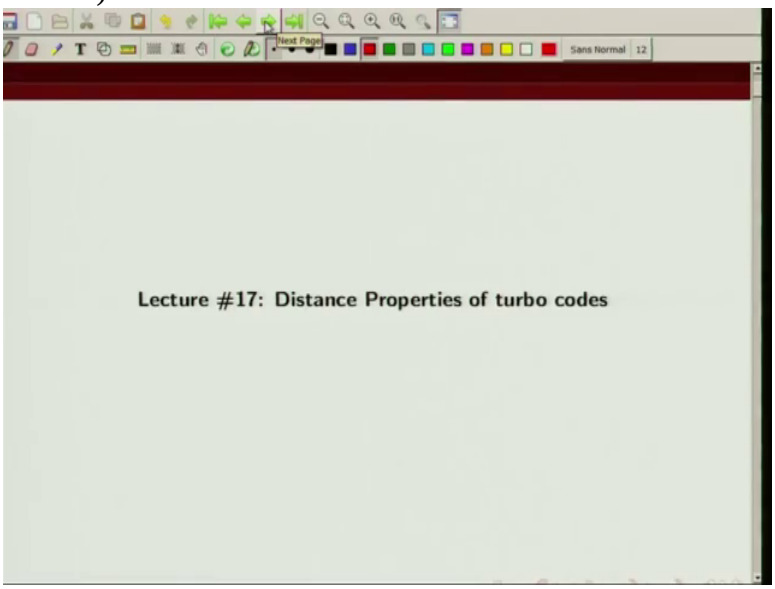
An Introduction to Coding Theory
Professor Adrish Banerji
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Module 08
Lecture Number 39
Distance Properties of Turbo Codes

(Refer Slide Time 00:13)

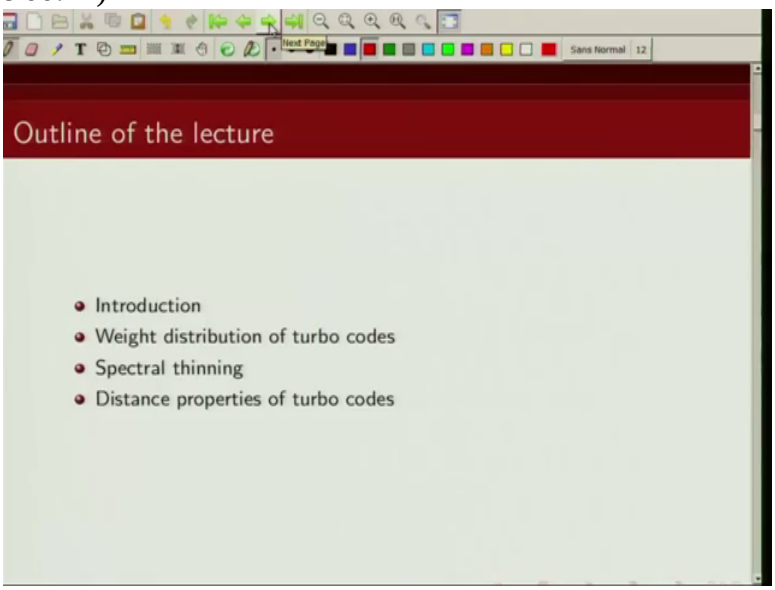
Welcome to the course on An Introduction to Coding Theory. I am Adrish Banerji from I I T Kanpur.

(Refer Slide Time 00:20)



In today's lecture we are going to talk about distance properties of turbo codes.

(Refer Slide Time 00:27)

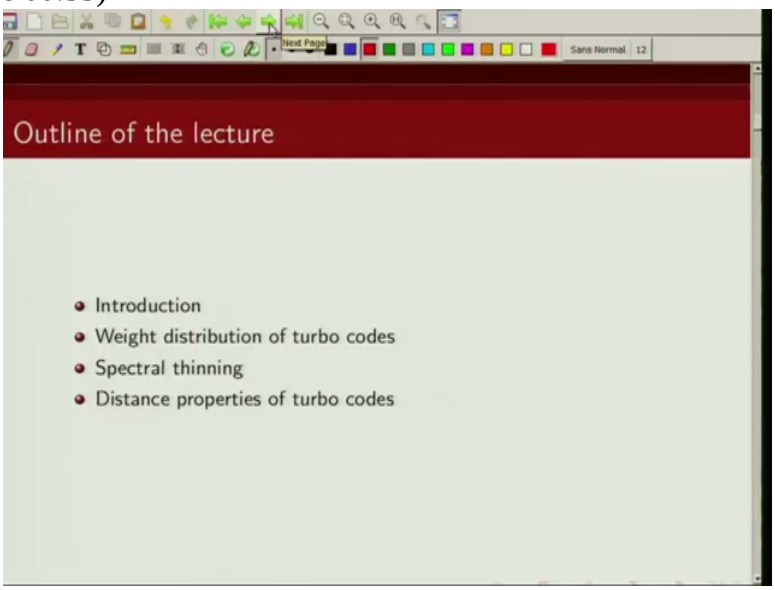


So this is the outline of today's talk. We will first give an example of weight distribution of a convolutional code

(Refer Slide Time 00:36)

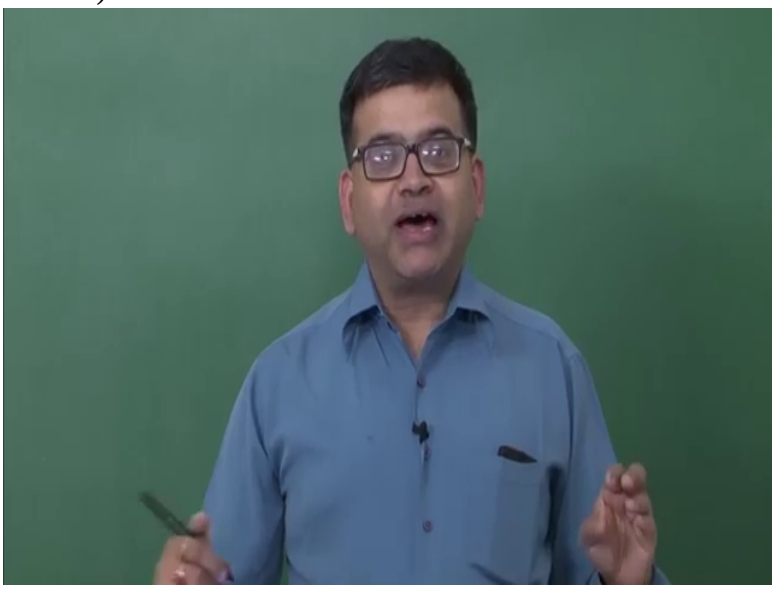
and we will compare it with the weight distribution of a concatenated code. And we will see what effect interleaver has on the weight distribution of a concatenated code. Then we will introduce this concept of

(Refer Slide Time 00:53)



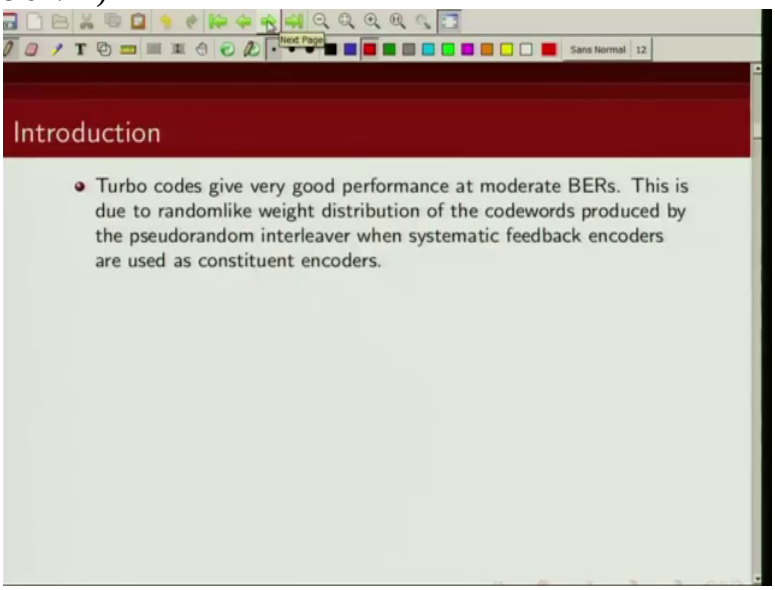
spectral thing which is visible in turbo code if we use feedback encoders and finally we will show how we can make use of this concept of uniform interleaver to derive

(Refer Slide Time 01:09)



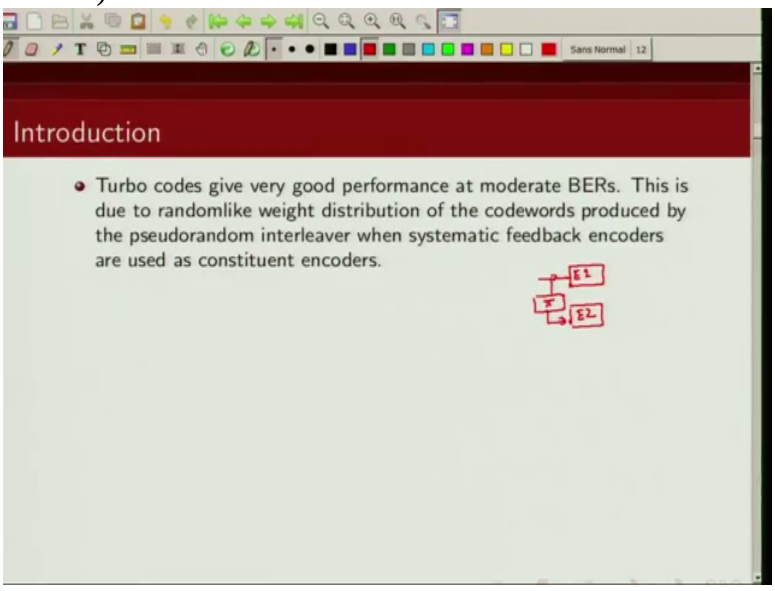
expression for average weight enumerating function of turbo code.

(Refer Slide Time 01:17)



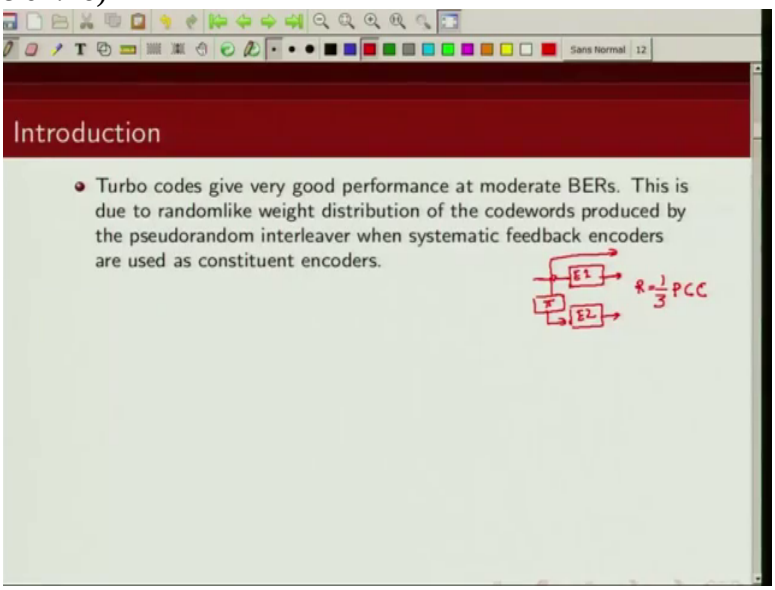
So as we know turbo codes have very good performance at low signal to noise ratio and at moderate b e r and this is due to its random like weight distribution which is produced when we consider, pseudorandom interleavers combined with systematic feedback encoders. If you recall a turbo code is a parallel concatenation of two consequent encoders. So this is encoder 1, encoder 2. Information bit enters encoder 1 and the interleaved version of information enters

(Refer Slide Time 01:58)



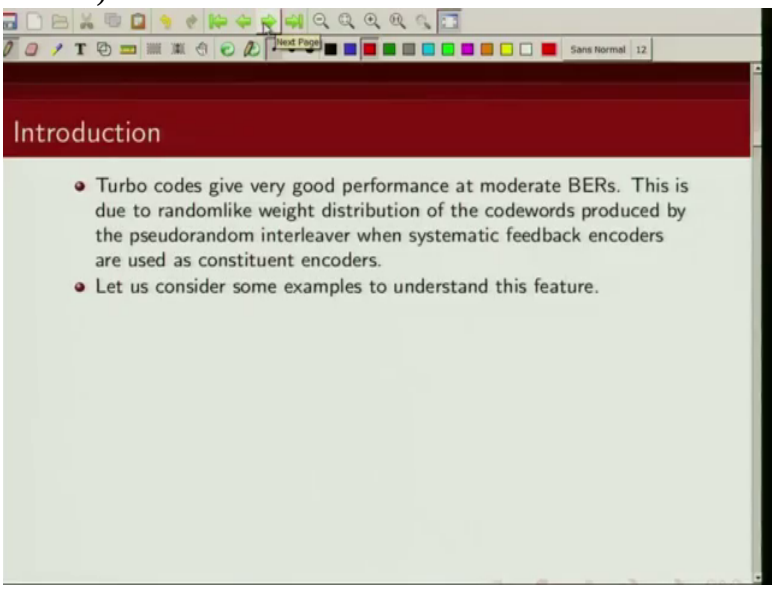
encoder 2. These are the parity sequence coming out of first encoder and second encoder and then this is systematic, so typically you have this example of rate one third parallel concatenated code.

(Refer Slide Time 02:16)



Now we will try

(Refer Slide Time 02:21)

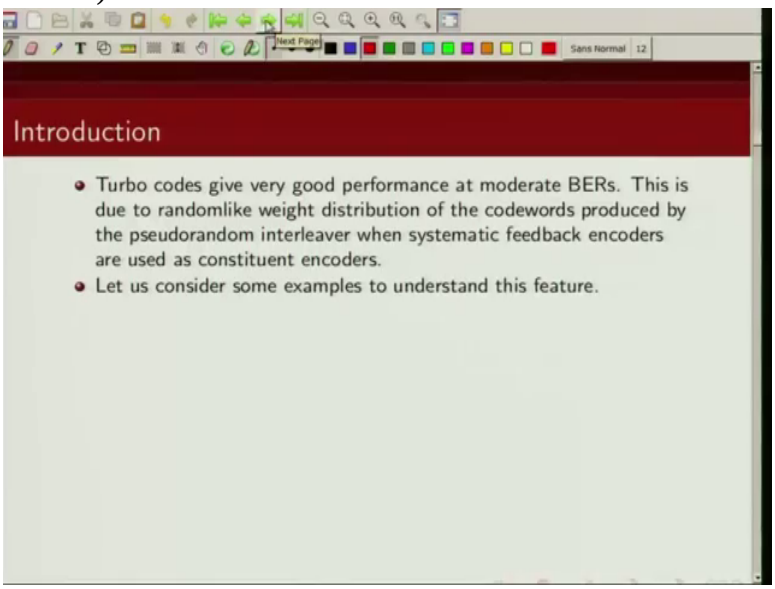


to see, as I said,

(Refer Slide Time 02:23)

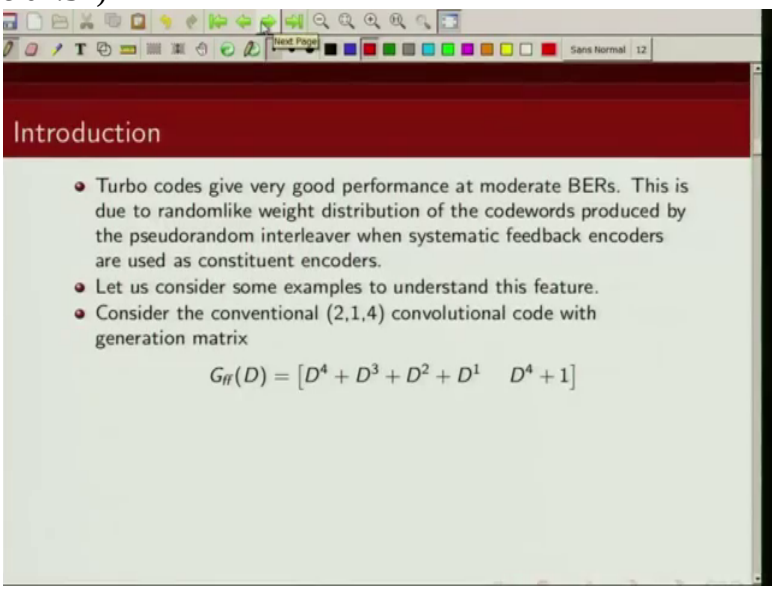
we will try to look at the distance spectrum of the concatenated code and see how it is different from a

(Refer Slide Time 02:30)



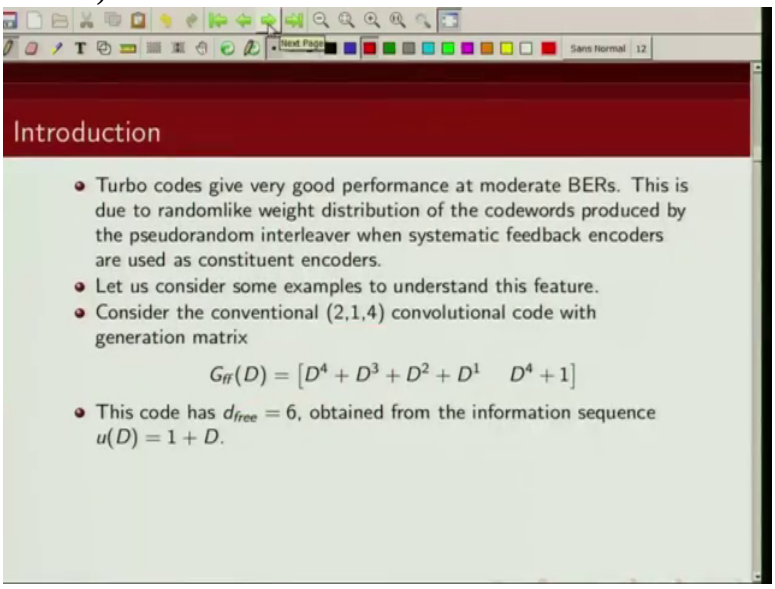
convolutional code.

(Refer Slide Time 02:31)



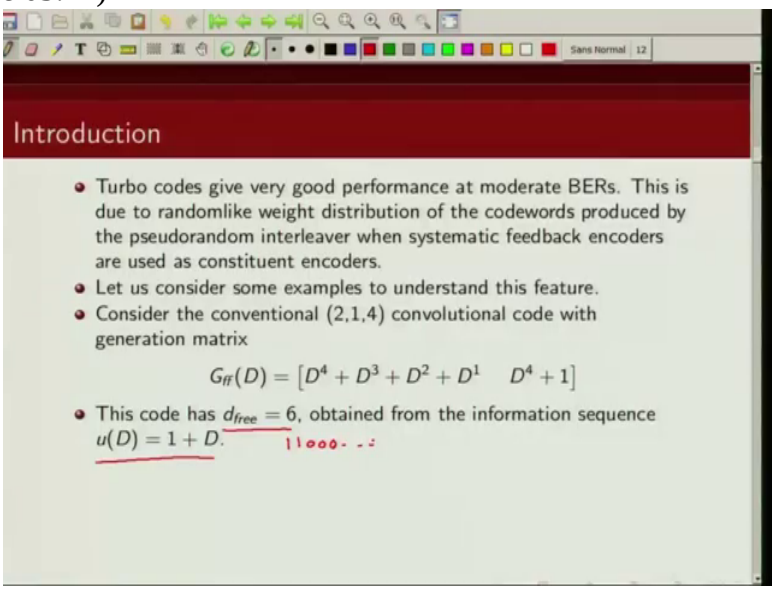
So let us start with an example. So we are considering a rate one half, rate half convolutional code with memory 4 whose generator matrix is given by this. So this is a feed forward encoder. Now

(Refer Slide Time 02:53)



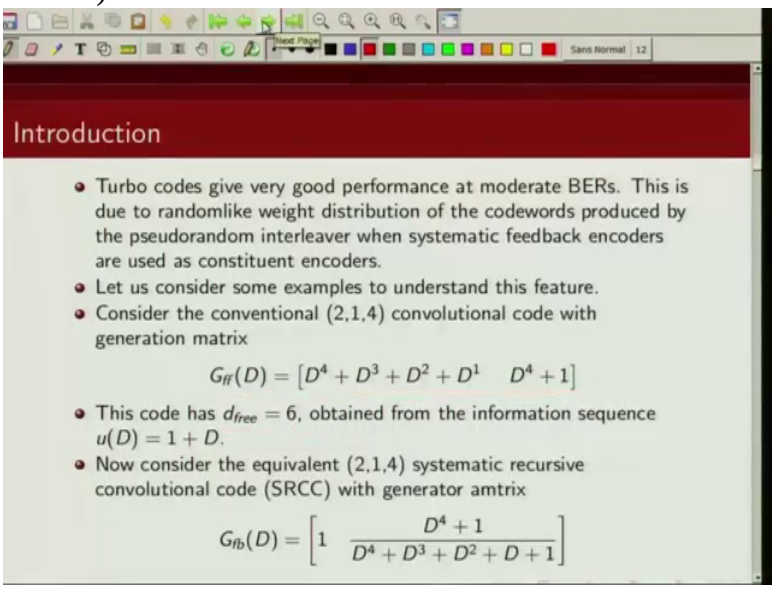
this code has a free distance of 6 which is obtained when we feed information of the form u d equal to 1 plus d. So this is of the form, input sequence is 11 and all zeroes. Now we already know how to find

(Refer Slide Time 03:12)



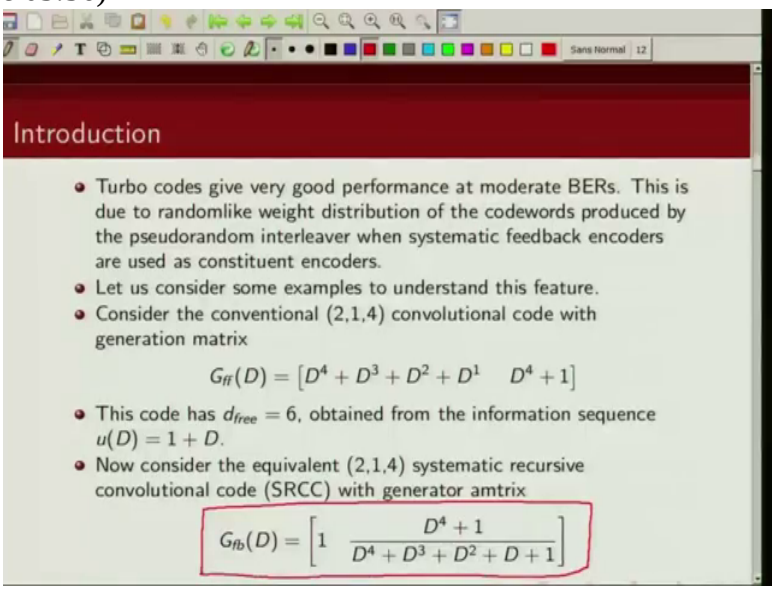
a d free of a convolutional code. Given a generator matrix you first have to find out its state diagram and then you draw modified state diagram and from there you can calculate weight enumerating function, input output weight enumerating function, so conditional weight enumerating function, so all those things we have already discussed in our earlier lecture. So you know how, given a generator matrix how to find out weight distribution of a convolutional code.

