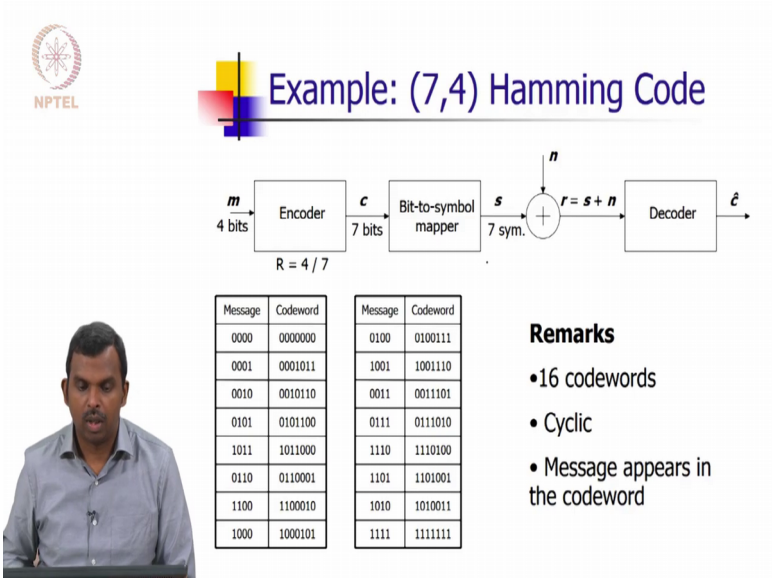
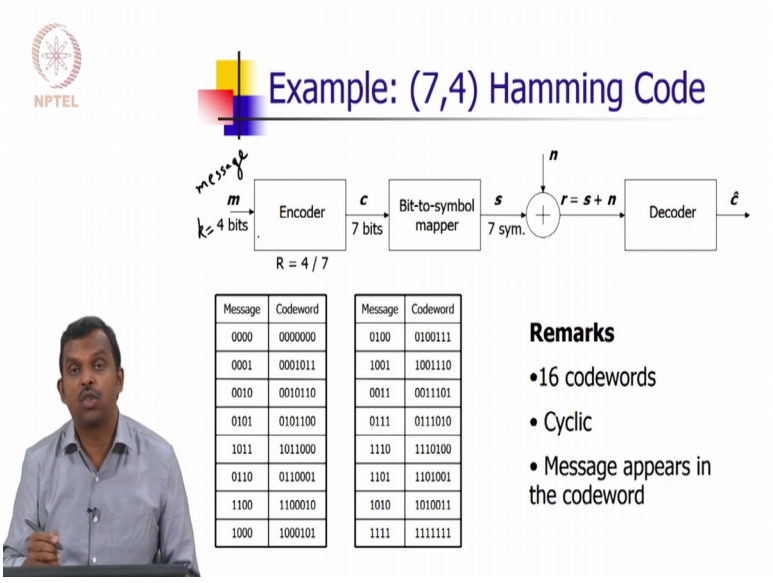
LDPC and Polar Codes in 5G Standard
Professor Andrew Thangaraj
Department of Electrical Engineering
Indian Institute of Technology Madras
(7, 4) Hamming Code

(Refer Slide Time 00:16)



Ok so this is the 7 comma 4 Hamming code. So where do the 7 and 4 come from? So this is your message vector, Ok. This is your message vector and this has k equals 4 bits

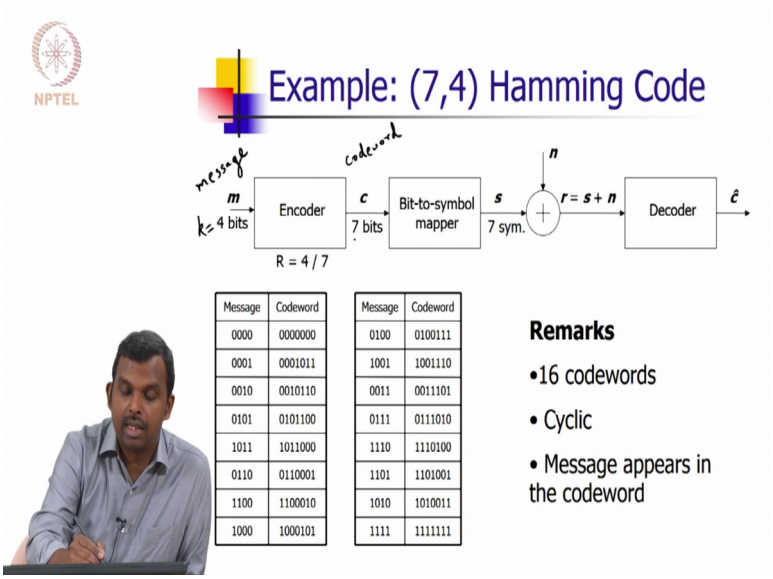
(Refer Slide Time 00:33)



of message. So previously in the repetition code, k was 1. In this case k is going to be 4.

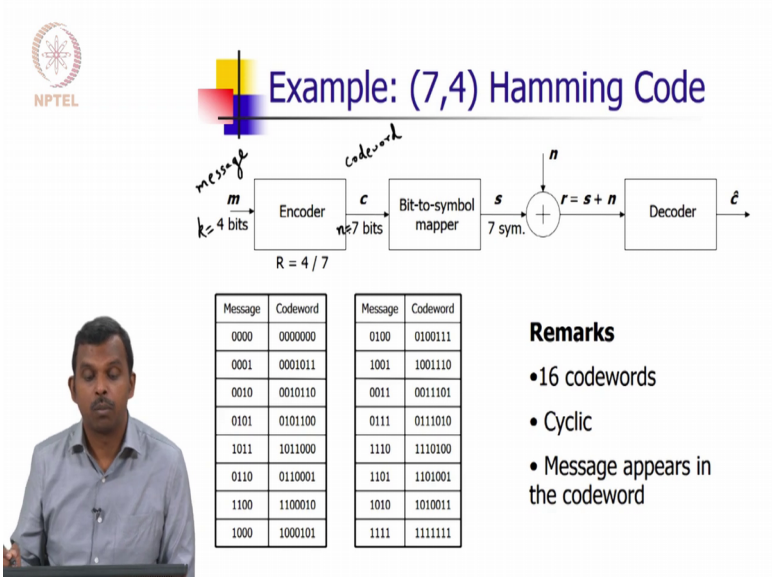
We are going to take 4 bits at a time and then do the encoding, Ok. And your codeword, which I will call c

(Refer Slide Time 00:47)



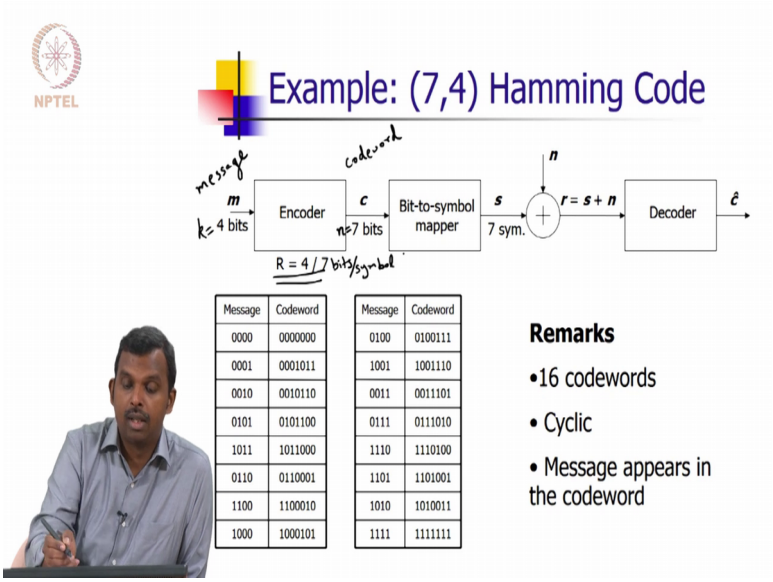
has n equal 7 bits, Ok.

(Refer Slide Time 00:50)



So 4 bits of message gets converted into 7 bits of codeword and your rate is 4 by 7 bits per symbol,

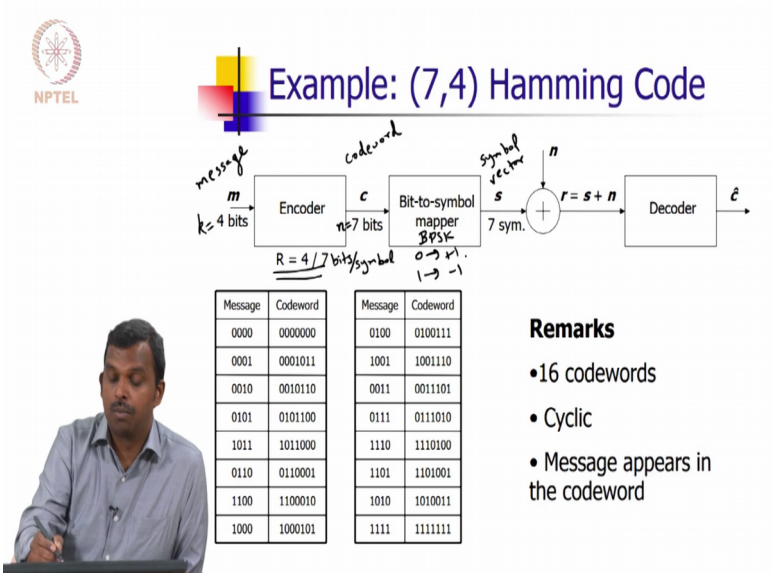
(Refer Slide Time 01:01)



Ok.

