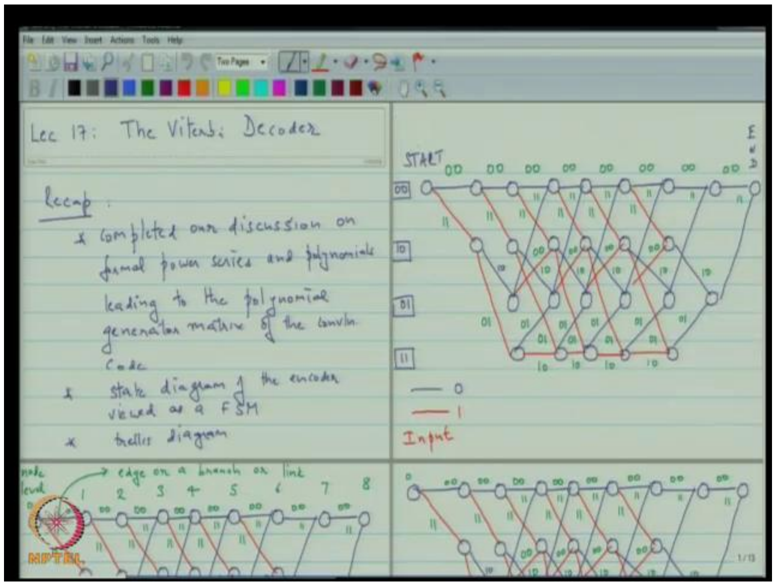
Error Correcting Codes
Prof. Dr. P. Vijay Kumar
Electrical Communication Engineering
Indian Institute of Science, Bangalore

Lecture No. # 18

Catastrophic Error Propagation

So, this today we began a lecture eighteen, we will continue discussing convolutional codes. Last time, we spent almost the entire lecture talking about the Viterbi decoded.

(Refer Slide Time: 00:35)

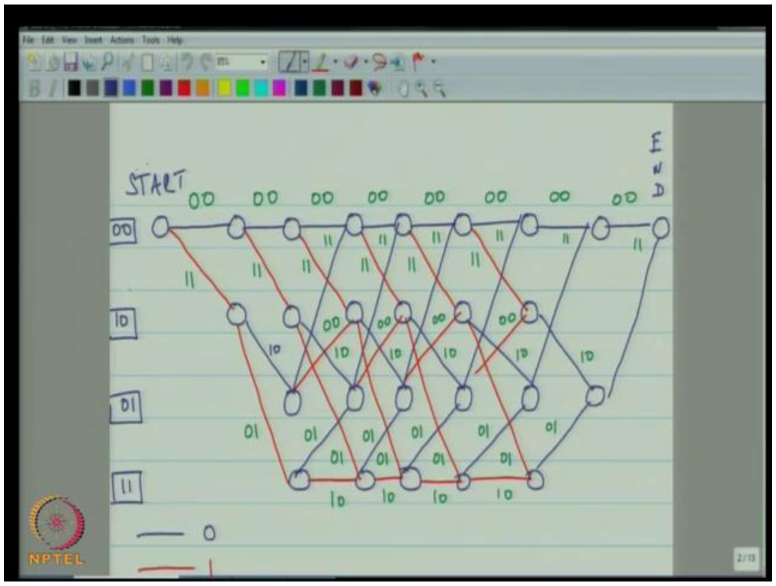


Lec 17: The Viterbi Decoder

Recap:
* completed our discussion on formal power series and polynomials leading to the polynomial generator matrix of the convl'n code
* state diagram of the encoder viewed as a FSM
* trellis diagram

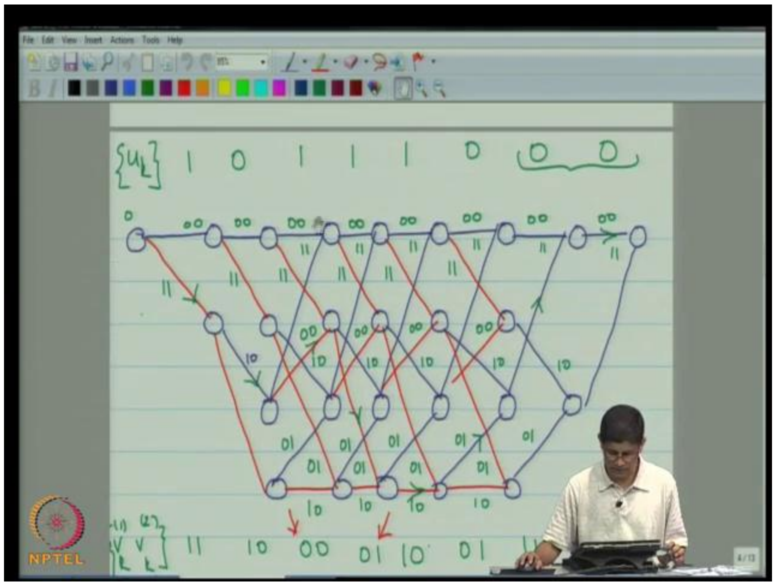
So let me, do the following, I will just run through the slides, and then I will do the quick recaps. So we started by looking at by sitting up the trellis of the convolutional code.

(Refer Slide Time: 00:54)



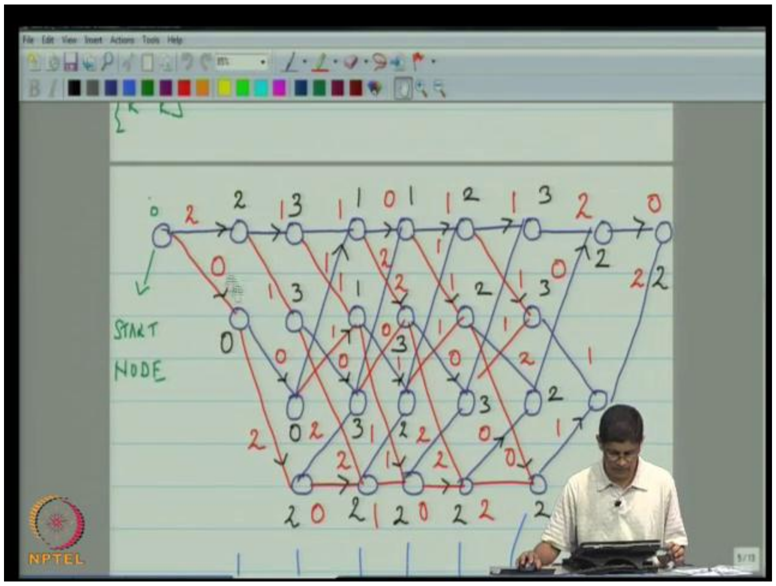
And the way the convolution code is set up is that the, you have all your every branch in the trellis is labeled with an output segment. In such a way that from start to finish and I am going from start to the end node, every path it takes response to code word. You can identify the message sequence associated to that path by just looking at the sequence of reds and blues, in these figure; and you can look at the code word just by looking at the output of the output symbols that are listed sequentially. So, for example if we went along with the top part it would be string of zeros, like this and so on.

