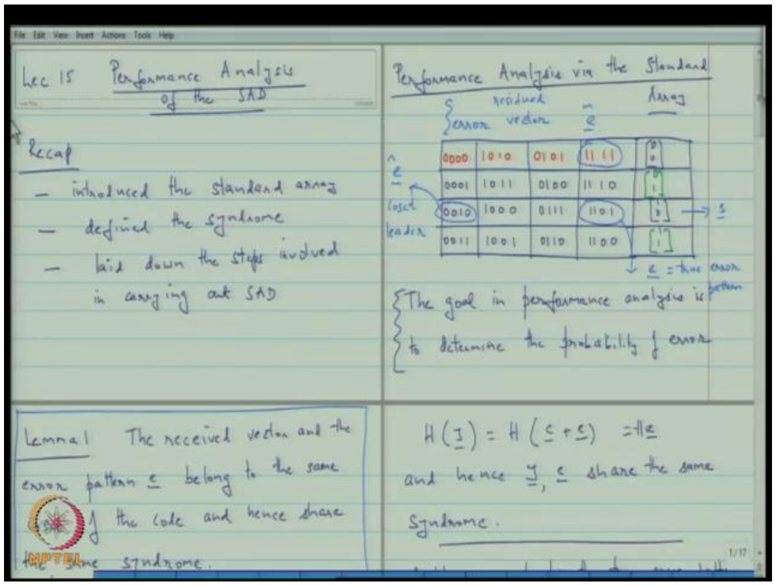
Error Correcting Codes
Dr. P. Vijay Kumar
Electrical Communication Engineering
Indian Institute of Science, Bangalore

Lecture No. #16
State and Trellis

Let us continue our discussions in the last lecture; we were talking about, we began by talking about the performance analysis of the standard array decoder, which I have also referred to as the syndrome decoder. And then after finishing that, we started a new topic, and we were dealing with the new classes of codes; these class of codes are called convolutional codes. And basically, what we were going to do in the in the beginning is just look at different ways of describing convolution codes.

And then after that, we will settle upon one view point; and I am doing this all initially through a simple example, which I have pointed out as the prototype example. And I am going to run the description right into the decoder. So, the sequence in which we describe convolution code is little different from block codes, but perhaps is the more natural sequence for a convolutional codes. Let us just quickly run through a last times lecture here on the pad.

(Refer Slide Time: 01:21)



Lec 15 Performance Analysis of the SAD

Recap
- introduced the standard array
- defined the syndrome
- laid down the steps involved in carrying out SAD

Lemma 1 The received vector and the error pattern $\underline{e}$ belong to the same coset of the code and hence share the same syndrome.

Performance Analysis via the Standard Array

residual error vector $\underline{\hat{e}}$

0000	1010	0101	1111	0 0
0001	1011	0100	1110	[illegible]
0010	1000	0111	1101	[illegible]
0011	1001	0110	1100	[illegible]

$\underline{\hat{e}}$ coset leader

$\rightarrow \underline{s}$

$\underline{e}$ = true error pattern

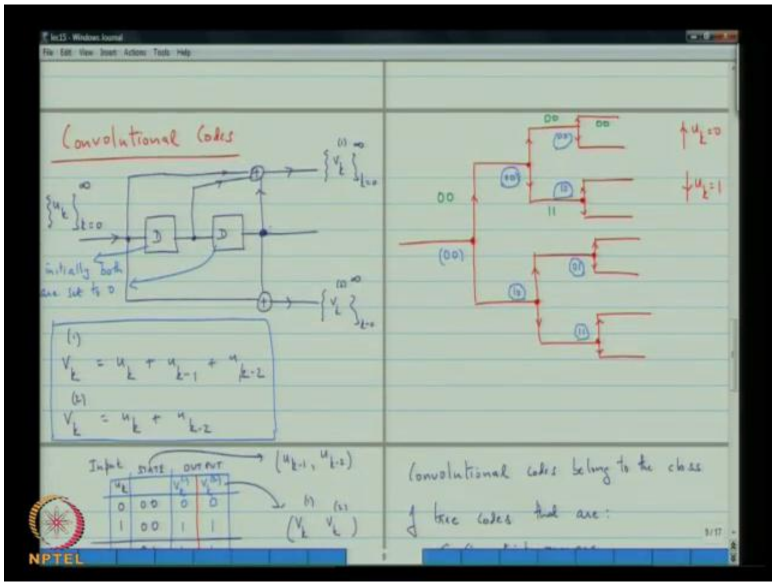
The goal in performance analysis is to determine the probability of error

$H(\underline{y}) = H(\underline{c}+\underline{e}) = H\underline{e}$

and hence $\underline{y}, \underline{e}$ share the same syndrome.

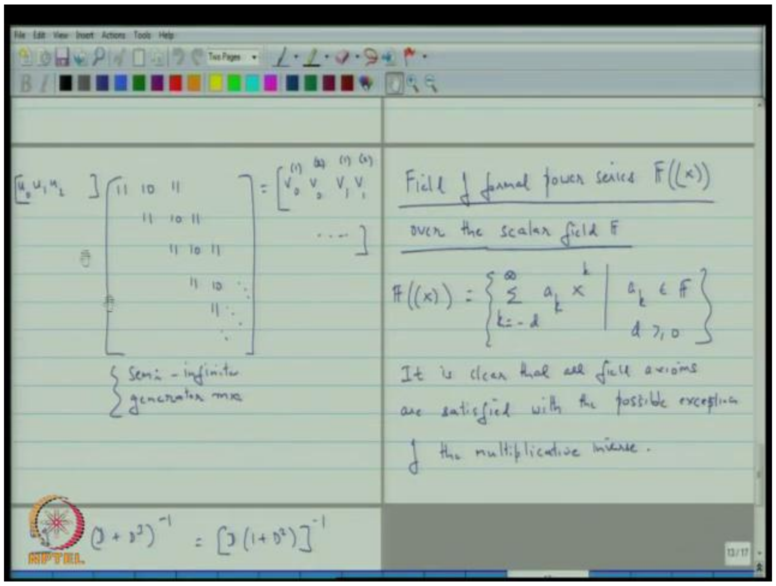
So, this started by looking at performance analysis of the standard array decoder. And which meant that we were interested in knowing how, it performance in terms of bit error and code word error probabilities; and we also wanted to know, if I have this standard array, in front of me, how can I actually compute these quantities; for any given code; and we handle that.

(Refer Slide Time: 01:47)


Convolutional Codes
initially both are set to 0
$v_k^{(1)} = u_k + u_{k-1} + u_{k-2}$
$v_k^{(2)} = u_k + u_{k-2}$
Input
State
Output
(u_{k-1}, u_{k-2})
$(v_k^{(1)} \; v_k^{(2)})$
$u_k = 0$
$u_k = 1$
Convolutional codes belong to the class of tree codes that are:
NPTEL

Then the next topic that we went on to is convolution codes, and these codes are codes that have description in terms of an encoder of this type. And the encoders of this type and we pointed out that this is properly viewed as belonging to the class of tree codes; and after talking about tree codes we actually said, we took care of some details here, and we said that amongst the class of tree codes, they have finite memory, linear and time invariant.

(Refer Slide Time: 02:30)



$$[u_0, u_1, u_2 \ \ ] \begin{bmatrix} 11 & 10 & 11 & & & & \\ & 11 & 10 & 11 & & & \\ & & 11 & 10 & 11 & & \\ & & & 11 & 10 & \ddots & \\ & & & & 11 & \ddots & \\ & & & & & \ddots & \end{bmatrix} = [v_0^{(1)} v_0^{(2)} \ v_1^{(1)} v_1^{(2)} \ \ldots]$$

{ Semi-infinite generator mx

$(D + D^2)^{-1} = [D(1+D^2)]^{-1}$

Field of formal power series $\mathbb{F}((x))$

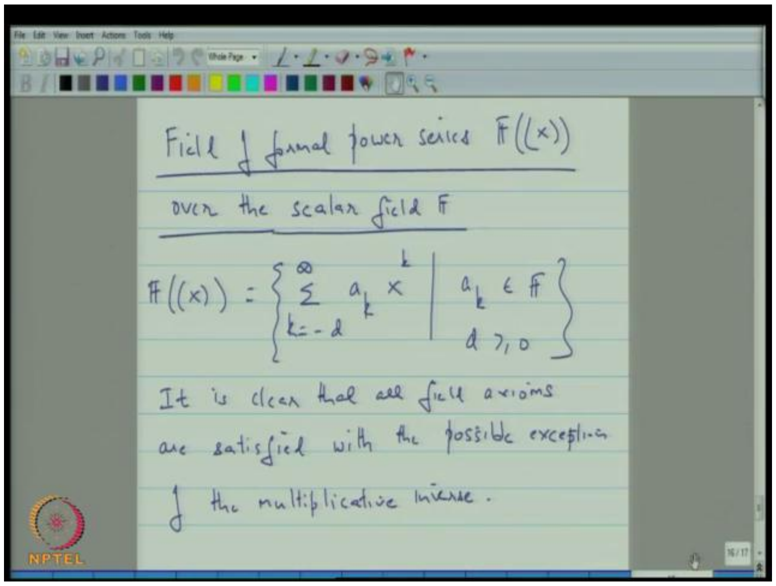
over the scalar field $\mathbb{F}$

$$\mathbb{F}((x)) := \left\{ \sum_{k=-d}^{\infty} a_k x^k \;\middle|\; \begin{matrix} a_k \in \mathbb{F} \\ d \geq 0 \end{matrix} \right\}$$

It is clear that all field axioms are satisfied with the possible exception of the multiplicative inverse.

And then, I said well, you could also talk about the generated matrix of a convolution code in the same way as you could, in the case of the block code except the returns out to be not very useful, because it is inconvenient to actually use it, because it is a (( )) infinite. What is more natural is polynomial slash power series view point, and we were getting into that. So, towards that, I started defining, what it means to talk about the field of formal power series?