(Refer Slide Time 03:49)



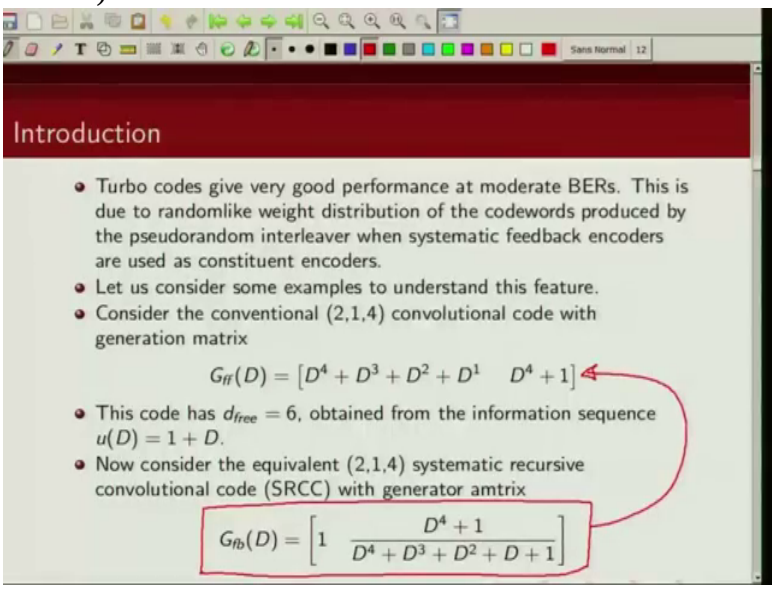
Now let us consider an equivalent convolutional encoder

(Refer Slide Time 03:56)



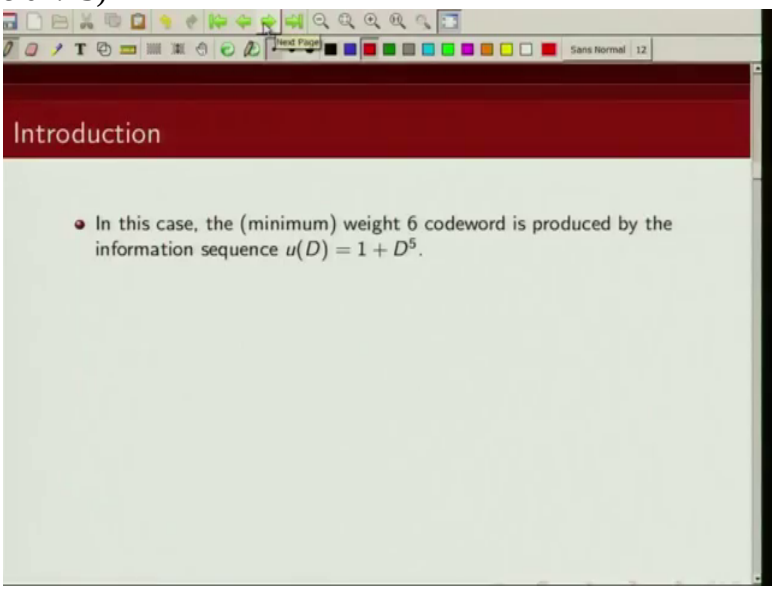
whose convolutional matrix is given by this. Now note this is equivalent to this feed forward

(Refer Slide Time 04:04)



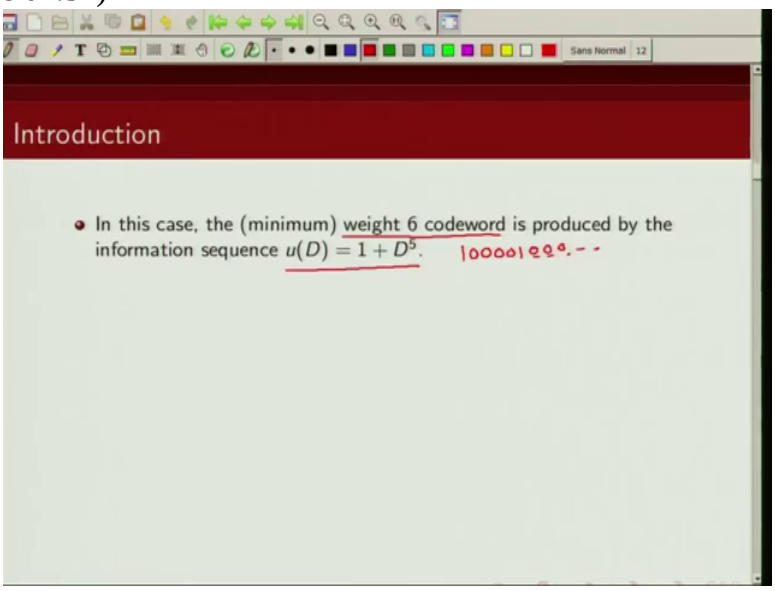
generator matrix, so this is a feedback encoder, and this is a systematic encoder. Now

(Refer Slide Time 04:13)

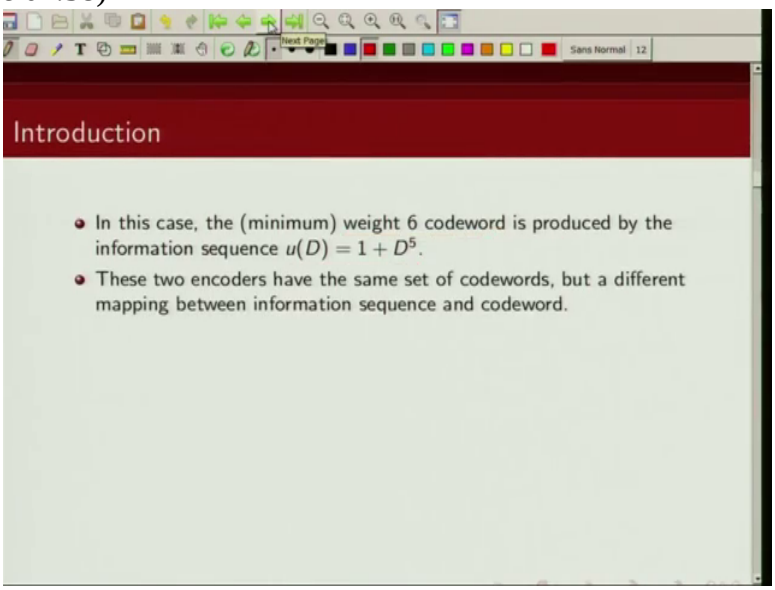


in this case of a systematic feedback encoder, the minimum weight codeword is again 6 and which is produced by input of the form 1 plus d 5, so this is 1 0 0 0 0 1, so this is 1 d d square, d cube, d four d five. So this is produced by sequence of this form.

(Refer Slide Time 04:37)

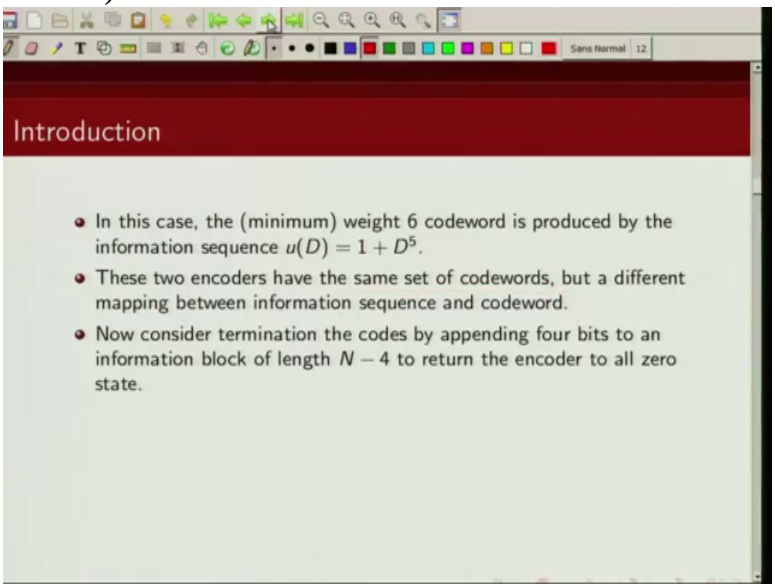


(Refer Slide Time 04:38)



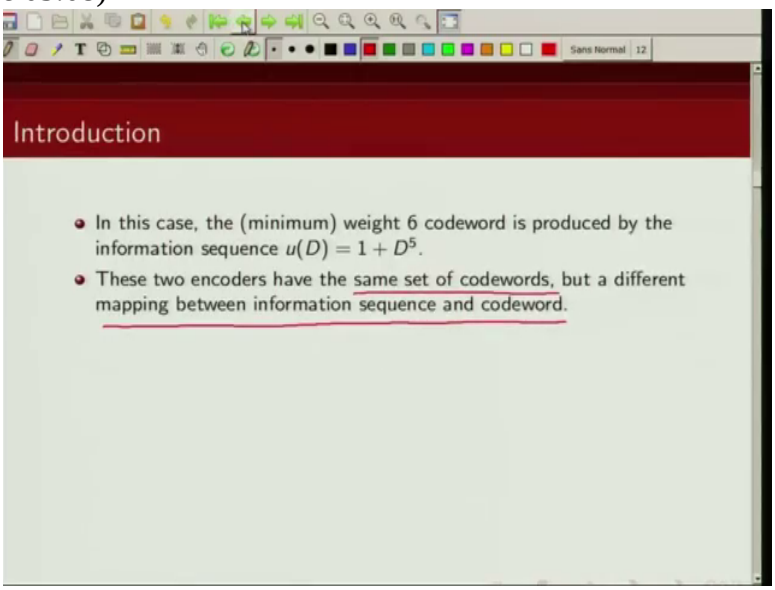
Now since these two generator matrix are equivalent, they will produce the same set of codewords. The only difference is the mapping between the information sequence and codeword will be different.

(Refer Slide Time 04:54)



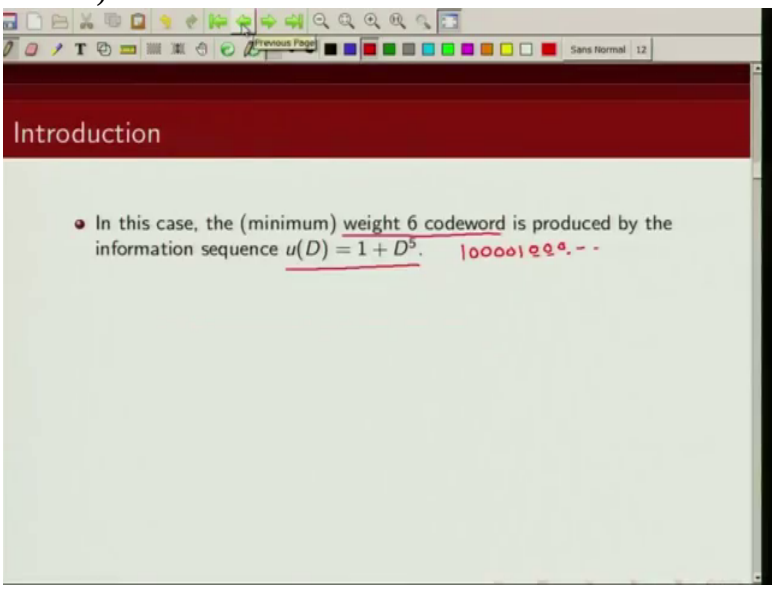
Now let us consider a terminated convolutional code. So let's say our overall information blocks we want is n. So in this example, since the memory

(Refer Slide Time 05:08)



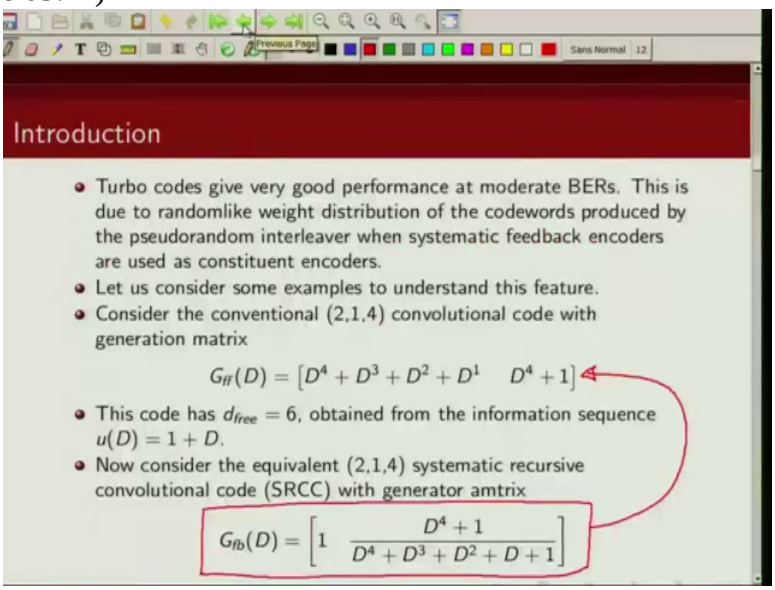
here is 4,

(Refer Slide Time 05:10)



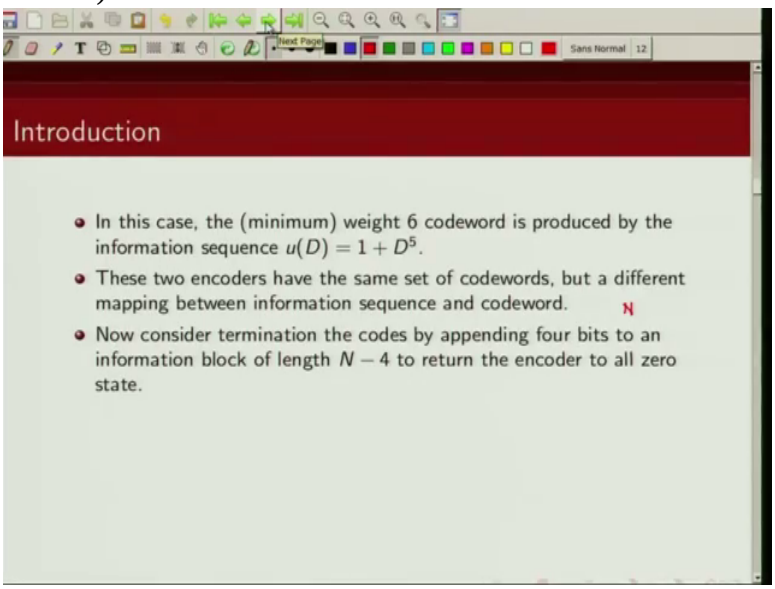
you can see

(Refer Slide Time 05:11)



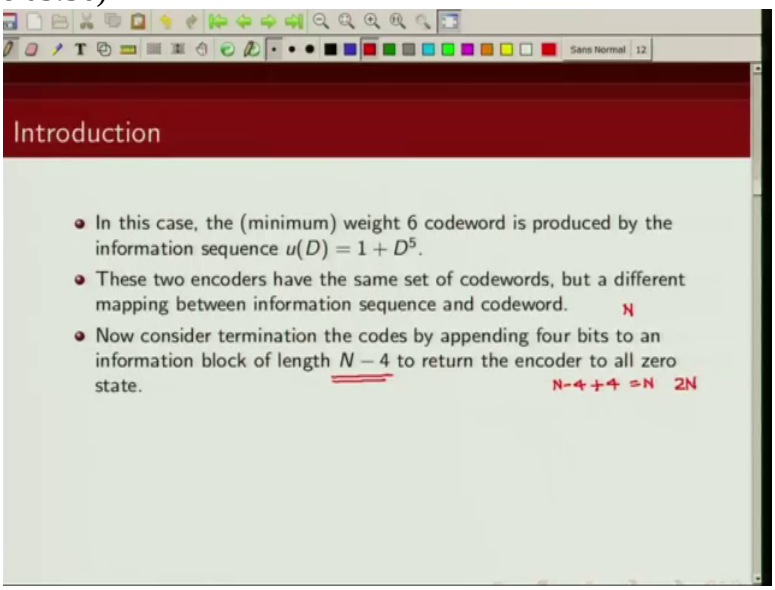
this is memory is 4, Ok. This is rate one half memory 4 code. So if we want to terminate this encoder we would require 4

(Refer Slide Time 05:25)



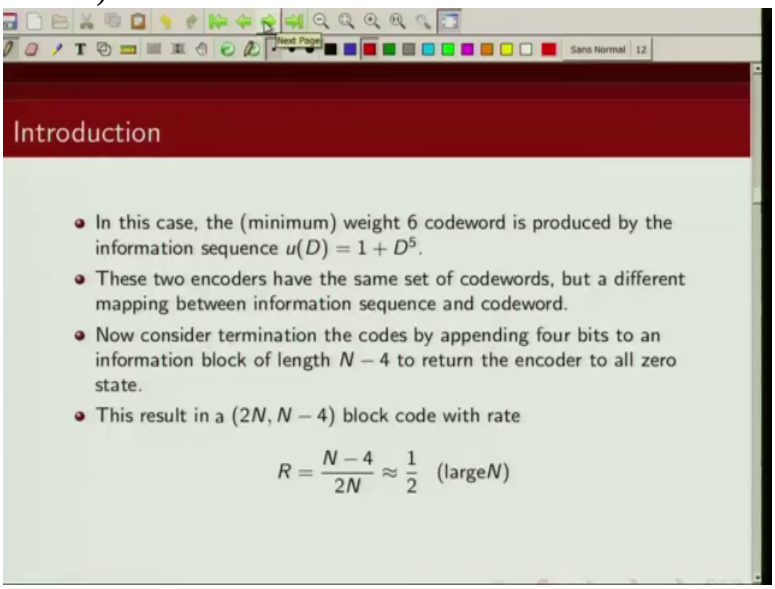
termination bits. So let us consider information block of length n minus 4 and to this we will add 4 termination bits. So overall codeword will be, because it is a rate half code, overall length of the codeword would be 2 times n. So this is information sequence n minus 4 and plus we add 4 tail bits, that is n and since the rate, of course the number of coded bits that we will get is

(Refer Slide Time 05:56)



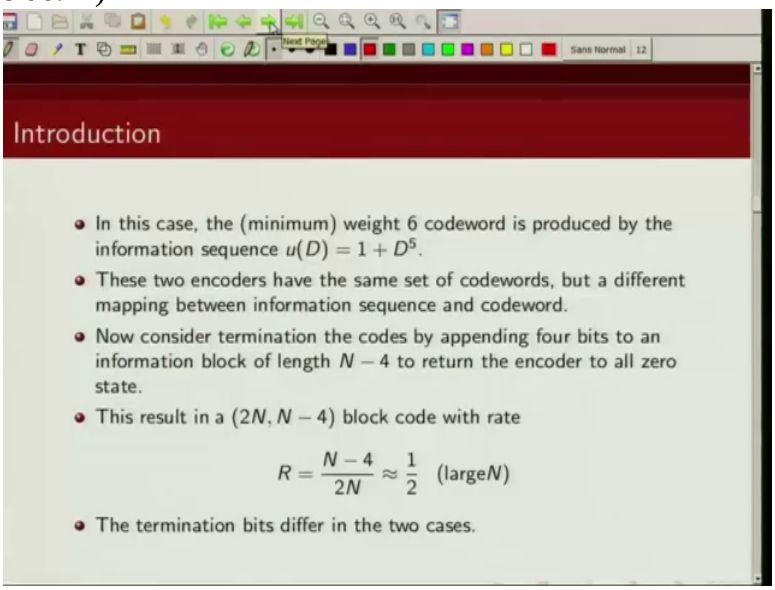
2 n.

(Refer Slide Time 05:58)



So this will result in a 2 n m minus 4 block code. So this is roughly a rate half code if n is very large.

(Refer Slide Time 06:11)

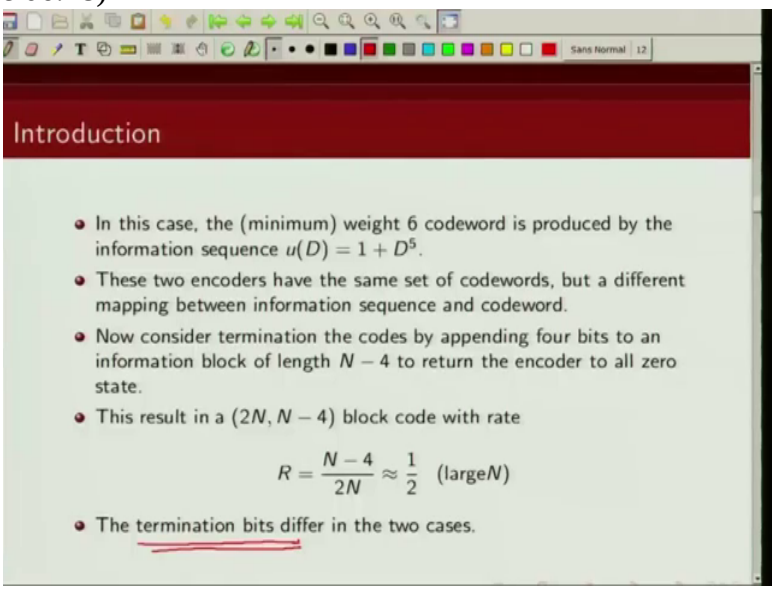


Now please note the termination bit depends on the type of the encoder that we are using. So the termination

(Refer Slide Time 06:20)

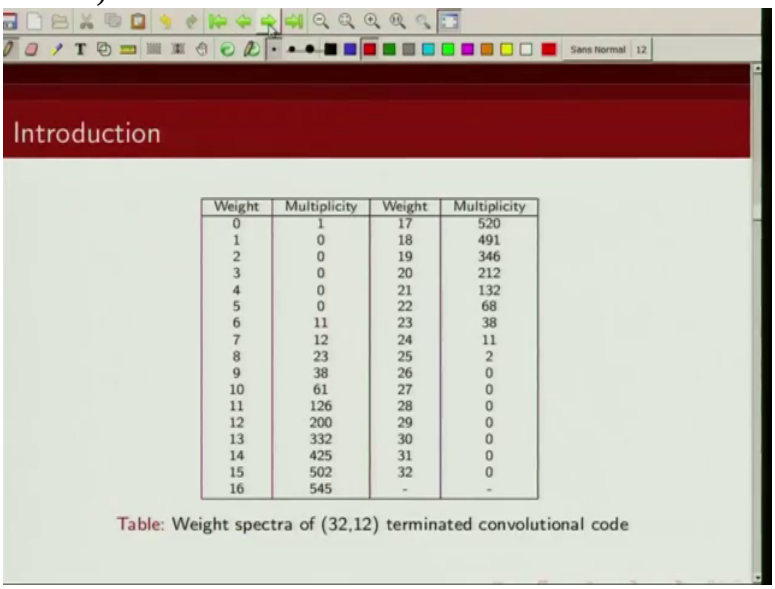
bit that we will use for the feedback encoders are different from the termination bits that we will be using for a

(Refer Slide Time 06:28)



feed forward encoder.

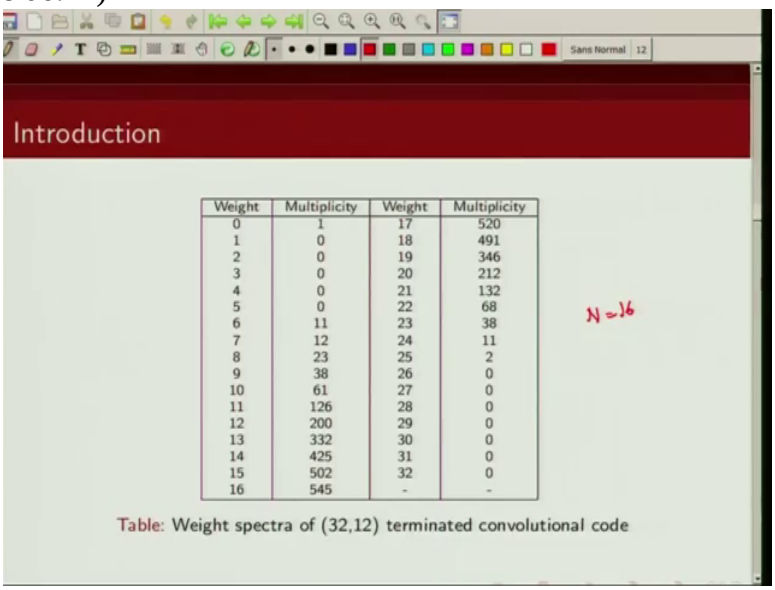
(Refer Slide Time 06:30)



Weight	Multiplicity	Weight	Multiplicity
0	1	17	520
1	0	18	491
2	0	19	346
3	0	20	212
4	0	21	132
5	0	22	68
6	11	23	38
7	12	24	11
8	23	25	2
9	38	26	0
10	61	27	0
11	126	28	0
12	200	29	0
13	332	30	0
14	425	31	0
15	502	32	0
16	545	-	-

So in this I have drawn, so , consider m equal to 16. So this is

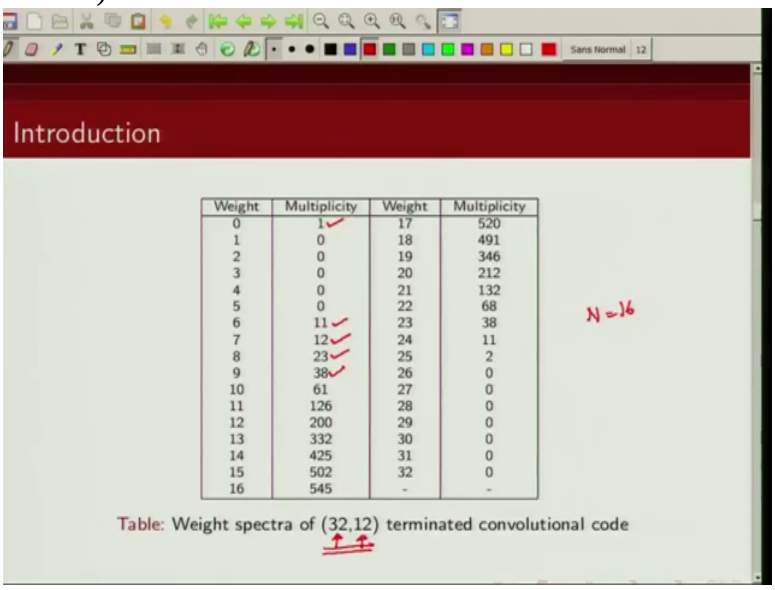
(Refer Slide Time 06:41)



a 12 32, so n is 32 and k is 12 here. Remember we are using a generator matrix of memory order 4 so we and you are terminating it, so 4 (()) 4 tail bits. So this is a 32 12 terminated convolution code. And look at the weight spectrum of that. So there is a codeword all zero codeword. Then we have 11 codewords of weight 6, 12 codewords of weight 7, 23 codewords of weight 8, 38 codewords of weight 9 and so on.