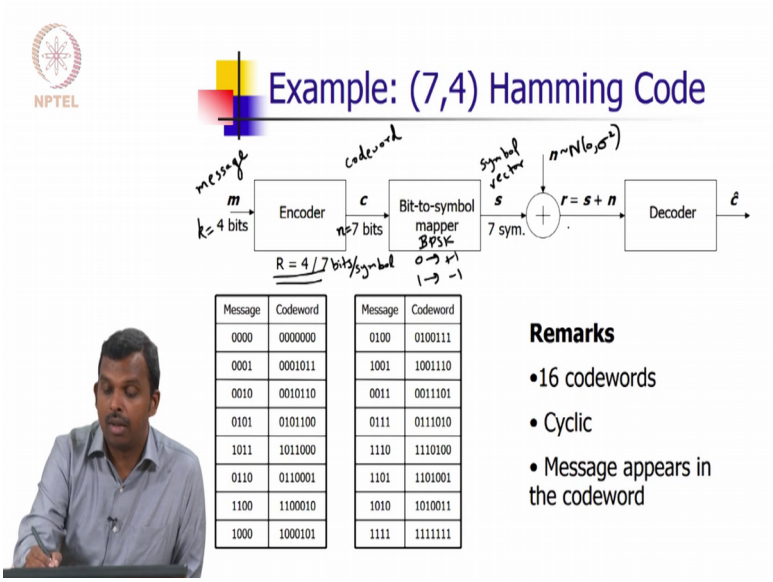
So once you convert it into a codeword, everything else remains the same. So each bit gets converted into a symbol vector. And the same B P S K mapper is used, 0 goes to plus 1, 1 goes to minus 1 etc and

(Refer Slide Time 01:19)



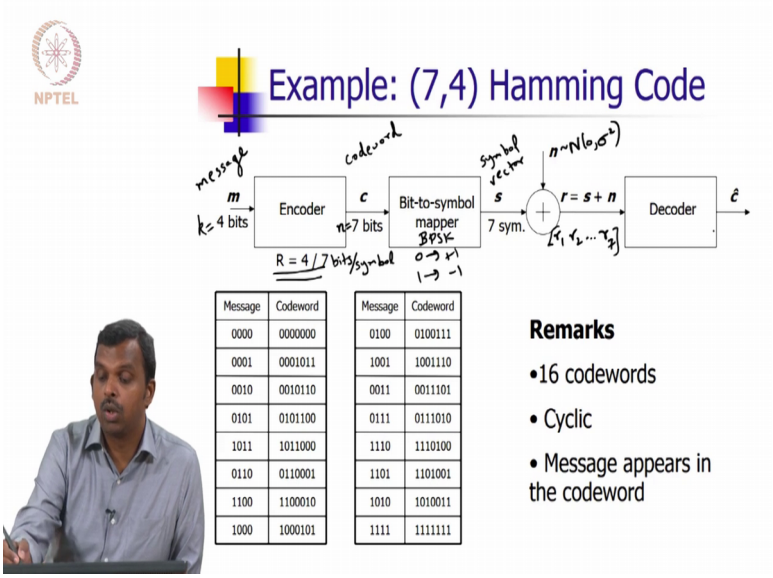
then noise gets added which is Gaussian mean zero, variance sigma square.

(Refer Slide Time 01:24)



You have 7 received values, r 1, r 2 till r 7.

(Refer Slide Time 01:31)



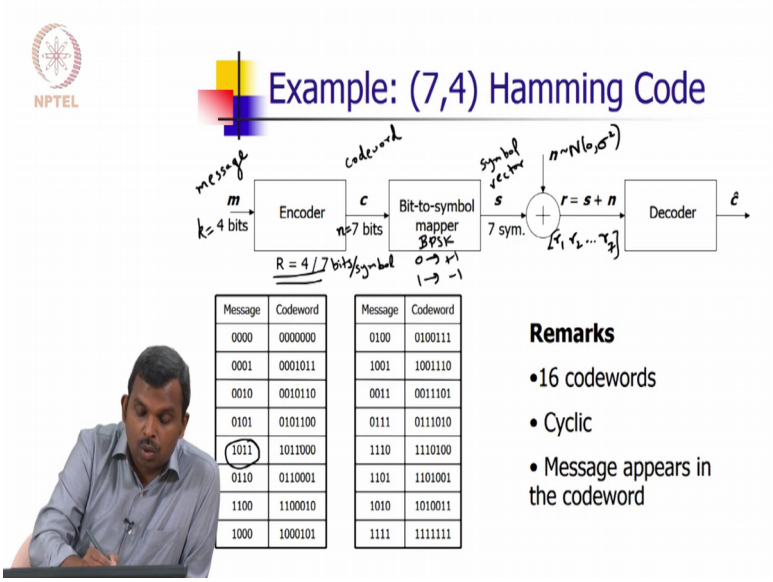
And your decoder needs to work. It could be hard decision or soft decision and you get your c cap, Ok.

So what is this encoding? And how does the Hamming code work? I have shown you encoding as a table. Remember the encoder can be written down as a table. You have 4 bit messages. There are 16 different possibilities, 0 0 0 0 all the way to 1 1 1 1, that is what is shown in the first column here and the first column here, 16 different possibilities.

Each of these 16 message vectors gets mapped into a unique codeword and what codeword is that? That is given on the second column here, Ok. So this is the codeword for the 7 4 Hamming code.

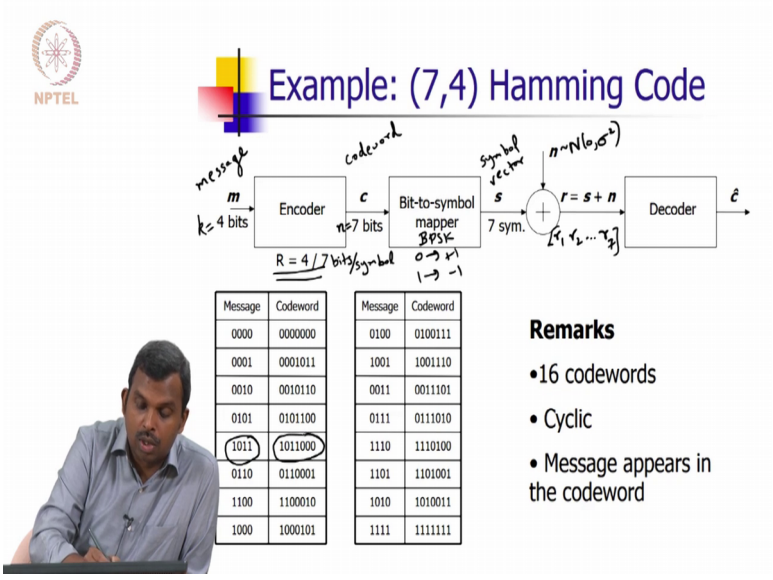
To just show you an example 1 0 1 1, if your message is 1 0 1 1,

(Refer Slide Time 02:11)



you put out a codeword 1 0 1 1 0 0 0,

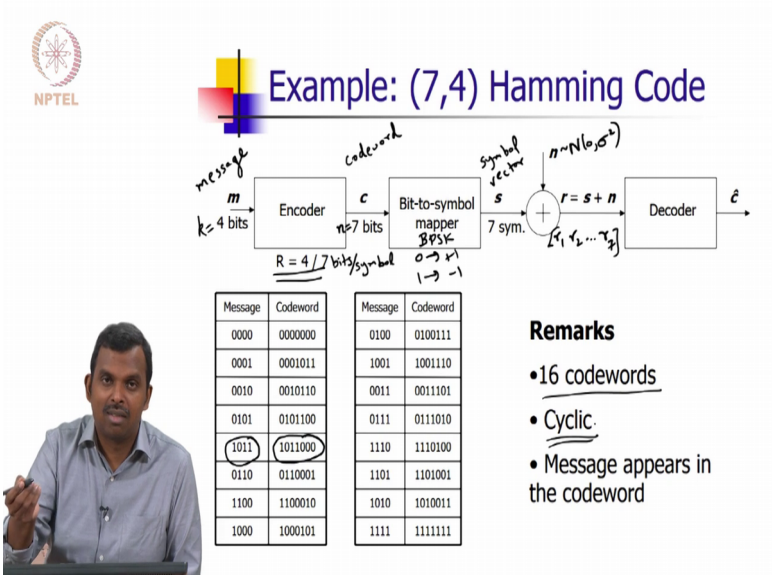
(Refer Slide Time 02:14)



Ok so that is it. That is just the code. I am not giving you a long explanation for where the Hamming code comes from but I am just defining the code for you. This is the code one can do it, Ok.

So it has got 16 codewords. It has the cyclic property which

(Refer Slide Time 02:30)

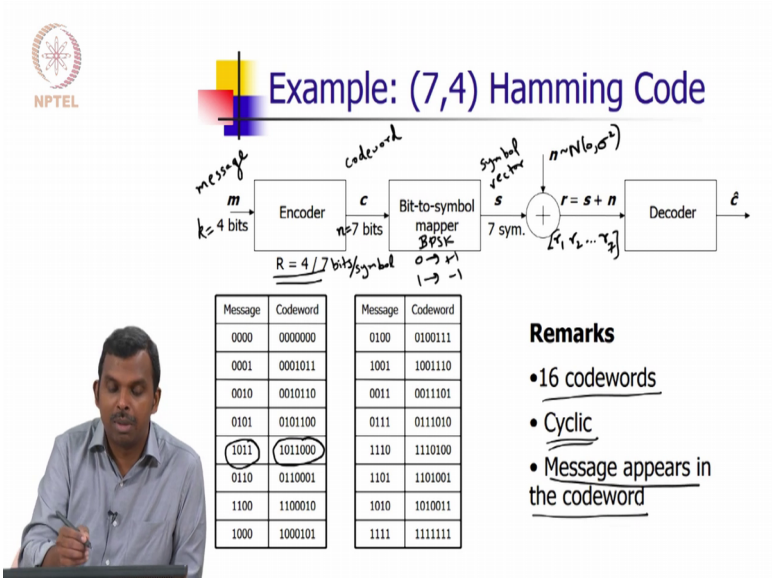


just mentioning for general interest, so take any codeword, you do a cyclic shift you get another valid codeword, Ok. So remember once again when you look at 7-bit vectors the total number of possibilities is 128.

There are 128 7-bit vectors. Out of that 128, these 16 codewords have been selected carefully, Ok. Only 16 have been selected. They have been selected carefully and put into this, put into this Hamming Code, Ok and only these 16 are shown here, Ok so all these properties are important.

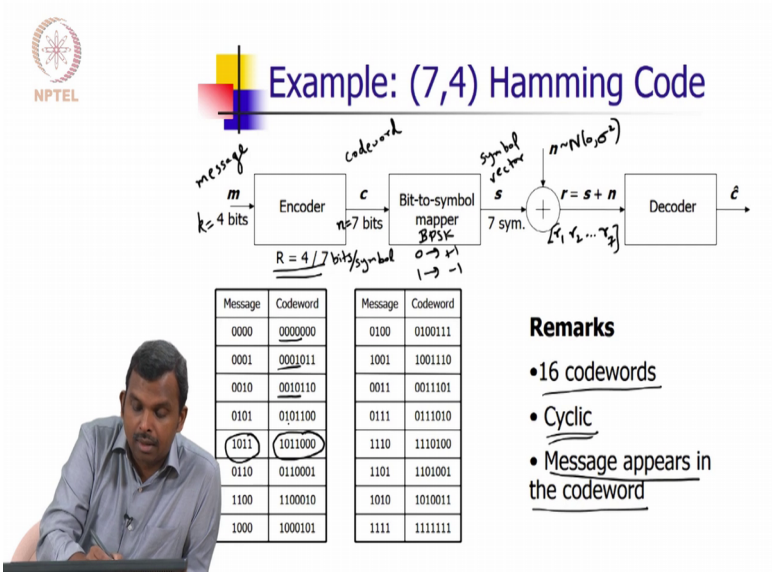
Another important property is message appears in the codeword. What do we mean

(Refer Slide Time 03:06)



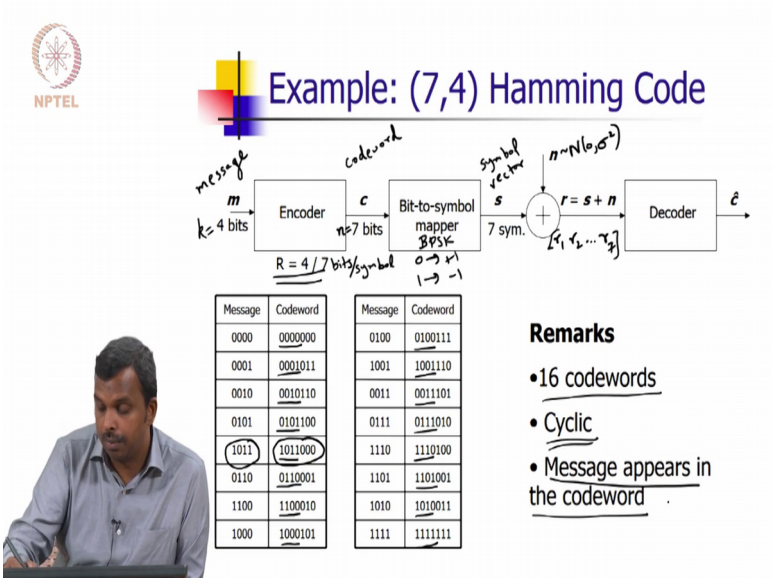
by message appears in the codeword? If you look at the first 4 bits of the codeword, what is that?