(Refer Slide Time: 03:17)



Field of formal power series $\mathbb{F}((x))$

over the scalar field $\mathbb{F}$

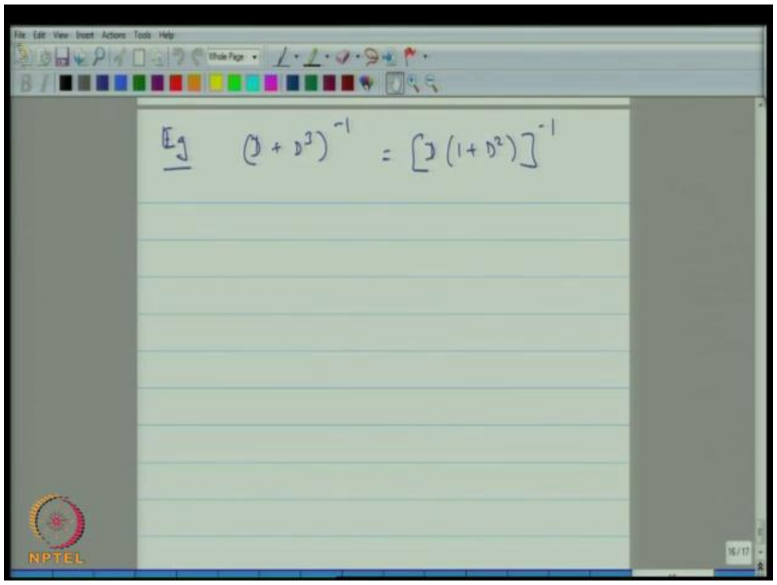
$$\mathbb{F}((x)) := \left\{ \sum_{k=-d}^{\infty} a_k x^k \;\middle|\; \begin{matrix} a_k \in \mathbb{F} \\ d \geq 0 \end{matrix} \right\}$$

It is clear that all field axioms are satisfied with the possible exception of the multiplicative inverse.

The field of formal power series is the collection of all objects of the following form, you are taking expressions of this form, and when you locate this, you should not start worrying about convergence, because in algebra, the word formal means, we just going to treat it as an object and we are going to say, you can manipulate the subject by adding objects and multiplying objects in a certain way, and that is enough for purposes.

We are not really going to evaluate it is not yet. So, you do not have to worry about convergence at this point. And the other thing to note, when you look at this expression is that on the one extreme, the exponents go to infinity, and the other extreme, they are, you are permitted to have a finite number of negative exponents. So, that is what the symbols d here represent.

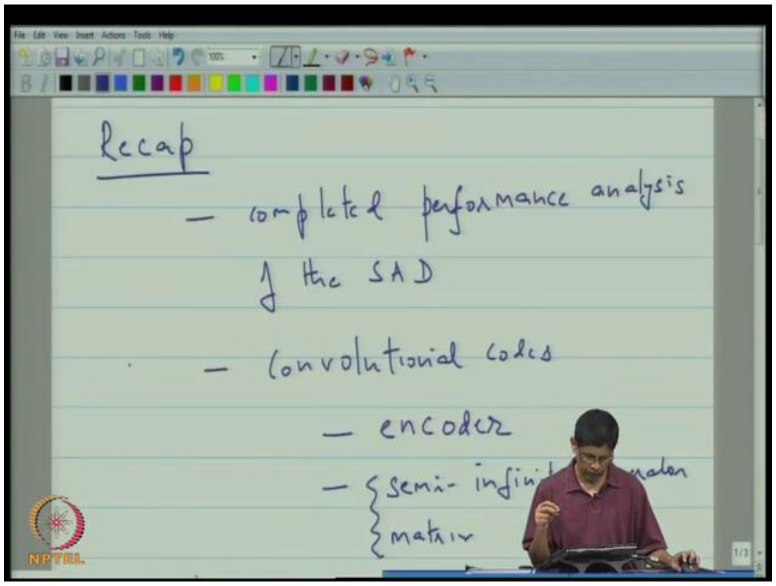
(Refer Slide Time: 04:00)



Eg $(D + D^3)^{-1} = [D(1 + D^2)]^{-1}$

And then, I meant made the common there all finite, they all field assumptions satisfied with the possible exception of the multiplicative inverse, because this is not quite clear, how one would actually go about computing the inverse. Let us pick up from that point, I am going to cut this page and paste it on to my current lecture; look it.

(Refer Slide Time: 04:50)

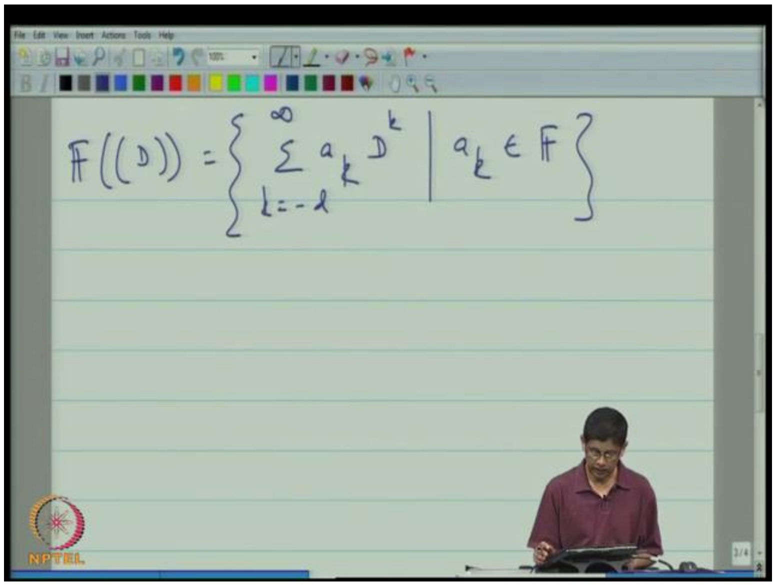


Recap

- completed performance analysis of the SAD
- Convolutional codes
  - encoder
  - semi-infinite generator matrix

For our recap, I was just say that we completed studying the performance analysis of the standard array decoder, then we moved on to study convolution codes and here, I introduced the encoder; I introduced, I introduced the semi infinite generated matrix. And then I believe that those all that, we have looked at; and towards the end, I said that you really need to know, you need to study formal power series, and polynomials.

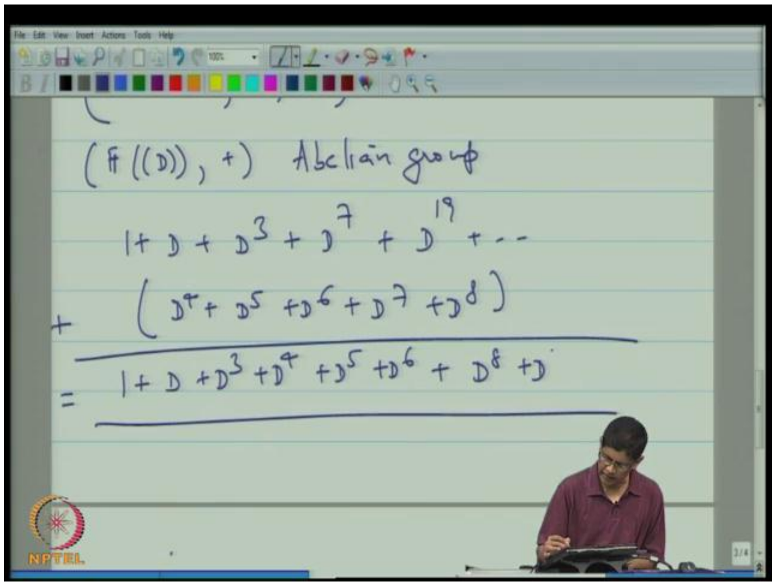
(Refer Slide Time: 06:38)



$$\mathbb{F}((D)) = \left\{ \sum_{k=-l}^{\infty} a_k D^k \;\middle|\; a_k \in \mathbb{F} \right\}$$

And the field of formal power series in D is the set of all expressions of the form, k is goes from minus D to infinity is a k D to the k, where the sub k are drawn from a field. Perhaps, I will be consistent and use this same symbol. And I said that if you want to check whether this is a field, it is fairly easy to check that the new, locate this together with plus in this.

(Refer Slide Time: 07:23)

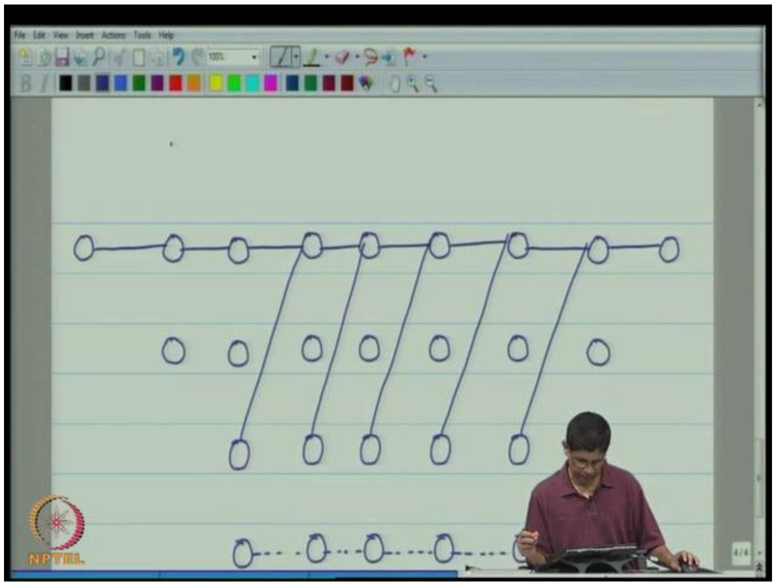


$(\mathbb{F}((D)), +)$ Abelian group

$$
\begin{array}{ll}
 & 1 + D + D^3 + D^7 + D^{19} + \cdots \\
+ & (D^4 + D^5 + D^6 + D^7 + D^8) \\
\hline
= & 1 + D + D^3 + D^4 + D^5 + D^6 + D^8 + D
\end{array}
$$

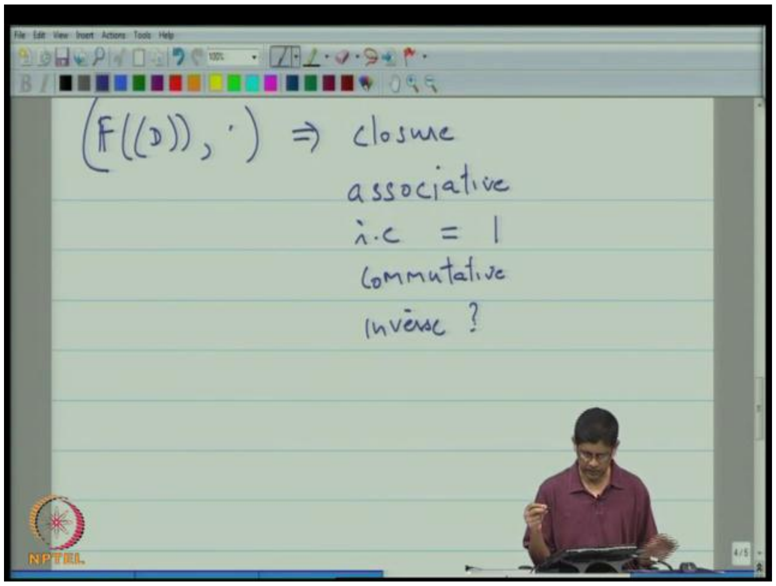
It is easy to check that under plus is an Abelian group. How do you define addition, because supposing for example, I want to add 1 plus D plus D cubed plus D to the 7 plus D to the 19 and so on, and I wanted it to add this, let us say D 4 plus D 5 plus D 6 plus D 7 plus D 8 then, you direct in the usual way eradicate term by term. So, the result could be, the result of adding this to this could be 1 plus D plus D cubed plus D 4 plus D 5 plus D 6 plus D 8 plus D 19 and so on. Because it is so happens that the D 7 will actually cancel. So, addition it is very straight forward.

