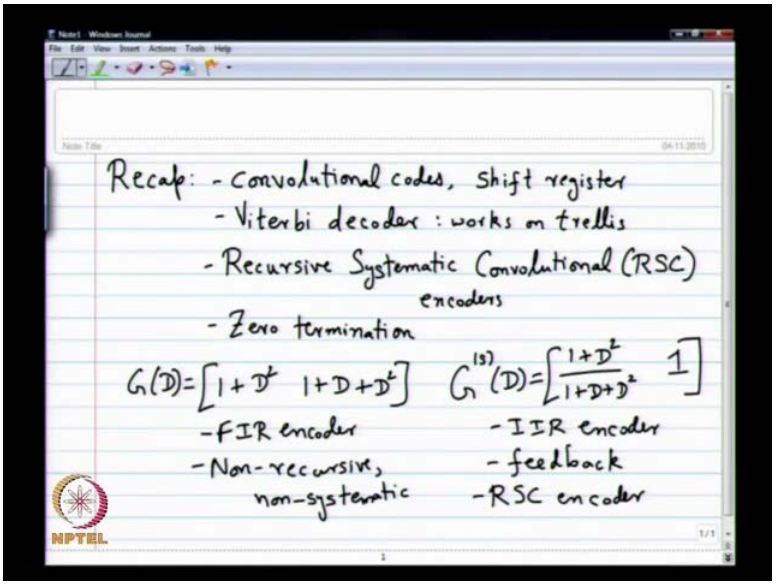
Coding Theory
Prof. Dr. Andrew Thangaraj
Department of Electronics and Communication Engineering
Indian Institute of Technology, Madras

Lecture - 33
Convolutional Codes in Practice

So, once again let us begin with a recap case we were talking about convolutional codes.

(Refer Slide Time: 00:24)



So, we spoke about how to describe this codes and then how to decode them, so the codes themselves are not specified usually you specify the encoder. So, that is that is with the with shift register right and the decoder viterbi decoder works on the trellis. Then what else should we see, the last thing I was talking about was yes recursive systematic encoders convolutional encoders, so this can be abbreviated as R S C encoders.

So, there are some notions here which show up again, one of these notions is 0 termination, so once you do 0 termination you can have you can have several different ways of describing the same code. So, that that is what gives you different encoders you can have it as an F I R filter kind of description or it could be an I I R filter and various other things are possible, anything else is also possible.

So, I gave you 2 examples, the canonical examples are important to understand, they are very typical and any other example also will be very similar to them. So, what are these examples, this example is basically G of D being 1 plus D square and 1 plus D plus D square, so this was this example. Another equivalent form for the same encoder was this, 1 plus D square by 1 plus D plus D square 1, so am I right. So, the choice of these connection like I said is usually there is no big theory behind the choice for these connections.

So, you do some, you do a computation of this minimum distance roughly and then for a given memory you optimize over all possibilities. So, that is why I do not know this is the optimal or not it might be optimal for memory too I am not sure, but it is a very typical nice example. So, the main thing I was trying to convey was that the code when u do zero termination with either G of D or G s of D is exactly the same thing, you do not get two different codes. So, you have the same code except that the encoder is different, here you have, and you can call this if you like F I R encoder or non recursive non systematic encoder.

This will be an I I R encoder, the picture will be slightly different you will have feedback, and there will be feedback in the shift register so this becomes R S C. So, since the code is the same if you just using this code if this if you are just using this convolutional code and you are not doing anything else fancy with it. You can pick either one and you would not lose much in fact the non systematic part is not even, it is not even a problem in convolutional codes because viterbi decoding algorithm anyway also picks up the message directly.

So, you do not even have to do an inversion from the codeword which you might think it is needed in the non systematic case even that is not needed. So, it seems like a very straight forward thing to pick either one, so you might want to go with the systematic encoder for some reason, so it might be might be interesting for you for might be required in your application that it needs to be systematic. So, then you have to pick the G s of D otherwise you can pick the same G of D and you do not really lose anything, so that is the general idea from a code point of view.

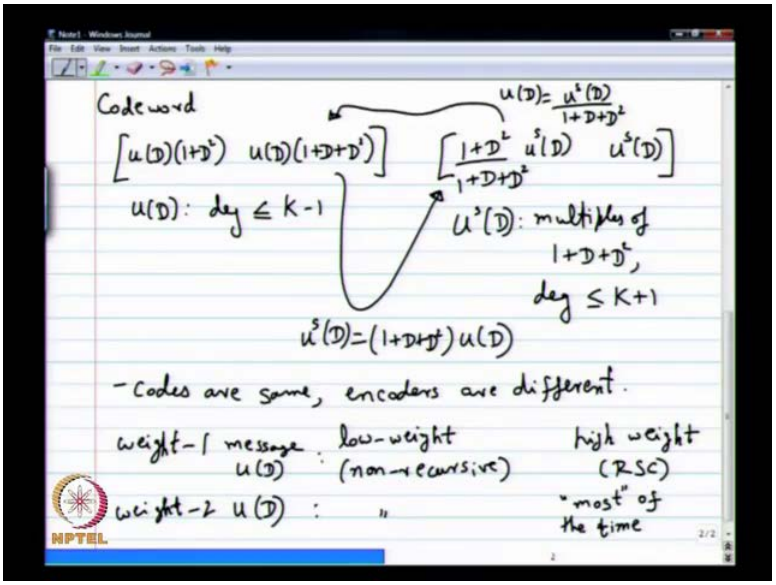
So, there is no difference as it is, but from a structural point of view there are some differences because of the mapping from mapping is different from message to code

word the same message may not give. The same will not give the same code, word in both these situations, so there might be a message a low weight message I will tell you why I am interested in low weight message later on. A low weight message usually gives you a low weight code word with the G of D, so here you get low weight code giving you a low weight message.