(Refer Slide Time 03:12)



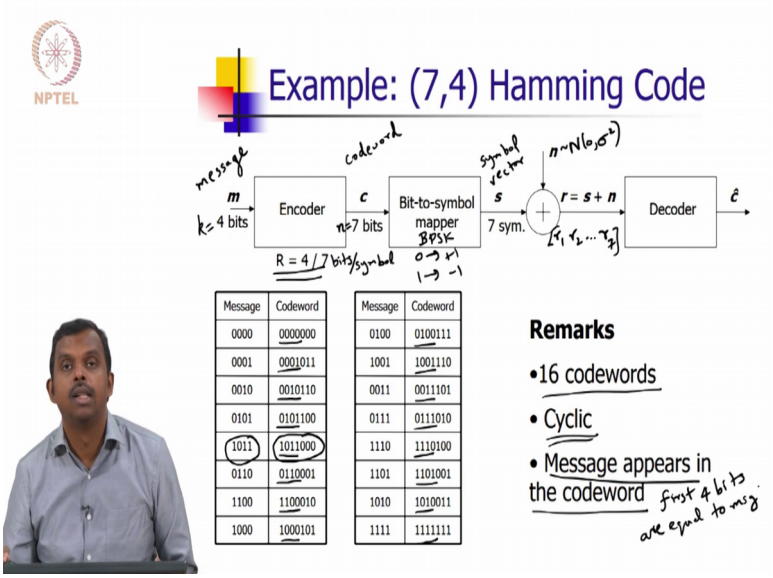
In every codeword the first 4 bits of the codeword agree with the message, Ok. You can see that throughout. The first 4 bits of the codeword equal the message.

(Refer Slide Time 03:26)



Ok, so that

(Refer Slide Time 03:41)

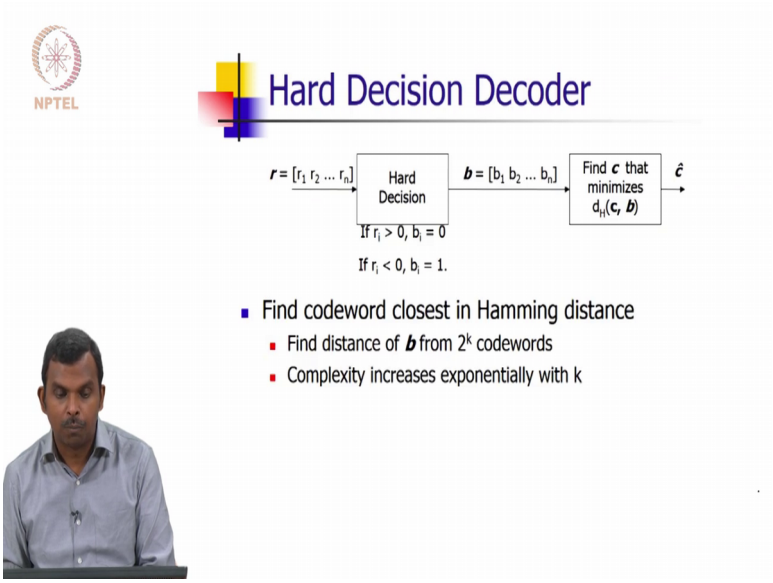


is the, that is one other property. If you do an encoding like that, it is called a systematic encoder. The message appears in the codeword. So these are all nice properties. In fact there are various other interesting properties. This is an example of what is called a linear code.

If you take any two codewords and do a bitwise XOR of the two codewords you will get another valid codeword. Ok, so all these are nice properties but this point we are not interested so much in these properties. Later on we will come back and look at it. But we are primarily interested in seeing how to build encoders and decoders for such a code.

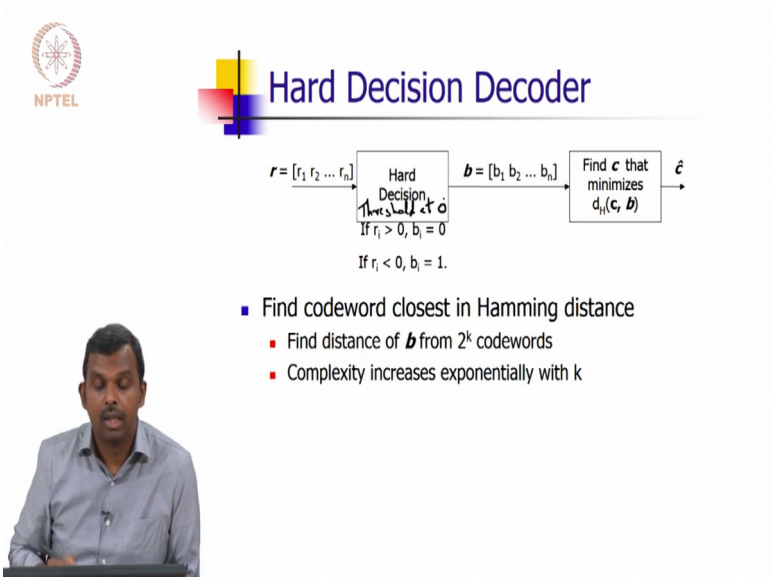
Ok, so now things are a little bit more complex than the repetition case. I cannot put down simple intuitive things and explain it to you, but nevertheless the ideas we picked up in the repetition code will work here as well, Ok. So how does that work?

(Refer Slide Time 04:29)



So first is hard decision decoder, Ok. So you can do a hard decision first. You can do a threshold at 0,

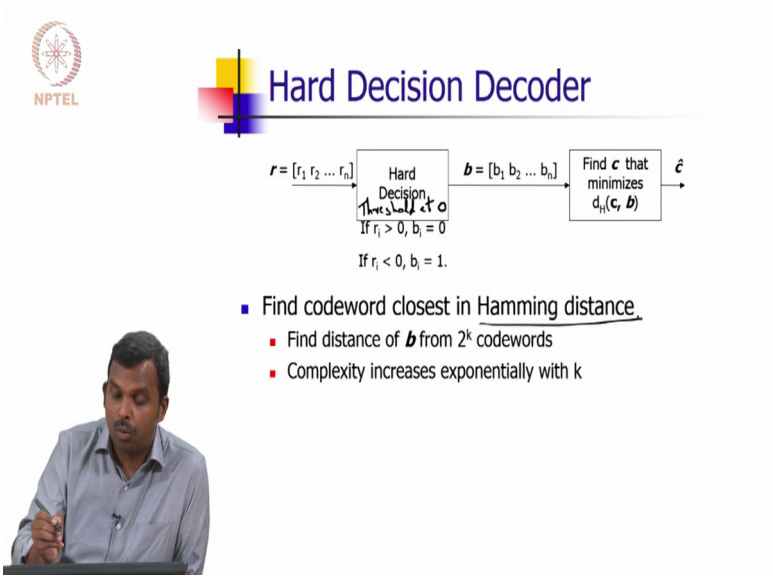
(Refer Slide Time 04:40)



Ok and you get this vector b which is the hard decision vector. And then one can find the codeword which is closest in Hamming distance, Ok.

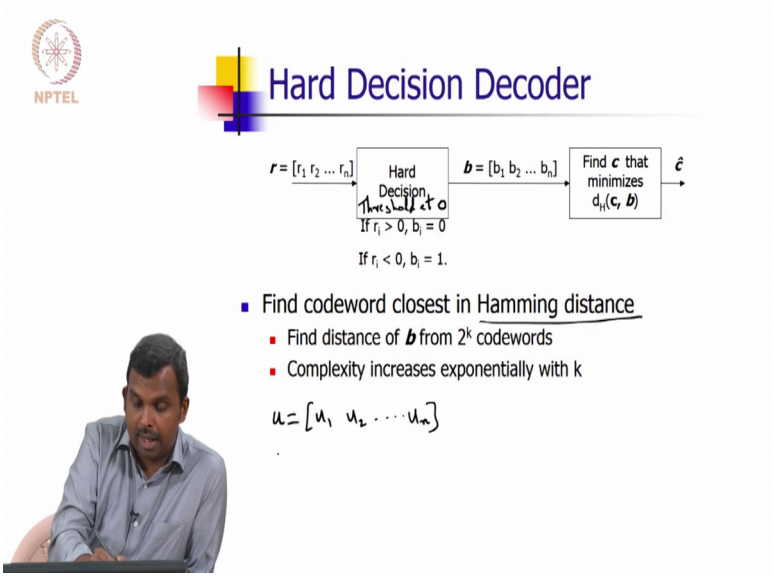
So far I have not described

(Refer Slide Time 04:53)

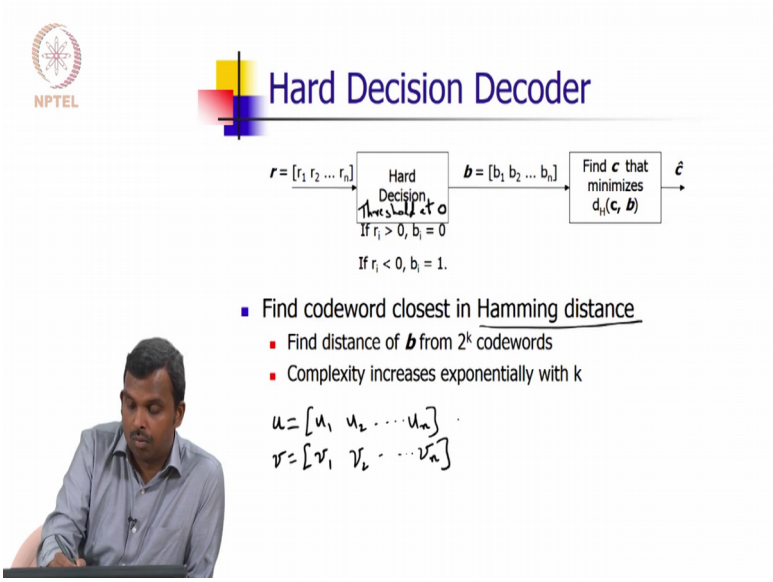


what Hamming distance is. So let me describe what Hamming distance is. If you have 2 vectors u and v which are both binary and length n,

