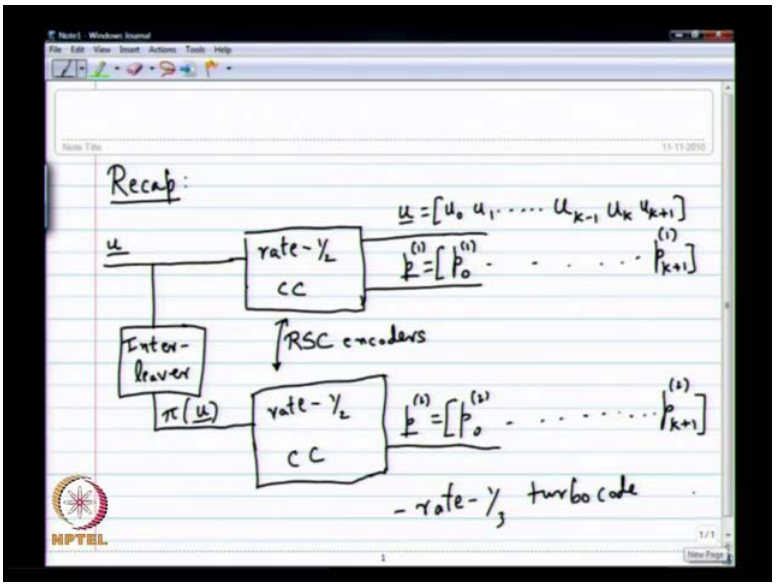
Coding Theory
Prof. Dr. Andrew Thangaraj
Department of Electronics & Communication Engineering
Indian Institute of Technology, Madras

Lecture - 36
Turbo Decoder

(Refer Slide Time: 00:18)



So, let us begin with a quick recap, we are at some kind of a momentous occasion in the codes. We introduced turbo codes, and this is the very famous picture, so the picture was a very typical situation with turbo codes you have a rate half convolutional code and the message goes in here and there is a what is this block inter leaver. Then once again there is a rate half convolutional code here you keep both the systematic version and the parity here you keep on the parity, so that is the way we are viewing it.

So, maybe you want to terminate it and might have, so I might decide to terminate the first one which case you would get the first parity sequence. If you are terminating the second convolutional code second convolutional encoder also, then you will need some additional bits that will have to add to this. So, if you do not added then you are not terminating this, so here you would have again the second parity which is which is let us say p 2 0 all the way to p 2 k plus 1.

If you want, so these are the terminations is not a big deal, I mean it is not some something that you have to take care of, but it does not change the performance as

significantly as long as you term terminate at least one. So, it is very typical to terminate the first one and not terminate the second one and then you are so nominally this is a rate one third turbo code.

Of course, if your account for the terminating bits there will be slight reduction in the rate, but it is one-third is fine and usually k is k will have to be large rest to be at least 50, 100 or something before meaningful results can be obtained. So, 50, 100 is typical, so in that case the termination is not a big deal, so once again to get higher rate codes like rate half and all that you would puncture and there are puncturing patterns that are possible depending on what rate you want. For rate half, there is a standard puncturing pattern where you keep all the odd numbered bits from one parity and keep all the even numbered bits in the other parity.

So, that is something that is bit more standard and the rate half any questions on this general structure. So, the crucial point is both these guys can be same code but they will have to recursive systematic convolutional encoders. So, once you make them recursive systematic convolutional encoder convolutional encoders and you put a random inter leaver here and when k is large, it is the number low weight code words code words go down significantly. So, at least that you can easily find go down very sharply so because of the because of the inter leaver changing something. You could have it different, but what people are found in practice after so many years of research is that you can keep it as the same code and you do not lose much their performance.

This will not work, the question was can you remove the inter leaver and simply do it is concatenation two different also it does not work that easily without the inter leaver. So, it is it is easy to find low weight code words if you do not have the inter leaver, so the inter leaver is something which is which adds a non trivial thing to it.

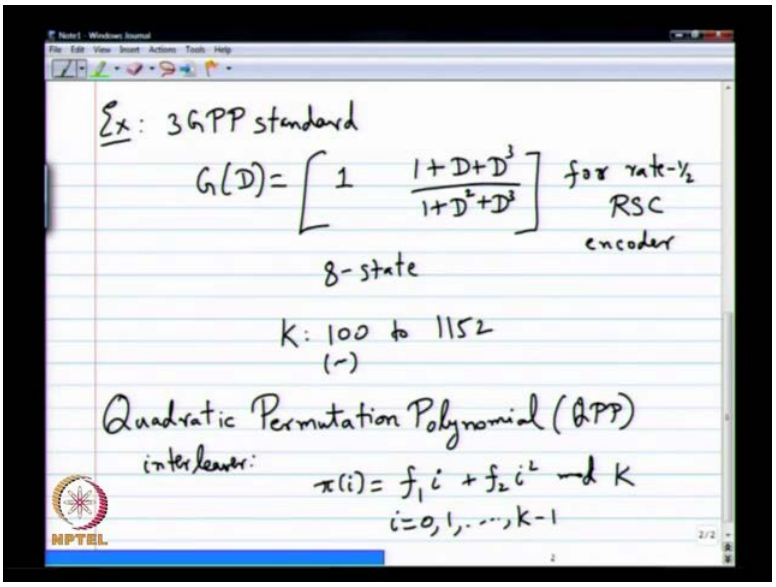
So, the question was if you put two different RSC encoders your denominator polynomial is different in both then what happens you will have a different thing. Then, you can always find the LCM of those two and that leads you to from a very simple formula for finding very periodic repetitions will happen. So, you do not get much advantage, but on the other hand if you do a random inter leaver, then it really kills the kills any symmetry in such kind of a situation. So, that is the structure like I said the encoder is motivated in a more or less ad hoc way.

