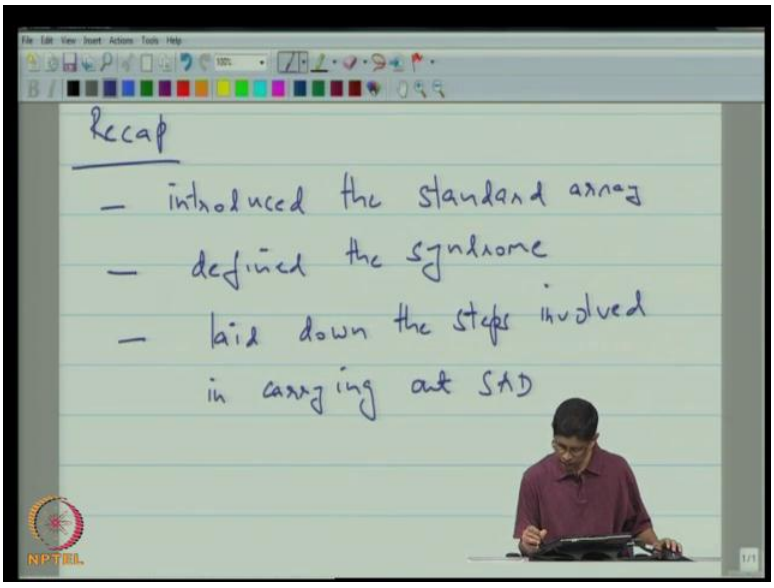
Error Correcting Codes
Prof. Dr. P. Vijay Kumar
Electrical Communication Engineering
Indian Institute of Science, Bangalore

Lecture No. # 15
Performance analyses of the SAD

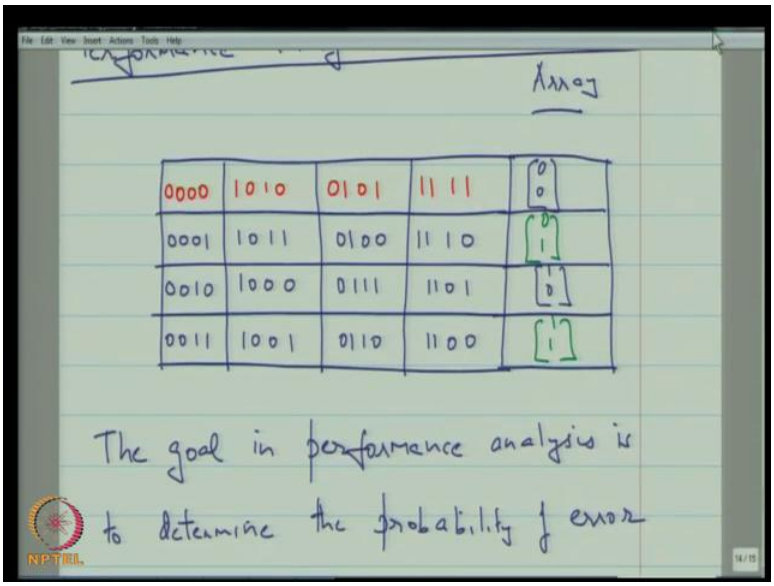
Good afternoon. This will now we are 15-th lecture. And what plans are for this lecture is to continue or discussion on the standard array decoder. And the last lecture, we were focused on using the standard array decoder or the syndrome decoder, for the purposes of decoding. Our purpose this time however is different. Here, we are interested actually trying to see, how does, what is the performance in terms of probability of error of this standard array decoder? So, we like to know for example, what is the codeword error probability etcetera.

(Refer Slide Time: 00:58)



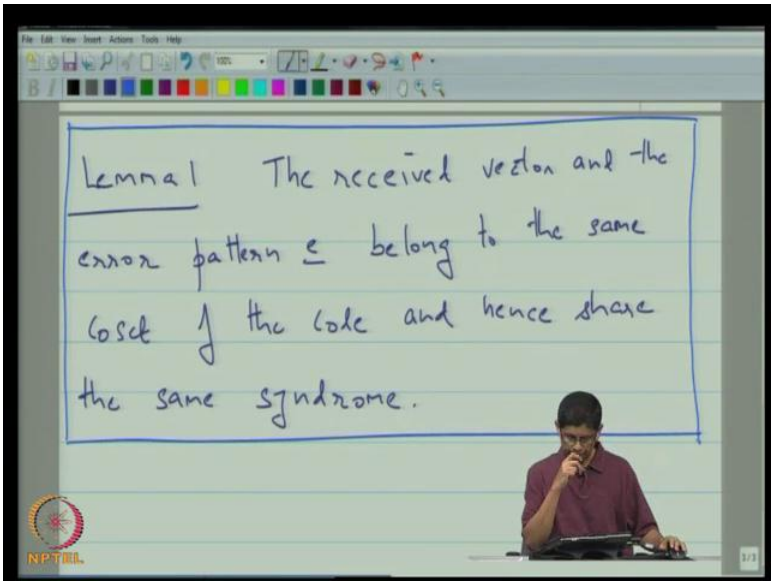
So, I am going to give the title of this lecture as performance analysis of the standard array decoder. Now so as a recap, we are introduced the standard array, or defined the syndrome, and set up the procedure for standard array decoder. So, we laid down the steps involved in carrying out standard array decoder.

(Refer Slide Time: 2:32)



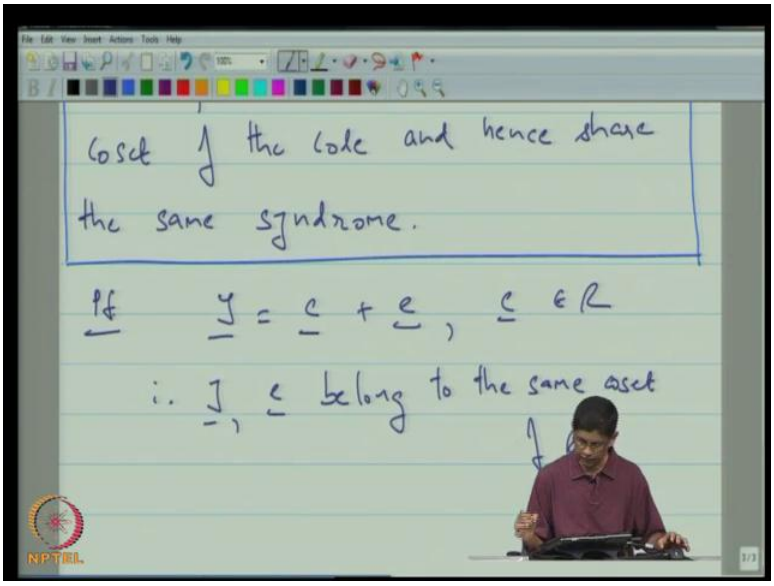
Now, towards end of the last lecture and so again let me see if I can actually copy that, so what I would like to do is copy this page here. And let us paste that here, so we want to use this standard array for carrying out performance analysis, and the goal in performance analysis is to determine the probability of error. They are different probabilities of error that come into play, and will begin with their focusing on the probability of codeword error.