(Refer Slide Time: 08:32)



You would add then just as you would add polynomials; and multiplication is not very difficult there. So, what you see here in this figure is the preview of something to come, so let me just differ that.

(Refer Slide Time: 08:53)



$(\mathbb{F}((D)), \cdot) \Rightarrow$ closure

associative

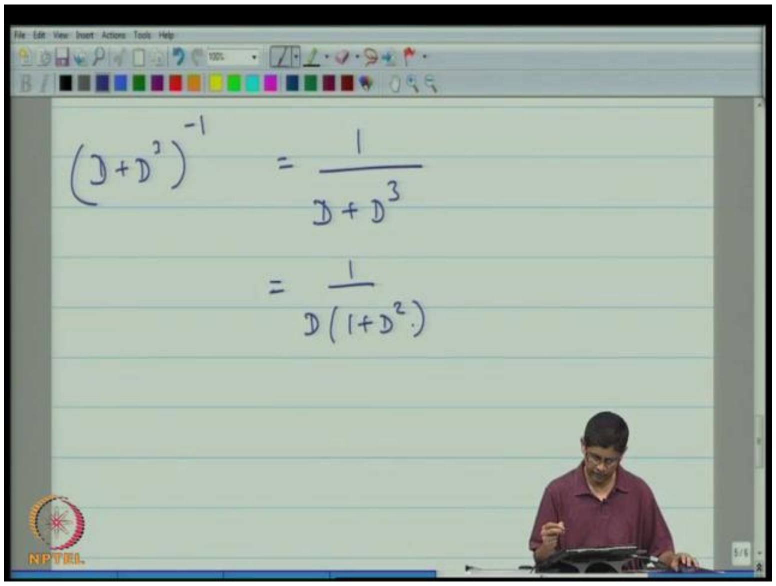
i.e = 1

commutative

inverse ?

I am going to introduce a blank page here, you can also check that as far as multiplication goes, that under this operation, this actually satisfies the axioms of closure and the multiplication; and that multiplication is associative, and that there is an identity element namely, which is the formal passive is one, and then multiplication is also commutative. The only question that you might have in your mind is, what about inverses? And towards that little just take an example, and show you, how one would actually compute inverses in this cases.

(Refer Slide Time: 10:00)

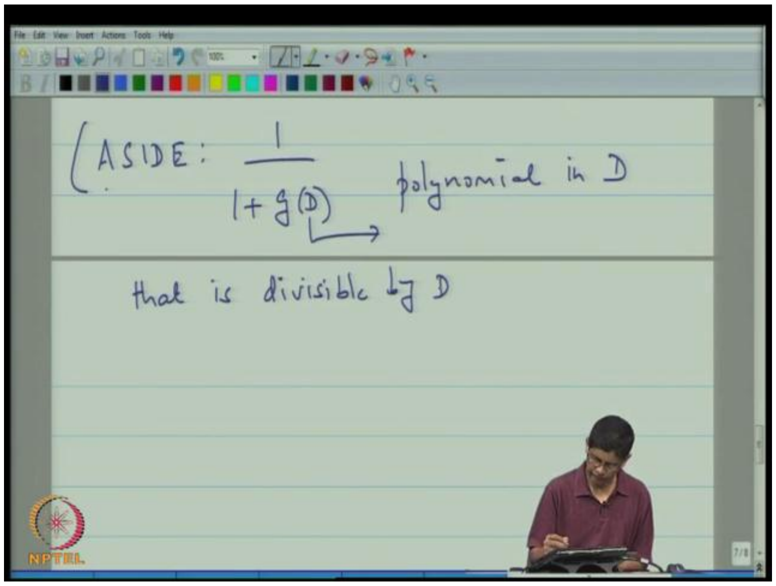


$$(D + D^2)^{-1} = \frac{1}{D + D^3}$$

$$= \frac{1}{D(1+D^2)}$$

Supposing you have to compute, let us say that you needed to compute D plus D squared, then you want to take inverse of that. You can write this as 1 upon D plus D cubed, which you can write as 1 upon D and 1 plus D square; and this 1 upon D becomes D inverse divided by 1 plus D squared.

(Refer Slide Time: 11:00)

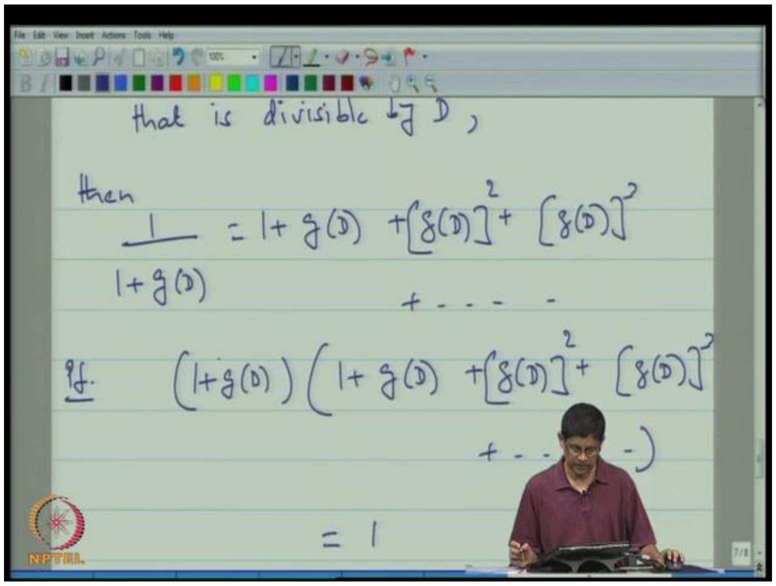


(ASIDE: $\frac{1}{1+g(D)}$ $\xrightarrow{}$ polynomial in D

that is divisible by D

And now, I will just tell you as in a side, I just tell you as an aside that that any time, you have an expression of the form 1 plus g of D where this thing here is a polynomial in D that is, divisible by D.

(Refer Slide Time: 11:20)



that is divisible by D,

then

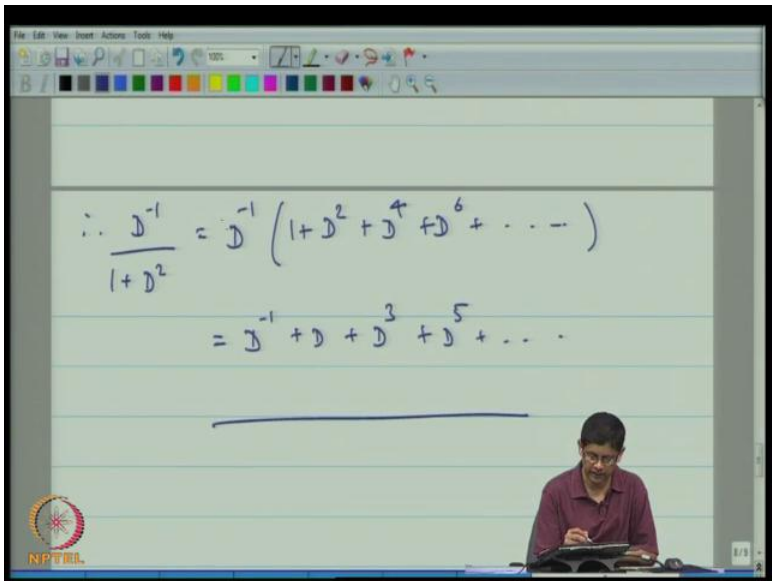
$$\frac{1}{1+g(D)} = 1 + g(D) + [g(D)]^2 + [g(D)]^3 + \cdots$$

Pf. $(1+g(D))\left(1 + g(D) + [g(D)]^2 + [g(D)]^3 + \cdots\right)$

$= 1$

Then you can write 1 upon 1 plus g of D as 1 plus g of D plus g of D the whole squared plus g of D the whole cube and so on, going up to infinity; and you might ask, why is that the case? Well, just because if you multiply; this by this you will actually get 1. The proof in case, our case is 1 plus g of D into now, let me, again take use of our technology here, put this whole thing here, put it down here. If take this, I have multiply by 1 plus g of D, then you see that there is lot of cancelations and you will actually be left with one again, because we will get a g of D term, when 1 multiply, g of D. It also gets it g of D term and g of D multiplies 1 and the 2 will cancel and it will happen for all this excessive terms. So, the distance equal to 1 enhance you actually, computed; 1 upon 1 plus g of D inverse; so getting that to our situations here.