(Refer Slide Time: 01:40)



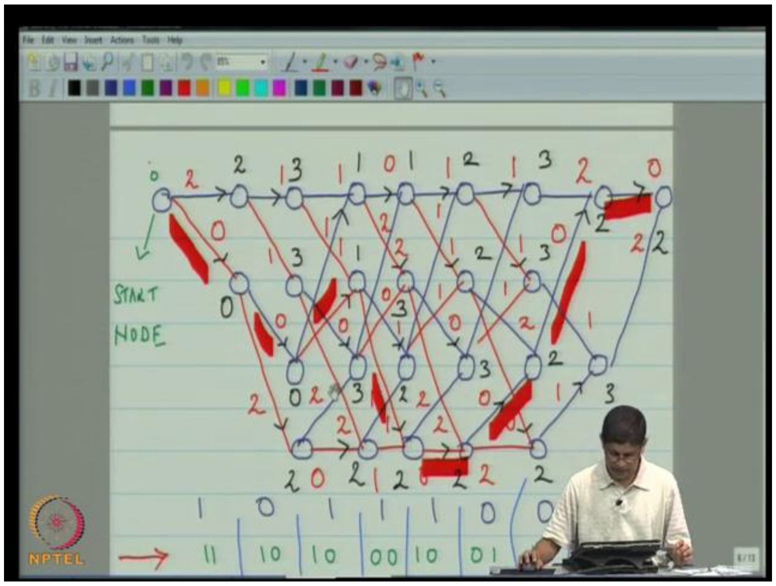
And then we actually showed how you would use this for decoding in a particular example. So in the particular example here is the trellis, and it is label just it we shown in, and let say that we took that particular message sequence 1 0 1 1 1 0 0 0, and then the corresponding output sequence over here. We introduce two errors as indicated by these red arrows, and this was then we receive sequence.

(Refer Slide Time: 02:00)



And then what we do even back to trellis in we label the trellis by actually list in the hamming distance between the output that the segment of the code word corresponding to a branch and the segment corresponding to the received sequence. So for example here we put 2, 3 and so on. In this way, we were able to label every branch. And then from a brute force point of view, all that we have to do is examine all path from this start note to end note, and then pick the one that has the closed, that is closest hamming distance to the receive vector, but that involves too many competitions, because the number of paths increase exponentially with the length of these depth of these trellis. So the clever thing that the Viterbi decoded does is that well, let us start from the left and keep going to the right, and you will see that in this way you can actually keep discarding certain paths; and that was the idea began the decoder.

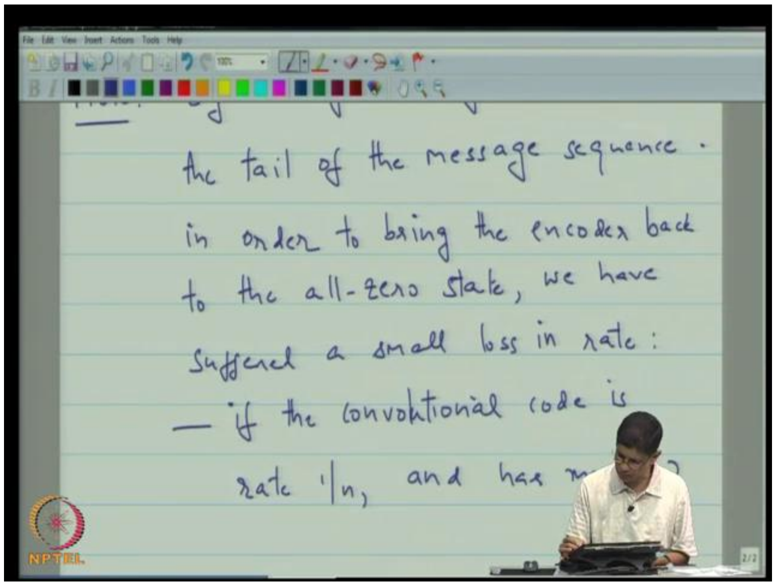
(Refer Slide Time: 03:02)



And we pursue that and we were able to actually decode the message correctly in these particular case. Now, so with that let me get on today's lecture, which is entitled catastrophic error propagation. But I will just (( )) some toolbars here, and put down recap, labeled the trellis diagram, and then explained the operation of the viterbi decoder. So, that is what we did last time; so today, I just before I start our main topic, which is catastrophic error propagation, what I would like to do is just make one point about last time viterbi trellis. So lets us just go back once that.

Now, if you look in the trellis, what I have actually done is put in two trailing zeros over here, and the I have already explains the last lecture the reason for bringing the (( )) so that you could actually bring the trellis are the finite state machine underlying this trellis back to the all zeros state. So that means last two message symbols are actually known, so even though the number of times you address a channel is 1, 2, 3, 4, 5, 6, 7, 8 in these instance, the number of message symbols that you actually transmitting is only 6. So these trailing zeros are a convenience and can be also be shown to improve performance but they do result in a loss in rate, and I thought I just documented before continue.

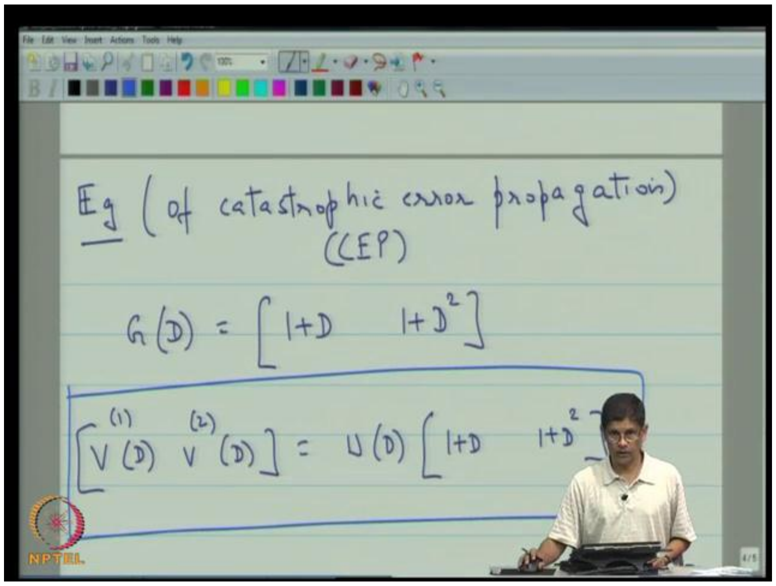
(Refer Slide Time: 06:13)



the tail of the message sequence .
in order to bring the encoder back
to the all-zero state, we have
suffered a small loss in rate :
— if the convolutional code is
rate 1/n, and has

Note, by adding a string of zeros at the tail of the message sequence in order to bring the encoder back to the all zero state, we have suffered a small loss in rate. How do you quantify that? If the convolutional code is rate 1 by n, and has rate and has memory nu, so the memory nu; so what we actually mean is that this is the number of shift registers on the input line, then the loss in rate per message sequence of length N is given by N minus nu, it is N minus nu by N into k by N. So it would be… Now nominal rate of the convolution code is 1 by n, but now the fact that you actually introduced these trailing zeros brings in these other factor. So this is your actual rate; so that the loss in rate is this. As you can see the loss in rate is negligible, if N, capital N is large.