I mean, I think of how you how you are getting rid of some low weight code words and you do not really do anything more beyond that it is tough to motivated in a very regress way. The most important thing for any error control code is the BR versus SNR curve, so if it goes very close to capacity and if it is giving you good performance you are happy. So, that is the main selling point for turbo codes they have very good decoder which works well.

So, that is what we have to discuss next everybody happy with the encoder as such it is clear so there are there are as many variations on this encoders as there are probably researchers this field you know people have different variations. One very popular variation is a serial concatenation what do the serial concatenation look like would have an outer convolutional code an inter leaver and then an inner convolutional code. That is also supposed to be pretty good and there are properties for that also that people study and what you do once you can do multiple times.

So, it is you have to put one more inter leaver and put one more convolutional code you get more random behavior. So, you can keep doing this and change this structure as long as you keep the main mantra in mind what is that mantra, there should be some random looking inter leaver to make sure that you cannot find low weight code words very easily. So, as long as you do that you will keep getting additional structures, so of course the decoder is important you make sure that the decoder also extends in a similar manner. So, when I discuss the decoder that will be little bit more clearly, so let us maybe see one more example maybe from the standards.

(Refer Slide Time: 07:43)



This time from the 3 GPP standard, so this is something I remember from memory, hopefully I am right, I would not make a mistake here so that g of d they use for the convolutional code is something like 1. Then, I think 1 plus d plus d power 3 by 1 plus d squared plus d power 3, so this is the G of D for the for rate half recursive systematic convolutional code. So, how many states how many states does this encoder have eight state, so it is an eight state encoder. I believe in 3 GPP, you terminate both the encoders, so there is some additional bits that you sent to terminate both encoders after the interleaving and the inter leaver.

So, usually in standards they have to specify different block lengths your block lengths is not fixed, so your k is not fixed k is usually can start at 50 or something, I think 50 or maybe 60. Then, keep keeps on going to hundred two hundred three hundred etcetera so on all the way, I think the largest block length they have for this in the 3 GPP standard is some 1152 something or something like that.

So, k will go from some let me say 100, this is only a rough number 2, I believe 1152, so the mother code is the rate one third code right the rate one-third code is the mother code with this is the component code and the only thing missing is the inter leaver. The inter leaver in the 3 GPP standard or maybe even in the other standards, they use this inter leaver it is what is known as a quadratic permutation polynomial inter leaver QPP inter leaver it is popular for a various reasons. I will simply provide a description, but basically they will define pi of i to be some f 1 me plus some f 2 I squared modulo k.

For each k, you define a suitable f 1 and f 2 and then to get to get the permuted position corresponding position i i is the original unpermitted position 0 1 to k minus 1 to find it is the position of i after permutation. What you have to do you have to evaluate pi of i i goes to pi of i and pi of i is f 1 i plus 1 i square mod k. So, this is called a quadratic permutation polynomial or obvious reasons it is quadratic and you have to you need some conditions on f 1 f 2 and k to make sure that it is a proper permutation that those things are quite standard you can do that.

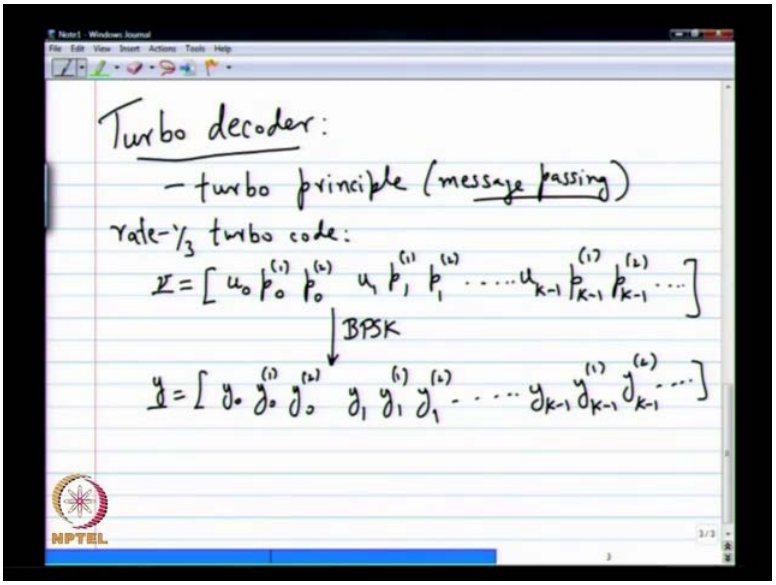
So, it is appealing for various reasons what are the various reasons you can think for which this inter leaver is appealing. So, remember the main selling point for the convolutional codes is that the encoder is simple, so it is really that simple very cheap. Now, if you look at this picture which is the most complicated part to implement in this picture the inter leaver rights everything else is trivial it is just three d flip flops. You do not even care about it, so the inter leaver you cannot expect a memory of 1152 or something a huge memory. Then, you then randomly access it pull it only it is going to increase your power spent, so this one is particularly simple.

