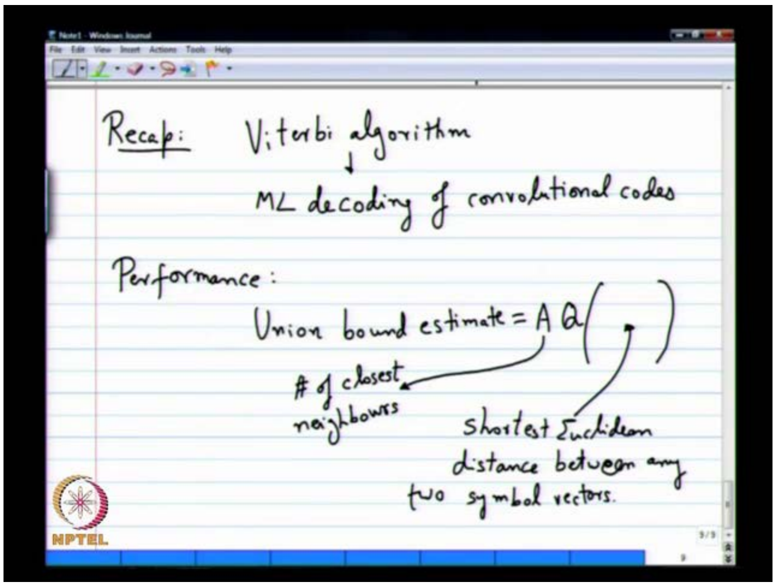
Coding Theory

Prof. Dr. Andrew Thangaraj
Department of Electronics and Communication Engineering
Indian Institute of Technology, Madras

Lecture - 32
Union Bound, Recursive Convolutional Encoders

(Refer Slide Time: 00:19)



Recap: Viterbi algorithm

$\downarrow$

ML decoding of convolutional codes

Performance:

Union bound estimate = A Q(,)

# of closest neighbours

shortest Euclidean distance between any two symbol vectors.

So, let us do a very quick recap of last lecture. Essentially, the previous lecture was the viterbi algorithm, which is soft M L decoding, let me just say M L decoding of convolutional codes. So, in fact anything that you can represent on a trellis with putting weights on the branches. Then you want to find the path with minimum branch weights by sum of minimum pathways, you can use the viterbi algorithms. So, for instance in the equalization problem for I S I channels in digital communication people use viterbi algorithm. So, that also you have a trellis natural trellis representation on which you can do you have to find the minimum weight path. So, and even if you are going from let us say Delhi to Chennai on the road. There are multiple cities that you can go through in the middle I want to find the minimum distance path from Delhi to Chennai.

Travelling salesman is more complicated, let us not go there. It is much more complicated there is no efficient algorithm for that, but this is just point to point, one point to one point. So, those things you can do using a viterbi like algorithm, which is there also other algorithms for it, but this is. So, the reason I am saying this is all this is it is just one instance of a very popular algorithm from other area some other areas. So, it is beneficial to learn other areas. So, only I am saying, so you learn some other area some other type of algorithm you might be able to use it for decoding.

So, we have to we have to talk a little bit about performance. We will use the union bound kind of argument if you remember for the M L decoding soft M L decoding for linear block codes, which is what viterbi algorithm is doing. Convolutional code is after all a linear block code, when you implement it with finite length. When you do M L decoding the behavior can be nicely captured using the union bound estimate, which is one way of doing it. The union bound estimate will have this form assuming. I can write it correctly union bound estimate. You remember what form it has there will be two terms. There will be first a n and then a q what I remember something here. So, D min will enter the picture.

So, minimum distance will enter the picture. So, basically this will be the this q will basically be the pairwise error probability. What this will be kind of distance between two short two shortest distance between two code words. So, let me right that down here what comes here is something that depends on the shortest distance between shortest Euclidean distance.

Remember, we have visualizing that each code word a symbol vector is some point in a large n dimensional space. If you want to find a pairwise error probability what is a if it given that, something was transmitted how will you mistake it for some other specific vector. Then all you have to do is only a one dimensional. So, you look at the distance between these two and q of that distance by two will give you the probability.

So, what enters here is shortest Euclidean distance between any two any two symbol vectors. So, that is what will enter the q what will be a, so that is where the union bound comes in. so, the here we will have a number of closest neighbors. So, if you have to do a union bound estimate for the viterbi algorithm, you have to first find the closest code word or the distance between two closest code words in a convolutional code. That is something you have to find. Then you also have to find how many such how many such code words are there. So, if you have let us say the all zero code word, all zero code word is a good place to start you have the all zero code word. Then what are the code words of least weight and how many of them are there, what are they, what is the weight exactly.