(Refer Slide Time: 11:39)



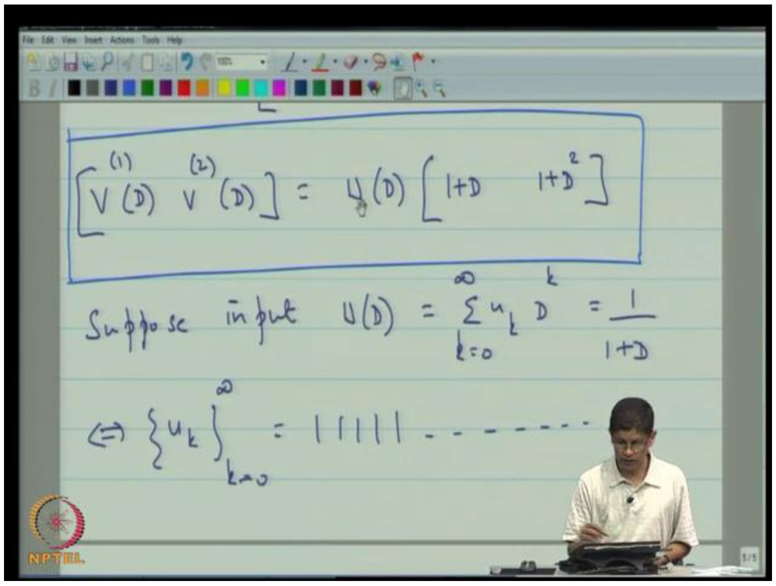
Eg (of catastrophic error propagation) (CEP)

$$G(D) = \begin{bmatrix} 1+D & 1+D^2 \end{bmatrix}$$

$$\boxed{\begin{bmatrix} V^{(1)}(D) & V^{(2)}(D) \end{bmatrix} = U(D)\begin{bmatrix} 1+D & 1+D^2 \end{bmatrix}}$$

Now I want to get into the topic of the lecture, which is catastrophic error propagation and we are actually began that is by using an example, in this example consider convolutional code. So this is an example of catastrophic error propagation, and we will abbreviate that to be CEP. So consider the case, when the polynomial generate a matrix is given by 1 plus D and 1 plus D square, and so the input output relationship, as you know is given by this equation. Now so, catastrophic, I am sure as convey the meaning that something bad is going to happen.

(Refer Slide Time: 13:31)



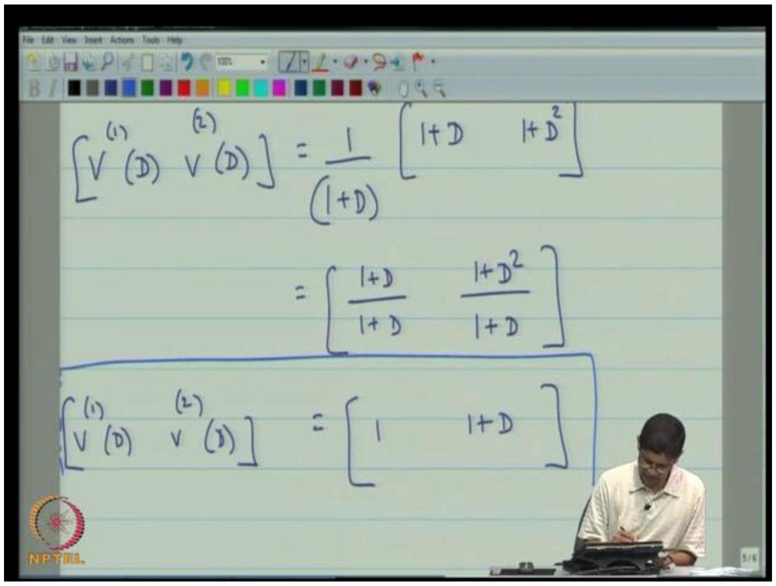
$$\boxed{\begin{bmatrix} V^{(1)}(D) & V^{(2)}(D) \end{bmatrix} = U(D)\begin{bmatrix} 1+D & 1+D^2 \end{bmatrix}}$$

Suppose input $U(D) = \sum_{k=0}^{\infty} u_k D^k = \frac{1}{1+D}$

$\Leftrightarrow \{u_k\}_{k=0}^{\infty} = 11111 - - - - - - -$

So I am going to tell you, what that is suppose your input, the input U of D, now of course this is the input power series. So this is the sum U k D to the k, k goes from 0 to infinity. Suppose the input sequence use of k is such that this is equal to 1 upon 1 plus D, which would correspond to continuous, so this is equivalent to say that your sequence U k, k goes from 0 to infinity is string of 1, it is going up to infinity. In this case, so now what I am going to do is, I am going to this equation and I am going to plug in 1 upon 1 plus D for U of D.

(Refer Slide Time: 14:35)



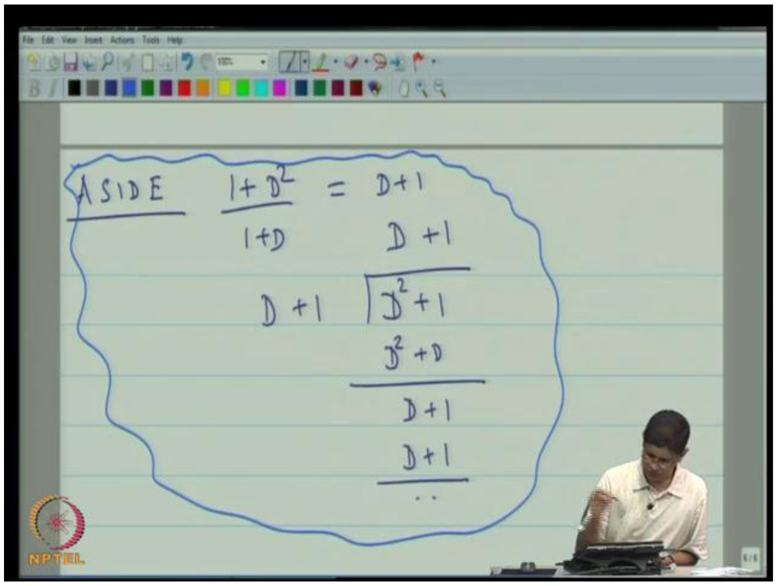
$$\left[V^{(1)}(D) \quad V^{(2)}(D)\right] = \frac{1}{(1+D)}\left[1+D \quad 1+D^2\right]$$

$$= \left[\frac{1+D}{1+D} \quad \frac{1+D^2}{1+D}\right]$$

$$\boxed{\left[V^{(1)}(D) \quad V^{(2)}(D)\right] = \left[1 \quad 1+D\right]}$$

Then, V 1 of D, V 2 of D is given by 1 upon 1 plus D into 1 plus D into 1 plus D square, and that is same as 1 plus D upon 1 plus D, 1 plus D square upon 1 plus D. So this is 1, and so the question is what do you get and when you divide 1 plus D square, 1 plus D. So let us do that the side calculation.