(Refer Slide Time: 3:26)



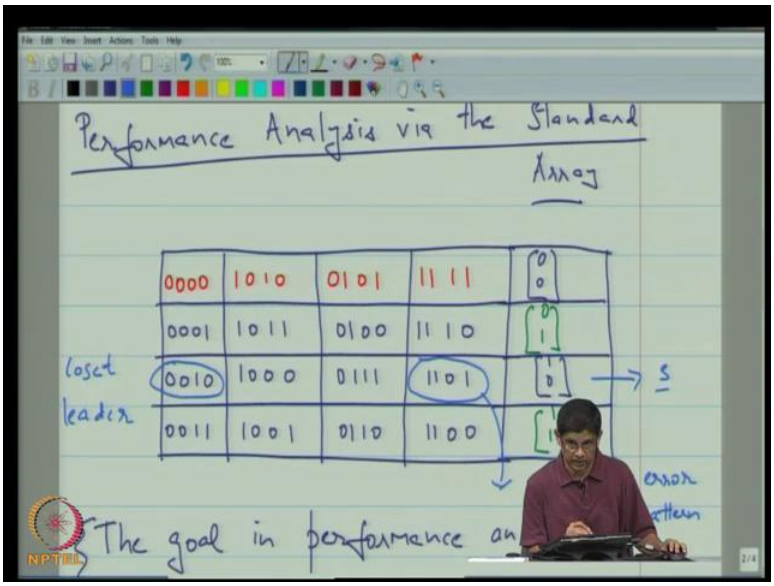
Now the first thing is, the first point to note is a Lemma 1, the received vector and the error pattern e belong to the same coset of the code; and hence share the same excuse me, the same syndrome. The proof is obvious I just thought it to nice to draw attention to it.

(Refer Slide Time: 04:45)



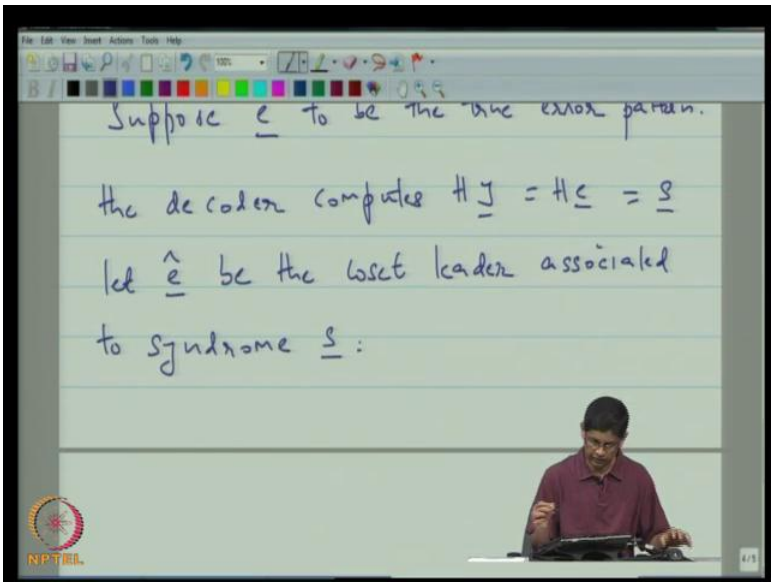
The proof is, because y is c plus e. It is clear that they that y and e belong to the same coset; therefore y, e belong to the same coset of the code. Now the part about sharing the syndrome, so H times y is equal to H time c plus e, which is H times e, and hence y and e share, the share the same syndrome. Now, let us go back to the standard array, and say and now let us view the elements of the standard array. Now earlier we viewing them as receive vectors, and we were saying, supposing this was the received vector what could happen on this one. But this time, we are going to change have you we can look at the same table, but now we are going to treat each entry in the table as if it was n error pattern.

(Refer Slide Time: 06:26)



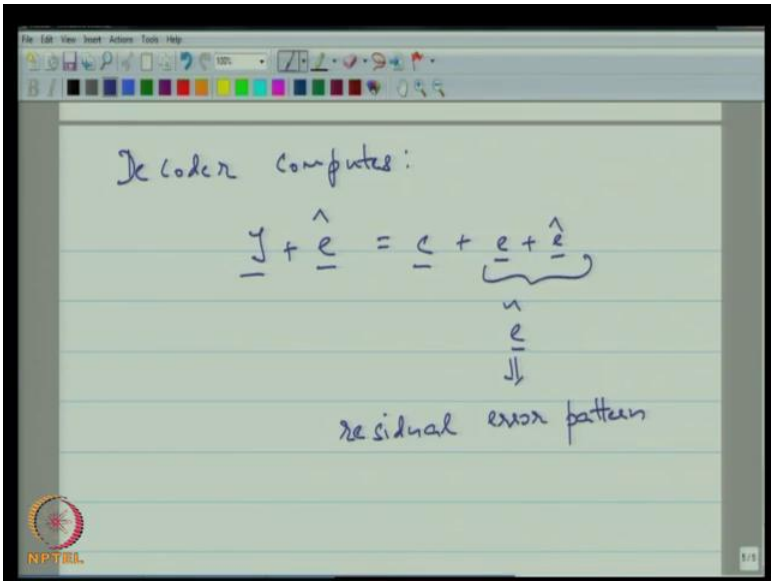
So, for example supposing going down to this table here. Let us say the error pattern for instance was 0111, and let me circle, let me take a different 1-s. I believe a pick that; let say that the error pattern is 1101. I am going to in circle that in blue. So let us say that was you have true error pattern, which is the true error pattern or vector. Now, what is the receiver a going to do? It is going to compute this syndrome of the receive vector. But the syndrome of the receive vector is the same as the syndrome of the error vector which is this. So, in this case the syndrome let us going to be compute is this, and once it is giving a syndrome it is going to use table look up, and it is going to jump to the coset leader. So it is going to jump to this coset leader. Now given this coset leader, and given this received vector the algorithm says that, you take this coset leader and added to the received vector. So, let us see what that does? So, this we finished our theorem.

(Refer Slide Time: 08:02)



Supposing, suppose e to be the true error pattern. So, we compute the decoder computes, the decoder the decoder computes H y, which is the same as H e and that is your syndrome. And then next let e hat be the coset leader associated to syndrome s.

(Refer Slide Time: 09:11)



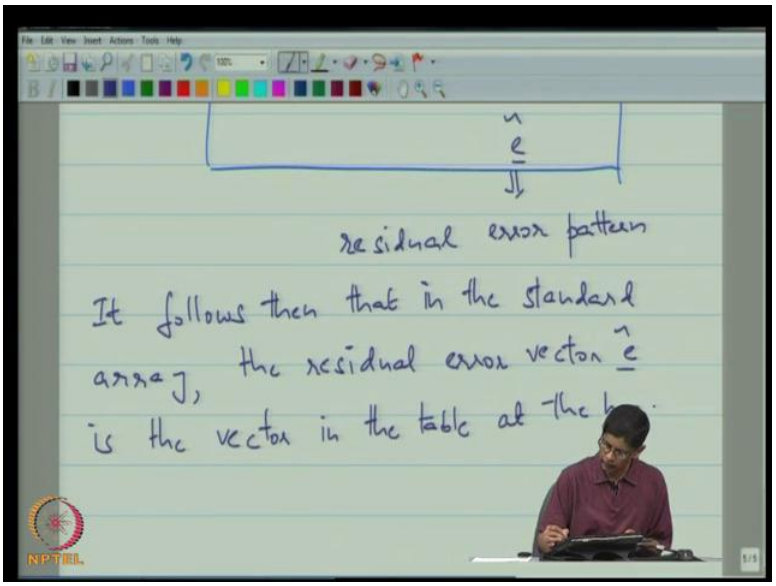
Then, the decoder computes the decoder comes computes, y plus e hat, which is c plus e plus e hat. So, let us called is e tilled. I am going to called is the residual error pattern. So, let this

actually saying is that the combination of the channel and the decoder result in a residual error pattern, which is really the sum of the true error pattern and the coset leader, because the coset leader is really are guessed the error pattern. If it happens that you guessed correctly then the error vector and the coset leader will cancel each other out. So, that your residual error vector is 0. But that may not happen always. So, in which case you would have a non zero residual error pattern; now what I want to identify in the standard array the residual error pattern.