So, you can compute the position you do not have to necessarily store it somewhere, so you only have to store the f 1s and f 2s you do not have to store as many as the entire array you do not have to store. So, you store f 1 and f 2 and compute it on the fly and figure out what position it goes to, so that is something that can be done without too much efforts. So, that is one reason why it is popular there is another more significant reason why it is popular I will address that later when we talk about the decoding. So, right now this is at least complexity is less and also the other condition is it should be ransom enough it is should not be very regular.

So, this quadratic function makes sure that the permutation looks random enough it will not be a very simple permutation if I give you the permutation you will never be able to guess that it comes from a quadratic function. So, it is just its looks random enough and it satisfies that constraint also, it is a pretty popular thing that is used is there a question hashing. So, I do not know hashing, I do not know I am not familiar with any connection maybe there is, but this is not really hashing. This is one to one mapping, no means a proper permutation hashing; usually you take something very large and then put it into a small set.

So, this is this is a one to one map so interleaving is not like hashing that way, but the method might have some similarity in hashing. They do a linear function usually quadratic also maybe is popular, but here you make sure the coefficients are chosen. So, that no such hashing effect happens as you do not reduce the size significantly keep the size the same so this should be an invertible map. So, this is just for information and the let us move on to the famous turbo decoder.

(Refer Slide Time: 13:34)



So, turbo decoder today is both a particular decoder for the turbo code and also a concept, so in general there is a turbo principle which is used in this concept of a turbo decoder. It can be used for anything other than codes also anytime you have a concatenation of two trellises or not just not necessarily trellis two structured entities. So, you can decode one and then use the turbo principle go to the other and then come back using the turbo principle etcetera. So, it is basically like a message passing type of decoder, but you can go back forth and back and forth and that is the turbo principle idea. So, apparently in cars, I do not know how many of you know about mechanical engineering, I do not know it is may a long time.

So, in cars there this turbo drive, so it is called turbo for that reason there is something like energy goes from one out and then comes back and something I do not know does anyone know more about this. Nobody knows that suppose to be the inspiration for the name that is where the name comes from the turbo.

Basically, I have something like there is a cycle or something some they use it in some way to make it better, but today this word turbo basically means in anything you give it a turbo boost so to speak. So, it is even in processor you have this turbo concept, so anything which seems to do better than it should is called turbo. So, that is the idea in this word today, I mean some more use, but in the main principle counts because you do in cycles over and over again and you gain some improvement small improvements. Eventually, they add up to something very big that is the idea the turbo principle, so in general what do you we have in it? So, let us look at the encoder once again and then see what we can say about that received word.

So, you have a code word, so let us say we are using the rate one third turbo code the typical code I will describe the decoder for this code and once again the extensions to the other codes are easy. So, if you have this then the transmitted code word which I think we call v is going to something like u 0 p 1 0, p 2 0 u 1 p 1 1 p 2 1. So, until I will stop with k, so I will not k minus 1 I will not worry about the termination, but when there is there is some termination going on depending on whether you are in the first code or the other code. There is some termination going on, so I will keep it at k minus 1 just for just to keep the notation simple and not worry about too many other things.

So, there is some other termination stuff and that is your code word and this goes through BPSK and you will get what I have been calling this. I think maybe y, so y this is going to be y 0. Now, we need slightly better notation, so what I am going to do, so I will put A 1 here to indicate that it corresponds to the first parity and then two here to indicate that it corresponds to the second parity. So, keep everything else the same just make the replace it as y. so, y 1 1 y 2 1 so on all the way to y k minus 1 y 1 k minus 1 y 2 k minus 1 you also have the termination stuff depending on how you have terminated.

Now, this is my received word and I have to do decoding, I know that convolutional codes have very good decoders they have very good soft input soft output decoders like the BCJR algorithm and all that. So, I am going to use that, but then the entire code word is not a code word of the convolutional code not a single convolutional code. So, it is a code it is a concatenated version from two different convolutional codes however from the code word, I can extract a part of it which will definitely be a code word of the of one of the convolutional codes.

In fact there are two different convolutional code words that are subsets, so to speak of this code word what are two convolutional code words u along with p 1. That forms a code word of the convolutional code on the top u along with p 2 forms a code word of the convolutional code at the bottom of it. So, I can run two different convolutional decoders maybe Viterbi I might to run or maybe I want to run BCJR. So, I can run two different convolutional decoders on this received word and get estimates of estimates of each of these bits. So, estimate of u at least I am interested only in estimates of u, remember the convolutional decoder for BCJR gives you only the l l rs for u, you can also find l l rs for the parity, but usually nobody does that, so you only find l l rs for u.