Here, in G s of D that is not, so clear for instance the weight one message always gives you a very long large weight code word with G s of D, so that is what I am pointing out towards the end of last class. So, showing how we can think of this as an I I R filter, so if we think of it as an infinite impulse response filter, if you give it just one impulse you are technically getting an infinite length output. So, you will your weight is going to be infinite in the code word of course,, we will terminate it at some point, but theoretically it will be very long.

So, of course you can another thing to think about is if your block length is fixed as n, if your impulse is coming at some n minus 2 or n minus 1 then obviously the weight will be low only. So, those things are always there, so you cannot whether it is this implementation or that implementation, if your code word itself is designed in that it is going to be bad. So, all those things will be there, anyway both the codes are exactly the same, so if you have a low weight code word here the same low weight code word will come there also right, so there is no problem there.

(Refer Slide Time: 06:35)



So, let me just reiterate that code idea, so the code word which we called as v of D did we call it as v of D, no I think what did I did I have a notation for it, I do not know if I have a notation for it the code words. In either case, in one case it is going to be u of D times 1 plus D square and u of D times one plus D plus D square and the only constraint I will impose is u of D should have degree less than or equal to K minus 1. So, that is the only one constraint I enforce here, here it is a little bit different I have to first have see have to 0 terminate.

So, which means I have to stop somewhere, so if that is going to happen I cannot allow all guys here, so I have to just say I will only allow those u s of D which are multiples of 1 plus D plus D square. Then degree is less than or equal to K plus 1, so K plus 2 minus 1 is what I would have written, basically K plus 1. So, it is very easy to come up with the message sequence which will give you the same code word in both cases.

So, if I want this code word to be generated by this I will simply say u s of D to be 1 plus D plus D square times u of D, so nothing else will happen and the fact that for any arbitrary u s of D which is a multiple of 1 plus D plus D square. You can also get back it is also very clear u s of D is a multiple of 1 plus D plus D square, so you know if you start on this side also you can go back to this. How do I go back to that simply said u of D to be u s of D divided by 1 plus D plus D square, why can I do this division because it is already a multiple this is a kind of trivial that there is nothing major in it.

But, you can go either way, so it is it is both of them are going to be identical sets there was a question about this, I think towards end of last class I wanted to answer that. So, basically the codes are the same the encoders are different that is the main point, here and the R S C encoder the recursive systematic convolutional encoder has application in turbo codes. So, it is it is interesting for us to look at that a little bit more carefully and see if it has any interesting properties so that is the that is the idea. So, the key thing like I said what characterizes the performance of L D P, not L D P convolutional codes is clearly on the trellis you have to look for this minimum code words, minimum weight code words on the trellis.

So, what is the minimum possible error that I can make on the trellis how what will be that, how can I deviate from the all 0 sequence and come back to it. In the non systematic, non recursive encoder any weight one message can give you that minimum

code word in the systematic recursive encoder a weight one message will not give you that minimum weight departure. So, you need at least weight two or higher, so I mean it is easy to see what you have to put in to get the minimum weight code word in the systematic trellis. So, this equation is there, you put u of D equals 1 you get the least weight code word, so u of D you multiply by 1 plus D plus D square.

So, if 1 multiplied by 1 plus D plus D square will give you the same thing, so you do 1 plus D plus D square as your message, you are going to get the least weight code word. So, it is really nothing major about this, but I am just saying it is slightly different something a little bit more interesting is weight 2 code words. So, suppose I look at weight 2 messages, so weight one messages message when I say message it is u of D, weight 1 u of D what will happen to that. It gives you low weight code word low weight in systematic in non recursive and gives you high weight in the R S C.

So, I mean these are all I mean roughly true of course there is a weight one sequence which when there is no weight one sequence. But, if you go if you push the weight 1 to the very end, you will once again get a very low weight code word even for the R S C also, I mean as a majority this is true. Many of the weight 1 code, weight one messages will give you low weight code words in the non recursive thing, but only many of the weight one code words will give you very high weight code words in the recursive systematic encoders.

So, that is definitely a true statement, so this is in that idea this is true what about weight 2, what happens to weight 2 code words u of D, so when I say low and high what I mean is with respect to the overall block length. So, here when I say low weight it will be a constant compared to the overall block length it would not depend on the overall block length at all. Here, when I say high weight it will be the roughly the overall block length by 2 it will be a very large number, so it will be increasing as a some constant multiple of that when I have weight 2 u of D.