(Refer Slide Time: 15:25)



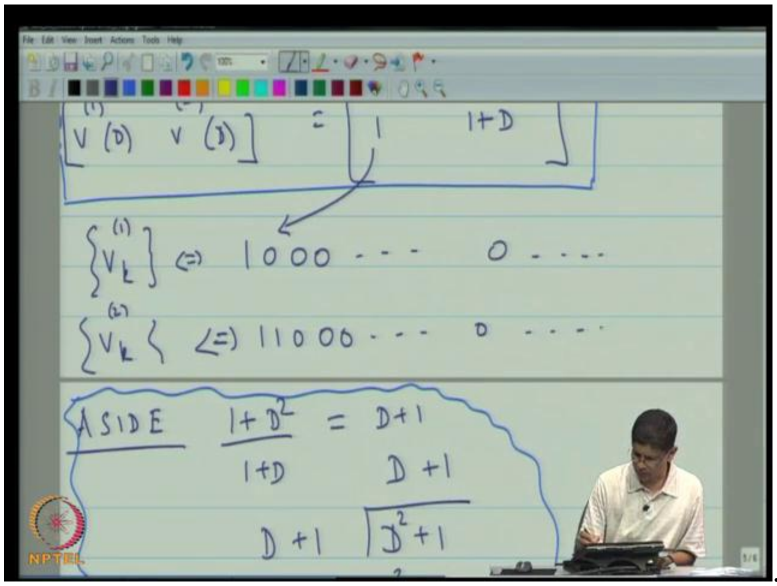
ASIDE $\frac{1+D^2}{1+D} = D+1$

$$\begin{array}{r|l} & D+1 \\ \hline D+1 & D^2+1 \\ & D^2+D \\ \hline & D+1 \\ & D+1 \\ \hline & \cdot\cdot \end{array}$$

So, this is an aside. So we want to determine, what is 1 plus D square upon 1 plus D. So we will put down D square plus 1 and D plus 1 and so it is important remember that we are dividing sometimes we are dividing power series and sometimes you are dividing polynomials, so it is important to keep in mind, which we are doing at any given instant of time. In this case, we are actually dividing polynomials; so we do not have terms going off to infinity; in which case what you do is whenever you write a term you actually put the highest term first, which is what we doing here. So, this is D, D square plus D, D plus 1, D plus 1. So what we see is that this thing is precise D plus 1. So that was the little side calculation. Now, let us go back in here and say that now, we know this is 1 and this is 1 plus D and so this is your V 1 of D and V 2 of D.

Now this is actually very bad and you thinking you have to looking at this since as well has not clear to me why is this very bad will the reason for that, is that here is the incidents and which your input sequence was infinite stream of 1s right. You input sequence here was an infinite string of ones and the other hand when you come to your output, your output is a pair of polynomials. So what is the corresponding output sequence is look like? This correspondence to the sequence, V 1 of k which is 1 followed by an infinite string of 0s.

(Refer Slide Time: 18:00)



$$\left[v^{(1)}(D) \quad v^{(2)}(D)\right] = \left[1 \quad 1+D\right]$$

$$\{v_k^{(1)}\} \Leftrightarrow 1000 \cdots \quad 0 \cdots$$

$$\{v_k^{(2)}\} \Leftrightarrow 11000 \cdots \quad 0 \cdots$$

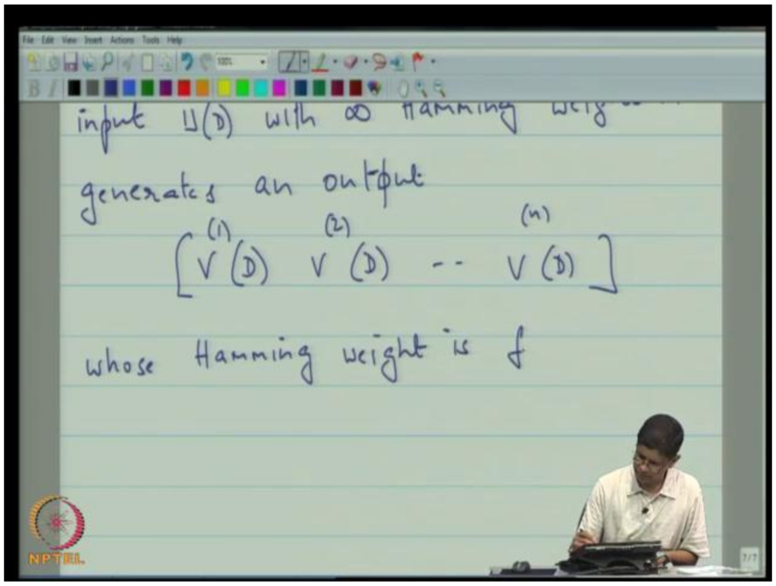
ASIDE $\frac{1+D^2}{1+D} = D+1$

$$D+1$$

$$D+1 \enclose{longdiv}{D^2+1}$$

On the other hand, V 2 of k corresponds to 1, 1 and then a string of zeros. So even though your input as infinite hamming weight your output has finite hamming weight. Otherwise that the problem because, now supposing by chance the channel well as to actually corrupt just three message symbols supposing the channel corrupts these bits. So let me just write channel corrupted here, if this bits by channel corrupted then the output sequence would look exactly like the output sequence corresponding to an infinite string of 0s. So, that means that a finite number of channel errors can cause an infinite number of message symbol errors and that situation is called catastrophic error propagation. So let me, try to give you definition for an encoder matrix G of D for a convolutional code is said to have catastrophic error propagation. If there is some input U of D with infinite hamming weight generating are rather that generates an output.

(Refer Slide Time: 21:00)



input $U(D)$ with ∞ Hamming weig

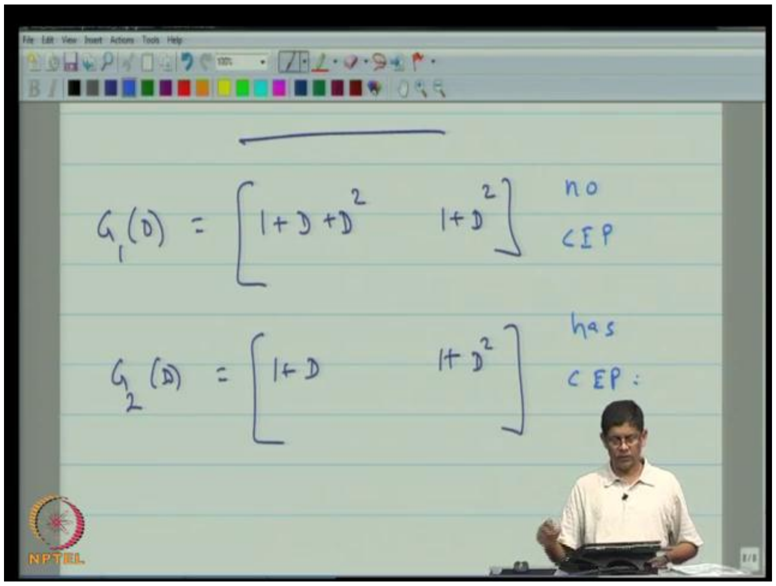
generates an output

$$\left[V^{(1)}(D) \quad V^{(2)}(D) \quad \cdots \quad V^{(n)}(D) \right]$$

whose Hamming weight is f

V 1 of D, V 2 of D are in general let see with is a rate one by an convolutional code, whose hamming weight is finite; so in a encoded matrix for a convoultional code is said to have catastrophic error propagation. Now, I do want infuses that whether are not the right question to ask, whether your encoder is catastrophic are not, sometimes people casually say that certain convolutional code is catastrophic, but actually that is not the right terminology. For the same code you could have two different encoders, one of which could have catastrophic error propagation and one which does not, so that why I have said and encoder set of catastrophic error propagation, because this is really of phenomenon. That has done with the mapping between input and output. If this is generated matrix G of D as you see over here which actually maps in infinite weight input, such as this to finite hamming weight outputs, then this encoded has catastrophic error propagation and the reason for that is simply because of finite number of channel errors can cause an infinite number of message errors. This terminology stems from the observation that if the encoder has catastrophic error propagation, then a finite number of channel errors can cause an infinite number of message symbol errors.