So, I will get two different l l rs for u, so anytime you have more than 1 l l r for your message bits what do you do you try to pass that message from one to the other and try and repeat the decoding to make sure that you can get more benefit out of it. Only you have to make sure when you do the passing is you should pass extrinsic information you should pass anything that is intrinsic. So, that is a little bit trickier to figure out in the convolutional code what is extrinsic and what is intrinsic is a little bit trickier. Once you figure that out you can do some message passing iterative decoding, so you have two different components decoders one for one part of the code word another per another for another part of the code word you decode it independently.

Then, pass extrinsic information from one to the other and try and improve the other decoding so in principle, it is still message passing so this turbo principle is actually a message passing principle and there are parallels between this. What we did with l d p c decoders also except that in the l d p c decoder what are the components decoders you think of the tanner graph the component decoders are the check nodes what the code are. The decoding the check node is doing something the check node is trying to decode which codes, think about it is decoding a sub code. So, see check node is connected to a subset of the bits and those bits satisfy a single parity constraint that is all and the check is doing decoding for that single parity code.

So, the difference between turbo codes and l d p c codes is there you have several component code all of them are operating at the same time. Then, you are doing message passing here you have only two component codes with long code words you decode them and pass messages. So, in principle they are these two are roughly similar I will comment on this a little also, but that is that is essentially the idea.

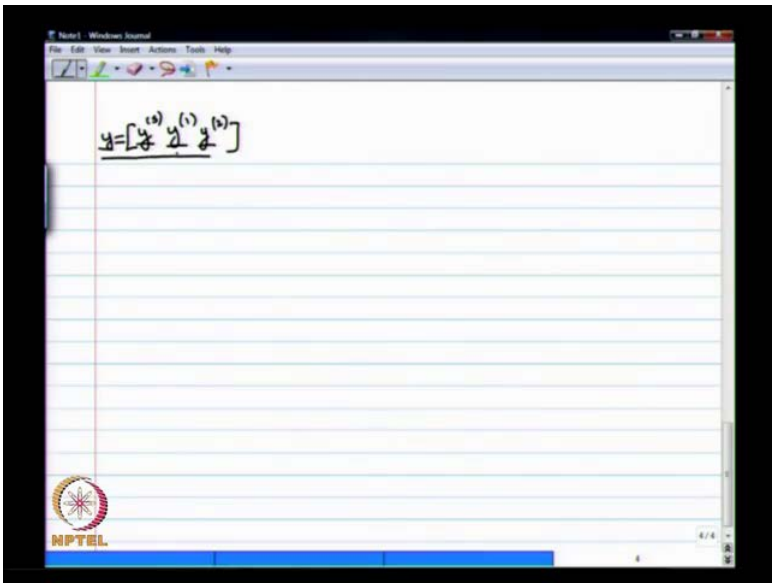
Anytime, you have a code a long code which you cannot directly decode very optimally what do you look for you look for sub codes smaller codes which have enough structure for which you have easy to implement optimal soft input soft output decoders. Once you do that, you decode the sub codes and then what do you do you do message passing to improve the performance all around that is the idea convolutional code by itself is not enough to get to capacity. You have to together and then they are operating in this message passing turbo fashion, then they can help each other out and get to capacity.

So, it is like two students who are not maybe the best students in class, but they keep solving problems and keep exchanging ideas and eventually they become much better than even the one single best student in class. So, that is the idea this something like that happen here or not I do not know maybe two students get together you become the first in class.

So, maybe that is something that you can also anticipate, but as long as you make sure you are passing extrinsic information what one does not know is what is being sent to the other. That is, I did not put it right what you get is what is something that you do not know some fresh information as long as is fresh there's no problem if you keep circulating the same old stuff. Then, you kind of go down only, so that is the key idea, so let me draw the structure.

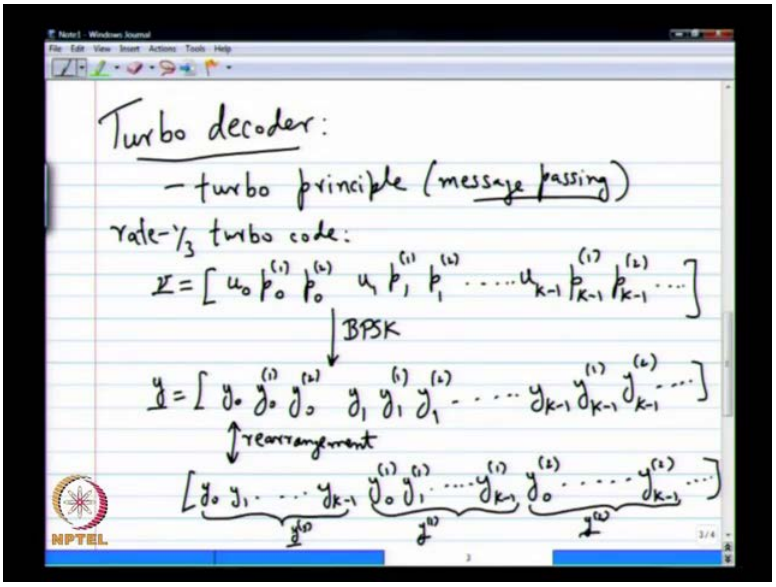
I will draw how the decoder looks and then you will see how the message passing idea is going to work, but then I will also have to address what the component decoder is, so I will do that also, but before that let me draw the overall structure of the decoder. You will see how the message passing idea is kind of inbuilt into this structure, so it is quite easy if you understand this principle.