Once again in the non recursive case I will get something low, only it will not increase with the block length, it will be something which will be two times five or something it cannot be more than that. So, it will be some constant it will not increase, if I increase block length what will happen in the recursive systematic case depends on.

Student: Depends on where we give the weight.

Depends on the relative position of the two ones, of course you are right there is no problem with that which is the which is worst for us which will give us very low weight things what.

Student: Which divides.

So, it should be a multiple of.

Student: 1 plus D plus D square.

1 plus D plus D square, so I am looking for some D power i plus D power j which will be a multiple of 1 plus D plus D square in D power i, D power j, I can pull out D power. Basically, I can look at 1 plus D power i, D power i by itself is not going to divide anything, so that is just you can throw it away, so I am only interested in 1 plus D power i will it be a multiple of 1 plus D plus D square.

Student: 1 plus D cube.

D power 3 or D power 6, so there are a subsets of shifts or gaps or intervals between the two ones which will really give you me a low weight code word in the recursive systematic code, but many of them will not. So, only 3, 6, 9 etcetera will give you 1 if the difference is just 1 or 2 you will not get a low weight code word you will get a high weight code word, so that is the rough idea about weight 2 code words. So, here you always get low weight once again, here you get high weight most of the times I will simply say most, I will put the most within quotes well two-thirds it is most of the time.

But, of course there are some points were you will get hurt, so there are specific places you can be you can really let the devil pick the points for instance, then you will get a very low weight code word. So, three for instance is a particularly bad example here, so in general if that is some other polynomial if it is not 1 plus D plus D square if it is some other polynomial. Usually, it will be a primitive polynomial, so it will be some irreducible primitive polynomial if that is the case then what will happen if it has degree m.

If it is a primitive polynomial what shift should is very bad two power m minus one is very bad, so you know the primitive polynomial of degree m will always divide 1 plus D power 2 power m minus 1. So, it will divide that, so if I put my ones with an interval of 2

power m minus 1 then I will get a low weight code word, otherwise I will get a high weight code word with weight 2 u of D. So, these are the differences between recursive systematic codes systematic encoders and non recursive encoders which are exploited in the turbo code construction.

So, weight 1 messages gives you high weight, weight 2 message gives you high weight, most of the time there are only very few cases which you have to somehow kill and you can kill it in a kind of pseudo random way and get advantages, get advantage out of it. So, you might wonder I mean is this the only way to get low weight code words there might also be some very high weight message which will give you a low weight code word.

So, it turns just like this folk theorem kind of idea in coding theory if you have a code for which you can easily find low weight code words then it is a bad code. When you can find many of them very easily then it is a bad code if you have to do a lot of weight work to find low weight code words then it is not a bad code it will be a good code. So, these are all not statements that you can rigorously prove, but if people have seen it in practice if you have a code for which the low weight codes are easily accessible you can quickly find them then it is bad.

If it is, if you have to do a lot work as in if you have to give in a lot of messages carefully orchestrated to get a low weight code word then it is not bad. So, the reason is nature is not going to be so bad you know it is all those crazy things do not happen often, so this is not a bad thing. But, on the other hand if the low weight code words are plenty like every weight one message is giving you a low weight code word right which is very easy to find. So, then the code is going to be bad as in when I say bad again you have to remember from a capacity approaching point of view it will be bad in general the code will be ok, it is not going to be a bad code. But, for the purposes of approaching capacity it may not be that good that is the idea behind this kind of codes.

So, it seems like in the recursive systematic convolutional code we have a reasonably good candidate at least the weight one messages do not directly give you a low weight code word the weight 2s, yes they do if we can maybe kill that. Then you have to do a lot more work to find the low weight code words and since nature is not going to be not, so bad you are going to you are going to get closer to capacity with that. So, how do you

kill this weight two messages giving you low weight code words is the main idea at the design of the turbo codes.

So, we will we will see that may be soon enough, but before that there are other ingredients into the turbo code we will see that also. But, before that this is the, this is the main idea in the, in the design of the encoder in the turbo code decoder. You need other ideas which we will see soon enough, but the encoder this is the crucial aim any questions, any thoughts.

Student: Anyway I will have as many message bits which giving me low weight in this that many message bits same will be give me the low weight.

Here also.

Student: Overall if I say take the probability of getting a low weight code word.

It should be roughly the same, yes know all that is true that is why I said it is a folk theorem, so you cannot rigorously prove these things that is how it works you know. Here, also actually it is not very bad to find, it is not very hard to find those messages which give you a low weight code words, and all you have to do is multiply by 1 plus D plus D square weight 3 will give you all the bad things. So, weight 3 you cannot do much about, but by the time you kill you reduce basically overall what will happen when you do these things. It is the number of low weight code words will keep going down I mean you have to do I am not saying the recursive convolutional code itself is good.

I am not saying that you have to do some work to get rid of these more things I will, I will show you in the turbo code construction how that works. So, that it is a much more fancy construction than this it is not just recursive convolution that itself will not work clearly it will not work you have to do more work. But, again this is just a general principle you know I mean it should be difficult to find low weight code words in your code. If tomorrow, if you want to come up with a design of a fancy code which you want to claim this is good or you want to hope that it is good.