(Refer Slide Time: 24:34)

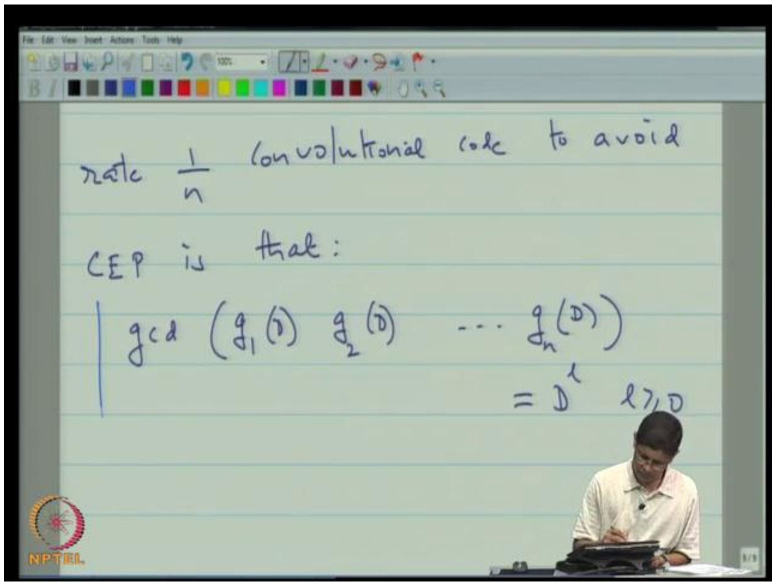


$$G_1(D) = \begin{bmatrix} 1+D+D^2 & 1+D^2 \end{bmatrix}$$ no CEP

$$G_2(D) = \begin{bmatrix} 1+D & 1+D^2 \end{bmatrix}$$ has CEP :

Now obviously this is a situation that you like to avoid, so the natural question is what is it that is different between the two encoders. So there are consider the following generative matrices. If you consider G of D, which was the older encoder corresponding to 1 plus D plus D square 1 D square it turns out that this does not have catastrophic error propagation. On the other hand, if you look at the second polynomial error matrix, which 1 plus D 1 plus D square this has catastrophic error propagation. So, the question is what is it that really is causing the trouble; and the problem is that in this second example, down here, the two symbols in the generative matrix share a common factor and that is really what is causing the problem. It terms of the there is no common factor here. So now, I would like to do is state of theorem which actually tells you when precise even catastrophic error propagation takes place.

(Refer Slide Time: 26:02)



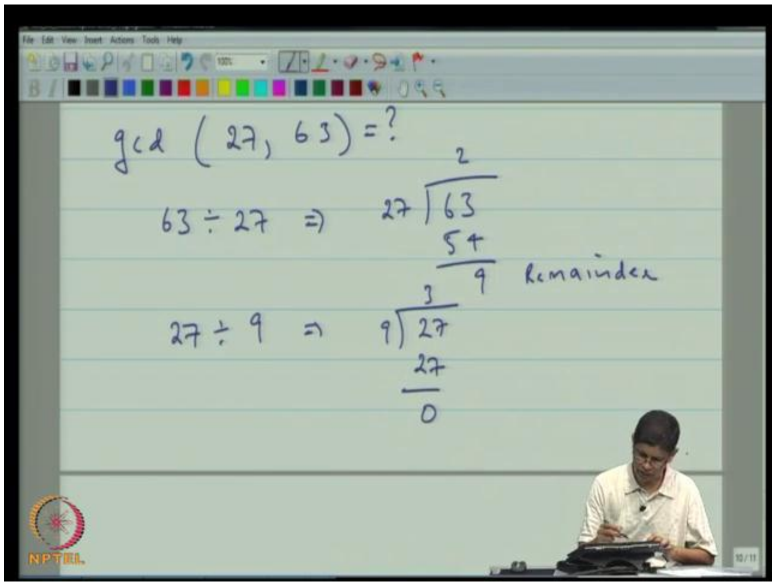
rate $\frac{1}{n}$ convolutional code to avoid

CEP is that:

$\gcd(g_1(D)\ g_2(D)\ \dots\ g_n(D)) = D^l \quad l \geq 0$

So, theorem a necessary and sufficient condition on the generator matrix of a rate 1 by n convolutional code to avoid catastrophic error propagation is simply that the gcd of the gcd of g 1 of D g 2 of D g n of D is equal to D to the l for sum l greater than equal to 0. So, what exactly are these g as well these are nothing but components of the generated matrix. So these things are the elements of the generated matrix. So well just put that in where g of D is g 1 of D g 2 of D g n of D is the polynomial generated matrix of the code. So the theorem just as that all that you have to do is compute the greatest common deviser. If the greatest common deviser is some pair of D, where that exponents greater than equal to 0. Of course the convention is that, if you take D to the zero then that is 1, so the gcd can be one and can be D, D square and so on. But only these are promise able, if this is the gcd and this is does not have catastrophic error propagation and the other hand it does. So for that in order to decal little deeper I like to explain go back in explain how gcd are compute, because we will also running to gcd when we actually deal with decoding algebraic codes such as a reads on code.