So, here getting back to an example I said let this we are true error pattern, and then here is the coset leader and we also agreed that we would actually called this coset leader we would called this e hat, and now what would the decoder do? It would actually compute y plus e hat when the process create a residual error pattern which is the sum of e hat plus e. That means a residual error pattern in this particular case is the sum of this plus this. The way this table is set up that sum exactly here and the head of this. So, that means that the residual error pattern is in fact nothing but this. So, this is e tilled. And now so let say that this is the let us point out the this is the residual error vector, which means the residual after decoding.

So, now you have a very nice way of thinking about, what happens when you imply standard array decoding? The error pattern can be any entry in the table, and the error residual error vector simply, the vector that is at the head of the column corresponding to the true error pattern. So, let us go down to the table again, and will look at some examples. So, if an example 1101 was in fact the true error pattern the residual error vector is 1111, if it was 1000 the residual error vector is 1010.

(Refer Slide Time: 12:26)

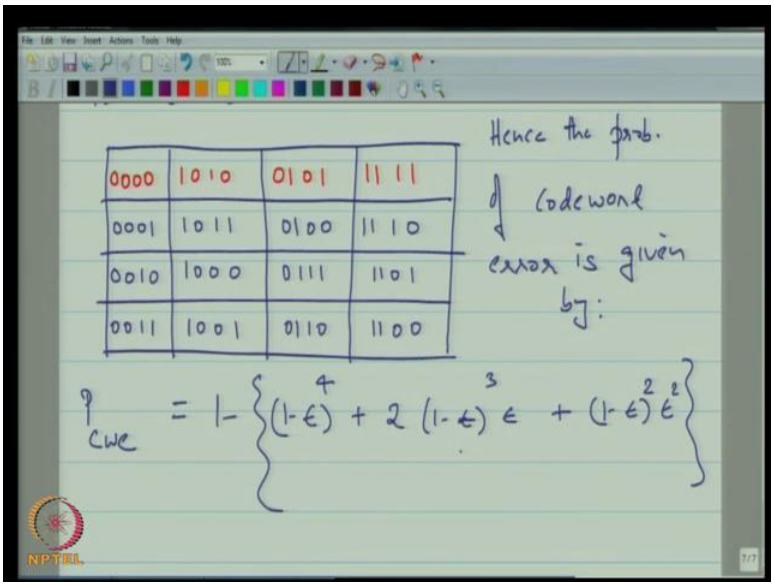


Let us make a note of that. It follows then that in the standard array, the residual error vector e tilled is the vector in the table at the head of the column at the head of the column to which e belongs. And just to cement this concept I thought will might draw a picture like this. The idea is that, you have the codeword and you have the true error vector e. This is followed by the syndrome computer by the syndrome computation. So, you compute H times y, and what you get is s then, you have table look up. So, standard array table look up, and what comes out of this is e hat which is the coset leader. And then what you do is you take the output of this and this, and you add the two, you add the two and now this is your c hat you are decoder codeword.

But what is interesting is that sincere estimated the error pattern. You can also write, you can also identify that down here. In some sense the combination of channel and the decoder create the residual error pattern. So, just a way of thinking about this; so do not back to this table now. So, in terms of performance error supposing I have to ask you, which error pattern is of the code able to correct? We look down this table and side look, if in fact I am able to correct in other pattern then, the residual error vector message could be 0. So, the only error patterns that I can in fact correct are those for which the residual 0. So, I look at this table, and I say where but I know that the residual error vectors always the vector the head of the column.

So, look at the table. Let us go down to the table please, and then you see that in fact the only place for the residual error vector 0 is in fact at the head of the first column; the only error patterns that the code is able to correct a precisely the coset leaders. These are precisely there are so let us put that down. It follows it follows that the only error patterns that the code is able to correct are precisely the error patterns corresponding to the coset leaders. Now all right, so then you can immediately write down also the probability of decoding error, because incorrect error precisely when the error pattern is not one of the coset leaders. So, let us do that. Let me, if I can select a section of this table. We does not want to copy the whole the table and will repeated.

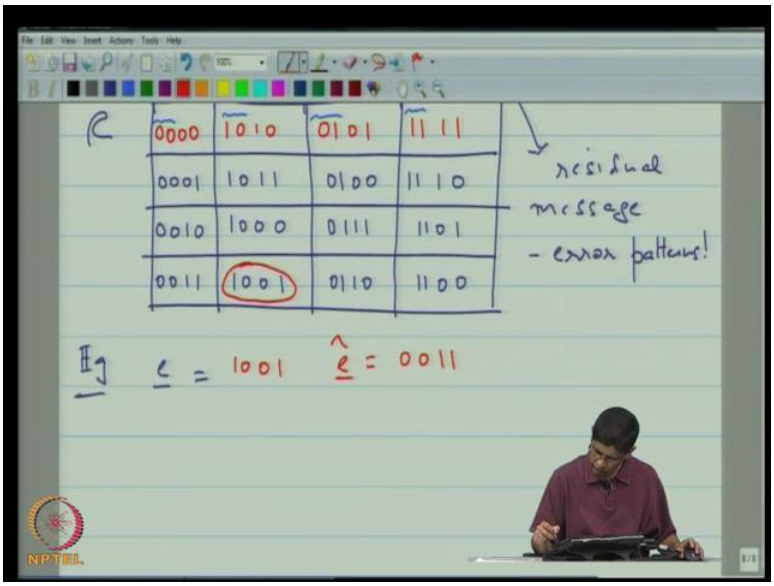
(Refer Slide Time: 18:20)



Let see if I can copy here, very good. So, able to copy a section of the table and I guess we do not need these lines. Hence, the probability of codeword error is given by, I am going to write cwe to denote the probability of codeword error is, 1 minus the probability of no error, and no error happens precisely even the error pattern is one of the coset leaders. I can write this as 1 minus epsilon to the 4 plus 1 minus epsilon cubed epsilon. Actually let me just erase that like it was correct but I really should write plus 2 into 1 minus epsilon cubed into epsilon plus, 1 minus square into epsilon square. In other words but I am looking at I am looking at these coset leaders here and the probability of this coset leader, being an error pattern is 1 minus epsilon to the 4 in the binary symmetric channels.

Because we assume that the error is independent from bit to bit or from symbol to symbol. Again here, if we look at the coset leaders, then the second and third coset leaders both of hamming weight 1. So, that the corresponding probabilities are given by this and since, the two of them multiply by 2 and this is the last term. So, this is the probability of bit error and of course the this whole thing is small, because this thing is roughly close to 1 when epsilon is small. So, now had the probability of codeword error, but which in to sign that you can also use the you can also it to actually determine other error probability that is well. So, let us do that and again I am going to select this thing here.

