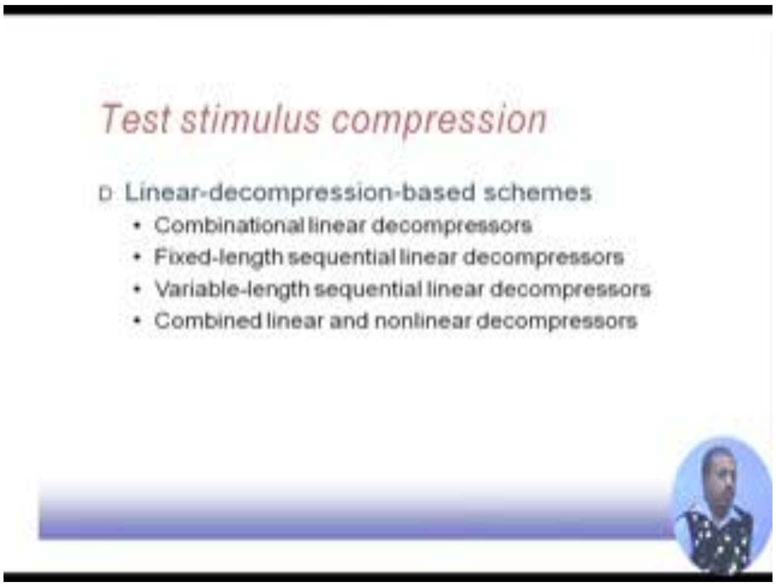
The length of T_D is 43 bits
The length of T_E is 32 bits

So, this is a typical example like say this one is Golomb code variable to variable. So, this is the test pattern that we have now, you see this is a run length of 2, this is run length of 4, this is run length of 3, this is 4, again 4. So, like that we have now, if you just do the corresponding coding. So, this is run length of 2. So, run length of 2 is coded as 1 sorry 0 1 0 you see in the instead of sending 0 0 1, we are sending 0 1 0 then I have got run length of 4. So, run length of 4 is coded as 1 0 0 0. So, this instead of sending these 5 bits, we are sending 1 0 0 0 similarly, this 1 this is a run length 3. So, that is again coded as 0 1 1. So, by consulting this table, we can see that we can get the corresponding um code words from this table and those code words will be used here for coding.

So, this original TD; it had 43 bits. So, when we do this encoding. So, this comes down to 32 bits. Now, this is a very common situation and. In fact, we can do many more wish you many more modifications to the test pattern say as we have already seen that the test patterns will have large number or do not care. So, this do not cares can be filled with 0.

So, that we can have we can say that will get the Golomb code will be benefited because it will find more number of larger run length of 0s and some there is a another modification where we considered both run length and 0 and 1. So, there has to be another bit which will be doing this compliment from 0 to 1. So, it tells whether it is a run of 0s or run of 1s and then the run length. So, if that is done then also there will be saving when we are looking into the coding.

(Refer Slide Time: 14:59)

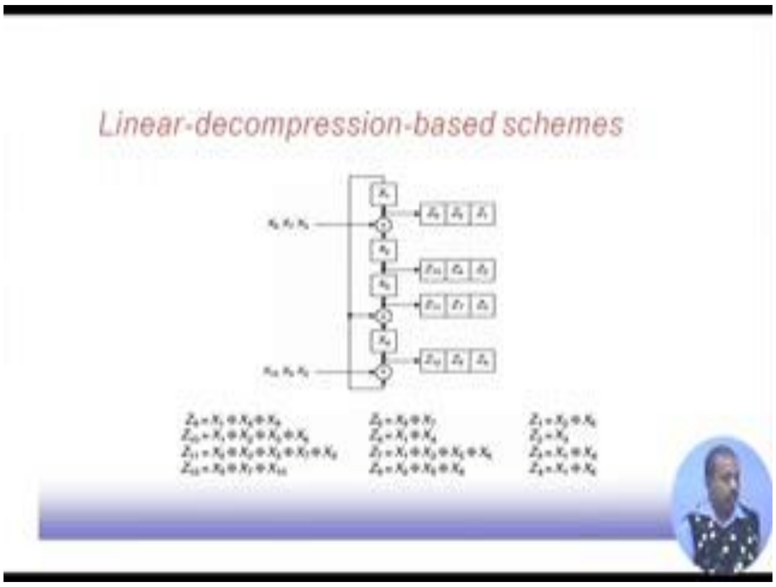


Test stimulus compression

- Linear-decompression-based schemes
  - Combinational linear decompressors
  - Fixed-length sequential linear decompressors
  - Variable-length sequential linear decompressors
  - Combined linear and nonlinear decompressors

So, this dictionary based coding techniques; the main problem that we have is that we have got this dictionary to be maintained and this dictionary size is it is an issue and many a time what happens is that this dictionary is not we cannot keep this dictionary to be very large as a result compression ratio may not be very big. So, next will look into linear decompression based schemes and under that category will look into several sub categories.