It should be difficult to find low weight codes in the code if it can be found just like that very easily then it is it is not going to be a good, that is the that is kind of like a thumb rule or rule of thumb or whatever just take it take it with a pinch of salt. So, the same

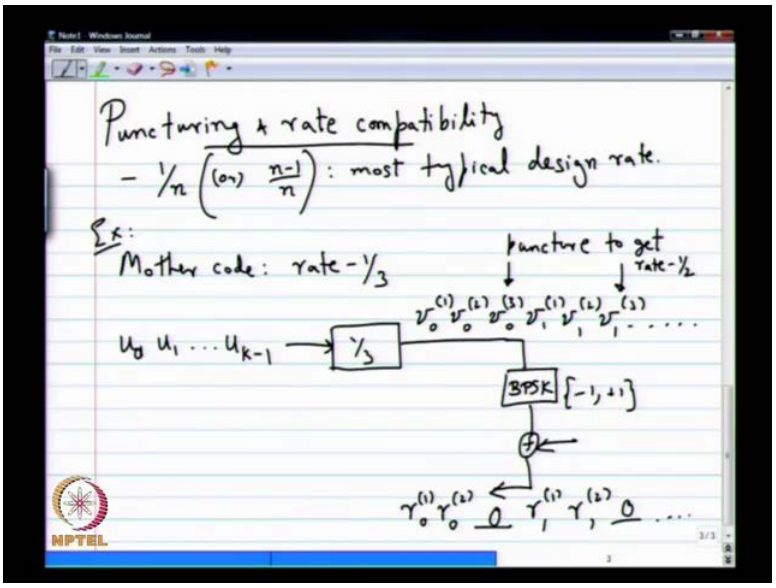
thing is true with L D P C if you think about it, it is low density parity check code and the way you put your ones you are putting them randomly you do not allow too many columns to overlap. Columns do not overlap then how many columns you will chose to add up to 0 it has to overlap, you know it has to overlap and it is not going to be very easy to find low weight code words.

So, again a principle which is valid there and that seems to get you very close to capacity similar principle can be used here. But, I should point out these principles are fairly new as in only like 10, 15 years old, so when the first time people proposed turbo codes and showed capacity achieving performance, many people doubted the simulations that you have made a mistake in the simulation. So, it took a lot of time for people to come out of the rigorous kind of world into this semi rigorous world of random code designs which kind of work that is the idea.

So, this is the part about the encoder I am thinking if there is anything else that I am missing out, here which I should mention let me think for a while maybe I should mention a couple of other things about convolution codes. Before moving on to the, to the bitwise M A P decode of a convolutional code, we saw the soft M L decoder for convolutional codes we did not see the bitwise M A P decoder.

So, we will see that and that is an important ingredient in the turbo decoder this idea that the recursive convolutional encoder is an important ingredient in the turbo encoder, the decoder, the bit wise M A P decoder is important, so we will see that soon enough. But, before that there are few other things that I want to mention about convolutional codes in as it is used being used today. So, we should know these things because these things are used often.

(Refer Slide Time: 22:04)



So, first thing is puncturing and rate compatibility, so usually when you design convolutional encoders you either design it for 1 by n rate or sometimes people design n minus 1 by n, but it is not very common. So, I will put that into brackets, so 1 by n is the most commonly most typical design rate, so nobody does anything different from 1 by n, so the question is of course 1 by n is not the only rate you might want to use. So, n equals, let us say, let us say 3, one-third third is a very popular rate used, so you may not want to use one-third all the time you might want half you might want two-thirds etc.

So, the question is why will you have something like that is a typical example something like a wireless channel, so it turns out because of so many effects the channel conditions do not remain constant in the wireless channel, even other ways other situations it may not be a constant over all time. So, in your armoury of codes you should have different rate codes you should have a rate one by third code then you should maybe have a rate half code when you will use the rate one-third code and the rate half code.

So, you need you need a lower rate when the channel quality is bad, so you want to pick up that when the channel becomes marginally better half you go to the half may be it becomes even better you go to two-thirds. So, two-thirds is kind of where people stop with convolutional codes usually, so you do not want go beyond two-third may be three-fourth may be four-fifth sometimes. But, rarely, but the performance drops quite drastically when you do that, so usually it is one-third, one half two-thirds. So, then how

do you, how do you come up with those codes from rate one-third encoder, so you have to use puncturing so the typical way in which it is done is the following.

So, you have a mother code which is may be a rate one-third code, so here is an example once again there is so many other ways of doing it. So, when you do a rate one-third what is going to happen when you have K bits, u 0, u 1, u K minus 1 this is going to go through your mother codes encoder. You are going to get code words which are let us say v 1 0, v 2 0, v 3 0, v 1 1, v 2 1, v 3 1 and so on, so this will be the code words, so the last thing you will get. So, if you send all the bits that you get then you will have a rate one-third suppose I want a rate half what can I do you can do puncturing clearly, so you simply drop let us say the third parity bit.