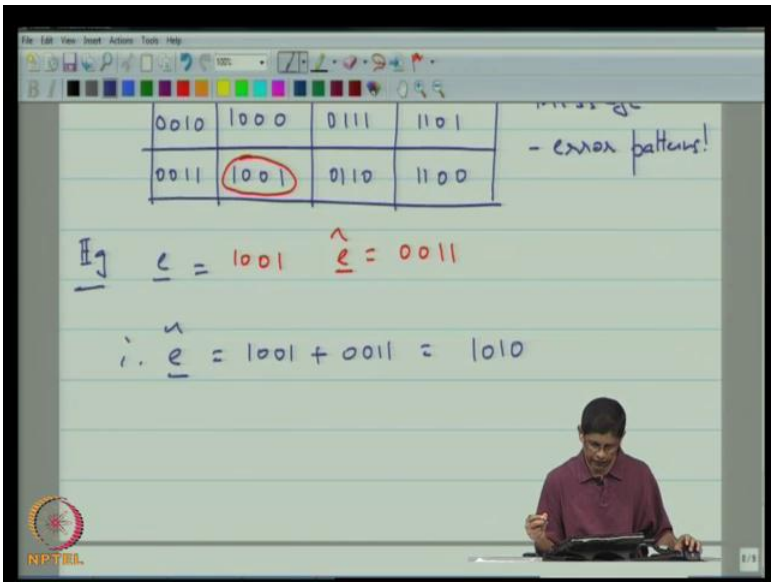
(Refer Slide Time: 21:25)



I am going to select the table again, let us copy it and put it down. Now this time what I want to do is with the same standard array, let us say that I am interested in finding out what is the probability that the first message bit is in error? I just going to do this an example but from this become clear that we will find out any kind of message bit error that you are interested. Now recall so these are the codewords. So, these are your codewords in the code. So, this is your code. I am going to write down a form in other row on path, which has which contains the underline message symbols. So, these are, now we agreed that m transpose G is c transpose is the equation there actually describes how the codeword is generated.

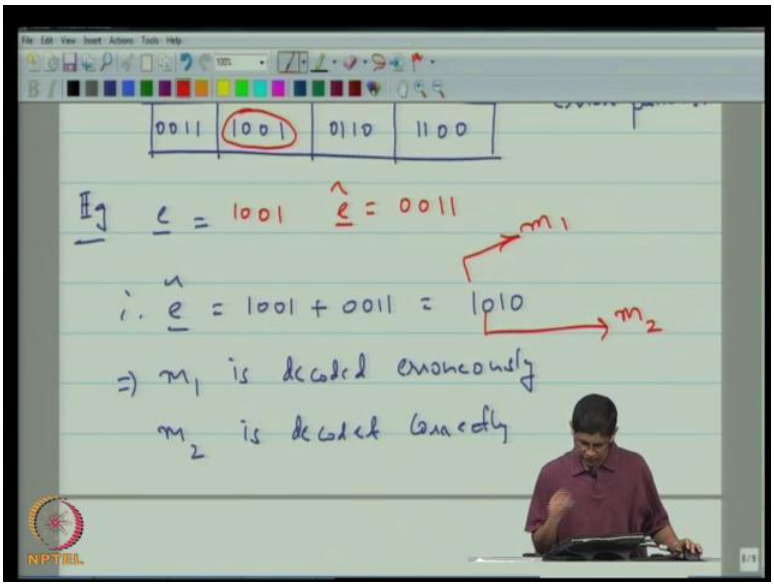
So, what I want to do is I want to put down in this table the corresponding message vectors transpose. But this being a systematic code the message symbols are explicitly present in the code as the first two symbols. So for example the message symbols here may be let me use blue are 00 the first two symbols 10 from here 01 and 11. If the residual error vector is always the codeword at the top of the column, the residual error pattern as far as the message symbols is concerned is are precisely these. So, you can also view these as residual message error patterns. So, what that means is that for example for example, if e let say e was 1001 then, e hat would be 0011.

(Refer Slide Time: 24:10)



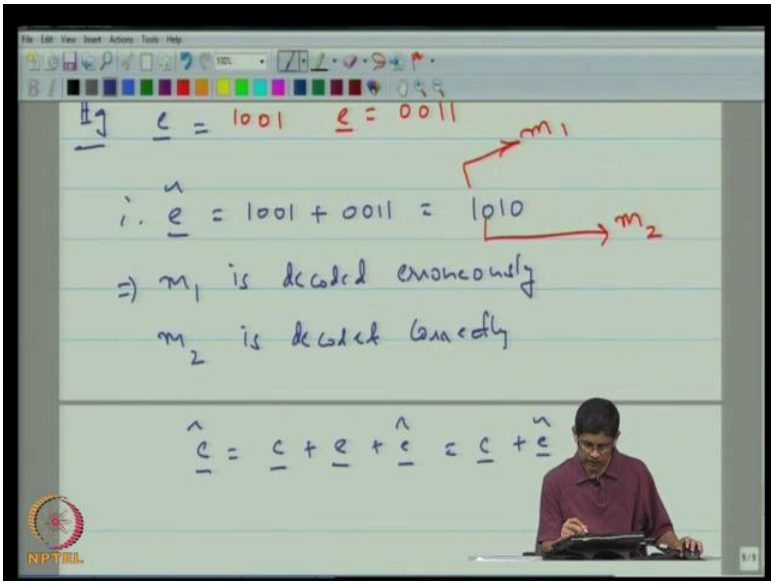
Therefore e tilled would be 1001 plus 0011 which would be 1010. We will find other pattern here this is now as discussed this is your residual error vector. But now we interested in what is the residual error in so far as the message with circumstance.

(Refer Slide Time: 24:40)



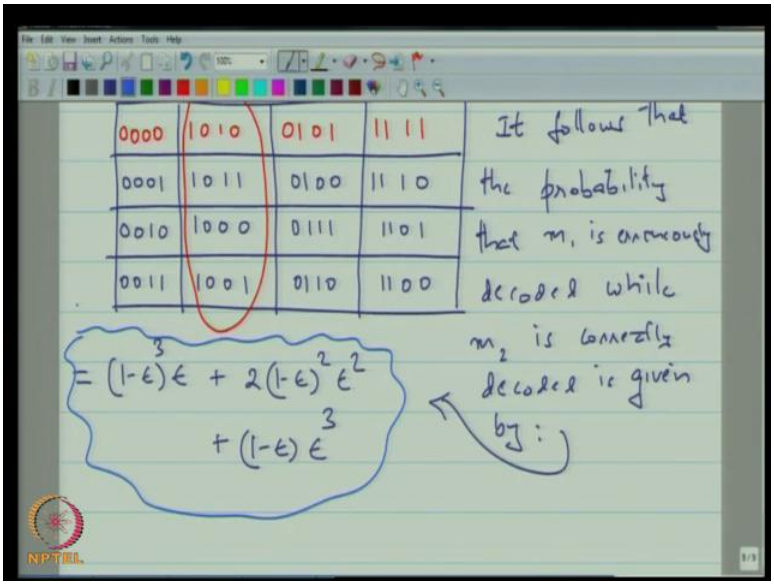
So you can see from here that since you recall the message symbols by stripping the first two symbols of the code. It is clear from this that m 1 is decoded erroneously, m 2 is decoded correctly. The reason behind that the 0 here that this symbol here corresponds to the first message symbols this 1 corresponds to m 2.

(Refer Slide Time: 25:45)



Now just in case in further explanation is needed and do not forget that your c hat is your c hat is c plus e plus e hat which is c plus e tilled. So, that is the reason, why I am saying that the first two message symbols there, whether they or correct or not to determine by the first two symbols in e hat e tilled because of this. Once you have this, now if I go back to this and I ask you what is the probability that what is the probability let say that the first message be does not. But not the second, if I want to know what is the probability that I what is the error but not m 2, that you can actually see that the only error patterns for which there is true are precisely those which correspond to this column, because here there is no error in even message symbol. Here there is no error in the first there is an error in the second here this is an error in both. So this is precisely this set of all this error patterns for which the first error, the first message symbol is decoded error only is 3 and the second is not.