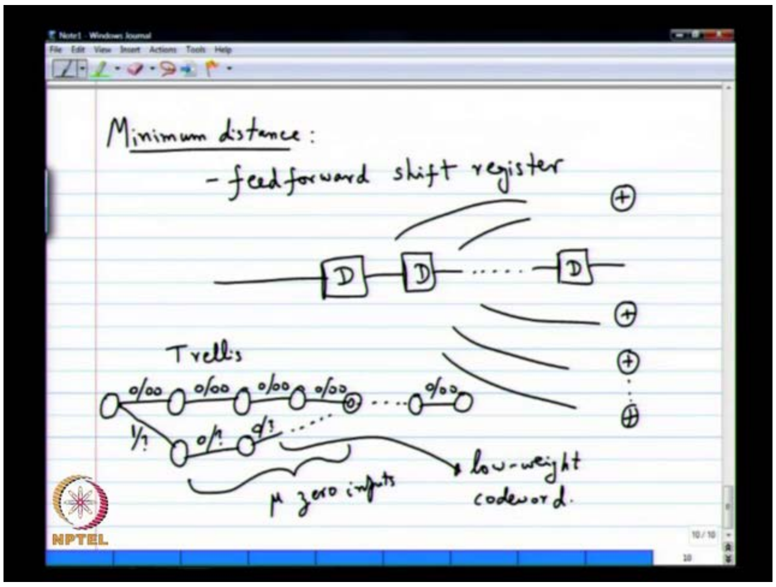
So, that is all you need to find once you find that then you can figure out the union bound estimate, which gives you a good handle on the soft M L decoding, so it is pretty good. So, indirectly what we are doing in this analysis is also trying to solve a design problem. If you can do this very nicely, then what should you be able to do? You should be able to come up with a convolutional encoder, which will give you the best possible union bound estimate. How will you get the best possible union bound estimate? You have to maximize the distance between the two shortest vectors. Then what else you should try to do? You should try and minimize the number of closest neighbors. So, we will see that this number of closest neighbors is not very easy to minimize in a convolutional code.

So, that will be very large and that we will have to well reasonably large. At least it will grow, but the Euclidean distance you can try and maximize. In fact there are no nice closed form methods available for this. So, you have to do a brute force search. So, it is pretty much done using come up with one encoder, after another given a certain memory you can exhaust all possible convolutional encoders. How will you exhaust all possible convolutional encoders just a question of all possible connections.

So, exhaust all possibilities and find this distance for each of them. There is an algorithm for doing that and then find that encoder, which gives you the maximum distance for a particular memory. So, you can do all of that, so this finding the performance of the union bound estimate, also gives you a handle on designing good convolutional encoders.

So, that is something goes along with it, but let us see how to figure out the shortest Euclidean distance. So, I am going to do it very roughly, in a intuitive way based on the trellis. There are regress guarantees for why this will work and sometimes it can go wrong also, but roughly it will work in most cases. So, let us just let us see how that works, so what you have to do is to find a minimum distance.

(Refer Slide Time: 07:16)



So, we have been assuming so far a feed forward shift register. There is no feedback link and that is quite important. Later on we will change this and see what happens to some of the assumptions that we are making. So, we are looking at a feed forward shift registers. So, if you look at the encoder there is there is an input stream there are a bunch of D flip flops mu of them and n. Then what do you do? You have a bunch of outputs and you have connections. There are connections, this is how it looks there is no feedback in the shift register. So, you must have come across feedback shift registers. Also, it is possible to do a feedback. So, you take some part of it and do that and also it is possible, but we will look at feed forward shift registers for now, just keep it simple.

So, in feed forward shift registers trellises are very easy to come up with. Given a particular trellis the next state is obvious. So, you have to just shift and put in the zeroes or the ones in the beginning, there is no problem if you have feedback. Then you have to worry about how that feedback is going to affect your next state. It is bit more complicated. So, let us first look at feed forward shift registers.

So, if you look at a trellis you are going to start at the all zero state. It goes on and on and ends at the all zero state. So, the all zero code word in a feed forward shift register is simply be a well. In fact in any anything I think all zero is all zero will simply correspond to the top line is that. So, in the trellis the all zero code word you show up on the top. That will be a valid branch on any trellis, that will be something.

So, to find the closest code word to the all zero code word, we will first deviate from a particular state. So, let us start with a start at the very beginning i deviate and then all my inputs will be zero. So, I have my first input as one and it has some output corresponding to it. Then all my other inputs will be zero, which means what? I am trying to get back to the all zero state, but in a feed forward thing how many zeroes will I need to get back to the all zero state? I have already put in a one and then I will need? Mu of them, right. I have to put in mu zeroes to flush out this one that got in into the state. Only then I will come back to the all zero state, but it’s all zeroes. So, after mu zero inputs eventually I will come back to the all zero state. So, here there are mu zero inputs.

So, in most convolutional feed forward shift registers, I am not sure if it is true for all this will be the minimum weight code word the code word, that you get on this path. So, it deviate from the all zero path. Then come back to the all zero path by putting in zeroes. After that you will get the you will get the minimum weight thing. So, I do not know if it is possible to come up with a diabolical encoder.