(Refer Slide Time: 22:57)



So, I am going to take the received vector y, I am going to think of it as two different parts, so I have to write something here. So, I will write it as y I will put s here to denote this systematic vector and then I have the y 1 vector which is the received values corresponding to the first parity and then I have y 2, so this is a bit different from what I had before.

(Refer Slide Time: 23:26)

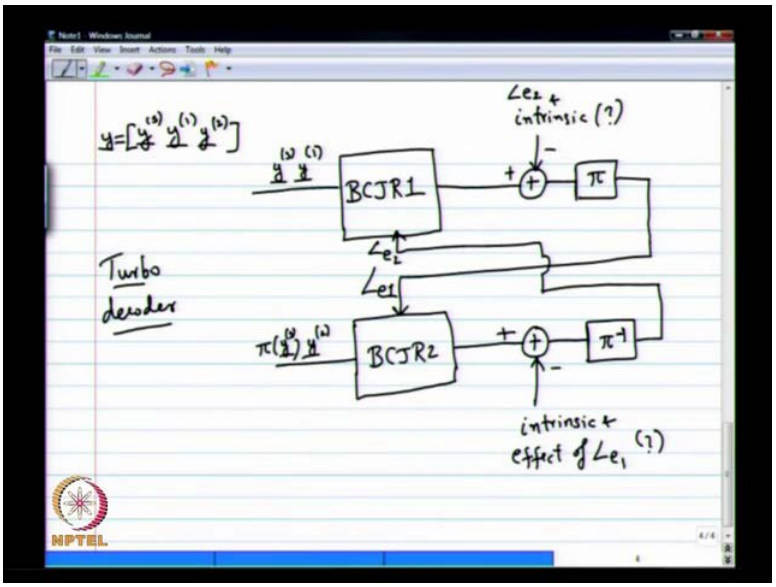


So, when you transmit maybe you transmit like this and then you can do rear arrangement what is the rear arrangement. Take all the systematic things together and put

them together because I am going to think of them as one block you can do a rear arrangement and think of it as y 0 y one all the way to y k minus 1. Then, you have y 1 0 y 1 1 all the way up to y 1 k minus 1 and y 2 0 all the way up to y 2 k minus 1.

Of course, you also have the termination once which you have to use suitably I am not showing how you split that you can split it depending how you want. So, this this guy I am going to call as y systematic, this one I am going to call as y 1 and this one I am going to call as y 2. So, that is the idea, so this is the received y 1 is the received vector corresponding to the first parity. This is y 2 is for the second parity and y s is what is corresponding to the systematic or the message part.

(Refer Slide Time: 24:33)



So, that is what comes into my decoder, so what I do here is I have to do some kind of a split so i am going to split into two parts. So, let me draw this a little bit differently I will keep it like this, then let me say so let me draw the decoder then will see what happens. So, I have BCJR 1, then I have BCJR 2, let me see if this d whose picture is going to come out very nicely or not BCJR 1 and BCJR 2 what is going to be input in BCJR 1 y s and y 1. So, that is what I want to be input to BCJR 1, but let me be A, so y s and y 1 is input to that, so let me write that down here y s and y 1 is input to that and it is going to put out output. So, what will it is output will be l l rs for each of the message bits.

So, that will give me, so likewise I can think of y s and y 2 going to the second BCJR decoder and that will give me once again l l rs for all the systematic bits for the same set

of bits. So, you have to somehow figure out, so maybe you want t be very specific there so i will say pi of y s. So, it is saying so because of the permutation so it is a permuted fashion list of that, but I guess it is yes if the account for it of course you have to account for it was a sequence is different.

So, that is that will give you that will give you are certain now it will give you again x l l rs for each of the message bits but in the permuted order. So, what you have to do is somehow figure out what BCJR 1 is producing which is extrinsic to BCJR 2, so you have to figure that out I will tell you how to figure that out. So, let us say somehow you figure out so you subtract what you do is you subtract intrinsic information, so you will get the extrinsic information what should I do to the extrinsic information it should go to BCJR 2.

So, this one should go to BCJR 2, so how can I draw that let me see it is going to get a little messy, but I guess there is no way to avoid that. So, let me do this, but when it goes here maybe you want to put up a permutation in there also because it is in a permuted order. So, maybe I want to say this is a permutation, so let us say I do a pi here, so how do you do this? We do not know it, we will see that and then what you do yes you have to do a pi inverse. So, you know that this is a something that has to inverted and then what you do you pass this back to this is no easy way to draw this.

So, I will do this and this will be l e 2 and that is extrinsic to BCJR 1, now what is BCJR ones job it is going to use y s y one and l e 2 and produce even better l l rs for all the message bits. Then, again you should subtract intrinsic and l e two you should subtract both of those and then send it once again back here. Then, this repeats, so you have this turbo processing, so to speak to get better and better l l rs for your message bits. This is a picture of the turbo decoder is it is not bad came out quite neat is it, so there are several questions, I mean when do you stop for instance. I mean there are several rules you can use some people are suggesting that l l rs, once they became sufficiently high they may not be all high no some l l rs might refuse to become high.