(Refer Slide Time: 27:38)



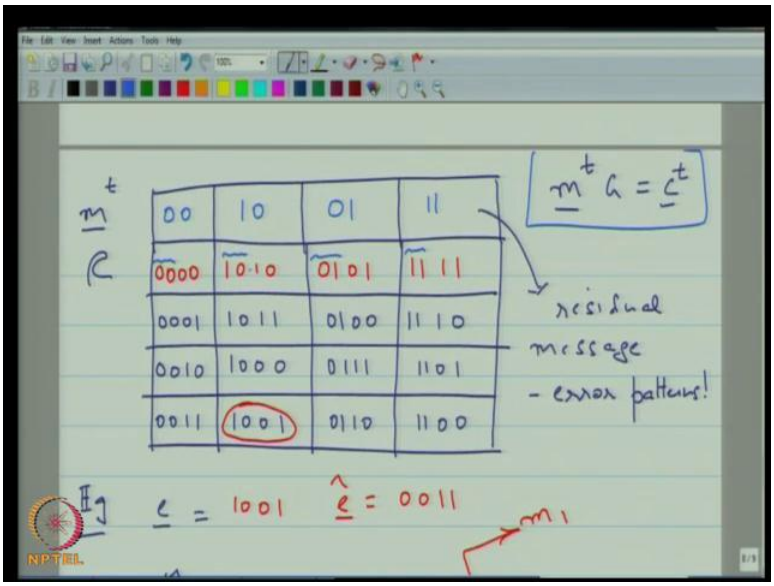
So, now let me just again do this, select this...

It follows it follows I am going to circle this column, because this is the key here. It follows that the probability that m 1 erroneously decoded while m 2 is correctly decoder is given by I am going to write this competition here. So, the competition is just corresponds to this it is equal to and I just sum the probability is that the vectors in this column at the error vector. So, you can see that two of them one of them has hamming weight 1, two have hamming weight 2 and this.

So, you can see that therefore it is 1 minus epsilon cubed epsilon this is for the single error vector plus 2, 1 minus epsilon squared epsilon squared plus. The case when you have three will have plus 1 minus epsilon into epsilon cubed.

So, this is the probability that the first message symbol is decoded correctly whereas the second decoded incorrectly second is decoded correctly. Now when one thought that might be crossing the mind that is fine, because this is the systematic code what effects not systematic can I still of this, and if you think about it, yes you can. The only thing is that only advantages systematic code really is that when you actually set up a table like this.

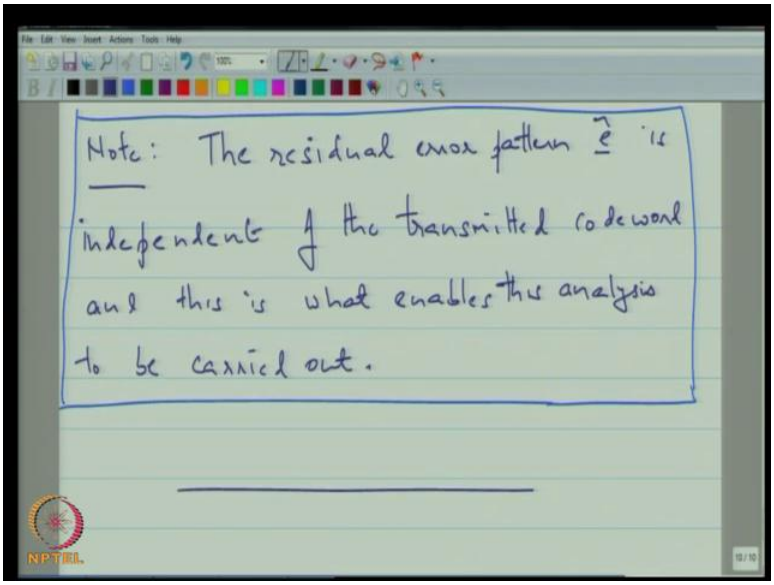
(Refer Slide Time: 29:50)



So, let us go back here. When you set up the table where you have the standard array and then you miss corresponding message symbols. In the case of the standard array the message symbols are a parent from the codeword, you can just simply pick them out is being the first two symbols. Now if this was not systematic then you have to work a little bit harder, because you have to go to generate a matrix and figure out what is the underlining message vector corresponding to these codewords. And then, you can do exactly what we did. So, and then you can exactly repeat what we did here. So, systematic are not you can use the standard array decoder to compute whatever expression you need relating to bit error probabilities.

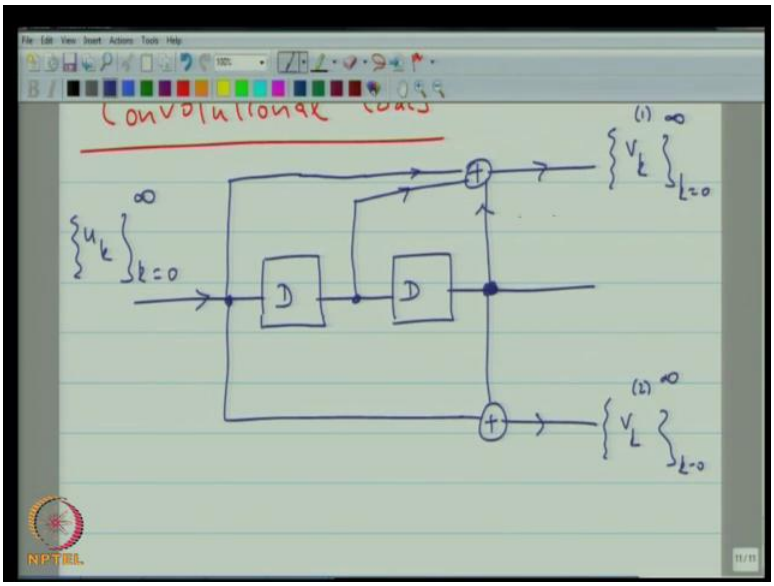
So, it is quite powerful in that sense. So, I think with this what I actually like to do is, I want to move on to the next topic, but perhaps I think just look at the notes here I think I should I precise one point before we complete this section conclude this section.

(Refer Slide Time: 31:00)



Note, the residual error pattern e tilled is independent of the transmitted codeword, and this is what enables this analysis to be carried out. Perhaps, no exclamations mark more out note of. Remind the note. So, let us go back here. So, when need did the analysis here and I started same well supposing this is the error pattern. Then, this will be a coset leader and this will be a residual error pattern. I never actually mention the transmitted codeword, because it terms of be the residual error pattern is independent of the transmitted codeword and that is where actually permitted all of this analysis to be carried out. That concludes our discussion on this topic, and now we are ready to start on talking about a new class of codes. This is also an important class of codes, and these go by the name convolutional codes.

(Refer Slide Time: 33:12)



The way we are actually like to do this I am going to introduce an example convolutional code to you. In fact that code that I will introduce to you is the prototype convolutional code in sense that if you open any book convolutional codes, it is very likely that this is the example code that there actually shown you. It is a nice example code. So, will take a look at this example code and will proceed with it. So, will actually study the encoder then, we look at methods of representing the code symbols then, we look at something again to the something which is similar to the generator matrix and so on. So, we will just keep perusing this example and little later I will come back and say, what is the general case? The general case will turn out to be quite or similar.