Well, you have to give in more than one input to get a minimum weight encoder. Maybe, it is possible I think is little bit complicated, but if you design it decently, then this will be true. At least one thing you can be sure of is this will give you a very low weight code word. If you just deviate once and then come back after this your all zeroes output is all zeroes. The output here might be something slightly non zero.

So, there will be some non zeroes here zeroes here roughly half of them will be one half of them will be zero. So, clearly the weight here is going to be of the order of mu, then even if you give as many inputs as you want. So, it will give you a low weight code word. In many cases this will be the minimum weight code word also we will simply take this as the minimum weight code word.

So, in an exam question if I ask you find the minimum distance, this is what you have to do in a feed forward case, nothing else. You just go out and then come back to the all zero state figure out what weight you got. That will be your minimum weight code word, is that clear? So, this path corresponds to the a low weight code word. We will take it as a minimum weight code word in this course, at least it is a good approximation anyway.

So, it will be its going to be a very low weight code word. Now, this fixes, the fixes what goes inside the q what about the factor a how many such things can you have? K right of the order of K, because I can do this deviation at any point, it can come back in mu states. So, my number of such low weight code words is going to be of the order of k. At least maybe k minus mu or something, if you want to be very careful B k minus 1 or something like that. So, of the order of k its going to be, so this is the low weight code word there are roughly, there are roughly k of them. Maybe, k minus 1 if you want to be exact this variety. So, that is fairly large I mean k is fairly large if you go to thousand, the thousand such neighbors around all zero all looking to get your decision.

I mean they introduce errors they trying to pull you towards the pull the soft decoder towards the wrong decision. So, all these guys are bad for you and there are so many of them thousand. As you keep increasing k it only grows it does not become a constant or anything. That is fairly bad you can come up with other block codes for which this number is smaller than k much you can do that.

So, in your union bound estimate what is outside the q is pretty much out of your control in a convolutional code, at least with feed forward shift register. Even otherwise you will see it is out of your control, what is inside you can control and that is the weight itself. So, what people do is they fix the memory and try all possible connections to figure out the best weight that you can get and these things are available in tables. So, you will go look up either in Google or look up any book, like book. You will have a list of best convolutional codes for a particular rate for particular memory.

So, that is what you have to pick, do not pick any other convolutional code. If you want a rate half code with memory six you go to that list pick the best possible code, which gives you a good minimum distance and use them. You can go to one of the standards and pick the codes that the standards are using. I am sure they would have done their homework well and they would know that what code will work.

So, you go pick that and then you cannot go wrong. So, that is the that is the crux of analyzing convolutional codes M L behavior and designing convolutional codes. Also, there is no rich theory behind the design of convolutional codes simply do simulations and pick the best. If there is more than one possibility with the same code then you have to do some exhaustive simulations. Maybe, there are some additional code words, which give you some advantage etcetera.

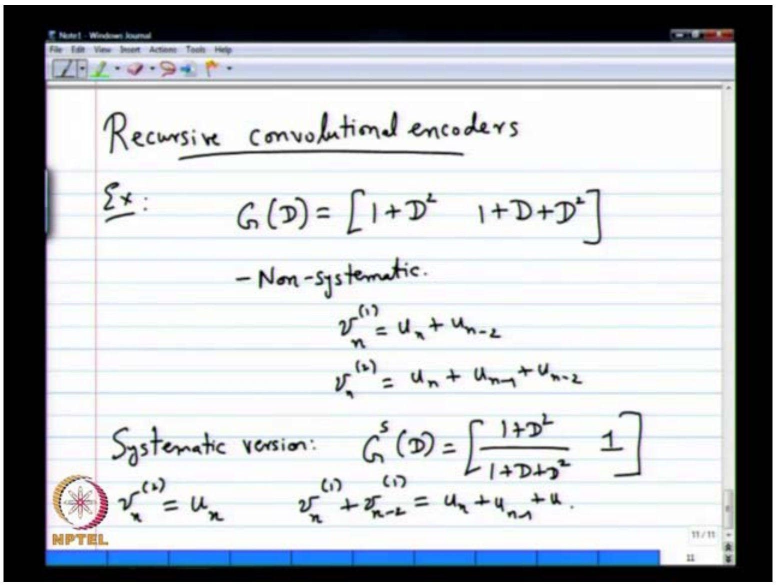
So, all those second order effects can come in also, but usually this design is done based on the minimum distance. So, that is where we will stop also, because this is kind of an old area. It is very well established you pick up a book you can read it very nicely, it is very simple and nice. So, I do not want to go into great details over design of convolutional codes and all that. In fact I do not even know it is more information than this. There are some more things I know, but not too much, not too illuminating. So, we will stop here as far as design of convolutional codes is concerned.

So, what we are going to do now is slowly move in a direction towards turbo codes. So, what is it about convolutional codes, that make it attractive for them to be part of turbo codes, which are therein every standard. So, that is the kind of direction in which we are going to move. For that we need to look at feedback shift registers also, it turns out the using feed forward shift registers alone is not good enough. When you want to try and reduce the number of low weight code words. So, the low weight code word weight itself we have optimized to the best extent possible, but if you want to do some design, where may be you decrease the number of low weight code words. It turns out you have to use feedback and you have to use some more other things not just one convolutional code.