(Refer Slide Time: 13:29)

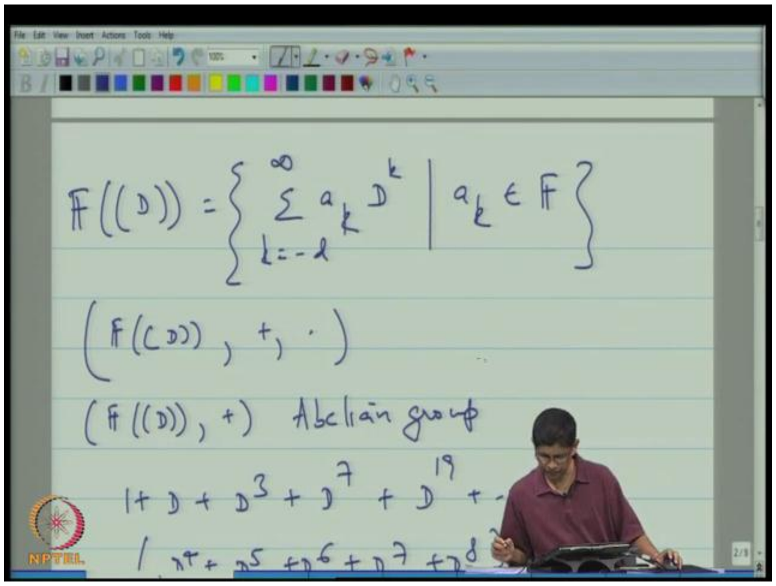


$$\therefore \frac{D^{-1}}{1+D^2} = D^{-1}\left(1 + D^2 + D^4 + D^6 + \cdots\right)$$

$$= D^{-1} + D + D^3 + D^5 + \cdots$$

Let me put the brackets, so, this was an aside all of this. Now, we were interested in computing D inverse upon 1 plus D square. Therefore, D inverse upon 1 plus D square is equal to D inverse into 1 plus D square plus D to the 4 plus D to the 6 on up to infinity. So, this is, D inverse plus D plus D cube plus D 5 and so on. So, that is have you compute inverses. Now, you see that it could have been impossible to compute the inverse. If you were not allowed a finite number of these negative exponents that, explains by you have D to the minus 1 here.

(Refer Slide Time: 14:26)



$$\mathbb{F}((D)) = \left\{ \sum_{k=-l}^{\infty} a_k D^k \;\middle|\; a_k \in \mathbb{F} \right\}$$

$$(\mathbb{F}((D)), +, \cdot)$$

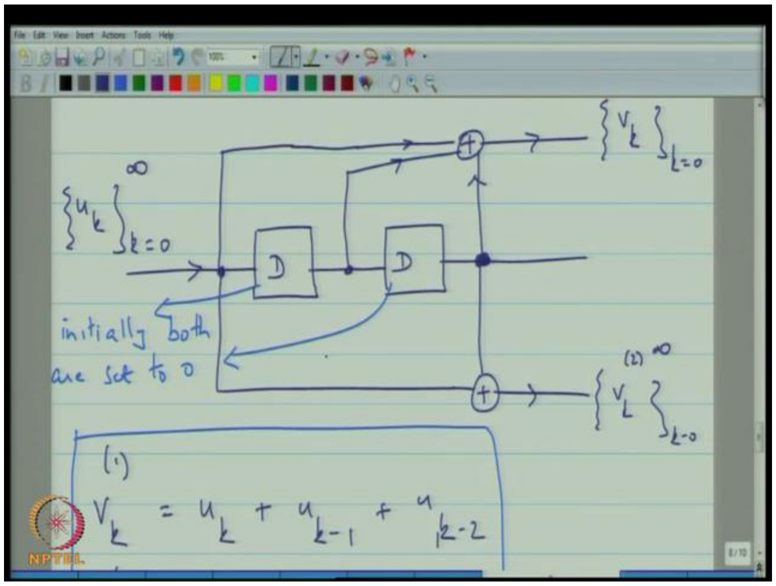
$(\mathbb{F}((D)), +)$ Abelian group

$$1 + D + D^3 + D^7 + D^{19} + \cdots$$

$$1 \quad D^4 + D^5 + D^6 + D^7 + D^8$$

Now, hopefully with that, we are convenience that, in fact you are convenience that, this set to be here, the field of formal passage this is in fact field. Now, what we are leading at through is an alternative description of the code. Then, we just, if can take care of that I taken the process; I have added a few blank pages. Let me see, if I can get rid of them. I think, I have what I want. For the encoder that we have to look at previously and I think, I may want to bring that back from an earlier lecture. Let us see, if we can get back to our earlier lecture here, yes here, we go so, I like to bring back this particular page.

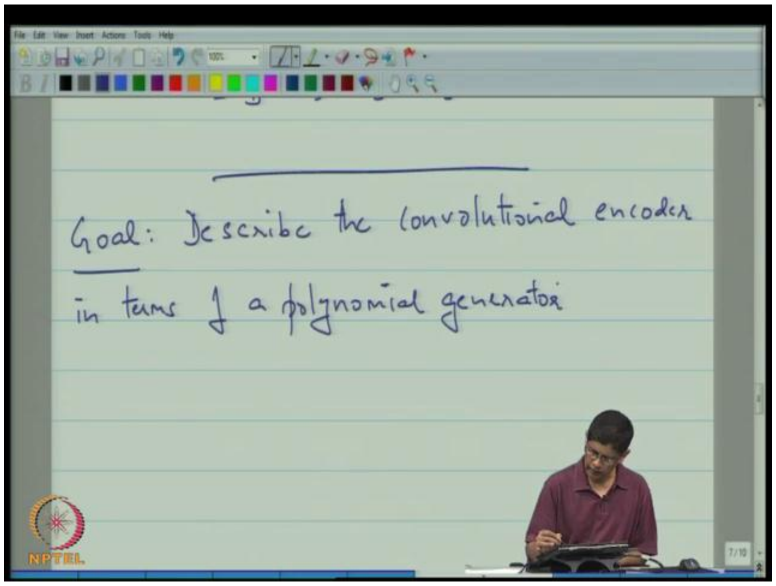
(Refer Slide Time: 16:50)



$$v_k^{(1)} = u_k + u_{k-1} + u_{k-2}$$

Let us, cut and paste that, here we have our encoder that, we had looked at last time and what I want to do is, I would like to describe this in, terms of polynomials, this is our encoder. Just, so, that you refresh your memory is one input; this one input over here, and there are two outputs and the input output relationship can be written like this.

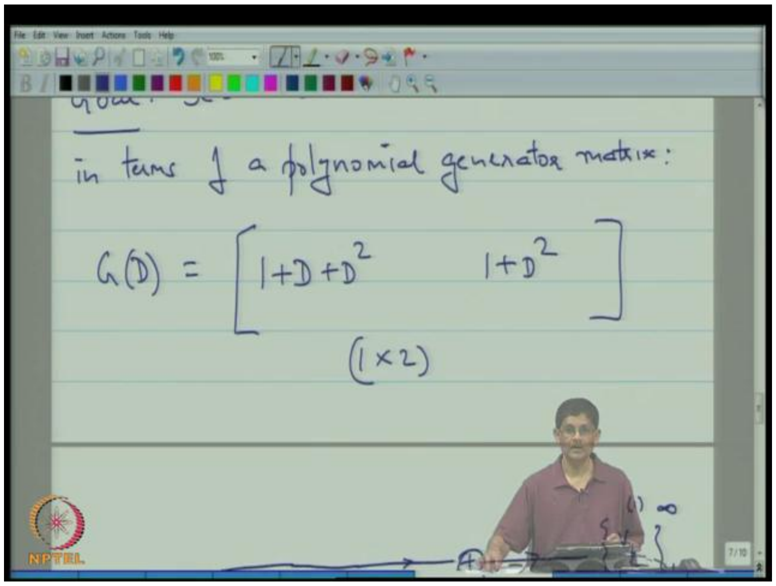
(Refer Slide Time: 17:12)



Goal: Describe the convolutional encoder in terms of a polynomial generator

And I am going to introduce through power series. So, our goal is describe the convolution encoder in terms of a polynomial generator matrix, and what I exact to say is that in fact, it is possible to describe the convolution code.

(Refer Slide Time: 18:00)

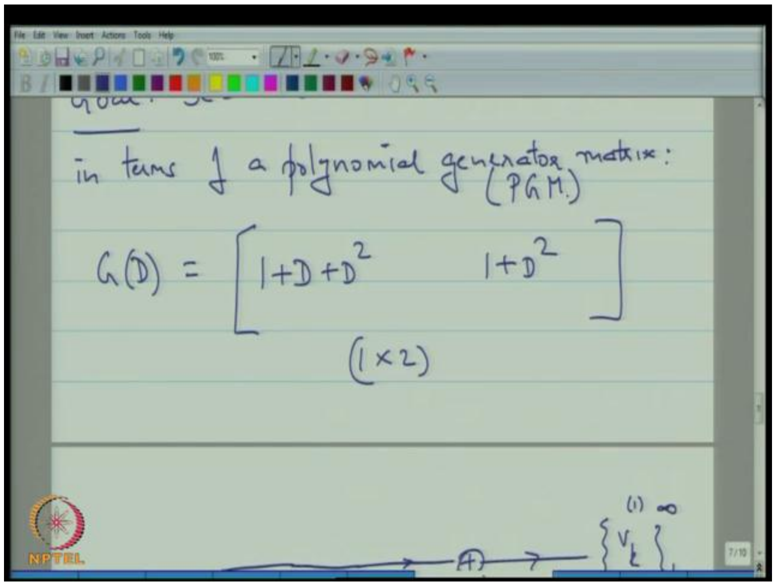

in terms of a polynomial generator matrix:

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

(1×2)

In this way, just say that it corresponds to the following polynomial generated matrix 1 plus D plus D squared 1 plus D squared. This is a 1 by 2 matrix; you might be a little bit surprise by that, what you mean by matrix? And you are putting terms 1 plus D plus D square and if you suppose support real number in a matrix, but actually, you can define matrix is in our any field. There is you can have a matrix and you can put numbers from many field in matrix. They are typically; we use to either putting real number or complex numbers. Especially, when we are studying linear algebra, but actually, you can put element of any field. Here in fact, not available on sub even beyond because we are saying; we actually going to put down the elements in the matrix that come not even from a field, but actually, from the collection of polynomials. The polynomials actually form a ring and we saw that earlier, when we discussing ring. We are taking matrices is over the ring, but that is fine. Are you all your matrix operations are valid?