(Refer Slide Time 05:05)

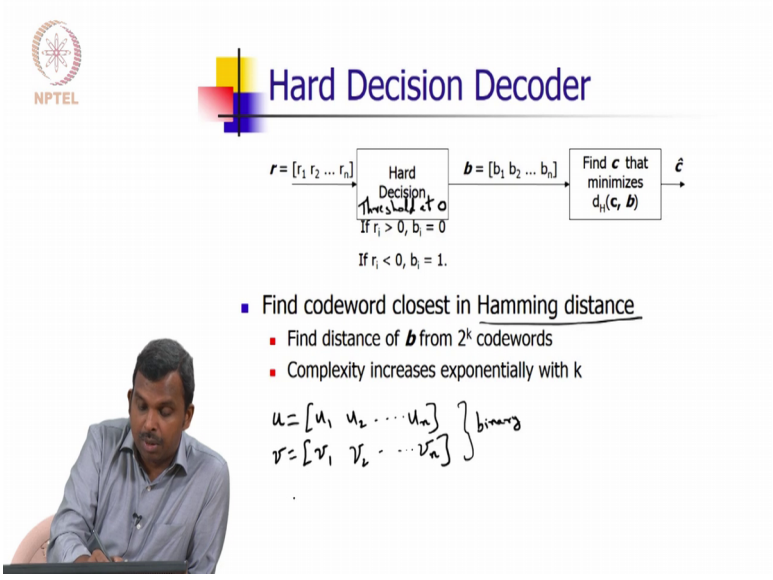


(Refer Slide Time 05:11)



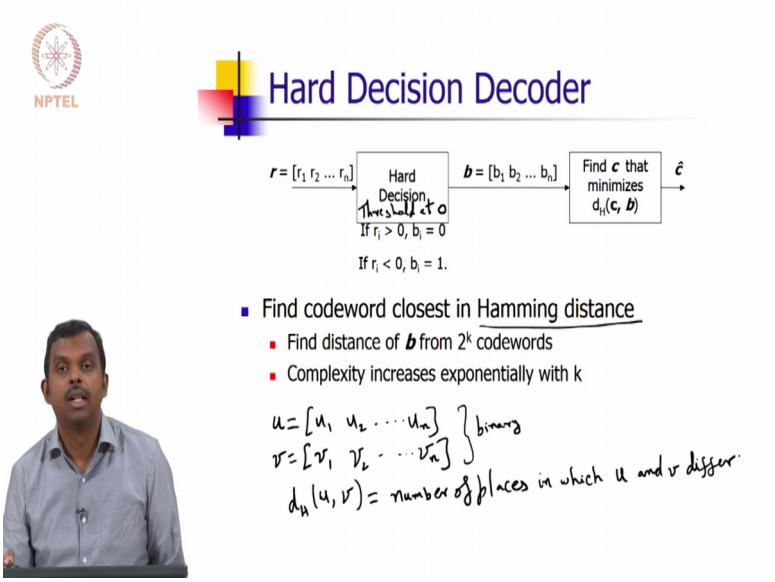
both are binary vectors, Ok the Hamming distance

(Refer Slide Time 05:17)



between these two, d H of u v, Ok is defined as the number of places in which u and v differ. So this is actually a quite a standard definition.

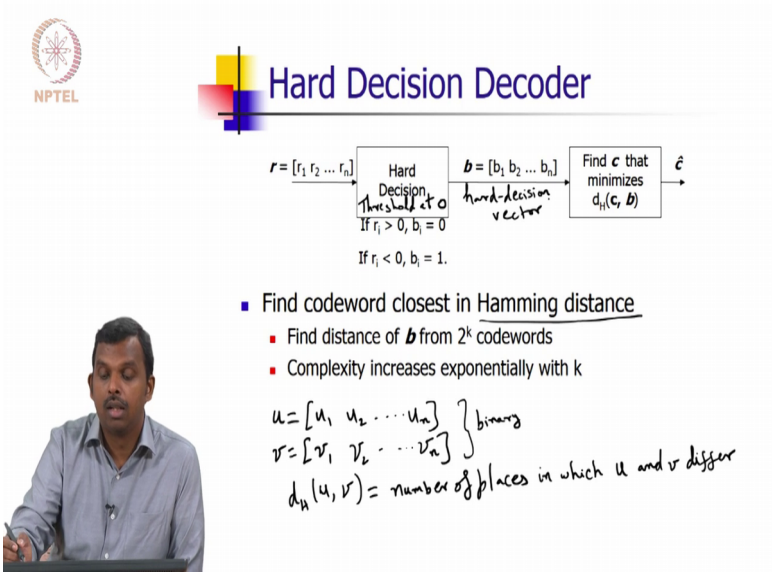
(Refer Slide Time 05:41)



If you haven't seen it this is the definition.

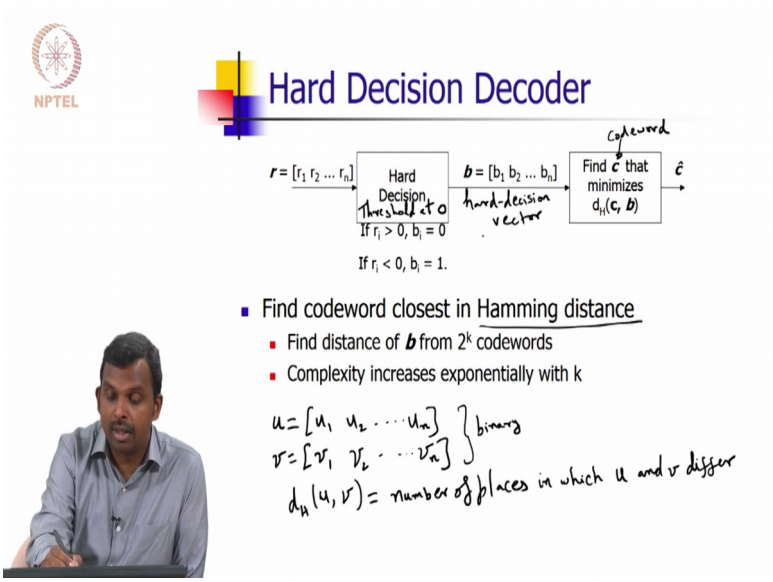
So you have 2 vectors. The number of places in which u and v differ is called the Hamming distance between the two binary vectors, Ok. So once you find the hard decision vector here

(Refer Slide Time 06:00)



Ok what you do is you find the codeword, find c as in the codeword, Ok you have 16 different codewords and this hard decision vector

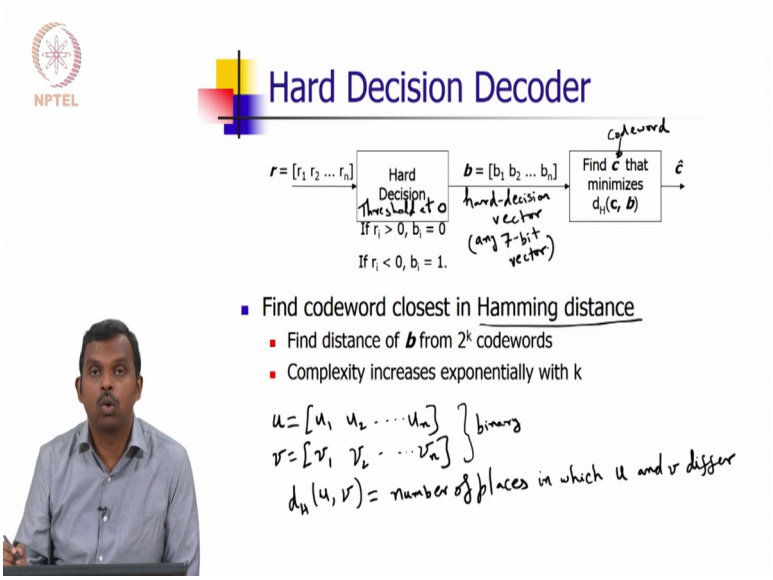
(Refer Slide Time 06:10)



can be any 7-bit vector.

Ok, so in general it won't be a codeword, Ok. So it can be any 7-bit vector

(Refer Slide Time 06:20)



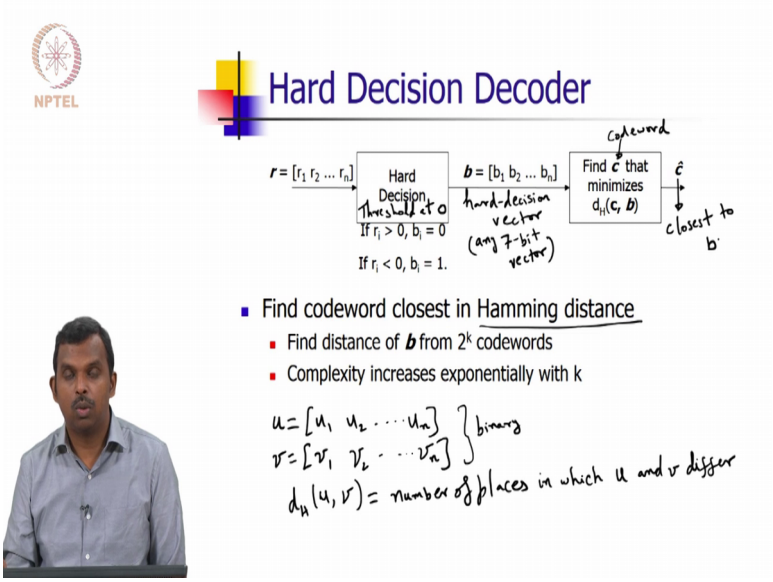
and it may not be a codeword. Your codeword is only 16 chosen things from the 128 possibilities. So what do you do is you go find that codeword among your 16 possibilities which are closest in Hamming distance to, which is, that codeword which is closest in Hamming distance to your hard decision vector, Ok.

So how will you do this? This will involve 16 different computations. You have your vector b. You find the Hamming distance with the first codeword. Then you find the Hamming

distance with the second codeword. Then you find the Hamming distance with the third codeword, so on.