We have to use two convolutional codes and do some fancy tricks between them to get your number of code words down, that plays a key role in the design of turbo codes. So, we are going to move in that direction. So, for that we have to look at what happens when you have feedback shift registers and not just feed forward shift registers. So, that is the kind of that is the direction that we are going to take.

So, we will stop with decoding and performance at this point. So, I am going to talk about that and then we will go on and see, what are the ingredients I needed for turbo codes. Many more ingredients are needed, but we will start with this first ingredient, which is what role does feedback play and how is it going to change. So, we are going to go towards what are called as recursive convolutional encoders.

(Refer Slide Time: 17:13)



Recursive convolutional encoders

Ex: $G(D) = [1+D^2 \quad 1+D+D^2]$

– Non-systematic.

$v_n^{(1)} = u_n + u_{n-2}$

$v_n^{(2)} = u_n + u_{n-1} + u_{n-2}$

Systematic version: $G^s(D) = \left[\frac{1+D^2}{1+D+D^2} \quad 1\right]$

$v_n^{(2)} = u_n \qquad v_n^{(1)} + v_{n-2}^{(1)} = u_n + u_{n-1} + u.$

So, far we have been using feed forward it’s all actually feed forward is not a very common term non recursive is the common term. So, people usually say non recursive, but we now look at recursive convolutional encoders. these play a very important role in the design of turbo codes, but before that they also play an important role, in one minor drawback of feed forward convolutional encoders. So, here the example that we have been looking at the generator matrix has this form 1 plus D square 1 plus D plus D square.This is the example that we have been looking at along.

So, the one drawback here is it is not systematic. Maybe, it is not a big problem for you, but maybe it is something that you do not desire. So, why is it not a big problem not being systematic can always get, but actually in the viterbi decoder the message bit will anyway come. If you do not need to really make it systematic, remember its juts you are finding the path explicitly, the path has both the input and the output.

So, you do not have to really find the code word and in word from the viterbi decoder. So, it anyway gives you the bits, but anyways that is one point to address. Maybe, you want to have systematic for other reasons. The other reasons are usually got to do with the distribution of the input bits and the spectrum, that it will induce in the final modulation signal and all that right.

So, you want your signal to be random looking and all that. So,. you do a lot of that to the message and then when your code if it all that is disappeared all that disappears. Then you are getting a very bad spectrum for your modulated signal. So, you want to control that then systematic is important. So, you want to have a good the same thing appearing after your coding also. So, for that also you need turns out you need a recursive convolutional encoders that is one thing. Of course, the other utility it is a major ingredient in the turbo codes. So, we will see that later for, now we will do this. So, to get a systematic form, what should I have? What will be needed if I need a systematic form in the G of d.

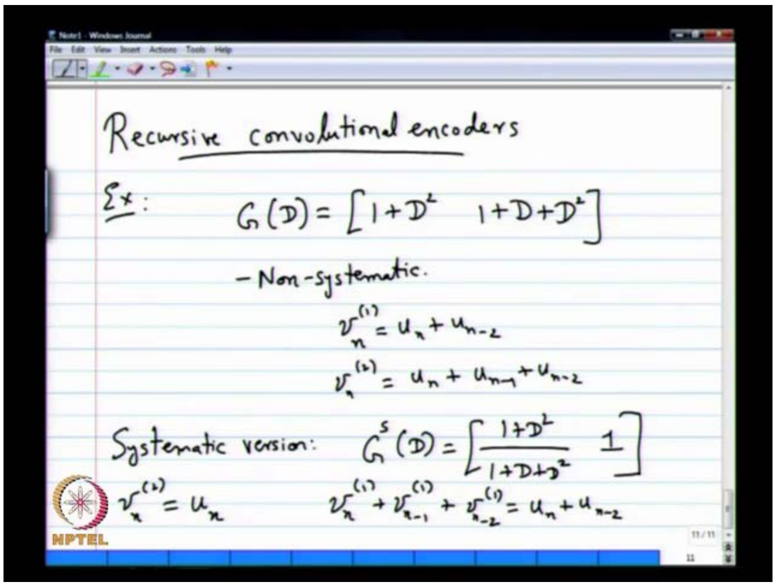
You need a 1 I need a one will give me systematic. Remember, what is this, this gives you two outputs v 1 n equals u n plus u n minus 2 and v n 2 equals u n plus u n minus 1 plus u n minus 2. If I want it to be systematic I want v n 1 to be equal to u n itself that only. Then I will get a systematic form, but remember it is a vague kind of systematic form. How I am transmitting it? I will do systematic parity systematic, parity systematic parity like that. It’s not like block codes where you have all the systematic bits occurring together and then the parity. So, that is the slight difference, but anyway that is good enough I am having the message bits in the explicitly in the code word.