So, you puncture here to get to get rate half, so that is that is basically how it is done in practice if we have a rate one-third code and you drop rate 1, 1 per 3 or you might say. So, the question is which bit to drop right which bit to drop is the important question, so you cannot really find answers to those things very easily. So, what people do is to just try all possibilities and pick the one that gives you the best performance with a V E R versus S N R plot. So, that is one thing to do it can be done all this has been done before like I said this codes are in the in the standards.

So, people have really simulated every possibility and you would know exactly, so this have this is called the puncturing pattern the pattern with which you puncture is called the puncturing pattern that is available. So, you might be even a little bit more fancy and say in every 6 bits, I will drop 2 and that gives me, so many more possibilities to try may be not these two you can try, so many others. So, how to puncture is one thing, but then how do you decode the punctured code remember my original trellis is the one-third trellis I am suddenly, I have dropped some bits.

So, how do I decode so you have to assume something in the receiver so what happens is what is typically done is if you let this goes through let us say B P S K and then you have noise adding and you get r. So, suppose you puncture every third one you will get the r 1 0, r 2 0 for r 3 0 which has been punctured you just put a blank and then you do a r 1 1, r 2 1 then you put a blank so on. What will you put in the blank, what is the safe thing to put in the blank or what is the only thing you can put in the blank.

Student: Equidistant from both symbols.

Yes, equidistant from both symbols what is that point which is equidistant from both symbols 0 B P S K is minus 1 plus 1, so this symbol was not this code this bit was not really transmitted. So, you have no idea whether it was plus 1 or minus 1, so simply put 0 as the received value, once you put that you can compute all your branch matrix. Once you compute all your branch matrix, you can run viterbi algorithm and decode that is no problem, so this is how it works this is how the decoder works with puncture. So, if you want to find the optimal puncturing pattern right then what do you do you try all kinds of possibilities, run this decoder and get different D E R versus S N R plot pick the one that gives you the best possible plot, so that is the program that you would have.

Student: Systematic will be easier.

Yes, if this were systematic one of these things will be messaged and the message you would not puncture, so you puncture only the parity, so what about two-thirds, what can we do for two-thirds rate, two-thirds.

Student: Every 2 things you have to puncture

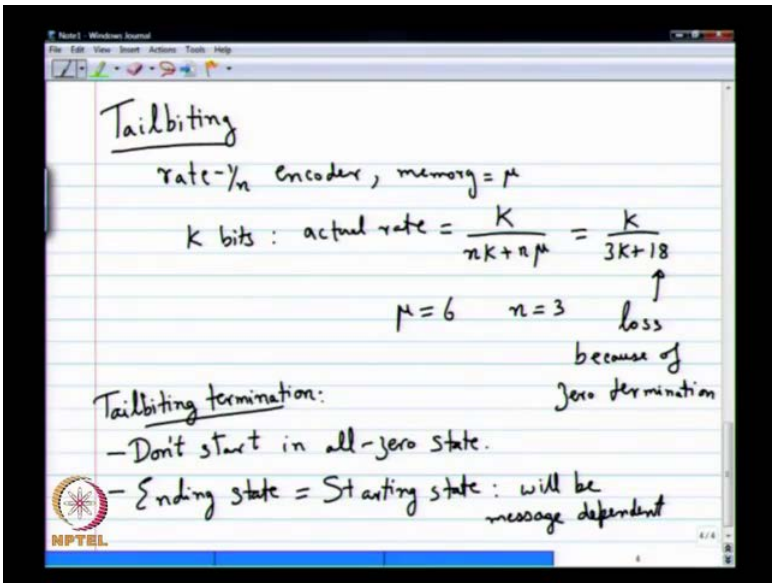
Yes, so you have to pick six at a time look at six at a time and then puncture, 4.

Student: 4, 4 I have to send them 4 out of 6

You send 3 out of for every 2 message bits, I have to send only 3 I am sending 6, now you drop 3 may be 2 is not enough, so usually people do not stop at 2 they go to actually 4 bits. So, that you get 12 coded bits out of this 12 you have to send only 6, you drop some 6 there is a puncturing pattern available for that the standards itself will be there. So, you pick that 6 and you send and the decoder it exactly the same thing whatever is missing you plug in a 0 and you decode. So, this is quite common puncturing is a very commonly done thing in convolutional codes.

So, what is rate compatible, now this idea of using one mother code and generating several higher rate codes from it is supposed to be rate compatible. So, you making the codes all these codes rate compatible in some sense some terminology, it is just used you can get used to it. So, when somebody says rate compatible this is what they mean one code from which all the other codes are derived in some easy way that is the idea, so this is one idea hence is commonly used in most standards.

(Refer Slide Time: 30:00)



Another idea as this notion of tailbiting it is an interesting name, so one problem with this rate 1 by n convolutional codes is, so if you have a rate 1 by n convolutional encoder when you encode K bits the actual rate is what it depends on the memory. So, let us say memory is mu the actual rate becomes K by what n K plus n mu, so it is actually, so for typical numbers if you choose, so usually people pick mu equals 6.