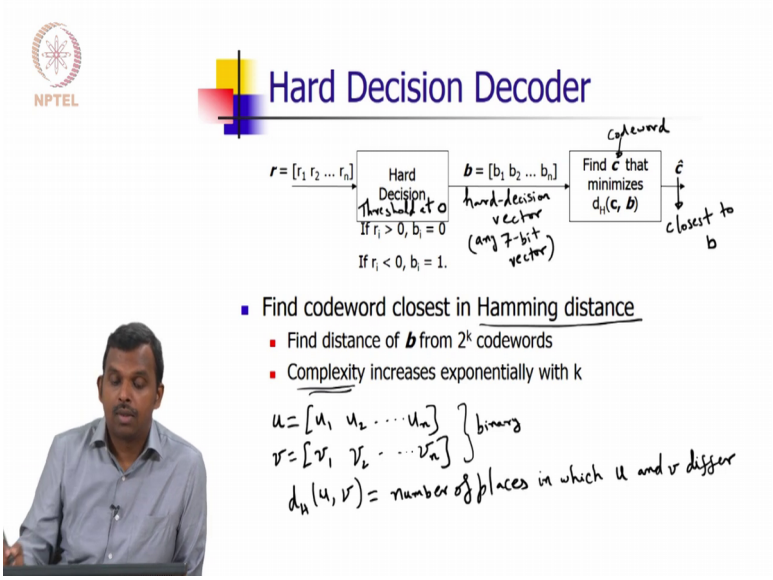
And then you find that guy which has the least Hamming distance and declare that as your c cap. Ok. So this has closest to b, Ok the vector b, alright. So,

(Refer Slide Time 07:10)



so quickly I want to comment on complexity, Ok.

(Refer Slide Time 07:14)



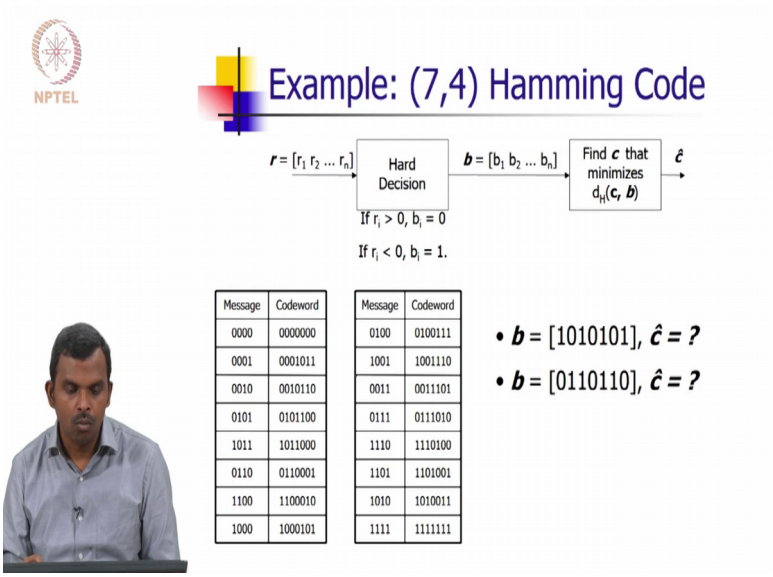
So here we have k equals 4 so 2 power 4 is just 16, 16 different codewords you can this exhaustively. But you see if k were to even be 100, Ok, 2 power 100 is a very large number Ok and you cannot do these kind of calculations very easily, Ok.

Finding the distance one codeword at a time is not going to work if you have large number of codewords and in real communication systems k could be 1000, 10000 and all that, Ok.

So while this decoder is nice and sort of easy to describe from theoretical point of view, it is not going to be very useful in practice and you will have to improve upon this definitely in the complexity front, Ok but for now let keep this decoder going.

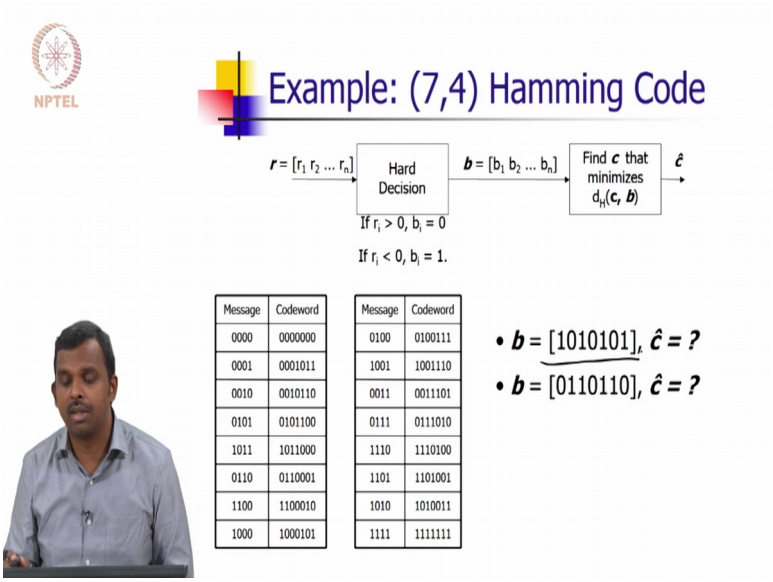
There are cases where this also might be useful; Ok at least for the Hamming code one can implement this. Ok. This is the hard decision decoder. So you make hard decisions threshold at 0 individually, you give up a lot of information but it is at least easier to handle, just bits, you find Hamming distances and then find the codeword which is closest in Hamming distance, Ok. So this is one decoder.

(Refer Slide Time 08:19)



Ok so here is an example just to illustrate how this can be complex, Ok. So here is the same system as before, Hamming code. I have listed out all the codewords here and just for example I am showing you, supposing your hard decision vector is 1 0 1 0 1 0 1

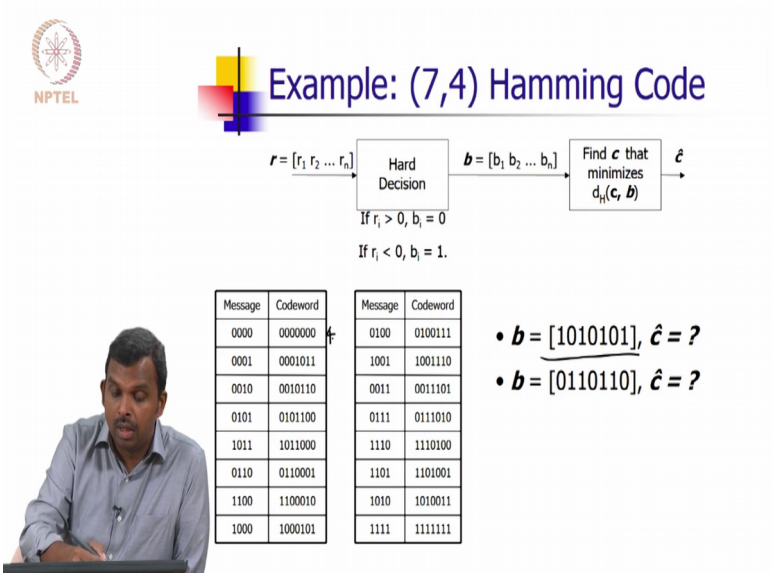
(Refer Slide Time 08:35)



what is going to be your c cap?

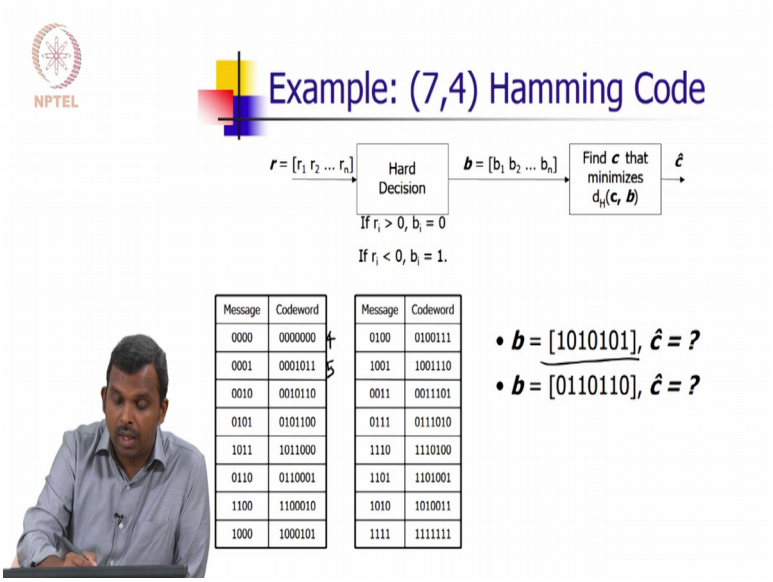
You have to take this 1 0 1 0 1 0 1 and look at each possible codeword and find the distance from each of the codewords and find the codeword which is closest, Ok. So for instance, distance between 1 0 1 0 1 0 1 and 0 0 0, the distance will actually be 4, Ok. You can keep writing it down.

(Refer Slide Time 08:58)



So likewise you have to find the distance. And finding the distance also is not so easy, you know. And it looks like; I mean you have to count very, very carefully. For instance if you look at 0 0 0 1 0 1 1, you see 1, 2, 3, 4, 5, the distance goes to 5, Ok.

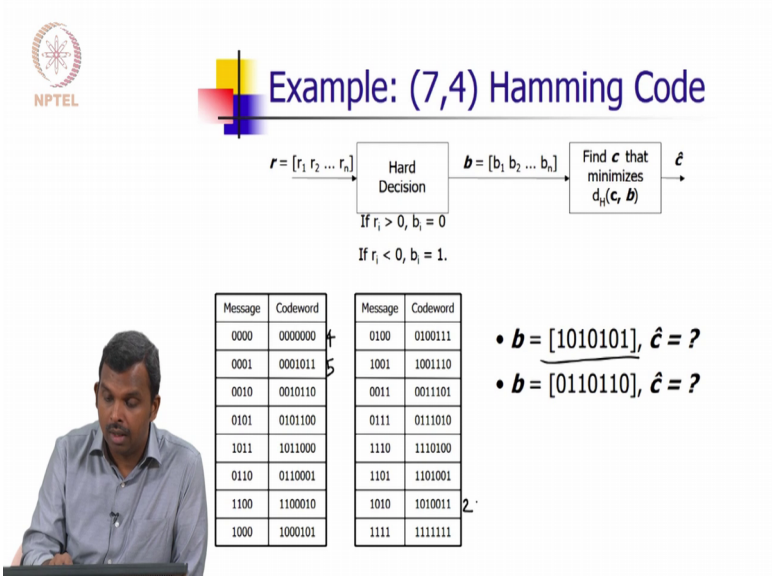
(Refer Slide Time 09:20)



So likewise you have to look at the various possibilities.