So, I want 1 here, so how can I get a 1? So, I have to divide, so that is the idea. So, you have to divide, that means you have to introduce some feedback in your shift register. Only then you can divide when you want to divide by something you have to do a feedback shift register. So, feedback shift register is a two division and looking at these two what do I think, what do you think? I should divide by dividing by 1 by D square looks a little ugly.

Then you can divide and actually do something some simplification, but if you divide by 1 plus D plus D square, it seems like a minimal form. So, maybe that is a better thing to do, but of course, this is a regress justification for why you might want to divide by 1 plus D plus D square, but anyway, so here is the systematic version. I will call it G S D, just to distinguish from that I will make it 1 plus D square by 1 plus D plus D square and 1. If I do this then I have a easy formula for v n 2, v n 2 is what? u n what will I do for v n 1 how can I write v n 1?

So, there will be a recursive definition. So, I think you have seen this in one form or the other. If you took a digital signal processing course, you would have seen it as an as a representation for the I I R filters. So, you have a numerator and a denominator simply D by z, then you will see everything happening. So, if you do that you have to write down a recursive definition. It is going to be v n 1 plus v n minus 2 1 is going to be equal to u n plus u n minus 1 plus will I get that or the other way round. So, anyway that is wrong, so it is going to be this way v n plus v n minus 1 1 plus v n minus 2 1 equals u n plus u n minus 2.

(Refer Slide Time: 22:39)



Recursive convolutional encoders

Ex: $G(D) = [1+D^2 \quad 1+D+D^2]$

– Non-systematic.

$v_n^{(1)} = u_n + u_{n-2}$

$v_n^{(2)} = u_n + u_{n-1} + u_{n-2}$

Systematic version: $G^s(D) = \left[\frac{1+D^2}{1+D+D^2} \quad 1\right]$

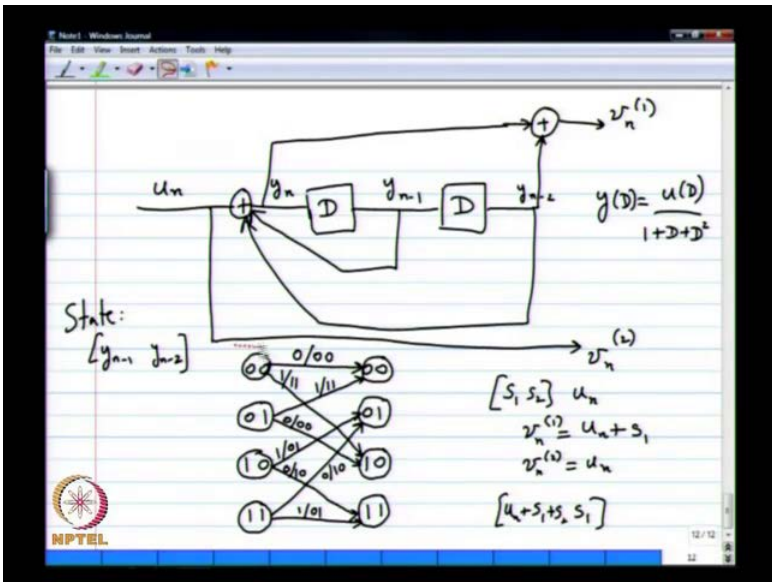
$v_n^{(2)} = u_n$ $\qquad v_n^{(1)} + v_{n-1}^{(1)} + v_{n-2}^{(1)} = u_n + u_{n-2}$

So, I have to do a F I R filters as systematic form is very easy for v n 2, but for v n 1 I have to do a I I R kind of filter implementation. So, it is a recursive implementation, it gives you that is that clear? First of all its not even clear to me, at least when I see it just like that that G of D and G s of D generate the same code. So, you remember if I think of , if I give k bit input and then a terminate to all zero.

Now, terminating at all zero will be complicated for the non systematic for the systematic form. I will come to that why come to that soon enough, but supposing I terminate at all zero will these two codes be the same. If I look at it as n k codes I can do that, will these two codes be the same, that is the first question I have to answer. Turns out they will be the same it’s another thing, but we have to check that.

So, we have to check that and then do all those things a little bit carefully. So, I am dividing by 1 plus D plus D square, it is a little bit different. So, you are checking your saying yes no. So, first thing I want you to do is draw the state diagram or maybe the draw the trellis draw the one stage one full stage of a trellis. So, let us do that for the systematic form for the non systematic form, we have already done it. Let us do it for the systematic form take about few minutes. It is not too complicated, but what is my state going to be once again. So, maybe before you do the trellis let me draw a feedback shift register implementation of this and then you can do a the trellis. So, you have a input u n coming in it turns out I can do it with two D flip flops.

(Refer Slide Time: 24:32)



You might have seen this in D S P classes. Even then I think it’s good to good to see division if you have not if you have not seen division before, it can be a little bit confusing to people. It turns out this if you look at what happens here if you call it y n what should yn be?