(Refer Slide Time: 29:50)



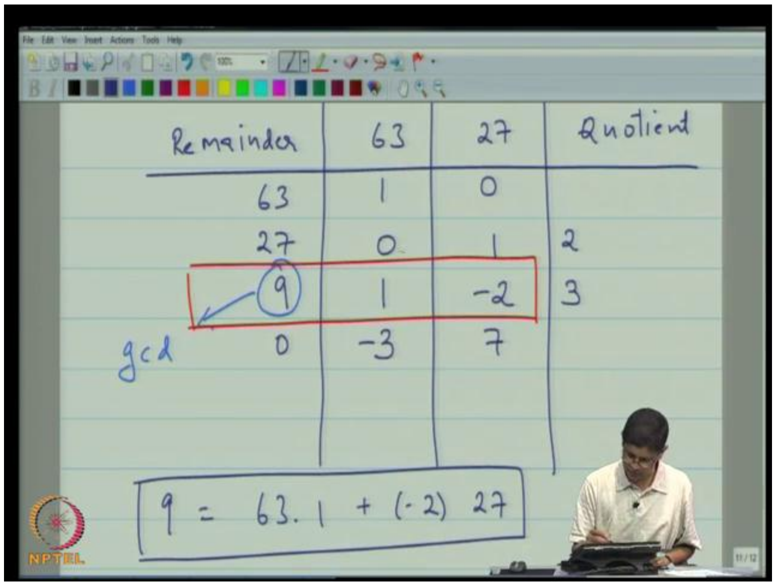
gcd $(27, 63) = ?$

$63 \div 27 \Rightarrow$ $\begin{array}{r} 2 \\ 27\overline{)63} \\ 54 \\ \hline 9 \end{array}$ Remainder

$27 \div 9 \Rightarrow$ $\begin{array}{r} 3 \\ 9\overline{)27} \\ 27 \\ \hline 0 \end{array}$

So, just a quick review of a computing the g c d, so first of all an example, what is the gcd of the integers 27 and 63. How do you compute it. Now, the algorithm that you would have learned has an high school is that you keep dividing and you get remainder 0. So 63 divided by 27, where so 63 divided by 27 is 2, 54 remainder is 9 then would you do is you go on to looking a 27 divided by the remainder whatever is the remainder here. So divided by 9, so that of course the you know that the remainder is going to be a 0. So, as soon as is 0 stop and then you declare the previous remainder to be the g c d, so in this case gcd terms are to be given by 9. Now, it terms up that the similar algorithm will also work in our case, but we want to extract the little bit more information than the working as gives us. So, what will do is we will organize this competition in the form of a table.

(Refer Slide Time: 31:32)



Remainder	63	27	Quotient
63	1	0	
27	0	1	2
$\boxed{9}$	1	-2	3
0	-3	7	

gcd

$$\boxed{9 = 63.1 + (-2)\ 27}$$

So, let us we have to do that, so will form a table, in which you will have a remainders on this side then you have 63 and you have 27, and then you have quotients on this side and then draw some vertical lines. So, are first two entries are going to be 63 and 27 themselves and I am going to put down a 1 and 0 here 0 and 1 here. Now the interpretation here is that 63 is 1 time 63 plus 0 times 27 27 is 0 times 63 plus 1 time 27. So you can think of these is somewhat very similar to an working with an excel spread sheet for you have some formula. Now to compute the gcd work. We kept on dividing and keeping note of the remainder.

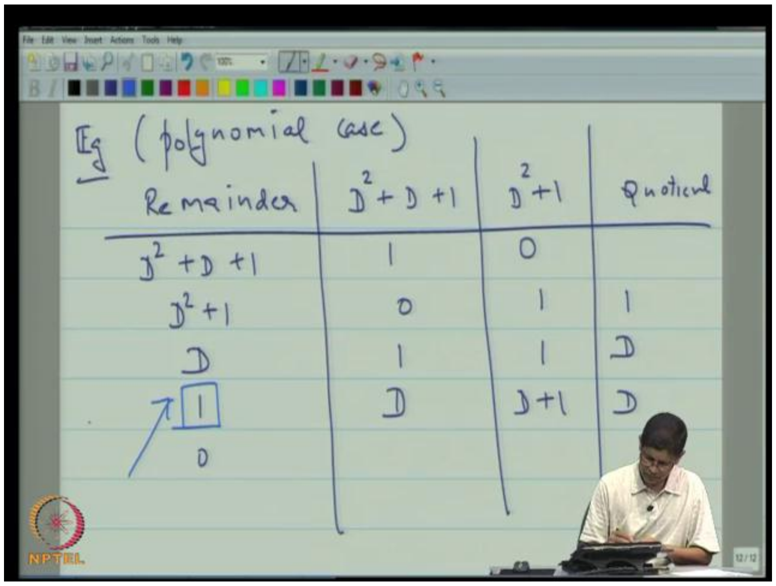
So, when we divide 63 by 27 the quotient is 2 and the remainder is 9. Now, these first entry in the third row are obtained in the address byte by taking 63 and subtracting twice 27 from it. So, the same thing of here I takes ones of two times of 0 and I will get 1, I take 0 subtract two times 1 and I will get minus 2 over here. Now the excel type formula is still valid 9 is one time 63 minus two times 27 because, 63 minus 54 is 9 and now will continue, so 27 divided by 9 gives you quotient of 3 and remainder of 0. Now this 0 was obtain by subtracting 3 and 9 from 27. So 0 is the entry here, should be 0 minus one time 3 that is negative 3 this is 2 into 1 minus 2 into 3. So, thus 2 into 1 plus 2 into 3 thus 2 plus 6 which is 8, and you can verify that.

So let see, excuse me there should be slight mistake here. So what we want do is we want to 2 into 1 plus 3 into 2 that is actually 8 so it left side. Alright so now, this 1 as they got the remainder 0 would be actually observed was that this is the gcd that we want and the focus on these. So what these table gives you apart from the gcd because whatever calculation you did in the left most column there really of know conscious where are similar what we did earlier. So there is nothing new that the table adds what the table does provides is this extra row that you actually see over here, in what this row actually tells you is that 9 is equal to this row actually tells, you that 9 is equal to 63 times 1 plus minus 2 into 27 .and it terms are their this expression is true useful for us.

This is one small correction I want to make here this thing here should actually be a 7 the reason being that like in the excel spread sheet 27 minus three times 9 is 0, 0 minus 1 time 3 is minus 3, 1 minus of minus 2 times 3 is 1 plus 6 which is 7. And you can verify that this is correct because, seven times 27 minus three times 63 and actually give you 0. So are that this extra information that this table gives you is precisely this equation over here and will use that. Now let is go on to by the way in case you a case this table also gives you the ingredients that you needs to actually true that what you got. You get this gcd because, if you look done here since when you divide 27 by 9 you get 0.

That is telling you that 9 divide 27, but since 63 is linear combination of 9 and 27 and 9 divides 9, 9 divides 27, so 9 also divides 63 so the these calculation. Here tells you that 9 divides both of the number on the top and these would hold in general even, if you had longer table this would still hold. This equation is that bottom is helpful, because it tells you that not only is a 9 a divides of 63 and 27 it is also the largest possible deviser reason being that anything that divides 63, anything that divides 63 or 27 must divide 9 from this equation. So this equation gives you it allows you to take the extra step in declare 9 to be the greatest common deviser.