Now let us compare

(Refer Slide Time 07:35)

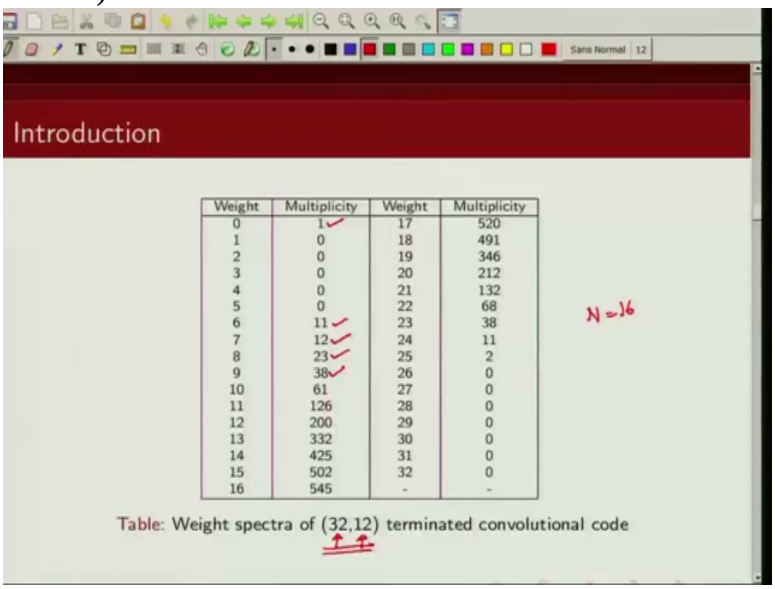


the weight distribution

(Refer Slide Time 07:37)

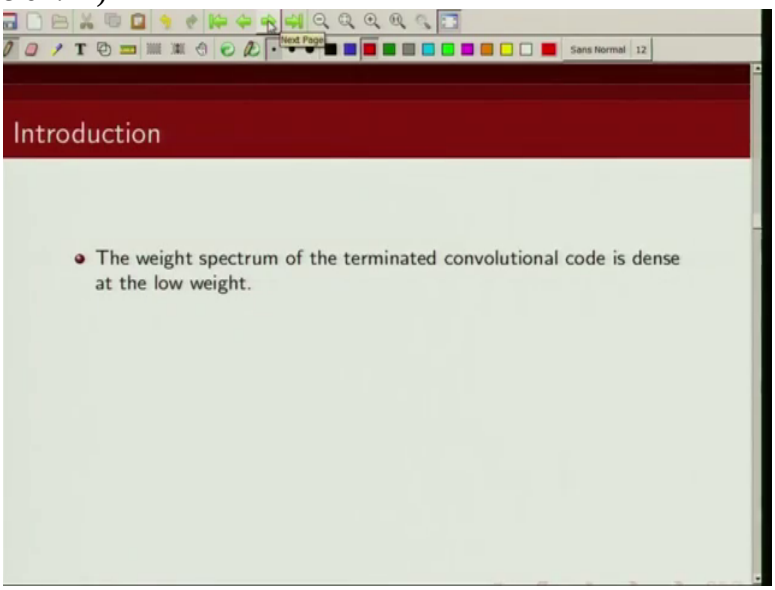
of a terminated convolutional code with a corresponding parallel concatenated code

(Refer Slide Time 07:44)



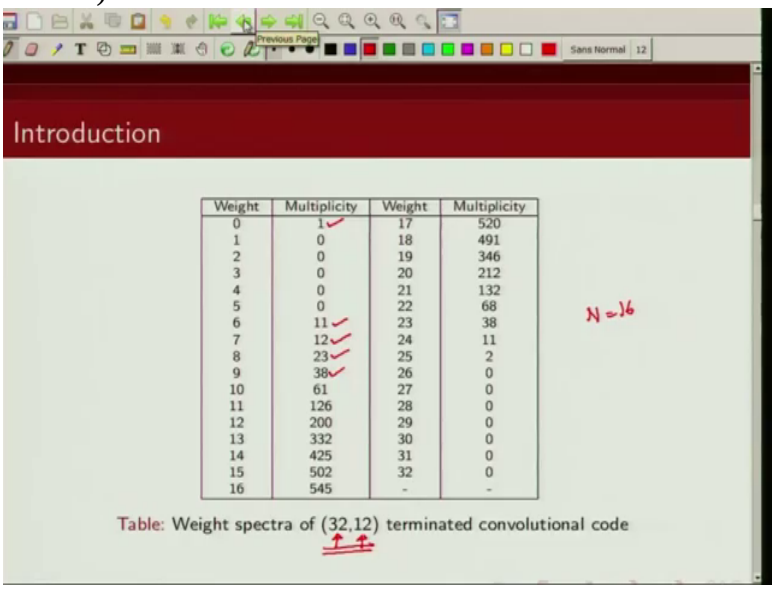
with similar parameters. So

(Refer Slide Time 07:47)



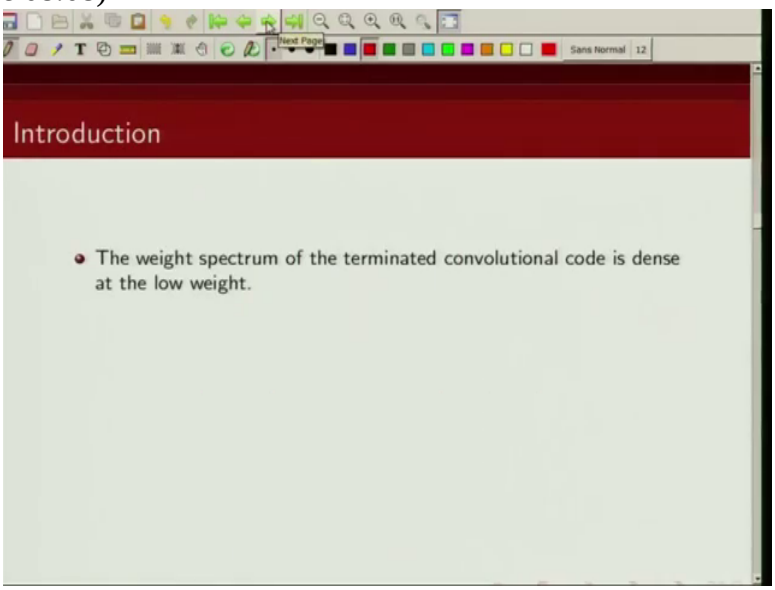
the thing to be noted here is if you look here,

(Refer Slide Time 07:51)



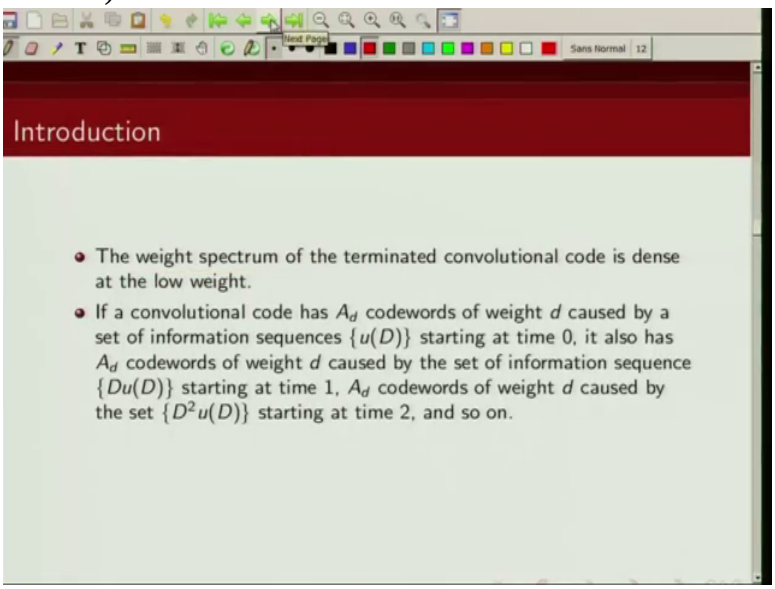
the multiplicity of codewords having small weight is quite high. There are 11 codewords of weight 6, 12 codewords of weight 7, 23 codewords of weight 8, 38 codewords of weight 9. So

(Refer Slide Time 08:08)



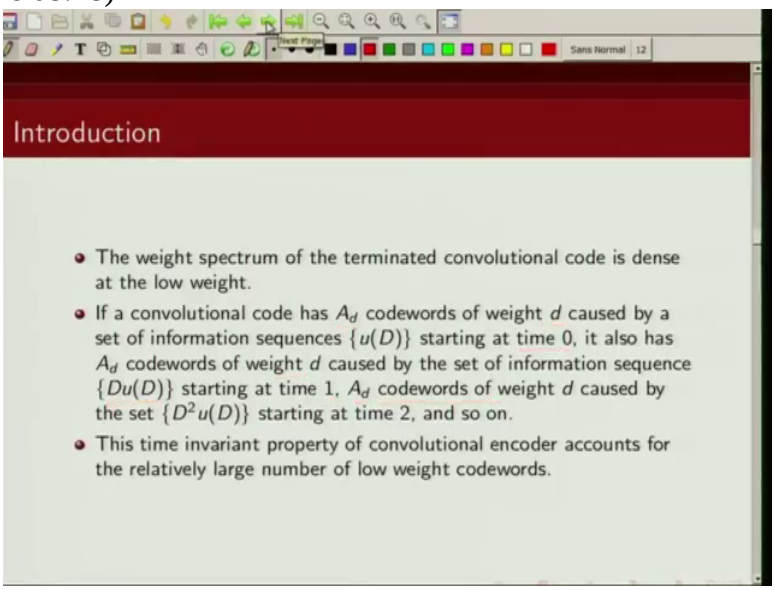
the statement I am making is spectrum of terminated convolutional code is dense at lower weight.

(Refer Slide Time 08:18)



Now if you have convolutional code which has 80 codewords of weight d caused by set of information sequence starting at time zero, then it also has these 80 codewords of weight d caused by set of information sequence of form D u d at time 1 or 80 codewords of weight d caused by the information sequence of the form D square u d stating at time 1 and so on.

(Refer Slide Time 08:48)

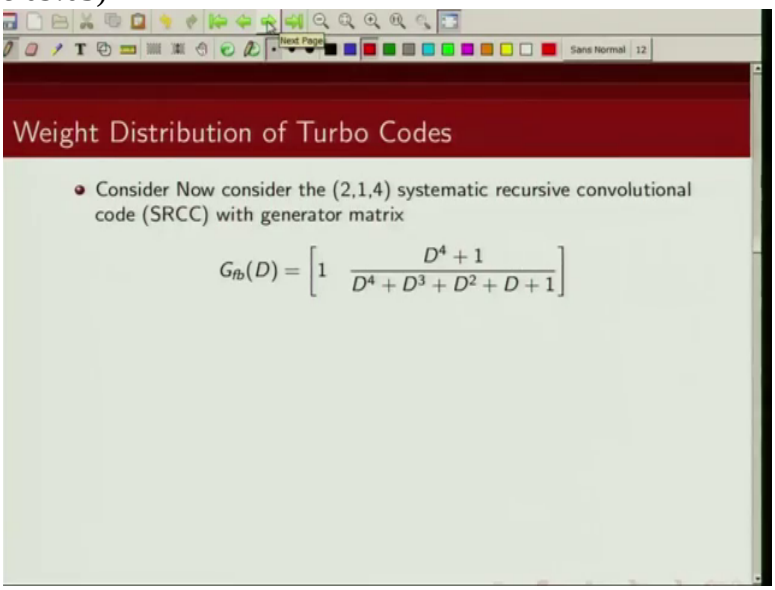


So this time invariant property of convolutional encoder accounts for their relatively

(Refer Slide Time 08:56)

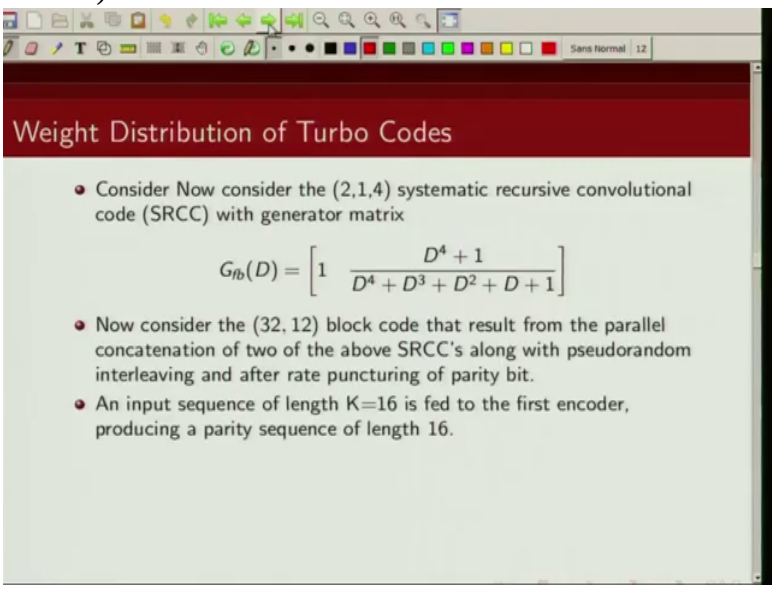
large number of low weight codewords in the weight spectrum of the convolutional code.

(Refer Slide Time 09:03)



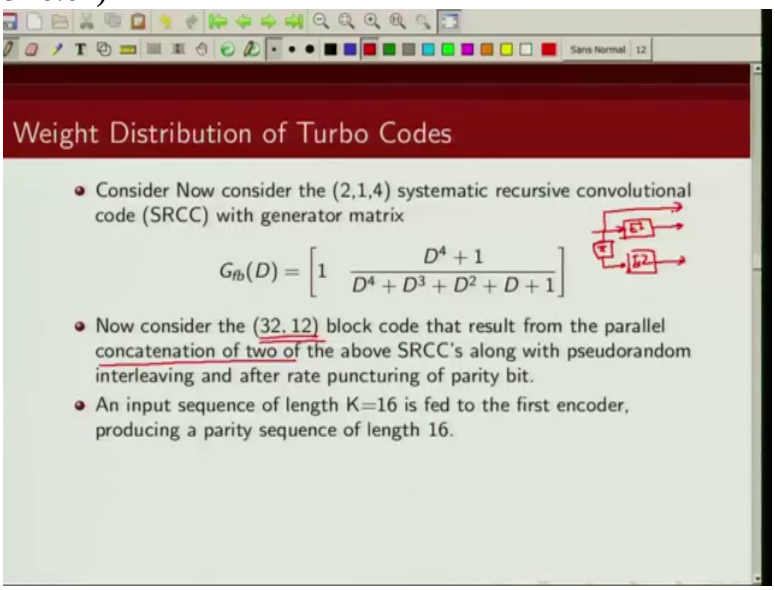
Now, let us now consider the equivalent systematic feedback encoder, this is equivalent to the feed forward encoder generator matrix that we saw earlier. Now

(Refer Slide Time 09:20)



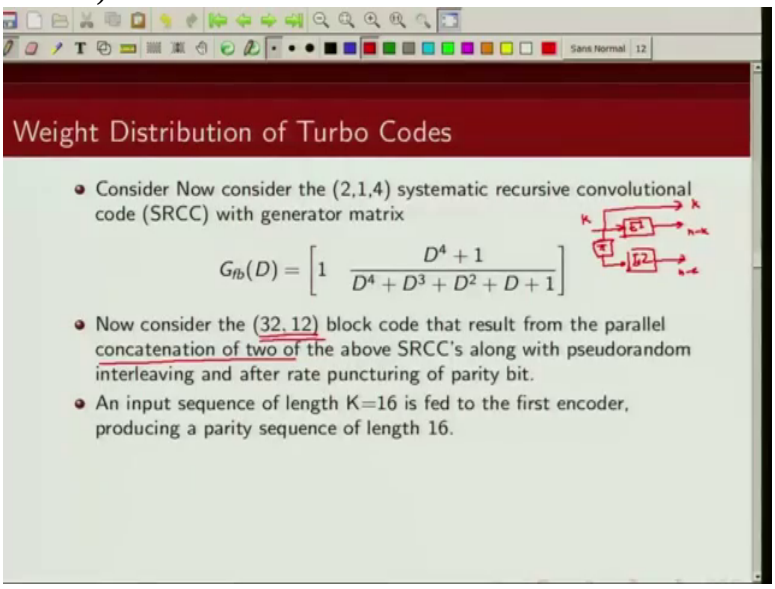
we want to construct a parallel concatenation of 2 encoders which has the same parameters as the terminated convolutional code that we had. So how do we get it? So if you recall, turbo code is of the form like this. So this is encoder 1, this is encoder 2, and then this is your interleaver and then your parity stream coming encoder 1,parity stream coming encoder 2 and then you have these systematic bits

(Refer Slide Time 10:01)



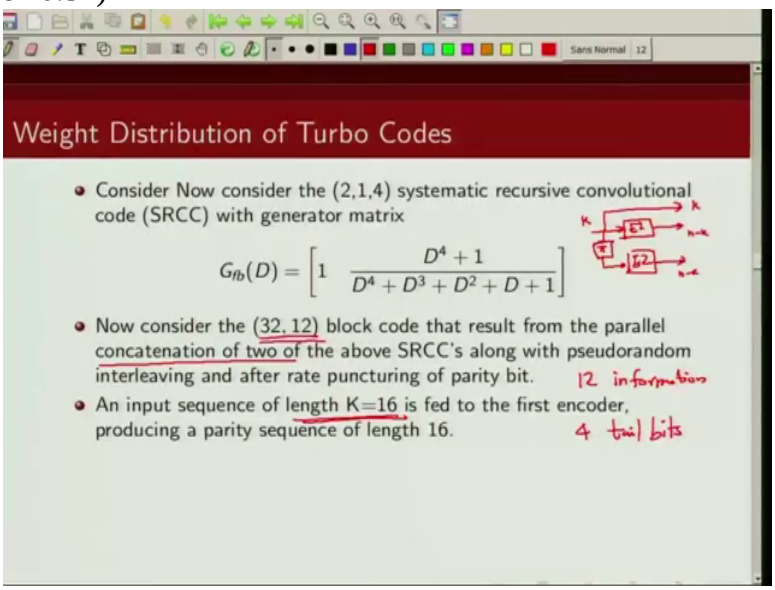
right. So if you have k information bits coming in, here you are getting k information bits, here you are getting k information bits, here you are getting n minus k parity bits, here you are getting n minus k parity bits.

(Refer Slide Time 10:17)



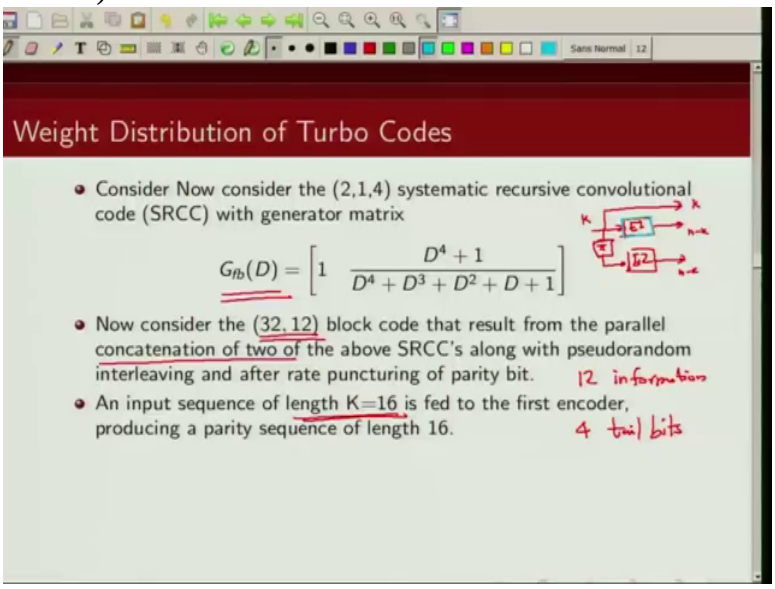
Now let us consider a input of length 16 where 12 of them will be information bits Ok and 4 of them will be tail bits which we use

(Refer Slide Time 10:34)



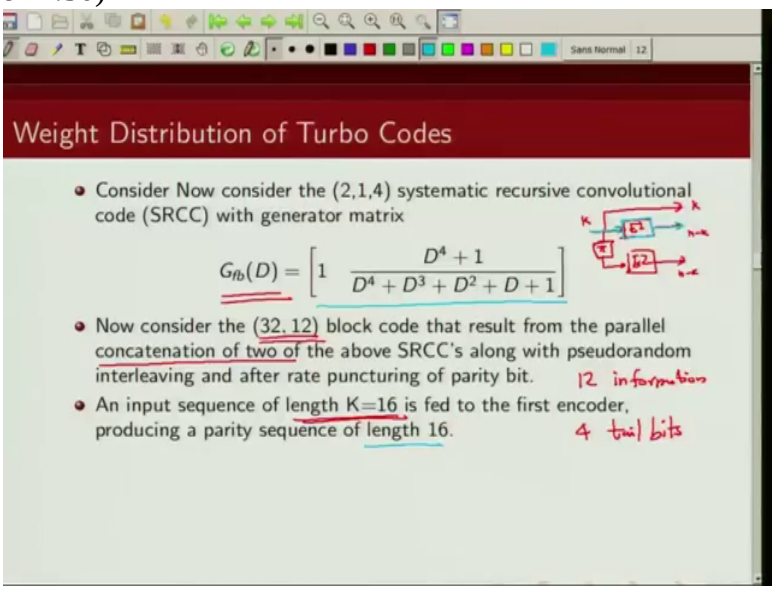
to terminate this convolutional encoder. So we are considering input of size 16, 12 of them is information sequence and 4 of these are tail bits which are used to terminate this convolutional encoder. So this we are feeding our encoder 1, which is this encoder e 1,

(Refer Slide Time 10:55)



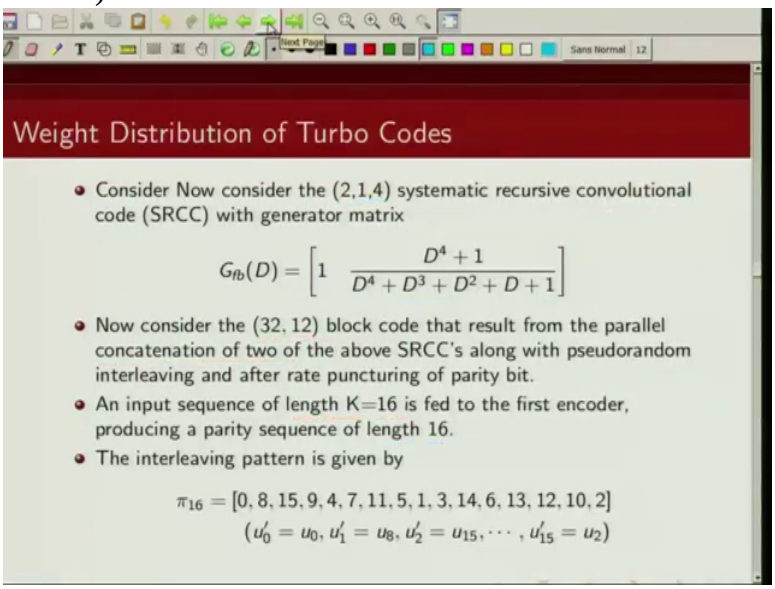
Ok. Now input sequence, and we are feeding an input sequence and remember this is our rate one half code, this is rate one half code. If we feed the sequence of input bit 16, the parity bit that will come out will have parity bit of length 16. Because here k is half of n. So if we feed in information sequence of length 16, here we are getting

(Refer Slide Time 11:30)



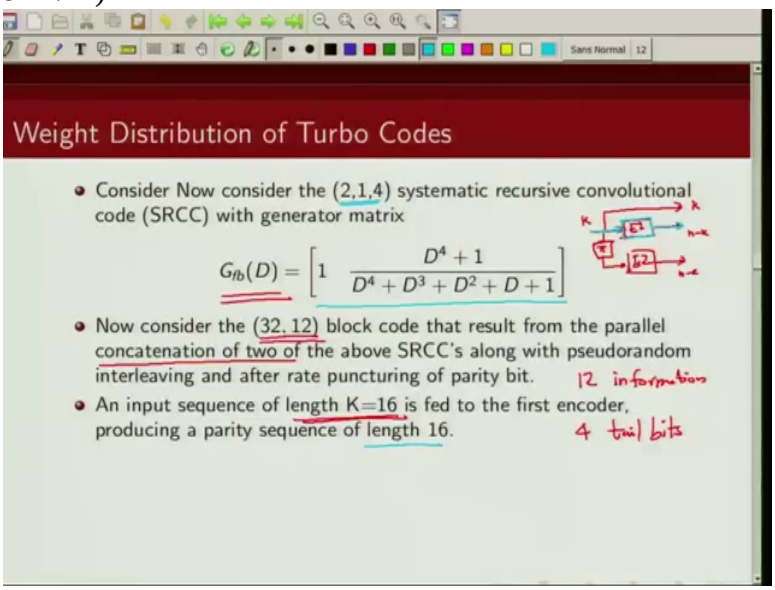
also a parity sequence of length 16, because it is a rate half code. Next what's happening

(Refer Slide Time 11:40)



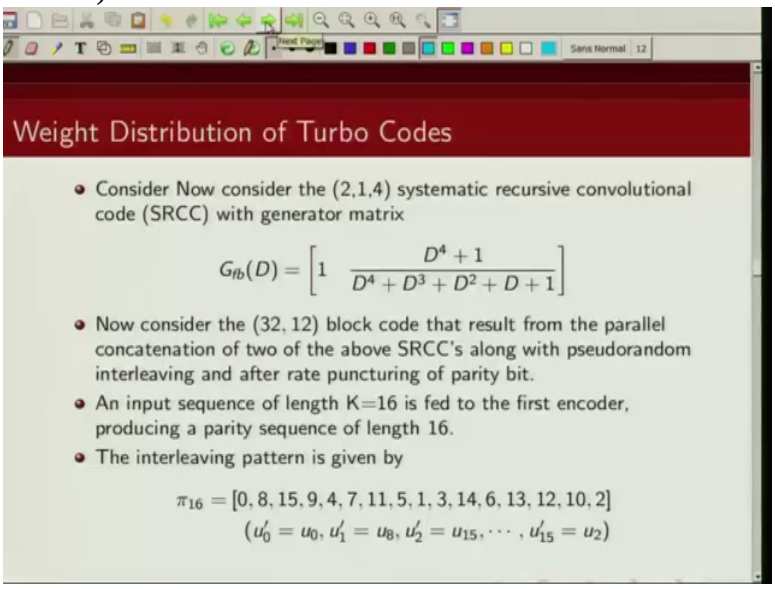
is, so this information sequence is interleaved

(Refer Slide Time 11:44)



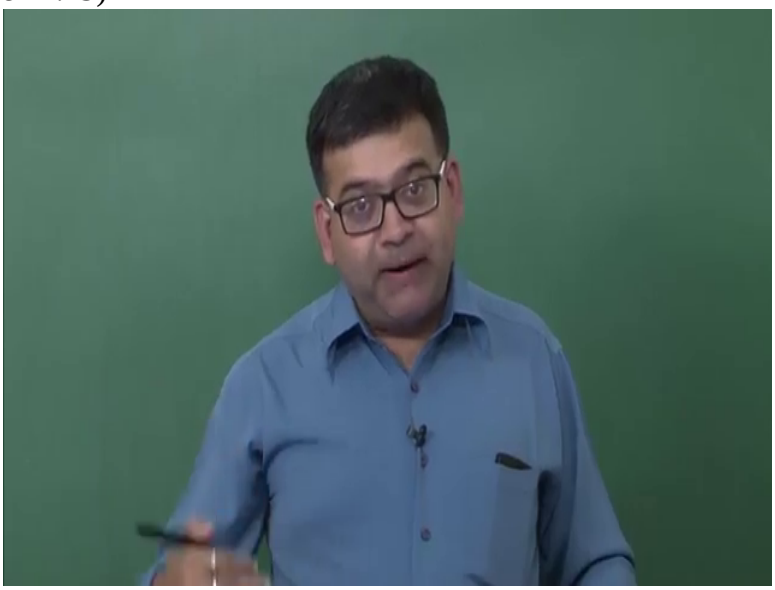
and fed to second encoder. So what the second encoder does, it is an interleaved version. Interleaved is nothing but it is a permuted version of this information sequence. So this is a 16 sequence that comes here and this also generates a parity sequence of length 16.