So, y n will actually follow this y of D equals u of D divided by 1 plus D plus D square, it is easy to see that. So, this is y n minus 1this is y n minus 2 and clearly y n equals u n plus y n minus 2 plus y n minus 1. So, that is this equation and how will I get a one plus D square on top simply take y n and then plus with y n minus 2. So, that will give you that will give you the v n 1. So, this is a very standard implementation of in D S P for the I I R filter. I think it is called type one or type two one of those types. It is a also there is also other names for it observer canonical controller, canonical all kinds of names for this implementation. It is a standard pictures for linear systems you might have studied at some point in time, but even otherwise is a god refresher is that.

So, this gives you the required transformation. So, clearly it is a memory to four state system that is not changed. So, how will the trellis diagram look like, that is the interesting question it will look… One stage is enough and definition of state definition of state is y n minus 2 and y n minus 1 y n minus 1 y n minus 2. It is not u n minus 1 u n minus 2 remember, what is state in a digital system output of D flip flops not successive inputs. They do not have any role to play here u n minus 1 u n minus 2 is not state. So, that is the first change. So, it is good that at least somebody pointed out state is basically y n minus 1 y n minus 2 see state is something, that will completely give me everything else. That needed to compute the output given the output, that is the way to think about state.

That is the right definition in a general context, I have a system, I want to describe it' state, so I should say everything that is needed to compute the output given the input given. The input u n what do I need to compute? v n 1 and v n 2 that is what you need think about it. So, it is little bit if you have not drawn these diagrams before it can be a little bit confusing to use the previous value or the next value. So, the time it takes for y n minus 1and y n minus 2 to propagate through the all that is very tiny. Do not worry about that you get a y n and then you shift everything. So, it is something that you must have learnt when you did digital systems, but suddenly when you look at it after a long time, it can be confusing. At least it confuses me, I will tell you that much .

The next state is always confusing, so you have to compute y n. So, you fix y n minus 1 y n minus 2 given a u n you compute y n first. Then you can compute v n 1 and v n 2 very easily, then the next state becomes y n minus 1. So, you would not go very easily like in the feed forward case from 0 0 to 0 0 or 1 0, you will go to other states. So, some strange things will happen, you have to look at it depending on the feedback connections, the next state will change.

I am going to try and draw the, so maybe it is not a good idea to draw it here, should let me see this is the easiest part another thing, that is very easy is another thing that is very easy is this. If everything is zero then no can error can happen. So, this is also very easy I will tell you the method I use, which really helps me at least. Maybe, you are very smart, you can look at it and figure out the way I do, it is I try and write down explicit expressions for the y n.

Next state, and the two outputs explicitly in terms of the present state and the input. So, suppose your present state is s 1 s 2, this is your state and the input is u n. What is output v n 1 is a little bit difficult to write down v n 2 is simply u n. What is v n 1? We have to compute y n first y n is s n 1 plus s 2 plus u n. Then your again exhorting with s 2, so you will get what? u n plus s 1 is that you simply write u n plus s 1 what about the next state this is a present state the next state s 1 will go to this side what about this one.

It will be y n right y n is what u n plus s 1 plus s 2. So, that is the way I try to write like this. Then for every state I simply plug in this formula, I will get my answer is that. This is what I like to do there can be any number of methods. I am not claiming, that this is the most efficient or the simplest or anything like that, but this method works in the exam situation. Maybe, there will be some partial credits for this method. So, you might have to be careful with this, it is clear right? So, that is all this is what is being implemented here. So, 0 0 from 0 0 if my u n becomes one then what will happen my output will be 1 1. Then next state will be 1 0 is it next state goes to 1 0 my output becomes 1 1.

So, in general in if you design your if you design your convolutional code reasonably well. Then the two branches that deviate out must be separated by two hamming distance. So, you want to maximize your hamming distance. So, when you get a one you should get at least two that is one thing to So, what about 0 1 0 1 and 0 what will happen?

So, I think maybe I should draw one of these things little bit differently. Another thing to keep in mind again once again in a well designed recursive encoder, the inputs coming into a particular state will be different in the feed forward, case the inputs going into a particular state will always be the same you can come to 0 0. Only with the zero input, but in a recursive encoder usually it will be the opposite. If you got in on one branch with zero the other branch will come in with the one, that is a twist in the recursive. If it is properly designed usually it will be like that.

Of course, there can be tricks where you can violate that. Maybe, I do not know, but usually it will be with a one. So, this one will be one and like I said the minimum distance should give you 1 1 and this 1 will be 0 and 0 0 is that. Now, you know the trellis by symmetry what should happen to the transitions. It is very easy to fix the transitions. Now, right from 1 0 you should definitely go to 0 1 and 1 1. Likewise from 1

1 you should definitely go to 0 1 and 1 1 again in a well designed trellis. All this will be true it will have two outputs from each state and two inputs into each state. I think in any trellis that will be true.