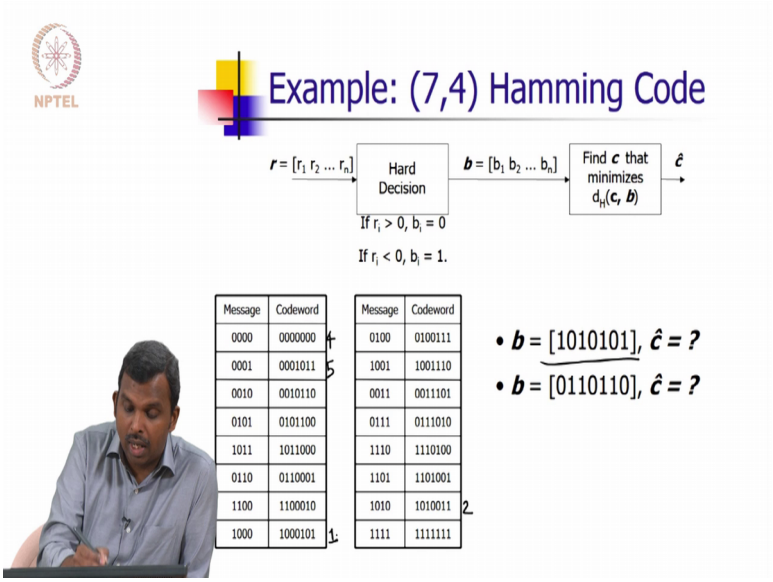
May be we will look at this 1 0 1 0 here. This is at distance 2 away

(Refer Slide Time 09:38)



I think, yeah, this is at distance 2 away. May be I need to look at, so you see it is not trivial, just to find which one is closest is not easy. May be one can look at this one, Ok, for instance, this guy 1 0 0 0 1 0 1. This is at a

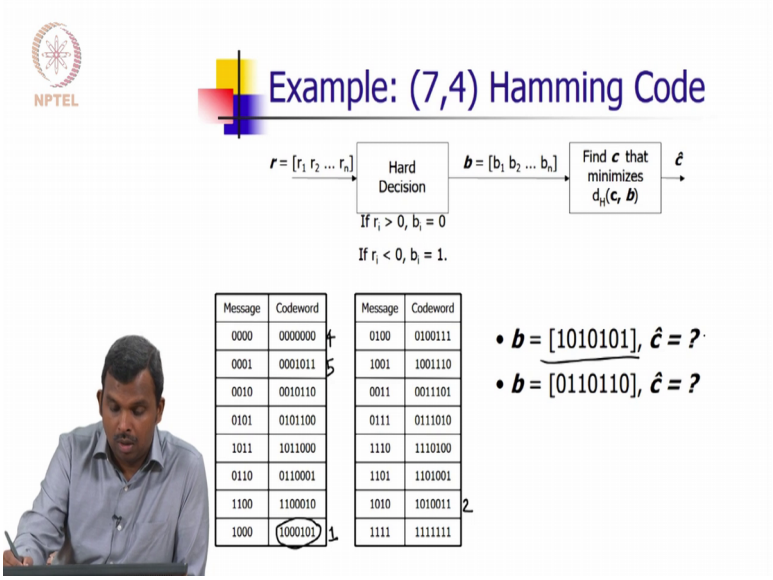
(Refer Slide Time 09:58)



distance 1 away, Ok. So it turns out this is the closest, Ok.

So c cap became

(Refer Slide Time 10:06)

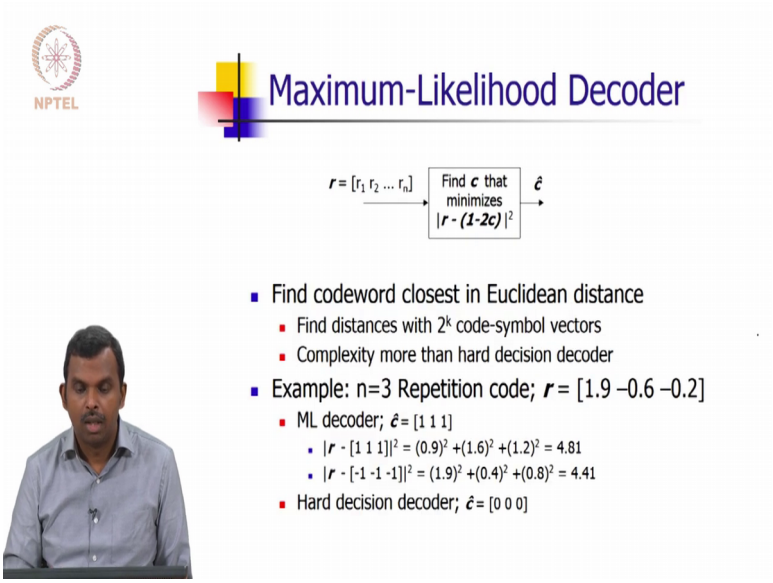


1 0 0 0 1 0 1, Ok, so it was not trivial for me to find it just like that. So of course if you write a computer program it will just brute force go through everything, find the distance and do it. It is not easy, Ok.

So here is another example. You might have to do a little bit of thinking and worry about how to go about this before you can be sure, Ok? So I don't want to do the next example also. So I have done you one, this is how it works, Ok.

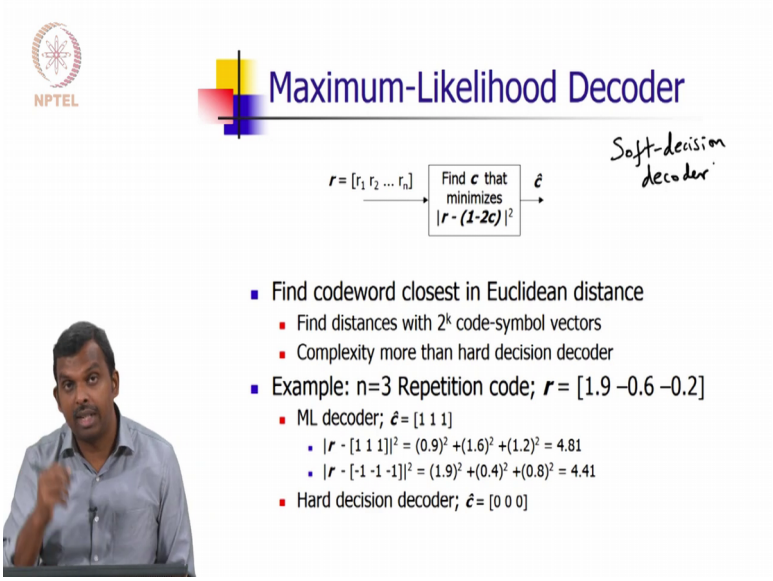
So we will see a small Matlab code for implementing these things just to get familiar with it, Ok. So this is the hard decision decoder. And like I said clearly this is not going to be easy when k becomes large, n becomes large, but nevertheless something to implement.

(Refer Slide Time 10:50)



Ok, the next is maximum likelihood decoder, the soft decision decoder that we spoke about, Ok. So this is the soft decision decoder. So even here, the principle is same as what we did in

(Refer Slide Time 11:04)



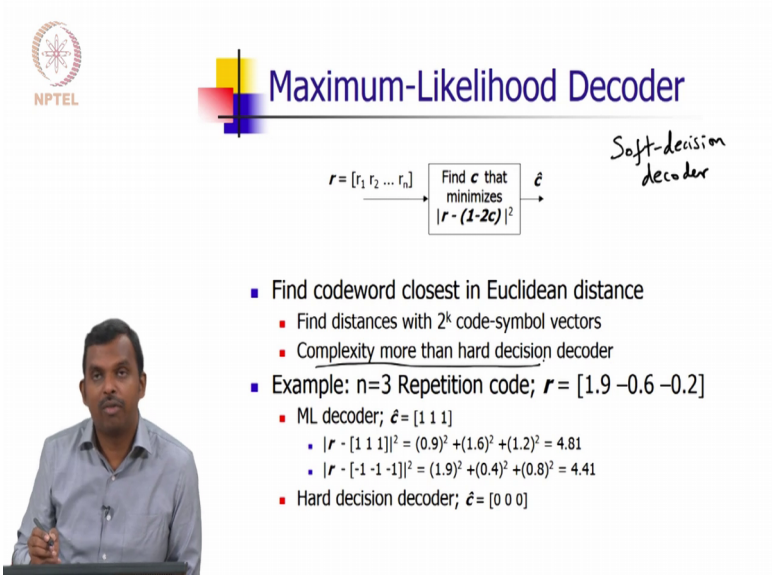
the repetition case, Ok. Instead of making hard decisions and finding distances in the Hamming domain using Hamming distance metric we directly find distances with symbol vectors.

Ok, so you have 16 different codewords. You do B P S K mapping on each of the codewords, you get 16 different symbol vectors. Ok, I have not written down all the 16 different symbol vectors here but you have 16 different symbol vectors, Ok.

And then you find the Euclidean distance or the actual vector distance between the received vector which is real and the symbol vector, Ok and then you find the one that is closest in the Euclidean distance sense, Ok.

So I am not describing this in more detail here. We will write a Matlab code for this. You will see how when we will write it, Ok. So complexity is clearly more than the hard decision decoder. You have

(Refer Slide Time 11:52)



2 power k calculations but each calculation is real number and all that, it is a little bit more than that, Ok.

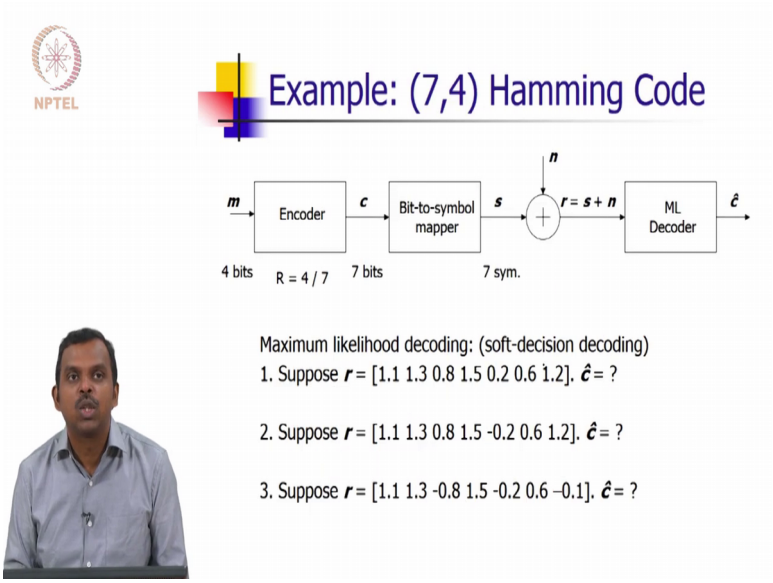
So it turns out we saw before in the repetition code itself. Here is a simple example that is illustrated here to show you how the soft decision decoder can make a different decision from the hard decision decoder, Ok.

So I am not going to go into the details of this example. Please look at it when you have time. The soft decision decoders can make different decisions from the hard decision decoder, Ok.

So here is the simple example for the repetition code. So the soft decision decoder can be strictly better.

So what have I described for the Hamming code, I have described the code itself just as a lookup table. I showed you how the mapping works and all that. And then we discussed the hard decision decoder and we discussed the soft decision decoder.