(Refer Slide Time 12:07)



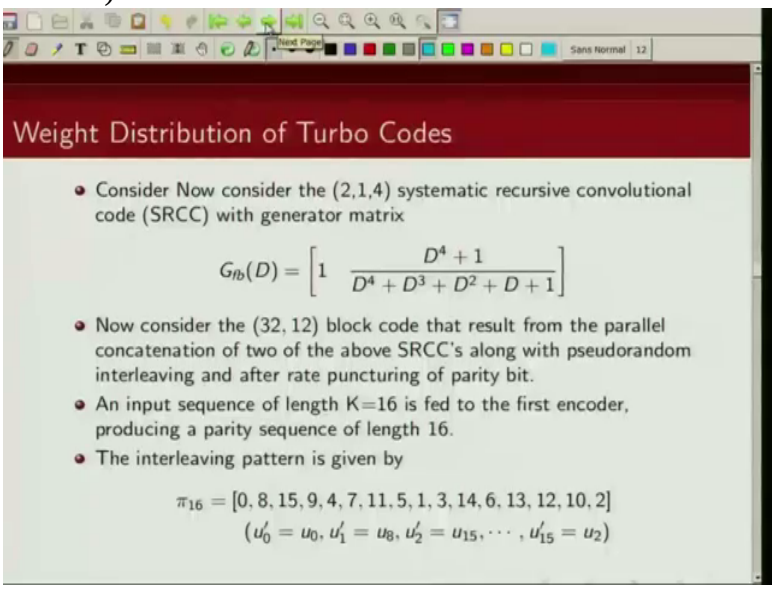
So we need to specify, remember we are trying to get to the distance profile of a parallel concatenated code and the distance profile of parallel concatenated code

(Refer Slide Time 12:19)



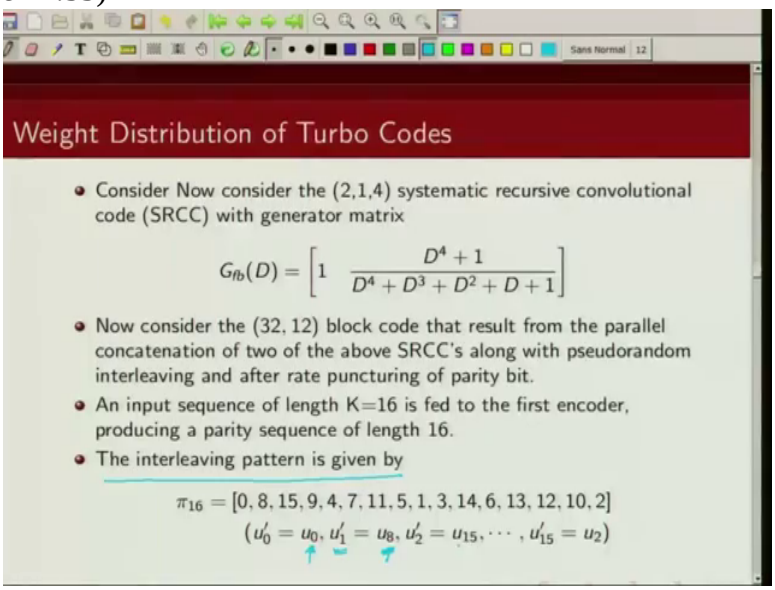
depends on what type of interleaver we are using. In this particular example

(Refer Slide Time 12:24)



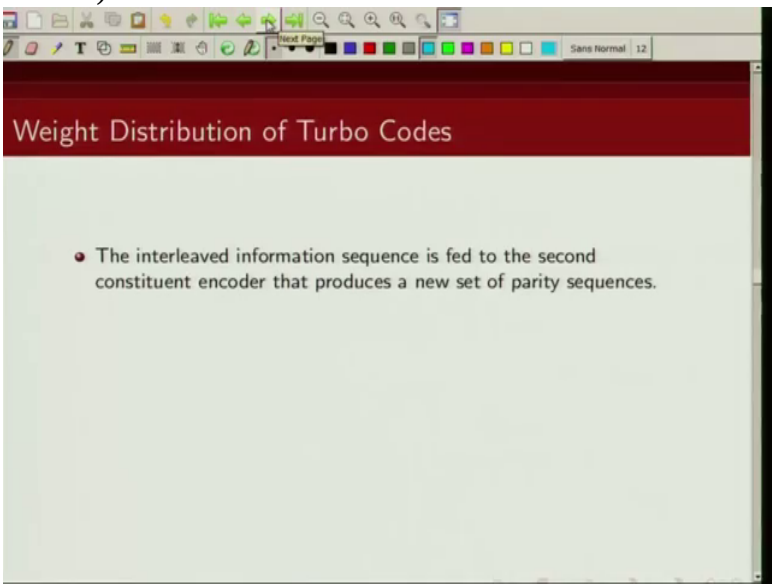
we are specifying one such interleaver pattern. So you can see here I have specified here so 0 layer means the bit stays here in the interleaved version the first bit belongs, the second bit belongs to the input u 8 and like that. So u 0s are the original input sequence

(Refer Slide Time 12:53)



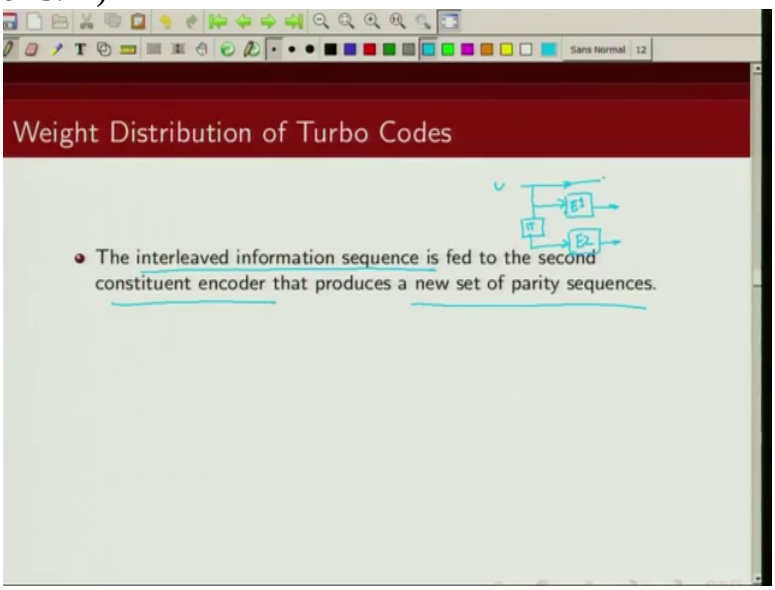
and u i hats are the interleaved information sequence. So this is the interleaving pattern I am using for this particular example to derive the weight distribution of this parallel concatenated code.

(Refer Slide Time 13:08)



Now this interleaved information sequence is fed to the second constituent encoder and which produces a new set of parity sequences. Now as you know the parallel concatenating encoder we drew is actually a example of a rate one third code. This was our parallel concatenated code. Our objective is to first compare the weight spectrum

(Refer Slide Time 13:41)

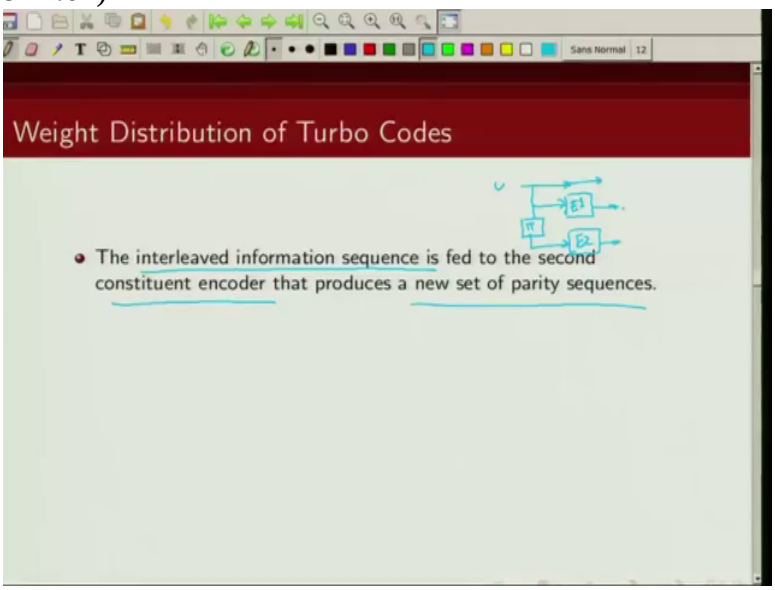


of a convolutional code to that of a turbo code. And the convolutional code that you studies was a rate half code. And this is a rate one third code. So what do we need to do to make it a rate one half? So we are going to puncture

(Refer Slide Time 13:58)

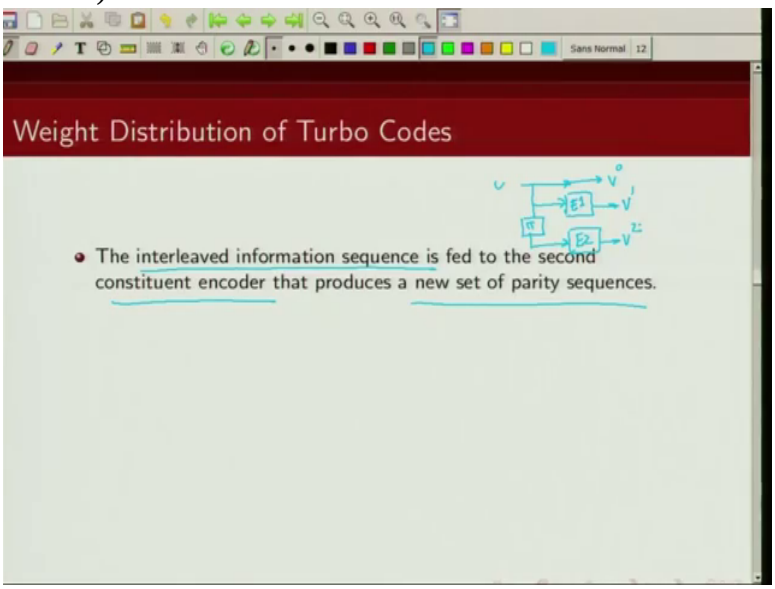
the parity bits. So we are going to send alternatively parity bits from

(Refer Slide Time 14:04)



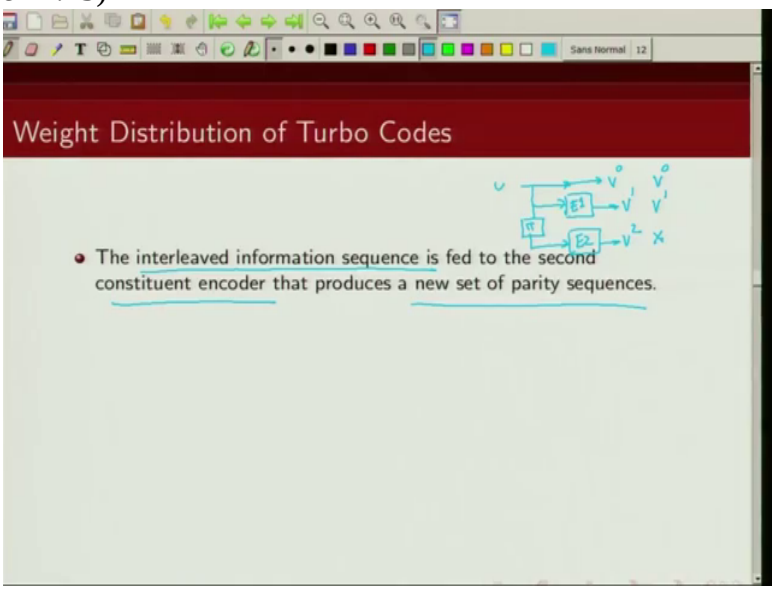
output of the first encoder or output of the second encoder. So for example, let's call this v 0, v 1 and v 2. So what we are

(Refer Slide Time 14:15)



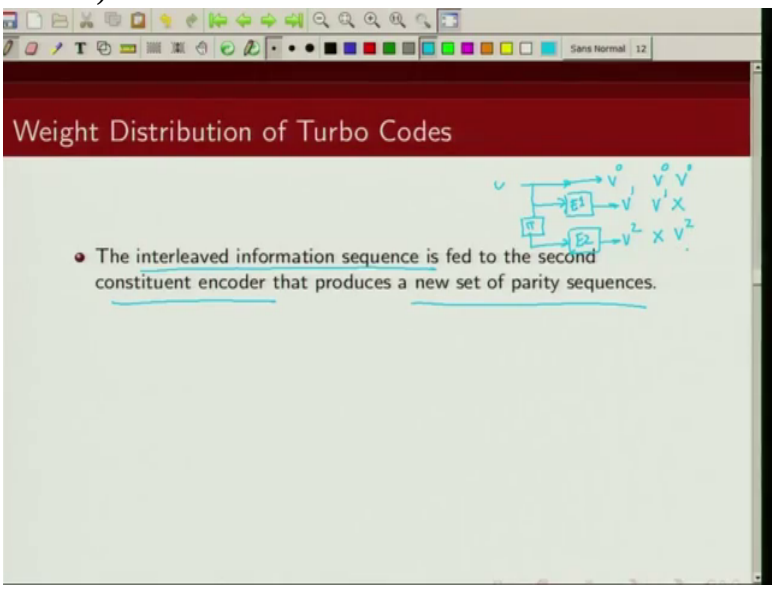
going to do is at a particular time instance we transmit v 0 and v 1 and would not transmit this parity bit

(Refer Slide Time 14:23)

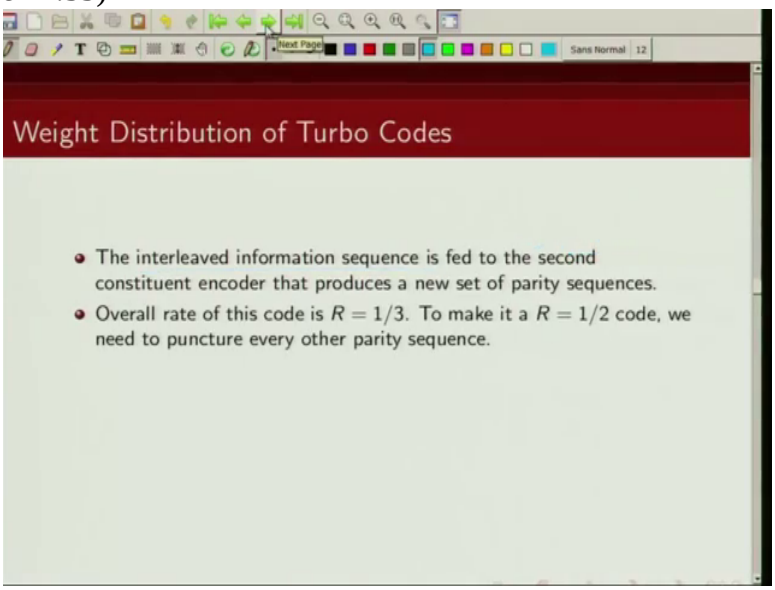


and the second instance will transmit this information sequence, we would not send this parity bit, we will send parity bit from this encoder.

(Refer Slide Time 14:31)

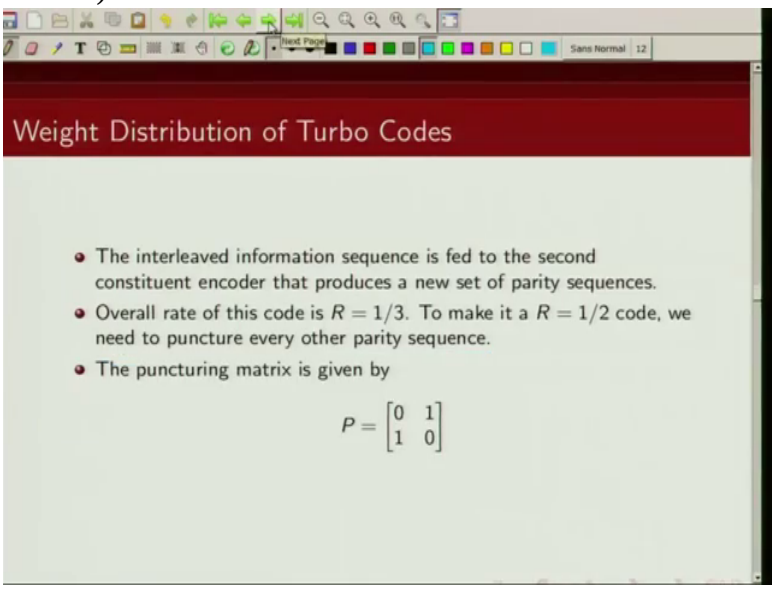


(Refer Slide Time 14:33)



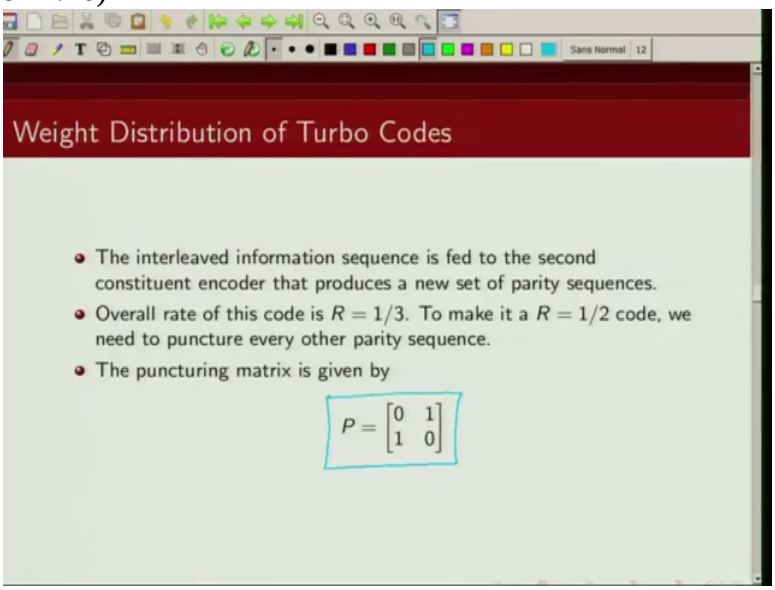
So we need to puncture every other parity bit in each of these streams to make it a rate one half

(Refer Slide Time 14:41)



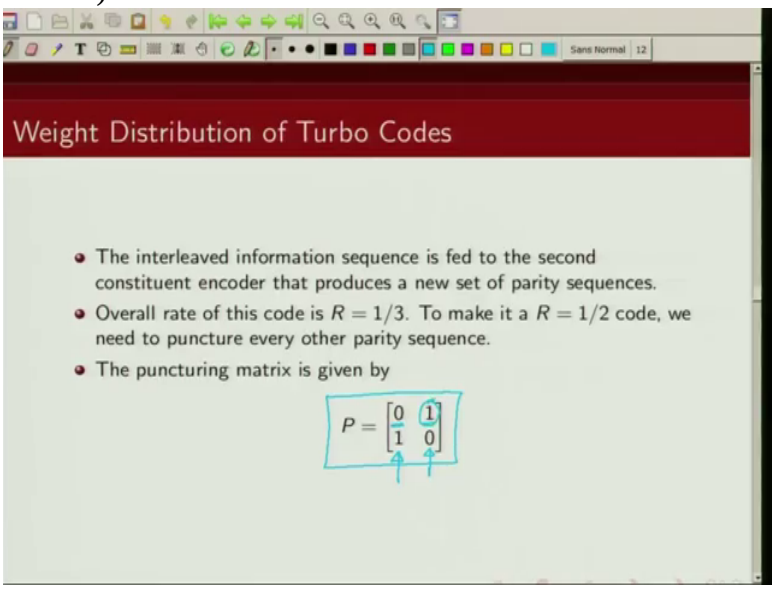
and we denote a puncturing pattern like this. So you can view

(Refer Slide Time 14:46)



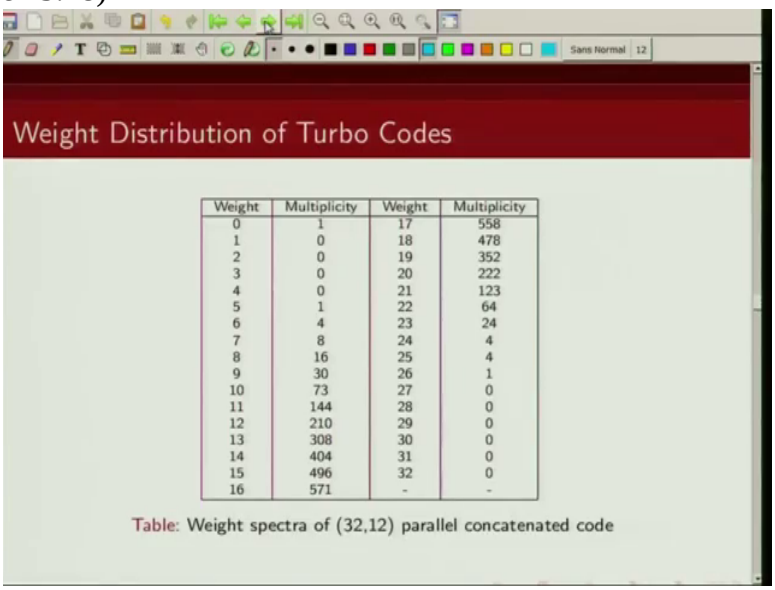
and we denote a puncturing pattern like this. So you can view this, if there is a zero, we are not transmitting that bit. If there is a 1, we are transmitting this bit. So you can look at, view look at, view it like this. At odd time instances I am sending parity bit from the second encoder where as in the even time instances I am sending

(Refer Slide Time 15:03)



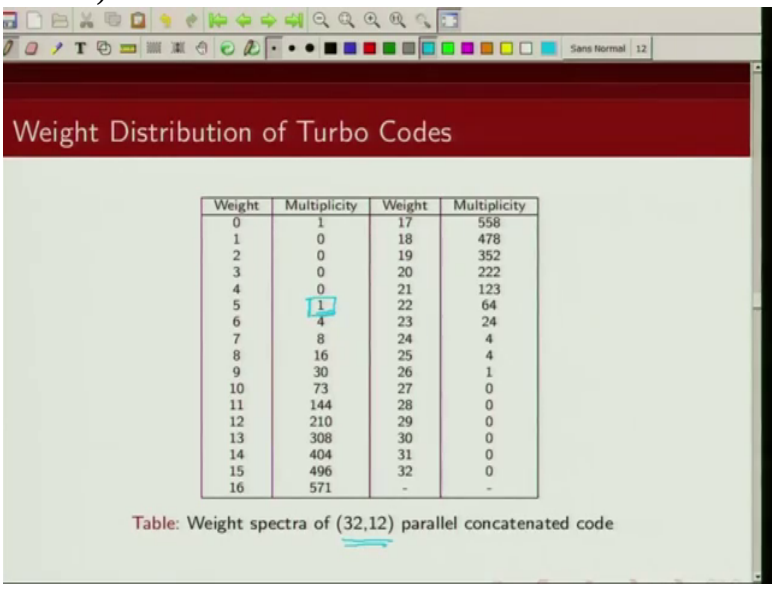
parity bit from the first encoder. So this will make the overall rate of this parallel concatenated code as rate half which is same as the original convolutional code that

(Refer Slide Time 15:18)



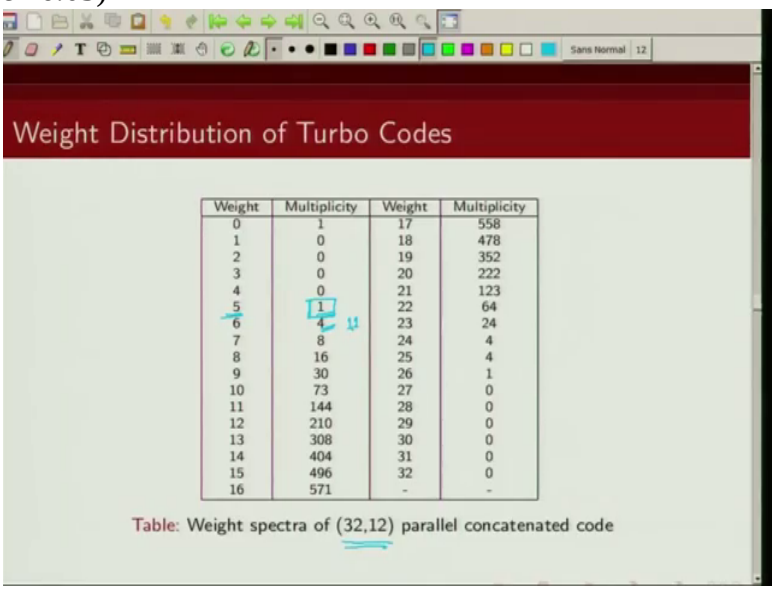
we were talking about. Now look at the way the distribution of 32 12 parallel concatenated code which we just talked about. Now note in case of convolutional code the minimum d free was 6. But here we see there is a codeword

(Refer Slide Time 15:38)



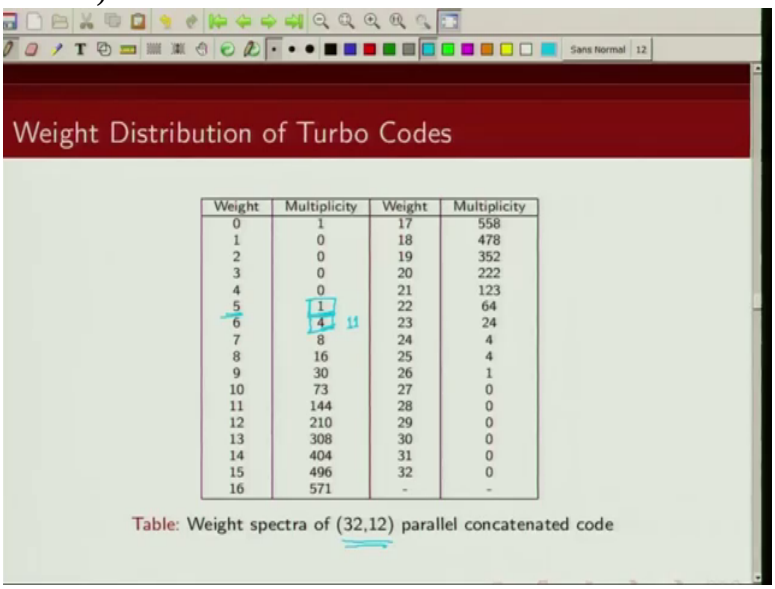
of weight 5. However the point to be noted is this. Now let's look at the multiplicity of codewords of low weight. In case of the example we considered earlier of the terminated convolutional code, in that case we had 11 codewords of

(Refer Slide Time 16:03)



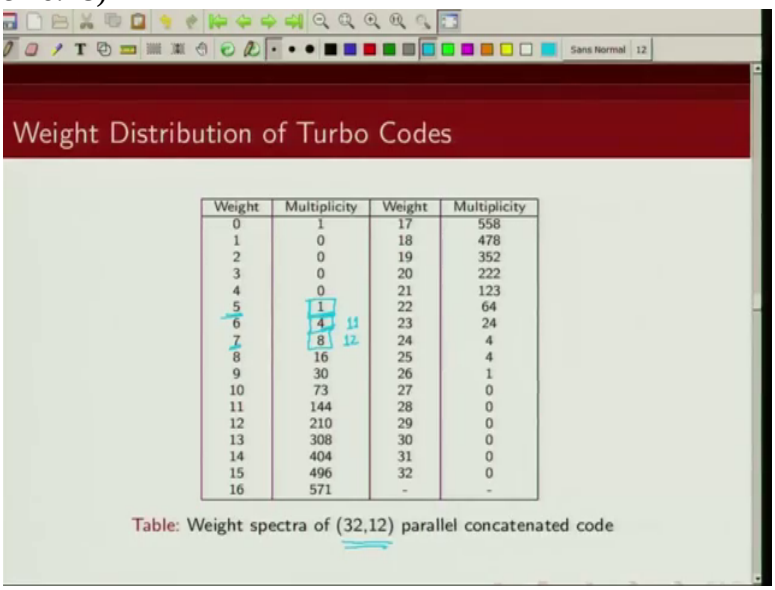
weight 6 whereas here we are having only 4.