(Refer Slide Time: 19:07)



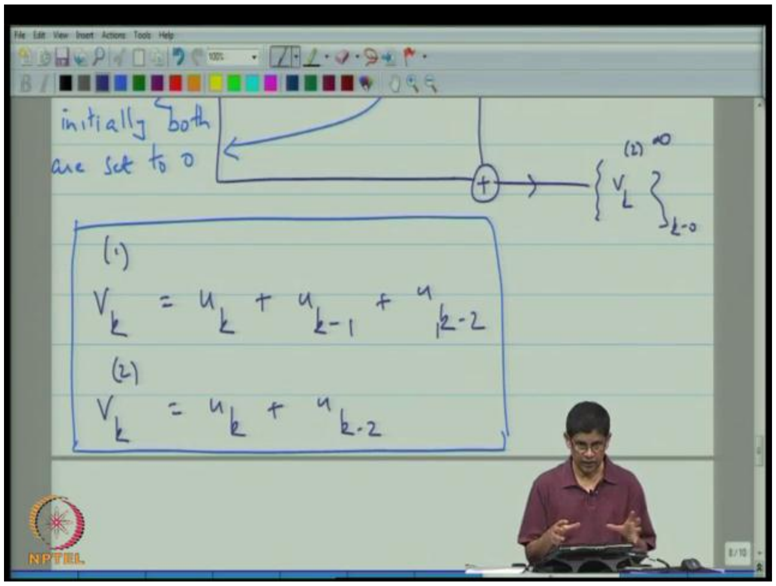
in terms of a polynomial generator matrix: (PGM)

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

(1×2)

This is your polynomial generated matrix and we will typically; abbreviate this and say, in differ to it is PGM. So, this is the PGM for the code, how do you actually, arrive at that? What is it mean exactly? To see that, you will have to look closely at this equations.

(Refer Slide Time: 19:27)

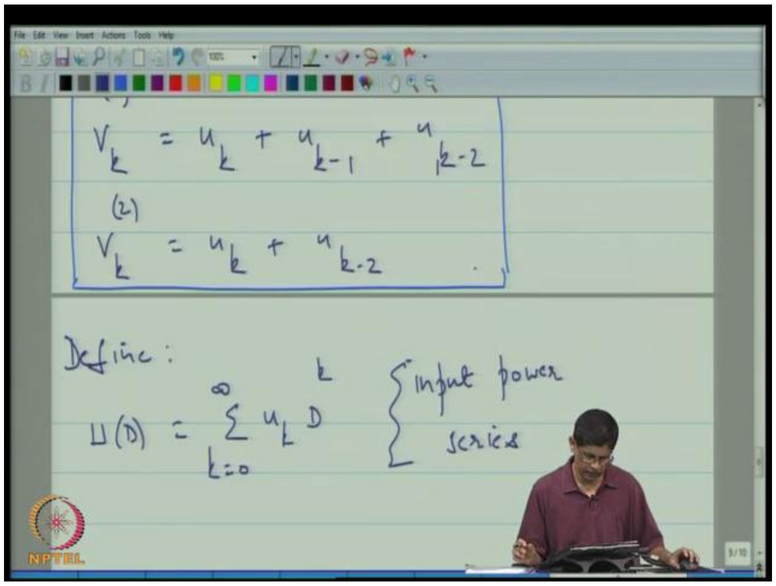


$$v_k^{(1)} = u_k + u_{k-1} + u_{k-2}$$

$$v_k^{(2)} = u_k + u_{k-2}$$

And if you look at it closely enough, it looks like you are actually; convolving the input to the certain sequence of finite length and that is, what is giving you the output? And as you know, if you have a convolution in the time-domain, you can represent it, as a multiplication in the transformed domain. Let for example, the g transforms and your D is very similar; to explain the role that z place in this z transform.

(Refer Slide Time: 20:00)



$$V_k = u_k + u_{k-1} + u_{k-2}$$

(2)

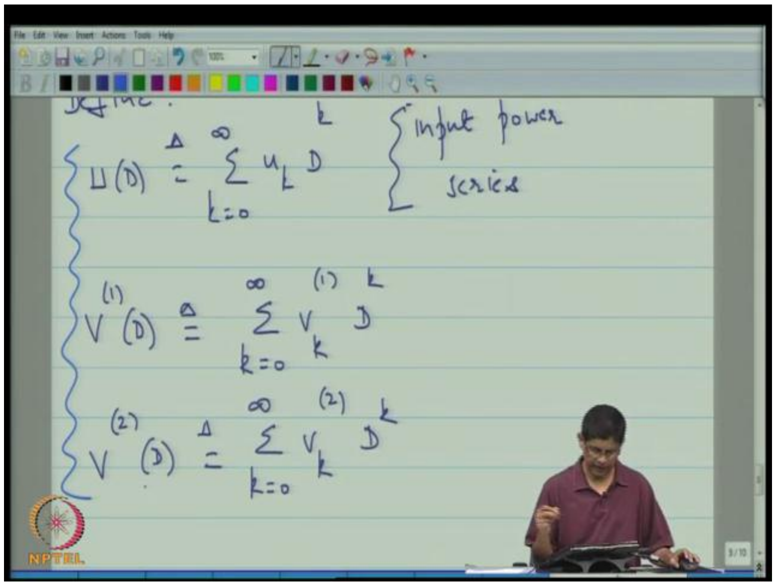
$$V_k = u_k + u_{k-2}$$

Define :

$$U(D) = \sum_{k=0}^{\infty} u_k D^k \quad \left\{ \text{input power series} \right.$$

Let us, define, to just make that little bit of formal. We will define, U of D to be the sum u k D k, k goes from 0 to infinity. This is the power series, associated with the input to their convolution code. We can think of this, if we like, as the input power series and now, follows output go there actually; two outputs.

(Refer Slide Time: 20:38)



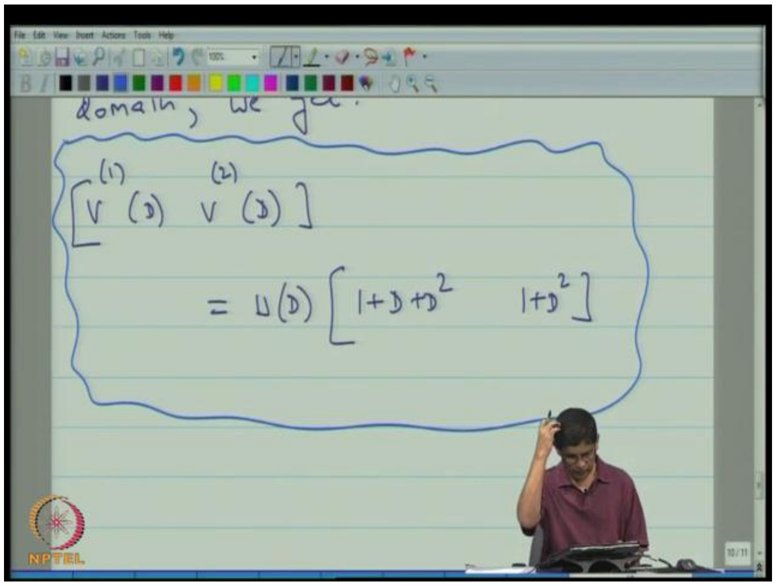
$$U(D) \overset{\Delta}{=} \sum_{k=0}^{\infty} u_k D^k \quad \Big\{ \text{input power series}$$

$$V^{(1)}(D) \overset{\Delta}{=} \sum_{k=0}^{\infty} V_k^{(1)} D^k$$

$$V^{(2)}(D) \overset{\Delta}{=} \sum_{k=0}^{\infty} V_k^{(2)} D^k$$

And we will associate the power series with each one of them. So, V 1 of D is define all of these are definitions, define to be the sum k goes from 0 to infinity, V k 1 D to the k and then, we have V 2 D. We have our three power series define and now, it is merely; a matter of converting the time-domain expressions 2, let us, call this. We will say that, converting the time-domain input through, let us call this.

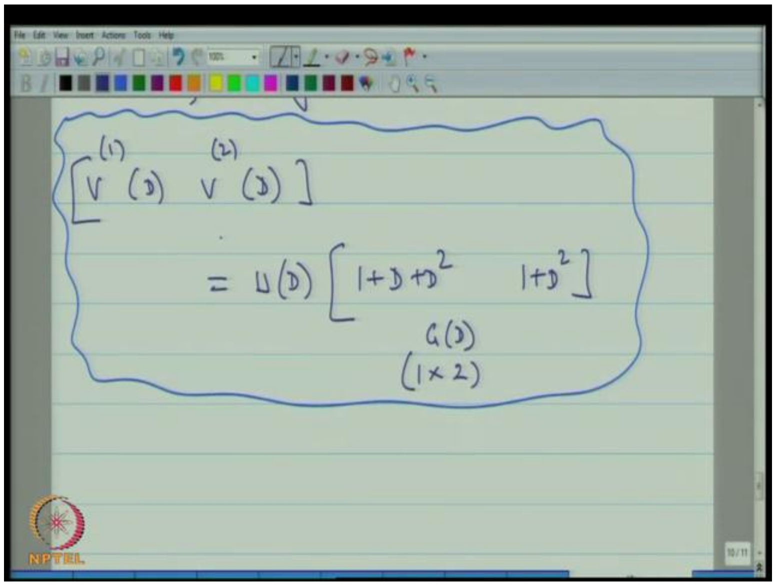
(Refer Slide Time: 21:45)



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

We will say that, converting the time-domain input output- relation given in 1 to the D to the D transform domain. We get that, V 1 of D v 2 of D is equal to is equal to U of D 1 plus D plus D squared 1 plus D squared and this is what? We called our polynomial generate matrix. Now, this equation is similar; in a sense to the equation, that describes the relationship between the message symbols and the code symbols in a case of the linear block code because after all, this code is the rate through the encode, were case one and m is two.

(Refer Slide Time: 23:50)

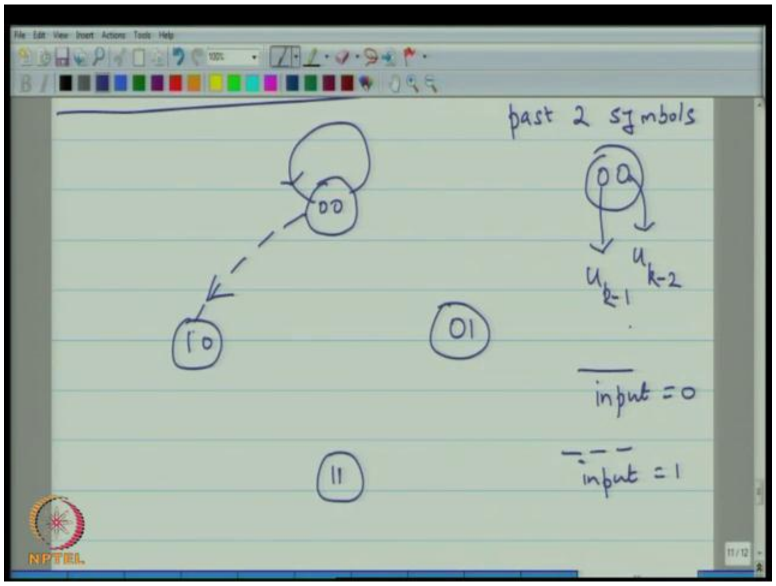


$$\boxed{\begin{aligned}&\left[V^{(1)}(D)\quad V^{(2)}(D)\right]\\ &\qquad = U(D)\underbrace{\left[1+D+D^2 \qquad 1+D^2\right]}_{\substack{G(D)\\(1\times 2)}}\end{aligned}}$$

You can see for this reason, this is a 1 by 2; it is a 1 by 2 matrix. You have the n in some sense whereas, in the of block code you have n output symbols; you have n output strings and k input strings. You can see, why? One would want to refer to this as a polynomial generated matrix? Now exactly, how do you go from the time-domain expression to the frequency domain?