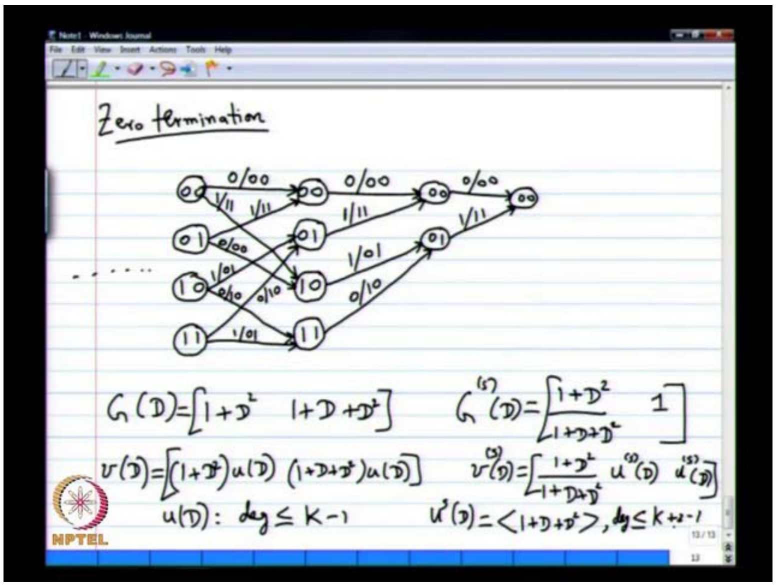
So, you have to have this things and you find the output for one branch, you can find the outputs for every other branch. Pretty much it is just a question of figuring out the inputs. Everything will work out very fast all these things are checks you can use in a like I said in an exam situation any answers. So, with just figuring out one branch, you can quickly write down everything else. Like I said this is true in a well designed trellis, if you did it carefully keeping minimum distance in mind all these things will come out quickly like that. So, all you have to find is just two branches everything else will very quickly work out.

See the thing to remember is I think I compared with the the non recursive version. It does not mean that when you go from 1 1 to 0 1. You should not have input 0 that is not the condition is. The condition is when you are coming in to 0 1, there will be two paths and they will take different inputs one path will have input one the other path will have input 0, that is all that is the only thing to remember it is not that, zero is not allowed.

So, this kind of confusion for instance, now if we did not bother to do it. If you do not bother to work about some example like this in the tutorial sheet, you think it is just simple binary stuff I can do this in the exam, believe me you will go wrong. So, this is some really tricky stuff you have to be very calm and cool. When you do it otherwise you it is very easy to make mistakes is that.

So, now the next question I want to address is termination. So, let us address the termination part. So, starting with all zero you clock in k input bits, suppose you came to the left part here, how will I force myself to all zero state. So, it is a little bit more confusing. So, let us let us do that carefully what did I do let us do this. So, let us address the termination part it is quite easy. It is not that complicated except that it is best to start from the all zero and then work back.

(Refer Slide Time: 39:33)


Zero termination

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

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

$u(D): \deg \leq k-1$ $\qquad u^{s}(D) = \langle 1+D+D^2 \rangle, \deg \leq k+2-1$

So, if you go from here then it might be a little bit confusing. So, you can get to all zero state from two other states what are the two other states 0 0 and 0 1. So, after one input I should definitely have only 0 0 and 0 0, I should not have anything else right is that. So, how will I go from here to there these things are not too bad. I have a input 0 here and then i have an input one here.

1 1 here and then from 1 0, I will have an input 1 here and 0 1 and from 1 1, I will have an input zero here and then 1 0, is that? So, that is that will be the input and then 0 0 is putting 0 0 0 then from here you go with a 1 and 1 1, that is all. So, the point is 0 termination does not necessarily happen with 0 input.

So, you need to figure out what input you need to drive it back to all 0. That will be also either 0 0 1 0 1 1 it will be any one of the four possibilities depending, on which state you are in depending, on which state you are in after k input bits. You will be in some state now you have to do is have a look up table saying from in this state. I have to clock in these things if I am in some other state I have to clock in these you do that we will go back to all zero states. There is no problem there this is a last stage and then you do the termination.

I mean you can some tricks right see for instance from 0 0 itself I can come to 0 1. Then come back to 0 0. I mean if you want to be a little bit more crafty or from 0 1 i can stay in 0 1 and then come back to 0 0, but clearly from 1 0 and 1 1. I cannot play those tricks if I try anything else it would not work. So, from 1 0 and 1 1 it is fixed between 0 0 and 0 1 you can do some minor trick. It would not change anything, but it is still go back to all 0. As long as you pick one thing and stay with that you will be okay, if you keep changing it around, then your decoder will get confused. So, you have to fix something and stick with it. So, this is one thing, which is this is the simplest thing to fix usually fix something like this, it is it will work without any problem.

So, maybe you do not want to I do not know I mean you should not be doing that do not do anything else just use this, this will work alright. So, zero termination is one thing I wanted to mention the other thing. I wanted to mention is the equivalence between the systematic form and the non systematic form. So, let us look at the non systematic form where of G of D was 1 plus D squared 1 plus D plus D squared. Then the systematic form where I suddenly have 1 plus D squared by 1 plus D plus D squared and then 1.