Then, what you do so the usual rule that used as after 5 times or 10 times stops that is that is the most implementable rule because that is what you can implement in a system after so many iterations after taking, so much clocks. So, any clocks you have to stop and say what it is, so that is something you can do some people have some more cleaver

ways of putting a CRC for their message and then figuring out if the CRC is satisfied for the decisions whenever the CRC is satisfied or stop CRC. By the way, cyclic redundancy check for figuring out if there are errors in your bit packets or not, so those are things that you can try so, but usually common rule that used at least in the standards they do not provide for CRC in the message maybe not.

So, then you have to use simply five or ten iterations and stop, so usual number is 5 or 10, so an iteration consists of two different BCJRs. So, you can think of the first half iteration and the next half iteration and stop at any point, you stop either there or here. So, that is it that is what triggered the revolution.

So, it is it is a bit non trivial why this work you know if you ask me the question why does it work why is that individual codes are not that great. When I say not that great, they are not capacity approaching they are not they cannot get to the capacity with somehow when you do this turbo idea. Then, the works well, I think likes they complement each other I do not know you can be happy with such answers, but if you want more precise answers, then what a marriage broker can give.

So, then you have to you have to look for more governing principles you know I mean what is the idea let me see what you think why is this suddenly all the better randomness is better is what you saying. So, first thing is it is very implementable, so that is the first big idea so the general rule that you can remember is anytime you have an implementable soft decoder. It is likely to be very good so that is kind of something which is generally true you know. I mean the only problem with achieving capacity is not being able to decode people knew that any random code you pick is good enough just construct your parity check matrix with putting random ones here and there is good enough.

So, it works it is not the code is good but then what is the problem I cannot decode it I cannot do soft decoding so that is roughly what is happening here so you come with a nice clever good soft decoder for your code and that code is sufficiently random. They cannot the code cannot be one code like repetition code and that is not going to be very good. So, you have to you have to have a sufficiently random code, so when I say sufficiently random what do I mean the code design method should be able to produce

lot of codes. So, what is how am I producing lot of codes, I have only two codes here right the trick is the inter leaver.

So, every type of inter leaver I have a different code, so when you are addressing a large space of codes like that you are highly likely to find good codes. So, finding good codes is not a big deal you might be able to find it quite easily, but then the problem is how you come up with the good soft decoder which is implementable and it is also good.

I mean it has to be a little bit good it cannot be some is like that you cannot take randomly some values it has to some something meaningful. Once you put this things together most things were go, so at this point even today people do not have a really good understanding very clean understanding of why is it that this works. Also, another thing I should point out is this works only for finite bit error rates finite frame error rates fine low but finite so you can go down to 10 power minus 5, 10 power minus 3 frame error rate you cannot you cannot give guarantees that it will keep on going down.

On the other hand capacity is something which you can achieve with arbitrarily low frame error rates. So, say really it does not fully solve the problem, but in practice 10 power minus 3 frame error rate is more than good enough. I mean no nobody is going to do more than as anything like that and today anywhere IP packets are busty and all that. So, you do not need to really guarantee very huge very low frame error rates except for things like storage etcetera for communications frame error rate of 10 power minus 3 is good enough and for those kind of frame error rates.

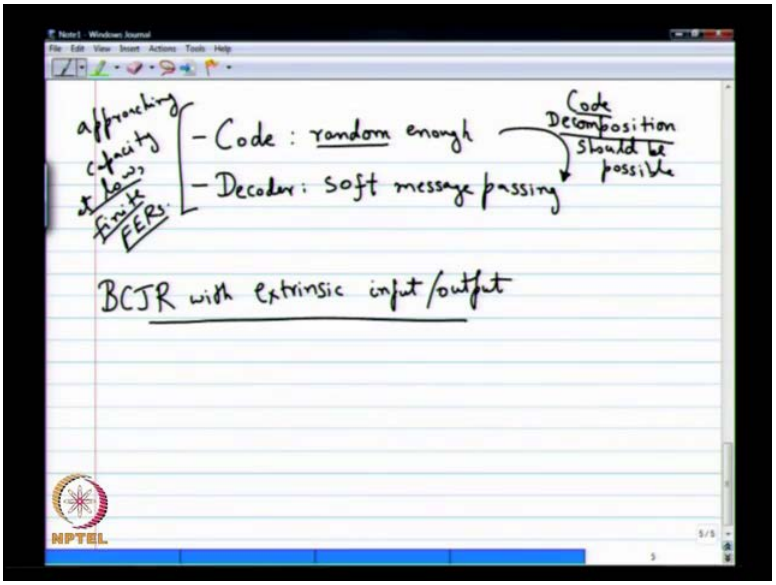
It is enough if you apply this general principles what the general principles your code design should have a random element which make sure that you are addressing a large space of codes. You not struck with just one code you have to able to address a large space if codes and next it in the decoding you should be doing soft decoding. You have to use the idea of message passing the message passing seems like a the really best idea for designing this good soft decoders as long as you use principles of message passing what is the most important principle in message passing extrinsic.