(Refer Slide Time: 15:34)



Linear-decompression-based schemes

$Z_9 = X_1 \oplus X_4 \oplus X_9$
$Z_{10} = X_1 \oplus X_2 \oplus X_5 \oplus X_6$
$Z_{11} = X_2 \oplus X_3 \oplus X_5 \oplus X_7 \oplus X_8$
$Z_{12} = X_3 \oplus X_7 \oplus X_{10}$

$Z_5 = X_3 \oplus X_7$
$Z_6 = X_1 \oplus X_4$
$Z_7 = X_1 \oplus X_2 \oplus X_5 \oplus X_6$
$Z_8 = X_2 \oplus X_5 \oplus X_8$

$Z_1 = X_2 \oplus X_5$
$Z_2 = X_3$
$Z_3 = X_1 \oplus X_4$
$Z_4 = X_1 \oplus X_6$

What do you mean by linear decompression based scheme is that say this is a structure, let us say that this is a structure where we have got some flip flops and some XOR gate this is a shift register and in between we have got this XOR. Now if this initial, if this cells initially have got X 1, X 2, X 3 and X 4 as its inputs and we have got this X 5 , X 6 , X 7 , X 8. So, they are fed as input to this XOR gate since successive cycles.

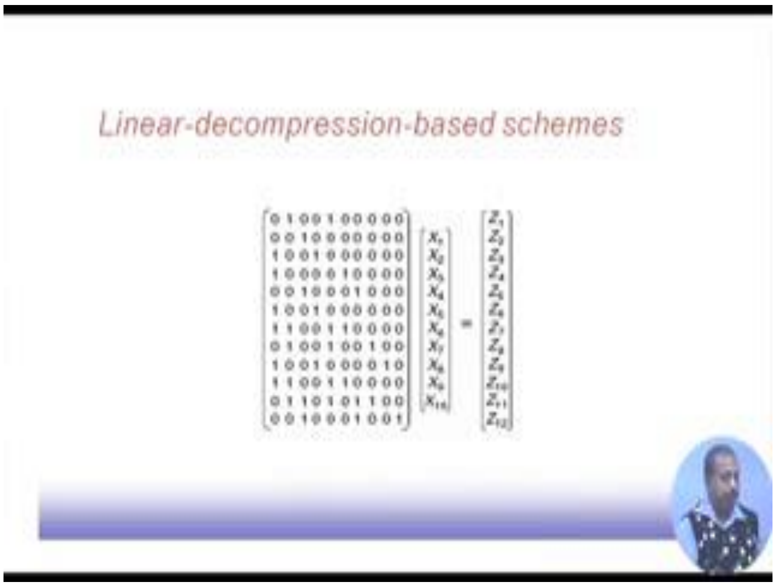
Then what will happen? So, if this machine is made to operate then what is Z 9? So, Z 9 is Z 9 is nothing, but this. So, this say let us what is Z 4? So, Z 4 we have to start with this one because first will get Z 4. So, Z 4 is X 1 XOR X 6 why because this X 1 is coming to this XOR gate and this X 6 is coming to the XOR gate as a result this Z 4 is a going to be equal to X 1 XOR X 6. Now what is Z 3? Z 3 is nothing, but this X 4 XOR X 4 because Z 3 is getting this X 1 from here and it is getting X 4 from here. So, these 2 values are XOR to get Z 3.

So, similarly this Z 1 So, Z 1 is nothing, but this X 2 XOR with X 5. So, this X 5, X 7, X 9, they are actually fed in sequence. So, initially these values X 1, X 2, X 3, X 4 are loaded into this cell into this flip flop and then this X 5 is fed, X 5 and X 6 is fed in the first cycle. So, they are added by this adder by this XOR gate and accordingly Z 1, Z 2, Z 3, Z 4; these outputs are computed then after one cycle what will happen this X 7 will come here, this X 8 will come here. So, accordingly this Z 5, Z 6, Z 7, Z 8, they will get computed so, but why this time there has been some shifts, if we explore properly this can be seen that Z 5 is nothing but X 3 XOR X 7 because X 3 will come here.

X 3 will get shifted to up to these points. So, of course, there are many more things, but after doing the simplification we will it can be found that the Z 5 is X 3 XOR X 7. So, what is happening is that you see ultimately we have got this pattern Z consisting of Z 1 to Z 12. So, Z 1 to Z 12, 12 bit pattern we are getting here and for that purpose we have stored this per X 1 to X 10 into we have to apply this X 1 to X 10 to this machine to get the values Z 1 to Z 12. So, if Z 1 to Z 12 happens to be my test pattern; Z 1 to Z 12 happens to be my test pattern and I have this machine available then I need to store this X 1 to X 10 only has the input to this machine. So, input to this machine is, I can consider this whole thing as if I have a machine here where the input is 12 10 bit input consisting of X 1 to X 10 and as an output it can generate Z 1 to Z 12.