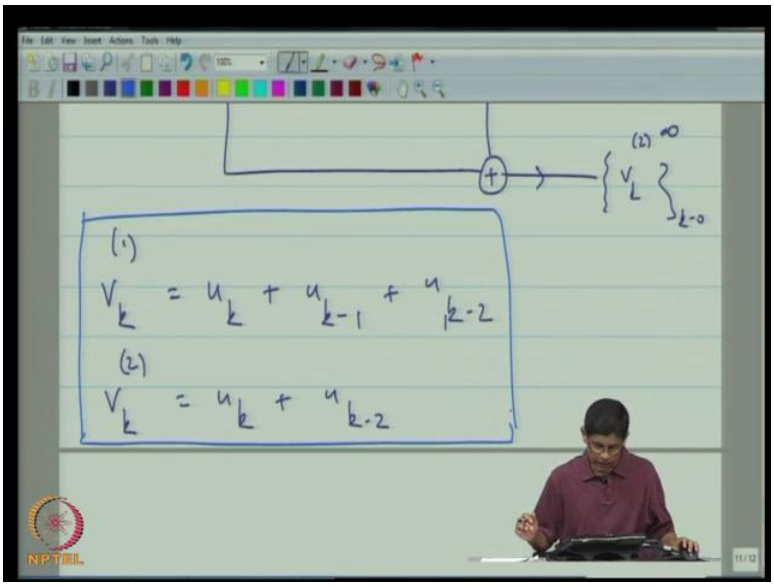
So here are example encoders. In this encoder, the encoder looks something like this. So, you have two bit two registers like this, and what you have in the input is the stream of bangle symbols k going from 0 to infinity. So, this is the stream so, let me do the writing latter, do not want to clatter the background. This is the input, this is single stream of bangle digits here and we are out that going to be two streams of bangle digits. These are generated as follows...

Let me just clean that applicant and I can draw that better. So, that is one output and that corresponds to the stream v k (1), and then the second output... and again these are senna infinite streams when in the start from 0 and go out to infinity. This is stream of symbols coming in and

they are two streams are symbols going out and already you can see that this devotional block codes. Because in the case of block codes you did not have this senna infinite streams, because with block codes what you have is, fixed collections of message symbols and they do not want to fix collection of code symbols. You did not have this infinite streams is continuing.

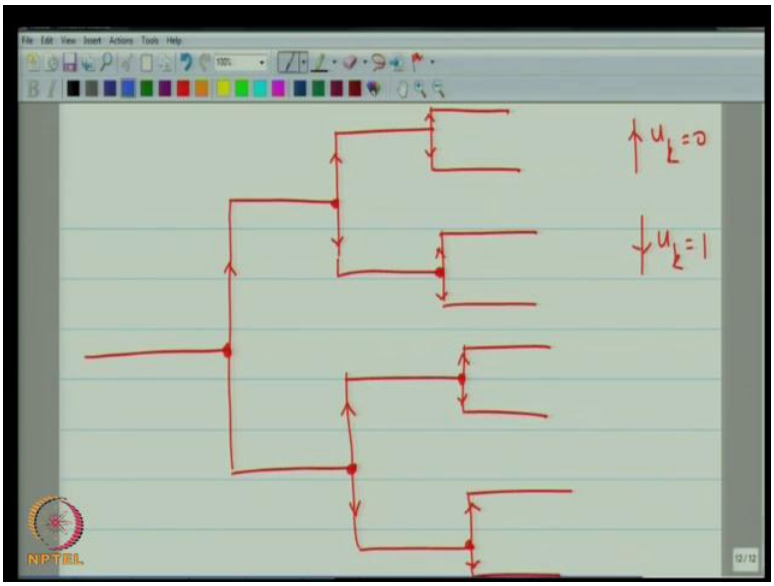
Also here, we are starting of with the encoder. So, what is the expression corresponds to this? So, what you really saying here is that the first output is the function of the current input and it adds to the current input the immediate pass to inputs. The second output adds to the current input the inputs two symbols ago.

(Refer Slide Time: 37:28)



So mathematically represent this like this. It say that v k (1) is u k plus u k minus 1 plus u k minus 2 and then, v k (2) is u k plus u k minus 2. So, this then is describes the encoder. Now convolutional codes belong to a class of codes which are known as tree codes. So, why are they called tree codes? Because you can actually represent each codeword as corresponding to a path in a tree. So, let me draw the tree and try to make things clear that way.

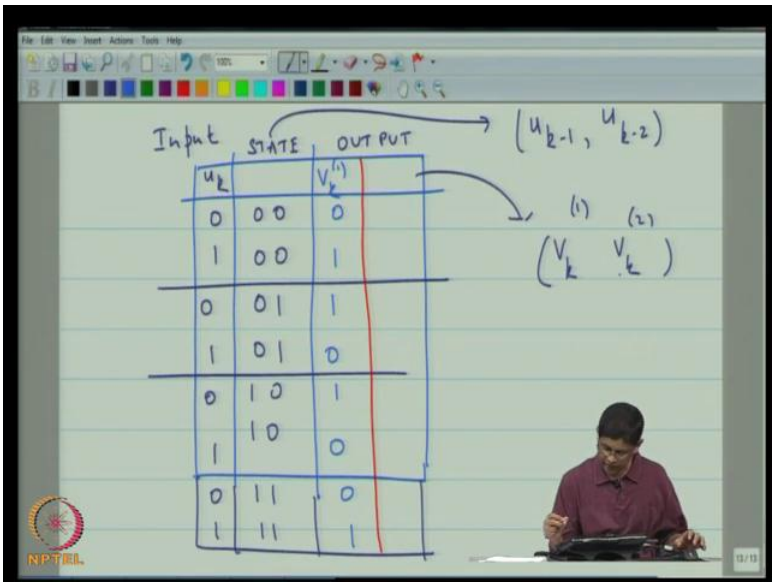
(Refer Slide Time: 38:30)



So, what I am going to draw here; let me put some arrows on this.

An up arrow means that the corresponding input symbol is 0, and the down arrow will mean that the corresponding input symbol is a 1. So, every codeword trace is corresponds to a path in this tree, and what about the output symbols? So, the output symbols are generated using this expression here. So, for that it is convenient to look at perhaps set up table. So, let us draw table, and which we put down both inputs and outputs.

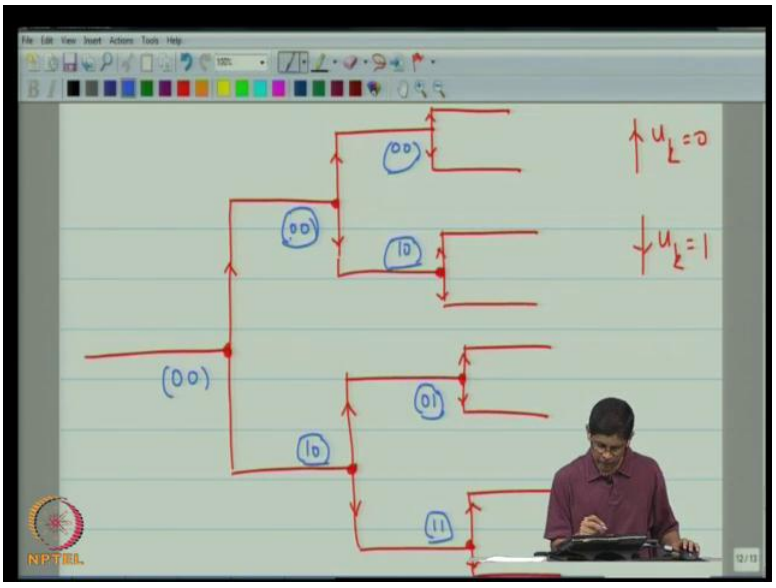
(Refer Slide Time: 40:50)



So, here you have the input. So, this is your u k this is what I will call this state of the convolutional code, and this is your output. Now the state is really captured by the past two symbols. So, that is u k minus let see u k minus 1, u k minus 2, and the output is v k (1), v k (2). So, this is what these things will represent. Now look, let us look at different possible states. So, you have 0 0, 0 0 that is one state then, you have 01 01 you have 10 10 and you also has 11 11. The input symbols can be the 0 or 1, 0 or 1, 0 or 1, 0 or 1. Now we have put they are two outputs. So, one says that you add the sum of these three symbols. So, that is v k (1), and so add the three 0 1, 1 0, 1 0, 0 1.