So, extrinsic is the most important principle you should not keep sending information that your other component decoders already knew they knew only increasing the instability in the whole process. So, you make sure you are not you are not making those mistakes and you use message passing ideas you going to have a good implementable

decoder. So, once you do that works fine another common idea is this concatenation the notation of looking at a long code word as a small part of a long code word being a separate code word by itself.

It might be as simple as a parity check code just one parity check or it might be more elaborate like a convolutional code, but as long as you do that you can have different component decoders. Then, you can connect them up use principles or message passing you going to be fine, so maybe I write down these principles it is interesting to write down stuff I suppose to do derivations.

(Refer Slide Time: 37:30)



The first idea is code needs to be random enough so the reason why you cannot they want to have a random element is you want to address a large space of codes of course this is not strictly necessary if you find the best code. Then, you do not need any random element in it you know that it works that is no problem, but if you do not if you not sure just make sure that you address a large enough code. So, any one code is good enough and then you can work with, so that is the idea in the code and then in the decoder you have to do soft message passing.

So, that is the main idea and what enables soft message passing n the construction of the code more than concatenation. I want to use the word decomposition the code decomposition should be possible this is an again a crucial idea in the design of the code.

What do I mean by decomposition a loose word again, but you can decompose a large code word into several small code words and these several small code words might even overlap. Whenever they are overlap you have to make sure that in your message passing you take care of the intrinsic part you remove the intrinsic part before you pass the extrinsic part. So, once you do these things you are fine so this is the general what are the ideas useful for these ideas are useful for approaching capacity at low finite SNR.

So, this is important this word is often not emphasized in when people talk about low turbo codes and l d p c codes because many industries. So, excited about it and in industry is enough if you have low finite SNR, so that is almost capacity approaching but in theory these are not capacity approaching. So, it is not a not at capacity approaching you cannot guarantee arbitrary low at, so this is this is good enough in practice anyways this is nothing to do with turbo codes by itself.

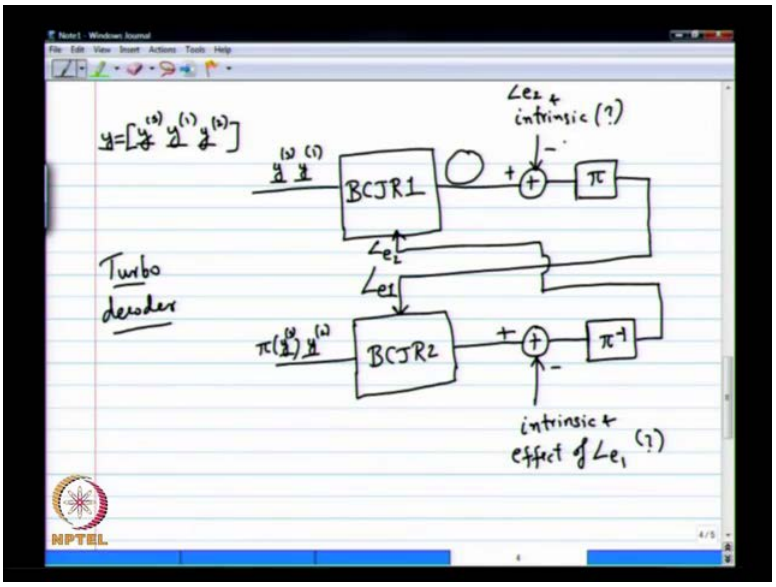
It is just a general principles that maybe we have observed maybe I have observed it and told you and maybe both of us observed together in these LDPC codes and turbo codes. In fact, these are not the only codes out there is something called a turbo product code which is again a manifestation of this principle then they something called a repeat accumulate code again a manifestation of this ideas. So, many other ideas out there of constructing codes which have a random component or maybe not even random component codes that have good code decompositions some code decompositions are there for which there are nice component decoders.

Then, you do message passing carefully you get good performance it goes down and I say good performance what mean low finite are possible and at close to capacity that is the that is the idea. So, the missing element in the turbo decoder is a BCJR for which which can use extrinsic information and how to how to get rid of the intrinsic information and the affect of the extrinsic information. That was input to the BCJR decoder and compute in the extrinsic out, so how to how to use extrinsic input and how to compute extrinsic output, so that is the idea that is the only thing that is missing here so, let us do that BCJR with extrinsic input output.

So, remember what the extrinsic input it is l l r it is l e it is interesting some people, I do not know I mean find this fascinating, I mean any place you always have this and people

come for the answers immediate answers. It is good, I mean that is an important skill believe me so, you have r, so for the message bits.