(Refer Slide Time 16:07)



We were having 12 codewords of weight 7, here we are having 8.

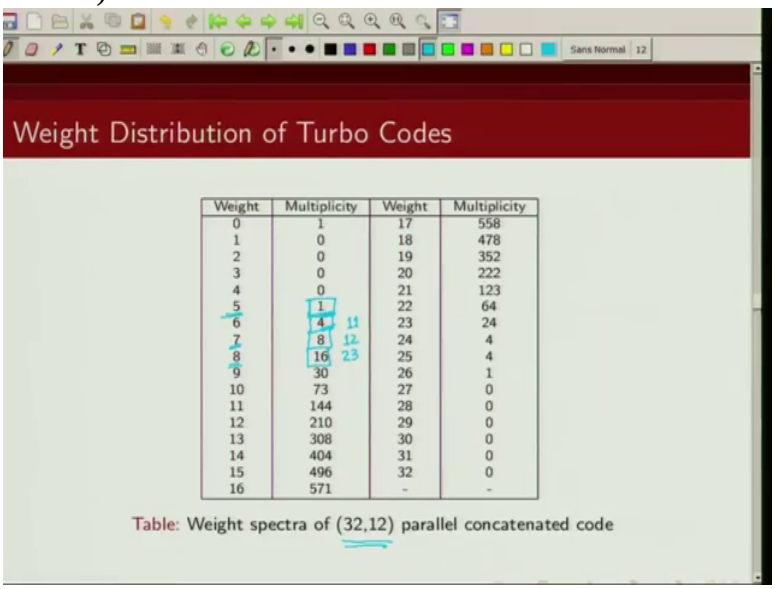
(Refer Slide Time 16:18)



Weight	Multiplicity	Weight	Multiplicity
0	1	17	558
1	0	18	478
2	0	19	352
3	0	20	222
4	0	21	123
5	1	22	64
6	4	23	24
7	8	24	4
8	16	25	4
9	30	26	1
10	73	27	0
11	144	28	0
12	210	29	0
13	308	30	0
14	404	31	0
15	496	32	0
16	571	-	-

We were having 23 codewords of weight 8, here we are having 16.

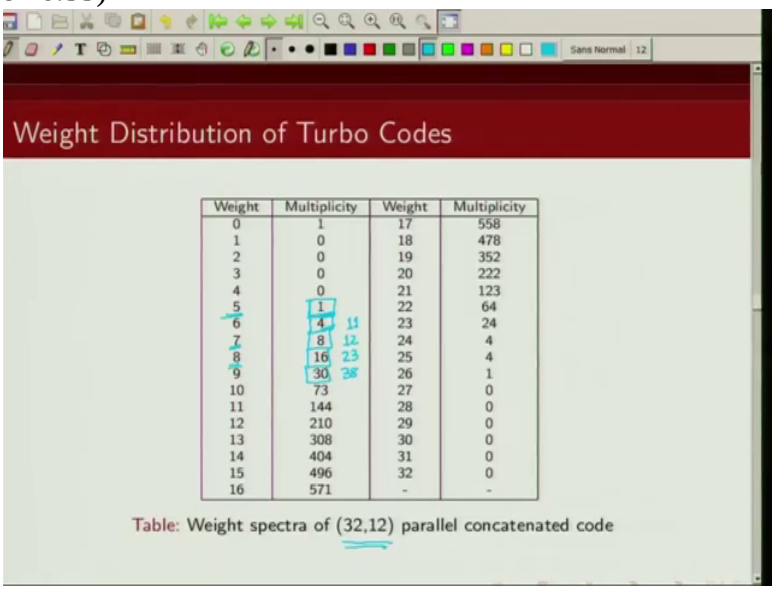
(Refer Slide Time 16:27)



Weight	Multiplicity	Weight	Multiplicity
0	1	17	558
1	0	18	478
2	0	19	352
3	0	20	222
4	0	21	123
5	1	22	64
6	4	23	24
7	8	24	4
8	16	25	4
9	30	26	1
10	73	27	0
11	144	28	0
12	210	29	0
13	308	30	0
14	404	31	0
15	496	32	0
16	571	-	-

We were having 38 codewords of weight 9, here we are having 30.

(Refer Slide Time 16:33)

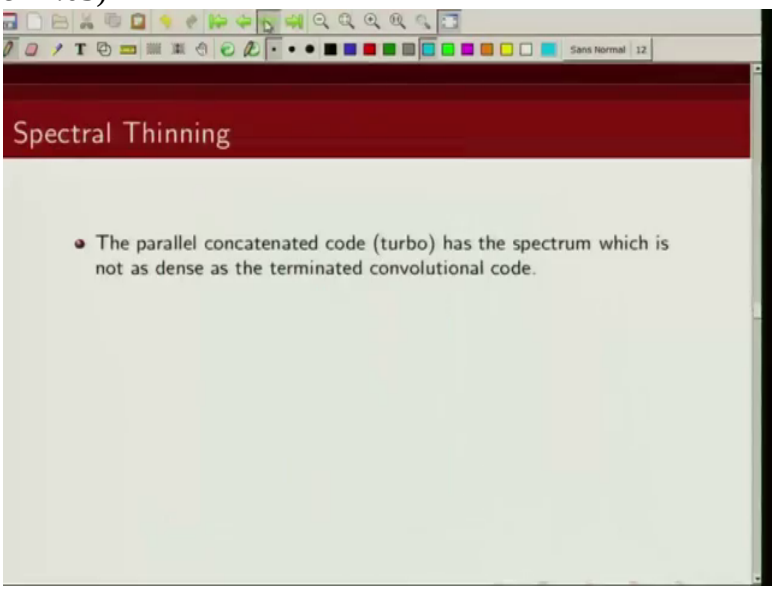


So note that in case of this concatenated code we are noticing that

(Refer Slide Time 16:41)

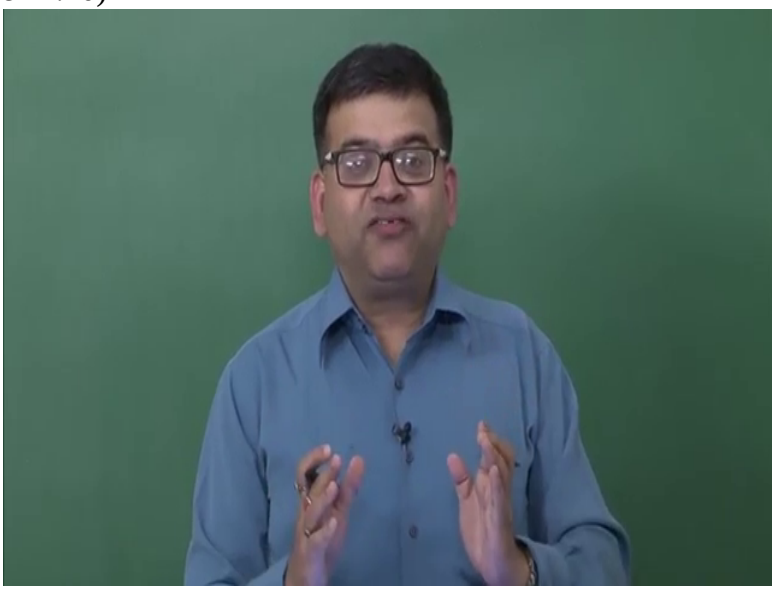
as a result of this interleaver the multiplicity of low weight codeword is reduced. Now we considered a very simple example with interleaver size of just 16. If we increase the interleaver size, this effect is even more pronounced. And this is

(Refer Slide Time 17:03)



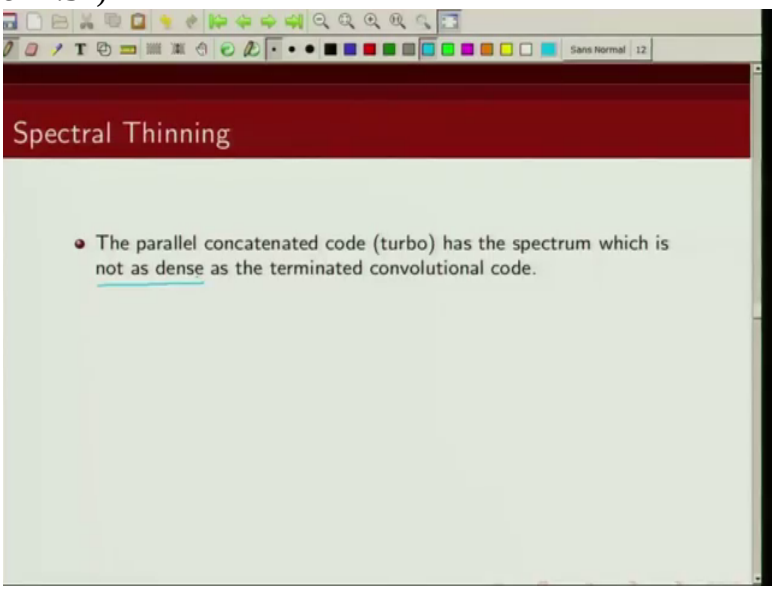
what we call spectral thinning. So parallel concatenate code has a spectrum which is not as dense as convolutional code. In fact we note, even with a very small example where we consider an interleaver size

(Refer Slide Time 17:20)



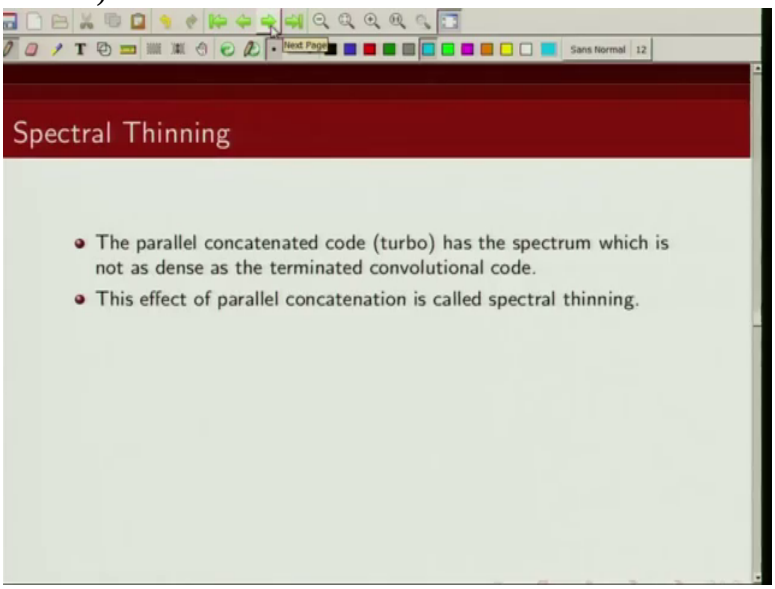
of 16, we saw that low weight codewords, the number of low weight codewords, the multiplicity of low weight codewords has reduced in case of

(Refer Slide Time 17:32)



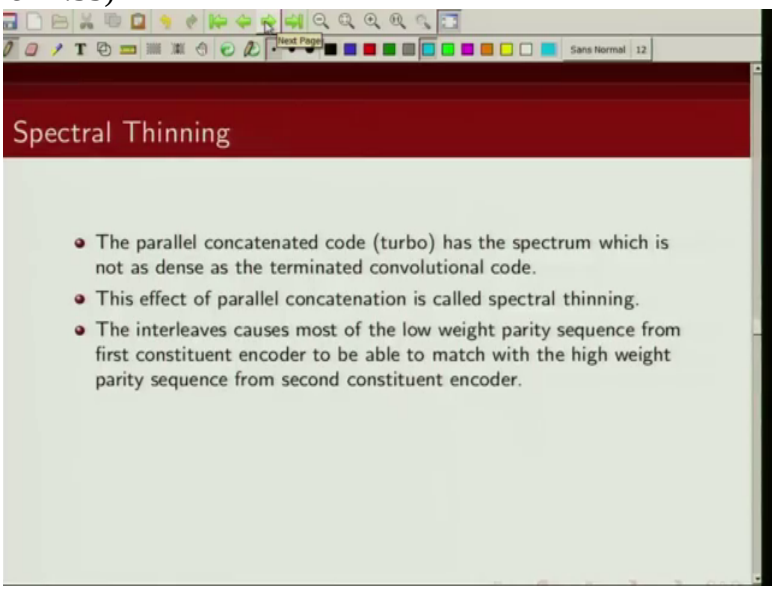
turbo codes.

(Refer Slide Time 17:34)



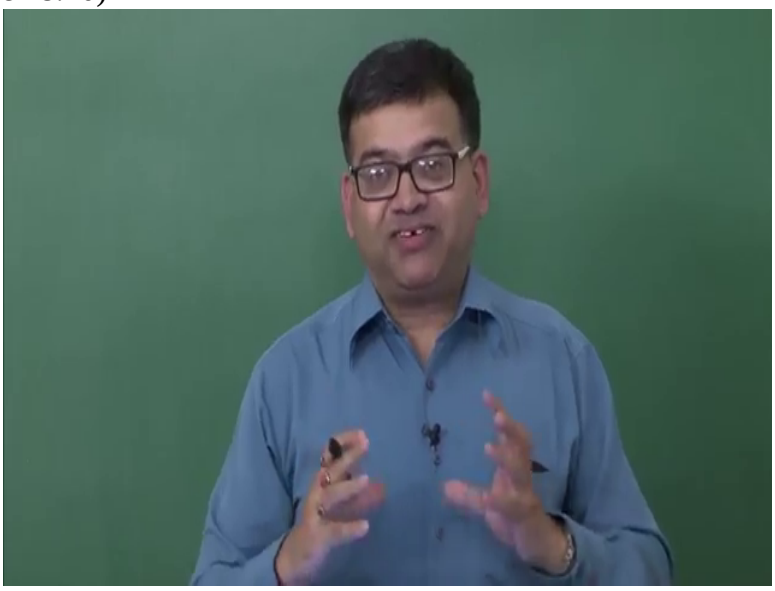
Now this effect of parallel concatenation where you know the multiplicity of low weight codewords getting reduced is known as spectral thinning. And this is observed when we are using systematic feedback encoder in convolutional code as constituent encoders.

(Refer Slide Time 17:55)



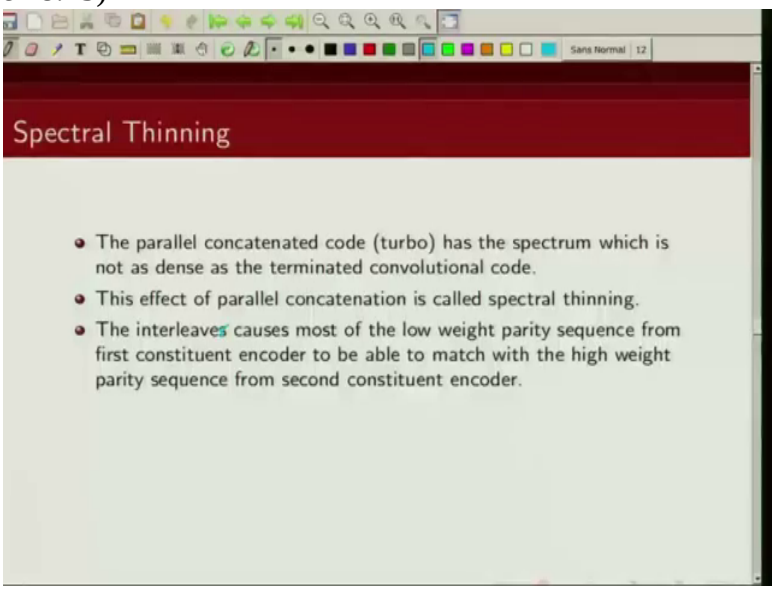
Now interleaver causes most the low weight parity sequence of the first encoder to be able to match to the high weight parity sequence from the second encoder. So the idea of design of interleaver

(Refer Slide Time 18:10)



is such that if the parity sequence coming out of the first encoder has small weight then the interleaved sequence when it is entering the second encoder it should produce a larger weight sequence so that overall code weight sequence is

(Refer Slide Time 18:25)

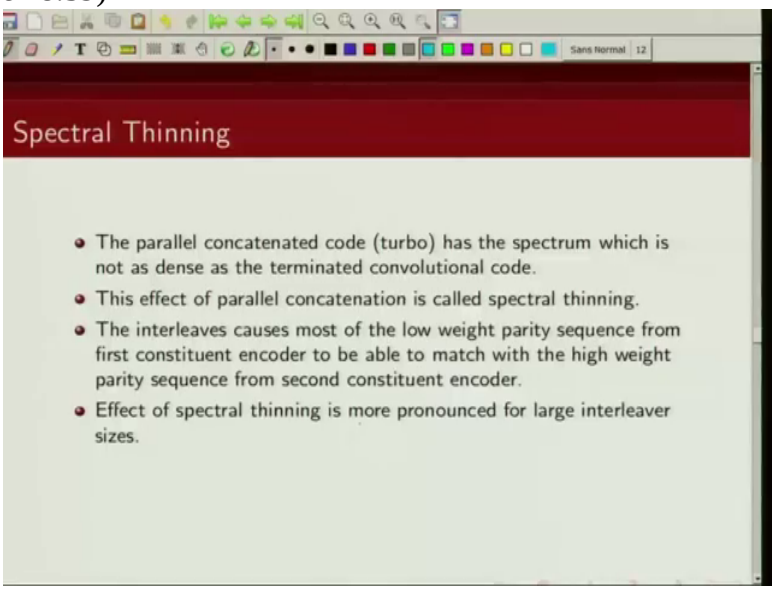


good. Now we just showed the effect of spectral thinning

(Refer Slide Time 18:33)

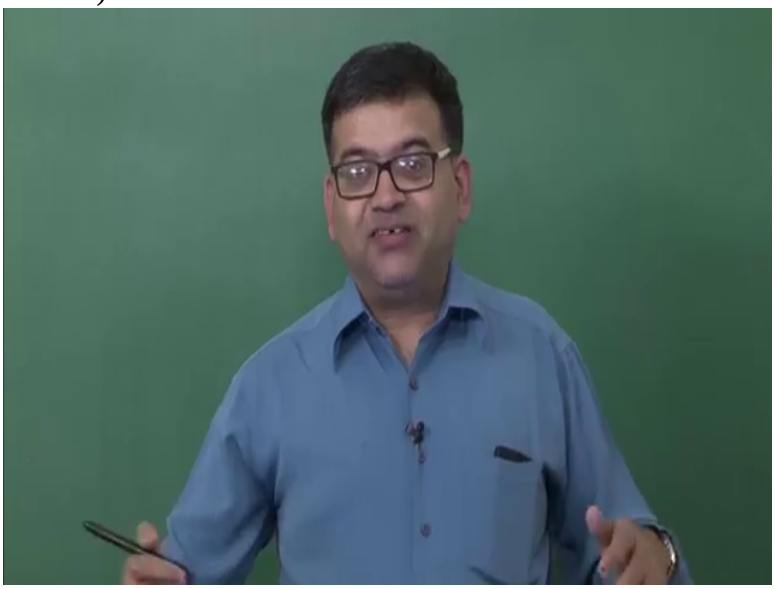
with the help of very small interleaver size of 16. Now this effect is more

(Refer Slide Time 18:39)



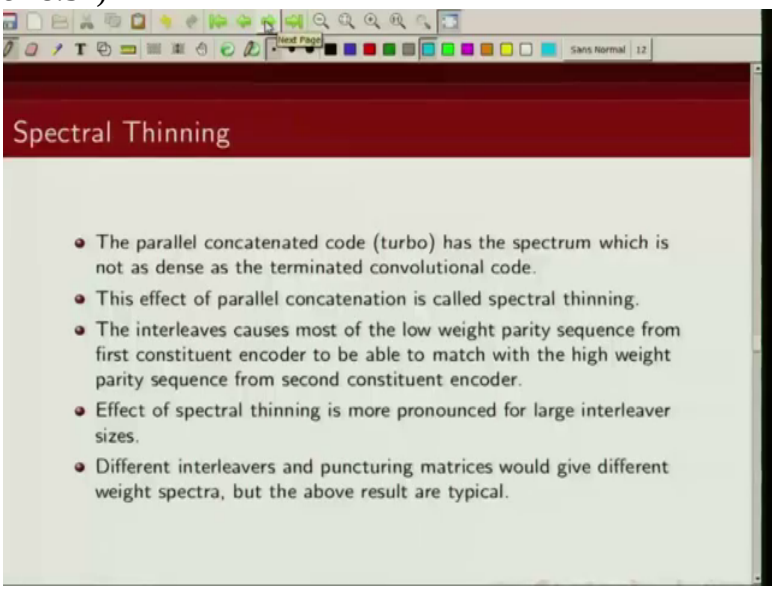
pronounced if we take larger interleaver. So let's say if I take interleaver size of 100

(Refer Slide Time 18:45)



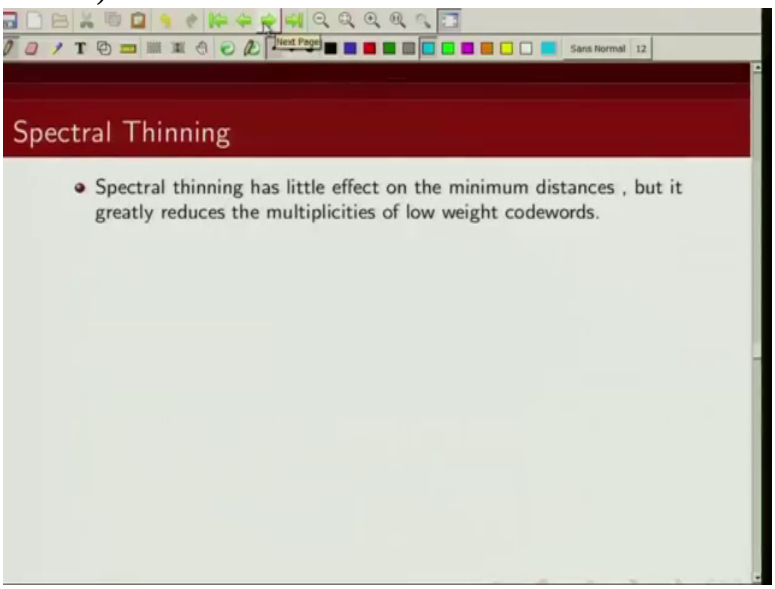
1000, 10000 you will see much pronounced effect of this spectral thinning. Now

(Refer Slide Time 18:52)



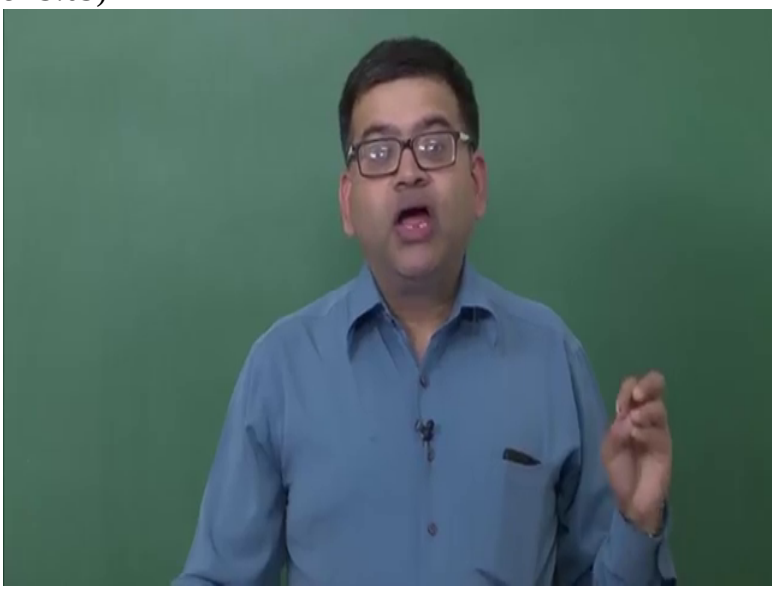
different interleavers and puncturing patterns would give different weight spectrum. But this result is very typical of turbo codes.

(Refer Slide Time 19:04)



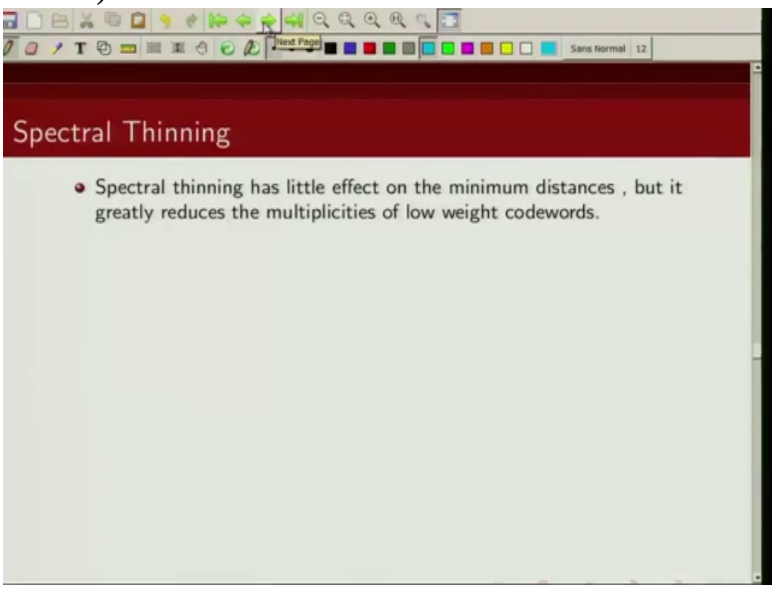
Now spectral thinning has very little effect on

(Refer Slide Time 19:09)



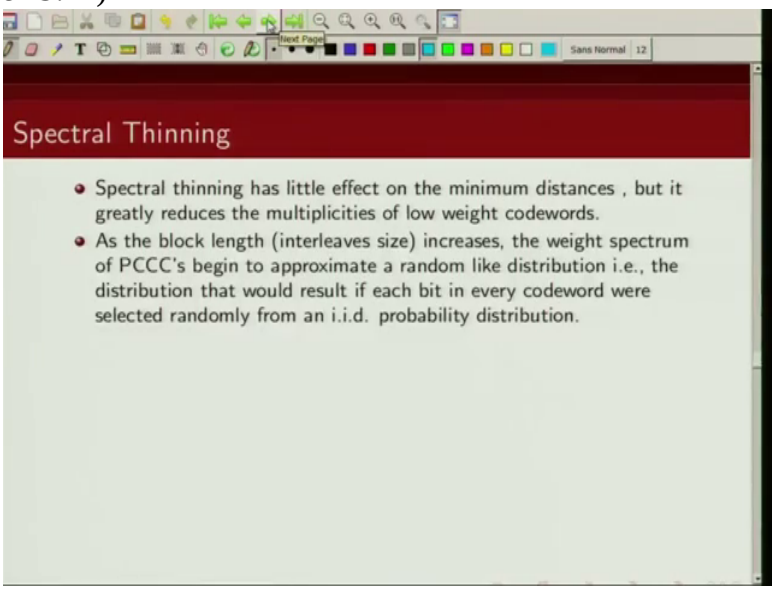
minimum distance of the turbo code. However it greatly reduces the multiplicity of low weight code words.