So, 12 bit output it can generate, if this happens to be my test pattern. So, I do not need to store 12 bits in the in the ATE, but I need to store only this 10 bit. So, this 10 bit can be given to the machine and the machine de-compressor; the de-compressor will generate the 12 bit patterns and every if ATE has got two channel then in the two channel. So, I am sending this X 5, X 7, X 9 and X 6, X 8, X 9 serially and this machine it is generating this test pattern. So, if i if it is as if it has got force scan chains for example, then these individuals scan chains are being fed by these 4 lines. So, this is the basic idea of the linear decompression strategy. So, we have got a linear machine which performs the decompression and it helps it is all the operation that it does is linear operation you see ultimately Z is that we have got. So, they are nothing, but linear combination of this X value. So, I given this machine, can we find a set of X values that can give rise to these the Z values that is the main question that you have.

(Refer Slide Time: 20:33)



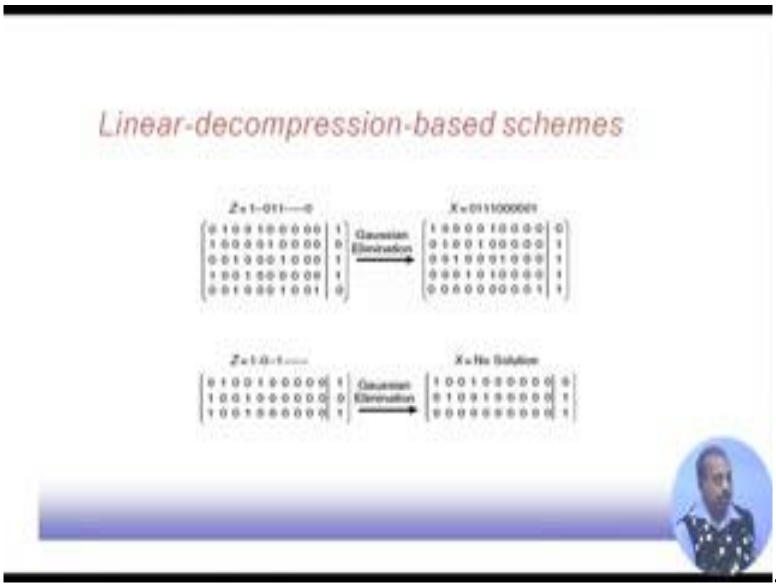
Linear-decompression-based schemes

$$\begin{bmatrix} 0&1&0&0&1&0&0&0&0&0 \\ 0&0&1&0&0&0&0&0&0&0 \\ 1&0&0&1&0&0&0&0&0&0 \\ 1&0&0&0&0&1&0&0&0&0 \\ 0&0&1&0&0&0&1&0&0&0 \\ 1&0&0&1&0&0&0&0&0&0 \\ 1&1&0&0&1&1&0&0&0&0 \\ 0&1&0&0&1&0&0&1&0&0 \\ 1&0&0&1&0&0&0&0&1&0 \\ 1&1&0&0&1&1&0&0&0&0 \\ 0&1&1&0&1&0&1&1&0&0 \\ 0&0&1&0&0&0&1&0&0&1 \end{bmatrix} \begin{bmatrix} X_1 \\ X_2 \\ X_3 \\ X_4 \\ X_5 \\ X_6 \\ X_7 \\ X_8 \\ X_9 \\ X_{10} \end{bmatrix} = \begin{bmatrix} Z_1 \\ Z_2 \\ Z_3 \\ Z_4 \\ Z_5 \\ Z_6 \\ Z_7 \\ Z_8 \\ Z_9 \\ Z_{10} \\ Z_{11} \\ Z_{12} \end{bmatrix}$$

If you just this whole operation can be represented by this particular matrix. So, you see that in this case after doing this analysis we have found Z 1 to be equal to X 2 XOR X 5. So, you see that here this matrix this X 2 and X these 2 bits are 1 less than or equal to 0. So, when you do a multiplication X 2 XOR X 5 becomes equal to Z 1 similarly, X 3 becomes equal to Z 2.

If you look into this Z 2 equal to X 3, in this that is represented by this matrix. So, whatever competition we have seen here, it can be represented by this matrix and by simple matrix multiplication we can get the test pattern. So, if there are large numbers of do not cares then of course, we do not have to this matrix can be made simple. So, we do not have to have large number of rows. So, that way we can see.