(Refer Slide Time: 38:38)



Eg (polynomial case)

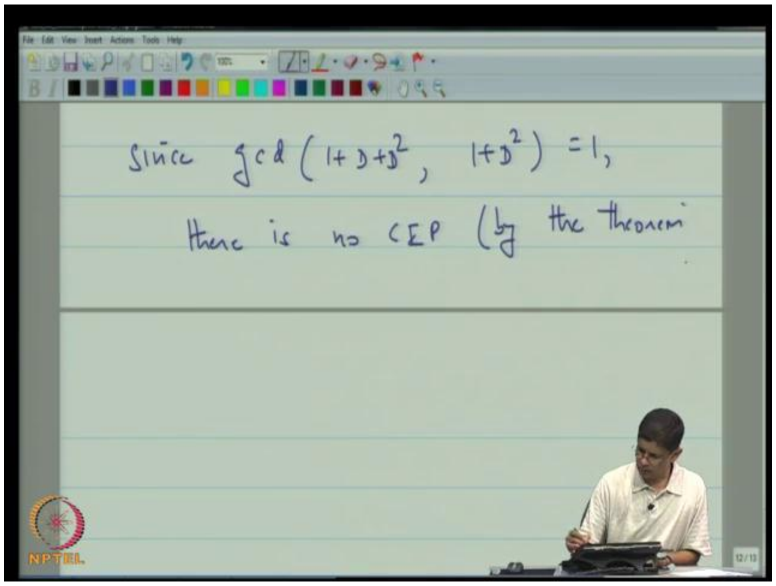
Remainder	D^2+D+1	D^2+1	Quotient
D^2+D+1	1	0	
D^2+1	0	1	1
D	1	1	D
$\boxed{1}$	D	$D+1$	D
0			

Now let us, look at the analog in the case of polynomials. So, let is do first of all take the example of code that we have just done earlier and what I am going to do is just cut these out. So in this case so apart from the remainder we will also have two polynomials D square plus D plus 1, D square plus 1 and a quotient. Why is this polynomial is interest, because if you go back a few slides you see that there were two generator polynomial generator may to see and I mention without proof admittedly that were as g 1 of D is an example of an encoder, which does not have a catastrophic error propagation; whereas g 2 of D is does have catastrophic error propagation and the difference really lies in that the gcd of the polynomial generated in the two matrices is different.

So what we doing here is not trying to compute the gcd of entries in first g 1 of D. So let us go down do that. So will list here D square plus D plus 1 and then D square plus 1 so this is 1, 0, 0, 1 just like. In the earlier case the next step is to actually divides D square plus D plus 1 by D square plus 1 the quotient is 1 and then D is to D and the remainder is D. And so now, in the next now the way to obtain this is multiplied this row by 1 is subtract from this so, we do the same thing for the remaining entries in the same row. So here it is 1 minus 0 times 1 that is 1, 0 minus 1 times 1 that is 1, the next step is to actually divide D square plus 1 by D, so the quotient is D the remainder is 1.

So, we do the same however we got this entry will reproduce the procedure to get these two entries. So we obtain this by taking these subtracting D times D. So here 0 plus 1 times D that is D this is 1 plus 1 times D that is D plus 1. And finally, we divide D by 1 so the quotient is D remainder is 0 as soon as the remainder is 0 we can actually stop and declare this to be the g c d. So, this is then a gcd and we see that the gcd is 1, now according to our theorem, which remains we proof in as long as the gcd is some part of d that I could be 0 then I your code does not have a catastrophic error propagation. So, that is the reason why this particular code does not have catastrophic error propagation.

(Refer Slide Time: 42:50)

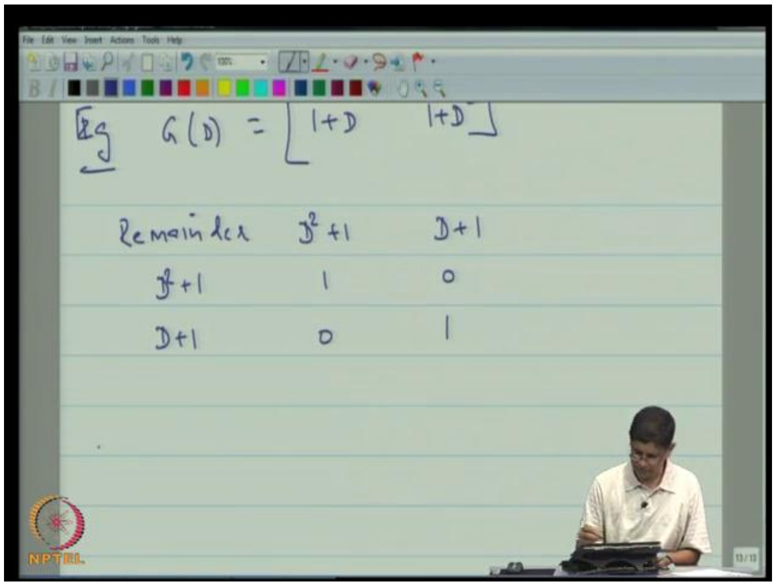


Since $\gcd(1+D+D^2, 1+D^2) = 1$,

there is no CEP (by the theorem

So, since the gcd of 1 plus D plus D square, 1 plus D square is equal to 1 there is no there is no catastrophic error propagation by the theorem.

(Refer Slide Time: 43:02)

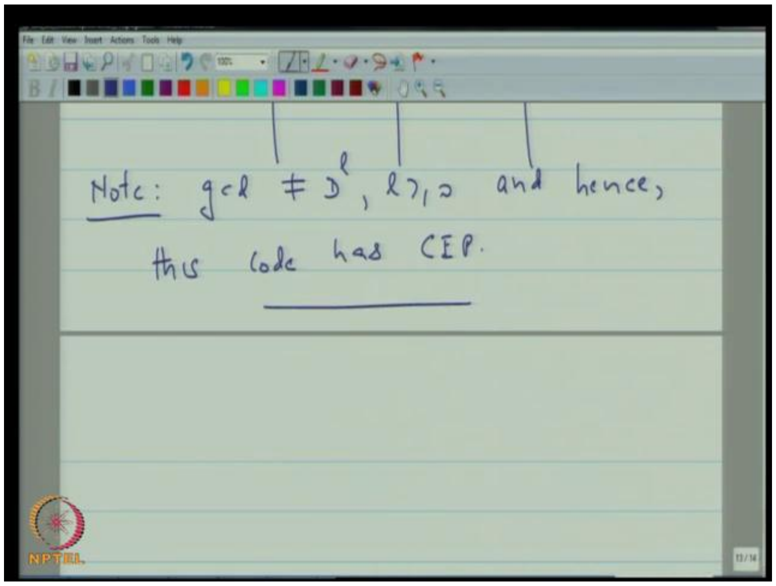


Eg $G(D) = \begin{bmatrix} 1+D & 1+D^2 \end{bmatrix}$

Remainder	D^2+1	$D+1$
D^2+1	1	0
$D+1$	0	1

Now let us look at the second example, where G of D was 1 plus D and 1 plus D square will setup the corresponding table. So here is your remainder here is 1 plus D square, 1 plus D actually let us write the highest term highest degree term first. So that division seems to be the natural. So D square plus 1 D plus 1 1 0 0 1 and then you are going to put down your quotient here. So, when you divide D square plus 1 by D plus 1 the quotient will be D and the remainder will be D plus 1 and here. It is 1 plus 0 times D, which is 1, 0 plus 1 times D that is D. Now you divide D plus 1 by D plus 1 the quotient is 1, the remainder is 0. So we stop and declare that the gcd is D plus 1. So in these the case of this particular generative matrix the gcd is not at the form to D to the l enhance by the it has theorem catastrophic error propagation.