(Refer Slide Time 19:18)



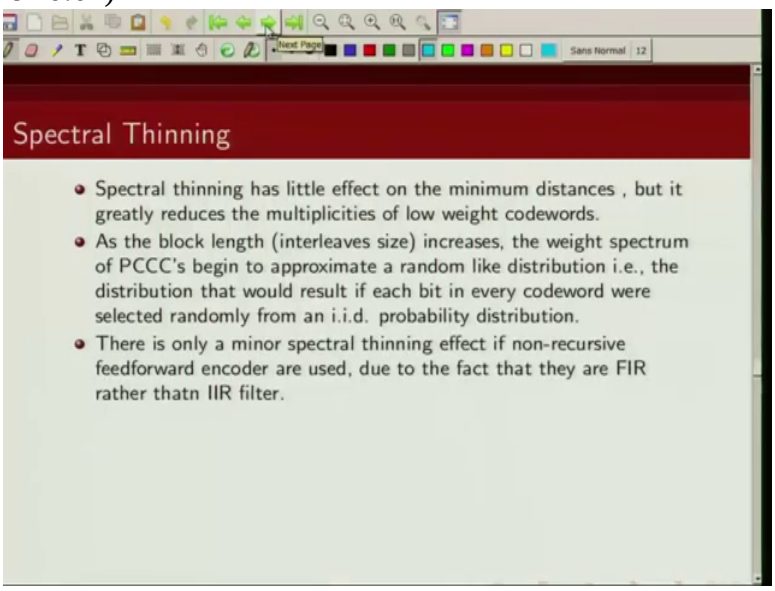
In the example that we considered, we saw that the minimum distance was 5 where as the convolutonal code, for the convolutional code it was 6 but we only had one code word of weight 5 where as if you look at codewords, a number of codewords of weight 6, 7,8, 9 it was reduced in the case of parallel concatenated code.

(Refer Slide Time 19:44)



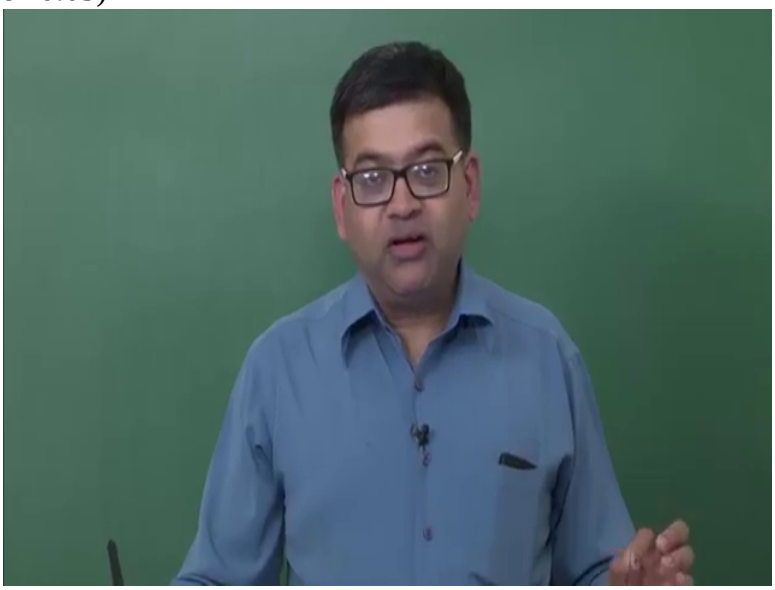
So as the block size increases, as the interleaver size increases the weight spectrum of turbo code looks more like random distribution so it is like the weight distribution as if every bit of the codeword was randomly selected according to some i i d distribution.

(Refer Slide Time 20:01)



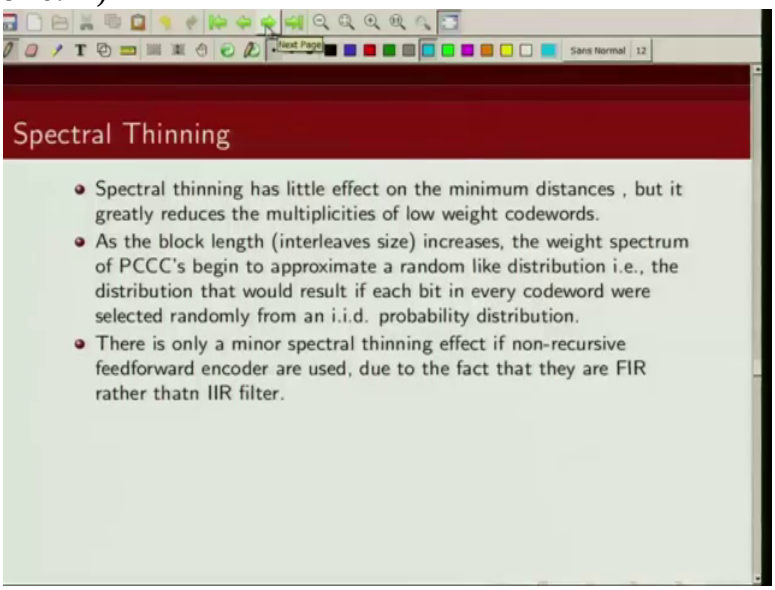
Now the example that we took,

(Refer Slide Time 20:05)



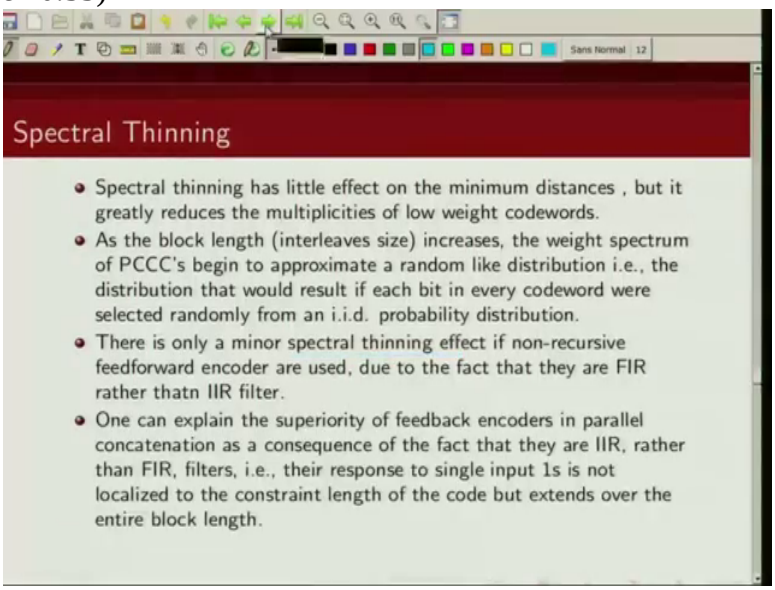
we had considered the feedback systematic encoder. If you replace that feedback encoder with a feed forward encoder we see a very small spectral thinning effect even with large block sizes. So

(Refer Slide Time 20:22)



this spectral thinning effect is pronounced in case of a recursive encoder, not in a case of non recursive encoder.

(Refer Slide Time 20:33)

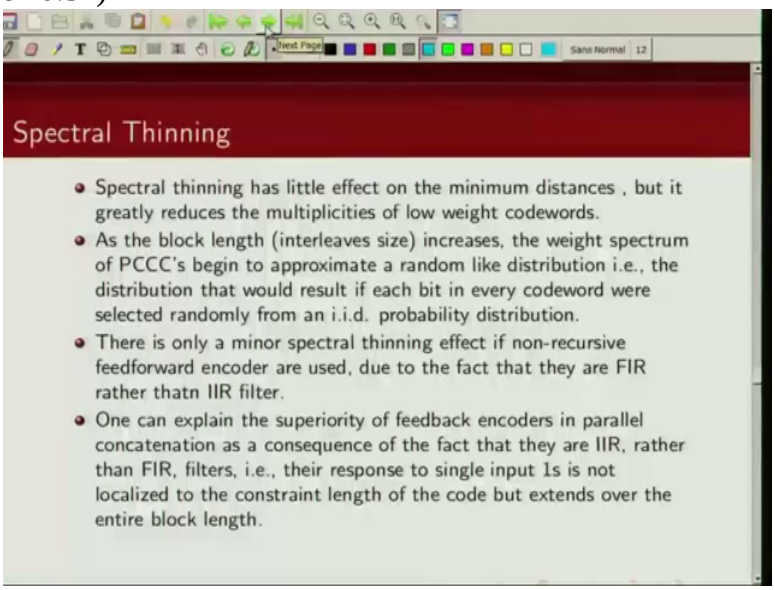


Now we can explain why feedback encoders are better if you feed in a way to one sequence to this

(Refer Slide Time 20:41)

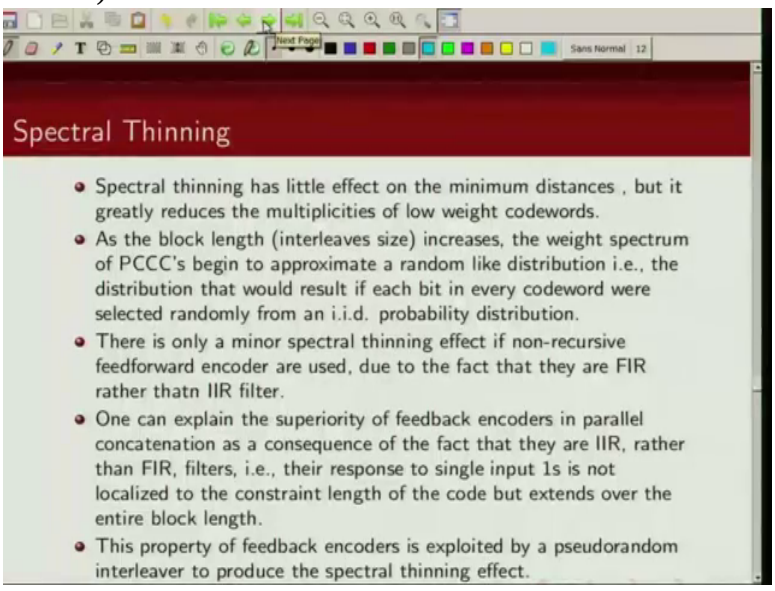
feedback encoders they would not get terminated. So they will result in large weight output sequences where as if we feed a weight 1s sequence to a feed forward encoder it can get terminated and we may get lower weight

(Refer Slide Time 20:57)



sequences. So the response to single input 1 in case of a feed forward encoder is localized where as for a feedback encoder it extends over this entire block size. Now this

(Refer Slide Time 21:15)

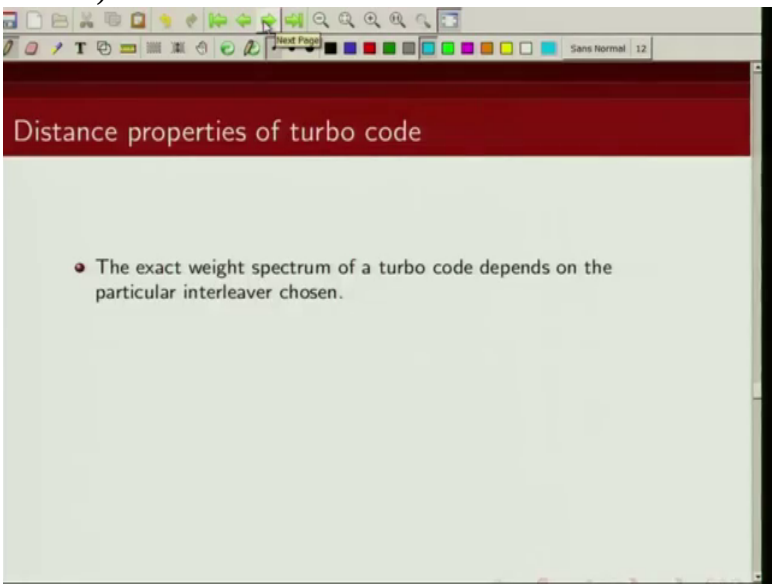


property of feedback encoders is exploited by the pseudo random interleaver to produce this spectral thinning effect.

(Refer Slide Time 21:25)

So what we have done so far is we have, with the help of the simple example for 32 12 terminated convolutional code and in 32 12 turbo codes we had shown the effect of spectral thinning. Now

(Refer Slide Time 21:42)

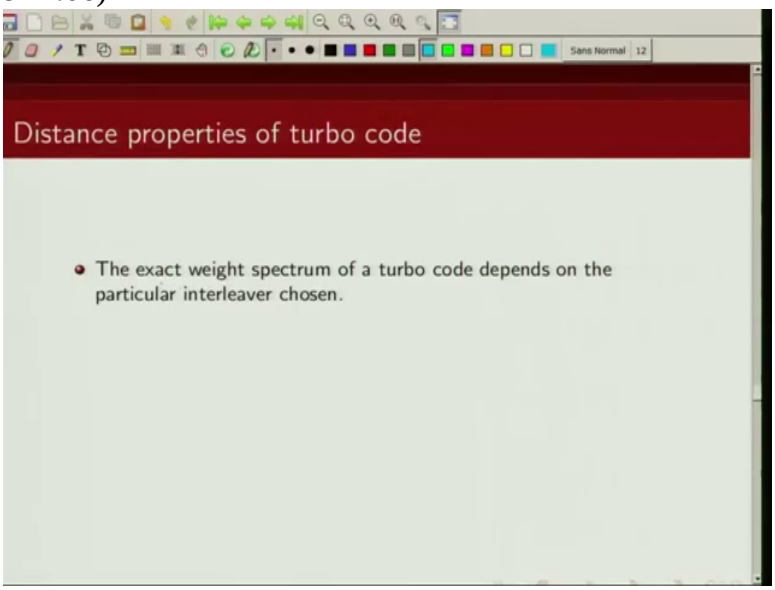


you want to find out what is the, how can we find out the weight distribution of a parallel concatenated code.

(Refer Slide Time 21:52)

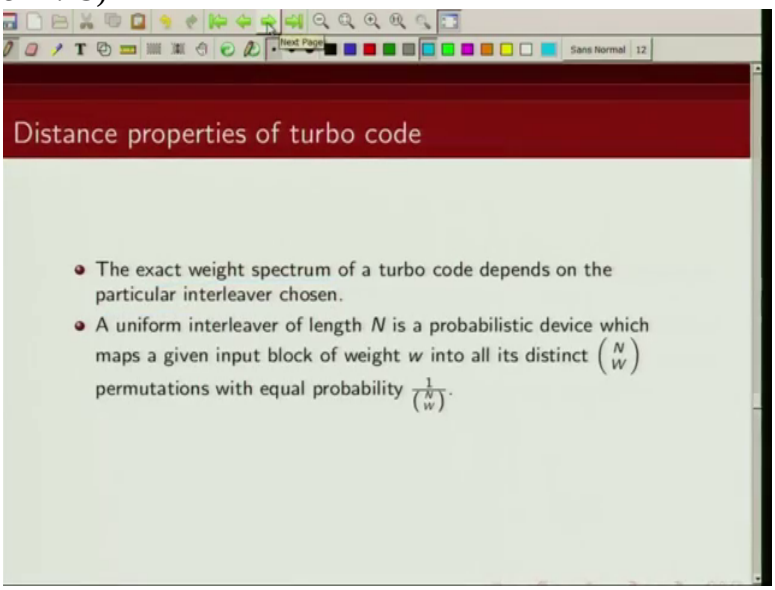
So again we will take a very simple example and with the help of this example we will try to illustrate how we can compute weight distribution of a parallel concatenated code. Now as you can understand

(Refer Slide Time 22:06)



the exact weight spectrum of a turbo code depends on a particular interleaver. So the weight spectrum is dependent on the interleaver we are using.

(Refer Slide Time 22:19)

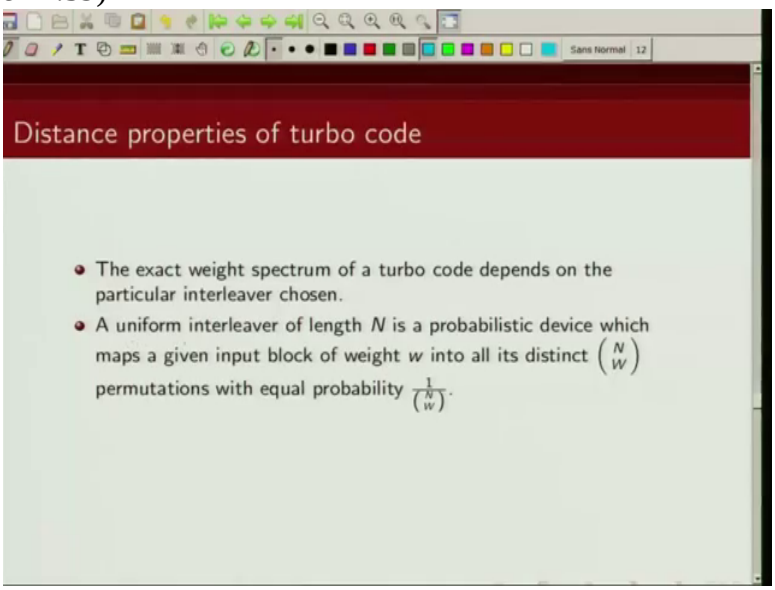


So we are going to make use of a device, we call it uniform interleaver. So this will help us

(Refer Slide Time 22:28)

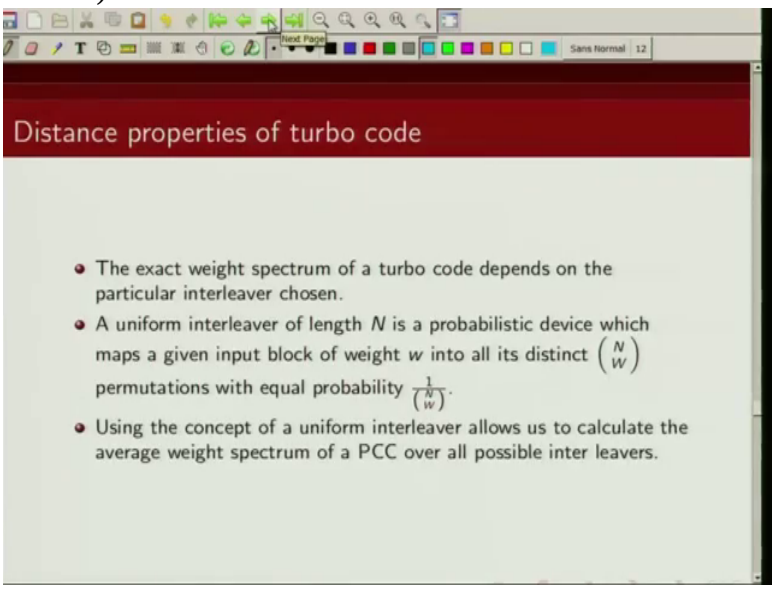
come up with the weight enumerating function of this parallel concatenated code averaged over all interleaver. So what is the uniform interleaver?

(Refer Slide Time 22:39)



A uniform interleaver of length n is a probabilistic device which maps an input block of weight w into all distinct n w permutations with equal probability. So

(Refer Slide Time 22:53)

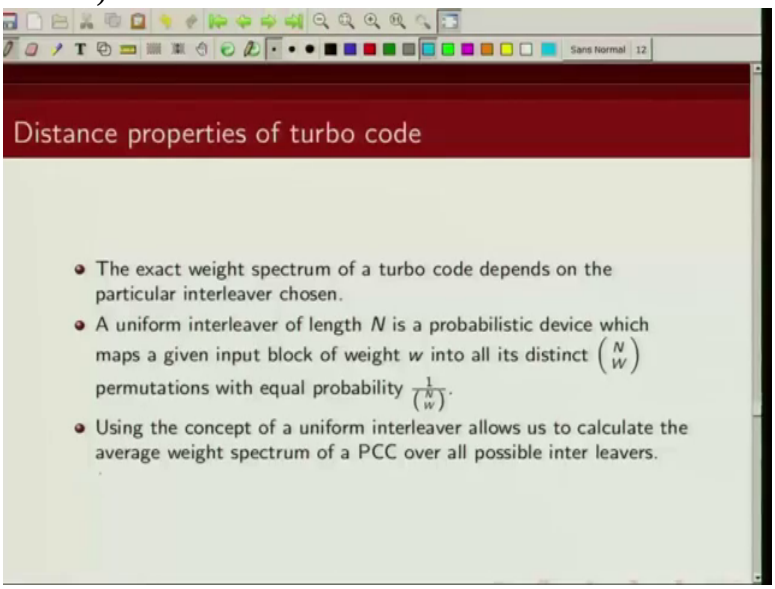


how does uniform interleaver concept helps? Now we can come up with weight distribution

(Refer Slide Time 23:00)

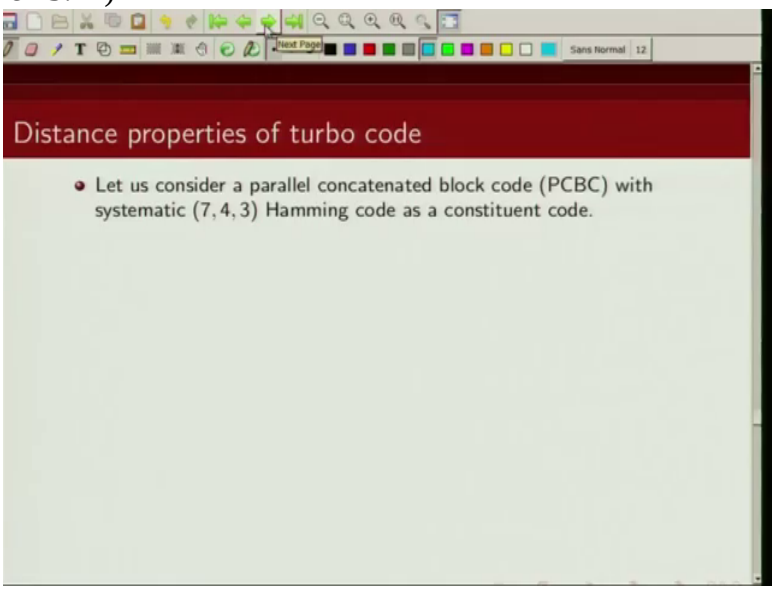
averaged over all possible interleavers and this helps us to compute the

(Refer Slide Time 23:06)



average weight spectrum of parallel concatenated codes.

(Refer Slide Time 23:12)

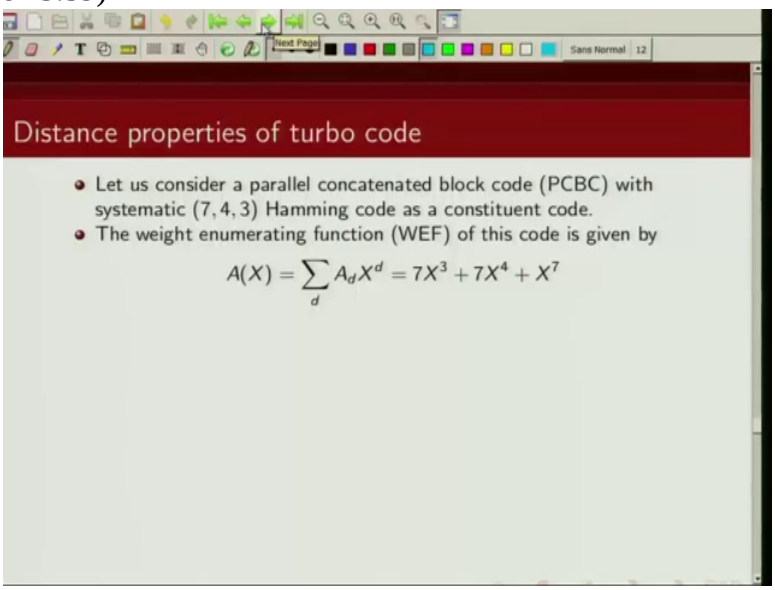


So again we are going to take a very simple example. This time we are going to use our favorite 7 4 Hamming codes as constituent encoders so we are going to use a 7 4 Hamming codes as constituent encoder for turbo code and both these constituent encoders use 7 4 Hamming code. So we are basically constructing a

(Refer Slide Time 23:36)

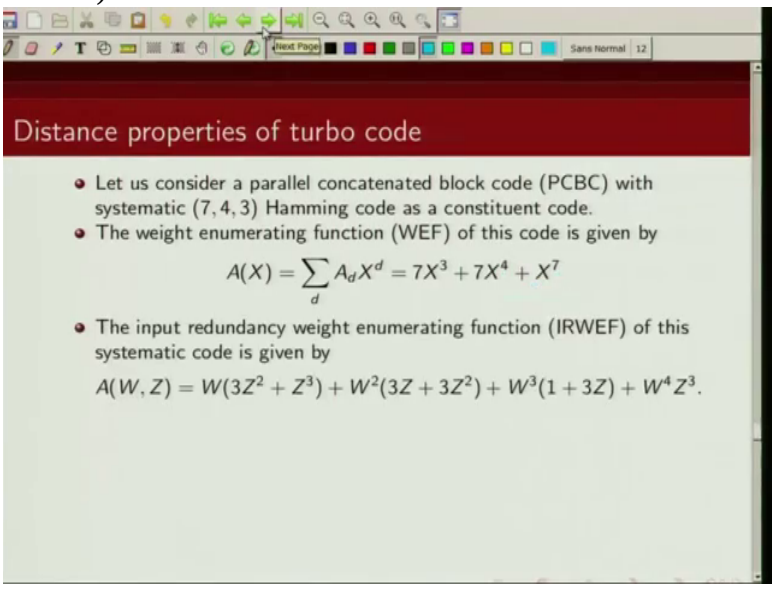
parallel concatenated block code.