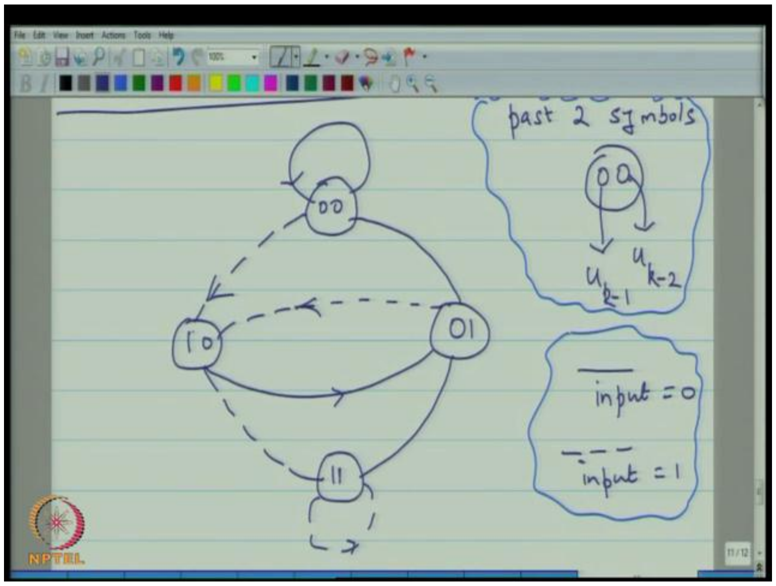
I am sure; most of you can actually; fill it out on around, given your back ground in transform theory. That will actually consider the general case a little later. So, I am going to differ that point. Let us now let us say, I accept that this is another description of the convolution code. Another one, which might have already occurred to you, is that, you can actually describe this in terms of a finite state machine. Of course, it is a reference to the encoder. You see that, the encoder has 2 shift register and since, each shift register can store either the values 0 or 1. In a sense, this finite state machine has the memory of 4 or in other words, it has 4 states.

(Refer Slide Time: 25:24)



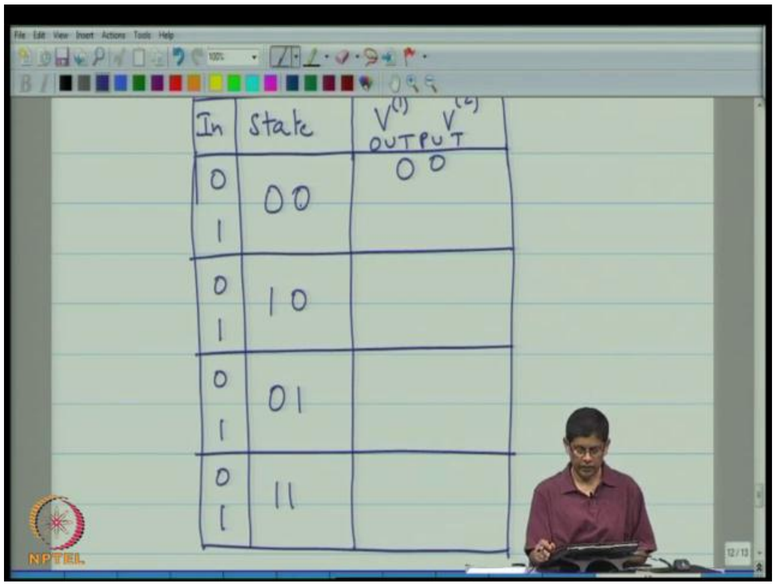
We are going to write down this 4 stage here; these are the 4 states and now, we are going to represent transitions. Now from 0, 0, what about the notations? When I write 0 0 here, what I mean is that, this is u k minus 1 and this symbol here, is u k minus 2. That is, this collection represents the previous the past 2 symbols represent the past symbols. Here, if I get 0 0 and let us say that the input is 0, then I am going to remain in this state, and on the other hand, if my input is 1, I am going to transition to 1 0. I am going to solve the convention in describing convolution codes that when the input is 1, you actually; draw solid line. If the input is 1, you put on a dotted line, this means that the input is 0 and the dotted line means we have the input is equal to 1.

(Refer Slide Time: 27:20)



Let us get back to our diagram and now from 1 0, if the input is 0, then it actually go to 0 1; if the input of 1, 1 continues here; if the input is 0 1, 1 continues remains in the same state. On the other hand, if the input is 0 you go here and so on. This is then, the final state machine. Now, but apart from this one should one would like to also require inform about; the corresponding output.

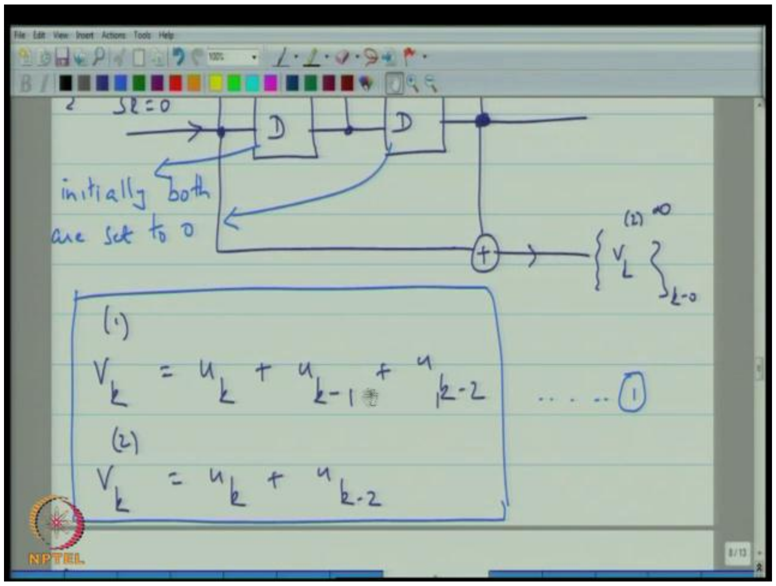
(Refer Slide Time: 28:39)



In	State	$V^{(1)}$ $V^{(2)}$ OUTPUT
0	00	0 0
1		
0	10	
1		
0	01	
1		
0	11	
1		

For that, let me do the following, I am going to put down a table here, and the table is going to have, so here, you have the input, the state and we have the output which is V 1 and V 2, which represent the output. There are four possible states that the final state machine could been 0, 0, 1 0, 0, 1, 1, 1 and in any state, there it is then input could be 0 or 1 0 or 1 0 or 1 0 or 1 and the output, if the input is 0 the state this is, 0 and the output is 0, 0 and if the input is 1, the state is 0 0. You can figure it, out because for that you have to just go back here.

(Refer Slide Time: 30:50)

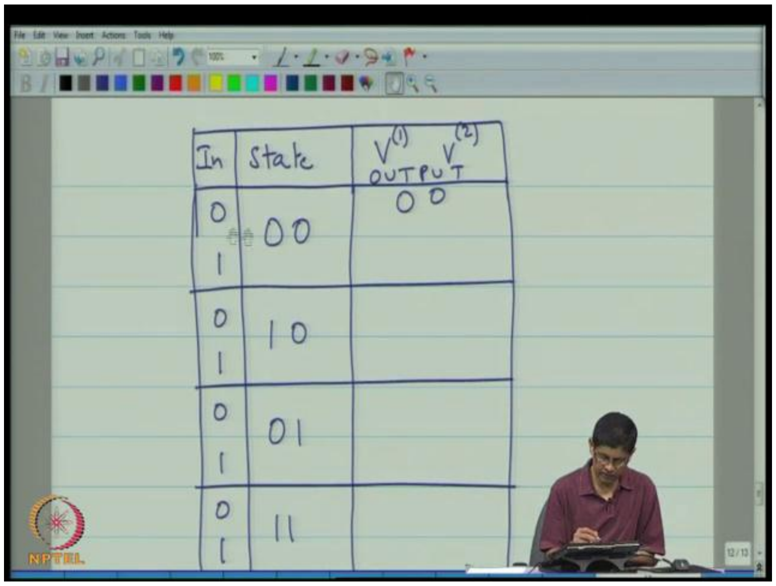


$$v_k^{(1)} = u_k + u_{k-1} + u_{k-2} \quad \ldots\ldots \boxed{1}$$

$$v_k^{(2)} = u_k + u_{k-2}$$

And just look at these expressions here, so the that means, that you are going to sum the past three symbols to actually; get the first output and the current output and the input to instance (()) to get the second.

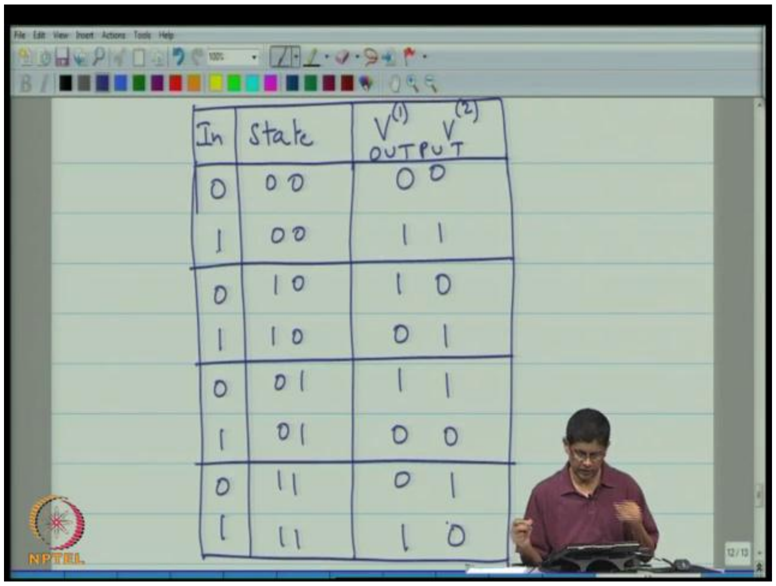
(Refer Slide Time: 31:10)



In	State	$V^{(1)}$ $V^{(2)}$ OUTPUT
0	00	0 0
1		
0	10	
1		
0	01	
1		
0	11	
1		

If we kept that in mind then since, these represent previous important the input previous to that and the first output is obtained simply by summing all the three symbols.