(Refer Slide Time: 21:34)



Linear-decompression-based schemes

$Z = 1{-}011{-}{-}{-}{-}0$ $\qquad$ $X = 0111000001$

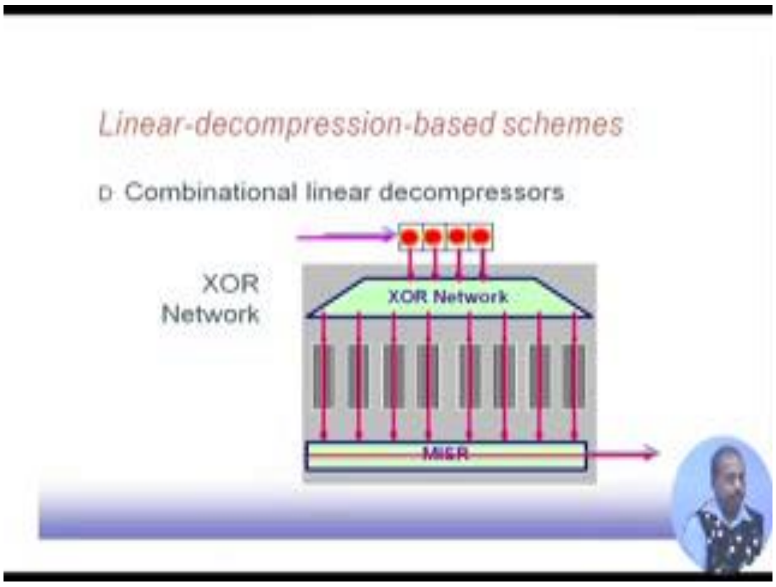
$$\left(\begin{array}{cccccccccc|c} 0&1&0&0&1&0&0&0&0&0&1\\ 1&0&0&0&0&1&0&0&0&0&0\\ 0&0&1&0&0&0&1&0&0&0&1\\ 1&0&0&1&0&0&0&0&0&0&1\\ 0&0&1&0&0&0&1&0&0&1&0 \end{array}\right) \xrightarrow[\text{Elimination}]{\text{Gaussian}} \left(\begin{array}{cccccccccc|c} 1&0&0&0&0&1&0&0&0&0&0\\ 0&1&0&0&1&0&0&0&0&0&1\\ 0&0&1&0&0&0&1&0&0&0&1\\ 0&0&0&1&0&1&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0&1&1 \end{array}\right)$$

$Z = 1\,0{-}1{-}{-}{-}{-}$ $\qquad$ $X = \text{No Solution}$

$$\left|\begin{array}{cccccccccc|c} 0&1&0&0&1&0&0&0&0&0&1\\ 1&0&0&1&0&0&0&0&0&0&0\\ 1&0&0&1&0&0&0&0&0&0&1 \end{array}\right| \xrightarrow[\text{Elimination}]{\text{Gaussian}} \left|\begin{array}{cccccccccc|c} 1&0&0&1&0&0&0&0&0&0&0\\ 0&1&0&0&1&0&0&0&0&0&1\\ 0&0&0&0&0&0&0&0&0&0&1 \end{array}\right|$$

So, how does it work? Suppose, my Z is equal to this 1, this is my test pattern 1 then these 2 bits are do not care then I have got 0 1 1 again, all these bits are do not care and this is a 0 bit. So, this is the test pattern set given by the ATPG. Now, for this do not care I really do not bother. So, whatever this de-compressor generates I do not bother about it, but the de-compressor must generate these bits; the clear bits properly. So, it must generate these bits properly. So, how to do this thing? So, we have just argument this with the output color 1 0 0 1 0 1 1 0 and then do a Gaussian elimination. So, when do a Gaussian elimination. So, we find that this X may be equal to 0 1 1 0 this particular pattern. So, that it will be giving us this solution. Whereas, in this particular case in this particular case. So, Z is this, but here the Gaussian elimination cannot give any solution. So, that way it is. So, this particular pattern cannot be generated by this machine and why it has not been able to generate, because you see that two rows have been become same. So, there Gaussian elimination will convert it this will be all 0. So, it will be not be able to do the solution finding. So, this Gaussian elimination cannot find it so whatever it may be, Gaussian elimination may be some other technique, but ultimately what do you want is that this X value are to be we have to find this X value so that this k r bits can be generated. So, if we can do that then we are then we can say that we can generate the test pattern using this X values.

(Refer Slide Time: 23:32)



Linear-decompression-based schemes

- Combinational linear decompressors

So, what works finally, is that the combinational linear de-compressor. So, this one; the XOR network is there and this XOR network will feed the scan chain and this scan chain output will be coming out. So, again the scan out bits they will be coming to the MISR. So, MISR will do the analysis and then there is a response will be put on to the output. So, this is the combinational linear de-compressor. So, this input is coming here from the ATE and this XOR network is doing the operation. So, continue in the next class.