(Refer Slide Time: 41:13)



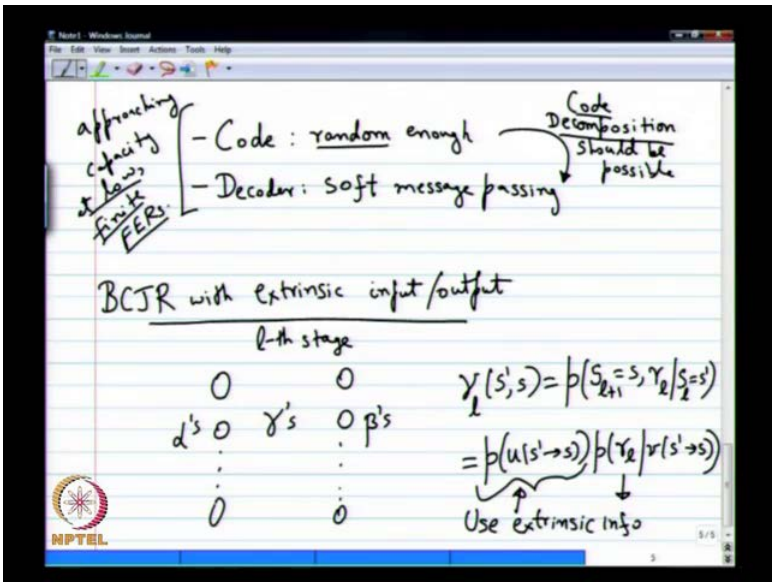
So, that is what extrinsic information what is extrinsic output again once again l l rs for the message bits. Now, what is intrinsic what do I am saying is intrinsic to both the decoders? So, it is I am defined it properly, but the way you have to think about it is the final total l l r that is computed here will have some component from l e 2 and it will also have some component from y s.

Now, both these things are known to BCJR 2, BCJR 2 knows y s it also knows l e 2, so I have to subtract that component of this total l l r which came from l e 2. Then, I also have to subtract that component of the total l l r which came from y s, once I subtract those too, I will get definitely my extrinsic output. So, that is the idea, so what we have to try to do is the total l l r what we computed in a particular stage there is a formula we used for computing the total l l r can we splitted into the l l r from the channel plus the extrinsic l l r. The extrinsic l l r input plus something else if we can spitted it like that, then what you do you can simply subtract these two things and you get the extrinsic. So, that is the idea so what we have to see is first how BCJR uses the extrinsic input that is something you have to see.

Next thing, you have to see is how do you get rid of the affect of intrinsic information and the extrinsic input information. So, for that the trick is to decompose the total l l rs as

systematic component intrinsic component plus the extrinsic input component plus the additional extrinsic output component, so that is what we do.

(Refer Slide Time: 43:27)



So, how we're going to use the extrinsic input that is the first question remember if you look at the l th stage what are the various probabilities that we are going to calculate in the l th stage there is alpha is on this side there are betas. On this side and these things have nothing to do with extrinsic or anything everything is computed first using the branch metric. So, the branch metric is the important thing which is which you compute after that alphas and betas are purely functions of the branch metric, so if at all extrinsic input is going to be used.

It should be used in the computation of branch metric that is all you cannot use anything else once you use once the branch matrix are fixed alphas and betas are simply functions of the branch metric you cannot do anything else in the alphas and betas. So, only thing you have to do is in your branch metric you have to input in you have to incorporate any possible extrinsic input, so how do you do that. So, let us see what how do you how do you define the gammas gamma l s prime s is that the notation that I used gamma l s prime s what is the formula for this it is p p what. So, we had we had a formula for this it is it is s s l plus 1 equals s comma r l r l no r l given s l equals.

So, this is our formula for gamma of s prime comma s and then I did some simplification on this and then we came up to one final version we said p of u s prime s. You remember

this simplification and then p of r l given v no v s prime s did we not come to this simplification so this is the branch metric and this part. There is no role for any extrinsic information there p of r l given the particular to outputs is computed using computed using what you receive in the channel it is just a function it is it is a normal p d f.

You substituted into that whatever you get e power 1 by root 1 by 2 pi sigma square e power minus r l minus v one squared plus r l 2 minus square. There is nothing you can do there with the extrinsic information or anything else the extrinsic information is telling you right the something about the priors. So, we are saying this p of u s prime s is going to be half we do not know if that bit is 0 or one we do not know anything we are going to say it is half, but do somebody some extrinsic guy is telling. You guess what my extrinsic l l r is 10 plus 10, then what does it mean this is much more probable to be 1, so I cannot say simply say p of u s prime s is half I have to use the extrinsic information in this part so here is where you use extrinsic information.

This seems simple enough, so how do you compute the extrinsic information is the question it is not to not to difficult right so l l r is given to you, how do you compute the probability. You use this you substitute into the formula and eventually work with it we do enough algebra. We will get the probability right l l r is basically log of probability that the bit is 0 divided by probability that the bit is 1 if somebody gives you l l r. You can definitely find the probability, so after all the two probabilities add to 1 and then you have given the other formula you use both of them together you are going to get something. So, you do that and compute this, so in BPSK, AWGN it is possible to do some simplification here and remember when we actually do it we will actually implement the BCJR algorithm.

We are not going to be using this gamma we are going to be doing log of this also, you can do some simplification and the computation and that is something we are going to try and do. Now, that is what two minutes left, so I think maybe this is a good point to stop, so we will pick up from here in the next lecture. We will see how to simply this computation a little bit and use the extrinsic information in the gamma in the branch metric. Then, when you compute the final expression you will see that the that the final l l r output l l r will nicely split into extrinsic plus the intrinsic plus the extrinsic out. So, then you can compute what is happening, so that is...stop here.