(Refer Slide Time: 31:17)

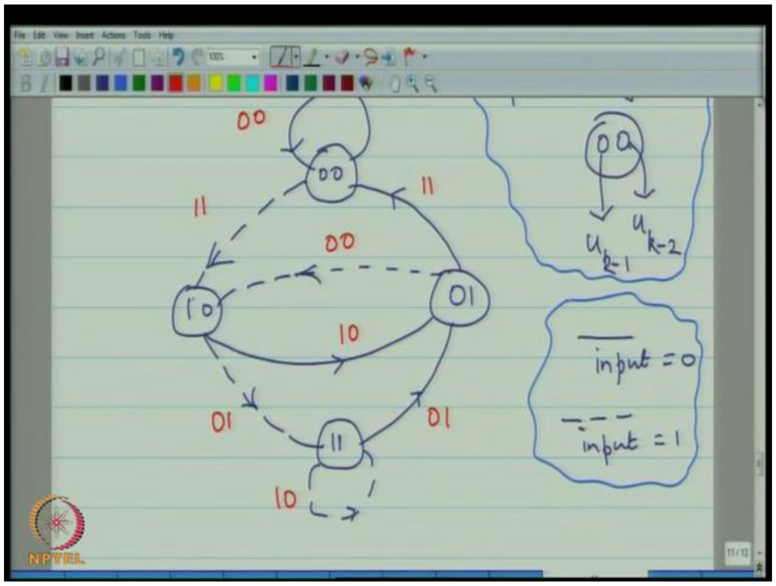


In	State	$V^{(1)}$ $V^{(2)}$ OUTPUT
0	00	0 0
1	00	1 1
0	10	1 0
1	10	0 1
0	01	1 1
1	01	0 0
0	11	0 1
1	11	1 0

In fact, let me just make this a little bit clearer and I will write down the state two times because this one is to confuse you, so this is the state is 0 0 here and so on. So, the input is 0, the state is 0 0, the output is 0 0. To get the first output, you always add the three symbols that is 1 1 0 1 0 0 1.

The second output, you actually; get adding the first and the third symbols because the current symbol and the symbol two instances tried at that. So, that in this case be 1 0 1 1 0 1 0 these are your outputs. In the finite state machine, what I like to do is I will also like to record these are so, if I just remember, what the outputs are corresponding to a state and an input I can actually; fill in and the finite state machine, that we had earlier.

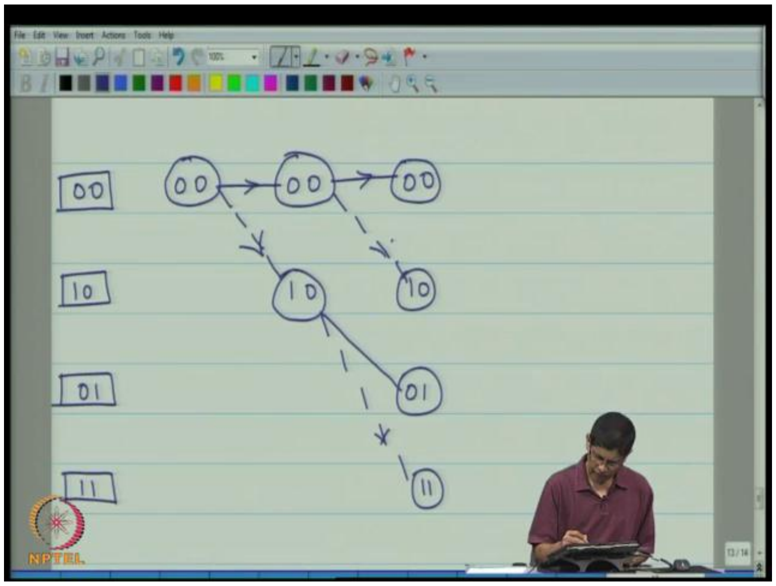
(Refer Slide Time: 32:23)



You can check, that the outputs in this case, I little bit put that in red, so that it stand out 0 0 1 1 and here, it is 1 0 0 1 1 1. This should be 0 0; this is 0 0; this is 0 1 and this is 1 0 and this is 0 1. You can check this on your own or you applying; proof at plan this table and putting in the entries from that here. Now, there is another very popular representation of the convolution code and it is really originate from the finite state machine, because what you do is? You take a finite state machine and say that I am going to make a distinction between the same states at different instance of time.

I am going to unfold the finite state machine; in some sense with respect to time and that leads to the trellis. Here, actually shown the trellis of the convolution code. I am thinking that may be, what I said do is you show how it evolves? And then, we will come back to that picture if necessary, let us say we begin at someone has 4 rows each to represents each of the possible state.

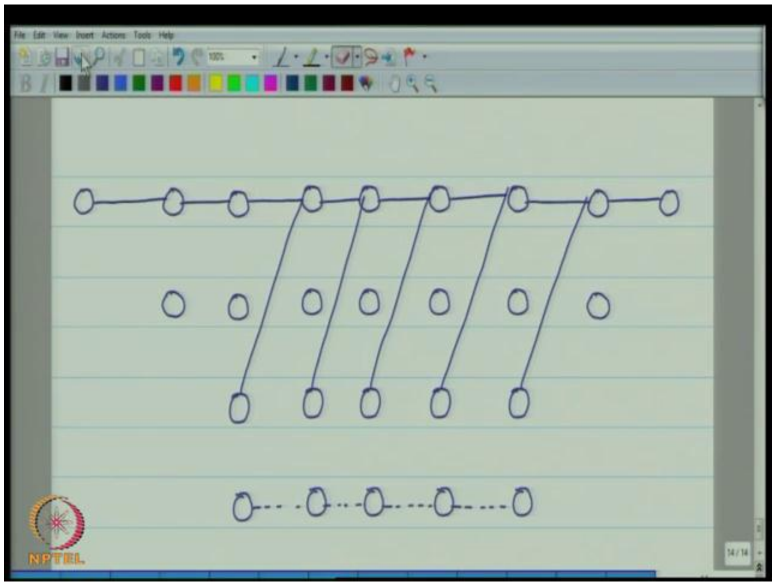
(Refer Slide Time: 34:35)



Let us get back that state, so I am going to write 0 0 here then, I am going to put down 1 0, 0 1 and 1 1. So, these are the four possible states. Let me say that I initially start out in state 0, 0 which is used in the case and then, the next instant of time. So, here is my starting state and going back to the finite state machine here, if I am in state 0 0; I can either stay in the same state or I can transition to 1 0 and so I am going to and I goes some 0 0 to 0 0 when the input is 0. So, I put that down and similarly, if the input is one I will go to 1 0 and so on.

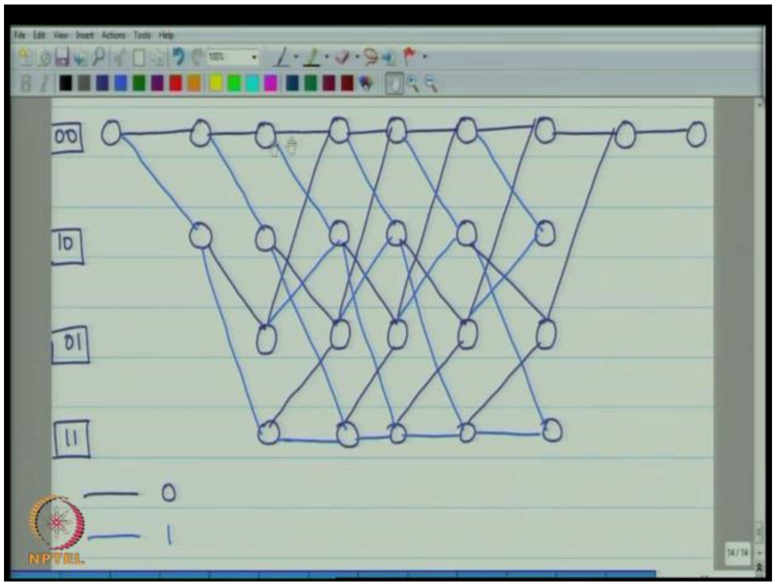
So now, I am going to complete the figure by trying in the transitions corresponding to the next input as well, when I am in state 1 0; I can either if I get a 0 you should actually; visualize 0 as coming in from the left and bumping this older symbol out. So, the 1 0, because the 0 come and shrunk this and transform this to 0 1. If a 1 concern it becomes 1 1; it will transition here, so this is the transition to 1 1 and this is what we had earlier and so on. So, this is the what the trellis is look like.

(Refer Slide Time: 37:14)



And now, I am going to bringing what I had drawn earlier its set is slightly; more advance state of completion. Let us, complete that, I will try to be careful because there are lots, of lines that we are going to have to try and try to keep in as clean as possible. One thing, I am realizing is that, this time we search for not very good drawing dotted lines and on the other hands, we have color like we actually; use color to distinguish input and output let me, try to do that instead.

(Refer Slide Time: 37:32)



Let us do, let us, use color here so, I will put, now also and it shifts this entire diagram a little bit to the right. Let us do that, perfect; so now, I have space to actually label the different states here. This will be 0 0 1 0 0 1 and 1 1. I am going to represent, let us use, this dark blue for a 0 and this light blue for a 1. I think that, might make this easier diagram to draw then, I know that, if I am in state here, and I get a one going down here, so, let me draw all of those transitions and then; if I am in state 1 0; if I get a 0, if I get a 1, I am going to transition around to 1 1; if I meant 1 0 and I get this 0 then I am I actually going to put down to 0 1.

I will take care of transitions originating from the first two. If I am instead 0 1 and I get a 0. I go there, if I get a 1 then, I am going to transition to from 0 1; if I am in a state 0 1 and a 1 comes in and going to transition to 1 0. Let us represents that, we taken care of three that leaves the one at the bottom if I am in state 1 1 the input is 0. I am going to move up to 0 1. On the other hand, if I get 1 I am going to remain in this same state. I think the only thing that we need to check is what is happening towards the end of the trellis in this region? And here, I think, I may want to make some slight changes; I would like to actually remove this states because I want and for reasons that will become clear a little bit later.