(Refer Slide Time 12:41)

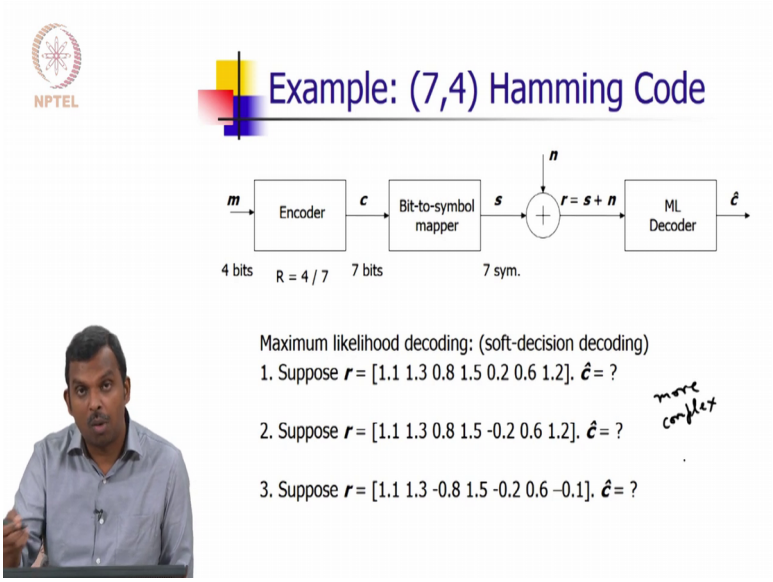


Here are some examples to see how soft decision decoder is going to work for your Hamming code. Remember you might have a vector, received vector like this 1 point 1, 1 point 3, point 8, 1 point 5 so on. So what will be your c cap, Ok?

So it is hard for me to do this calculation. I have to take this received value and go through every single codeword, right and then find the distance etc. There are possibilities to cleverly minimize that work, Ok. I am not going to go in great details but it is lot of work to do, right.

So you have to go 16 different codewords, calculate the distances, the Euclidean distances from each of these things. And then find the one that is least and put out as a codeword, Ok. So definitely more complex, Ok and as k becomes more complex

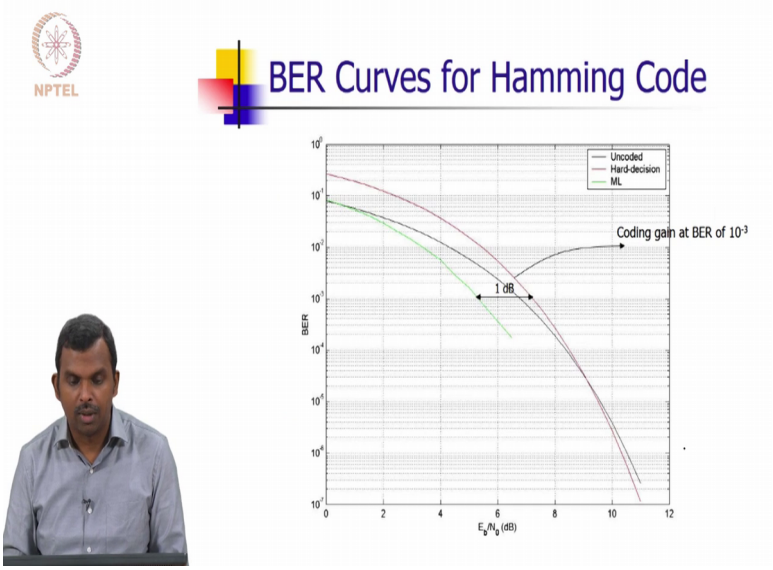
(Refer Slide Time 13:30)



than hard decision it is not going to be easy for me to do this for you.

And as k becomes larger and larger, more or less impossible to implement, Ok, hopefully you understand why this is so.

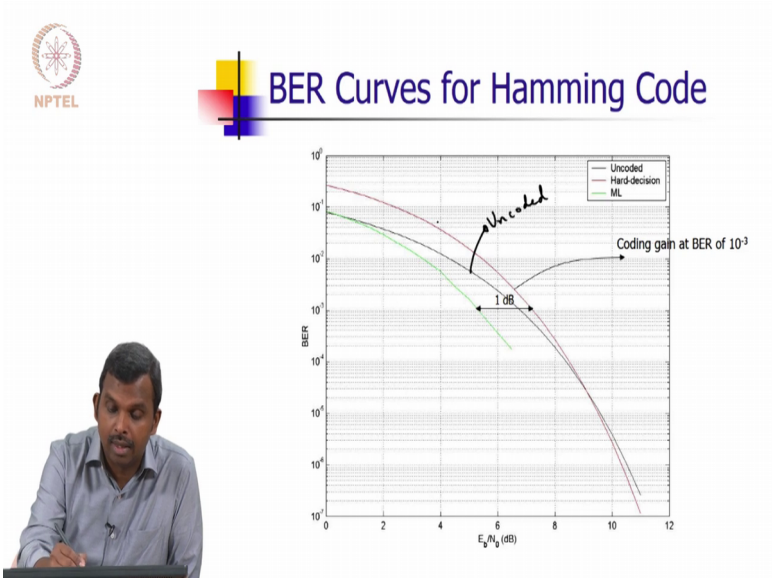
(Refer Slide Time 13:41)



Ok, so if you manage to implement the hard decision and the M L decoder, here is how the picture will look.

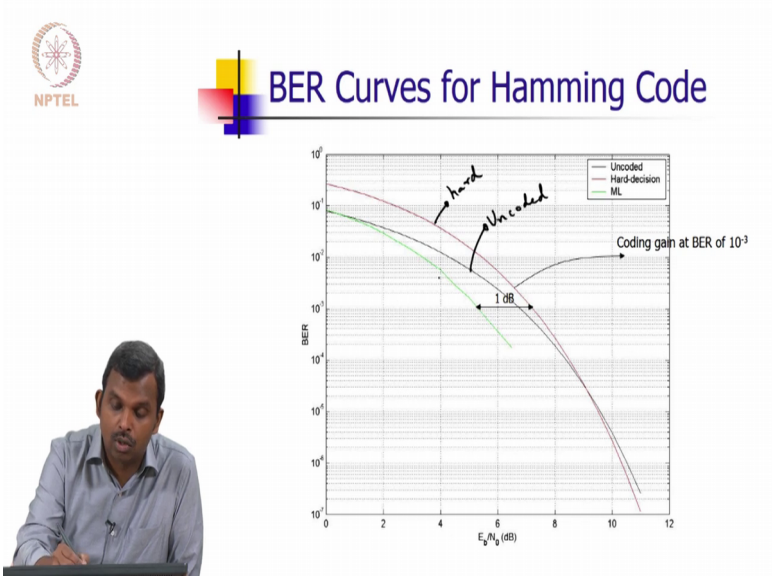
Remember this is uncoded,

(Refer Slide Time 13:52)



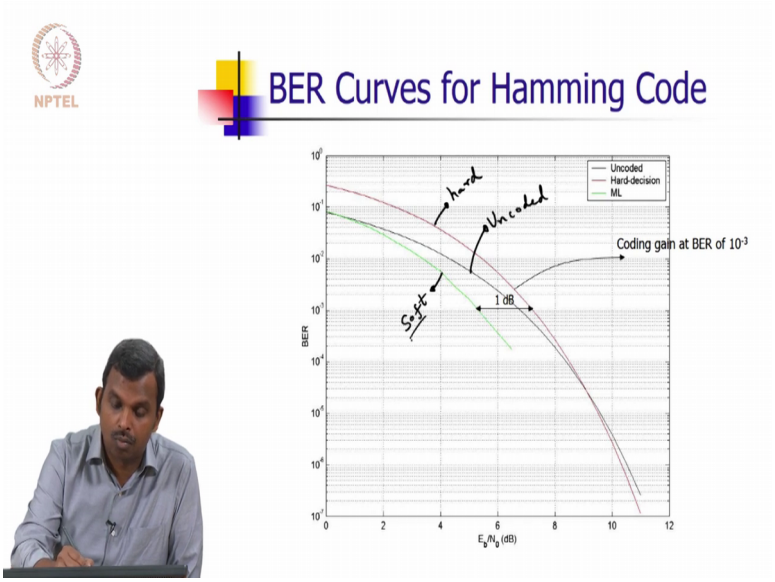
this guy is hard decision; this guy is

(Refer Slide Time 13:56)



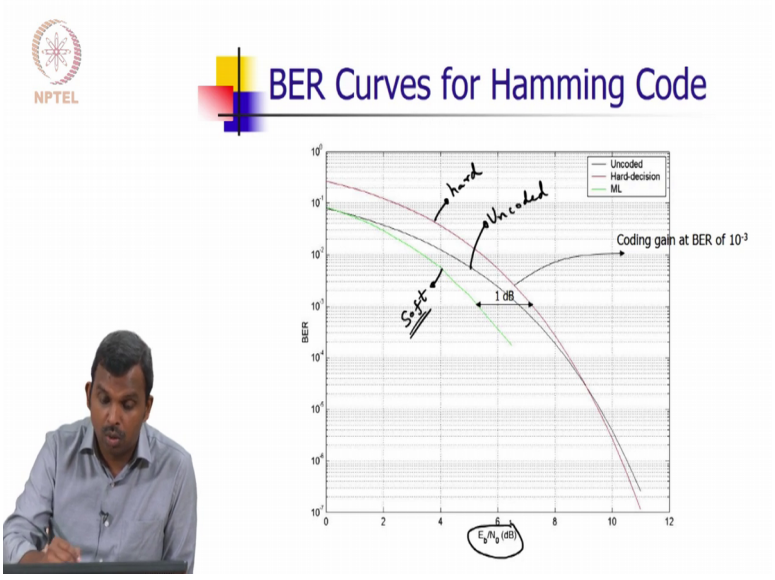
Oops, this guy is soft decision.

(Refer Slide Time 14:03)



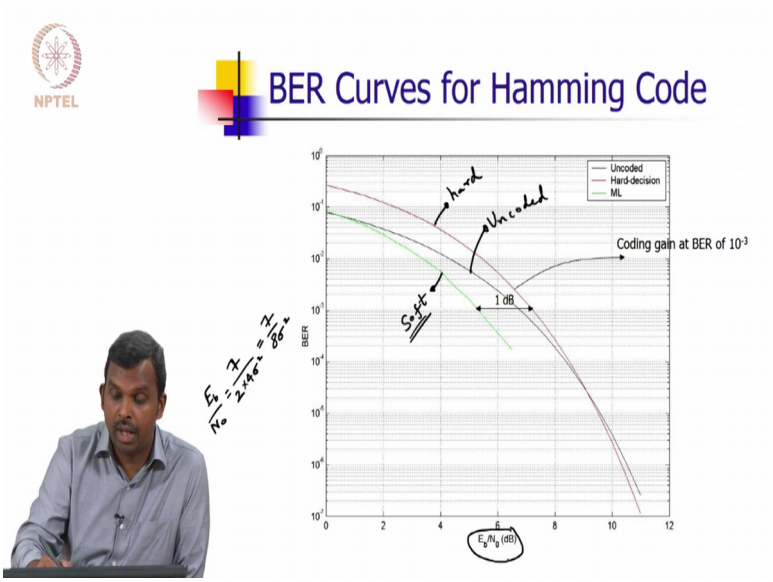
Ok and you can see the gains already. Remember this is still a plot over E b over N naught and what is E b over N naught?