So, remember how do I describe the words that I get. So, code words that I get are if I want to talk about A v of D it is going to be 1 plus D squared times u of D. Then 1 plus D plus D squared times u of D. Then what will I take u of D here to be u of D is degree less than or equal to k minus 1. That is all any polynomial with binary coefficients degree less than or equal to k minus 1, I am going to get a valid v of D. Then you play around with that you will get the answer, what can I do here? Can I say something very similar here what happens to V of D here vs you get 1 plus D squared by 1 plus D plus D squared times us of D. Then us of D remember I also want to 0 terminate. I want to 0 terminate, which means what is 0 termination mean by the way vs of D has will eventually go to 0.

After, a certain point after exactly k plus mu minus one stages vs of D is going to be 0 always. So, which means the polynomial as a polynomial in D vs of D has finite degree. It is not a rational form or anything like that is finite degree polynomial. I cannot have an arbitrary us of D of degree all possible us of D of degree less than or equal to k minus 1. I cannot take I should take us of D to be a multiple of 1 plus D plus D square. Only then it will work and then one multiple of 1 plus D plus D squared, but then what degree? I can always go to k plus 1 I can go all the way to k plus 1.

So, k minus 1 plus 2, so I can add the mu k plus mu minus 1 i can go to, but it should be a multiple of 1 plus D plus D squared is that. So, I have to take us of D I will write like this to say multiple of 1 plus D plus D squared degree less than or equal to k plus 2 minus 1. The reason I am writing it as k plus 2 minus 1 is to bring out the memory explicitly memory is mu is going to be k plus mu minus 1.

So, once I interpreted this way both the code words will be exactly the same. If I do not do zero termination then things will become more complicated. I cannot say talk about equivalence between these two things, because I am doing zero termination. Both will be the same the only thing that has changed is what my us of D, now has becoming a multiple of 1 0 plus D plus D squared. How is that possible? I mean my input data can be anything know how is it always a multiple of 1 plus D plus D squared.

That is what you are doing in the zero termination. So, if it is if it happens to be not a multiple of 1 plus D plus D squared you have to add enough things to make it a multiple of 1 plus D plus D squared. So, doing it by some division with remainder kind of thing that is the idea. So, it is all it all come together if you know something about polynomials, but you know otherwise it will work out.

When you can actually find out the if I know my input screen I can find out, what I should that is what you are doing. When you are doing the encoding the shift register actually does a division, the input gets divided. Finally, you all will be left with the remainder. Then based on that remainder you are going to figure out what needs to be added to make it a multiple. So, that is the idea, so in the non systematic case you do not have to do any such fancy thing to figure out the encoded code word. You do not have to do anything here you have to actually terminate carefully and that terminating carefully.

All that you are doing is making the overall input stream a multiple of 1 plus D plus D squared. That is the another interpretation for the same thing that is all is happening. It sounds interesting enough no it is not too terribly interesting, but anyway some things that are more interesting is. So, what has happened now is the mapping from input to output has changed, mapping from input to output has changed, but the list of code words itself remains the same. You have not changed the list of code words you only map change the mapping from input to output. In fact the input itself changes in a weird way. Initially when you do the input you think of the first k bits as input and the last two are fixed here.

We have to think about little bit more twisted the termination also is important you have to play a role. Some things that are interesting to note here which plays a crucial role in the turbo code idea is if you have a length one input weight one input if u of D here is weight one on the in the non systematic part you will get a very low weight code word is that correct right if u of D is just say one then you simply get one plus D square one plus D plus D square weight is just five again in terms of the trellis what does it mean to have a weight one code word weight one input message you are deviating only once from the all zero state and then you are going back again so that will give you very low weight code word.

What will happen here if you have a weight one message? Maybe, somewhere in the beginning what will happen in the systematic case systematic case, what will happen? It will end. So, you will get a lot of output code weights till you terminate see what will happen. If us of D if your initial message is just one and all are zero. Then you will terminate, so your us of D actually will be a high degree polynomial. It will be a huge degree polynomial that you are going to divide by 1 plus D plus D squared. So, it will give you a lot of ones.

So, it is like an iir filter you know impulse response is going to go on and on forever it is like an infinite impulse response, that is happening till you terminate. When you terminate it dies, but otherwise if you just have an impulse. It is going to give you an technically an infinite sum, but of course in practice you are going to terminate somewhere, but if you want to think along with the termination. Then you have to say that us of D even though there is a one bigger. It is actually a very high degree polynomial, because I will put once once somewhere at the end.

So, that high degree polynomial is getting divided by 1 plus D plus D square. It will get a lot of ones that is another way of thinking about it, but essentially a better intuition is to think of it as a I I R filter I have an I I R filter here. So, if I give it an impulse I will get actually an infinite amount of output till I terminate I will keep on getting something. So, I will get high weight code words. Something is happening here with weights and that is what is interesting in the turbo code context. So, we will stop here for now pick up from here tomorrow at three.