(Refer Slide Time 23:39)



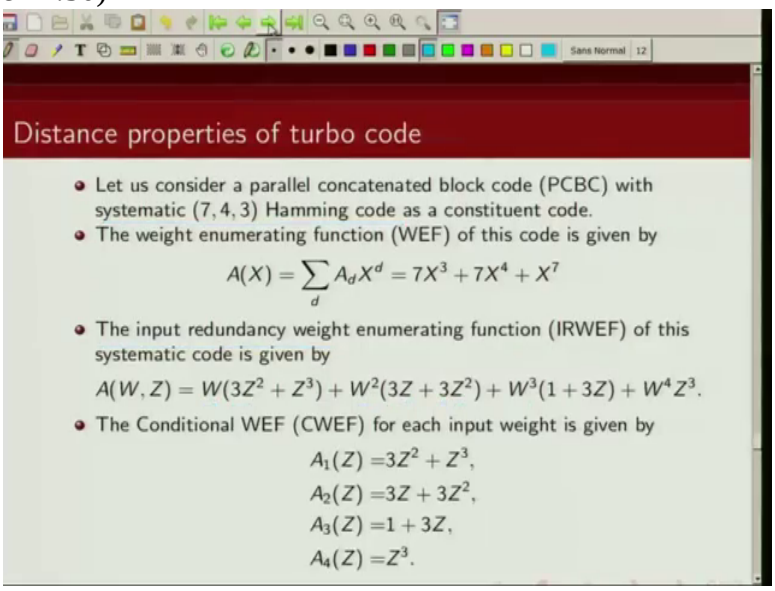
Now we know in systematic form the weight enumerating function of Hamming code is given by this expression. There are 7 codewords of weight 3, 7 codewords of weight 4 and 1 codeword of weight 7. These are the

(Refer Slide Time 23:58)



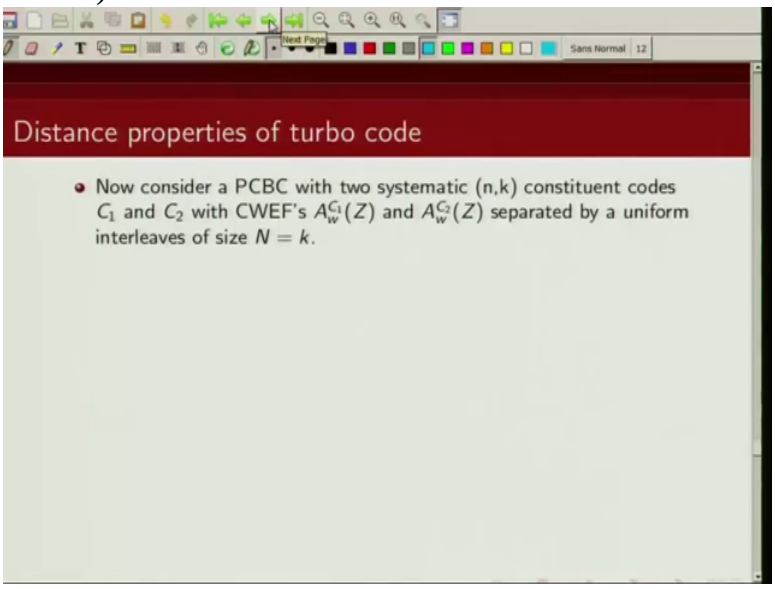
non zero codewords in a systematic 7 4 Hamming code. Now the input redundancy weight enumerating function, this we have already defined when we talked about weight distribution of convolutional code. So the input redundancy weight enumerating function for this systematic Hamming code is given by this, so for our information sequence of weight 1, there exists 3 sequences which are parity weight 2 and one which have parity weight 3. For information sequence of weight 2, there are three sequences which are parity weight 1, three sequences which are parity weight 2 and so on. Now we can also find out the

(Refer Slide Time 24:50)



conditional weight enumerating function. So from this input redundancy weight enumerating function we can find out conditional weight enumerating function. So when input is 1, this is what we get. So the conditional weight enumerating function is given by, for input 1, is 3 z square plus z cube. Similarly for input weight 2, the conditional weight enumerating function is given by this, so that's this. Similarly for weight 3, the conditionally weight enumerating function is 1 plus 3 z and for input weight 4, the conditional weight enumerating function is given by z cube.

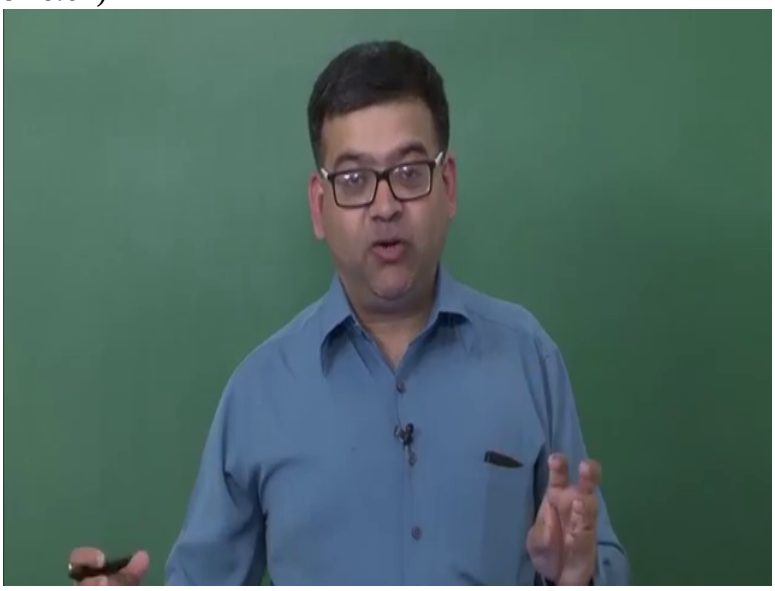
(Refer Slide Time 25:36)



Now let us consider a parallel concatenation of these two Hamming codes, Ok. So we have these 2 systematic Hamming codes c 1, c 2 whose conditional weight enumerating function is

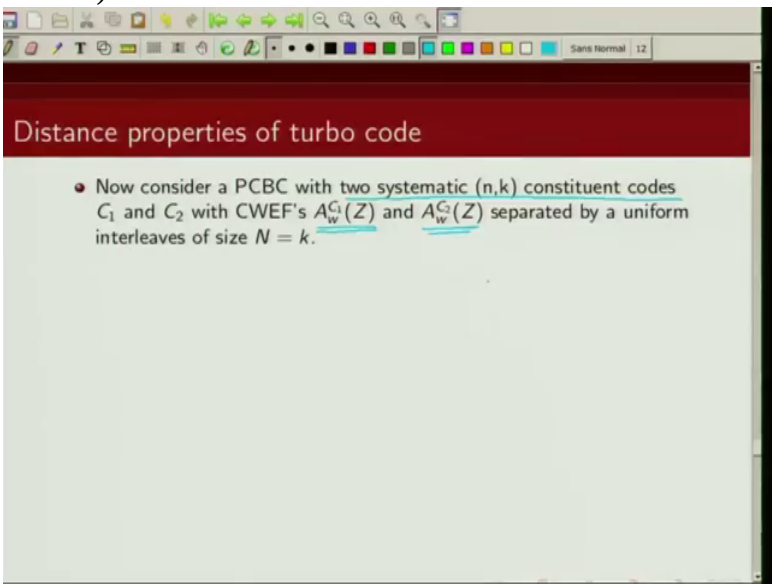
given by this and this. And we apply this concept of uniform interleaver to find out what is the conditional weight

(Refer Slide Time 26:02)



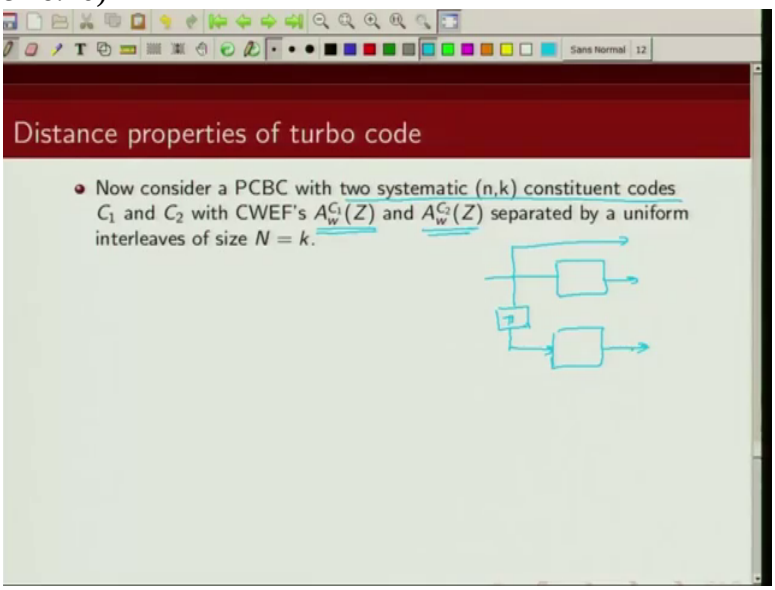
enumerating function of the parallel concatenated code. Now again recall

(Refer Slide Time 26:06)



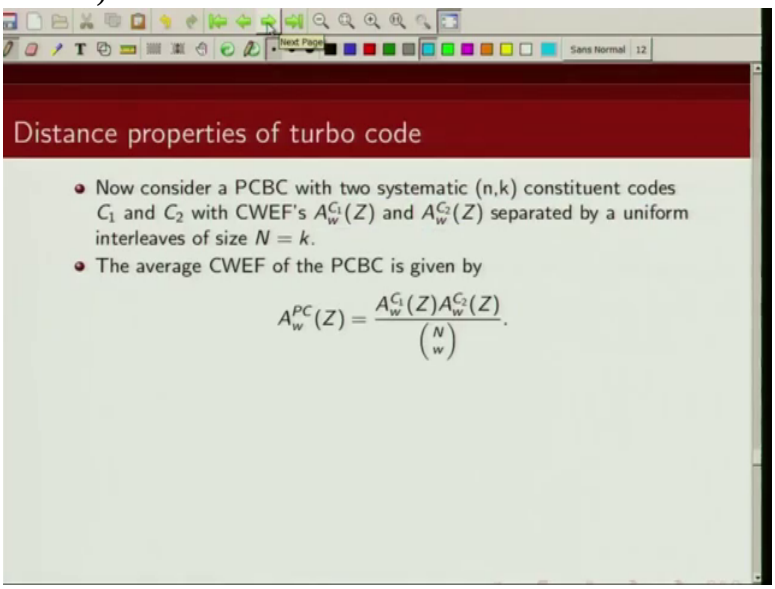
our encoder is something like this. So we have systematic encoder and then we have information bit which is interleaved and fed to second encoder. This is the output.

(Refer Slide Time 26:20)



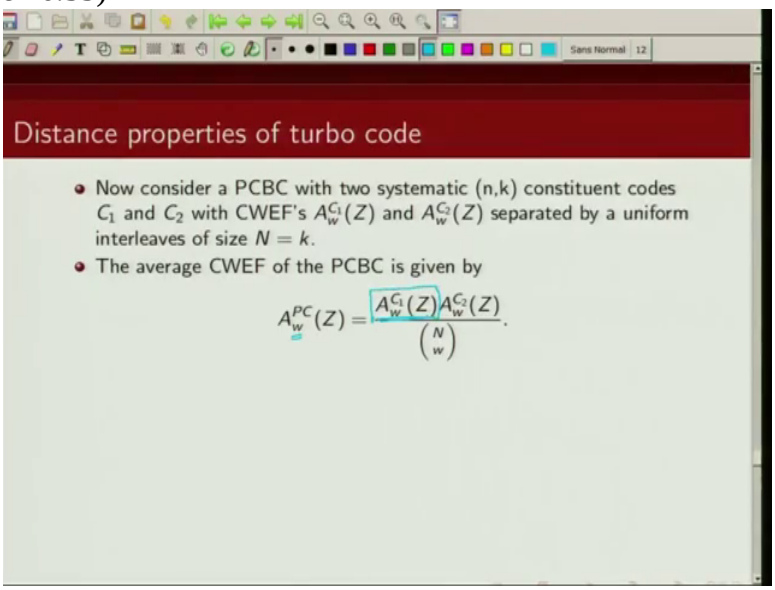
So

(Refer Slide Time 26:22)



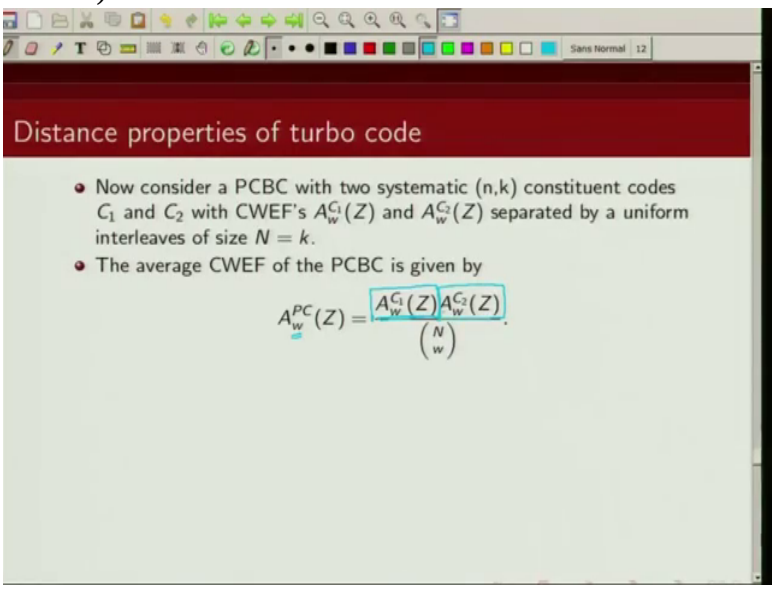
conditional weight enumerating function of the parallel concatenated code can be given by this, so for first encoder we have input w getting in and this will give me the conditional

(Refer Slide Time 26:35)



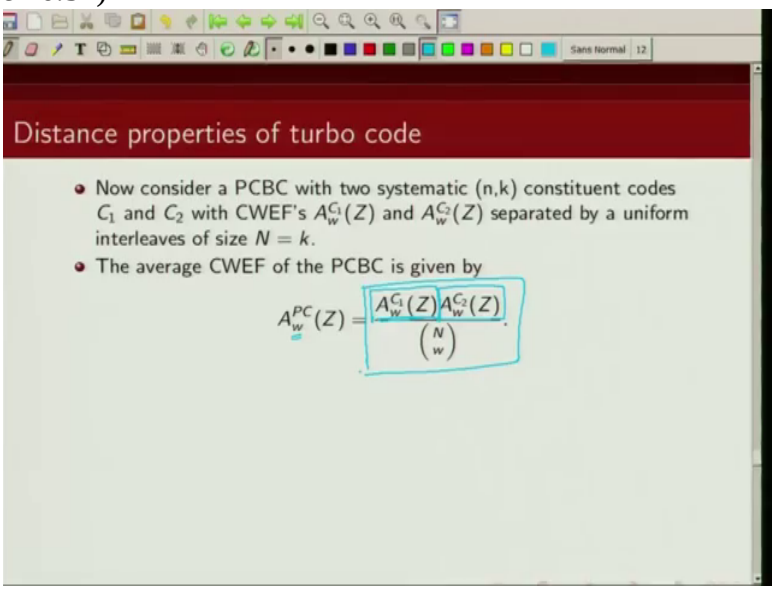
weight enumerating function corresponding to input weight w. Similarly for the second encoder

(Refer Slide Time 26:41)



this will be the weight enumerating function for input w and then averaging it out over all

(Refer Slide Time 26:52)

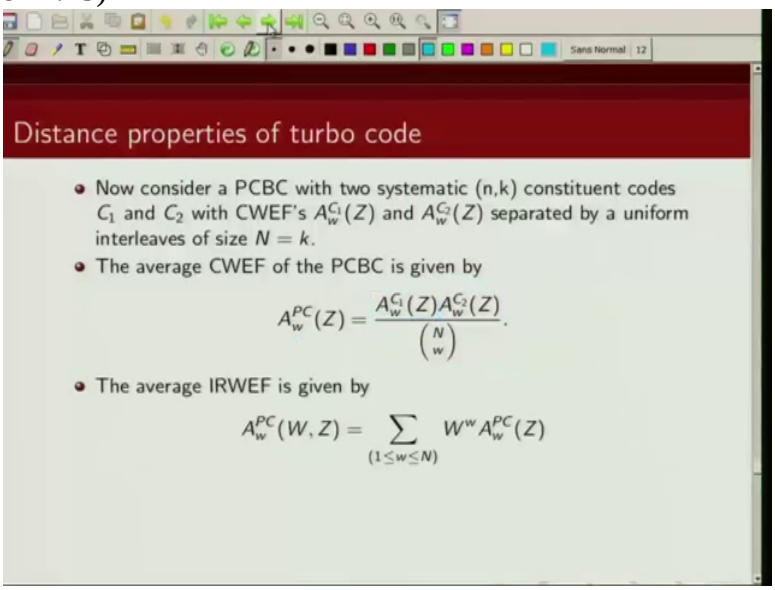


possible interleavers I get this average conditional weight enumerating function given by this expression. So the nice thing about this uniform

(Refer Slide Time 27:02)

interleaver concept is it allows us to compute the average weight enumerating function of this parallel concatenated code which is independent of the type of interleaver that we are using.

(Refer Slide Time 27:15)

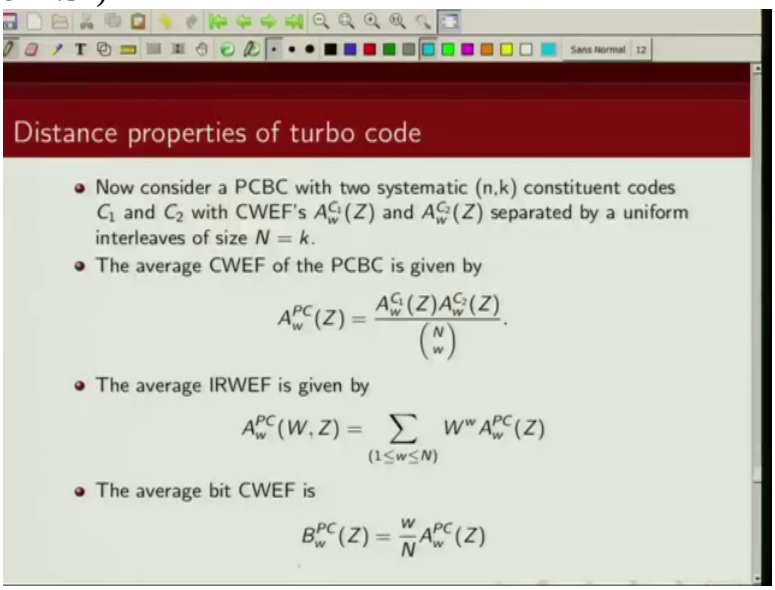


Now similarly we could define average input redundancy weight enumerating function is given by this expression. Now please note this is a, these are straightforward extensions of the definitions that we have studied when talked about

(Refer Slide Time 27:32)

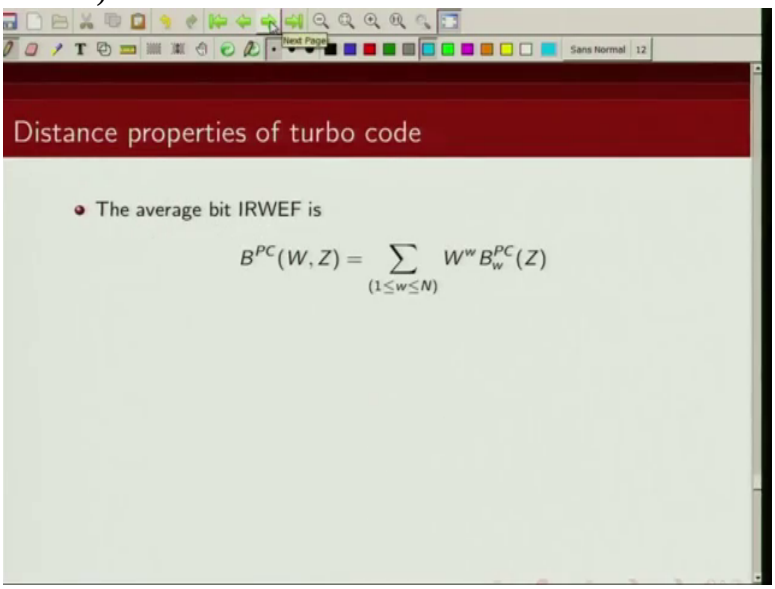
weight enumerating function of convolutional code.

(Refer Slide Time 27:37)



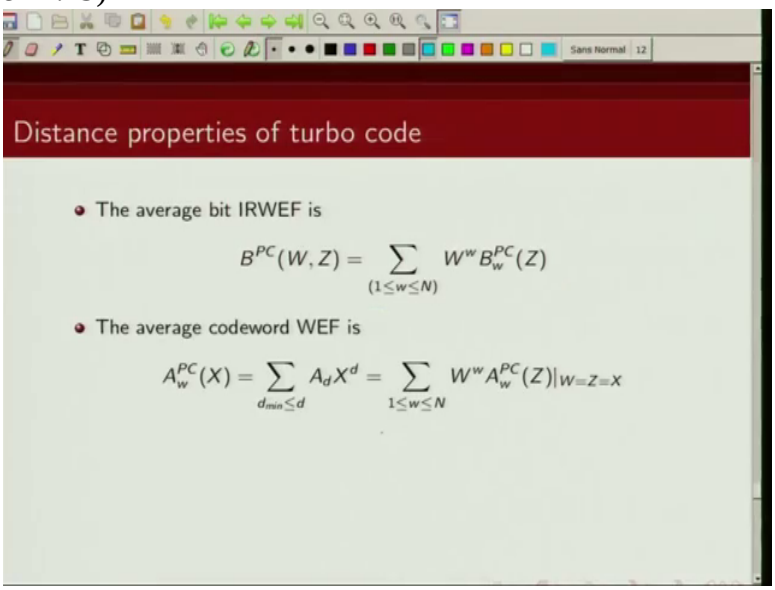
We can define bit conditional weight enumerating function,

(Refer Slide Time 27:42)



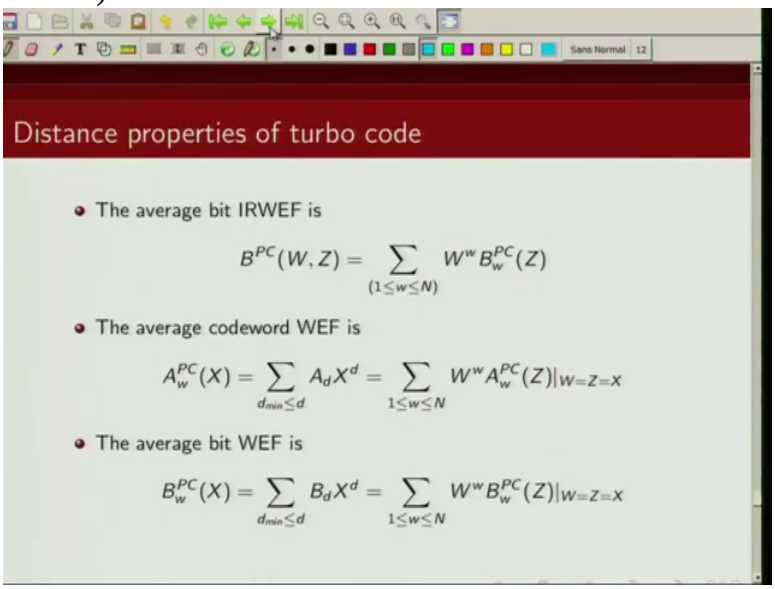
average bit input redundancy weight enumerating function,

(Refer Slide Time 27:49)



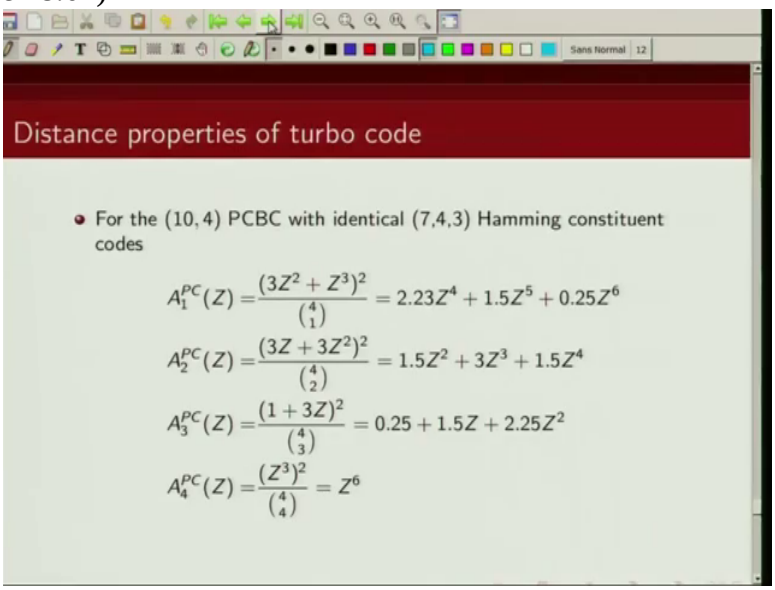
average codeword weight enumerating function and

(Refer Slide Time 27:55)



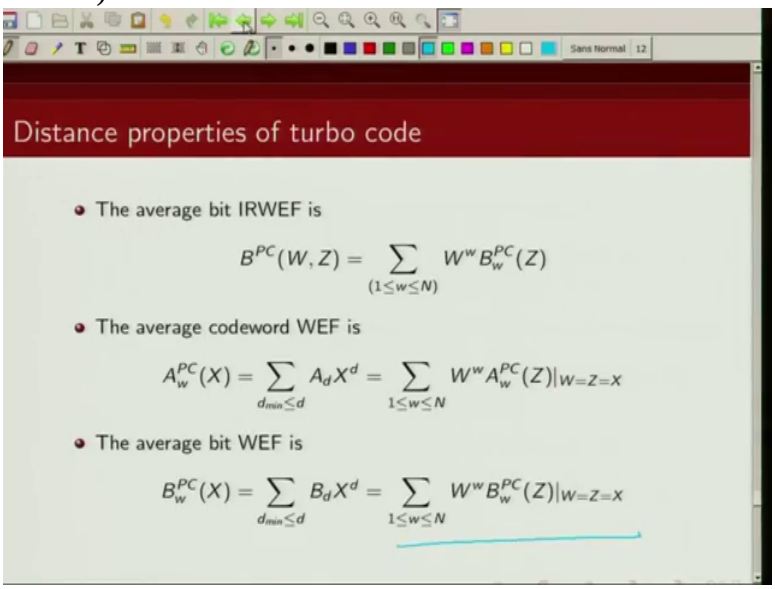
average bit weight enumerating function .