(Refer Slide Time 14:09)



E b over N naught, remember the rate is 4 by 7, so it is 1 by 2 into 4 by 7 or 7 by 4 sigma square, so it is 7 by 8 sigma square, Ok.

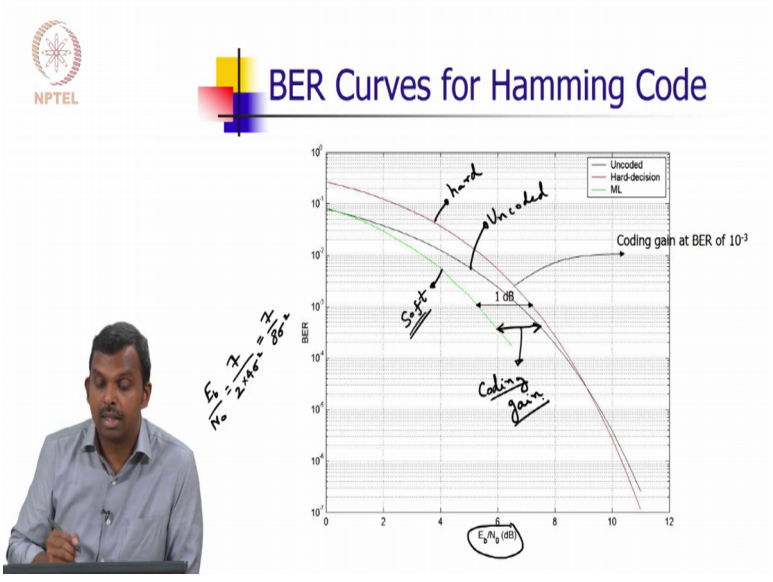
(Refer Slide Time 14:27)



So E b over N naught is that for the coded case; uncoded case will be 1 by 2 sigma square, Ok. So E b over N naught is different in the two cases but nevertheless in spite of penalizing for the rate in E b over N naught the soft decision decoder for instance, Ok gives you a coding gain.

Look at this; this is the coding gain of the Hamming code. Isn't that nice? So simple enough code, constructed well.

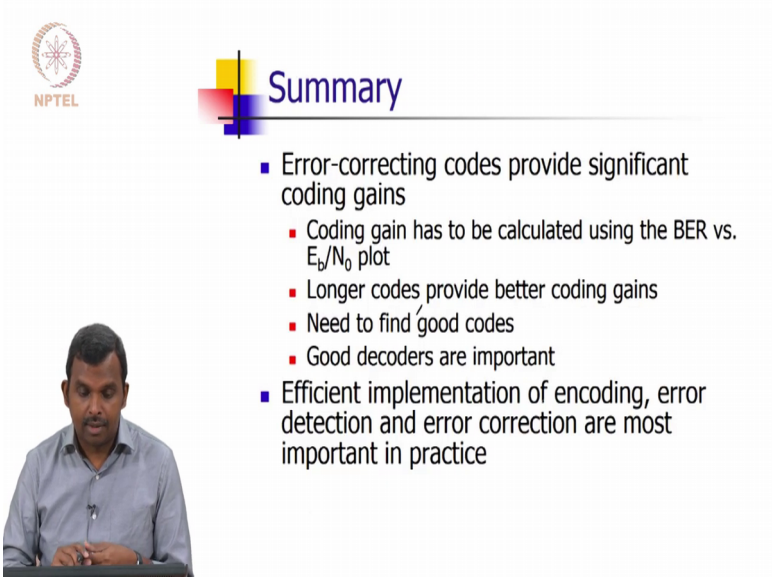
(Refer Slide Time 14:55)



I didn't tell you how the construction came from. It is a bit of a clever construction but if you do that and if you manage to implement the soft decision decoder, the maximum likelihood decoder you get coding gains.

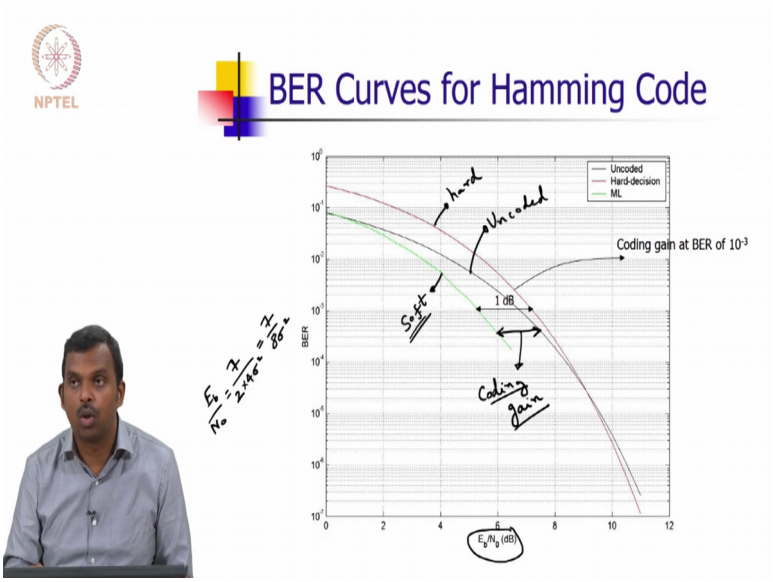
It is not much, it is about a d B or so but still it is good coding gain and the complexity was quite small, Ok? So what we are going to do next

(Refer Slide Time 15:19)



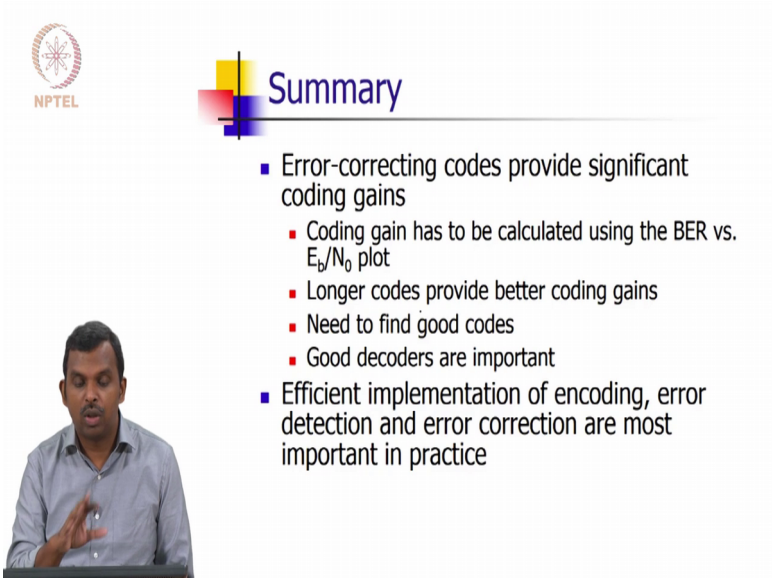
is we will, we will do Matlab simulations for

(Refer Slide Time 15:24)



the Hamming code as well

(Refer Slide Time 15:26)



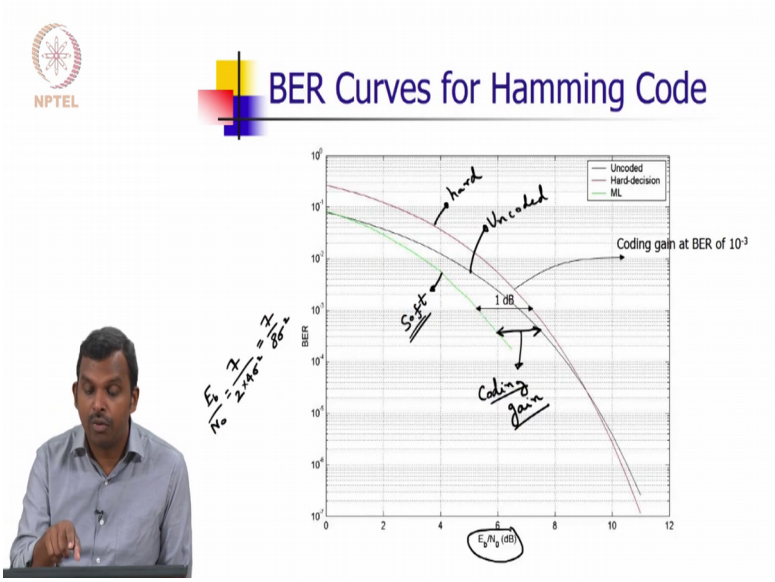
but before that I just want to summarize real quick what we saw.

So error control codes do provide significant coding gains in practice. So we did not see any code already that gives that kind of gains. But you can see the Hamming code which is very small case, very small n is already showing a lot of promise, about a d B of coding gain you are able to obtain, Ok and it turns out longer codes have better coding gains.

Ok so typically you want to have codes which are at least like 100, 200 in length so that you can have coding gains. But we need to find good codes. It is not easy to work with obvious repetition sort of strategies. You have to design the code smartly and modern codes like L D P C and polar codes we will study, a quite sophisticated in the way they are constructed.

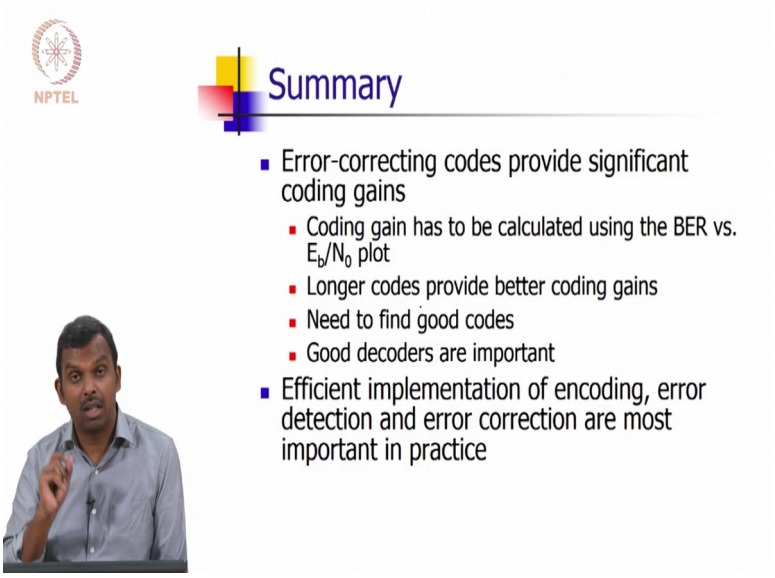
Not only that, we saw that good decoders are important so let us go back here.

(Refer Slide Time 16:09)



If you do a hard decision decoder you do not see coding gain at all, right so there is still a loss for, for up to some point. After that it crosses over mildly. But soft decision decoders are good, Ok.

(Refer Slide Time 16:21)



Particularly soft decision decoders are very, very important and you have to design them well.

But the problem is implementation, right. So if you have a good decoder, maximum likelihood decoding is optimal but implementing it is very difficult. So you want to have good decoders as well.

So here is the summary. The next thing we will see is Matlab implementation for 7 4 Hamming code. I will do that in the next lecture, thank you.