(Refer Slide Time: 45:00)



<u>Note:</u> gcd $\neq D^l$, $l \geq 0$ and hence, this code has CEP.

gcd is not equal to D to the l, l greater than 0 and hence this code has catastrophic error propagation. Now I was actually interning to go through the proof of this theorem, but just keeping in mind that we have a fair number of topics to cover. What I am going to do is I am just going to give you verbally a scratch of the proof and leave it in that. So, we want to actually go through the process of writing it. The idea is that, we have two other topics to cover the fairly major topics. So I like to be able to get to them, so the way this proof actually works in the zoom out here is basically it obsess. So, the question to look at in trying to understand this is perhaps, this one. Now this is a particular example, where the gcd was 1 plus D and the input is 1 upon 1 plus D the general case is very simple if the gcd is 1 plus some polynomial the input can be that 1 plus that polynomial.

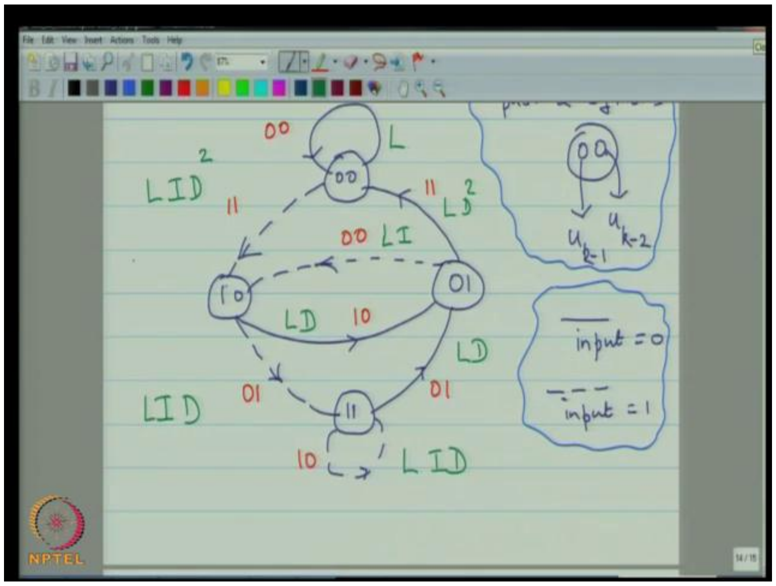
And, you can prove that thus the valid input and because this divides the denominator divides both the entries what you will get this output there are finite at polynomial enhance at the finite hamming weight whereas the input itself being of this form will always have infinite hamming weight. You can prove that, so that is why the greatest common deviser cannot be of the form 1 plus something because, then you can play this same trick in that case to actually exhibit and output sequence as finite hamming weight. Even though input has infinite hamming weight. Now on the other hand supposing the gcd of the form D to the l then you by are extended, you clear the divisional algorithm. Thus the algorithm, we just carried out you can actually show that the linear combination of these will give you a pair of D. So, that means intern that means a linear combination is that output will give you some delayed version of the input which means that the output themselves must have infinite hamming weight.

So I know I went to the rather fast, but presumably you can ravine listen to it again and any case that you just give you quick overview. So that we could move on, so in unactual a necessary and sufficient condition. For an polynomial generated matrix of catastrophic error propagation is that not do not have catastrophic error propagation is that greatest common deviser of the components. Be of the form some pair of D and this is analogist condition for the case from the convolutional code is of a k by n, so k other than 1 and will come to that. Now I want to get on to something else which is now if you consider the way in which we pursued convolutional codes it has been a little bit different than are treatment of block codes and intentional so because, what we dread was we follow the rather natural train of thought, we first should you for the convolutional encoder was what look like then I should you why belongs to class of tree codes and is thus different from a block code and then from that, we went on to see that the finite state machine produce give presence.

You can presence destruction of the code and this, like to the trellis is of course extremely useful because, you could do Viterbi decoded over the trellis. And now I just pointed out by the way when you actually select your polynomial generated matrices. You have to be careful to ensure that the phenomenon of catastrophic error propagation cannot take place there is your to make sure that the components of the polynomial generated matrix are actually rarely will be prime are at least the gcd is some pair of D. Now what we do is do something and to same you know after all performance in the case of block code depended upon the minimum hamming distance between a pair of distance code word.

So, is in that is in something like that also true here any answers absolutely convolutional codes are very much like linear block codes and their exactly linear block codes when you actually terminate the trellis, so how do you actually find the minimum distance of the convolutional code well these are linear code so finding the minimum distance of the convolutional code is exactly the same is finding the minimum hamming weight of a non zero code word, but the code. But it turns out that these easiest done in terms of generating function.

(Refer Slide Time: 51:08)



So, that is what we are actually going to do next. So I will begin and I will continue these in the next lecture. I will begin by redrawing are finite state machine for the convolutional code. Here are finite state machines for an encoder and let me just select the page, copy the page, moves on the current lecture and paste it again. So now, this is then are finite state machine description from earlier. What we want to do is to use this to enumerate all code words in the trellis. We want to use this enumerate all code words in the trellis of infinite length and it is not exactly set of all code words it turns out to be the set of all code words of the interest. That a miner shuttle point so, we want to do a some enumeration and we like to make use of the finite state machine for that purpose and will use generating function technique.

So I will just begin with the certain step and then will continue in the next class here, we have outputs the outputs associated with each of the transitions and, we know the inputs are either 0 or 1 depending upon the line is dotted or not. So I am going to associate to the transition like this a term L and with the term like this for the input is 1 the output is hamming weight 2 I am going to associate L I D squared. So, the exponent of L tells me what the length of the sequences and the length of the sequences always going to be 1 this is measure in terms of number of input symbols the exponent of I is how many information symbols by 1 during that. So this is 1 that is side to 1 here it is 0 so that is side to the 0 and the exponent of these hamming weight of the output since, this is term has hamming weight 2 you have D squared over here since, this is hamming weight 0.

You have D to the 0. So with that we can now, continue labeling the others. So this is all of them will have an I to the 0 and D squared, this is L D, this is L I D, this is also L I D, this 1 is L I, this 1 over here is L D and I think, we were labeled all of it in this way. So, now we are going to use generated function technique should actually count all the paths that start from a all 0 state and go through and journey to this finite state machine and return to the all 0 state for the first time to like to enumerate those paths and in terms of these trellis background here. What we trying do is we are trying to count the number of paths that originate from this start node and which end up at one of these all 0 nodes without revisiting it in between, so paths of the kind that were interested in for example, is this path this path which goes down like this in merges with this or a path like this which goes down like this goes up, goes down and then merges like this.

So, interested in all such path and we want to enumerate them in the sense that for each such path we want to keep track of not just the length of the path but also the number of information symbols that was one along the path and hamming weight of the output and that will turn around to used. So, I think you are almost out of time here. So this I just to recap what we did today is a we continued are discussed convolutional codes and a after the short node having to deal do with loss in rate of a convolutional code with whether are trailing zeros. We talk to about catastrophic error propagation we wrote down and necessary and sufficient condition for a rate 1 by n convolutional code to avoid catastrophic encoded to avoid catastrophic error propagation and now, we are beginning some enumeration of paths or code words and will continue that in next class.