Now v k (2) what it does is that so, if we want to see where on getting that from see v k (1) adds the current symbols to the preceding two symbols, v k (2) adds the current symbol to the symbol to symbols back. So, here is v k (2) then. So, here I add the first and the last third entry. So, that is 0 1, 1 0, 0 1, 1 and 0. So, these are your entries.

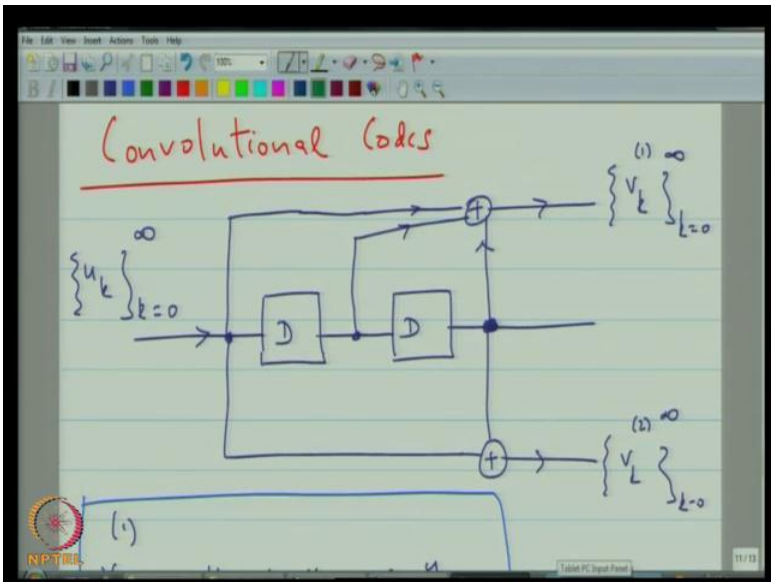
(Refer Slide Time: 43:40)



So, now with that you can actually go head and put this down. Now initially assume so use these notes here a correspondence to a state. So, you assume the state (0 0) here, and sincerely to see 0 here it continue to be 0 this became (1 0), this became (0 0), this is (1 0), this is (0 1) and this one, this state here is (1 1). So, these in some sense I have states. So, you have the states and we have inputs, because for it is an up arrow or down arrow tells you a inputs are and from that you can actually proceed to write down what the corresponding outputs are? You can check with that when the state is 0 0 the input is 0, your output is actually going to be 0 0. And, that will continue here, and it will continue here.

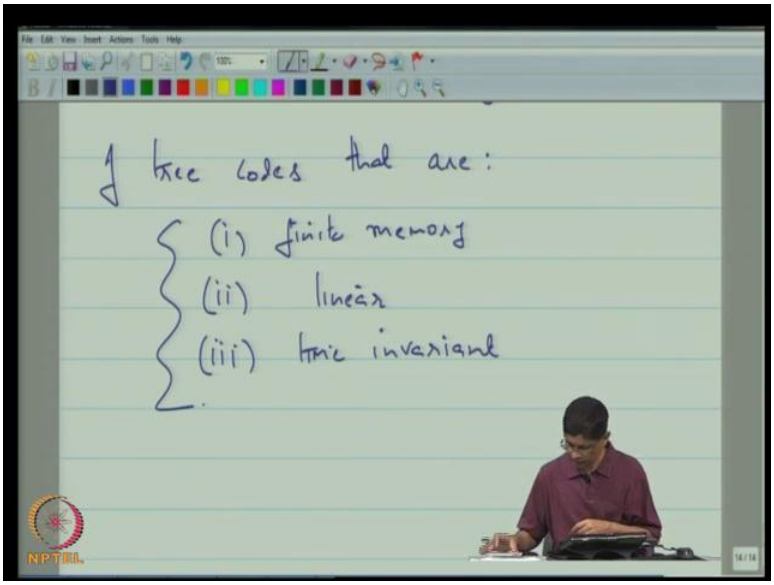
But if your state is 0 0, and you actually get a 1 then, in turns are that the output is going to be 1 1. And in similar manner you can actually fill in this entire tree. So, let this tells you is that convolutional codes are different from block codes in the sense that they can be represented the paths in a tree. And, the reason by introduce the tree; because of after all, it is a senna infinity sequence. There is no motion of block of message symbols and block of code symbols. And then but convolutional codes at least the class, there are the class of tree codes we are interested in are little bit more restricted than a complete general tree code. The reason for that is, because in a tree code you can have whatever rules for generating outputs that you want it could be time varying, it could be non linear. But in the case of convolutional code, convolutional codes are always generated by encoder of this type.

(Refer Slide Time: 46:00)



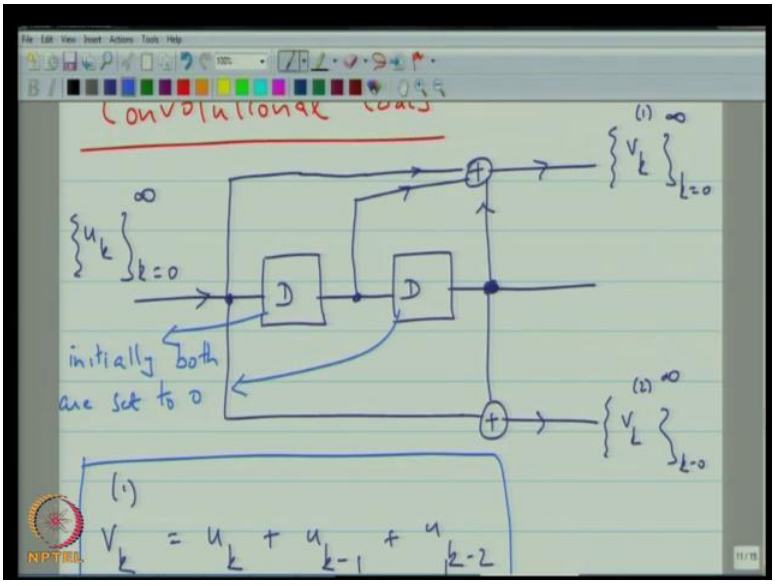
So, what is it that a characterized is these encoders? Well, first what the time invariant? That is the output only depends upon the current and the immediately passed inputs. It does not depend upon the time keep does not matter it is morning or afternoon, the output is still same. So, it is time invariant and then, it is linear. So, these in these ways it is a specialized tree code. Just to put it in perspective. So, let us says put that down in writing.

(Refer Slide Time: 46:40)



Convolutional codes belong to the class of tree code that are, 1, finite memory 2, linear and 3, time invariant. Now if you want to do something an anguses to what we did for the case of block codes, you could write down a generator matrix, except that it is not very convenient to write down the generator matrix, because it look something like this. So, here would be an input symbols u 0, u 1, u 2, extra and your output symbols you could view them has been generated using the matrix of this type. So, it will be a senna infinite generator matrix of this type, and this is what will need to v (1) 0, v (2) 0, and v (1) 1, and v (2) 1, extra. So, this is and you can check with this is true because what will really saying as for example take this symbol. We are saying that the by the way this is one thing I should clarify that.