So, let us say we pick mu equals 6 and maybe n equals 3, so what happens to this formula it becomes K by 3, K plus what I am sorry that should be a actual number no 18. So, if your K is something like 10 or 8 or something very low then what happens to this rate it becomes really low much lower than your originally designed one-third and that is not desirable. But, you might ask why K will be 8 or 10 why do you want to use a convolutional code when K is 8 or 10. What do you think can there be any reason why you might want to use a convolutional code for very short messages not just ten maybe twenty thirty something very short?

Student: For the G S M also you will get

Yes, actually if you look at the standards they will specify lot of codes they will not just specify one code for data communication or encoding voice or something they will keep specifying a lot of codes. The reason is before actual communication can take place a lot of other signalling happens something about channel conditions have to be send some all kinds of signalling. Some may be some other control parameters that you want to

exchange various other signalling happens in your in your cell phone for instance, so even for those kind of things there might be some data which is very important.

It will be a something very small, it could something like your some I D that your cell phone sense for figuring out the account. That might need some protection it will be a very small maybe 10 bits, 20 bits and that still needs protection because it is very important data. If you lose it then something seriously goes wrong maybe you are speaking or somebody else gets billed you know something like that can happen. So, you want to avoid that, so you want to encode that also typically people will use a rate one-third convolutional code rate for that. You do not want to saddle your encoder with some mega L D P C code for that you know when you want to use some simple encoder you put a convolutional code it is very standard to do that.

So, there when you use it with a low K, you are going to get hit with this rate loss because of 0 terminations, so this is rate loss because of 0 terminations. So, one of the smart ideas that have come up in the recent past may be I do not know how old it is may be 10, 15 years this is notion of tail biting termination. So, what do you do in a tail biting termination, the only condition you impose in tail biting termination is you do not necessarily start in all 0 state, you do not start at the all 0 state.

So, you start at some state such that such that you do something called tail biting, so the tail should be the same as the head you start at some state such that the ending state. After all, the message bits only equals the starting state that that is the only condition so only condition that is imposed is ending state equals starting state given a set of K bits. You have to figure out that initial state which will also be the ending state after you have encoded those cables. So, for a feed forward encoder it is almost trivial to do that what will what will what do you have to start within a feet forward encoder.

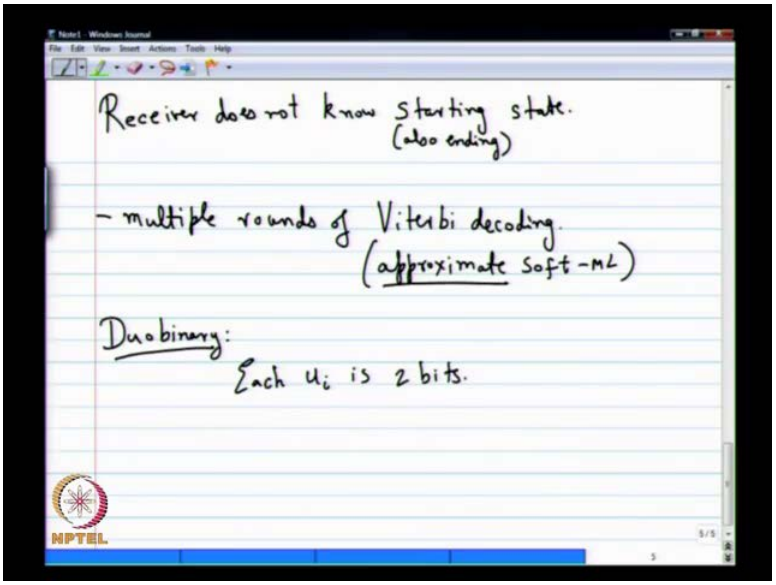
Student: Last mu bits.

Last mu bits, so after you shift in K bits for the feet forward encoder it is very obvious the last mu bits will may be the state, so you take the last mu bits of your, of your, of your message put that as the initial state. But, you have to do a bit reversal it will be like a reversal because of the way you are clocking in it, but that is the only thing once you do that your initial state will be equal to the final state. So, for the feed forward thing tail biting is very easy it turns out even for recursive systematic codes you can do tail biting.

So, here you have to do a little bit more work to figure out that state which will give you which will be the equal to the beginning and ending, so that is the termination.

So, here, so you do not add additional bits after your message bits to get you to the same state as the starting state if you adjust, so that for that particular message bit sequence the starting state equals the ending state. But, what is the catch, here there is a catch receiver does not know that state, so that is the catch, so you cannot avoid that. So, right is that clear because the starting state and ending state the state will be message dependent, so the will be message dependent, so the catch obviously is receiver does not know.

(Refer Slide Time: 36:34)



Know this starting state, starting state which is also the ending state right if somebody magically tells the receiver that this was the unknown state then you have no loss with 0 termination. You have you have gotten way without doing any termination, without any loss in your decoder but of course that cannot be done that is cheating, so you do not know that. So, what can you do what is the possible strategy.

Student: We will set all possibilities in the first state also anyway because of trellis we will get back because of the.