(Refer Slide Time 28:01)



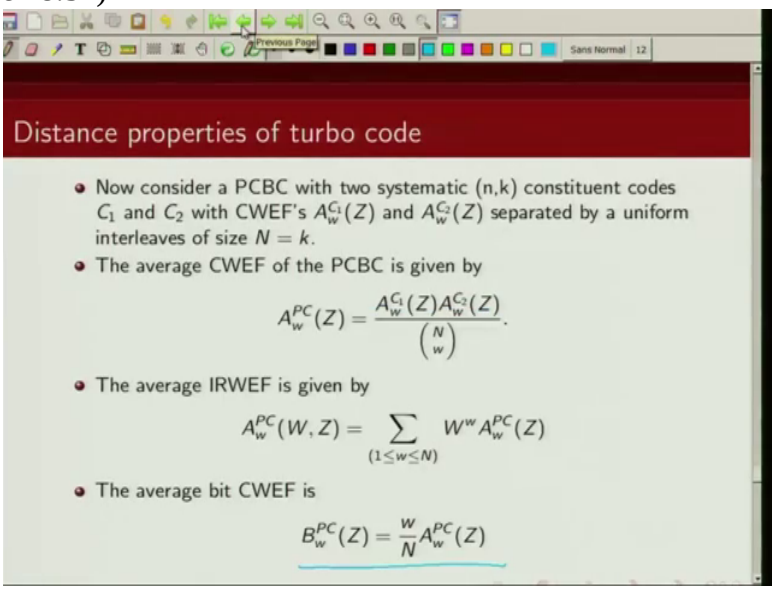
Now let us take an example to illustrate this. So we have this parallel concatenated code using 7 4 Hamming code. So input is length 4 and we get 3 parity bits from first encoder, we get 3 parity bits from the second encoder and of course there are 4 systematic bits, so overall codeword length is 10. So what we get is a 10 4 parallel concatenated block code which I am using identically considering encoders and there is the odd 7 4 Hamming code. Now using the expressions I have said before, we already know the conditional weight enumerating function of the constituent encoder. Now using this

(Refer Slide Time 28:53)



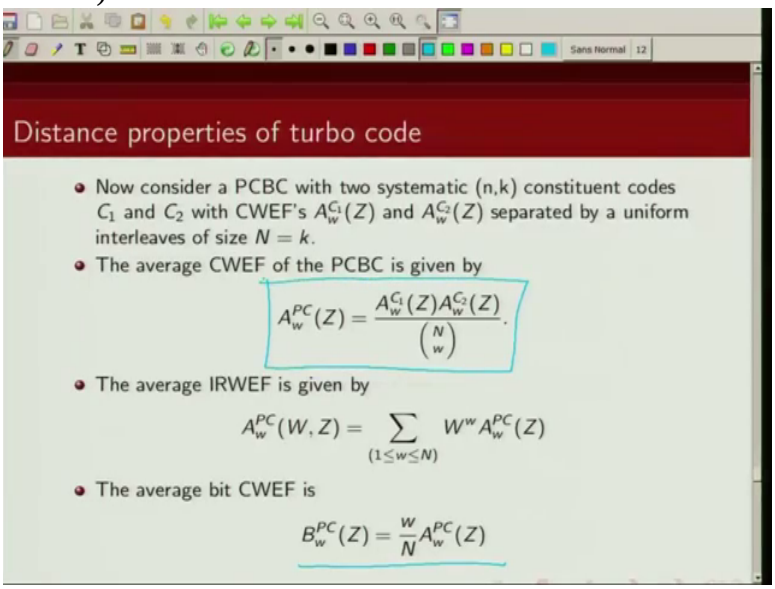
expression,

(Refer Slide Time 28:57)



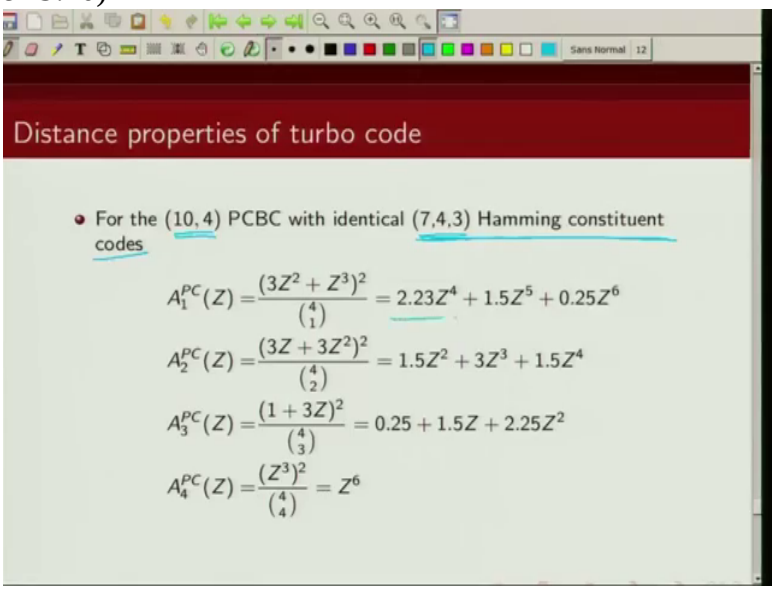
I can compute

(Refer Slide Time 29:00)



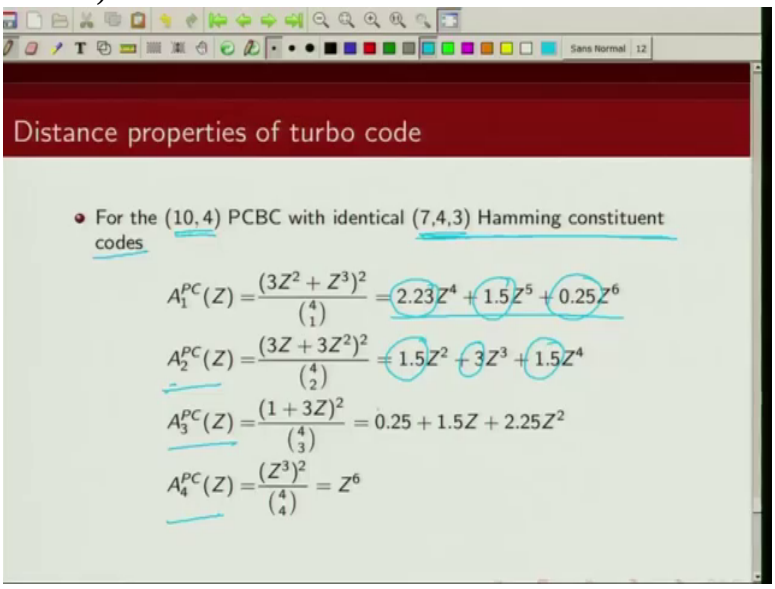
the conditional weight enumerating function of the parallel concatenated block code. So these are the

(Refer Slide Time 29:10)



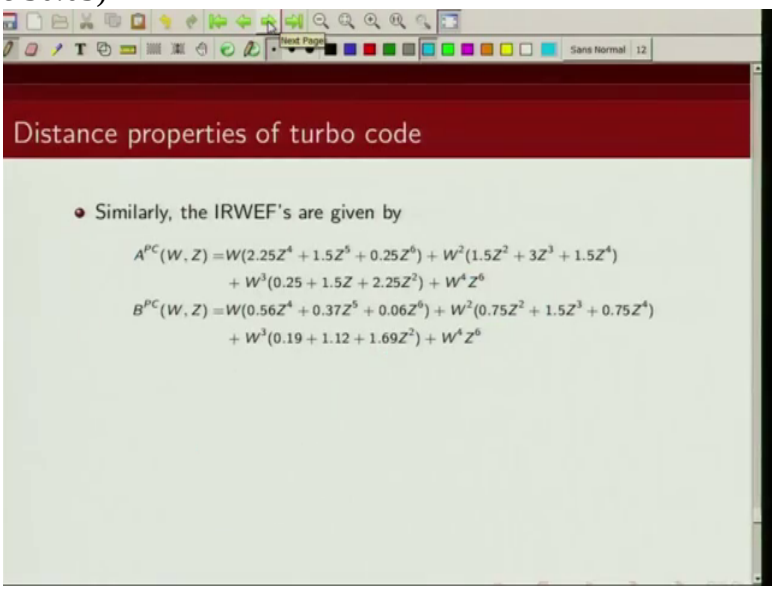
average weight enumerating, so corresponding to input 1, so you can there are 2 point 2 3 on average sequences of parity weight 4, 1 point 5 sequences of parity weight 5 and point 2 5 on an average sequences of parity weight 6. Similarly we could find out for input weight 2, input weight 3, input weight 4. Now note the difference from the constituent encoders that we have studied. Here you can see these numbers, its coefficients are coming as non integers,

(Refer Slide Time 29:56)



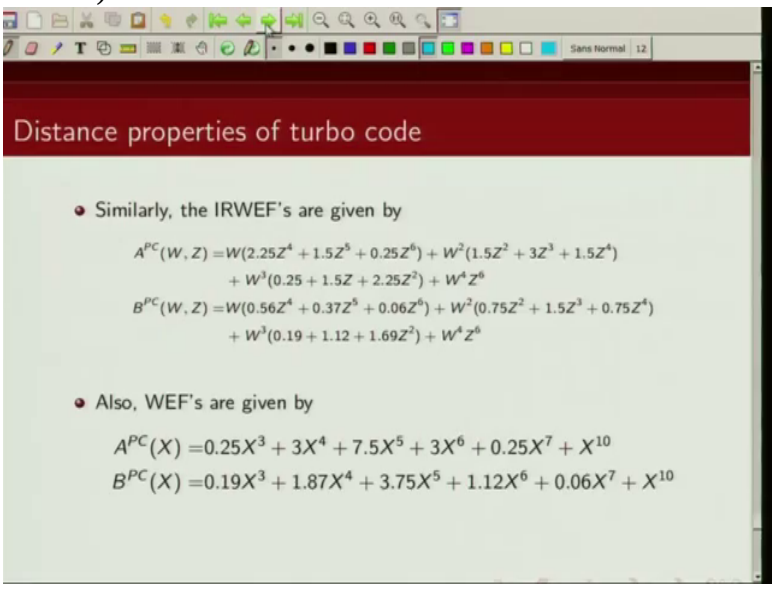
can come as non integers, right? That's because we are averaging over all possible interleavers.

(Refer Slide Time 30:05)



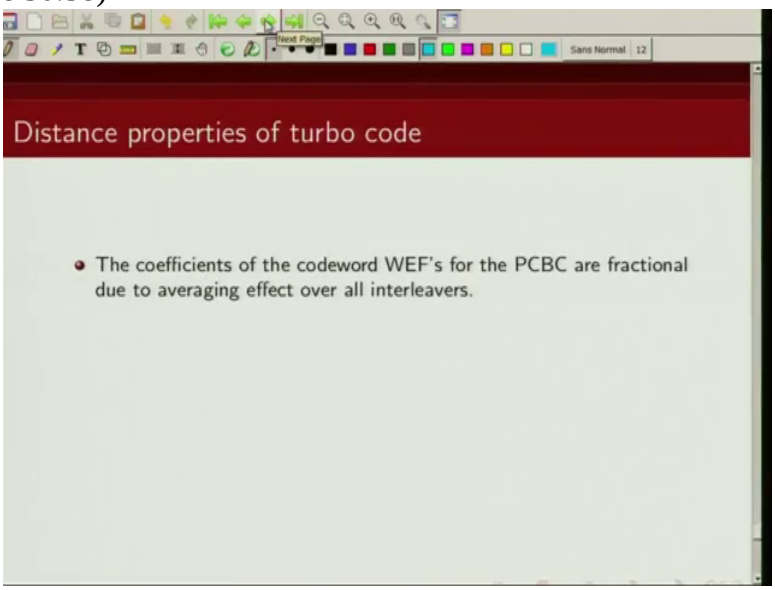
And again following the definition of input redundancy weight enumerating function we can find out the input redundancy weight enumerating function, bit input redundancy weight enumerating function which is given by these two expressions. And

(Refer Slide Time 30:22)



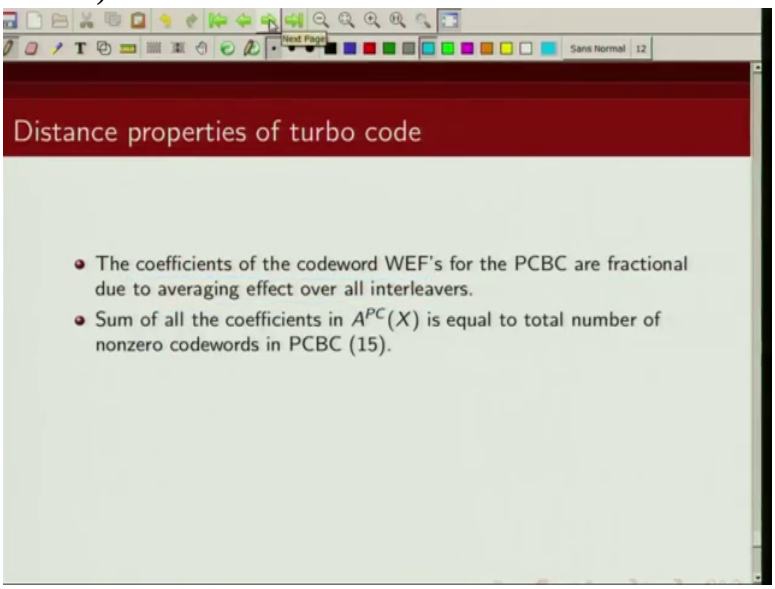
similarly if we put w equal to, z is equal to x, we can get the weight enumerating function and the bit weight enumerating function which is given by this. Now points to be noted.

(Refer Slide Time 30:38)



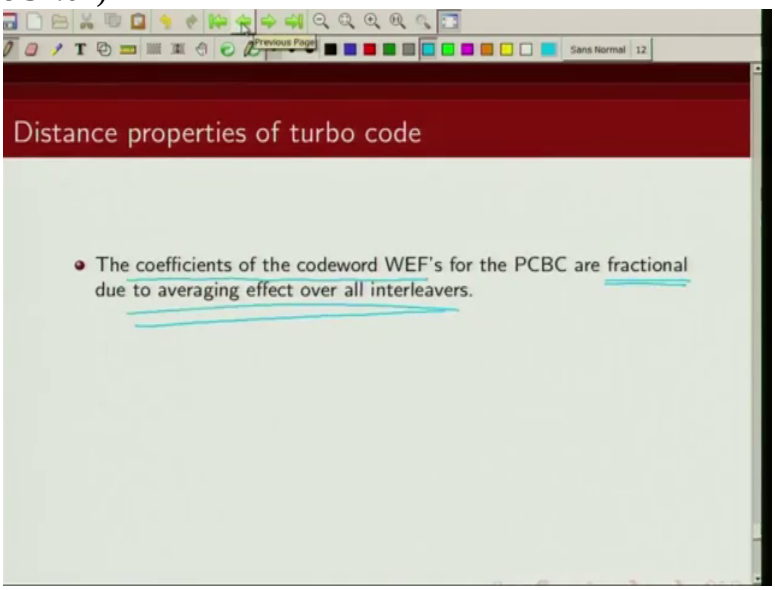
We saw the coefficients of the codeword weight enumerating function are fractional and why they are fractional, because this is due to averaging effect over all interleavers. However

(Refer Slide Time 30:54)



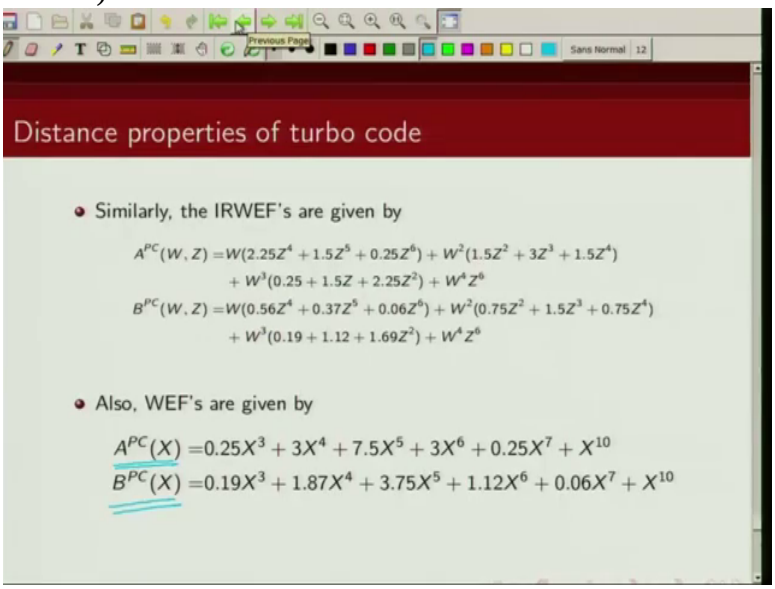
if you sum up all the coefficients, they add up to total number of non zero codewords. You can

(Refer Slide Time 31:02)



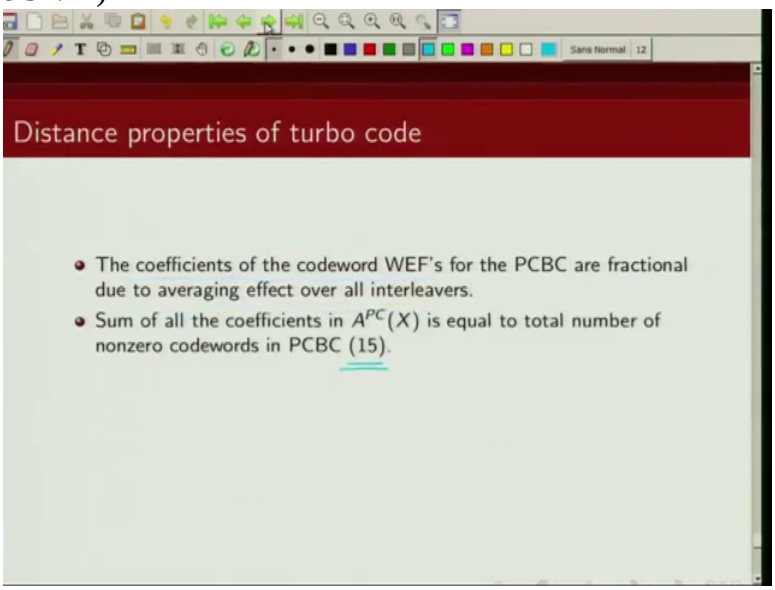
verify that. If you

(Refer Slide Time 31:04)



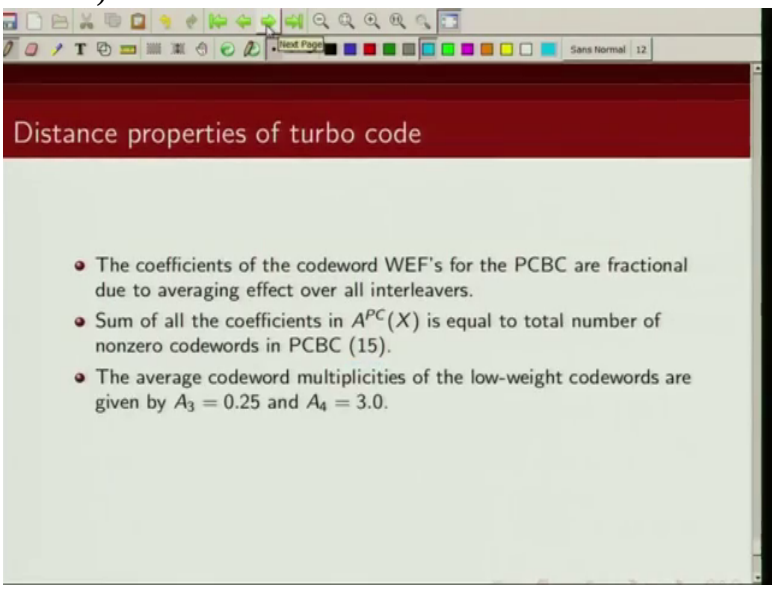
add up this, Ok, point 2 5, point 2 5 is point 5, 7 point 5 this is 8, 8 plus 6 is 14 plus 1 15. So you can see, if you add

(Refer Slide Time 31:17)



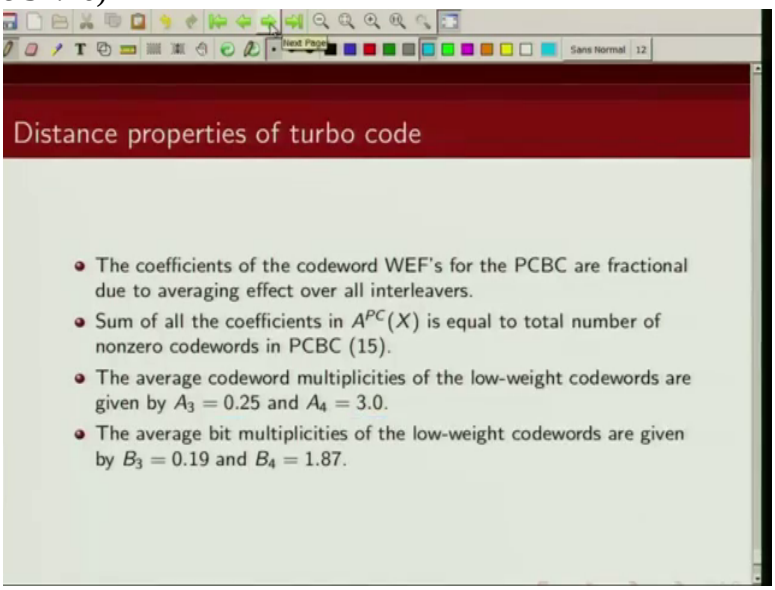
them up we get the total number of non zero codewords.

(Refer Slide Time 31:23)



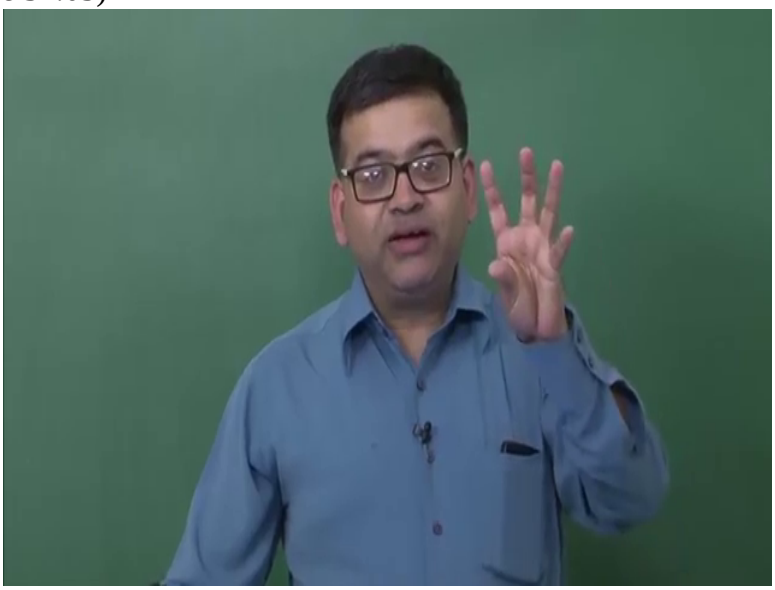
And you can also see that average codeword multiplicity has reduced. Its point 2 5 when the input, corresponding to input of 3 and it is 3 corresponding to input of 4,

(Refer Slide Time 31:46)



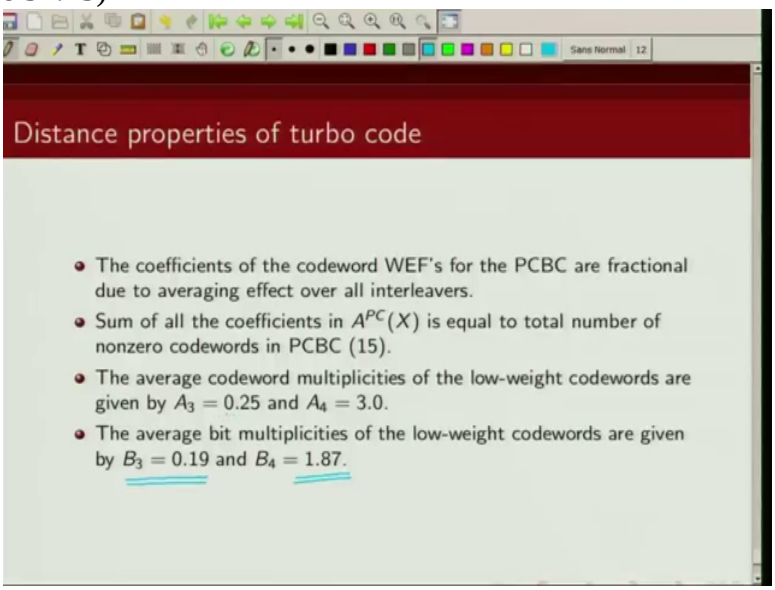
similarly bit multiplicity for input of 3 is point 1 9 and it is 1 point 8 7 for input of length, of weight 4. So if you compare this code multiplicity from the original code that we used, we were using

(Refer Slide Time 32:05)



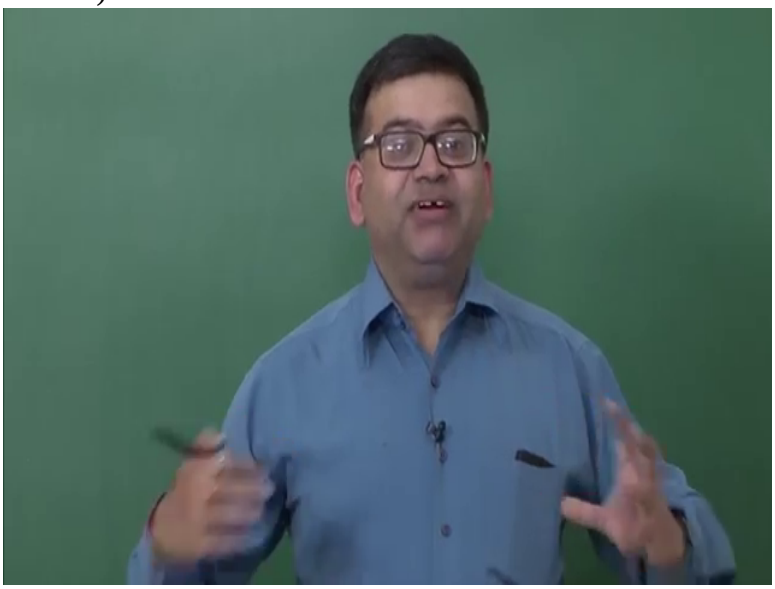
7 4 Hamming code which had 7 codewords of weight 3. Here you can see

(Refer Slide Time 32:13)



the weight multiplicity is point 2 5. So you can see the effect

(Refer Slide Time 32:18)



of interleaver on code multiplicity and you can use this concept of uniform interleaver to compute your average weight enumerating function , average input output weight enumerating function, average bit conditional weight enumerating function. So all those quantities that we have defined in the context of weight enumerating function for convolutional code, you could do similar analysis for these parallel concatenated codes using this idea of uniform interleaver. So with this I am going to conclude the discussion on these distance properties of turbo codes, thank you.