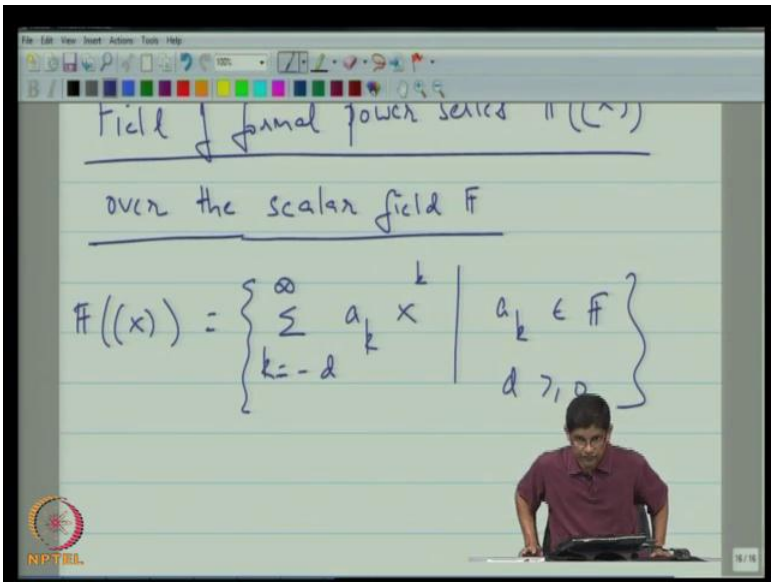
(Refer Slide Time: 49:00)



In this we always assume that this encoder that the initial conditions are that both these registers are set equal to 0. So, may as well mention that. So, both are set to 0 so, both these registers are initially set to 0. So, that is your initial conditions. So, then you can actually verify that your output symbols are related to the input symbols like this, but this is not very convenient again, because it is senna infinite. So, the more appropriate so, let me just call this the senna infinite generator matrix. So, the more appropriate description of the code is in terms of polynomial and formal power series. So, will make a slide detour and discuss the field of formal power series.

So, in a way this could be a continuation of an earlier discussion, where we set that whenever we need some math what I will do this can actually slide detour discuss the math and then bring you back to coding theory; when you talking about codes, I introduced groups, sub groups, cosets, rings, and fields, and vector spaces. So, similarly here in terms of that the math that you need in order to comfortably discuss convolutional codes is nothing but algebra, and particularly you need to know what a polynomial is? And what a formal power series is?

(Refer Slide Time: 51:00)



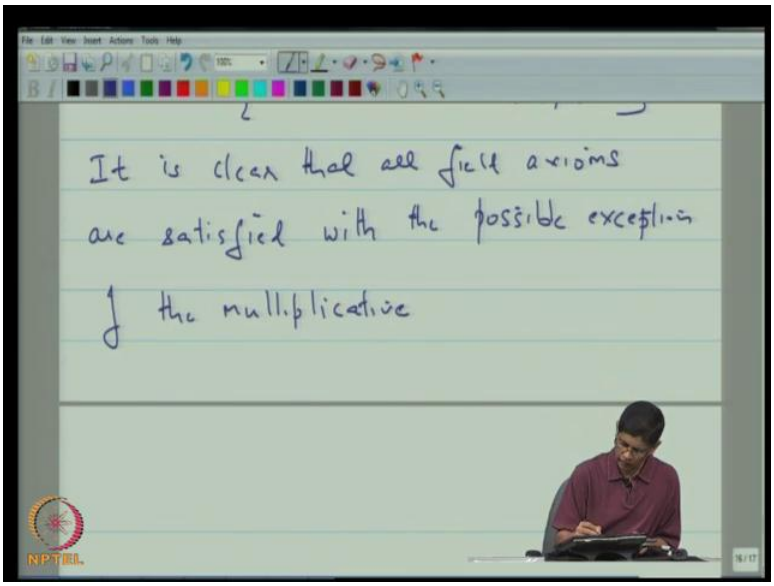
So, the field of formal power series over F. F is also a field. As in fact this make that clear let me write this as over the scalar field F. So, this is the set of all terms of the type k is equal to minus d to infinity, a k X to the k where a k belongs to F, and d is greater than or equal to 0. So, you look at this and say that is in this a polynomial? And the answer is no, because first of all here you are permitting finite number of negative coefficients and negative exponents. So, the X you could have X to the minus 1, X to the minus 2, and so on. But you only allowing a finite number of them which is probably actually say k goes from negative d to infinity, where d is itself greater than or equal to 0. But also polynomials have the notion of a degree.

That is there is a maximum exponent which you before to as the degree, but here there is no maximum, because it just keeps in general running off to infinity. So, in this way objects within the field of formal power series actually differ from polynomials. Now how do actually add and

multiply and divide? Now I am sure that any of you a given a formal power series then, now how to actually add and divide? Here have used X as the in determinant, but in our application to convolutional codes are they in determinate will be called d, where d stands for the delay operator. So, it is very clear how you would add two formal power series, because you just add them term by term. So, there is no confusion there.

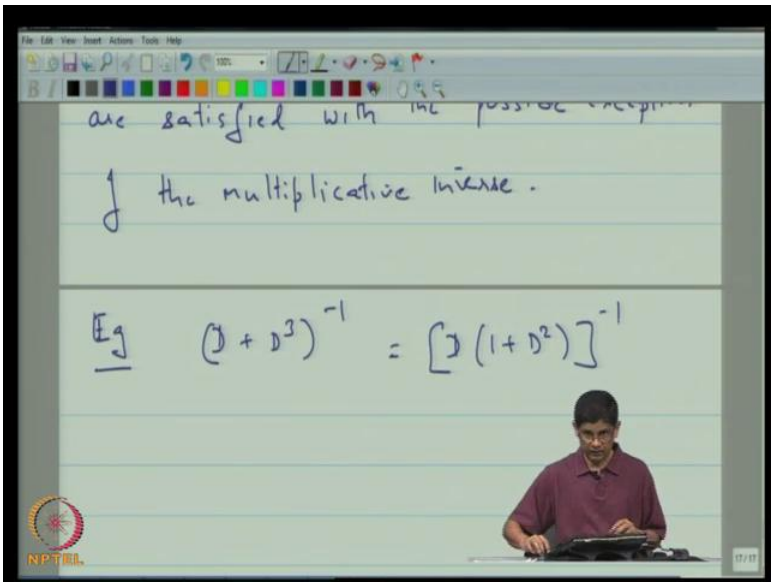
The quotient as well, how do you multiply them? Well, again that is not if you know how to multiply polynomial I am sure you figure out how to multiply formal power series. When a quotient is how do actually divide? So, I for that reason I only illustrate division or why the, since you are talking about the field what you really want to know is how do you compute inverses? Because you really have in order make something a field you need a set, and you need two operations addition and multiplication. So, the set is already define here, and then addition I have told you follows when a very straight forward way multiplication also. But when you go through the checking the axioms the most tousling axioms probably the axiom of the multiply get when inverse. So, we just illustrate that.

(Refer Slide Time: 54:40)



So, it is clear that all field axioms are satisfied with the possible exception of the inverse. So, will be this by example.

(Refer Slide Time: 55:50)



Suppose you are asked me what is D plus D cubed inverse. So, will be write this is D into 1 plus D squared inverse, and I just see that from my clock, there are I just have about 2 minutes to wind up. So, I think let me now perusing this example will do the competition next time perhaps, I should spend my time just recapping. So, what we actually did was the first part of the lecture was concern with standard array decoding and we said let us actually use the standard array decoder and figure out how to carried out performance analysis by way finalizing bit error probabilities code error, codeword error probabilities. Then we note to a completely new class of code known as convolution codes; and I am giving you to begin with various descriptions of the code. Eventually will settle upon one or two, which are convenient but we just introducing various descriptions, and we are somewhere in the middle and will continue next time by completing this discussion on formal power series. So, thank you.