The decoder is a little bit more complicated, so it turns out you have to do multiple rounds of viterbi decoding before you can be sure about anything, so you cannot stop with one round of viterbi decoding. So, I am not going to describe the decoder in detail

here there are some papers that have been written people have implemented this. So, receiver needs to do multiple rounds of viterbi decoding, yes whether it is hard decision or soft decision you have to do multiple rounds. So, you cannot start at 0, so you do not know where you have started you do not know where you are ending, so it is, it is complicated.

So, all the, so even at the 0-th stage all the branches will have matrix, so at the zero-th stage in the proper 0 termination only 2 branches will have matrix, so you go very easily you start very easily. But, the 0-th stage itself all the branches will have matrix, so everything will have a matrix every state will be active and at the end you would not know which state to follow, when you end. Also it is not that you are ending at one state in the previous case, we were ending at 0, so you are able to pick that path now you would not end at 0. You will end at everything every state will have one survivor path which one will you pick yes, so one choice for instance is to take the one with the minimum one, so that may not be tail biting then what do you do.

So, it is a complicated problem right some research is needed it has been done people know how to do it, but you cannot do optimal m l or anything you can only do some approximate decoding and also it will be. See this will not work for K, for K you will really struggle why will it not work for K anything you do will not work for K, see you can have a see if you look at, if you allow any tail biting path. Then the closest neighbours can differ only in the 0-th stage and the K minus 1 stage, see if you have, if you have look at the all 0 code word which is a valid tail biting path.

Then you differ at the 0-th stage differ at the K minus 1 stage everywhere else you keep it as 0 if I do that, if I can still get a tail biting path, if your trellis is like that. Then that is the closest neighbour and that will have a very short distance from it and if your K particularly becomes very large this distance many are not enough. So, tail biting may not be a good idea when K becomes large and any way you do not need it when K becomes large, so only when K is small you can get away with this tail biting. So, you can do multiple rounds of viterbi decoding and you can do approximate soft m l soft m l approximate is the key here you cannot do accurate soft m l that easily.

So, you do approximate soft m l and this turns out to be good enough for low for k, it is I am not I know, I am not describing the decoder in detail. For you it is a little bit more

complicated, but the idea involved something like that what was suggested, so you look at all the survivors and then look at the ones with minimal path. See, if it is tail biting if it is tail biting then you can declare that as the best, but then if it is not tail biting you carry your matrix to the initial state. Once again the same matrix you retain to the initial state because you know it was tail biting, so you keep the same matrix you kind of fold your trellis and then keep on doing circular go round and round in this it is called the circular viterbi algorithm.

So, since you know that initial and final state was the same, so you can close your trellis like that and when you start at one point go around and then continue again. But, then you need to once again see what happens, so what happens finally will you get a tail biting path which is good there are conditions for that you can implement that. So, you get a decode, so little bit complicated than viterbi, but it can be done and it is it is nice in the sense that you get a you get rid of this 0 termination loss, so for k it is very useful for sending some small information tail biting is quite useful.

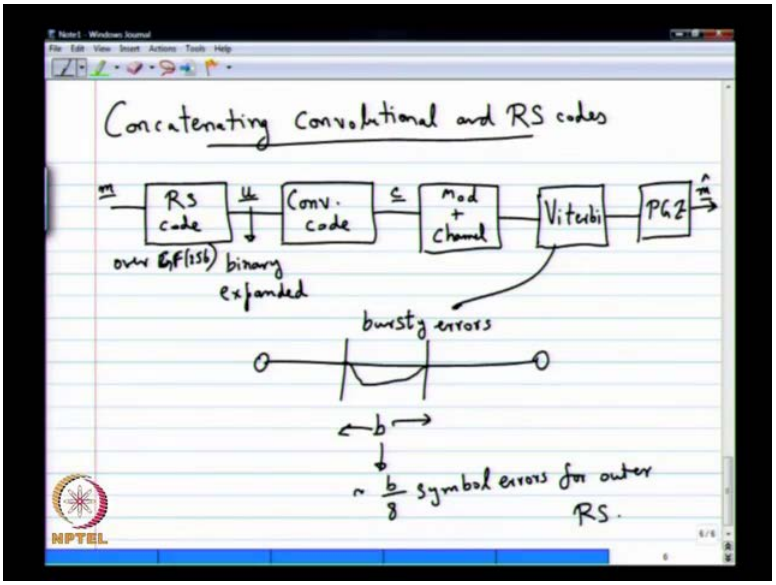
So, I really do not have any more details about I do not want to give any more details about tail biting, but it is there in the standards most standards have tail biting convolutional codes they do not have non tail biting things. Another thing you will find in the standards is something called duo binary, I am going to only briefly mention what it is I am not I have not seen this very closely myself. So, I do not know why when there are some reasons for why they use it so here instead of each message bit being 1 bit, it will be 2 bits that is all, that is the idea.

So, each message input each u i is 2 bits that is the idea, so it is like your D flip flop which is actually holding 2 bit output, 2 bits are coming in, 2 bits are going out. There are some constraints I mean you cannot you cannot suddenly make the memory very large or something. So, the states will still be small so there are ways in which you can design it and for some reason this is considered good, so you will see almost all standards used to binary codes particularly in the, in the turbo code or whatever.