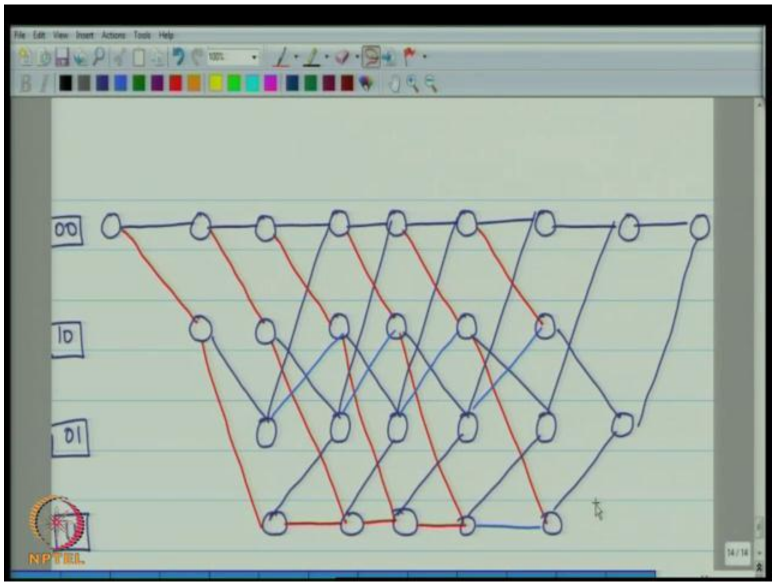
I want to start on the all 0 state, but also want to end on all 0 state. It is a lot of communions; it is not really essential. Now, you might ask why is it that you want that, it is a matter of convenience and it also makes the analysis easier. In practice, you might do things a little bit differently, but it is in it simplifies the way one things about the code so, the way you actually bring the convolutional encoder back to the all 0 state.

That is by it obvious, you look at this the generator here, and all that, you need to do is you need to bring in two training0s to bring this to the all 0 state. So, that is what we are going to be 0, what up means is that, in this trellis. What trellis is the sectional state that is the fact state machine goes through as a result of feeding all possible inputs sequences to the finite state element? In that sense, every path in this dwell is represents the possible input sequence. Now, since, the last symbols are 0 that means if it should only have block everyway over here.

I want to bring those back, the only way I can get to the all 0 state is, if in the previous state I was eager on the all 0 state itself, let us, go back to the finite state machine to make that clear. So, only way to get back to the all 0 state is, if I am in the previous instant, either in the 0 ,1 state or the 0, 0 state. I am going to make use of that, so, the 0 1 state is out here in the bottom.

That is the only state, I am going keep here because I know that, I am the input is going to force it to the 0 1 state. We will just, draw only; this state because no matter where you are here? The next, this penalty; makes state either going to be here or here because you know that, your input is going to be 0. If you are at 1 1 and you get a 0 then, you will actually; move here; if you are at 0 1 then, you will go if you are in zero one you would go here; if you are 1 0 then, you will come down lower with 1; if you are in 0 0 you stay over there and finally, over here.

(Refer Slide Time: 45:11)



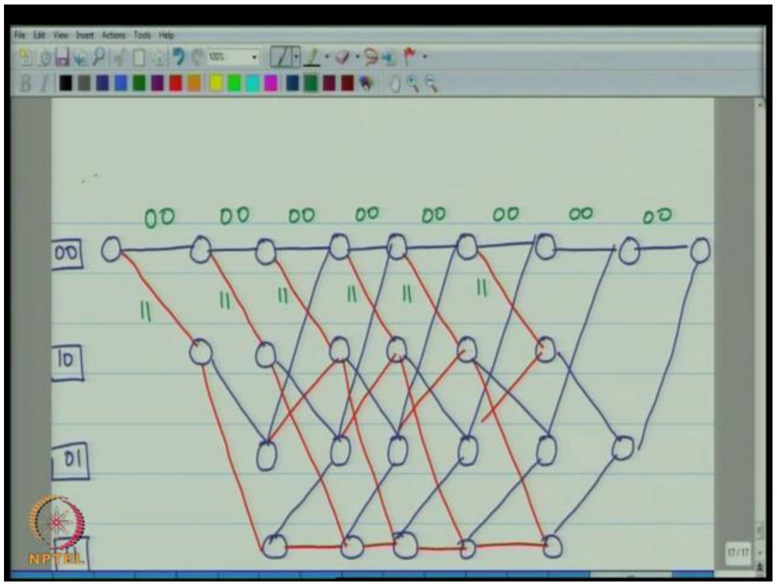
You will come here; you can see that in these last two stages, all the lines are black. I just realize that, what is happening is that, the blue and black turns show up very differently in your screen. So, maybe it is what taking a second to change all this blues to 1 that, they really stand out that you do not get confused. We will just take a second to do that, here it seems to one to select all of these at one times, maybe I should just cut this and draw this again. Let me do that, this is I guess some; I have to work little harder to change the remaining ones so let us take care of that in the old fashion way.

Now, if you done things correctly; between any two levels you must have equal number of 0 s and 1s 1, 2, 3, yes I think 1, 2, 3, 4. We have and towards the end, we are only putting in 0. I think, the trellis is just a little bitten more clearly so, what this trellis is helps you? Visualizes all possible input sequences and now, what we like to do? We like to also use it to the present output sequences.

The output sequences, we are going to take from the finite state machine and in fact, we already; started when we, let me do this on this part of the trellis is here. We know that, if you are in state 0, 0 and if your input is 0 then, your output is going to be 0, 0. Let me, also represent that, let us, see which color will look good here? Perhaps, I should let us, see, if green is visible on here screen, yes it is. We will represent the output in use green. Now, the output is 0 0 here; 0 0 here and you can verify; it 1 1 here; 1 1 here and so on.

Let us gets back to the trellis and the label all of the edges of the trellis and so, recall that a solid indigo is 0 and a solid red is input. This is at the input now, as for as the output goes and that present in that in green here. Let us show that in also and the outputs here, in fact would one thing this side filled at in the next lecture, I am going to call upon this trellis several times. I am going to actually copy this page and keep it as a spare because I am certain to need it, in our next lecture, so I am going to save that here. Let us start that again I am going select page; copy page; insert new page; paste page. Solve the trellis, the way I want it for the next time and for now, I will free to write on this version.

(Refer Slide Time: 50:50)

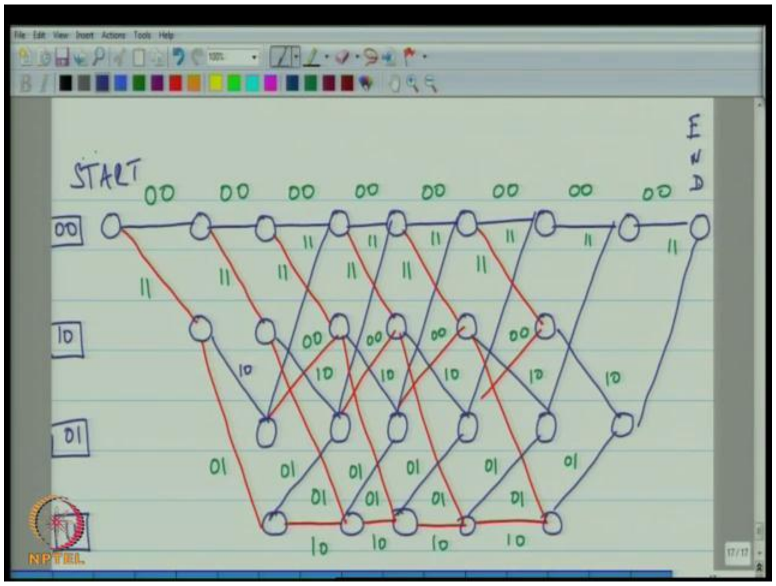


Let us, complete this now, so this is the outputs are 1, 1 from 1, 0, when your transition within input of 1 to 1 1 the output is, here we go; this is the table that; we at use, when you are in state let start remember this; when we are in state 1 0, when the input is 0 we have output is 1, 0, 1, 0 within input 0 and the output is 1 0. So, this thing here, is 1 0 and you can similarly verify, that this output here is 0 1, that when it is in state 1, 1 and the input is 0, the output is 0 1.

Here, I will have 0 1; when the input is 0? I have 0 1 so, this is 0 1 0 1 0 1 0 1 0 1, but on the other hand when the input is 1, the output is the 1 0 1 0 1 0 1 0. Let us see, I think; we from the only other, there are eight transitions within to write within a segment. We will able 1 2 3, when you are in 0 1 with 0? When you are in state 0 1 and you get a 0 and the output is 1. Let us, put that in, this is 1, 1 as well and the only other transition we need to worry about is that transition of 1 0 to 0 1. When you are in 1 0, when are in 0 1 and we get a 1 the output is 0 0 here, in 0 1 the input, the output is 0 0. So, this transition here, is 0, 0 with this I am, we are almost coming to the end of our lecture; we have another, this involved a little bit of that, but I think we will put this to good use in the lectures to come.

In this is the graphical representation of the code and what you can see in this graph? Is that, for every input sequence there is a path in the graph that leads from start to finish. Here, this is your start note; this is your end node and every code words path in this trellis. Now, if you can also see the code words which are part of the code because associate in every path, you have sequence of output symbols and that represents the code words.

(Refer Slide Time: 55:00)



For example, if the input sequences are all 0 then, the output sequences also 0s and that is your code word and then, if your input sequences are 1s, except the last two 0s then, your output sequence is 1 1, 0 1, 1 0, 1 0, 1 0, 1 0, 0 1 and 1 1 that would represent your code word. In that way, every sequence from start node to finish node, you can actually call this the start and you can call this the end or finish node.

Every part from start node to end node gives you a code word. So, this is another discussion of the trellis of the convolution code and actually, it turns out that this is the very convenient starting point for describing decoding techniques of the convolution code. In particular, most of you would have already heard of famous viterbi algorithm for decoding convolution code. Once, we look at this graph, it is very easy to actually understand the decoding algorithm.

What will do next time is actually continue this discussion on the trellis, but actually using it to decode the code. Assuming it is actually used for a binary symmetric channel; and beyond that will actually continue our discussions to first include the general class of convolution codes. And also with actually, so how convolution codes can be used over the (( )) channel as a post just the binary symmetric channel. So, at all I think time is almost up, so I will stop here. Thank you.