So, they used duo binary convolutional codes a lot and maybe it is a part of reading assignment somebody can look at why a duo binary code is used. So, you can try and find out come and educate all of us on why do a binary codes are considered better than a binary code there is some reason to it I have read some reasons. But, I have not seen

myself that closely so this is another thing that is always used with convolutional codes and, so the one final thing that I want to mention within the in this idea of convolutional codes is.

(Refer Slide Time: 43:18)



Concatenating convolutional and Reed Solomon codes, so these were very popular codes for quite a while, so for a long time about maybe in the 80s or 70s or maybe they have invented at that time for space application. Even in other places, this was very popular, so you concatenate the convolutional code and a Reed Solomon code I will tell you why this was very popular and why it worked very well. So, before that let us see a block diagram of how this would look the outer code, so when you concatenate you put one code with the other, so you will have an outer code and an inner code. So, the outer code usually the Reed Solomon code and the inner code is the convolutional code and then this goes through the channel modulation etc.

Yes, so the code words of the Reed Solomon code will be encoded using the convolutional encoder, so that is how it works is that, so we will have one message you will have an original message. Here, this will become a code word this will become a u, so t speak then that becomes the code, so this is s very popular kind of thing and then you will have first what should you decode. First you will have to decode the convolutional code first because this is the inner code, so you run a viterbi and out of that then you run maybe a P G Z, P G Z is not really run. What is usually run is you call the

algorithm, so that is what you would run for the Reed Solomon code, so this will give you an m hat finally

Student: Sir, this knows the binary expanded equivalent.

Yes, it has to be binary expanded equally everything should be binary expanded, so the reason why this is such a successful combination it is like what to say something maybe what can you say it is like rice and sambhar. For instance I mean a very vegetarian example I actually thought of a non vegetarian example I did not want to say it so, so where are we why this is an interesting combination. So, remember a convolutional decoder is decode is runs on the trellis you have a valid path, so this viterbi algorithm is trying to find a path in the trellis.

So, you have a legitimate path which is the actual path that was followed in the transmitter and then there will be a decoded path and we know that this is a soft m l decoder. So, if at all it makes errors it will make errors at the closest neighbour it is not going to go so far away it is usually. So, suppose let us assume that we are operating at a s n r where if at all its makes an error.

It is going to only go to the some closest neighbour and how will the closest neighbour look for a convolutional code, usually it is going to look like this, so it is going to be a sequence and then it is going to go back. So, any error you make is going to be a burst a burst in your received word, so there will be a starting position and ending position and all the errors will be confined within that burst why is this good for an outer Reed Solomon code.

Student: Because they treat as symbols.

Exactly, so if you have a burst of length say b and if you have a r s code over let us say 256, G of 256 essentially this will translate into a translate into a roughly b by 8 symbol errors and your Reed Solomon code will handsomely correct the code. So, if it were the errors were to be random inside then your Reed Solomon code may not really help you too much, but it might help still. But, it would not be that effective, so the reason why this combination is really good is the errors that the viterbi gives out are suited very nicely for the Reed Solomon code.

So, you all the bursts get reduced to a very few symbol errors then you can correct it very nicely, so this is a this was a very popular combination for a long time till the L T P C turbo codes of course took over and even today when you use L D P C codes. Sometimes people use it as a inner code and put some B C H code or Reed Solomon code outside and that is for a different reason not because of the errors are matching or anything. That is because when a L D P C code makes errors, it will make a very few bit errors, usually if you design it well it will make very few bit errors may be 3 errors 4 errors something like that.

So, you will have an outer B C H code to clean up those few errors whenever it is possible that is the reason why they do that. Here, it is it is more than 1, of course you want to clean up residual errors also, but more than that the errors that the viterbi is going to make is mostly align towards burst. So, it is going to be bursty and those bursts can be very nicely handle by an outer Reed Solomon code, so this is this is a popular combination it was a popular combination. Now, of course the other things are more interesting, so anything else that I should mention about convolutional codes and reason standards anything else that you came across, yes I am sorry.

Student: Concatenation.

Yes, the overall rate will be the code rate will be the product of the 2 code, so because it is bursty you can do away with a very small minimum distance with the outer code it means the rate hit is not good are. So, usually the outer code will be like a rate 0.95 or very high rate code 0.9 may be very high rate code convolutional code might be rated half or something. So, you do not really take that bigger hit when you do that, but yes, but rate is the problem rate will go down when you concatenate. So, in general though concatenation is shown not to be a bad idea like this not like this, but something slightly different concatenation is usually a good idea there are, there are powerful other ways of concatenating.

So, one of the greatest advantages of concatenation is what as you can see here at the decoder you do not have to decode the total code, you decode the inner code first and then you decode the outer code. So, concatenation is a very powerful idea even in turbo codes you will see the concatenation is a powerful idea because the decoder you can have you can easily come up with sub optimal decoders.

You decode the inner code first and then you decode the outer code beastly it is not optimal right if you have the overall code and you can decode the overall code then that will be the best thing you can do. But, then that complexity is going to go become very high, so you decode something inner first. Then outer manageable things it becomes very that is another main reason why concatenation is more popular, so we will stop here for this lecture.